LUCIO VASCONCELOS DOS SANTOS

MINICURSO DE ALGORITMOS E PROGRAMAÇÃO: O DESIGN INSTRUCIONAL DE UM MINICURSO ONLINE PARA O ENSINO SUPERIOR

Produto Educacional referente à Dissertação de Mestrado intitulada: o uso da metodologia ADDIE no Design Instrucional de um minicurso online de algoritmos e programação para o Ensino Superior, apresentado ao Programa de Mestrado Profissional em Ensino de Ciências, Matemática e Tecnologias como parte dos requisitos para obtenção do título de Mestre Ensino de Ciências, Matemática e Tecnologias.

Orientadora: Prof.ª Dra. Isabela Gasparini

JOINVILLE, SC
2017

Instituição de Ensino: UNIVERSIDADE DO ESTADO DE SANTA CATARINA
Programa: ENSINO DE CIÊNCIAS, MATEMÁTICA E TECNOLOGIAS
Nível: MESTRADO PROFISSIONAL
Área de Concentração: Ensino de Tecnologias
Linha de Pesquisa: Tecnologias Educacionais

Título: MINICURSO DE ALGORITMOS E PROGRAMAÇÃO: o Design Instrucional de um minicurso online para o Ensino Superior.
Autor: Lucio Vasconcelos dos Santos
Orientador: Prof.ª Dra. Isabela Gasparini
Data: 2017

Produto Educacional: Minicurso Online.
Nível de Ensino: Superior.
Área de Conhecimento: Computação, Licenciaturas, Engenharias.
Tema: conceitos iniciais de AP (Algoritmos e Programação).

Descrição do Produto Educacional:
Este minicurso é o resultado de ampla pesquisa e desenvolvimento a partir de uma situação de aprendizagem, fundamentado pelo Design Instrucional, tornando-se uma solução inicial pronta para ser utilizada, mas que pode ser referência para futuras melhorias, adaptações ou ampliações. Tem como objetivo geral proporcionar que estudantes do Ensino Superior tenham acesso a um minicurso online, extracurricular e gratuito para aprendizagem de conceitos iniciais de AP.

Biblioteca Universitária UDESC: http://www.udesc.br/bibliotecauniversitaria
Publicação Associada: O uso da Metodologia ADDIE no Design Instrucional de um Minicurso Online de Algoritmos e Programação para o Ensino Superior.

URL: http://www.cct.udesc.br

Arquivo	*Descrição	Formato
	Texto completo	Adobe PDF

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 Apresentação

Aos colegas, professoras e professores:

O ensino de programação é foco de muitos currículos de cursos de Ensino Superior, não somente daqueles diretamente ligados à área de Computação. A oferta de disciplinas com ênfase em algoritmos e conceitos iniciais de programação é uma proposta para minimizar as dificuldades de aprendizagem enfrentadas por estudantes ingressantes no Ensino Superior, especialmente em outras disciplinas que envolvam a resolução de problemas. O ensino de fundamentos comumente utilizados em programação, como algoritmos e estratégias para resolução de problemas, podem ampliar as possibilidades dos estudantes desenvolverem seu pensamento abstrato e seu raciocínio lógico-matemático. Porém, pesquisas apontam que professores e estudantes têm encontrado desafios na aprendizagem de tais conteúdos, e que em cursos de computação podem estar relacionados a índices de reprovação e de evasão. Desta forma, nos propusemos a elaborar um minicurso online, como apoio às disciplinas presenciais de Algoritmos e Programação (AP).

Nosso objetivo é apresentar a você o minicurso completo, tal como foi concebido e aplicado, a partir de nossa pesquisa de mestrado, a estudantes de cursos nas áreas da computação, licenciaturas e engenharias, que tinham disciplinas de AP em seu currículo. Tal minicurso, denominado MAP (Minicurso de Algoritmos e Programação), foi desenvolvido no AdaptWeb® (Ambiente de Ensino-Aprendizagem Adaptativo na Web) e aplicado em turmas de cursos de graduação do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina (CCT-UDESC). Para seu desenvolvimento, utilizaram-se técnicas de Design Instrucional (DI) com base no modelo ADDIE (Análise, Design, Desenvolvimento, Implementação e Avaliação). Foram elaborados mais de 200 objetos de aprendizagem, dentre conceitos, exemplos e exercícios, permitindo assim ao estudante rever os conteúdos que eram apresentados em sala de aula, durante o curso de uma disciplina presencial, de modo que ele pudesse acessar este conteúdo online, na hora e local mais conveniente, de forma sequencial ou livre, no seu próprio ritmo de aprendizagem. Para você, professor, existem ferramentas que permitem acompanhar a participação de cada estudante por meio de relatórios, e utilizar avaliações para verificar seu desempenho.

Desta forma, o professor poderá conhecer melhor o minicurso e verificar a possibilidade aplicação para seus próprios objetivos, podendo inclusive melhorá-lo, adaptá-lo ou ampliá-lo, sendo para nós uma satisfação poder colaborar para a expansão da pesquisa a partir do nosso trabalho.

Apresentamos, inicialmente, o ambiente utilizado. Em seguida, descrevemos a concepção e elaboração dos conteúdos do MAP, e, finalmente, a implementação do minicurso.

2 Introdução

Em sala de aula, como professor de Matemática no Ensino Fundamental e Médio, venho percebendo como é importante o uso da tecnologia na minha prática docente e também no processo de aprendizagem dos estudantes. Essa questão me lembra de uma parte das minhas experiências ligadas ao uso da tecnologia: eu tive a oportunidade de aprender a programar computadores muito cedo, em uma época que essas máquinas estavam ainda começando a se popularizar no Brasil, poucas pessoas tinham acesso ou conhecimento, o material de ensino era escasso e ainda não existia Internet.

De lá para cá, de uma forma ou outra, estas questões me instigaram e me levaram a continuar estudando e a querer pesquisar na área, pois comecei a relacionar algumas estratégias de programação aos conteúdos da escola, como professor de Matemática. Mesmo não tendo me tornado um especialista da computação – uma vez que cursei a Licenciatura em Matemática –, estou certo de que muitas das habilidades que acabei desenvolvendo na área de computação, mesmo sem perceber, me ajudaram (e ajudam) a resolver problemas nas mais diversas situações. Atualmente, em minhas aulas de matemática, percebo em muitos estudantes dificuldades no processo de desenvolvimento de certas habilidades, como o raciocínio lógico e o pensamento abstrato, ao se deparar com a resolução de problemas. Partindo da minha experiência pessoal e profissional e dos estudos que venho fazendo sobre o tema, penso que a tecnologia pode ajudá-los nesse processo. Especificamente, acredito que o ensino de fundamentos de algoritmos e programação podem ajudar os estudantes a desenvolver tais habilidades, especialmente àqueles que não tiveram esta oportunidade na Educação Básica, e que precisam delas para seguir seus estudos no Ensino Superior.

3 Ambiente AdaptWeb®

O ensino de programação nos cursos de graduação é tema de debate na área acadêmica, tanto pela sua importância na formação geral dos estudantes quanto pelos desafios ligados ao seu ensino, notadamente as reprovações em disciplinas de AP e a evasão dos cursos. É preciso, pois, atacar esse problema. Uma das maneiras de se fazer isso aparece na forma de propostas de desenvolvimento, utilizando as Tecnologias da Informação e da Comunicação (TIC), de ambientes online de aprendizagem como apoio ao ensino presencial.

O AdaptWeb® (Ambiente de Ensino-Aprendizagem Adaptativo na Web) é um ambiente de aprendizagem online utilizado no CCT-UDESC. Trata-se de um sistema hipermídia adaptativo que permite a adaptação da navegação, da apresentação e do conteúdo conforme o perfil do estudante (GASPARINI et al., 2009). O ambiente armazena alguns dados do estudante, tais como curso, conhecimento, preferências e histórico navegacional e adequa o conteúdo, a apresentação e a navegação ao perfil de cada estudante.

Uma importante característica em sistemas de EaD (Educação a Distância) é procurar a melhor maneira para apresentar a informação aos estudantes, e a utilização de sistemas adaptativos, versáteis e poderosos para organização e acesso a informação, pode ser uma alternativa para aumentar a qualidade dos sistemas de EaD via web.

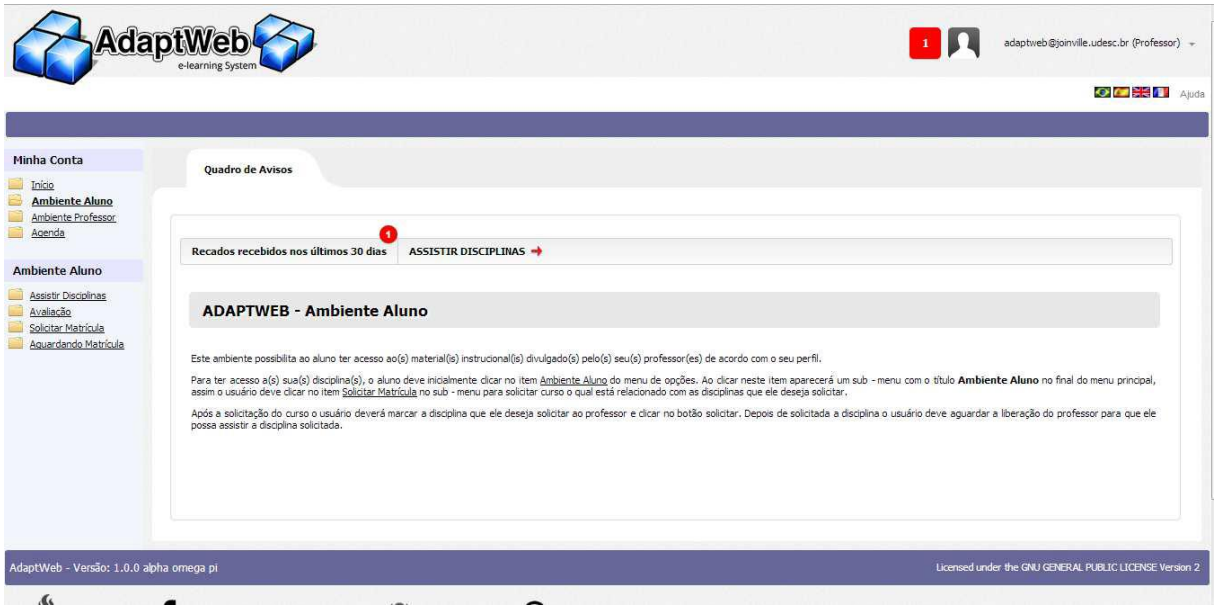
O AdaptWeb® é apresentado em dois ambientes distintos: o ambiente de autoria, para os professores, e o ambiente de navegação, para os estudantes. O cadastro das contas de professor e de estudante é feito da mesma forma, porém a conta de professor necessita de uma aprovação especial pelo administrador do sistema. O acesso ao ambiente de autoria permite que o professor crie e disponibilize um curso ou disciplina, além de poder controlar as solicitações de matrículas aos cursos por ele criados. A conta de estudante permite a escolha e solicitação de acesso aos cursos ou disciplinas desejados.

Figura 1 – Ambiente do professor no AdaptWeb®.

Fonte: AdaptWeb® (2016).

Pelo ambiente de autoria, mostrado na Figura 1, o professor pode organizar os materiais instrucionais em tópicos hierárquicos. Para cada um dos tópicos, o material pode ser incluído na forma de conceito (um para cada tópico), exemplos, exercícios e materiais complementares.

Figura 2 – Ambiente do estudante no AdaptWeb®.



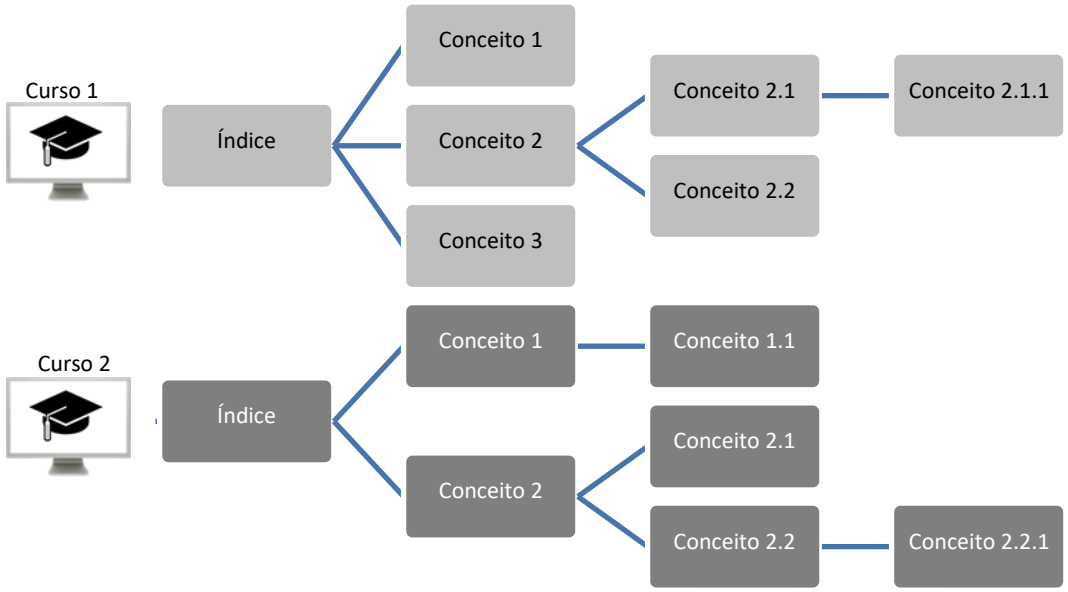
Fonte: AdaptWeb® (2016).

Uma vez dentro do ambiente de navegação, descrito como “Ambiente Aluno” na Figura 2, o estudante deverá selecionar o *link* “assistir disciplinas”, que o levará a uma página onde poderá optar pelo modo tutorial ou livre. No modo tutorial, os tópicos do curso são apresentados conforme a sequência pré-estabelecida pelo professor na fase de autoria, garantindo que um tópico seja visto somente se seu pré-requisito tiver sido estudado; em

modo livre, qualquer tópico pode ser visitado, independente da classificação estabelecida pelo professor.

Processo de pré-autoria. Sugere-se antes do processo de autoria, um processo de pré-autoria onde o material a ser disponibilizado é organizado. É importante que se defina a disciplina e seus objetivos, estabelecendo os conteúdos que a disciplina abrange, e organizando-os em tópicos, que no ambiente são chamados de conceitos.

Figura 3 – Conceitos por curso no AdaptWeb®.



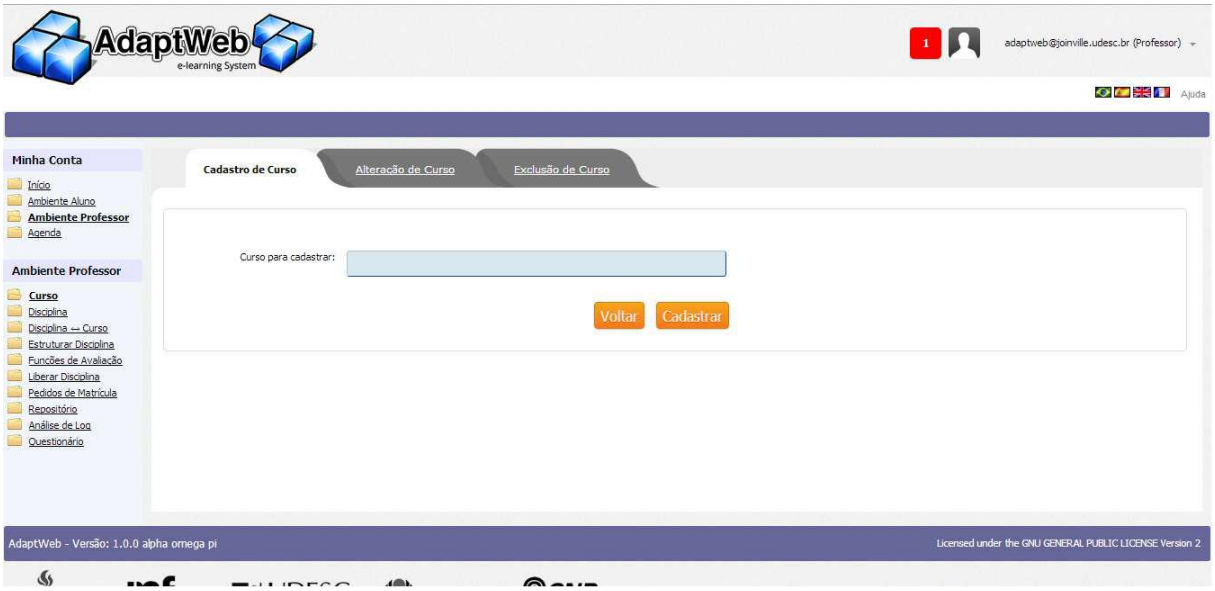
Fonte: elaborada pelo autor, 2016.

Como mostra a representação apresentada na Figura 3, o professor pode organizar os materiais instrucionais uma única vez para uma disciplina ministrada em dois ou mais cursos, definindo quais materiais estarão disponíveis para cada um deles. Por exemplo, o professor pode criar uma única vez uma disciplina de cálculo, e definir que conceitos, exemplos, exercícios e materiais complementares serão vistos para diferentes cursos (Matemática, Computação, Engenharia).

Processo de autoria. Inicialmente, deve-se acessar o ambiente como usuário de professor. A partir deste momento, pode-se cadastrar os cursos e as disciplinas, bem como estruturar os conteúdos que se deseja disponibilizar no ambiente.

Cadastro do curso. Nesta etapa o professor deve informar o nome do curso que deseja cadastrar.

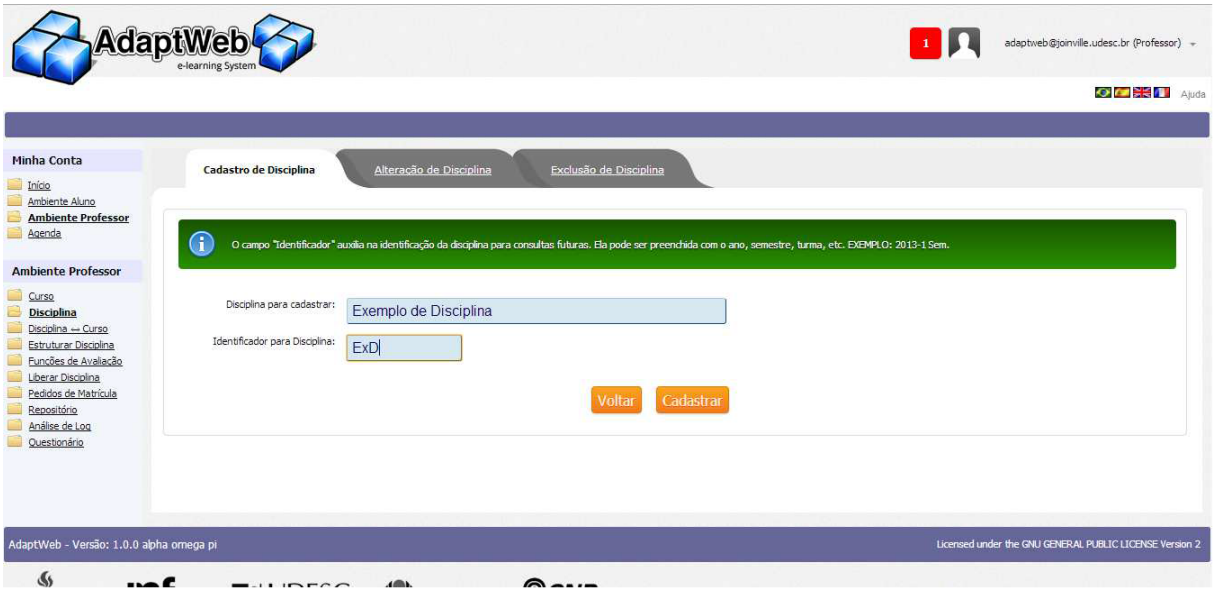
Figura 4 – Tela de cadastro de curso no AdaptWeb®



Fonte: AdaptWeb® (2016).

Cadastro de disciplina. Após ter cadastrado os cursos relacionados ao conteúdo a ser inserido no ambiente, deve-se cadastrar a disciplina, através do item Disciplina no menu à esquerda da tela, como mostra a Figura 5.

Figura 5 – Tela de cadastro de disciplina no AdaptWeb®

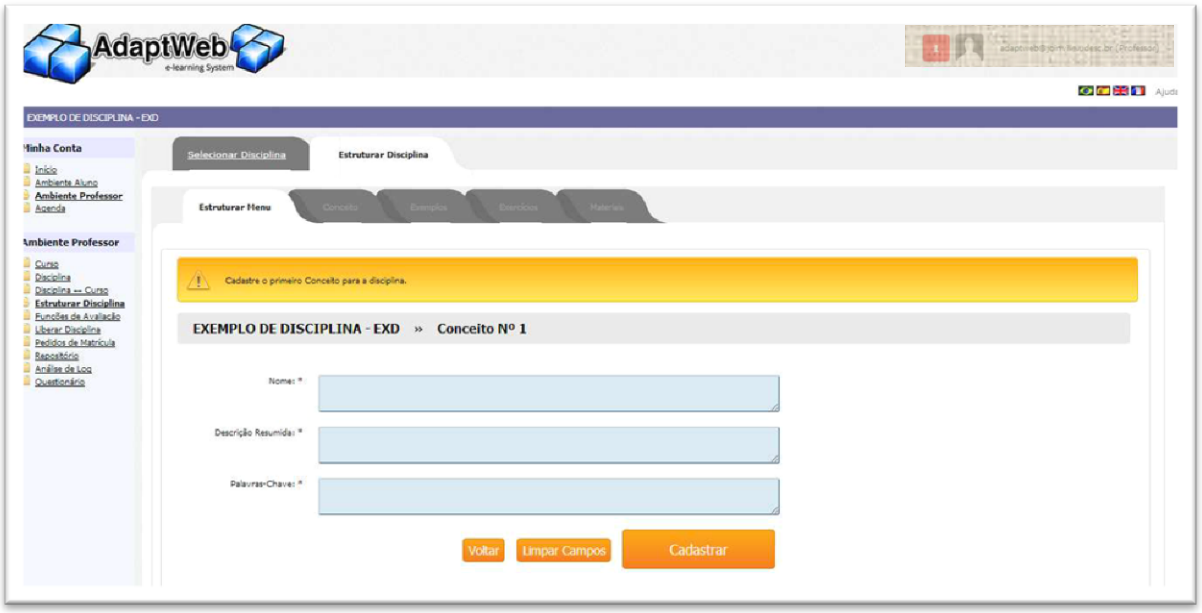


Fonte: elaborada pelo autor, 2016.

Pode-se cadastrar uma mesma disciplina para vários cursos diferentes, e cada disciplina poderá ter seus conceitos, exemplos, exercícios e materiais complementares específicos.

Estruturar conteúdo. A estrutura dos tópicos – ou conceitos – de um curso dentro do AdaptWeb® é configurada por uma tela de cadastro.

Figura 6 – Tela de cadastro de conceitos no AdaptWeb®



Fonte: AdaptWeb® (2016).

Com isto deve-se inserir o conceito inicial da disciplina, assim como a descrição resumida do conceito e as palavras chaves e clicar em Cadastrar para salvar os dados, como pode ser visualizado na Figura 6.

Os exemplos, exercícios e materiais complementares devem estar associados aos respectivos conceitos. Por isso, durante a etapa de estruturação do conteúdo devem-se inserir estes itens corretamente, associando-se os arquivos necessários a cada conteúdo.

O professor também pode criar avaliações e analisar a participação de seus estudantes. O estudante, por sua vez, acessa a disciplina por meio do Ambiente da Disciplina, em que tem acesso ao Ambiente de Aula (i.e., onde se pode ver todo o material instrucional destinado àquela disciplina), ao Mural de Recados, ao Fórum de Discussão e à ferramenta de Análises de Aprendizagem.

Ao final do processo de autoria, o curso estará disponível para ser liberado pelo professor, e assim poder ser acessado pelo estudante.

4 Metodologia

Este capítulo apresenta o desenvolvimento do produto educacional, e contou com a parceria, além do próprio autor e da professora orientadora da dissertação, Prof.ª Dra. Isabela Gasparini, de mais três colaboradores: as então mestrandas, Ana Carolina Tomé Klock e Barbara Moissa, do Programa de Pós-Graduação em Computação Aplicada da UDESC, e o graduando Vitor Mulazani dos Santos, do Curso de Bacharelado em Ciência da Computação da UDESC. A esta parceria denominamos "equipe do MAP", e creditamos a esta equipe todos os recursos que não foram gerados exclusivamente por um único autor.

De acordo com Demo (1996), pesquisa "é uma atitude, um questionamento sistemático, crítico e criativo, mais a intervenção competente na realidade, ou o diálogo crítico permanente com a realidade em sentido teórico e prático". Sampieri, Collado e Lucio (2013), definem pesquisa científica como sendo "um processo composto por múltiplas etapas relacionadas entre si, que acontece ou não de maneira sequencial ou contínua. [...] é um processo composto por diferentes etapas interligadas" (SAMPIERI; COLLADO; LUCIO, 2013).

Nesse sentido, serão apresentadas as etapas e procedimentos metodológicos desenvolvidos ao longo do período de mestrado para a concepção deste produto educacional.

Pode-se classificar esta pesquisa como de natureza aplicada, pois "objetiva gerar conhecimentos para aplicação prática, dirigida à solução de problemas específicos"(KAUARK; MANHÃES; MEDEIROS, 2010, p. 26).

Quanto à abordagem do problema, pode ser classificada como quali-quantitativa ou mista:

> Os métodos mistos representam um conjunto de processos sistemáticos e críticos de pesquisa e implicam a coleta e a análise de dados quantitativos e qualitativos, assim como sua integração e discussão conjunta, para realizar inferências como produto de toda a informação coletada e conseguir um maior entendimento do fenômeno em estudo . (HERNANDEZ, SAMPIERI E MENDOZA, 2008 APUD SAMPIERI, CALADO E LUCIO, 2008)

As etapas da pesquisa, no sentido proposto por Sampieri, Collado e Lucio (2013) foram planejadas e executadas para compreenderem as fases de elaboração de conteúdo do MAP como produto educacional derivado da pesquisa de mestrado profissional. Tais etapas são listadas a seguir, sendo que as etapas 2 e 3, referentes ao desenvolvimento do MAP, serão explicadas em detalhes na seção seguinte. Os processos referentes às etapas 4 e 5 serão descritos e discutidos na publicação associada a este documento, que é a dissertação de mestrado.

- **Etapa 1.** Como primeira etapa, fez-se uma revisão de literatura sobre temas ligados ao desenvolvimento do produto educacional. Estudaram-se, assim, questões relacionadas ao ensino de programação no Ensino Superior, o Design Instrucional e o estudo de Algoritmos e Programação. Esses estudos serviram para embasar as concepções da proposta de criação do MAP. Além disso, foi

estudado o funcionamento do AdaptWeb®, ambiente que ambientou o minicurso.

- **Etapa 2.** Os conteúdos do MAP foram elaborados na segunda etapa da pesquisa. Para tal, além dos estudos sobre o tema, foi realizado um levantamento dos conteúdos ministrados nas disciplinas do CCT-UDESC, observando ementas, planos das disciplinas e por meio de Entrevistas Estruturadas com os respectivos professores.
- **Etapa 3.** Na terceira etapa, o MAP foi concebido e implantado no ambiente AdaptWeb®.
- **Etapa 4.** A quarta etapa compreende a coleta e análise dos dados referentes à observação da participação e do desempenho dos estudantes no minicurso. Os dados foram avaliados tanto de forma quantitativa como qualitativa com o objetivo de evidenciar a satisfação dos estudantes participantes e fornecer dados para verificação do objetivo da pesquisa realizada.
- **Etapa 5.** Apresentação dos resultados da pesquisa.

Neste documento referente ao produto serão mais exploradas as Etapas de Elaboração do Conteúdo (Etapa 2) e de Concepção e Implementação (Etapa 3).

5 Design do Produto Educacional Utilizando ADDIE

Nesta seção são descritas as ações de pesquisa relacionadas às etapas 2 e 3 dos procedimentos metodológicos, onde são apresentados os processos de concepção, elaboração e implementação do Minicurso Algoritmos de Programação (MAP), que é o produto educacional online resultante da pesquisa aqui apresentada.

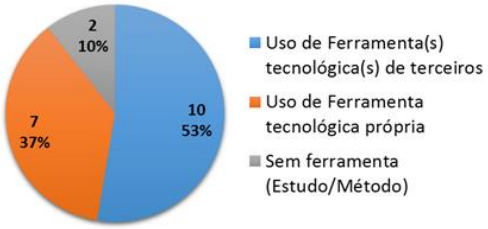
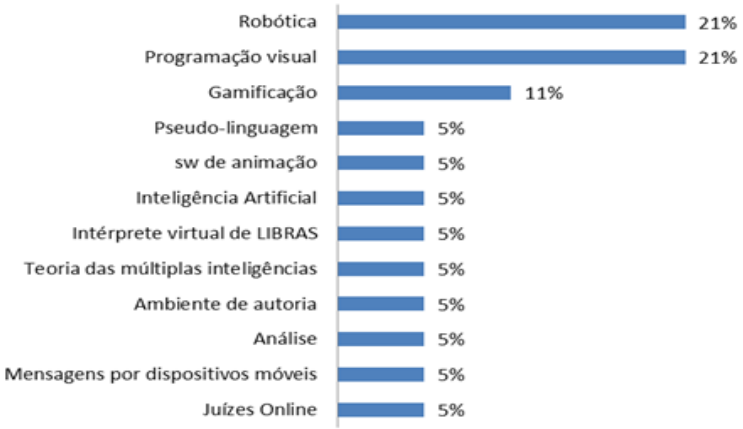
O MAP foi concebido e implementado no ambiente AdaptWeb® da UDESC, segundo o modelo ADDIE (Análise, Design, Desenvolvimento, Implementação e Avaliação), para apoiar os estudantes de disciplinas presenciais de cursos de graduação do CCT-UDESC.

As etapas de criação do curso, no modelo ADDIE, são descritas a seguir.

5.1 ADDIE – Análise

Para criação do MAP, foi levantado o referencial teórico sobre o ensino de programação no Ensino Superior. Tal busca e análise levou ao processo de criação do produto educacional. Porém, investigaram-se também outros artigos científicos, indexados no APÊNDICE A, para responder às questões de pesquisa na Tabela 1.

Tabela 1 – Análise contextual – questões de pesquisa

Q1 – Quais as características que mais se destacaram?
• Ferramentas tecnológicas – criação de nova ferramenta, uso de ferramenta de terceiros ou não uso de ferramentas (apenas com referência de técnicas ou teorias). • Estratégias de ensino ou aprendizagem utilizadas para auxiliar na compreensão dos conteúdos.
Q2 – Que tipo de ferramentas tecnológicas foi utilizado?

Q3 – Que estratégias didáticas fundamentaram os trabalhos?


Fonte: elaborada pelo autor, 2015.

Assim, foi realizada uma análise contextual guiada pela seguinte questão: quais estratégias didáticas e tecnológicas para o ensino de AP vêm sendo utilizadas voltadas a um público iniciante na graduação?

Três questões derivaram desta primeira, tornando-se fundamentais nesta etapa do método ADDIE. As perguntas e os resultados encontrados são apresentados na Tabela 1.

Como relação à ferramenta de apoio ao ensino de programação, como visto no gráfico de setores apresentado na Tabela 1, encontram-se tanto propostas que sugerem novos aplicativos (37% dos trabalhos avaliados), a exemplo do Mojo, Ceebot e o iVProg, quanto outras que se utilizam de softwares ou recursos tecnológicos de terceiros, isto é, já existentes (a maioria, com 53% do total destes trabalhos), tais como Scratch, Lego Mindstorms, Juizes Online, Moodle, Portugol Studio. Apenas 10% (dois documentos) não utilizaram qualquer ferramenta, como o que sugere a aplicação de técnicas da teoria das múltiplas inteligências para o aprendizado de AP. Verificou-se uma preocupação comum aos trabalhos estudados, que era a de tornar o ato de aprender em uma atividade atrativa e agradável para os estudantes iniciantes.

A partir desses resultados iniciais, a busca pelo estado da arte ampliou-se com a inserção dos artigos publicados nos anais de 2015 do SBIE[1], por ser esse um importante simpósio sobre informática na educação que envolve o contexto brasileiro. Buscaram-se, nos trabalhos publicados nos anais do evento, enfoques teóricos e práticos relacionados ao ensino de programação a iniciantes. A lista artigos estudados está no APÊNDICE B.

Nesta nova análise, observou-se que as principais informações destas publicações foram: (a) o problema de dificuldade de aprendizagem em disciplinas iniciais da computação ou mesmo disciplinas curriculares, causando alto índice de reprovação e evasão dos cursos; (b) alguns artigos do evento trazem como referência publicações em geral recentes, mas estas referências têm referências primárias bem mais antigas, e a maioria cita a mesma dificuldade no aprendizado, remetendo a publicações já da década de 1980; (c) alguns mapeamentos sistemáticos desta edição do evento fazem estudo de publicações anteriores e um, inclusive, destaca que a quantidade de estudos recentes no simpósio é indício de que o problema ainda preocupa; (d) também é frequente o tema de ensino de programação a iniciantes, propondo soluções para auxiliar no aprendizado.

O modelo proposto por Filatro (2008) quanto ao método ADDIE prevê a apresentação de um relatório de análise, ou seja, de um documento resultante da etapa de análise na forma de "relatório estruturado que deve ser apresentado ao cliente e a outros envolvidos diretamente no problema" (FILATRO, 2008, p. 38). Devido às características de um trabalho acadêmico dentro de uma universidade, e não tendo a definição exata de um "cliente" em nosso processo, considerou-se como Relatório de Análise a discussão resultante da pesquisa realizada, conforme Tabela 2.

[1] O Simpósio Brasileiro de Informática na Educação (SBIE) de 2015 ocorreu durante o IV Congresso Brasileiro de Informática na Educação e X Conferência Latino-Americana de Objetos e Tecnologias de Aprendizagem (CBIE & LACLO). O CBIE é um evento anual de caráter internacional promovido pela Sociedade Brasileira de Computação (SBC), disponível em: http://ic.ufal.br/evento/cbie_laclo2015/.

Tabela 2 – Relatório de Análise: definições iniciais para o produto educacional

Tópico	Definição
Estratégias de ensino	(a) Elaboração conteúdos para o ensino de AP por meio de exemplos e exercícios em linguagem C, mas oferecendo ao estudante a oportunidade de estudar pela pseudolinguagem. (b) Traços de gamificação inseridos com o intuito de potencializar os resultados desejados com a aplicação dos métodos de DI. (c) Possibilidade de aplicação dos conceitos de adaptabilidade no produto educacional.
Público-alvo	(d) Estudantes matriculados em todos os cursos de graduação do CCT-UDESC que têm disciplinas voltadas ao ensino dos conceitos iniciais de programação.
Modalidade (presencial / à distância)	(e) O ensino a distância, na modalidade de minicurso online, é utilizado como apoio ao ensino presencial das disciplinas de AP.
Teoria de aprendizagem	(f) O minicurso foi baseado na perspectiva socioconstrutivista, desde a elaboração dos conteúdos à avaliação da aprendizagem, passando pelas etapas de ministrar os conteúdos e compartilhar conhecimentos.

Fonte: elaborada pelo autor, 2015.

Da Tabela 2, seguem as explicações.

(a) Por ser referência para o ensino e aprendizagem de AP, a linguagem C ainda é utilizada em muitos trabalhos pela sua relativa simplicidade de compreensão e aplicação, sendo um caminho de fácil acesso ao aprendizado de outras linguagens mais avançadas. Porém, é uma linguagem estruturada, ou seja, que exige uma estrutura rígida em sua escrita, e que ainda tem seus comandos escritos em inglês, o que pode ser uma dificuldade extra para estudantes que não dominam este idioma. Por isso, neste minicurso, optou-se por oferecer a possibilidade de acesso a um interpretador de código, que "traduz" os comandos para o idioma português, além de simplificar a estrutura da linguagem C – é a chamada pseudolinguagem. O foco do estudante passa a ser, então, a funcionalidade do programa, e não a sua escrita. Assim, decidiu-se, como uma das estratégias de ensino em nosso projeto, elaborar conteúdos para o ensino de AP por meio de exemplos e exercícios em linguagem C, mas oferecendo ao estudante a oportunidade de estudar pela pseudolinguagem.

(b) Dentre as diversas estratégias de ensino relatas nas publicações analisadas, notaram-se importantes resultados positivos no uso de soluções que envolvem o incentivo ao engajamento dos estudantes e sua consequente satisfação com o aprendizado. Dentre elas, mais recentemente, a gamificação. Por ser um tema recente, mas com grande potencial de aplicação na implantação do minicurso, elementos de gamificação foram inseridos no AdaptWeb® pelo trabalho de Klock (2017), com o intuito de potencializar os resultados desejados com a aplicação dos métodos de DI.

(c) A adaptabilidade também é uma estratégia que mostra resultados eficazes, principalmente em relação à satisfação dos estudantes. Um minicurso aplicado de forma adaptativa oferece ao estudante a possibilidade de estudar conteúdos direcionados às suas necessidades, em quantidade, grau de dificuldade e momento mais adequado. Estruturou-se o minicurso, então, considerando os contextos de curso já se prevendo a possibilidade futura de adaptabilidade no produto educacional. Porém, neste trabalho optou-se por criar objetos de aprendizagem para as diferentes situações e serem utilizados por todos os participantes. A partir dos resultados obtidos, outros trabalhos podem continuar o desenvolvimento e realizar adaptações a cada perfil.

(d) A proposta da pesquisa do mestrado profissional iniciou com o objetivo de investigar os benefícios do ensino de programação aos estudantes o mais cedo possível na fase escolar, ou seja, na Educação Básica. Porém, com a evolução do estudo pesquisas, observou-se que há outro grupo de estudantes que carece de apoio para enfrentar dificuldades prévias de aprendizagem na área: os estudantes que concluíram o Ensino Médio ou aqueles ingressantes no Ensino Superior. Analisando-se as possibilidades de aplicação do produto educacional para apresentar uma alternativa de solução para um problema encontrado nos cursos de graduação da universidade onde foi realizada a pesquisa do mestrado, optou-se por voltar a pesquisa para estudantes da graduação. Mais especificamente, escolheram-se como público alvo os estudantes matriculados nos cursos de graduação do CCT-UDESC que têm disciplinas voltadas ao ensino de AP, já que este tema e as dificuldades de aprendizado não se restringem apenas aos estudantes dos cursos de computação.

(e) Com relação ao ambiente do curso, optou-se pelo modo online, pois nos estudos realizados notou-se grande ênfase às vantagens oferecidas pelo ensino híbrido, que integra as formas de ensino presencial e de ensino à distância. Assim, neste produto educacional, o ensino a distância, na modalidade de minicurso online, é utilizado como apoio ao ensino presencial das disciplinas de AP.

(f) A respeito da concepção do curso, optou-se por utilizar uma postura socioconstrutivista, como a apresentada em Filatro (2008): a aprendizagem do estudante se dá sobre conceitos e habilidades existentes; a construção do conhecimento é apoiada pelo ambiente social, no qual o estudante desenvolve uma compreensão compartilhada dos conteúdos; utilização de ambientes colaborativos, com recursos e desafios apropriados. Dessa forma, o minicurso foi baseado na perspectiva socioconstrutivista desde a elaboração dos conteúdos à avaliação da aprendizagem, passando pelas etapas de ministrar os conteúdos e compartilhar conhecimentos.

Definidos os fundamentos do produto educacional até este momento, ainda faltava selecionar os conteúdos do tema principal. Que conceitos poderiam ser considerados fundamentais ou suficientes aos iniciantes em estudos de AP? A resposta não era clara nos

textos pesquisados. Cada estudo defendia um conteúdo diferente, mas com alguns pontos em comum. Como proposta inicial, os autores basearam-se em sua própria experiência em sala de aula como professor da disciplina e nos parceiros do projeto, além de estudo de literatura especializada, como Forbellone e Eberspächer (2000), Mizrahi (2008) e Pereira (2010).

Tabela 3 – Proposta inicial dos conteúdos do MAP

1 Seja bem-vindo(a)!
2 Noções de algoritmo
Definição
Motivo para aprender
Meios de representar
Estrutura básica
Como criar
3 Portugol IDE
Pré-requisitos
Download
Utilização
4 Pseudolinguagem
Definição
Estrutura básica
5 Dados, constantes e variáveis
Dados
Constantes
Variáveis
Identificadores
6 Comandos básicos
Comando de atribuição
Comando de saída
Comando de entrada
7 Operadores
Operadores aritméticos
Operadores relacionais
Operadores lógicos
Precedência entre todos os operadores
8 Estruturas condicionais
Se...entao
Se...entao...senao
Escolhe...caso
9 Estruturas de repetição
Enquanto...faz
Faz...enquanto
Repete...ate
Para

Fonte: elaborada pelo autor, 2015.

Desta forma, conforme a Tabela 3, delimitou-se um conteúdo considerando alguns conceitos básicos, como algoritmos, dados, constantes, variáveis e operadores, além das estruturas mais comumente utilizadas em programação, como as condicionais e de

repetição. Para que se pudessem exercitar estes conteúdos, foram incluídas instruções para utilização de duas ferramentas, a da pseudolinguagem e a da linguagem C.

Na próxima seção é feito um estudo dos cursos que têm disciplinas referentes à AP no CCT-UDESC e seus conteúdos. Também são utilizadas as informações coletadas nas entrevistas com os professores para definição do design final do minicurso.

5.2 ADDIE – Design

O planejamento e a elaboração do conteúdo do minicurso, sua abrangência e demais elementos que farão parte do produto, como exemplos, exercícios e materiais complementares correspondem à etapa de design.

No planejamento das unidades de aprendizagem (UA), aqui denominados Objetos de Aprendizagem (OA), bem como para estabelecer uma relação adequada do conteúdo do minicurso online com as disciplinas presenciais, optou-se por (a) fazer levantamentos de todas as disciplinas iniciais dos cursos do CCT-UDESC, observando suas ementas e planos de ensino, e (b) após a compilação dos resultados, consultar professores do CCT-UDESC que ministram disciplinas relacionadas à AP. Esta investigação foi feita por meio de entrevistas estruturadas, que foram gravadas em áudio.

Uma entrevista envolve a interação entre, pelo menos, duas pessoas e podem ser de três tipos: (a) semiestruturadas, que envolvem perguntas abertas, pré-planejadas e são influenciadas por estímulos como “Por quê?” ou “Fale mais”; (b) não estruturadas, que apenas possuem um tema e não são planejadas; e (c) estruturadas, apoiadas por um questionário planejado e com rigoroso controle do tempo (OLSEN, 2015). Ainda segundo o autor, a entrevista estruturada tem o objetivo principal de obter respostas quantificáveis, que sejam facilmente registradas, e nelas, o entrevistador deve procurar não influenciar as respostas do entrevistado e utilizar um dispositivo para gravação.

As entrevistas com os professores de AP do CCT-UDESC foram planejadas e executadas porque uma das necessidades na definição dos conteúdos do minicurso era estar em sintonia com as disciplinas presenciais dos cursos que o minicurso apoiaria e, nesse caso, os professores regentes dessas disciplinas poderiam ser fontes de referência. A princípio, um simples questionamento a estes professores seria suficiente para saber se o conteúdo proposto estaria em consonância com o seu planejamento, e se o início e duração seriam adequados aos principais marcos de conteúdo, atividades e avaliações correspondentes. Mas, para isso, a entrevista não seria necessária, pois estas informações são disponibilizadas no *site* universidade ou constam nas páginas e repositórios online dos próprios professores. No entanto, nosso objetivo era entender a abordagem do professor, buscando obter outras informações mais relevantes, como as estratégias didáticas que eles aplicam em sala, a forma de abordagem dos conteúdos, as maiores dificuldades que os estudantes apresentam.

Assim as entrevistas tiveram como objetivo ajudar a: decidir que conteúdos atenderiam a prática da maioria dos professores das disciplinas presenciais; perceber se período de aplicação do minicurso estaria adequado à maioria dos professores considerando o andamento da disciplina presencial; reunir opiniões sobre o uso da pseudolinguagem (e o

Portugol IDE), da linguagem C (ou outras sugestões) ou de outras linguagens nas práticas do minicurso; reunir impressões sobre a utilização do minicurso online como apoio à disciplina presencial (e sua possível aplicação em outros contextos); conhecer a opinião sobre a utilização do AdaptWeb® como ambiente do minicurso; reunir experiências e sugestões sobre exemplos de conteúdos, exercícios, formas de avaliação e estratégias didáticas que poderiam ser utilizadas.

Além da coleta de respostas referentes aos objetivos, a entrevista foi um momento em que os professores foram apresentados à proposta inicial do curso (Tabela 3) e, sobre ela, convidados a sugerirem ajustes.

O roteiro das entrevistas (APÊNDICE D) foi elaborado a partir dos resultados de experiências e de pesquisas de satisfação obtidas em minicursos anteriormente aplicados. Associado ao roteiro, elaborou-se um Termo de Consentimento Livre e Esclarecido (TCLE), que consta no APÊNDICE C, necessário instrumento de autorização de aplicação da entrevista e uso sigiloso das informações.

A seleção dos professores entrevistados foi feita por meio de um mapeamento. Foi feito um levantamento de todos os cursos e disciplinas do CCT-UDESC que abordavam, em seus planos de ensino, conceitos iniciais de programação. Os resultados são apresentados na Tabela 4.

Tabela 4 – Cursos e disciplinas de AP no CCT-UDESC

		Disciplinas				
		AGT	ALP	PRE	ICC	APG
		algoritmos	algoritmos e linguagens de programação	programação para engenharia	Introdução à ciência da computação	Algoritmos e programação
Modalidade	Curso					
Computação	Ciência da Computação	1ª fase				
	TADS	1ª fase				
Licenciaturas	Licenciatura em Física		3ª fase			
	Licenciatura em Matemática		5ª fase			
Engenharias	Engenharia Civil		2ª fase			
	Engenharia Elétrica		1ª fase			
	Engenharia Mecânica			1ª fase		
	Engenharia Produção e Sistemas				1ª fase	2ª fase

Fonte: elaborada pelo autor, 2016.

Na Tabela 4 observam-se os oito cursos do CCT-UDESC que possuem disciplinas de AP em seus currículos, com indicação dos nomes e siglas e da fase do curso em que são ministradas. Na tabela, as disciplinas estão agrupadas em três modalidades: os cursos de computação, os cursos de licenciatura e os cursos de engenharia. Destas modalidades, analisando em termos de envolvimento com a programação, a de Computação contém os cursos mais representativos, sendo disciplinas fundamentais para diversas outras subsequentes; a de Engenharia tem aplicações que envolvem eventualmente a continuidade do seu estudo em algumas disciplinas, como o uso em máquinas, equipamentos ou

dispositivos; e para as Licenciaturas, geralmente, a programação é estudada em apenas um semestre do curso, sem continuidade explícita em outras disciplinas, somente possíveis aplicações em situações específicas de cada área.

O período de preparação para entrevistas foi de 18 de fevereiro de 2016 a 5 de junho de 2016. A média de duração das entrevistas com cada professor foi de 1 hora e 10 minutos. Esta etapa do planejamento das entrevistas com os professores também proporcionou um estudo das ementas de todas as disciplinas envolvidas. Isso foi relevante para se estabelecer uma visão geral dos conteúdos abordados nas disciplinas presenciais. O resultado é apresentado na Tabela 5.

Tabela 5 – Estudo de equivalência de conteúdos por disciplina

Disciplina / Tema	Algoritmos	Algoritmos e Linguagem de Programação (1)	Algoritmos e Linguagem de Programação (2)	Algoritmos e Programação	Introdução à Ciência da Computação	Programação para Engenharia
Arquitetura e hardware	Noções de arquitetura e				Conceitos básicos de Hardware. Principais unidades funcionais do computador.	
Sistemas de computação		Noções básicas sobre sistemas de computação.	Noções básicas sobre sistemas de computação.			Conceitos básicos de lógica.
Princípios de Programação	Programação de computadores.	linguagens de programação.	Noções sobre linguagens de programação.		Conceitos básicos de software. Principais softwares básicos. Principais softwares aplicativos.	Linguagens de programação.
Algoritmos	Algoritmo, fluxograma e pseudo-codificação.	Noções sobre algoritmos.		Revisão dos conceitos de algoritmos.	Conceito de algoritmo e programa.	
Dados	Entrada e saída de dados.				Representação de dados.	
Variáveis	Constantes e variáreis.					Sistemas de numeração.
Operadores	Operadores e expressões.					
Estruturas decisão	Desvios e laços.			Estruturas de decisão e controle.	Comandos mais comuns de um sistema operacional.	
Vetores e matrizes	Vetores e matrizes.			Vetores.		
Programação	Programação estruturada.	Programas.	Programas.	Elaboração e implementação de programas em uma linguagem de programação.	Algoritmos: representação, técnicas de elaboração, estruturas para elaboração.	Programação básica.
Linguagem de alto nível	Experimentação em linguagem de alto nível.	Estudo de uma linguagem de alto nível.	Estudo de uma linguagem de alto nível.			
Funções				Funções.		

Fonte: elaborada pelo autor, 2016.

O estudo feito a partir dos dados da Tabela 5 aponta os conteúdos definidos pelas ementas da UDESC para cada disciplina relacionada com os conteúdos iniciais de

programação. Notaram-se diferenças de acordo com o curso, além do que alguns termos deixam dúvidas quanto a sua equivalência de conteúdo. Ampliando a análise das ementas, foi realizado um estudo dos conteúdos programáticos de cada professor, registrados em seus planos de ensino, baseado nas informações disponíveis no site da UDESC. Depois disso, viu-se reforçada a necessidade de realizar as entrevistas com os professores das disciplinas de AP para melhor entendimento dos conteúdos ministrados e do encadeamento dos tópicos para cada curso.

Em etapa seguinte, foi feito o levantamento dos nomes dos professores que ministram as disciplinas relacionadas. Ao todo, foram identificados 13 professores. Com todos eles foi feito um contato por e-mail, convidando-os a participarem de uma reunião onde a pesquisa seria apresentada, quando seria informada a contribuição que cada um poderia dar para o estudo e para o desenvolvimento do produto educacional.

A reunião inicial (antes das entrevistas) foi realizada em 18 de fevereiro de 2016, na sala F107 do CCT-UDESC, com duração de 1 hora, contando com a participação de 5 professores. Nela, os professores participantes foram informados sobre o experimento que estava sendo planejado, sua relação com experimentos anteriores desenvolvidos em outros projetos que, eventualmente alguns já conheciam ou mesmo tinham participado. Foi esclarecida a proposta diferenciada de contar com a participação deles, de forma voluntária e anônima, no planejamento do conteúdo por meio do registro de suas observações e sugestões dadas em entrevistas individuais que seriam gravadas em áudio. Ressaltou-se que os dados das entrevistas seriam considerados na preparação do conteúdo do minicurso, atendendo opiniões expressas pela maioria em caso de divergência sobre algum tema.

Ainda durante a reunião, alguns dos professores presentes declararam apoio à proposta de sua participação no processo de desenvolvimento do minicurso, já se predispondo a colaborar com o que fosse necessário. Nenhum dos presentes se pronunciou contra ou fez qualquer citação que os posicionasse de forma negativa à continuidade da proposta. Não foi feita nenhuma forma de votação, nem registro de nomes ou opiniões, pois isso não era o objetivo deste encontro.

O passo seguinte foi o convite individual, por e-mail, para realização das entrevistas. Foram enviados convites para 13 professores, dos quais 7 aceitaram participar. A partir disso, foi elaborado um cronograma das entrevistas. Foi realizado um encontro com cada professor, de aproximadamente 1 hora cada, no período de 08/06/2016 a 13/06/2016, geralmente em suas salas de trabalho, todos registrados em áudio utilizando um aparelho de celular.

A cada encontro, foi seguido o seguinte roteiro macro, que organizou o conjunto das 23 questões que podem ser consultadas no APÊNDICE D:

- Apresentação e assinatura do TCLE;
- Realização de perguntas sobre o professor e sua disciplina;
- Apresentação da proposta inicial dos conteúdos do MAP, baseada na Tabela 3 ;
- Questões sobre a opinião do professor em relação à proposta;
- Questões sobre a proposta do minicurso como apoio à disciplina;

- Questões investigativas sobre o uso do minicurso em outros contextos.

Mesmo sendo uma entrevista estruturada, com questões claras e pré-definidas, ocorreram nos discursos dos professores algumas transposições de temas, de forma natural, mesmo sem influência do entrevistador. Procurou-se não interromper a fala do professor, apenas interferindo quando se estendia muito além do tema ou do tempo. Desta forma, optou-se por não transcrever as respostas completas, mas dividi-las em trechos que foram então agrupados por temas. As transcrições das falas dos entrevistados são apresentadas no APÊNDICE E, classificadas por tema e agrupados de forma a permitir uma análise comparativa. Cabe ressaltar que as falas dos professores influenciaram na elaboração e nos ajustes dos conteúdos do minicurso.

Após a análise de cada tema abordado nas entrevistas, foi decidido, em conjunto com a professora orientadora, o que seria alterado, o que poderia mudar, e o que permaneceria inalterado. As informações foram resumidas e apresentadas na Tabela 6.

Tabela 6 - Análise das entrevistas com os professores

Certamente será alterado	**Talvez mude**	**Não será mudado**
- Pseudo + C. - Pseudo "como apoio" ao C. - Básico: Matrizes. - Básico: Vetores. - Mais exemplos. - Mais exercícios. - Mais recursos audio-visuais. - Estratégia: exemplos com teste de mesa. - Estratégia: focar mais na compreensão do problema, menos na estrutura e sintaxe. - Estratégia: Usar o ensino de programação para compreender a construir a lógica, não focar tanto nas ferramentas (linguagens). - MDI (pseudo+C) explicitar aonde entra a pseudo nos conceitos 7a 10. - Usar, por exemplo, cores para destacar bem a diferença de exemplos e exercícios em C ou em pseudo. - Buscar exemplos e exercícios dos professores para adaptar no minicurso.	- Pseudolinguagem: Portugol IDE [não mudou] - Temas complementares: procedimentos e funções. [mudou] - Complementar: elementos do computador. [mudou] - Mais interação, atividades no fórum. [não mudou] - Exemplos e exercícios de acordo com o curso. [mudou] - Elaborar os conceitos focando bastante o modo livre de navegação no minicurso. [mudou, em partes] - Laços: avaliar formas diferentes de ensinar – ex. Forbellone. [mudou, em partes] - Planejar intervenções de acordo com o andamento médio das diversas disciplinas. [não mudou]	- Linguagem C. - Conceitos básicos que já estavam estabelecidos. - Exercícios em todos os conceitos. - Avaliação final. - Minicurso como apoio à disciplina presencial. - Tempo de início e duração do minicurso. - Gamificação. - Não gastar o tempo do minicurso com conteúdos informativos, explorar mais a vantagem do recurso online.

Fonte: elaborada pelo autor, 2016.

O resultado das entrevistas foi a definição de uma MDI (Matriz de Design Instrucional) em sua versão final (APÊNDICE F), referência fundamental para a elaboração das etapas seguintes do projeto.

Portanto, pode-se dizer que as entrevistas com os professores se tornaram a principal fonte de referência para especificação dos tópicos abordados. Como materiais, entendem-se todos os recursos necessários para a elaboração do minicurso, em acordo com os objetivos especificado para cada UA seguem a estrutura do ambiente AdaptWeb®:

- Conceitos;
- Exemplos;
- Exercícios;
- Materiais complementares;
- Links de apoio;
- Ferramentas de apoio à programação (compiladores e interpretadores).

No período de 14 de junho de 2016 a 11 de agosto de 2016, todos os materiais foram criados, testados e validados pela Equipe do MAP, conforme descrito na próxima seção. De 12 a 14 de agosto de 2016, todo o minicurso foi validado com professores ou em testes pilotos com convidados.

5.3 ADDIE – Desenvolvimento

Nesta fase foram elaborados todos os conteúdos necessários ao MAP, a partir da MDI definida na fase anterior. Todo o material concebido no design foi inserido no ambiente online do AdaptWeb®, sendo testado e revisado até atingir os objetivos estabelecidos na MDI. Ao final, obteve-se o minicurso completo e funcional no ambiente online, pronto para ser acessado pelos estudantes.

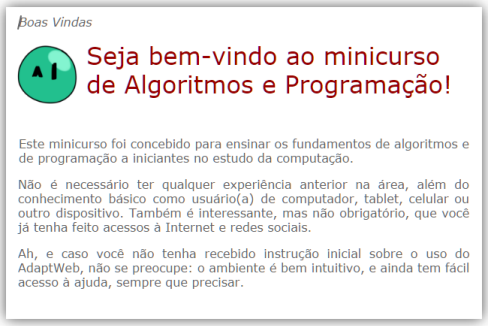
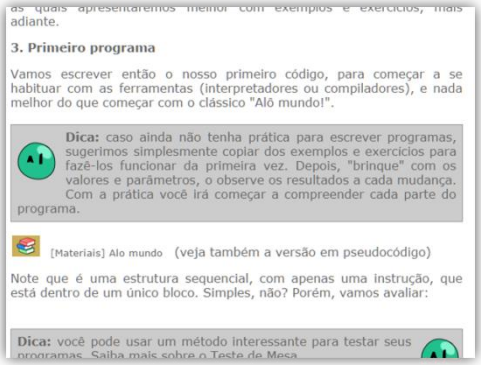
A respeito da produção dos materiais, os conteúdos desenhados anteriormente tiveram que ser repensados levando-se em conta o aprendizado no formato online. A forma de apresentação, como fontes de letras, cores, destaques, ilustrações, foi desenvolvida de modo a permitir clareza e objetividade das informações.

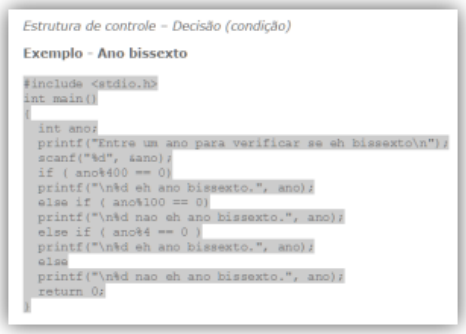
Os conteúdos foram criados fundamentalmente em texto puro, sem formatação, com características de roteiro, e depois transformados para apresentação no ambiente. Todos os objetos criados para o MAP estão no APÊNDICE G.

É importante lembrar que este conteúdo tem o objetivo de desenvolver habilidades e promover a aquisição de conhecimentos pelos estudantes, e por isso deve-se deixar claro nestes roteiros como deve ser a apresentação final de cada Objeto de Aprendizagem, de modo que este objetivo seja alcançado. A disposição dos textos e das imagens, títulos ou caixas de comentários, devem ser destacados para que o objeto final tenha os elementos adequados.

A Tabela 7 mostra alguns exemplos de como esta formatação de conteúdos ocorreu.

Tabela 7 – Exemplos de formatação do texto para o ambiente

Conteúdo em modo texto	Conteúdo formatado
Boas Vindas 01 Seja bem-vindo ao minicurso de Algoritmos e Programação! Este minicurso foi concebido para ensinar os fundamentos de algoritmos e de programação a iniciantes no estudo da computação. Não é necessário ter qualquer experiência anterior na área, além do conhecimento básico como usuário(a) de computador, tablet, celular ou outro dispositivo. Também é interessante, mas não obrigatório, que você já tenha feito acessos à Internet e redes sociais. Ah, e caso você não tenha recebido instrução inicial sobre o uso do AdaptWeb®, não se preocupe: o ambiente é bem intuitivo, e ainda tem fácil acesso à ajuda, sempre que precisar.	
(...) 3. Primeiro programa Vamos escrever então o nosso primeiro código, para começar a se habituar com as ferramentas (interpretadores ou compiladores), e nada melhor do que começar com o clássico "Alô mundo!". ---------- Dica: caso ainda não tenha prática para escrever programas, sugerimos simplesmente copiar dos exemplos e exercícios para fazê-los funcionar da primeira vez. Depois, "brinque" com os valores e parâmetros, o observe os resultados a cada mudança. Com a prática você irá começar a compreender cada parte do programa. ---------- [Materiais] Alo mundo (veja também a versão em pseudocódigo) Note que é uma estrutura sequencial, com apenas uma instrução, que está dentro de um único bloco. Simples, não? Porém, vamos avaliar: ---------- Dica: você pode usar um método interessante para testar seus programas. Saiba mais sobre o Teste de Mesa. [Materiais] Teste de mesa ---------- (...)	

Estrutura de controle – Decisão (condição)
09.1.5 Exemplo - Ano bissexto
 	
#include <stdio.h>
int main()
{
int ano;
printf("Entre um ano para verificar se eh bissexto\n");
scanf("%d", &ano);
if (ano%400 == 0)
printf("\n%d eh ano bissexto.", ano);
else if (ano%100 == 0)
printf("\n%d nao eh ano bissexto.", ano);
else if (ano%4 == 0)
printf("\n%d eh ano bissexto.", ano);
else
printf("\n%d nao eh ano bissexto.", ano);
return 0;
}
	

Fonte: produção do autor, 2017.

Os desenhos que compõem os conteúdos foram criados exclusivamente para o MAP, e depois incorporados no ambiente. O estudante Vitor, da Equipe do MAP, foi o responsável pela criação destas ilustrações exclusivas do minicurso.

Figura 7 – Processo de criação, do croqui à interface do minicurso

Fonte: elaborado pela equipe do MAP, 2016.

A Figura 7 apresenta o processo de criação, desde o croqui da primeira ideia até a implementação na interface do minicurso.

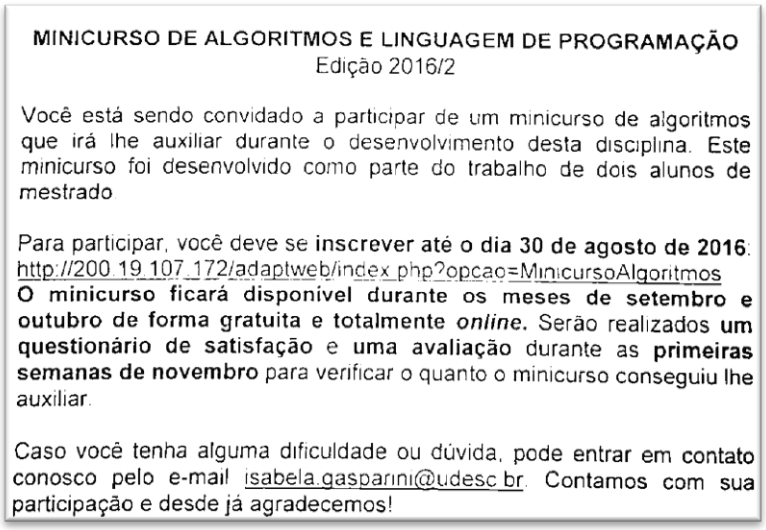
Todas as imagens, animações, gráficos e tabelas presentes no conteúdo foram validadas em suas versões finais, diretamente no ambiente de aprendizagem. Foram realizados vários testes, simulando os diferentes públicos e em situações diversas. Resultaram, então, algumas correções ou adequações, tendo sempre como foco a minimização de problemas e imprevistos para a fase seguinte, a implementação.

Os testes piloto (de conteúdo, materiais, navegação pelo ambiente) foram feitos por professores de disciplinas de AP e por estudantes de grupos de pesquisa do CCT-UDESC.

5.4 ADDIE – Implementação

O MAP foi oferecido a todos os estudantes do CCT-UDESC que estavam matriculados em disciplinas de AP no segundo semestre de 2016. Para tal, foi feita uma apresentação presencial em cada sala de aula, durante as referidas disciplinas, explicando sobre o projeto de mestrado e seu objetivo como apoio às disciplinas presenciais. Nessas apresentações foi entregue a cada estudante uma mensagem impressa, como a apresentada na Figura 8.

Figura 8 – Foto do panfleto original de divulgação do MAP, distribuído no CCT-UDESC



MINICURSO DE ALGORITMOS E LINGUAGEM DE PROGRAMAÇÃO
Edição 2016/2

Você está sendo convidado a participar de um minicurso de algoritmos que irá lhe auxiliar durante o desenvolvimento desta disciplina. Este minicurso foi desenvolvido como parte do trabalho de dois alunos de mestrado.

Para participar, você deve se **inscrever até o dia 30 de agosto de 2016**: http://200.19.107.172/adaptweb/index.php?opcao=MinicursoAlgoritmos
O minicurso ficará disponível durante os meses de setembro e outubro de forma gratuita e totalmente *online*. Serão realizados **um questionário de satisfação** e **uma avaliação** durante as **primeiras semanas de novembro** para verificar o quanto o minicurso conseguiu lhe auxiliar.

Caso você tenha alguma dificuldade ou dúvida, pode entrar em contato conosco pelo e-mail isabela.gasparini@udesc.br. Contamos com sua participação e desde já agradecemos!

Fonte: elaborada pela equipe do MAP, 2016.

Nesta imagem observa-se um texto de apresentação em um panfleto, explicando sobre o projeto de mestrado e do que se trata o MAP. Também foi colocado o link completo para que os estudantes pudessem acessar e se inscrever, caso aceitassem participar do minicurso. Esta inscrição poderia ser feita imediatamente na aula, caso o professor permitisse, ou o estudante levaria o panfleto para se inscrever depois da aula. Os estudantes que aceitaram participar tiveram um período para realizar matrícula (de 15 a 30 de Agosto de 2016). Ao todo, 139 estudantes se matricularam no minicurso, que foi realizado no período de 01 de Setembro de 2016 a 31 de Outubro de 2016. Finalizado o período de divulgação e inscrições, liberou-se o MAP para que os estudantes pudessem acessá-lo online. Neste novo período, os estudantes puderam acessar a qualquer momento, de qualquer lugar, assistindo aos conteúdos ou respondendo às atividades da maneira que melhor lhes conviesse.

Para aqueles estudantes que acessaram uma quantidade mínima de conteúdos, de 01 a 11 de novembro de 2016 foram liberados a avaliação final e o questionário de satisfação. Os estudantes que concluíram o minicurso, num total de 60 receberam certificado de participação.

A respeito de questões relacionadas à capacitação da equipe, observando o modelo ADDIE, foi necessário esclarecer a toda a equipe que colaborou no trabalho, como ocorreriam as atividades nesta fase de implementação. Cada etapa, planejada previamente,

foi estudada pelos responsáveis para que tudo ocorresse sem contratempos e dentro dos prazos previstos. Entende-se por equipe todos os envolvidos nesta etapa de implementação. Além da Equipe do MAP, incluem-se os estudantes, que formam o público-alvo do projeto e os professores das disciplinas de AP, que acompanharam o progresso e desempenho dos estudantes de suas disciplinas durante a realização do minicurso acessando os relatórios online com as notas e dados de análise do aprendizado.

Aos desenvolvedores e organizadores do MAP, coube a revisão das atividades da implementação, como a divulgação, controle de inscrições, acompanhamento dos acessos aos conteúdos e avaliações, tutoria e geração de relatórios. Os professores tiveram um treinamento informal e individual sobre os recursos que poderiam usar para acompanhamento dos seus estudantes, e também sobre a interpretação dos dados em relatórios gerados. No caso dos estudantes, não houve treinamento prévio, mas eles receberam as orientações necessárias, e em momentos adequados, para realizar cada etapa planejada.

Com materiais e recursos prontos, e a equipe capacitada, deu-se início efetivamente à fase de implementação. A partir deste momento os estudantes tomaram ciência do MAP, e o planejamento foi seguido para que o cronograma pudesse ser cumprido nas datas determinadas.

O minicurso contou com intervenções periódicas da equipe desenvolvedora (ver 0). Com o intuito de organizar e manter um ritmo de estudo, além de manter um contato frequente e mais pessoal com o participante, foram enviadas mensagens periodicamente, sugerindo tópicos para estudo. A cada uma destas mensagens foi dado o nome de intervenção. Elas eram enviadas por email ou disponibilizadas em um mural de recados do próprio ambiente.

Tabela 8 – Relação das intervenções da etapa de Implementação

Intervenção nº	Ordem	Data	Conceito(s)
1	Semana 0	qui 01/09/16	1. Boas Vindas
2	Semana 1	dom 04/09/16	2 a 6. Lógica, algoritmos, interpretadores, compiladores, narrativas.
3	Semana 2	dom 11/09/16	7. O que é um programa?
4	Semana 3	dom 18/09/16	8. Dados, variáveis e operadores
5	Semana 4	dom 25/09/16	1 a 8
6	Semana 5	dom 02/10/16	9. Estrutura de Controle - Decisão
7	Semana 6	dom 09/10/16	10. Estrutura de controle – Laços (repetição)
8	Semana 7	dom 16/10/16	11. Vetores e matrizes
9	Semana 8	dom 23/10/16	9 a 11
10	Semana 9	dom 30/10/16	1 a 11
11	Semana 10	seg 31/10/16	1 a 11

Fonte: elaborada pelo autor, 2016.

O conteúdo completo de todas as intervenções pode ser visto no APÊNDICE H.

A seguir, alguns exemplos de intervenções que foram enviadas aos estudantes.

Figura 9 – Intervenção enviada na 1ª semana

Assunto: Início do Minicurso de Algoritmos e Linguagem de Programação - Edição 2016/2

Seja bem-vind@ ao Minicurso de Algoritmos e Programação!

A partir de agora você já pode acessar o ambiente do minicurso através do sistema AdaptWeb.

Para ganhar o **certificado de participação**, você precisa:
- Estudar os conceitos, exemplos, exercícios e materiais complementares que achar importante, acessando o ambiente quando desejar (no período de 01/09 a 31/10);
- Fazer a Avaliação (que estará aberta no período de 01/11 a 11/11);
- Responder o Questionário de Satisfação (que estará aberto no período de 01/11 a 11/11).

Após completado esses itens, você ganhará o certificado de participação com carga horária de 20 horas que poderá ser validado como **Atividade Complementar**. Além disso, teremos um brinde surpresa como um Bônus de participação!

Teremos mensagens periódicas! Durante o curso, você receberá mensagens como esta, para lembrá-l@ de:
- Datas de início e término de atividades.
- Sugestões de estudo para que você não esqueça dos conteúdos abordados, e nem deixe acumular conteúdos para a última hora.
- Propostas de desafios de programação, baseados nos conteúdos sugeridos nas mensagens.

Sugerimos agora que você acesse e explore o ambiente. Aproveite este período inicial para tirar dúvidas e resolver eventuais problemas de acesso.

Obs.: Elaboramos o conteúdo deste minicurso com muito carinho e atenção, mas nem tudo é perfeito!

Caso encontre algum erro, discorde de alguma informação ou tenha sugestões de melhoria, por favor, não hesite em nos informar.
Sua participação é muito importante para nós.

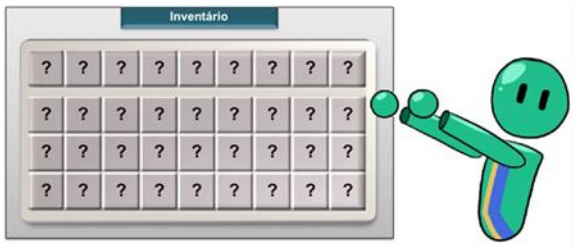
Obrigad@!

Fonte: elaborado pela equipe do MAP, 2016.

Figura 10 – Intervenção enviada na 3ª semana

Hora de programar!!!

Para aprender Algoritmos, diferentes tópicos devem ser trabalhados. No nosso minicurso, estes tópicos podem ser considerados**Coleções de Conhecimentos** para serem usados mais adiante.



Vamos conquistar estes conhecimentos? Sugerimos então, para esta semana, o estudo do seguinte tópico:

O que é um programa?
- Estrutura de um programa (main, system, include);
- Constantes e comandos de atribuição;
- Comandos de entrada e saída (printf, scanf);
- Strings.

Até a próxima semana!

Fonte: elaborado pela equipe do MAP, 2016.

Com uma frequência menor, foram lançados alguns desafios, que eram enviados junto com algumas intervenções. O objetivo foi provocar o estudante de forma a resolvê-los utilizando os conhecimentos de AP. Os conteúdos de todos os desafios constam no APÊNDICE I.

Figura 11 – Intervenção enviada na 4ª semana

Ah, a lógica matemática e o raciocínio abstrato... Uhul!

Vamos entendê-los juntos! (:p) Para esta semana, veja este tópico:

DADOS, VARIÁVEIS E OPERADORES:
- O que são dados e para que servem;
- Como declarar variáveis;
- Operadores aritméticos, relacionais e lógicos.

Estes itens podem ser úteis para o seu aprendizado:

Colocamos uma atividade no Fórum de Discussão! Participe! Você tem até 25/09 para responder. Dia 26/09 daremos a resposta!

Bom estudo!

Fonte: elaborado pela equipe do MAP, 2016.

Observa-se na Figura 11 que os desafios foram anunciados na mensagem, mas eram disponibilizados em um fórum de discussão, acessível pelo próprio ambiente do minicurso.

Segue, na Figura 12, um exemplo de desafio.

Figura 12 – Desafio 2, enviado durante as intervenções

DESAFIO 2:

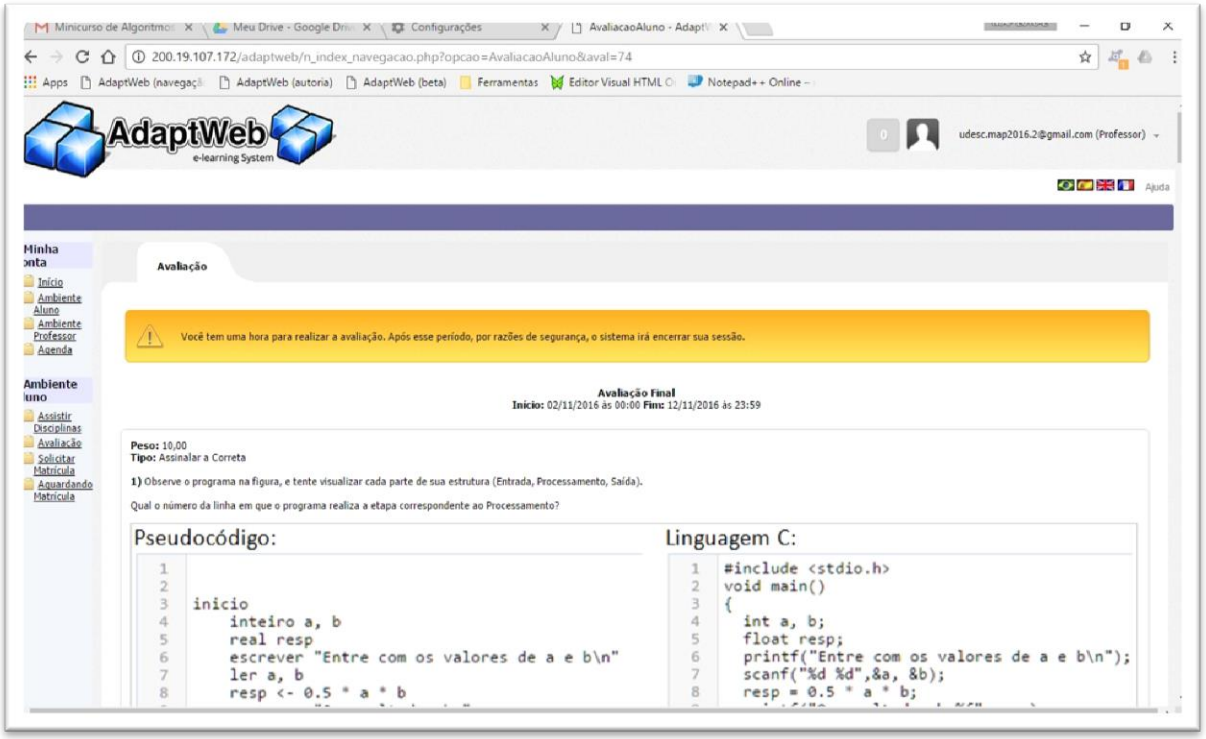
(Prazo até 02/10) Observe a sequência: 2, 5, 10, 17, 26, ... Escreva um programa que forneça o n-ésimo elemento desta sequência. Sabemos que não existe uma única forma de se resolver, e que cada pessoa pensa de maneira diferente. Compartilhe com os outros participantes, através do fórum de discussões, como você desvendou a sequência, e qual estratégia você usou para resolvê-la.

Fonte: elaborada do autor, 2016.

Concluindo-se o período do minicurso, o ambiente não permitia mais que o conteúdo fosse acessado. Neste projeto, o período encerrou-se em Outubro, e em Novembro foi

liberada a avaliação final do minicurso. Na Figura 13, apresenta-se um trecho da tela da avaliação final.

Figura 13 – Trecho da tela da avaliação final



Fonte: AdaptWeb® (2016).

Após o envio das respostas, o estudante recebia um *feedback* caso respondesse incorretamente. Era necessário tirar uma nota superior a 7,0 para o seu desempenho ser considerado satisfatório.

No mesmo período da avaliação final, também foi disponibilizado o questionário de satisfação. Este questionário foi elaborado a partir dos objetivos do minicurso, e é a principal fonte de informação para responder a um questionamento desta pesquisa a respeito do aumento da satisfação do estudante, motivando-o a permanecer no curso, após ampliação de seus conhecimentos sobre AP.

O questionário de satisfação é apresentado no APÊNDICE J.

5.5 ADDIE – *Evaluation* – Avaliação

Esta etapa é apresentada em dois momentos: acompanhamento da execução e revisão.

Acompanhamento da execução: a atividade de avaliação inicia antes mesmo da fase de execução (realização do minicurso pelos estudantes), ao prever todos os envolvidos no processo. Para tanto, foi solicitado aos estudantes o preenchimento de um formulário antes de acessarem o conteúdo do minicurso, denominado “levantamento do perfil inicial”. Da mesma forma, ao final do período de acesso ao minicurso, ficou disponível aos estudantes mais dois questionários: a avaliação de conhecimentos e o questionário de satisfação.

Também foram coletados, de forma automática e transparente ao participante, os dados de acesso ao minicurso.

Revisão: neste momento os estudantes já tinham concluído sua participação no minicurso, não podendo mais acessá-lo sequer para consulta. Isso garantiu que os dados do minicurso não seriam mais alterados, permitindo assim que pudessem ser coletados e tabulados. Os dados coletados na fase de acompanhamento da execução foram tabulados e analisados, podendo ser consultados na dissertação associada a este produto educacional.

6 Considerações Finais

Considerando a história da evolução dos computadores, que começaram a se popularizar a partir dos anos de 1970, o ensino de programação tem sido discutido desde então. A preocupação com as dificuldades de ensino e aprendizagem para estas novas tecnologias já era tema de diversos trabalhos na década seguinte. A grande quantidade de trabalhos recentes sobre o tema indica que essa ainda é uma questão a ser debatida.

O estudo da revisão de literatura, bem como o estudo do processo ADDIE e do ambiente AdaptWeb®, foram essenciais para a criação do minicurso. Além disso, foram relevantes as ideias apresentadas por professores de disciplinas de programação do CCT-UDESC que contribuíram para a seleção e preparação dos conteúdos. Assim, esse conjunto levou à elaboração de uma proposta que poderia auxiliar os estudantes com dificuldades iniciais de aprendizado de AP, por meio de recursos e métodos que aumentassem a sua satisfação ao cursar as disciplinas presenciais, e, por consequência, reduzisse as dificuldades com o raciocínio lógico e o pensamento abstrato para estas e outras disciplinas que envolvam a resolução de problemas.

A elaboração do minicurso exigiu grande planejamento e envolvimento de professores e acadêmicos, com base teórica fundamental para orientação dos trabalhos. Observou-se uma importante colaboração por parte dos professores, o que incentivou o grande número de inscrições. A estratégia de aplicação do minicurso online como apoio à disciplina presencial foi bem aceita pelos estudantes e pelos professores das disciplinas.

A aplicação do minicurso pode ser considerada como uma ação de ensino que utiliza o reforço dos conteúdos iniciais de AP para ajudar no desenvolvimento das habilidades e competências, como o raciocínio lógico-matemático e o pensamento abstrato, necessárias para a resolução de problemas, em disciplinas que necessitam destas habilidades, e ajuda também no desenvolvimento de habilidades das próprias disciplinas presenciais de AP.

O Brasil tem obtido resultados positivos com a melhoria da qualidade de ensino na Educação Básica, pois hoje já se observa uma maior preocupação com o desenvolvimento de habilidades para a resolução de problemas. Para estudantes que estão no final do Ensino Fundamental e início do Ensino Médio ainda é possível recuperar estas dificuldades com ações direcionadas a este público. Mas ainda há um contingente que não teve a oportunidade de receber os benefícios destas melhorias, e já está entrando no Ensino Superior com estas dificuldades acentuadas, e para estes torna-se necessário ações como a apresentada neste trabalho.

Ainda há pouca pesquisa para a Educação Básica, e diversas publicações com busca de soluções para o Ensino superior. Talvez ainda não se dê tanta atenção à formação básica dos estudantes, e por isso ainda se necessitem ações intensas para os ingressantes no Ensino Superior. Pode-se estar, de certa forma, “remediando”, e não “prevenindo”, em relação às dificuldades de aprendizado.

Como sugestão de complemento ou continuação desta pesquisa:

- Adaptar o conteúdo do MAP para o Ensino Médio, e posteriormente para o ensino Fundamental, verificando, a cada um desses grupos, o resultado da satisfação e motivação dos estudos em disciplinas críticas.
- Aplicar o MAP a estudantes concluintes do Ensino Médio, verificando antes quantos são interessados em seguir os estudos na área da Computação na graduação, e destes, quantos continuam interessados após a realização do minicurso.

Pelas características de adaptabilidade, é possível utilizar-se o MAP para outras linguagens e/ou pseudolinguagens de programação, permitindo-se diferentes combinações entre elas. Neste projeto foram utilizadas a linguagem C e uma pseudolinguagem, com o foco na aprendizagem do conceito de programação estruturada. No entanto, poderiam ser utilizadas linguagens tais como Java, C++, Scratch, Python, Ruby on Rails, Haskell), podendo combiná-las de acordo com:

- público-alvo (e.g. Ensino Médio, Ensino Fundamental, Ensino Técnico ou profissionalizante, treinamento de colaboradores de uma empresa);
- objetivo de aprendizagem (e.g., como apoio à disciplinas presenciais, como material didático de uma disciplina ou curso);
- conceito de programação (e.g., estruturada, orientada a objetos, linear, modular);
- classificação das linguagens (e.g., paradigma, tipagem, geração).

Propõe-se como desafio até mesmo a implementação de mais de duas linguagens simultaneamente, permitindo ao estudante não somente optar por uma ou duas, mas conhecer as diferentes possibilidades de cada uma e desenvolver habilidades competências pela comparação prática entre elas.

Outra possível aplicação seria no ensino de uma nova linguagem ou conceito de programação, podendo no MAP ser compreendida por meio da comparação com outras linguagens existentes.

7 Referências

ADAPTWEB. **AdaptWeb (Ambiente de Ensino-Aprendizagem Adaptativo na Web)**. Disponível em: <http://ead.joinville.udesc.br/adaptweb/>. Acesso em: 21 nov. 2017.

DEMO, P. **Pesquisa: Princípio científico e educativo**. 4a. ed. São Paulo: Cortez, 1996.

FILATRO, A. **Design instrucional na prática**. São Paulo: Pearson Education do Brasil, 2008.

KAUARK, F. DA S.; MANHÃES, F. C.; MEDEIROS, C. H. Metodologia da Pesquisa: Um guia prático. p. 88, 2010.

KLOCK, A. C. T. **Análise da influência da gamificação na interação, na comunicação e no desempenho dos estudantes em um sistema de hipermídia adaptativo educacional.** 2017. Dissertação (Mestrado em Ciência da Computação) - Universidade do Estado de Santa Catarina, Joinville.

MIZRAHI, V. V. **Treinamento em Linguagem C**. 2ª ed. São Paulo: Pearson Prentice Hall, 2008.

OLSEN, W. **Coleta de dados: debates e métodos fundamentais em pesquisa social**. [s.l: s.n.].

SAMPIERI, R. H.; COLLADO, C. F.; LUCIO, P. B. **Metodologia de Pesquisa**. 5a. ed. São Paulo: Penso, 2013.

APÊNDICE A **Trabalhos estudados para a Análise Contextual**

Seq	Documento	Breve resumo
1	Abordagem Visual para Aprendizagem Colaborativa de Programação	Uso de aprendizagem colaborativa de programação visual com ambiente 3D em programas desenvolvidos pelos estudantes organizados em times. Baseado em um estudo sobre esquema progressivo para aprendizagem de programação em grupo.
2	Ambiente de Simulação e Animação para o Ensino de Programação	Utilização de softwares de animação como ferramentas educacionais para auxiliar o aprendiz a caminhar da compreensão abstrata para o entendimento real da funcionalidade de uma determinada estrutura de programação.
3	Aplicação de Técnicas da Teoria Das Múltiplas Inteligências No Ensino de Algoritmos e Programação	Uso de técnicas relacionadas à teoria de aprendizagem cognitivas das múltiplas inteligências para amenizar as dificuldades de ensino e aprendizagem de disciplinas que usem algoritmos e linguagens de programação.
4	Complementando o Aprendizado Em Programação: Revisitando Experiências No Curso de Sistemas de Informação da USP	Competições entre equipes para resolução de problemas, para a complementação do aprendizado de lógica de programação, algoritmos e estruturas de dados.
5	Construção e Uso de Ambiente Visual para o Ensino de Programação Introdutória	Uso de ambiente visual de programação para disciplinas de introdução à computação a estudantes do Ensino Fundamental.
6	Desenvolvimento de Um Dispositivo para Apoio ao Ensino de Computação e Robótica	Utilização de dispositivo robótico como elemento motivador ao aprendizado de computação e robótica.
7	Ensino de Programação para Alunos Do Ensino Médio Utilizando o Robô Lego Mindstorms	Ensino de programação à estudantes do EM através da robótica.
8	Ferramenta Computacional de Apoio ao Processo de Ensino-Aprendizagem dos Fundamentos de Programação de Computadores	Implementação de um Sistema Tutor Inteligente que faz uso de uma metodologia de ensino baseada em analogias entre os conceitos de programação e situações comuns do dia-a-dia dos estudantes.
9	Incentivando Meninas Do Ensino Médio À Área de Ciência da Computação Usando o Scratch Como Ferramenta	Utilizar uma linguagem de programação visual para meninas do EM no intuito de aumentar o interesse na área da computação, por meio de sua aplicação no ensino da Química, Física e Matemática, além de ajudá-las pedagogicamente em ciências exatas.
10	Iquiz-Ambiente de Autoria para Avaliação Do Aprendizado No Moodle	Utilização de ferramenta auxiliar (i-Quiz) para ajudar a compreender e reduzir as dificulda- des enfrentadas por professores no emprego do módulo Quiz do Moodle, usado para a produção de questionários.
11	Mojo: Uma Ferramenta para Integrar Juízes Online ao Moodle No Apoio ao Ensino e Aprendizagem de Programação	Utilização de sistemas denominados Juízes Online (repositórios de problemas de competições de programação) para auxiliar o professor de disciplinas de programação na elaboração de questões, com o objetivo de treinar as habilidades em programação de seus estudantes.
12	O Ensino de Programação de Computadores Baseado Em Jogos	Uso de gamificação para ensino de algoritmos e programação de computadores aos estudantes de cursos técnicos e superiores na área de Computação.
13	Portugol Studio: Uma IDE para Iniciantes Em Programação	Uso de ferramenta de programação (Portugol) para auxiliar estudantes iniciantes em lógica de programação.
14	Proposta de Uma Nova Abordagem para Desenvolvimento de Algoritmos de Programação	Uso de abordagem de resolução de problemas e desenvolvimento de algoritmos a partir da “Metodologia da Problematização” e da “Abordagem Baseada em Problemas”.
15	Protótipo de Uma Ferramenta para Auxiliar No Ensino de Técnicas de Programação	Uso de ferramenta educacional para auxiliar professores e estudantes no ensino das disciplinas que envolvem programação de computadores.
16	Trabalhando Lógica de Programação Com Portadores de Deficiência Auditiva: A Experiência Com A Linguagem Proglib e A IDE Hands	Criação de linguagem de programação baseada em LIBRAS (intérprete virtual) para ensino da lógica de programação para deficientes auditivos.
17	Uma Proposta de Ensino de Física para Turmas Noturnas	Ensino da Física a estudantes do EJA, utilizando aplicações do sistema Arduino na área da Física e linguagem de programação. Utilização do sistema POE (Previsão, Observação e Explanação).
18	Uso de Semiótica e Análise de Normas Em Práticas de Ensino de Programação de Computadores Utilizando Robótica Pedagógica	uso de Robótica Pedagógica no ensino de programação de computadores, utilizando conceitos de normas e affordances da Semiótica Organizacional para análise de práticas pedagógicas e avaliações do aprendizado.
19	Uso Do Software Ceebot No Processo de Ensino aprendizagem Nas Disciplinas de Algoritmos e Programação	Resolução de exercícios elaborados com base nas principais dificuldades em relação ao aprendizado de Algoritmos e Programação, apontadas por uma pesquisa de campo com estudantes. Estes exercícios são resolvidos através de uma ferramenta (CeeBot) cujos resultados vão auxiliar no ajuste dos processos de ensino e aprendigagem das aulas de Algoritmos e Programação.

APÊNDICE B **Documentos selecionados do SBIE 2015**

Seq	Documento
1	XP & Skills: gamificando o processo de ensino de introdução a programação
2	Uma dinâmica para ensino de conceitos fundamentais de programação
3	Interdisciplinaridade, programação visual e robótica educacional: relato de experiência sobre o ensino inicial de programação
4	Sala de aula invertida, ambientes de aprendizagem e educação online: a junção de três métodos para potencialização do ensino de algoritmos
5	Uma Experiência Piloto de Integração Curricular do Raciocínio Computacional na Educação Básica
6	Pré-Comp: introduzindo os fundamentos da Computação e contribuindo com a motivação e aproveitamento acadêmico
7	Avaliação de um Jogo Educativo para o Desenvolvimento do Pensamento Computacional na Educação Infantil
8	Aprendendo linguagem de programação através da auto-explicação de exemplos em vídeo
9	Introdução à Robótica e Estímulo à Lógica de Programação no Ensino Básico Utilizando o Kit Educativo LEGO® Mindstorms
10	Uma Experiência no Ensino de Pensamento Computacional para Alunos do Ensino Fundamental
11	Jogos de Programar como uma Abordagem para os Primeiros Contatos dos Estudantes com à Programação
12	Diseño e Implementación de un Taller de Programación de Juegos Digitales con Scratch como Apoyo a Fundamentos de Programación
13	Robótica Educativa na aprendizagem de Lógica de Programação: Aplicação e análise.
14	Programação de computadores para alunos do ensino fundamental: A Escola de Hackers
15	Ensino de programação para Olimpíada Brasileira de Informática
16	Diseño y evaluación de un taller de robótica basado en Estilos de Aprendizaje para la enseñanza de Fundamentos de Programación
17	Softwares Educacionais para o Ensino de Programação: Um Mapeamento Sistemático
18	A Comparação da Realidade Mundial do Ensino de Programação para Iniciantes com a Realidade Nacional: Revisão sistemática da literatura em eventos brasileiros
19	Um Mapeamento Sistemático sobre Iniciativas Brasileiras em Ambientes de Ensino de Programação
20	Robótica Pedagógica Aplicada ao Ensino de Programação: Uma Revisão Sistemática da Literatura

APÊNDICE C Entrevistas com Professores – TCLE

UDESC - Universidade do Estado de Santa Catarina
PPGECMT - Programa de Pós-Graduação em Ensino de Ciências, Matemática e Tecnologias

Aluno: Lucio Vasconcelos dos Santos
Orientadora: Prof. Dra. Isabela Gasparini
Título do Projeto: O ensino de programação no ensino superior: uma proposta para elaboração de conteúdo online como apoio às disciplinas curriculares.

ENTREVISTA COM PROFESSORES - Instrumento para coleta de informações necessárias ao desenvolvimento do projeto de pesquisa de mestrado.

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Propósito do Estudo

Você está sendo convidado a participar de uma entrevista sobre a proposta de um minicurso online, como apoio a disciplinas curriculares que tenham conteúdo relacionado aos conceitos iniciais de programação. O intuito da participação dos professores destas disciplinas, como você, é de auxiliar na elaboração e melhoria do conteúdo do minicurso. Para tal, serão definidas entrevistas estruturadas e previamente agendadas.

O objetivo geral do projeto é avaliar se a utilização de um minicurso, elaborado com técnicas de Design Instrucional e recursos tecnológicos adequados, oferecido por meio de um ambiente *e-learning* chamado AdaptWeb, pode melhorar o aprendizado dos conceitos básicos de programação.

Os alunos participantes do experimento serão solicitados a acessar o ambiente AdaptWeb, e terão acesso aos conceitos de algoritmos e programação. No início do minicurso, os alunos serão orientados a instalar dois aplicativos externos (interpretadores/compiladores de linguagens de programação), necessários para executar algumas tarefas (exemplos e exercícios propostos). Ao final de um período pré-determinado para acessar ao minicurso, uma avaliação será aplicada para verificar os conhecimentos dos alunos. Após a realização desta avaliação, um questionário de satisfação será apresentado.

Reforçamos que este minicurso será aplicado como apoio à disciplina presencial, e não tem a intenção de concorrer ou interferir no planejamento da sua aula.

Procedimento

A entrevista com o(a) professor(a) ocorrerá de forma presencial, mediante agendamento prévio de data, hora e local. Duração estimada para todo o processo da entrevista: 45 minutos.

O entrevistador irá entregar uma cópia impressa deste Termo de Consentimento Livre e Esclarecido a você, que deverá ler e assinar caso concorde com os todos termos expostos.

Como um instrumento auxiliar para a análise dos dados, solicitaremos sua permissão para o uso de gravador de áudio. Você pode decidir se permite este recurso. Os dados gravados serão utilizados somente para a análise, e de modo algum a gravação será exposta.

A entrevista ocorrerá de forma verbal, onde o entrevistador lhe perguntará sobre questões relacionadas às suas disciplinas ministradas na área de algoritmos e programação. Você terá o direito de omitir sua resposta quando desejar.

Caso seja pertinente, o entrevistador poderá complementar as perguntas previamente planejadas, ou até mesmo acrescentar um tempo ao final para conversas ou discussões extras, de forma a enriquecer as informações para a sua pesquisa.

Lembramos que este estudo avalia as informações referentes ao recurso educacional, e não o professor entrevistado.

Riscos e desconfortos

Esses procedimentos não possuem riscos diretos aos participantes, apenas a possibilidade de cansaço para responder as questões. Nós daremos oportunidades de descansos ou intervalos se você achar necessário.

Benefícios de participação

Esperamos que os resultados deste estudo sejam úteis para melhorar a elaboração do recurso educacional online como apoio às disciplinas presenciais. A participação do professor neste momento é fundamental para melhorarmos os conteúdos do minicurso.

A vantagem em participar deste estudo é ter um minicurso com conteúdo bem planejado e voltado para a sua disciplina, obtendo um melhor rendimento e satisfação do aluno em relação aos conceitos ensinados.

Alternativas de participação

A participação deste estudo é voluntária. Você pode se retirar do estudo ou descontinuar sua participação quando quiser, lembrando que cabe ao entrevistado o direito de não responder a qualquer pergunta, se assim o desejar.

Custos e compensação

A participação neste estudo não acarretará nenhum custo a você. Também não iremos realizar nenhum pagamento pela sua participação.

Confidencialidade

Toda informação coletada durante o período de estudo será mantida confidencialmente. A sua identidade será preservada pois cada indivíduo será identificado por um número. A pessoa responsável por conduzir a entrevista é o aluno do Programa de Pós-Graduação em Ensino de Ciências, Matemática e Tecnologias, Lucio Vasconcelos dos Santos.

Nós queremos saber sua opinião. Não existem respostas corretas ou erradas para as questões.

Solicitamos a sua autorização para o uso de seus dados para a produção de artigos técnicos e científicos. A sua privacidade será mantida através da não-identificação do seu nome.

Agradecemos a sua participação e colaboração. Qualquer dúvida sobre o estudo entre em contato com:

Aluno: Lucio Vasconcelos dos Santos <vasconceloss.lucio@gmail.com>
Orientadora: Prof. Dra. Isabela Gasparini <isabela.gasparini@udesc.br>
PPGECMT - Programa de Pós-Graduação em Ensino de Ciências, Matemática e Tecnologias
CCT-UDESC - Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina.

TERMO DE CONSENTIMENTO

Local:_____, Data:____/____/____

Declaro que fui informado sobre todos os procedimentos da pesquisa e, que recebi de forma clara e objetiva todas as explicações pertinentes ao projeto e, que todos os dados a meu respeito serão sigilosos. Eu compreendo que neste estudo, as respostas fornecidas na entrevista serão utilizadas, mas que meu nome não será citado em nenhum momento.

( ) Aceito gravação de áudio durante a entrevista.

( ) Não aceito gravação de áudio durante a entrevista.

Nome: _____

Assinatura: _____

APÊNDICE D Entrevistas com Professores – Roteiro do entrevistador

UDESC - Universidade do Estado de Santa Catarina
PPGECMT - Programa de Pós-Graduação em Ensino de Ciências, Matemática e Tecnologias

Aluno: Lucio Vasconcelos dos Santos
Orientadora: Prof. Dra. Isabela Gasparini
Título do Projeto: O ensino de programação no ensino superior: uma proposta para elaboração de conteúdo online como apoio às disciplinas curriculares.

ENTREVISTA COM PROFESSORES - Instrumento para coleta de informações necessárias ao desenvolvimento do projeto de pesquisa de mestrado.

ROTEIRO DO ENTREVISTADOR (mestrando - Lucio Vasconcelos)

Ao iniciar estes passos, o professor entrevistado já deverá ter recebido um e-mail convite para esta entrevista, com TCLE anexo para leitura prévia, solicitando data e local para a sua realização.

- ❑ Confirmar envio do e-mail convite.

Passo 1: antes da entrevista, conferir material necessário ao entrevistador:

- ❑ 02 cópias impressas do TCLE, cada cópia em uma única folha frente e verso (01 será recolhida após assinatura).
- ❑ 01 cópia impressa do roteiro do entrevistador (se não tiver computador - para registro manual das respostas).
- ❑ 01 cópia do roteiro do entrevistado (poderá ser entregue se solicitado, somente no final da entrevista).
- ❑ MDI(s) e descritivo dos conteúdos do minicurso (apenas para mostrar ao entrevistado).
- ❑ Computador com acesso à rede/Internet, se possível (neste caso não precisará do roteiro do entrevistador).

Passo 2: ainda antes da entrevista, rever os planos de aula e materiais do(a) professor(a) que será entrevistado(a).

- ❑ Rever planos de aula e materiais do entrevistado.
- ❑ Verificar se o gravador está funcionando, com memória, etc

SESSÃO 1 - INTRODUÇÃO E TCLE.

ENTREVISTADOR: leitura e assinatura do TCLE.

Passo 3: apresentar-se:

- ❑ Falar brevemente sobre você (aluno do PPGECMT, sob orientação da prof. Isabela).
- ❑ Falar brevemente sobre o projeto de pesquisa (ensino de programação introdutória no ES, proposta para minicurso online como apoio às disciplinas presenciais).
- ❑ Explicar o motivo desta entrevista (coletar informações para auxiliar na elaboração do conteúdo e estrutura do minicurso, em acordo com as disciplinas envolvidas e seus professores).

Passo 4: entregar 01 via do TCLE:

- ❑ Entregar TCLE ao professor.
- ❑ Explicar que o Termo de Consentimento é um documento que explica o que é a pesquisa, de maneira que ele, entrevistado, possa tomar uma decisão sobre a sua participação ou não nesta entrevista.
- ❑ Lembrar ao entrevistado que ele já recebeu o TCLE previamente por e-mail.

Passo 5: Revisão dos principais pontos do TCLE:

- ❑ Perguntar se o entrevistado deseja reler ou se prefere que você faça um breve resumo dos principais pontos:

Passo 6: Se o entrevistado não leu previamente o TCLE ou prefere que você resuma:

- ❑ Falar sobre o objetivo do estudo (avaliar se um minicurso online, elaborado com estratégias didáticas e usando recursos tecnológicos adequados, pode ajudar na compreensão dos conceitos básicos de programação).
- ❑ Ressaltar a importância da participação e validação do professor (a experiência e opinião do professor neste momento é importante para a melhor integração do conteúdo ministrado com o que está sendo projetado, além de planejar atividades, exercícios e avaliações compatíveis).
- ❑ Tempo e forma da entrevista (35-45 minutos, em 05 partes ou sessões, perguntas verbais).
- ❑ Lembrar que este estudo avalia as informações referentes à disciplina e seu conteúdo, e não o professor entrevistado.
- ❑ As informações compartilhadas serão usadas em artigos e a identidade do professor não será revelada.

- ❏ O entrevistado poderá se omitir a responder a qualquer das questões.
- ❏ O entrevistado poderá desistir de continuar participando a qualquer momento.
- ❏ Deixar claro que o minicurso não concorrerá com a disciplina presencial, não tem a intenção de interferir de forma alguma no plano de aula do professor.

Passo 7: assinatura do TCLE:

- ❏ Solicitar que o entrevistado assine o TCLE.
- ❏ Enquanto ele lê ou assina, preencher a data e o nome do professor.
- ❏ Recolher o TCLE assinado e entregar a outra via impressa para o professor.

ENTREVISTA - Tempo total 40 a 55 minutos, divididos nas etapas abaixo:

Sessão	Tema	Qtd. questões	Tempo estimado
1	Introdução e TCLE	-	5-10 minutos
2	Sobre o(a) professor(a) e sua disciplina	12	10 minutos
3	Apresentação das MDIs	01	5-10 minutos
4	Sobre a proposta do minicurso	07	10 minutos
5	Questões investigativas	03	5 minutos
TOTAL		23	35-45 minutos

SESSÃO 2 - SOBRE O(A) PROFESSOR(A) E SUA DISCIPLINA

2.1 DADOS DO(A) PROFESSOR(A)

Q1. Qual data da entrevista e o nome do(a) professor(a) entrevistado(a)? Você permite que a entrevista seja gravada (somente áudio)?

Data e hora da entrevista:	
Nome do(a) professor(a) entrevistado(a):	

Q2. Que disciplinas relacionadas ao ensino de programação introdutória você ministra? E quantas turmas por curso?

	Qtde.	Curso	Fase	Disciplina	
a)		Ciência da Computação	1ª	AGT0001	Algoritmos
b)		TADS	1ª	AGT0001	Algoritmos
c)		Engenharia Civil	2ª	ALP0001	Algoritmos e Linguagem de Programação
d)		Engenharia Elétrica	1ª	ALP0001	Algoritmos e Linguagem de Programação
e)		Engenharia Mecânica	1ª	PRE1002	Programação para Engenharia I
f)		Engenharia Produção e Sistemas	1ª	ICC0001	Introdução à Ciência da Computação
g)		Engenharia Produção e Sistemas	2ª	APG0001	Algoritmos e Programação
h)		Licenciatura em Física	3ª	ALP0001	Algoritmos e Linguagem de Programação
i)		Licenciatura em Matemática	5ª	ALP0001	Algoritmos e Linguagem de Programação

Q3. Dá aula a quanto tempo (neste tema)?

- ❑ a) Até 6 meses;
- ❑ b) Mais que 6 meses até 1 ano;
- ❑ c) Mais de 1 ano até 3 anos;
- ❑ d) Mais de 3 anos.

Q4. Já conhece o AdaptWeb? Se sim, qual a experiência com a ambiente?

- ❑ a) Somente se cadastrou (aluno ou professor).
- ❑ b) Navegação como aluno.
- ❑ c) Navegação como professor de disciplina
- ❑ d) Como professor de disciplina (acompanhamento dos alunos).
- ❑ e) Como professor conteudista (criação de cursos e conteúdos)

2.2 EM RELAÇÃO À(S) SUA(S) DISCIPLINA(S)

O que eu pretendo saber:

- O que professor acrescenta ou exclui de conteúdo (e o quê);
-

Q5. O conteúdo programático e referências da sua disciplina são seguidas rigorosamente ou são adaptadas de acordo com o andamento das aulas (ou desempenho dos alunos)?

- ❑ a) Sim, sigo rigorosamente o conteúdo programático..
- ❑ b) Não, varia de acordo com o andamento das aulas.
- ❑ c) Não, varia de acordo com o desempenho dos alunos.

Q6. Se o conteúdo não é seguido rigorosamente, quais tópicos tem a mais ou a menos, e em que ordem (se relevante)?

Tópicos básicos (acreditamos que estes são apresentados primeiro...):

- ❑ ( ) a) INTRODUÇÃO AOS CONCEITOS DE LOGICA E PROGRAMAÇÃO
- ❑ ( ) b) ALGORITMOS (narrativas)
- ❑ ( ) c) PROGRAMA(estrutura, comandos entrada-saída, ...)
- ❑ ( ) d) DADOS, VARIÁVEIS E OPERADORES
- ❑ ( ) e) ESTRUTURAS DE DECISÃO (CONDIÇÃO)

- ❑ ( ) f) ESTRUTURAS DE REPETIÇÃO (LAÇOS)

Tópicos complementares (podem ter tópicos similares com nomes diferentes...):

- ❑ ( ) g) ARRAYS (não apenas matrizes, envolve arranjos com textos e números)
- ❑ ( ) h) VETORES (ou matrizes unidimensionais)
- ❑ ( ) i) MATRIZES
- ❑ ( ) j) PONTEIROS
- ❑ ( ) k) MANIPULAÇÃO DE STRINGS
- ❑ ( ) l) PROCEDIMENTOS E FUNÇÕES
- ❑ ( ) m) MODULARIZAÇÃO
- ❑ ( ) n) REGISTROS
- ❑ ( ) o) OPERAÇÕES COM ARQUIVOS

Q7. De que forma (e quando) você prepara seus alunos?

- ❑ Exercícios em sala
- ❑ Trabalhos individuais
- ❑ Trabalhos em equipe
- ❑ Avaliações
- ❑ Atividades extras
- ❑ Sugestão de links de sites
- ❑ Indicação de livro na biblioteca
- ❑ Outros:

Q8. De que forma (e quando) você incentiva seus alunos?

- ❑ Exercícios em sala
- ❑ Trabalhos individuais
- ❑ Trabalhos em equipe
- ❑ Avaliações
- ❑ Atividades extras
- ❑ Sugestão de links de sites
- ❑ Indicação de livro na biblioteca
- ❑ Outros:

Q9. De que forma (e quando) você avalia seus alunos?

- ❑ Exercícios em sala
- ❑ Trabalhos individuais
- ❑ Trabalhos em equipe
- ❑ Avaliações
- ❑ Atividades extras
- ❑ Sugestão de links de sites
- ❑ Indicação de livro na biblioteca
- ❑ Outros:

Q10. (caso o professor use incentivos, na questão 9) Você utiliza incentivos diferentes dependendo do curso?

Q11. (se o professor dá aula em cursos diferentes, na questão 3) Que tipos de exemplos e exercícios em sala você acha importante para cada curso (tem diferenças)?

Q12. (considerando os tópicos básicos de programação...) Na sua opinião é melhor ensinar a pseudolinguagem junto com a linguagem C, ou é melhor cada um separadamente?

- ❑ Pseudolinguagem junto com a linguagem C.
- ❑ Primeiro pseudolinguagem, depois a linguagem C.
- ❑ Outras opções:

Q13. Você acha interessante a realização de um minicurso online como apoio às disciplinas presenciais do CCT?

SESSÃO 3 - APRESENTAÇÃO DA MDI

ENTREVISTADOR: apresentação da MDI proposta para o minicurso (de acordo com a resposta da questão 7), do ambiente e dos resultados esperados para o projeto. apresentação do planejamento. Mostrar como definimos a sua abrangência e a duração.

2 semanas de matrícula (de 01/08 a 14/08)
2 meses para acesso ao conteúdo (15/08 a 14/10)
1 semana de avaliação (15/10 a 21/10) > mais ou menos no meio do semestre.

Q14. O tempo previsto e o conteúdo proposto se adequam a(às) sua(s) disciplina(s)? O que sugere? (estender ou reduzir a duração, começar mais tarde, …)

Q15. Sobre a proposta apresentada (MDI), as estrutura selecionada ficou clara? Dúvidas sobre os conteúdos de cada conceito (comentários aqui. Sugestões de alteração, na próxima pergunta)

SESSÃO 4 - SOBRE A PROPOSTA DO MINICURSO (segue a sequência das colunas na MDI)

4.1 OBJETIVOS E CONTEÚDOS

O que eu pretendo:
- Apresentação da proposta do minicurso;
- Sugestões de alterações do conteúdo e ordem.

Q16. Se a proposta não atende totalmente o que você espera, quais alterações sugere?

4.2 EXEMPLOS E EXERCÍCIOS

ENTREVISTADOR: apresentação de alguns exemplos e exercícios do AdaptWeb.

Q17. Nos tipos de exemplos apresentados, existem diferenças em relação ao que você aplica em sala? Quais?

4.3 MATERIAIS LINKS DE APOIO

Q18. Existem materiais e links específicos para a área que você indica a seus alunos? Pode citar alguns?

4.4 TEMPOS E DURAÇÃO

Q19. Quanto tempo você acredita que o aluno vai disponibilizar, por semana, para o minicurso?

4.4 AVALIAÇÃO

A avaliação do minicurso abrange dois aspectos: o aprendizado do conteúdo proposto e a satisfação quanto ao formato e conteúdo do minicurso. Serão dois questionários distintos aplicados após o cumprimento do período destinado ao estudo do conteúdo pelos alunos.

ENTREVISTADOR: apresentação das questões da avaliação de aprendizagem.

Q20. Qual sua opinião sobre a avaliação proposta para avaliar a aprendizagem do aluno?

SESSÃO 5 - QUESTÕES INVESTIGATIVAS

O que eu pretendo saber:
- Sugestões do professor de outros usos para o minicurso;
- se aplicado antes do semestre;
- se aplicado depois do semestre.

Q21. Além do minicurso ser utilizado como apoio à disciplina presencial, você acredita que ele pode ser realizado em outros contextos? Quais?

Q22. (caso não tenha sido comentado na questão anterior) O que você acha do minicurso ser aplicado no período de férias acadêmicas antes do início das aulas, para alunos repetentes ou aos que vão iniciar seu curso de graduação?

Q23. E se o minicurso puder ser aplicado como reforço ao término do semestre, aos alunos que tiveram dificuldades durante as aulas?

APÊNDICE E **Transcrições das falas dos professores**

Sobre os conceitos:

- Falar dos elementos do computador, só que "homeopaticamente".
- Mostrar o conceito, o problema, e quais as ferramentas disponíveis para resolver (deixar o estudante construir a partir da necessidade).
- Iniciar com algoritmos para puxar a abstração.
- Falar da ambiguidade, porque a linguagem de programação é mais adequada que a linguagem natural (algoritmo não deixa de ser uma conceituação matemática, não pode ter ambiguidade).Envolver os estudantes já nas primeiras aulas, de forma até divertida (resolver desafios algorítmicos não-computacionais).
- Fazer link de algoritmos com a vida cotidiana, associar ao computador, mostrando quais problemas o computador resolve, e como usar algoritmos para resolvê-los.
- Fazer com que os próprios estudantes criem os comandos, a partir da necessidade.
- Usar a programação para aprender a construir a lógica. O foco não é instituir uma linguagem, mas sim a lógica.
- Passar visão ampla da programação e sua importância, para que o estudante futuramente, como profissional, saiba como ir atrás da informação.
- Para começar a falar de programa, faz uma "brincadeira" como se fosse um simulador de robô (ex.: o robô só entende essas instruções, e vocês devem, utilizando essa linguagem que ele entende, fazer determinada tarefa (ideia de formalização).
- Mostrar resultados rapidamente. O estudante é muito visual. Se o professor passa o conceito e ele não vê funcionando, eles não entendem ou aprendem.
- Não introduzir novos conceitos junto com os de programação, mesmo sendo da área/curso destes estudantes. Ao fazer isso, o professor está introduzindo uma nova dificuldade, ou seja, além deles terem que aprender programação vão ter que aprender um outro conceito, que não lhes é familiar ainda (mas isso é diferente se os estudantes já está lá pela 5a ou 6a fase).
- Procurar apresentar os temas em áudio-visual, porque os estudantes não lêem (ex.: os estudantes com dificuldade durante a aula não costumam consultar no Google, mas sim no Youtube). Eles não querem saber, por exemplo, o nome do comando, eles querem saber como usar.
- Procurar montar os conceitos "orientados a exemplos".
- Não precisa ampliar tanto os conceitos do minicurso. Considerando o que é básico de programação, já está ótimo para o objetivo do minicurso. Como é usado como apoio à disciplina presencial, qualquer outro conceito adicional pode ser melhor explorado pelo professor em sala de aula.
- O minicurso é bastante válido para a análise. O estudante pode não aprender a programar, mas vai aprender a analisar. Essa parte de análise na sala de aula é muito mais cansativa - o curso online ajuda bastante nisso. Fazer a análise de algoritmo ajuda bastante depois a programar.

Sobre os exemplos:

- Usar exemplos reais, do dia-a-dia. Idem Exercícios.
- Focar mais na compreensão dos problemas, e não na estrutura da linguagem ou na sintaxe.
- Usar testes de mesa.
- Mudar a cor dos exemplos, por exemplo, para ficar claro ao estudante quando está na linguagem C ou quando está na pseudolinguagem. Idem Exercícios.
- Para vetores (e matrizes) tem muito exemplo na internet, mas não mostrar muitos (nem sempre é bom, porque o estudante acaba copiando e não aprende).
- Usar GIFs animados para explicar as estruturas.
- Evitar usar slides.
- Mostrar o programa primeiro numa estrutura sequencial, e depois mostrar a simplificação colocando a estrutura de repetição.
- Elétrica: calcular a tensão elétrica num circuito. Idem Exercícios.
- Engenharias: dificuldades em dados, variáveis e operadores, por causa de Cálculo (na computação isso geralmente não acontece). Idem Exercícios.

Sobre os exercícios:

- Coisas práticas funcionam melhor. Fazer um parte inicial mais teórica, e no restante bastante prática.
- Propor muitos exercícios, e diferentes.
- Propor muitos exercícios de interpretar o que um programa faz.
- Focar nos exercícios com respostas objetivas, pois em sala o professor pode focar na construção dos programas.
- Começar com os exercícios básicos, e na sequência fazer variações (se o estudante só decorar, ele não saberá fazer as variações).
- Colocar alguns exercícios mais complexos, para prender a atenção dos que já sabem.
- Propor exercícios para o estudante começar do zero (ele tem de sentir a necessidade de gerar ou decidir usar uma variável).
- Usar questionários tipo quizz (se o estudante não responder em no máximo 6 segundos, ele passa para a próxima).
- Colocar no minicurso problemas que exijam que o estudante aplique o conceito que foi aprendido na aula presencial (ideia de "ampliação do escopo").
- Fonte: maratona de programação.
- Fonte: SciLab.
- Fonte: URI.
- Fonte: apostila da UFMG.
- Tema: Fibonacci.
- Tema: Fatorial.
- Tema: número primo.
- Tema: cálculo dos números perfeitos.
- Tema: calcular um percurso usando matrizes.
- Tema: travessia.
- Tema: preencher matrizes com os resultados dos exercícios.

Sobre a interatividade estudante-estudante, estudante-professor e estudante-conteúdo:

- Indicar para os estudantes estudarem em grupo, mas focar nos desafios mais individuais. Tem que variar entre individual e grupo, para não se acomodarem.
- Usar bastante a interação com o estudante.
- Indicar para os estudantes estudarem em grupo, mas focar nos desafios mais individuais. Tem que variar entre individual e grupo, para não se acomodarem.
- Criar atividades para incentivar a participação dos estudantes.

Sobre as estratégias didáticas:

- Ensinava antes Depuração, mas agora usa o teste de mesa (que é semelhante mas o resultado é melhor).
- Tem um livro texto, slides que vai disponibilizando de acordo com o andamento na sala de aula. Não libera antes

de passar em sala de aula.
- Prefere que os estudantes foquem mais na parte de compreensão do problema, de estruturar dados.
- Adotou no 1o mês sem computador, só no papel e caneta, mesmo no pseudocódigo, até começar o IF. A 1a prova foi no papel.
- Coisas mais práticas funcionam melhor. Dá 1 aula teórica e o resto é aula prática.
- Fazer com que os próprios estudantes criem os comandos, a partir da necessidade.
- Passar visão ampla da programação e sua importância, para que o estudante futuramente, como profissional, saiba como ir atrás da informação.
- Usar a programação para aprender a construir a lógica. O foco não é instituir uma linguagem, mas sim a lógica.
- Investigar o nível de conhecimento dos estudantes na área, e descobrir quais são os vícios deles.
- Colocar os que sabem mais para ajudar aos que tem dificuldade.
- A parte introdutória, faz por vídeos (bem rápido), mostra figuras, leva dispositivos físicos.
- Para começar a falar de programa, faz uma brincadeira como se fosse um simulador de robô (ex.: o robô só entende essas instruções, e vocês devem, utilizando essa linguagem que ele entende, fazer determinada tarefa (ideia de formalização).
- No laboratório, fazer atendimento em máquina.
- Comentar sobre filmes, por exemplo, como "O Vale do Silício", fala sobre a história da Apple.
- Mostrar resultados rapidamente. O estudante é muito visual. Se o professor passa o conceito e ele não vê funcionando, eles não entendem ou aprendem.
- Usar bastante a interação com o estudante, brincar, ter um companheirismo com o estudante.
- Atendimento exclusivo, eventualmente. Tem estudantes que vão ficando para trás lá no início, e o professor dá uma atenção especial, senta junto, um por um, dando atendimento na máquina. Assim ele recupera a maioria dos estudantes.
- Primeiro conhecer bem os estudantes, conversar bastante, tentar saber o nome de cada um, essa questão de fazer muitas aulas no laboratório e passar por cada um também ajuda, até para conhecer as dificuldades de cada um.
- Indicar os monitores com certa antecedência (1,5 mês).
- Indicar para os estudantes estudarem em grupo, mas focar nos desafios mais individuais. Tem que variar entre individual e grupo, para não se acomodarem.
- Tem estudantes que já programam muito, e ficam incomodados quando o professor chega perto. Nestes casos é melhor se afastar, respeitar o estudante.
- Não introduzir novos conceitos junto com os de programação, mesmo sendo da área/curso destes estudantes. Ao fazer isso, o professor está introduzindo uma nova dificuldade, ou seja, além deles terem que aprender programação vão ter que aprender um outro conceito, que não lhes é familiar ainda. Portanto, é melhor trabalhar para que o foco seja na solução do problema, e não criar um problema novo. Se os estudantes já estiverem lá pela 5a, 6a ou 7a fase, já é diferente. É possível pedir para eles usarem os recursos computacionais para fazer um cálculo avançado da área deles, por exemplo.
- Incentivar a participação em aula, chamar todos pelo nome, esse envolvimento com os estudantes é importante.
- Usar "fragmentos de pontos" por participação em sala. Os estudantes chegam a se envolver tanto que até cobram do professor. Apesar do receio de estar dando pontos apenas para quem não precisa, percebe uma participação geral, com algumas exceções, mas o grupo fica motivado.
- Apresentar aos calouros, logo no início, estudantes do mesmo curso/área deles que estão mais avançados e participando, com sucesso, de grupos, eventos ou competições.
- Depois das 3 semanas iniciais, passar temas diferentes para os estudantes, por exemplo, ao falar de Sistemas operacionais, não mencionar o Windows. Dar uma aula sobre Linux.
- Na 1a aula de laboratório, ninguém senta nas cadeiras, ficam em pé no quadro com o professor. É feito uma revisão de tudo o que foi visto na teoria. Só então são passadas as atividades que serão feitas ali no laboratório. O professor tomou esta iniciativa porque cansou de ver excelentes professores dando aula e os estudantes ficarem no facebook, Internet, ou seja, ninguém prestando atenção. Não há como evitar. Por isso, o professor faz essa "preleção" em 15 a 20 minutos, toda aula. Com isso, o desgaste dele é mínimo em laboratório.
- O professor sugere fortemente a apresentação dos temas em áudio-visual, simplesmente porque os estudantes não lêem (ex.: os estudantes com dificuldade durante a aula não costumam consultar no Google, mas sim no Youtube). Eles não querem saber, por exemplo, o nome do comando, eles querem saber como usar.
- Procurar montar os conceitos "orientados a exemplos". Sugere também que pensemos para o nosso curso a ideia de "ampliação do escopo", ou seja, colocar no curso problemas que exijam que o estudante aplique o conceito que foi aprendido na aula presencial.
- O conteúdo está muito bom, o que é básico está perfeito, não precisa alterar. O que passa disso, é aprimoramento, é a sequência do conteúdo, e isso não precisa estar aqui. Está ótimo.

Sobre os elementos do computador:

- Junto ao (a) Introdução, se inclui álgebra de Boole e sistemas de numeração, histórico da computação, conversão de base (mas não se aprofunda), um pouco de circuitos lógicos. São esses os conteúdos que ele acha importante para que os estudantes possam construir os conceitos que virão na sequência (como IF por exemplo).
- Não dá para chegar em todos estes conteúdos. Ele incluiria no início, elementos do computador. O professor "perdeu" pelo menos umas 3 semanas até explicar tudo isso, como exemplo o que tem dentro de um smartphone.
- Falar dos elementos do computador, como a memória RAM e a forma de manipular a memória, mas sempre "homeopaticamente".
- Compreender o funcionamento do computador para poder imitar este comportamento, e programar este comportamento.
- Associar os novos conceitos de comandos às partes do computador. O computador só vai entender o que está dentro da limitação dele. Se ele tem dispositivos de entrada, tem comandos que acionam estes dispositivos de entrada. o mesmo para as saídas, memórias (armazenar), processamento (calcular). Só a partir daí é que se começa o algoritmo computacional.
- Fazer o uso , já de início, o paralelo da teoria-máquina.
- Acrescentaria, antes de todos os outros conteúdos, sobre a estrutura do computador (1 aula), para poder entender algumas características da linguagem C, e poder referenciar quando necessário, por exemplo, vetores e matrizes.

Sobre os algoritmos:

- Iniciar com algoritmos para puxar a abstração.
- Falar da ambiguidade, porque a linguagem de programação é mais adequada que a linguagem natural.

Algoritmo não deixa de ser uma conceituação matemática, não pode ter ambiguidade.
- As primeiras aulas são absolutamente cruciais, são as aulas de envolvimento, trazendo um grau até de diversão, fazendo eles resolverem desafios algorítmicos não computacionais, que costumam ser muito divertidos.
- Dar o conceito de algoritmos logo no início, geralmente a 1a e 2a aula, e fazer o link com a vida cotidiana (exemplos como a travessia, como se vestir, fazer pipoca, trocar pneu, etc). Depois começa a explicar o computador, as suas partes, e associa com o comando, o que o computador tem disponível, uma parte mínima de hardware, componentes básicos, e faz essa associação dos conceitos já aprendidos de algoritmos com o computador.
- A partir da 4a aula é que começa a parte "dura", quais problemas o computador resolve, e como usar algoritmos para resolvê-los.

Sobre a linguagem de programação:

- Usa o C, mas seria melhor outras linguagens (como o "falecido" Pascal), ou alguma de tipagem, indentação, teste de mesa.
- As linguagens escolhidas, como o C, é uma demanda da Elétrica. Deveriam ser outras como o Python, mas sabe que ela é muito ágil para ensinar os estudantes, e já existem muitos estudos de linguagens funcionais como o Haskell, que requer o estudo de recursividade. Também existe o Prolog, largamente utilizado atualmente em muitas universidade no mundo. Para se ter uma ideia, é possível se ter em 2 a 3 semanas estudantes iniciantes já fazendo programas avançados, por exemplo, resolvendo problemas do Racha-Cuca, como o Teste de Einstein. Também há outra linguagem, como o Picat, que ele considera um grande passo após o Prolog.

Sobre a pseudolinguagem:

- Algumas turmas, abordou mais o VisuAlg, mas como ferramenta de apoio, uma "bengala".
- Optou por não usar o VisuAlg.
- Em AGT, a turma C é só de repetentes, eles tinham resistência ao C. Então o foco foi a linguagem (não faz sentido falar em pseudo).
- Usar ferramentas (como o VisuAlg). Quando o estudante interage com a máquina, ele tem um interesse maior.
- Introduzir a pseudolinguagem (o VisuAlg) mais para o final de Algoritmos, como intermediário antes do C, principalmente, para eles verem que o algoritmos tinha vida e para acompanhar os resultados.
- O professor não usa a pseudolinguagem, pois acredita que esta vertente da pseudolinguagem não ajuda a fazer código. Usando a pseudolinguagem, você perde muitos elementos, como quando você fala de compilador, são 3 fases, código fonte, código objeto, código executável. Numa pseudolinguagem você não vê nada disso. Eles aprendem então linhas de comando, gerar o objeto, gerar o executável.
- Na proposta pseudolinguagem e depois C, no momento que você termina a pseudolinguagem e volta para explicar tudo de novo em C, você perde os estudantes bons. o professor não gosta deste processo "não-linear" (ir e voltar no mesmo conceito). É como se estivesse "brincando" com o tempo do estudante.

Sobre a adaptabilidade por curso:

- Vai adaptando conforme o andamento da turma, na questão pedagógica, mas não foge muito do que foi planejado. Adapta conforme a turma (Civil ou Produção), pois andam diferente, mesmo sendo a mesma disciplina.
- No caso de ALP, tem uma programação para ver introdução a ponteiros, mas geralmente é feito de forma muito vaga, apenas para dar uma ideia do assunto aos estudantes.
- Pela experiência, a 1a coisa que o estudante faz é colocar código no editor e ver se compila ou não, não pensa na lógica de programação (tentativa e erro). Então decidiu atrasar um pouco a ida ao computador, e foi direto ao C, sem usar pseudolinguagem.
- Se o professor tivesse a mesma disciplina em mais de um curso, mudaria em termos de tipos de algoritmo a ser desenvolvido. Ex.: para Elétrica, calcular a tensão elétrica num circuito. Na Matemática, poderia ser cálculo de séries, por exemplo.
- Não vê diferença. O estudante gosta de exercícios que ele vê que é uma coisa do dia a dia, não é tão filosófico. Ex.: um menu, ele vê funcionando e vê utilidade no dia a dia. Independente do curso.
- Já usou em outro curso (Mecânica) o Sci Lab, e deu bastante certo. Quanto mais exercícios, e diferentes, melhor como incentivo para que eles aprendam.
- No início há dificuldade de estudantes das engenharias em Dados, Variáveis e Operadores por causa da disciplina Cálculo, eles confundem bastante. Na computação isso geralmente não acontece. Algumas turmas, como a de repetentes, o professor procura explicar de maneira diferente (as vezes já sai implementando VisuAlg), insiste que eles implementem durante a aula, e a motivação é desafiar através de problemas, a entrega.
- Isso depende muito de qual é o curso. Quando é para computação, ele "pega mais pesado", pois é a área deles. Apesar de ele não facilitar, se o estudante souber o "feijão com arroz", para ele está bom (ex. variável, escopo de variável, etc). no caso da Computação, tem que exigir um pouco mais, porque faz parte da formação deles; dos outros cursos não. O professor não acha ruim se os estudantes de outros cursos forem pedir ajuda para os monitores ou estudantes da Computação, mas se o pessoal da Computação fizer o trabalho deles, o professor vai descobrir na hora, a não ser se eles souberem explicar.
- Não, inclusive o professor não faz muitas variações na forma de ensinar. A diferença é que o ritmo em ALP é um pouco mais acelerado, e os estudantes geralmente acompanham.
- Sim, tem que adaptar de acordo com o curso.

Sobre os exemplos e exercícios:

- Muitos exercícios, passa para eles, e vai andando pelo laboratório, esclarecendo dúvidas pontuais, e um ou outro exercício faz junto com todos. Mas a maior parte, ela deixa que eles façam no tempo deles.
- Usar exemplos reais, do dia a dia.
- Lista de exercícios com alguns complicados, para prender a atenção dos que já sabem.
- Deixar que os estudantes façam as atividades, já que eles não reagem muito, daí nestes momentos é que emergem as perguntas.
- Listas de exercícios individuais.
- Fazer exercícios com os estudantes, mas dar muitos exercícios para eles começarem do zero. Eles tem de sentir a necessidade de gerar ou decidir usar uma variável.
- Elaborar níveis de exercícios, propostos no dia ou para entregar em alguns dias (não muitos), os primeiros a resolver recebem notas de participação. Os estudantes que estão mais avançados, não fica ali perdendo tempo.
- Usar desafios da maratona de programação, que fazem bastante a ligação entre a teoria e a prática, com problemas bem hipotéticos e outros bem reais.
- Em cada início de aula, o professor passa as tarefas, e faz a ligação da teoria com a prática. Quando faltam 5 minutos

finais, ele resolve e disponibiliza a solução online.

- O professor sugere melhorar a MDI para que fique claro ao estudante de que há intenção de apresentar esta mesma forma, tanto na pseudolinguagem quanto na linguagem C. Na proposta pseudo+C, pode causar algum tipo de confusão para o estudante; poderia se destacar de alguma forma, por exemplo, com cores, para diferenciar em que exemplo está estudando, se em pseudo ou se em C.
- Sugere mais materiais complementares de ponteiros (é muito chato em C), vetores, matrizes, modularização (que é bem complicado na engenharia), Mais que isso, "já está quase substituindo o professor", mas isso também é bom. Mesmo materiais bons em C param nestes tópicos. Vetores tem muitos problemas, mas nem sempre é bom ver muitos exemplos na Internet, porque ele acaba copiando e não aprende.
- Em alguns momentos, como no início sobre dados, variáveis e operadores, são muito parecidos, devido a possibilidade de respostas únicas. Depois, em matrizes e vetores, tem exercícios que eles preenchem as matrizes com os resultados dos exercícios. No mais, os exercícios são de criação de algoritmos. Ainda assim, os exercícios neste formato também são importantes.
- Acha bastante interessante os exercícios em termos de respostas objetivas, nos Estados Unidos tem bastante. Exercícios tem que ter dinâmica, se em 6 segundos o estudante não tiver a resposta, ele passa para a próxima. São exemplos os cursos que usam questionários tipo Quiz.

Sobre os materiais complementares e links de apoio:

- Dependendo do conteúdo, aprofunda mais e busca materiais complementares.
- Indicar muitos links de apoio.
- Sim, mas não é muito frequente. Ex.: URI. Também passar problemas inspirados em aplicações reais. Ex.: detecção de colisão em jogos, cálculos de geometria, cálculo de potência elétrica. Quem está realmente aprendendo, adora isto. Ou seja, aplicações simples, mas aplicações do mundo real.
- Muitos. Ex.: strings (que é bem difícil em C). Prefere sugerir links bem direcionados ao tópico que está sendo ensinado. Indica apostila da UFMG.
- Gosta de usar o maior número de recursos possíveis, e faz com que os estudantes procurem. Quando os estudantes acham algo interessante, também indica aos outros. Usa vídeos de apoio. As vezes é uma forma diferente, um momento diferente, ou porque desenhou de forma diferente no quadro.
- Exercícios de interpretar o que o programa faz são muito bons. O professor citou exemplo que outro professor usou em uma prova, e foi um massacre. Ele particularmente acha que o estudante que tem que construir, mas acha bem interessante que estejam em um minicurso, pois ele pode focar em sala os exercícios de construção do programa. Poderia indicar aqueles básicos, como Fibonacci, fatorial, número primo, e depois começar a fazer variações. Se o estudante só decorou, ele não vai saber fazer as variações. Outro ex.: cálculo dos números perfeitos (a soma dos seus divisores). Lá pelo 5o número perfeito esta lógica já não funciona mais. Alguns encontram o teorema na Internet, mas se ele não entende que tem que usar o "long", que é uma questão básica de matemática. O professor procurava pegar coisas aplicadas, Ex.: usar "guia 4 rodas" (quando não tinha ainda GPS) para calcular um percurso usando matrizes, os estudantes acham muito fácil, mas funciona para o aprendizado.
- Não. Ele já usa a apostila da UFMG, e mais atualmente uma de Uberlândia, que é mais bem elaborada (está na página do professor), e por isso não acha necessário ficar fazendo indicações. Qualquer busca que o estudante fizer para qualquer conceito da disciplina, o estudante encontrará diversos exemplos e exercícios. Para não dizer que não indica nada, ele passa o link do Code Blocks par que eles possam treinar em casa. Ele não usa o VisuAlg, porque acredita que está introduzindo outra dificuldade para o estudante. Se for para empenhar um tempo neste tipo de dificuldade, que seja direto em C.
- Nas 2 primeiras aulas, tem vários exercícios, ex. travessia, para fazer em sala, ele sugere outros para fazer em casa. Um dos que o professor sugere, nestes casos, é tipo o Racha-Cuca, as vezes como exemplo, para eles experimentarem. Para ALP, em alguns momentos do curso, ele obriga a usar o URI, e dá pontos por isso. Principalmente porque o URI permite que se cadastre a sua turma e acompanhe a participação. Também sugere links para guias da pseudolinguagem e da linguagem C. Mas os estudantes de hoje estão antenados, e muitas vezes os próprios estudantes sugerem links e compartilham com a turma. Sempre que pode, sugere outros links.
- Não. Tudo o que ele precisa está no Git Hub.

Sobre o minicurso como apoio à disciplina presencial:

- Sim, é sempre bom. Quanto mais disciplinas tiver, mais de uma ferramenta é capaz de ajudar (porque os estudantes aprendem de maneira diferente). O único problema que o professor vê é a questão da motivação - convencer este estudante a tentar de outra forma não é uma tarefa trivial. No momento da pseudolinguagem, para os estudantes que são bons, isso se torna rapidamente desmotivador (pois eles querem ir além); e o contrário, na linguagem C, os estudantes que são fracos acabam achando difícil demais. Por isso o professor optou por fazer direto na linguagem C, para manter um nível de dificuldade constante. Se der ênfase na pseudolinguagem e deixar o C para o final, vai acontecer de ter grandes teóricos com pouca habilidade prática. Se der pouca ênfase na pseudolinguagem e muita ênfase na linguagem C, terá estudantes com muita dificuldade no desenvolver porque tiveram pouca teoria.
- Com certeza, pois ele tem o foco nesta parte que é a base, e o estudante tem aonde pegar conteúdo (teoria e exercício), a professora dá exercício em sala, mas as vezes não chega a corrigir, ou dá o gabarito e o estudante não olha. No online ele vai fazendo e vendo o resultado, é uma ferramenta essencial. Se neste semestre tivesse sido jogada ao mesmo tempo em que foi dada a disciplina, isso teria sido um ponto chave. Ele veio um pouco atrasado, mas foi bom porque os estudantes que fizeram disseram que estava "bem fácil", e isso para o professor é bom porque ele realmente deveria saber aquilo bem.
- Sim. A professora gosta de usar o máximo de recursos possíveis, e gosta que eles usem. É uma forma diferente de verem as mesmas coisas, terem outros exercícios, não ficar só nos da sala de aula. Outra coisa interessante por ser curso online é a possibilidade de se usar gifs animados para explicar as estruturas. Evitar usar slides. Outra, sugere colocar, por ex., uma estrutura sequencial e depois simplificar colocando uma repetição no programa.
- Sim, pois alguns participam mais (aqueles que não falam muito, não se sentiam a vontade), e o ranking deu muito incentivo para estes estudantes, que se mostraram bastante competitivos, eles queriam subir no ranking. O importante é o estudante estar motivado, trabalhando. Ela acha que a partir do momento que tiver mais conteúdos, vai ficar melhor ainda. No final, é difícil elencar muitos problemas interessantes por tópico, com essa sensibilidade de o estudante não ficar interpretando muito o conteúdo do problema sem se perder. As vezes,

montar um conjunto de exercícios interessantes para cada tópico demanda um pouco, e se tiver mais exercícios no ambiente, melhor.
- É um apoio interessante. Mas o professor fica frustrado de os estudantes não procurarem muito a monitoria. Um recurso a mais que tenha para o estudante poder entender, auxiliar, ele acha válido. Ele nunca parou para pensar se "seria o ideal que tivesse um minicurso", mas qualquer ajuda para o aprendizado do estudante é bem vinda e ele não se opõe em hipótese alguma. Falando da participação do minicurso atual, ele não acompanhou a turma dele. Mas na turma do semestre anterior só tiveram 1 ou 2. Neste semestre, dos 21 que se matricularam, 10 já desistiram na 1a prova, só uns 8 vieram fazer a última prova, mas são vários fatores, talvez por pré-requisitos (essa disciplina não tem), entre outros.
- Sem dúvida, todo recurso adicional é sempre bem vindo.
- Sim, claro. O professor pode disponibilizar aos estudantes, poderia ocupar uma das aulas da semana inicialmente para fazer os estudantes verem o que eles podem usar. Poderia fazer uma aula por semana, para ensinar o conteúdo do professor com C e o conteúdo na pseudolinguagem que ele está aprendendo. Quando o estudante vier com a pergunta para o professor, ele pode confirmar.

Sobre o minicurso estar no AdaptWeb:

- Usar o AdaptWeb® como ponto extra.
- A possibilidade de ter mais exercícios, e poder navegar pelo curso no ritmo dele, é bem interessante.
- Em paralelo com as aulas, o professor não "perderia tempo", explicaria mas não faria os exercícios, a prática da análise fica no AdaptWeb® e a prática de programação no curso presencial.
- (Q4) Professores têm pouco ou nenhum contato com o AdaptWeb®.
- É uma ferramenta interessante, diferente de sala de aula, que o professor passa o conteúdo de um livro, um material, mas no minicurso ele tem um conteúdo inteiro, perde menos tempo para consultar.
- Não é necessário gastar o tempo do minicurso com outros conteúdos informativos, é melhor se aprofundar mais coisas de programação, o ganho é melhor. O recurso tem mais coisas para explorar por ser online, e a vantagem é que se o estudante não se sair muito bem no minicurso, ele poder vir para o professor com mais dúvidas de programação.
- Não é necessário gastar o tempo do minicurso com outros conteúdos informativos, é melhor se aprofundar mais coisas de programação, o ganho é melhor. O recurso tem mais coisas para explorar por ser online, e a vantagem é que se ele não se sair muito bem no minicurso, é ele poder vir para o professor com mais dúvidas de programação.
- O minicurso online é perfeito para complementar o que se faz em sala de aula.

Sobre a aplicação do minicurso em outros contextos:

- Ferramenta interessante para aquele estudante que está precisando de recuperação. Também para formação de professores (como exemplo essa discussão sobre os conteúdos do minicurso entre professores).
- O conteúdo da parte básica, por exemplo poderia até substituir os slides da professora, poderia desenvolver a aula em cima daquele conteúdo, poderia ser usado como guia durante a aula. Pode ser no futuro, acrescentando mais exemplos, mais conteúdo, como apoio de aula presencial.
- Em paralelo com as aulas, o professor não "perderia tempo" , explicaria mas não faria os exercícios, a prática da análise fica no AdaptWeb® e a prática de programação no curso presencial.
- Sim, por exemplo, para quem está tentando programar sozinho, fora da faculdade, mas tem que ver a questão do suporte. Quando se aprende sozinho, procura livros ou ambientes, mas não é só isso. As dúvidas maiores acabam procurando em fóruns. Como apoio também é muito bom, eles fazem um programa e esquecem como é. Tem que revisar, refazer para fixar.
- Talvez abrir para quem ainda não está matriculado na disciplina, ou que queira começar a tomar contato com isso, para ir se preparando para quando vier a disciplina.
- Aplicar nos minicursos da OBI, e em alguns casos, ser o minicurso da OBI. Seja como complemento às aulas presenciais, seja como o curso em si, para estudantes do Ensino Médio.
- Como apoio à disciplina sim, é atemporal. Poderia deixar como algo externo à comunidade, em um programa de extensão à comunidade, às escolas do Ensino Médio.
- Antes do semestre: talvez não, pode até ser "assustador" para um estudante que nunca teve contato com isso. Pode ser bom para aqueles estudantes autodidatas (para quem, aliás, qualquer método é bom).
- Antes do semestre: A parte básica, se viesse antes da disciplina, em um curso de férias, o professor já entraria direto na programação, até mais pesada, não perderia tempo, avançaria como uma disciplina mais aplicada.
- Antes do semestre: Como pré-requisito para a aula (sonho de todo professor). Interessante antes, como se fosse um "pré-algoritmos". Daria para ir mais rápido em algumas coisas iniciais e focar em problemas mais complexos durante o semestre. A possibilidade de lógica de programação que se poderia colocar seria muito maior. Eles viriam com a dificuldade formulada, também.
- Antes do semestre: Talvez não, pois o tempo entre o período de matrícula e o início é muito curto. Mas quando o minicurso estiver mais estabelecido, mais maduro, divulgar para quem quiser tentar se adiantar um pouco, talvez seja interessante.
- Antes do semestre: Poderia ser, talvez no momento que já fazem a matrícula. Todas as formas são interessantes.
- Após o semestre: Para o estudante que tem dificuldade, é importante ter diferentes métodos. Para recuperação de estudantes pode ser interessante a partir do segundo mês de aula, após a primeira prova. É interessante o minicurso desde que o estudante possa escolher o método (texto, vídeo, áudio), e o online permite isso.
- Após o semestre: Seria bem interessante também, principalmente para os que reprovam.
- Após o semestre: Não, acha apenas interesante ser aplicado antes do curso.
- Após o semestre: Alguns repetentes tem problema de conteúdo (ou de atitude), mas tudo que oferece abordagem de aprendizado alternativa é válido. Alguns estudantes são contra gastar tempo com linguagem de programação, não que seja errado, mas acabamos perdendo alguns estudantes por isso. O que é alternativo à forma tradicional, pode estimular ou ajudar em alguma deficiência, é válido, mas depende da vontade dele também.
- Após o semestre: Se você chega no final, quando ele já reprovou, e você mostra o minicurso, o estudante pode pensar: "por que não me mostraram isso antes?", este enfoque tem que ser mostrado com cuidado. Talvez mostrar antes como opção para ele saber que existe, e aí sim reapresentar se ele reprovar.
- Após o semestre: Ótima ideia. Todos os casos, ela acha positivo. Até neste caso, sugerir que o estudante faça quando ele não teve êxito, é talvez mais motivador que sugerir antes do início do semestre.

Sobre o tempo de início e duração:

- Não precisava ser tão no começo, mas não tem prejuízo nenhum começar nesta data.
- Não vê problema se o estudante avançar mais no minicurso que na aula presencial. Isso até facilita o trabalho dele.
- Possivelmente, próximo do prazo final, pedirá para estender um pouco mais o acesso ao minicurso, mas somente para permitir que os estudantes continuem acessando os conteúdos.
- Única coisa que vai estar em descompasso é aquela semana inicial sobre os componentes do computador (obs.: este professor não gosta de sair da programação nem permitir que os estudantes vejam conteúdos antes da hora...).
- A opção de navegação livre ou tutorial é boa, pois nem todos gostam do tutorial.
- Sim. O professor não gasta muito tempo na parte de narrativa. É 1 aula de boas vindas, introdução à lógica, e já vai para narrativas, estrutura de programa, até mostra um fluxograma para exemplificar, mas é menos tempo que isso.
- Tudo soma, mas tem que tomar cuidado se o pré-requisito da turma é a linguagem C. Tem que dar a ferramenta de tal forma que o estudante possa fazer todos os saltos que ele quiser.
- Os estudantes não têm muita gana de ficar sentado em um ambiente online "autodidata", principalmente de tiver muito texto. Se tiver a oportunidade de chegar em um ambiente e rapidamente começar a desenvolver soluções, será diferente. O ambiente, a ferramenta, deve ser um mecanismo para acelerar o desenvolvimento, e o texto retarda. O professor passa problemas pequenos no começo do semestre, em seguida já passa problemas maiores, e no final do semestre já está mostrando problemas com alta complexidade. É a "bola de neve". Isso se aplica independente da turma. A complexidade crescente exige o raciocínio abstrato.
- Os estudantes não vão disponibilizar muito devido a outras prioridades de estudo.
- Se eles dedicarem 1 hora por semana, já vai ser lucro. A não ser que tenham algum tipo de incentivo. O que ele recomenda é que eles "percam" um meia hora por semana, senão não vai dar certo. O professor sempre informa na aula anterior o que vai ensinar na próxima aula, para que o estudante tenham a chance de ir atrás das informações por conta.
- Depois de começar a ir ao laboratório, pensar em uns 15 minutos finais parar antes da aula e tentar direcionar o uso do minicurso. O ideal é que o minicurso fosse tutoreado.

Sobre a avaliação final de aprendizagem:

- Exercícios altamente focados em memorização de comandos são desnecessários (as próprias ferramentas já oferecem isso). O estudante tem de entender a diferença entre a programação e a codificação. Eles têm de saber "programar", ou seja, dar problemas e o estudante ser capaz de confeccionar uma sequência sistemática de passos que resolvem aquele problema. Aí sim, vem a codificação, em qualquer linguagem. Sugere exercícios para interpretar algoritmos (propor entradas e o estudante dizer o que era na saída).
- O curso é bastante válido para a análise. O estudante pode não aprender a programar, mas vai aprender a analisar. Essa parte de análise na sala de aula é muito mais cansativa - o curso online ajuda bastante nisso. Fazer a análise de algoritmo ajuda bastante depois a programar.
- Se o exercício é muito complexo, e fizer por alternativas, fica complicado avaliar. Ele não tem certeza sobre isso. Talvez a compreensão dos recursos didáticos no meio "informático", seja o "pulo do gato". Para ele, há a convicção de que ninguém conseguiu ainda criar um framework de aprendizado usando recursos de computação ou Internet. Não é apenas usar uma lousa digital, ou escrever no computador ao invés de escrever a mão. "O paradigma não foi estabelecido ainda", talvez nem vá, ou promover uma imersão no assunto.
- O ideal seria escrever um algoritmo e avaliar completamente. Mas entende o objetivo para o curso online, e neste sentido está perfeito. Quem vai fazer este outro julgamento é o professor presencial, que pode subsidiar o quanto “interferiu" na aula dele, até a opinião do professor, o quanto ele gostou ou não. Algum questionamento que os estudantes tiveram em minicurso anterior foi quanto ao tempo disponível para realização, mas foi só isso.

APÊNDICE F **Matriz de Design Instrucional (MDI)**

MDI completa, em sua representação com todos os campos necessários para orientar a inclusão dos objetos no ambiente de aprendizagem.

Unidade Instrucional	Conceito	Nº	Sub-conceito	Pré-requisitos	Exemplos	Exercícios	Materiais complementares	Links de apoio	Tempo (minutos)
1. Introdução à lógica, algoritmos, pseudolinguagem.	01. Boas Vindas	1	Boas Vindas	Conhecimento de informática como usuário, acesso à Internet e redes sociais.					6
		1.1	Objetivo do minicurso	1			Materiais 01.1 Vídeo - S Materiais 01.2 Créditos		6
		1.2	Ferramentas de aprendizagem	1.1					6
		1.3	O que você vai ver neste curso?	1.2					6
		1.4	Como estudar	1.3					6
	01. Boas Vindas Total								30
	02. Da lógica à programação	2	Da lógica à programação	1.4					10
		2.1	Por que aprender a programar?	2			Materiais 02.1 Papo BJ Materiais 02.2 Jogo Lig		10
		2.2	Aplicações de Algoritmos	2.1					5
		2.3	Algoritmos e Programação	2.2					5
	02. Da lógica à programação Total								30
	03. Interpretadores e compiladores	3	Interpretadores e compiladores	2.3					6
		3.1	Interpretadores de algoritmos (Portugol IDE)	3				Link 03.12 Microsoft Link 03.13 Code::Blo Link 03.14 Codepad.c Link 03.15 Ideone.co Link 03.16 TutorialsP Link 03.17 CppDroid	8
		3.2	Compiladores de linguagem (DevC++)	3.1					8
		3.3	Outras ferramentas de aprendizagem	3.2					8
	03. Interpretadores e compiladores Total								30
	04. Algoritmos	4	Algoritmos	3.3			Materiais 03.3.1 Outros Materiais 03.3.2 Outros		10
		4.1	Criando um algoritmo	4		04.1 04.2 04.3 04.4 04.5 04.6			25
		4.2	Tipos de algoritmo	4.1			Materiais 04.1.1 Passo		25
	04. Algoritmos Total								60
	05. Sobre narrativas	5	Sobre narrativas	4.2					5
		5.1	Tipos de narrativas	5					5
		5.2	Exercitando as narrativas	5.1	Exemplos 05.1.1 Faz Exemplos 05.1.2 Tom Exemplos 05.1.3 Troc	05.1	Materiais 05.1.1 Narrat Materiais 05.1.2 Narrat Materiais 05.1.3 Narrat		50
	05. Sobre narrativas Total								60
	06. Sobre pseudocódigos	6	Sobre pseudocódigos	5.2	Exemplos 05.2.1 Som Exemplos 05.2.2 Divi Exemplos 05.2.3 Divi				10
		6.1	Estrutura de um pseudocódigo	6					25
		6.2	Predefinições para escrita de pseudocódigos	6.1			Materiais 06.1.1 Video		25
	06. Sobre pseudocódigos Total								60
2. Estrutura de um programa. Dados, variáveis e operadores.	07. O que é um programa?	7	O que é um programa?	6.2					30
		7.1	Linguagem C - Principais conceitos	7		07.1 07.2 07.3 07.4			30
		7.2	Estrutura de um programa	7.1			Materiais 07.1.1 Pseud Materiais 07.1.2 Alo m Materiais 07.1.3 Pseud Materiais 07.1.4 Teste		5
		7.2.1	main()	7.2					5
		7.2.2	system()	7.2.1					5
		7.2.3	#include	7.2.2					5
		7.3	Constantes e comandos de atribuição	7.2.3					40
		7.4	Comandos de Entrada-Saída	7.3	Exemplos 07.3.1 Atri Exemplos 07.3.2 Pse Exemplos 07.3.3 Pse Exemplos 07.3.4 Con Exemplos 07.3.5 Car Exemplos 07.3.6 Cad Exemplos 07.3.7 Pse Exemplos 07.3.8 Con		Materiais 07.3.1 Pseud		20
		7.4.1	printf	7.4	Exemplos 07.4.1 Alo Exemplos 07.4.2 Pse Exemplos 07.4.3 Entr Exemplos 07.4.4 Troc		Materiais 07.4.1 Pseud Materiais 07.4.2 Tabela Materiais 07.4.3 Tabela		20
		7.4.2	scanf	7.4.1					20
		7.5	Strings	7.4.2			Materiais 07.4.2.1 Pseu Materiais 07.4.2.2 Func Materiais 07.4.2.3 Pseu Materiais 07.4.2.4 Entra		60
	07. O que é um programa? Total								240
	08. Dados, variáveis e operadores	8	Dados, variáveis e operadores	7.5	Exemplos 07.5.1 Fun Exemplos 07.5.2 Fun Exemplos 07.5.3 Fun Exemplos 07.5.4 Fun Exemplos 07.5.5 Fun Exemplos 07.5.6 Fun		Materiais 07.5.1 Tabela		5
		8.1	Dados	8	Exemplos 08.1.1 Vari	08.1 08.10 08.11 08.12 08.13 08.14 08.15 08.2 08.3 08.4 08.5 08.6 08.7 08.8 08.9			10

MDI resumida, representando os níveis e sub-níveis de cada conceito, bem como o tempo estimado necessário para o cumprimento do estudo de cada conceito.

Nº	Conceito	Pré-requisitos	Tempo (min.)
1	Boas Vindas	-	6
1.1	Objetivo do minicurso	1	6
1.2	Ferramentas de aprendizagem	1.1	6
1.3	O que você vai ver neste curso?	1.2	6
1.4	Como estudar	1.3	6
2	Da lógica à programação	1.4	10
2.1	Por que aprender a programar?	2	10
2.2	Aplicações de Algoritmos	2.1	5
2.3	Algoritmos e Programação	2.2	5
3	Interpretadores e compiladores	2.3	6
3.1	Interpretadores de algoritmos (Portugol IDE)	3	8
3.2	Compiladores de linguagem (DevC++)	3.1	8
3.3	Outras ferramentas de aprendizagem	3.2	8
4	Algoritmos	3.3	10
4.1	Criando um algoritmo	4	25
4.2	Tipos de algoritmo	4.1	25
5	Sobre narrativas	4.2	5
5.1	Tipos de narrativas	5	5
5.2	Exercitando as narrativas	5.1	50
6	Sobre pseudocódigos	5.2	10
6.1	Estrutura de um pseudocódigo	6	25
6.2	Predefinições para escrita de pseudocódigos	6.1	25
7	O que é um programa?	6.2	30
7.1	Linguagem C - Principais conceitos	7	30
7.2	Estrutura de um programa	7.1	5
7.3	Constantes e comandos de atribuição	7.2.3	40
7.4	Comandos de Entrada-Saída	7.3	20
7.5	Strings	7.4.2	60
8	Dados, variáveis e operadores	7.5	5
8.1	Dados	8	10
8.2	Operadores básicos	8.1	10
8.3	Operadores aritméticos em C	8.2	10
8.4	Operadores relacionais	8.3	10
8.5	Operadores lógicos	8.4	10
8.6	Precedência entre todos os Operadores	8.5	10
9	Estrutura de Controle - Decisão	8.6	20
9.1	Estrutura de decisão simples	9	20
9.2	Estrutura de decisão composta	9.1	20
9.3	Estrutura de decisão encadeada	9.2	60
10	Estrutura de controle – Laços (repetição)	9.3	60
10.1	Laço while (enquanto)	10	60
10.2	Laço do...while (repita)	10.1	0
10.3	Laço for (para)	10.2	0
11	Vetores e matrizes	10.3	0
11.1	Matrizes de uma dimensão (Vetores)	11	120
11.2	Matrizes	11.1	120

MDI resumida, apresentando os exemplos, exercícios, links de apoio e materiais complementares.

Nº	Conceito	Conteúdos	Exemplos	Exercícios	Materiais	Links
1	Boas Vindas	(01) Seja Bem-Vindo	(54) Exemplos - Fazer sanduiche	(108) Exercício - Identificar etapas algoritmo opção 1	(147) Materiais 01.1 Video - Seja bem-vindo	(178) 03.2.1 Links - Aplicativo - DevCpp
1.1	Objetivo do minicurso	(02) Objetivo do minicurso	(55) Exemplos - Tomar banho	(109) Exercício - Receita de bolo	(148) Materiais 01.2 Creditos	(179) 03.2.2 Links - Download DevCpp 5.4.0
1.2	Ferramentas de aprendizagem	(03) Ferramentas de aprendizagem	(56) Exemplos - Trocar lâmpada	(110) Exercício - Ordenar sequencia algoritmo	(149) Materiais 02.1 Papo BJPnet	(180) 03.3.1 Links - Aplicativo - G-Portugol
1.3	O que você vai ver neste curso?	(04) O que voce vai ver neste curso	(57) Exemplos - Soma de dois números	(111) Exercício - Identificar etapas algoritmo opção 2	(150) Materiais 02.2 Jogo Light-bot	(181) 03.3.2 Links - Aplicativo – MACP
1.4	Como estudar	(05) Como estudar	(58) Exemplos - Divisao de dois números	(112) Exercício (Narrativa de seleção) Divisão de 2 números	(151) Materiais 03.3.1 Outros interpretadores de pseudolinguagem	(182) 03.3.3 Links - Aplicativo – VisuAlg
2	Da lógica à programação	(06) Por que aprender a programar	(59) Exemplos - Divisão de dois números se possível	(113) Exercício - Somar 2 números	(152) Materiais 03.3.2 Outros compiladores de linguagem C	(183) 03.3.4 Links - Aplicativo - Portugol online
2.1	Por que aprender a programar?	(07) Aplicacoes de Algoritmos	(60) Exemplos - Atribuicoes de variáveis	(114) Exercício - Somar conteúdos retângulos opção 1	(153) Materiais 04.1.1 Passos para criar algoritmo	(184) 03.3.5 Links - Aplicativo – Webportugol
2.2	Aplicações de Algoritmos	(08) Aplicacoes de Algoritmos	(61) Exemplos - Pseudocodigo - Atribuicoes de variáveis	(115) Exercício - Somar 3 números	(154) Materiais 05.1.1 Narrativa sequencial	(185) 03.3.6 Links - Aplicativo – AMBAP
2.3	Algoritmos e Programação	(09) Interpretadores e compiladores	(62) Exemplos - Pseudocodigo - Aplicacao de atribuicoes usando variaveis	(116) Exercício - Somar conteúdos retângulos opção 2	(155) Materiais 05.1.2 Narrativa de selecao	(186) 03.3.7 Links - Aplicativo - Microsoft Visual Cpp
3	Interpretadores e compiladores	(10) Interpretadores de algoritmos (Portugol IDE)	(63) Exemplos - Constantes numéricos	(117) Exercício - Criação de algoritmos	(156) Materiais 05.1.3 Narrativa de repetição	(187) 03.3.8 Links - Aplicativo – CppDroid
3.1	Interpretadores de algoritmos (Portugol IDE)	(11) Compiladores de linguagem (DevCpp)	(64) Exemplos - Caracteres constantes	(118) Exercício - Pseudocódigo - Teste dos códigos especiais	(157) Materiais 06.1.1 Animacao - estrutura do pseudocódigo	(188) 03.3.9 Links - Aplicativo – CodeBlocks
3.2	Compiladores de linguagem (DevC++)	(12) Outras ferramentas de aprendizagem	(65) Exemplos - Cadeia de caracteres constantes	(119) Exercício - Teste dos códigos especiais	(158) Materiais 07.1.1 Animacao - Comparativo estrutura pseudo e C	(189) 03.3.10 Links - Pagina - codepad.org
3.3	Outras ferramentas de aprendizagem	(13) Algoritmos	(66) Exemplos - Pseudocodigo - Aplicacao de caracteres especiais	(120) Exercicio - String	(159) Materiais 07.1.2 Alo mundo	(190) 03.3.11 Links - Pagina - ideone.com
4	Algoritmos	(14) Criando um algoritmo	(67) Exemplos - Constantes no C	(121) Exercício – programa	(160) Materiais 07.1.3 Pseudocodigo - Alo mundo	(191) 03.3.12 Links - Pagina - tutorialspoint.com
4.1	Criando um algoritmo	(15) Tipos de	(68) Exemplos -	(122) Exercício -	(161) Materiais	(192) Avaliação –

Nº	Conceito	Conteúdos	Exemplos	Exercícios	Materiais	Links
		algoritmo	Alo mundo	Avalie Alo mundo opção 1	07.1.4 Teste de mesa	QUESTÃO 01 (estrutura do programa)
4.2	Tipos de algoritmo	(16) Sobre narrativas	(69) Exemplos - Pseudocodigo - Alo mundo	(123) Exercício - Avalie Alo mundo opção 2	(162) Materiais 07.3.1 Pseudocodigo - Comando de atribuição	(193) Avaliação – QUESTÃO 02 (condição)
5	Sobre narrativas	(17) Tipos de narrativas	(70) Exemplos - Entrar com inteiro e mostrar	(124) Exercício – Saída	(163) Materiais 07.4.1 Pseudo x C - Comandos de saída	(194) Avaliação – QUESTÃO 03 (algoritmo/narrativa)
5.1	Tipos de narrativas	(18) Exercitando as narrativas	(71) Exemplos - Trocar 2 numeros	(125) Exercício - Tipos de operadores aritméticos	(164) Materiais 07.4.2 Tabela de códigos especiais	(195) Avaliação – QUESTÃO 04 (compreensão do código)
5.2	Exercitando as narrativas	(19) Sobre pseudocódigos	(72) Exemplos - Funcoes string – imprimir	(126) Exercício - Dados do tipo inteiro	(165) Materiais 07.4.2.1 Pseudocodigo - Funcao de entrada	(196) Avaliação – QUESTÃO 05 (estruturas de controle)
6	Sobre pseudocódigos	(20) Estrutura de um pseudocódigo	(73) Exemplos - Funcoes string – copiar opção 1	(127) Exercício - Operadores - Soma de dois números	(166) Materiais 07.4.2.2 Funcao de entrada	(197) Avaliação – QUESTÃO 06 (repetição)
6.1	Estrutura de um pseudocódigo	(21) Predefinicoes para escrita de pseudocódigos	(74) Exemplos - Funcoes string – copiar opção 2	(128) Exercício - (aritméticos) área do retângulo	(168) Materiais 07.4.2.3 Pseudocodigo - Entrada de dados	(198) Avaliação – QUESTÃO 07 (dados, variáveis, operadores)
6.2	Predefinições para escrita de pseudocódigos	(22) Programa	(75) Exemplos - Funcoes string – comprimento	(129) Exercício - (aritméticos) área do triangulo	(169) Materiais 07.4.2.4 Entrada de dados	(199) Avaliação – QUESTÃO 08 (dados, variáveis, operadores)
7	O que é um programa?	(23) Estrutura de um programa	(76) Exemplos 07.5.4 Funcoes string – reverter	(130) Exercício - (aritméticos) volume do cilindro	(170) Materiais 07.4.3 Tabela de codigos de formatação	(200) Avaliação – QUESTÃO 09 (repetição)
7.1	Linguagem C - Principais conceitos	(24) Linguagem C - Principais conceitos	(77) Exemplos - Funcoes string – concatenar	(131) Exercício - (aritméticos) volume do cubo	(171) Materiais 07.5.1 Tabela das funcoes de string	(201) Avaliação – QUESTÃO 10 (matrizes)
7.2	Estrutura de um programa	(25) main()	(78) Exemplos - Funcoes string – comparar	(132) Exercício - Operadores - condição existência triângulo	(172) Materiais 08.1.1 Pseudocodigo - Atribuicoes com variáveis	
7.3	Constantes e comandos de atribuição	(26) system()	(79) Exemplos - Variáveis no C	(133) Exercício - salário bruto-líquido	(173) Materiais 08.1.2 Atribuicoes com variáveis	
7.4	Comandos de Entrada-Saída	(27) #include	(80) Exemplos - Quatro operações básicas	(134) Exercício - Torre de Hanói	(174) Materiais 08.1.3 Tabela - Pseudo x C - Tipos de dados	
7.5	Strings	(28) Constantes e comandos de atribuição	(81) Exemplos - Pseudocódigo - Área do circulo	(135) Exercício - Condição - nota maior igual a 7	(175) Materiais 08.4.1 Tabela - Operadores relacionais	
8	Dados, variáveis e operadores	(29) Comandos de Entrada-Saida	(82) Exemplos - Área do circulo	(136) Exercício - numero par impar pos neg nulo	(176) Materiais 11.1.1 Declaracao de vetores	
8.1	Dados	(30) printf	(83) Exemplos - Decisao simples	(137) Exercício - idade_atleta x categoria	(177) Materiais 11.2.1 Declaracao de	

Nº	Conceito	Conteúdos	Exemplos	Exercícios	Materiais	Links
					matrizes	
8.2	Operadores básicos	(31) scanf	(84) Exemplos - Pseudocódigo - Divisão de dois números	(138) Exercício - número é par ou impar		
8.3	Operadores aritméticos em C	(32) Strings	(85) Exemplos - Maior de três números	(139) Exercício - while e do-while		
8.4	Operadores relacionais	(33) Dados Variaveis e Operadores	(86) Exemplos - Par ou impar	(140) Exercício - Mesa de testes 1		
8.5	Operadores lógicos	(34) Dados	(87) Exemplos - Ano bissexto	(141) Exercício - Mesa de testes 2		
8.6	Precedência entre todos os Operadores	(35) Operadores básicos	(88) Exemplos - Média de duas notas	(142) Exercício - mostra pares de 2 a 20		
9	Estrutura de Controle - Decisão	(36) Operadores aritmeticos em C	(89) Exemplos - Pseudocódigo - (SE) numero maior que 0 ou não	(143) Exercício - Matrizes - Compara palavras		
9.1	Estrutura de decisão simples	(37) Operadores relacionais	(90) Exemplos - Pseudocódigo - (SE composta) numero maior que 0 ou não	(144) Exercício - Matrizes - Entrada valores		
9.2	Estrutura de decisão composta	(38) Operadores lógicos	(91) Exemplos - Pseudocódigo - (SE composta) numero maior menor ou igual a zero	(145) Exercicio - Matrizes - Palavras em ordem inversa		
9.3	Estrutura de decisão encadeada	(39) Precedência entre todos os Operadores	(92) Exemplos - Pseudocódigo - 4 operações	(146) Exercício - Vetores - Inverte ordem		
10	Estrutura de controle – Laços (repetição)	(40) Estrutura de Controle – Decisão	(93) Exemplos - Pseudocódigo - Ano bissexto			
10.1	Laço while (enquanto)	(41) Estrutura de decisão simples	(94) Exemplos - Pseudocódigo - 4 operações			
10.2	Laço do...while (repita)	(42) Estrutura de decisão composta opção 1	(95) Exemplos - Checar vogal			
10.3	Laço for (para)	(43) Estrutura de decisão composta opção 2	(96) Exemplos - Pseudocódigo - x menor que 3			
11	Vetores e matrizes	(44) Estrutura de decisão encadeada opção 1	(97) Exemplos - Somar dígitos			
11.1	Matrizes de uma dimensão (Vetores)	(45) Estrutura de decisão encadeada opção 2	(98) Exemplos - Quadrado dos números de 0 a 20			
11.2	Matrizes	(46) Estrutura de controle – Laços	(99) Exemplos - Pseudocódigo - Quadrado dos números 0 a 20			
		(47) Laco while (enquanto)	(100) Exemplos – Pseudocódigo - x menor que 3			

Nº	Conceito	Conteúdos	Exemplos	Exercícios	Materiais	Links
		(48) Laco do...while (repita)	(101) Exemplos - Pseudocódigo - Mostra x de 0 a 2			
		(49) Laco para (for)	(102) Exemplos - x de 0 a 3			
		(50) Laco para (for)	(103) Exemplos - Pseudocódigo - x de 0 a 3			
		(51) Vetores e matrizes	(104) Exemplos - Matemática - Conjuntos numéricos			
		(52) Vetores	(105) Exemplos - Vetores - Entrada de dados			
		(53) Matrizes	(106) Exemplos - Vetores - Ordenação por seleção			
			(107) Exemplos - Matrizes - Entrada de dados			

APÊNDICE G **Objetos de Aprendizagem**

Os Objetos de Aprendizagem foram inicialmente escritos em modo texto, durante a etapa de Desenvolvimento, e posteriormente adaptados para o formato web, conforme descrito na seção 5.3. Todos os objetos estão representados a seguir, cada um deles identificado, em negrito, por um número sequencial e uma breve descrição. Nas duas linhas seguintes de cada, está o Conceito e o Sub-conceito, tal qual representado na MDI, no APÊNDICE F.

CONCEITOS

(1) Seja Bem-Vindo

1 Boas Vindas
1 Seja bem-vindo ao minicurso de Algoritmos e Programação!

Este minicurso foi concebido para ensinar os fundamentos de algoritmos e de programação a iniciantes no estudo da computação.
Não é necessário ter qualquer experiência anterior na área, além do conhecimento básico como usuário(a) de computador, tablet, celular ou outro dispositivo. Também é interessante, mas não obrigatório, que você já tenha feito acessos à Internet e redes sociais.
Ah, e caso você não tenha recebido instrução inicial sobre o uso do AdaptWeb®, não se preocupe: o ambiente é bem intuitivo, e ainda tem fácil acesso à ajuda, sempre que precisar.

(2) Objetivo do minicurso

1 Boas Vindas
1.1 Objetivo do minicurso

Oferecer conteúdos, exemplos e exercícios que complementem a disciplina de introdução à programação ou algoritmos que você cursará nesta universidade.
Esperamos assim poder contribuir para que você, a partir destes conhecimentos adicionais, possa melhorar suas habilidades e manter sua motivação nos estudos desta ou de qualquer outra disciplina que envolva o raciocínio lógico para solução de problemas.

(3) Ferramentas de aprendizagem

1 Boas Vindas
1.2 Ferramentas de aprendizagem

Durante o minicurso você precisará de uma ferramenta para associar os conteúdos aprendidos (conceitos) à prática de programação.
Para tanto, você terá acesso a exemplos e exercícios, propostos aqui em forma de uma pseudolinguagem e também na linguagem C.

(4) O que voce vai ver neste curso

1 Boas Vindas
1.3 O que você vai ver neste curso?

Nos capítulos iniciais você verá conceitos de lógica, algoritmos e seus tipos, como as narrativas e os pseudocódigos; também conhecerá sobre o Portugol IDE e o DevC++, que serão suas ferramentas de apoio de aprendizagem neste minicurso:
(figura)
Os capítulos seguintes são fundamentais para a começar a entender a programação. São apoiados por exemplos e exercícios práticos e apresentam algumas das principais funções na linguagem C:
(figura)
Apoiados pelas funções do C apresentadas, você vai aprender sobre dados, variáveis e operadores:
(figura)
Nos últimos capítulos entram os conceitos de programação estruturada, com foco nas estruturas de condição e repetição, vetores e matrizes:
(figura)

(5) Como estudar

1 Boas Vindas
1.4 Como estudar

(figura)
1. Para navegar no conteúdo deste curso utilize os links disponíveis no menu localizado à sua esquerda.
2. Em paralelo a leitura dos Conceitos, você poderá consultar exemplos, responder a exercícios ou consultar materiais complementares, quando disponíveis, selecionando a Categoria no lado esquerdo superior. Também há disponíveis alguns links de apoio, sugeridos para incentivar o aprofundamento nos temas.
3. A qualquer momento, você poderá consultar os links de apoio, sugeridos durante todo o minicurso.
4. Para saber quais as categorias disponíveis para cada conceito, você pode abrir o Mapa da Disciplina (canto superior direito) a qualquer momento.
5. Saiba mais sobre os recursos do AdaptWeb®, clicando em Ajuda no canto superior direito.

(6) Por que aprender a programar

2 Da lógica à programação
2.1 Por que aprender a programar?

É desafiador. Estudar lógica de programação é quase como resolver um quebra-cabeça: você deve analisar os argumentos e reordená-los para que façam sentido. De início, as conexões podem não ser tão óbvias e, por isso, você precisará exercitar sua mente para encontrar os encaixes de modo a torná-los válidos. A tarefa não é fácil, mas o desafio é motivador!
Rapidez de raciocínio. Ao se deparar com um problema, os profissionais devem pensar rapidamente para encontrar uma solução. A pressão e o fato de não saber o que deu errado podem complicar a resolução desta situação, mas se você pratica com frequência a análise de fatos aparentemente isolados e suas possíveis relações, como propõe a lógica, você conseguirá mais velocidade nas questões relativas ao trabalho.
Argumentação melhor fundamentada. Embora a argumentação seja atribuída a advogados e publicitários, esta habilidade diz

respeito a todas as profissões e não se restringem ao ambiente profissional, afinal persuasão é exigida tanto durante uma entrevista de emprego, na qual você tenta convencer o gestor de que você é o melhor candidato para a vaga, como nas tradicionais discussões de família sobre política ou futebol. Até as redações dos vestibulares prezam que os candidatos aprovados saibam debater suas ideias.

(7) Aplicacoes de Algoritmos

2 Da lógica à programação
2.2 Aplicações de algoritmos

Os algoritmos auxiliam a compreender a lógica de programação de maneira fácil e sem a barreira do idioma. Assim, os conceitos aprendidos podem ser aplicados para programar em qualquer linguagem.
Podemos descrever, por exemplo, algoritmos para:
Ir de casa para a universidade
(figura)
Fazer um sanduíche de presunto e queijo
(figura)
Localizar produtos no mercado
(figura)
Elaborar bons algoritmos e aplicá-los no momento certo é a chave para se escrever programas eficientes e interessantes. Observe alguns exemplos bem sucedidos de algoritmos:
Como um aplicativo de transmissão de vídeo ao vivo pela internet pode ser tão rápida? Ele usa algoritmos de compressão de dados de áudio e vídeo.
(figura)
Como um aplicativo de mapas descobre o caminho correto para ir de Joinville a Florianópolis? Ele usa um algoritmo de localização de rotas.
(figura)
Como é que um programa pode gerar luzes e sombras realistas para o personagem de uma animação 3D? Ele usa algoritmos de renderização.
(figura)
Como uma agência espacial decide como e quando colocar painéis solares na estação espacial? Ela utiliza algoritmos de planejamento e otimização.
(figura)
Dos mais simples aos mais complexos, os algoritmos remetem à mesma coisa: um conjunto de passos para realizar uma tarefa.

(8) Aplicacoes de Algoritmos

2 Da lógica à programação
2.3 Algoritmos e Programação

Por conceito, um programa de computador é um conjunto de instruções que descrevem uma tarefa a ser realizada por um computador.
Podemos dizer, então, que um programa nada mais é do que um algoritmo, já que as “instruções que descrevem uma tarefa” nada mais são do que um conjunto de passos com o mesmo objetivo.
(figura)
Linguagem de programação
Uma linguagem de programação é um meio de dar instruções ao computador. Existem diferentes linguagens de programação, tais como: C, C++, Java, PHP, Python, etc. Todas elas são diferentes uma das outras e remetem ao idioma inglês.
(figura)
O que nós fazemos com eles é o resultado de uma sequência de instruções. Se o programa permitir e assim desejarmos, podemos utilizar códigos para programar algumas funções específicas, por exemplo, descobrir os melhores movimentos para um programa de jogo de xadrez.
Se você aprender a elaborar bons algoritmos, evitará esforços desnecessários escrevendo programas mais eficientes.

(9) Interpretadores e compiladores

3 Interpretadores e compiladores
3 Interpretadores e compiladores

Um computador só entende internamente a chamada linguagem de máquina. Esta linguagem nada mais é que uma sequência ordenada de 1s (uns) e 0s (zeros), ou fisicamente falando, a presença ou ausência de um sinal elétrico. Apenas para ilustrar, veja como fica a primeira frase que você acabou de ler, no início deste parágrafo, convertida para números binários:
(figura)
(é claro que para um programa com instruções e comandos, a conversão não é tão simples assim...)
Para nos comunicarmos com o computador, precisamos converter a linguagem que nós, humanos, entendemos, para a linguagem que a máquina entende. Para isso, usamos os interpretadores ou compiladores.

(10) Interpretadores de algoritmos (Portugol IDE)

3 Interpretadores e compiladores
3.1 Interpretadores de algoritmos (Portugol IDE)

Mesmo sendo os algoritmos representados por uma descrição narrativa, é possível escrevê-los e interpretá-los dentro de um computador através de programas interpretadores. Estes programas reconhecem algumas palavras pré-definidas em linguagem humana, e as convertem em instruções para o computador.
Portugol IDE
Em nosso curso optamos por utilizar um programa desenvolvido para o idioma português, o Portugol IDE. Trata-se de um programa gratuito, distribuído pela Bloodshed Software, e que não requer instalação (é só baixar um arquivo e descompactar). Seu uso é bastante intuitivo e de fácil analogia com linguagens, utilizando o português estruturado, ou "portugol".
Guia rápido do Portugol IDE - instalação e uso

Atenção: essas informações podem ser atualizadas pelo fornecedor sem aviso prévio. Sempre consulte o site do fornecedor.
---
1. Verifique o Java (em navegador diferente do Chrome). Para utilizar o Portugol IDE você precisa ter o Java JRE instalado em seu computador. Se você estiver usando um navegador que não seja o Google Chrome, você pode verificar se o Java JRE está instalado neste link:
► Verificar Java (não Chrome)
2. Caso necessário, instale o Java. Se o Java JRE não estiver instalado, em computadores com MacOS você deve baixá-lo através da Apple Software Update. Se estiver usando Windows ou Linux, você pode baixá-lo através do link:
► Download Java (Win ou Linux)
3. Baixe o Portugol IDE. Acesse o link:
► Página do aplicativo - Portugol IDE
4. Descompacte o arquivo baixado (portugol23.zip). Escolha um local de sua preferência para salvar os arquivos descompactados.
5. Execute o arquivo run.bat para abrir o Portugol IDE (sugerimos criar um atalho para facilitar o acesso). A seguinte janela abrirá:
(Figura - Menu inicial do Portugol IDE)
6. Selecione "Editor de texto". Será mostrada a seguinte janela:
(Figura - Janela de edição de texto do Portugol IDE)
7. Esta é a interface do Portugol IDE que iremos utilizar neste minicurso para escrever pseudocódigos e testá-los. Escreveremos nossos algoritmos na área destacada na figura abaixo:

(Figura - Escrevendo um programa no Portugol IDE)
8. Após escrever o programa, basta executar clicando em F3, ou use os ícones no menu superior (sugerimos consultar a ajuda no site do fornecedor para conhecer melhor outros recursos do aplicativo). Os resultados aparecerão na janela de execução na parte inferior:
(Figura - Janela de execução no Portugol IDE)
9. Agora é só continuar a estudar os conceitos em nosso minicurso, testar os exemplos sugeridos e começar a desenvolver seus próprios programas!

(11) Compiladores de linguagem (DevCpp)

3 Interpretadores e compiladores
3.2 Compiladores de linguagem

Compilação: Sempre que se codifica um algoritmo numa linguagem de programação, este programa precisa ser "traduzido" para a linguagem entendida pela máquina. A este processo chama-se compilação (ou interpretação).
Compiladores: O trabalho de compilação é feito por um programa editor, chamado linkeditor, que além de compilar o código ainda cria um produto final com a extensão .EXE, o qual pode ser executado diretamente no sistema operacional.
DevC++
Como optamos por utilizar a linguagem C para o aprendizado da programação, vamos utilizar um compilador para a linguagem C, chamado DevC++ (ou DevCpp, do inglês "plus plus" = "mais mais"), gratuito, desenvolvido pela Bloodshed. Acesse à versão mais recente pelo link:
► Pagina do aplicativo DevCpp
Caso tenha problemas para instalar a versão mais recente, tente este:
► Download DevCpp 5.4.0
Guia rápido do DevC++ Instalação e uso

Atenção: essas informações podem ser atualizadas pelo fornecedor sem aviso prévio. Sempre consulte o site do fornecedor.

Abra o programa DevC++.
Selecione o menu Arquivo > Novo > Arquivo fonte.
Digite o programa na janela aberta.
Compile o programa, no menu Executar > Compilar (ou clique em F9).
Neste momento será solicitado para salvar o arquivo – selecione o tipo C sources file (*.c), e digite um nome para o seu programa.
Se não houver erros surgirá uma janela informando que est[a tudo ok – clique em Fechar.
Execute o programa clicando no botão "Executar".

(12) Outras ferramentas de aprendizagem

3 Interpretadores e compiladores
3.3 Outras ferramentas de aprendizagem

Existem diversas soluções para auxiliar no aprendizado de programação, algumas bem práticas e interessantes. Não nos dedicamos a testar todas elas profundamente, nem era esse o objetivo. Portanto, atenção para diferenças de sintaxe, e também para a disponibilidade dos respectivos sites.

Atenção! Todos os exemplos e exercícios deste curso foram escritos e testados nas ferramentas Portugol IDE e DevC++. Não é obrigatório usar as mesmas, mas caso decida usar outras, lembre-se que você terá de adaptar tudo de forma a fazê-los mostrar resultados semelhantes.
Bem, pode ser um ótimo exercício para quem gosta de desafios...
---

(13) Algoritmos

4 Algoritmos
4 Algoritmos

Um algoritmo é um conjunto de instruções bem definidas e não ambíguas disposto em uma ordem específica, cada uma das quais devendo ser executadas mecânica ou eletronicamente em um período de tempo finito, no qual se admitem dados de entrada que serão tratados (processados) e transformados em dados de saída (resultado).
Frequentemente são ilustrados por exemplos de receitas culinárias, embora muitos algoritmos sejam mais complexos. Eles podem repetir passos (fazer interações) ou necessitar de decisões (como lógica ou comparação) até que a tarefa seja finalizada.
(Figura - Receita de bolo. Fonte: própria)
Um algoritmo não necessariamente é um programa de computador, e sim os passos para realizar uma tarefa. Sua implementação pode ser feitar por um computador, por equipamentos ou mesmo pelo ser humano. Como na matemática em que um resultado pode ser obtido por diversos cálculos diferentes em alguns casos, nos algoritmos, podemos realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo e espaço que outros.
[Exercícios] Será que você entendeu os conceitos básicos de Algoritmos?

(14) Criando um algoritmo

4 Algoritmos
4.1 Criando um algoritmo

Para que você possa criar um algoritmo é necessário que você analise cada tópico descrito abaixo:
Entender o problema;
Determinar os dados de entrada;
Determinar como os dados serão processados;
Determinar os dados de saída;
Construir o algoritmo;
Testar o algoritmo.
(Figura - Processo de criação de um Algoritmo. Fonte: própria)
Imagine que você vai escrever uma receita de bolo para alguém. Seu problema é "fazer um bolo" e então você precisa saber o que é um bolo, quais são os ingredientes da receita e como você irá misturá-los para ter um bolo.
ENTRADA: Para determinar quais são os dados de entrada você descreve os ingredientes;
PROCESSAMENTO: Para determinar como eles serão processados você descreve o modo de preparo;
SAÍDA: Para determinar os dados de saída você descreve como é o bolo desejado.
Por fim, você constrói o algoritmo (a receita) escrevendo um passo-a-passo. Para testar o algoritmo criado, você o executa. Neste caso, você segue a receita escrita avaliando os resultados de cada passo. Se no final você tiver um bolo conforme o desejado, sua receita está certa. Caso contrário será necessário corrigir os passos desta receita.

(15) Tipos de algoritmo

4 Algoritmos
4.2 Tipos de algoritmo

Existem diversos tipos de algoritmos. Dentre eles, podemos citar: descrição narrativa, fluxograma, pseudocódigo e diagrama de Chapin.
Digamos que o nosso problema seja verificar se um estudante é aprovado ou reprovado, a partir da informação de duas notas. Observe a solução em cada um dos tipos apresentados (não se

preocupe em entender os códigos ainda):
Narrativa
A descrição narrativa utiliza uma linguagem natural para especificar os passos para realizar tarefas.
Exemplo:
Obter as notas das duas provas do estudante;
Somar a nota das provas e ividir o resultado por 2;
Se o resultado for maior ou igual a 7, o estudante foi aprovado; caso contrário, ele foi reprovado.
Fluxograma (ou diagrama de blocos)
Esta é uma forma universal de representação, pois se utiliza de figuras geométricas para ilustrar os passos a serem seguidos para resolução dos problemas.
(Figura - Exemplo de fluxograma. Fonte: própria)
Pseudocódigo
Um pseudocódigo utiliza linguagem estruturada e se assemelha em sua forma normal a um programa escrito na linguagem de programação, também chamado de português estruturado. Neste minicurso iremos representar os algoritmos através da pseudolinguagem.
Exemplo (lembre-se que pode haver diferença na sintaxe de algumas pseudolinguagens):

```
inicio
   variavel real N1, N2, media
   escrever "Digite a primeira nota"
   ler N1
   escrever "Digite a segunda nota"
   ler N2
media <- (N1 + N2)/2
   se N1 >= 7 entao
      escrever "\nAprovado"
   senão
      escrever "\nReprovado"
   fim se
fim
```

Diagrama de Chapin
Apresenta a solução de problemas por meio de um diagrama de quadros através de uma visão hierárquica e estruturada.
(Figura - Exemplo de diagrama de Chapin. Fonte: <http://image.slidesharecdn.com/logicaalgoritmo02algoritmo-1231615383342346-2/95/logica-algoritmo-02-algoritmo-22-728.jpg?cb=1231594394>)
Em nosso curso teremos noções de NARRATIVA, e usaremos PSEUDOCÓDIGO para intermediar a linguagem natural e uma linguagem de programação (como o C, no nosso caso).

(16) Sobre narrativas

5 Sobre narrativas
5 Sobre narrativas

Antes de começarmos a escrever algoritmos através do pseudocódigo, precisamos treinar a construção de algoritmos em uma linguagem mais "humana".
As NARRATIVAS podem parecer a princípio um pouco distantes do nosso objetivo de escrever programas. Porém, são recursos muito úteis para construir em nossa mente as habilidades para resolver problemas através da LÓGICA.

(17) Tipos de narrativas

5 Sobre narrativas
5.1 Tipos de narrativas

Utilizaremos a descrição narrativa para descrever as ações de forma ordenada (narrativa sequencial), para descrever caminhos alternativos através de uma seleção (narrativa de seleção) ou repetir determinadas ações até conseguir o resultado esperado (narrativa de repetição).
[Materiais] Narrativa sequencial
[Materiais] Narrativa de seleção
[Materiais] Narrativa de repetição

(18) Exercitando as narrativas

5 Sobre narrativas
5.2 Exercitando as narrativas

Podemos descrever algumas narrativas sobre as atividades mais simples, como fazer um café. Mas o nosso objetivo aqui é desenvolver as suas habilidades com lógica, para a resolução de problemas mais complexos. Então, como "algoritmizar" um problema em forma de narrativa?
[Exemplos] Soma de dois numeros
Agora, vamos fazer o mesmo para a divisão de dois números:
[Exemplos] Divisao de dois numeros
Note que em uma divisão a ordem de entrada dos números é importante.
Mas... e se o segundo número for zero?
Precisaremos utilizar o tipo de Narrativa de Seleção, pois se o denominador da divisão for zero não poderá haver divisão.
[Exemplos] Divisao de dois numeros se possivel

Dica: nem sempre os critérios, como o da divisão por zero, estarão tão explícitos assim. Avalie sempre todas as situações possíveis de ocorrer, e procure criar seus próprios critérios (claro, usando o bom senso e dentro do que for realmente necessário).
---

(19) Sobre pseudocódigos

6 Sobre pseudocódigos
6 Sobre pseudocódigos

O objetivo de um pseudocódigo (ou pseudolinguagem) é intermediar a linguagem natural e linguagem de programação para que possamos representar um algoritmo. Para isso, é importante entendermos como é a estrutura de um pseudocódigo e também estabelecermos algumas predefinições para "falarmos a mesma língua" quando formos escrevê-los no computador.

(20) Estrutura de um pseudocódigo

6 Sobre pseudocódigos
6.1 Estrutura de um pseudocódigo

A estrutura de um pseudocódigo corresponde às seguintes partes:
Na primeira parte (nem sempre obrigatória) é a identificação do algoritmo.
A segunda parte é onde identificamos as variáveis a serem usadas no restante do programa (declaração de variáveis);
Na terceira parte fica o corpo principal do programa, identificada entre “marcas” de início e fim;
De uma forma genérica, podemos representar assim a estrutura descrita acima:
(animação)
Dependendo do tamanho do programa, começa a ficar mias difícil de lembrar como você o criou e organizou. Por isso é importante fazer “anotações” ou comentários, sobre o que faz cada linha ou cada parte do programa.
Observe o mesmo texto do programa anterior, agora com os comentários:

```
algoritmo “nome”
// Primeiro exemplo de pseudocódigo
declarar (ou variaveis, var)
...
Inicio
// Fazendo o algoritmo imprimir um alô mundo
...
```

Fimalgoritmo

(21) Predefinicoes para escrita de pseudocódigos

6 Sobre pseudocódigos
6.2 Predefinições para escrita de pseudocódigos

Para “falarmos a mesma língua”, é importante combinarmos algumas definições, que serão futuramente adotadas em um aplicativo para escrever pseudocódigos no computador.
1. Sintaxe
Nomes de variáveis: sensíveis a maiúsculas/minúsculas.
Todos comandos em minúsculas.
Usar sublinhado (no lugar de negrito) para destacar as palavras-chave.
Não precisa de “;” no final das linhas.
Procurar escrever tudo sem acentos ou cedilhas.
Seta de atribuição: pode ser <- no lugar de ←.
Assim como nos conjuntos da Matemática, aqui também utilizaremos números inteiros ou reais.
Números: 3.14 e não 3,14 (usar ponto, e não a vírgula, para casaas decimais).
2. Constantes
Informação que não sofre alteração, ou seja, é fixa.
Ex.: atribuir a uma constante pi o valor real de 3,14.
Pode-se declarar constantes conforme abaixo:
```
  declarar
  constante pi <- 3.14 real
```
3. Variáveis
Variáveis do tipo caractere usaremos entre aspas simples;
variáveis do tipo alfanumérico, escreveremos entre aspas duplas;
Variáveis do tipo lógico, escreveremos com "Verdadeiro" e "Falso".
Informação que pode sofrer alteração, ou seja, é variável.
Ex.: pode-se definir variável x dentro do pseudocódigo e este valor poderá se alterar durante a execução do programa.
Pode-se declarar variáveis conforme abaixo:
```
  declarar
  X: inteiro
```
4. Tipos de dados
Inteiro. Exemplos:
-5
0
32
Real (usa-se ponto, e não vírgula). Exemplos:
-9.0
0.5
Caracter (entre aspas simples). Exemplos:
‘1’
‘E’
‘%’
Alfanumérico (entre aspas duplas). Exemplos:
“15”
“Eu”
“Pare!”
“&%@”
Lógico. Exemplos:
Verdadeiro
Falso
Pode-se declarar vários dados em pseudocódigo na mesma linha quando eles forem do mesmo tipo:
```
   declarar
   telefone, idade, valor, cor: alfanumérico
```

(22) Programa

7 O que é um programa?
7 O que é um programa?

Figura - O que é um programa.

Um programa de computador é um conjunto de instruções que descrevem uma tarefa a ser realizada por um computador. Ele pode se referir ao código fonte, escrito em alguma linguagem de programação, ou ao arquivo executável deste código fonte.
O programador, por sua vez, é o profissional que cria os programas e os disponibiliza em mídias ou na internet.
O usuário busca estes programas e os instala em seu computador. Este usuário não precisa entender de programação para utilizar o programa.

(23) Estrutura de um programa

7 O que é um programa?
7.2 Estrutura de um programa

Programar é codificar um algoritmo em uma linguagem de programação específica.
Todo programa de computador tem uma estrutura, que determina a sua forma de funcionamento e a lógica que lhe será permitida.
1. Estrutura básica de um programa em C
```
tipo nomeFunc (declaracao dos parametros)
{
   declaracao de variaveis;
   instrucao 1;
   instrucao 2;
   ...
   instrucao n;
   return var_tipo;
}
```
Comparando com a estrutura de programa da pseudolinguagem:
(figura)
Observe os conjuntos de instruções nas estruturas apresentadas (intrucao 1, instrucao 2, ..., instrucao n). Podemos chamar estes conjuntos de BLOCOS. De forma geral, dizemos que os blocos são conjuntos de ações com uma função definida.
Cada linha de um programa poder conter uma ação, então definimos aonde um bloco começa, com uma marcação de início, e ao terminar as ações com uma marcação de fim.
Desta forma, podemos dizer que um algoritmo pode ser um bloco, pois tem as marcações de “início” e “fim”. Dentro de um programa podem haver vários blocos como, por exemplo, o de declaração de variáveis.

Dica: sempre que estudar um programa, procure identificar aonde estão estas estruturas, seus blocos, início/fim de comandos, instruções, comentários, indentações, etc. Estes elementos agilizam muito a compreensão da lógica de quem programou (mesmo que tenha sido você), e pode ser muito útil, por exemplo, na identificação de erros ou na implementação de uma melhoria.
---
2. Tipos de estruturas
Mostramos nos exemplos anteriores um tipo de estrutura chamada SEQUENCIAL, pois as ações são executadas numa ordem sequencial, ou seja, de cima para baixo, da esquerda para direita. Além desta ainda existem ainda a estrutura CONDICIONAL e a estrutura de REPETIÇÃO, as quais apresentaremos melhor com exemplos e exercícios, mais adiante.
3. Primeiro programa
Vamos escrever então o nosso primeiro código, para começar a se habituar com as ferramentas (interpretadores ou compiladores), e nada melhor do que começar com o clássico "Alô mundo!".

Dica: caso ainda não tenha prática para escrever programas, sugerimos simplesmente copiar dos exemplos e exercícios para fazê-los funcionar da primeira vez. Depois, "brinque" com os valores e parâmetros, o observe os resultados a cada mudança.

Com a prática você irá começar a compreender cada parte do programa.

[Materiais] Alo mundo (veja também a versão em pseudocódigo)
Note que é uma estrutura sequencial, com apenas uma instrução, que está dentro de um único bloco. Simples, não? Porém, vamos avaliar:

Dica: você pode usar um método interessante para testar seus programas. Saiba mais sobre o Teste de Mesa.
[Materiais] Teste de mesa
---

(24) Linguagem C - Principais conceitos

7 O que é um programa?
7.1 Linguagem C - Principais conceitos

Compilação
Sempre que se codifica um algoritmo numa linguagem de programação, este programa precisa ser "traduzido" para a linguagem entendida pela máquina. A este processo chama-se compilação (ou interpretação).
Compiladores
O trabalho de compilação é feito por um outro programa editor, chamado linkeditor, que além de compilar o código ainda cria um produto final com a extensão .EXE, o qual pode ser executado diretamente no sistema operacional. Exemplos de compiladores:
DevC++
Microsoft Visual C++
CppDroid (para celulares Android)

(25) main()

7 O que é um programa?
7.2.1 main()

O ponto de entrada de qualquer programa escrito em C é a função main. Ela é obrigatória e é por ela que o sistema operacional sabe por onde começar a execução do programa.
Parênteses
Os parênteses após o nome main() são a característica que permite que o compilador saiba que se trata de uma função.
Sem eles, o compilador poderia pensar que o nome refere-se a uma variável.
No nosso programa, os parênteses estão vazios, mas nem sempre isso ocorre.
Blocos
Tudo o que estiver entre a chave de abertura de bloco"{" e a chave de fechamento de bloco "}". Equivale, respectivamente, às marcas de início e fim, no pseudocódigo.
Espaços
Você pode inserir espaços em branco, tabulações e pular linhas à vontade em seus programas. O compilador ignora estes caracteres.

É importante destacar que, apesar da estrutura e sintaxe rígida, não há um estilo obrigatório para a escrita de programas em C.
---

(26) system()

7 O que é um programa?
7.2.2 system()

Executa um comando interno do sistema operacional ou um programa (.EXE, .COM, .BAT).
A função system() não faz parte da linguagem C, e sim da biblioteca stdlib.h.

(27) #include

7 O que é um programa?
7.2.3 #include

Diretivas do pré-processador

As duas primeiras linhas não são instruções da linguagem C (não há ponto-e-vírgula no final), mas sim diretivas do pré-processador.
Diretiva é uma indicação, instrução ou norma que deve orientar uma ação ou atividade.
O pré-processador é um programa que examina o programa fonte em C e executa certas modificações com base em instruções chamadas diretivas.
Inicia por (#) e deve ser escrita em uma única linha.
A diretiva #include: inclui outro arquivo em nosso programa fonte.
Na verdade, o compilador substitui esta linha pelo conteúdo do arquivo indicado.

(28) Constantes e comandos de atribuição

7 O que é um programa?
7.3 Constantes e comandos de atribuição

Constantes
Podemos dizer que uma informação é constante quando não sofre alteração durante a execução de um processo. Por exemplo: se definirmos dentro de um algoritmo ou programa que a variável PI valerá 3.14, este valor permanecerá o mesmo até o final da execução.
Tipos de constantes
Constantes numéricos: são valores numéricos escritos na forma usual das linguagens de programação. Podem ser inteiros ou reais. Neste último caso, o separador de decimais é o ponto e não a vírgula, independente da configuração regional do computador onde o programa está sendo executado.
Caracteres constantes: Números que cabem em um único byte (0 a 255). São delimitados por aspas simples ('), ou apóstrofos.
Cadeia de caracteres constantes: qualquer sequência de caracteres delimitada por aspas duplas ("").
Comando de Atribuição
Nas linguagens de programação em geral o comando de atribuição é representado pelo símbolo igual (=), e é utilizado da seguinte forma:
variavel = valor
[Exemplo] Atribuições de variáveis
Veja como funcionam os comandos de atribuição para o pseudocódigo:
[Materiais] Pseudocódigo - Comando de atribuição
[Exemplo] Pseudocódigo - Atribuições com variáveis

(29) Comandos de Entrada-Saida

7 O que é um programa?
7.4 Comandos de Entrada-Saída

Os comandos de entrada e saída são usados para permitir uma interação entre o computador e o usuário (ou o programador).
Saída
A saída é o resultado retornado por uma máquina. É a saída padrão do sistema operacional.
[Materiais] Pseudo x C - Comandos de saida
Códigos especiais
Alguns caracteres não podem ser digitados diretamente dentro do argumento da função. Para tanto, a linguagem usa um recurso, colocando uma barra invertida "\" combinada com outros caracteres.
O \n, por exemplo, informa à função que a impressão deve

continuar na linha seguinte. Da mesma forma funciona para os demais códigos.
[Materiais] Tabela de codigos especiais
Códigos de formatação
[Materiais] Tabela de codigos de formatacao
Entrada
Entendemos como entrada quando, por exemplo, o usuário fornece algum dado para o computador (usuário digita um dado). Primeiro o usuário digita este valor pelo teclado, e em seguida o valor é atribuído à variável.
Note que a atribuição é uma forma de entrada, só que feita pelo próprio algoritmo, e não pelo usuário.

(30) printf

7 O que é um programa?
7.4.1 printf

Exemplo:
printf("Alo mundo!")
A função printf é associada à saída padrão do sistema operacional, geralmente vídeo. Ela não faz parte da linguagem C, e sim da biblioteca stdio.h.
Dentro dos parênteses desta função fica o seu argumento. No exemplo, este argumento está entre aspas "Alo mundo!". Ou seja:
Função: printf();
Argumento: "Alo mundo!"

(31) scanf

7 O que é um programa?
7.4.2 scanf

A função scanf() nos permite ler os dados digitados pelo teclado (entrada padrão). Ela é o complemento da função printf(), e da mesma forma, já está incluída na biblioteca-padrão dos compiladores C.
Marcadores de formatação mais comuns para scanf():
%d Número inteiro
%f Número real (notação normal)
%.1f Número real com 1 casa decimal
%s Sequência de caracteres (string)
%c Caractere simples
Leitura de mais de um valor ao mesmo tempo:
scanf(“%s %d”, &nome, &num);
Vetor de caracteres:
char palavra[15];
scanf (“%s”, &palavra);

(32) Strings

7 O que é um programa?
7.5 Strings

Strings armazenam um texto, chamado cadeia de caracteres. Elas podem conter nenhum, um ou mais caracteres.
As strings sempre estarão entre aspas duplas (""), enquanto o tipo caractere (que é como uma string de apenas um caractere) estará entre aspas simples (').
Internamente, para o computador, a string termina com um caractere nulo '\0', que indica o final desta cadeia.
Por exemplo, esta declaração e inicialização cria uma string com a palavra "MARIA":
char nome[6] = {'M', 'A', 'R', 'I', 'A', '\0'};
Para manter o \0 (caractere nulo) no fim da cadeia, o tamanho total será de um número a mais dos caracteres na palavra "MARIA".
Também podemos escrever a declaração acima desta forma:
char nome[] = "MARIA";
Se a string armazenar um texto com mais de uma linha, cada uma delas deve finalizar com um caractere identificador de nova linha.
Funções das strings em C
Para usar as funções de string deve-se importar a diretiva string.h.
[Materiais] Tabela das funções de string
[Exemplos] (veja os exemplos de funções de string)
Aritmética com strings
A maioria dos operadores funcionam como nos valores numéricos (+=, >, < , ==, etc)
Exemplo:
string texto1;
string texto2;
texto1 = "Fulano ";
texto2 = "da Silva";
texto1 = texto1 + texto2 ; // texto1 contém "Fulano da Silva"

(33) Dados Variaveis e Operadores

8 Dados, variáveis e operadores
8 Dados, variáveis e operadores

Os algoritmos ou programas que você criou até agora realizam operações envolvendo dados e variáveis.
Agora iremos aprender quais são os dados, as variáveis, operadores e quais suas precedências.

(34) Dados

8 Dados, variáveis e operadores
8.1 Dados

Dados
A informação é o que faz com que os computadores sejam necessários, já que eles são capazes de armazenar e processar um grande volume de dados de forma que nós, seres humanos, possamos nos dedicar a outras atividades. O computador armazena quatro tipos primitivos de dados: inteiro, real, caractere ou lógico.
Entretanto cada linguagem de programação pode definir seus tipos de maneiras diferentes. Em nosso pseudocódigo poderemos utilizar os seguintes tipos:
Inteiro: Os dados do tipo inteiro são todo e qualquer número inteiro, sejam eles negativos ou não.
Real: Os dados do tipo real são todos aqueles pertencentes ao conjunto dos números reais (todo número inteiro é também um número real). Para representá-los utilizaremos um ponto no lugar da vírgula (10.4 ao invés de 10,4).
Lógico: Os dados do tipo lógico aceitam apenas dois valores: verdadeiro ou falso (sem fazer a distinção de maiúsculas e minúsculas, ou seja, podemos escrever VERDADEIRO ou verdadeiro e o interpretador irá reconhecer o valor).
Caractere: Os dados do tipo caractere são todos aqueles compostos por um único caractere alfanumérico, ou seja, números (0-9), letras (a-z) e especiais (*, +, -, /, %, $, #, etc.). Sempre escreveremos os caracteres dentro de aspas duplas.
Texto: Os dados de do tipo texto são todos aqueles compostos por quantos caracteres alfanuméricos desejarmos. Se quisermos um texto que contenha aspas, devemos escapá-las, escrevendo o código \"".
Ao definir utilizá-los para declarar constantes e variáveis usamos as seguintes palavras reservadas: inteiro, real, logico, caractere e texto.
[Materiais] Tabela - Pseudo x C - Tipos de dados

(35) Operadores básicos

8 Dados, variáveis e operadores
8.2 Operadores básicos

Em programas de computador uma das principais

funcionalidades é a capacidade de realizar cálculos. Para tanto, é importante sabermos quais os operadores disponíveis e como podemos utilizá-los.
Os operadores básicos são os aritméticos (soma, subtração, multiplicação e divisão).
+ Adição. Ex.:
2+3, x+y
- Subtração. Ex.:
4-2, n-m
* Multiplicação. Ex.:
3*4, A*B
/ Divisão. Ex.:
10/2, X1/X2

(36) Operadores aritmeticos em C

8 Dados, variáveis e operadores
8.3 Operadores aritméticos em C

Além dos operadores básicos, existem em C alguns outros operadores bastante úteis:
pow - potenciação. Ex.: pow(2,3)
sqrt - radiciação. Ex.: sqrt(9)
% - resto da divisão. Ex.: 7%3 (equivalente ao 7 mod 3, na matemática)
Para usar os operadores pow e sqrt, deve-se incluir a biblioteca math.h (o operador % não precisa, pois já está incluso na linguagem C).
Operadores de divisão
% Resto da divisão
9 % 4
(resulta em 1)
27 % 5
(resulta em 2)
div Quociente da divisão
9 div 4
(resulta em 2)
27 div 5
(resulta em 5)
Operadores de potenciação e radiciação
Para usar os operadores pow e sqrt, deve-se incluir a biblioteca math.h (o operador % não precisa, pois já está incluso na linguagem C).
pow Potenciação: x elevado a y. Ex.:
pow(2,3) ou 2^3
sqrt Radiciação: raiz quadrada de x. Ex.:
sqrt(9)
Precedência entre operadores aritméticos
Ao se trabalhar com operadores aritméticos em programação, é imprescindível saber a ordem em que o computador os executa. A isso se chama precedência. São similares à precedência utilizada na matemática.
Primeiro, são realizadas as operações que estiverem entre parênteses. Em seguida, são executadas as operações de potência e raiz quadrada. Depois as operações de multiplicação e divisão, e por último, as operações de soma e subtração. Reescrevendo:

1ª	Parênteses mais internos: (...(...(...)...)...)			
2ª	pow	sqrt		
3ª	*	/	div	%
4ª	+	-		

(37) Operadores relacionais

8 Dados, variáveis e operadores
8.4 Operadores relacionais

Além de fazer cálculos aritméticos, os programas podem trabalhar com operadores relacionais, ou seja, comparar os termos que estão de cada lado do operador. O resultado é do tipo lógico (VERDADEIRO ou FALSO). Por exemplo:
3=3
Estamos afirmando para o computador que 3 é igual a 3.
O computador nos responde dizendo se esta afirmação é verdadeira ou falsa (verdadeira, no caso, pois 3 é, de fato, igual a 3).
Se afirmarmos que 3 é menor que 3 (3<3) então a resposta será falsa (pois, obviamente, 3 não é menor que 3).
Em outro exemplo:
x <- 5 (atribuimos previamente à variável x um valor de 5).
y <- 8 (atribuimos previamente à variável y um valor de 8).
y > x
Como afirmarmos que y é maior que x, o computador nos retornará VERDADEIRO (pois 8 é maior que 5).
por outro lado, se afirmarmos que x>y, ou y<x, a resposta é FALSO (pois 5 não é maior que 8).
No mesmo exemplo acima, se fizermos x<-5 e y<-5, o que aconteceria na expressão y>=x ?
Observe que usamos >=, pois o computador não tem o símbolo equivalente ao "maior ou igual a", da matemática. Neste caso, temos dois operadores na mesma expressão. É como se disséssemos: x>y ou x=y.
A resposta seria, então, VERDADEIRO, pois não atende à expressão x>y, mas atende à x=y.
Operadores relacionais
= Igual a. Ex.: 3=3, x=y
> Maior que. Ex.: 5>4, x>y
< Menor que. Ex.: 3<6, x<y
>= Maior ou igual a. Ex.: 5>=3, x>=y
<= Menor ou igual a. Ex.: 3<=5, x<=y
<> Diferente de . Ex.: 8<>9, x<>y
Precedência entre os operadores relacionais
1ª =
2ª <
3ª <=
4ª >
5ª >=

(38) Operadores lógicos

8 Dados, variáveis e operadores
8.5 Operadores lógicos

Os operadores lógicos são muito importantes na solução de problemas que envolvem lógica.
Eles trabalham de forma semelhante aos operadores relacionais, e podem ser usados de forma combinada.
Podem ser de 3 formas: negação (não), conjunção (e) ou disjunção (ou).
não - negação.
e - conjunção.
ou - disjunção.
Os operadores lógicos obedecem a algumas regras, que colocamos aqui como tabelas verdade. As afirmações (ou proposições) podem ser o resultado de operadores relacionais (Ex.: A <- 15 mod 4 < 19 mod 6)
Operação de negação
Na tabela da negação, se a afirmação (ou proposição) A for verdadeira, então NÃO A será falsa.

A	não A
F	V
V	F

Operação de conjunção
Na tabela da conjunção, se a proposição A for FALSO e a proposição B for VERDADEIRO, então a conjunção da proposição A com a B será FALSO.

A	B	A e B
F	F	F
F	V	F
V	F	F
V	V	V

Operação de disjunção

O mesmo acontece com a tabela dos operadores de disjunção (OU).

A	B	A ou B
F	F	F
F	V	V
V	F	V
V	V	V

Precedência entre Operadores Lógicos
Da mesma forma que os operadores aritméticos, também existe uma precedência para os operadores lógicos (resolver primeiro os NÃO, depois os E e finalmente os OU).

1ª	Não
2ª	e
3ª	ou

(39) Precedência entre todos os Operadores

8 Dados, variáveis e operadores
8.6 Precedência entre todos os Operadores

Avaliando todos os tipos de operadores vistos até aqui, também se estabelece uma precedência entre eles (primeiro resolve-se tudo o que está entre parênteses, depois os operadores aritméticos, seguidos pelos relacionais, e finalmente os operadores lógicos).
1ª Parênteses mais internos
2ª Operadores aritméticos
3ª Operadores relacionais
4ª Operadores lógicos

(40) Estrutura de Controle – Decisão

9 Estrutura de controle – Decisão
9 Estrutura de Controle – Decisão

Estrutura de controle – Decisão (condição)

Tomar decisões - esta é uma das funções mais importantes de um algoritmo ou programa. A partir de escolhas, que podem ser tomadas a partir de uma EXPRESSÃO CONDICIONAL, o programa decide a próxima ação a tomar.
O resultado desta expressão poderá ser Verdadeiro ou Falso, Sim ou não, 0 ou 1, dependendo dos critérios adotados no seu desenvolvimento.
(Figura - Tomada de decisão)
Estudaremos estes tipos de estrutura de condição:
Decisão simples - se...então (ou if...then)
Decisão composta - se...senão...se...(ou if...else...if...)
Múltipla escolha - escolha...caso (ou switch...case)

(41) Estrutura de decisão simples

9 Estrutura de controle – Decisão
9.1 Estrutura de decisão simples

Estrutura de Controle - Condição
Quando esta sequência é organizada como um fluxo de execução, podemos determinar estruturas básicas de controle para criar algoritmos e solucionar problemas. Começaremos estudando as estruturas de decisão (simples ou composta) e as estruturas de seleção (múltipla escolha).
Estrutura de Decisão Simples (se...então)
Uma estrutura de decisão funciona exatamente como o nome diz: o programa deverá tomar uma decisão de que caminho tomar, baseado em uma condição.
Na estrutura de decisão simples, SE determinada condição for verdadeira, ENTÃO faça determinada ação.
Após a decisão, o fluxo de ações segue normalmente.

(42) Estrutura de decisão composta opção 1

9 Estrutura de controle – Decisão
9.2 Estrutura de decisão composta

Uma estrutura de seleção nada mais é do que uma estrutura de seleção disposta de forma encadeada. Mas o que é uma estrutura de seleção encadeada? São várias decisões, uma dentro da outra. Lê-se:
Se a primeira condição for verdadeira, execute a primeira ação; senão avalie a segunda condição. Se esta for verdadeira, execute a segunda ação; senão avalie a terceira condição... E assim por diante.
Note que estamos fazendo escolhas, como em um menu:
se condição 1, ação 1; Se condição 2, ação 2; (e assim por diante).

(43) Estrutura de decisão composta opção 2

9 Estrutura de controle – Decisão
9.2 Estrutura de decisão composta

Uma estrutura de seleção nada mais é do que uma estrutura de seleção disposta de forma encadeada. Mas o que é uma estrutura de seleção encadeada? São várias decisões, uma dentro da outra. Lê-se:
Se a primeira condição for verdadeira, execute a primeira ação; senão avalie a segunda condição. Se esta for verdadeira, execute a segunda ação; senão avalie a terceira condição... E assim por diante.
Note que estamos fazendo escolhas, como em um menu:
se condição 1, ação 1
Se condição 2, ação 2
(e assim por diante).

(44) Estrutura de decisão encadeada opção 1

9 Estrutura de controle – Decisão
9.3 Estrutura de decisão encadeada

Digitar uma estrutura encadeada é muito trabalhoso, quanto mais decisões se necessitar. Para nosso alívio, existe um comando que substitui toda esta estrutura, de forma mais simples: ESCOLHA (CASO).
Interpretamos assim: Caso Condição 1 seja verdadeira (V1), escolha ação 1 (C1); Caso V2, escolha C2; Caso V3, escolha C3; e assim por diante.

(45) Estrutura de decisão encadeada opção 2

9 Estrutura de controle – Decisão
9.3 Estrutura de decisão encadeada

Digitar uma estrutura encadeada é muito trabalhoso, quanto mais decisões se necessitar. Para nosso alívio, existe um comando que substitui toda esta estrutura, de forma mais simples: ESCOLHA (CASO).

Interpretamos assim:
Caso Condição 1 seja verdadeira (V1), escolha ação 1 (C1); Caso V2, escolha C2;
Caso V3, escolha C3;
e assim por diante.

(46) Estrutura de controle – Laços

10 Estrutura de controle – Laços (repetição)
10 Estrutura de controle – Laços (repetição)

Mostraremos aqui o funcionamento das estruturas de repetição (ou laços de repetição). Estas estruturas permitem a repetição de um conjunto de ações, até que a condição desejada seja atingida.

Estudaremos aqui os laços:
do...while (ou Enquanto...Faça)
while (ou Repita...até)
for (ou para)

(47) Laco while (enquanto)

10 Estrutura de controle – Laços (repetição)
10.1 Laço while (enquanto)

Este laço se repete com base em uma PRECONDIÇÃO (a condição é verificada no início do laço):
Enquanto (condição) for verdadeira, faça (ação).
Neste tipo de estrutura, se a condição for já de início falsa, a ação jamais será executada.
Exemplo:
Enquanto (x<3) for verdadeira, escreva o valor de x e incremente x.
Note que cada vez que a estrutura é repetida, o valor de x aumenta em 1, ou seja, é incrementado.
Se o valor inicial de x era 0, na primeira vez que a estrutura for repetida ele valerá 1, na segunda, 2, e assim por diante.
Quando x valer 3, a estrutura não será mais repetida.

(48) Laco do...while (repita)

10 Estrutura de controle – Laços (repetição)
10.2 Laço do...while (repita)

Diferente do enquanto...faça, este laço se repete com base em uma POSCONDIÇÃO (a condição é verificada ao final do laço):
Faça (ação) até que (condição) seja verdadeira.
Este tipo de estrutura é usado quando não sabemos a princípio quantas vezes a ação será repetida, mas precisamos que ela seja executada pelo menos uma vez.
Exemplo:
Escreva o valor de x e incremente x, até que (x<3) seja verdadeira.
Note que cada vez que a estrutura é repetida, o valor de x aumenta em 1, ou seja, é incrementado.
Se o valor inicial de x era 0, na primeira vez que a estrutura for repetida ele valerá 1, na segunda, 2, e assim por diante.
Quando x valer 3, a estrutura não será mais repetida.

(49) Laco para (for)

10 Estrutura de controle – Laços (repetição)
10.3 Laço para (for)

O laço Para repete uma ação por um determinado número de vezes.
Diferente das estruturas Enquanto ou o Repita, este tipo de laço não necessita de uma expressão de incremento, pois este incremento é declarado junto ao próprio comando.
Exemplo:
Neste programa, x vale de 0 até 2. Isso quer dizer que da primeira vez que a estrutura for repetida, x valerá 0; na segunda, x valerá 1, e assim por diante, até que x seja igual a 2.
Adicionalmente, este comando permite definir o passo, ou seja, de quanto em quanto o x será incrementado.
Neste exemplo o incremento é de 1, mas poderia ser de 2 em 2, de 3 em 3, e assim por diante.
Pseudocódigo:

```
inicio
   inteiro x
   para x de 0 ate 2 passo 1
      escrever ""\nO valor de x eh: "",x
   proximo
fim
```

Linguagem C:

```
#include <stdio.h>
void main()
{
  int x;
  for (x=0; x<3; x++){
    printf(""\nO valor de x eh: %d"",x);
  }
}
```

11.3. Exemplo 1 - Somar n números.

(50) Laco para (for)

10 Estrutura de controle – Laços (repetição)
10.3 Laço para (for)

O laço Para repete uma ação por um determinado número de vezes.
Diferente das estruturas Enquanto ou o Repita, este tipo de laço não necessita de uma expressão de incremento, pois este incremento é declarado junto ao próprio comando.

(51) Vetores e matrizes

11 Vetores e matrizes
11 Vetores e matrizes

As variáveis que vimos até agora armazenam somente um valor de cada vez. Mas, às vezes, precisamos agrupar mais de um valor, relacionados à mesma variável. Para isso, precisamos utilizar os conceitos de Vetores e Matrizes.
Matriz é uma coleção de variáveis de mesmo tipo que compartilham o mesmo nome.
Vetor é um termo que tem vários significados, dependendo da literatura. Em nosso minicurso, consideraremos vetor como uma matriz de uma única dimensão (1 linha por n colunas).
As matrizes em geral são caracterizadas por se tratarem de uma única variável de um determinado tamanho que guarda varias informações do mesmo tipo.
Essas informações são gravadas na memória sequencialmente, e são referenciadas através de índices.
Estudaremos as matrizes unidimensionais (vetores) e multidimensionais (matrizes com duas dimensões ou mais).

(52) Vetores

11 Vetores e matrizes
11.1 Vetores

São estruturas indexadas para armazenar dados de mesmo tipo. Pode-se considerar que são matrizes de uma única dimensão, ou seja, de 1 linha por n colunas.
Declaração em C:
Tipo nome_vetor[tamanho];
Exemplo: Matriz "Tempo", contém 10 elementos, que vão de 0 a 9:

Índice	Valor
0	1.2
1	1.5
2	1.3
3	1.4
4	1.4
5	
6	
7	
8	
9	

1.2
1.4
1.5
1.3
1.8
Como declarar:
Tempo[10] = {1.2, 1.5, 1.3, 1.4, 1.4, 1.2, 1.4, 1.5, 1.3, 1.8}
Dica: sempre some ao número da dimensão (Ex.: 9) mais um, que corresponde ao índice do elemento "0" (ou seja, 10 elementos).

(53) Matrizes

11 Vetores e matrizes
11.2 Matrizes

São matrizes linha-coluna, onde o primeiro índice indica a linha e o segundo a coluna.
Matrizes de duas dimensões
Exemplo:
valor[5][2] é uma de matriz de duas dimensões:

valor	0	1	2
0	5.4	3.7	7.5
1	7.8	8.5	4.4
2	8.0	9.3	3.8
3	6.1	9.2	5.2
4	4.9	5.0	4.6

Observação: Em programação, os índices das matrizes sempre começam com zero.

No exemplo acima, dizemos que o valor da linha 1 e coluna 2 é: 4.4
Representamos como:
valor[1][2] = 4.4
Matrizes de mais de duas dimensões
Em programação é permitido que se crie matriz de matriz, conhecido como matriz multidimensional.
Por exemplo: digamos que a matriz bidimensional do exemplo acima são os valores referentes ao ano passado, e queremos adicionar em uma matriz "periodo" os mesmos indices para este ano; temos então 2 anos para a matriz [2][2]:
periodo[1][2][2]
Exemplo 12.2.2 [LngC] Transferindo matrizes multidimensionais para funções

EXEMPLOS

(54) Exemplos - Fazer sanduiche

5 Sobre narrativas
5.1.1 Exemplo de narrativa: fazer sanduíche

Pegar o pão.
Cortar o pão ao meio.
Pegar a maionese.
Passar a maionese no pão.
Pegar e cortar alface e tomate.
Colocar alface e tomate no pão.
Pegar o hambúrguer.
Fritar o hambúrguer.
Colocar o hambúrguer no pão.

(55) Exemplos - Tomar banho

5 Sobre narrativas
5.1.2 Exemplo de narrativa: tomar banho

Entrar no banheiro e tirar a roupa.
Abrir a torneira do chuveiro.
Entrar na água.
Ensaboar-se.
Sair da água.
Fechar a torneira.
Enxugar-se.
Vestir-se.

(56) Exemplos - Trocar lâmpada

5 Sobre narrativas
5.1.3 Exemplo de narrativa: trocar lâmpada

Pegar uma lâmpada nova.
Pegar uma escada.
Posicionar a escada embaixo da lâmpada queimada.
Subir na escada com a lâmpada nova na mão.
Retirar a lâmpada queimada.
Colocar a lâmpada nova.
Descer da escada.
Testar o interruptor.
Guardar a escada.
Jogar a lâmpada velha no lixo.

(57) Exemplos - Soma de dois números

5 Sobre narrativas
5.2.1 Exemplo de narrativa: soma de dois números

Problema: escreva um algoritmo para mostrar o resultado da soma de dois números.
Solução:
Receber os números que serão somados.
Somar os dois números informados.
Mostrar o resultado desta soma.

(58) Exemplos - Divisao de dois números

5 Sobre narrativas
5.2.2 Exemplo de narrativa: divisão de dois números

Problema: escreva um algoritmo para mostrar o resultado da divisão de dois números.
Solução:
Receber os números que serão divididos.
Dividir os o primeiro número informado pelo segundo.
Mostrar o resultado desta divisão.

(59) Exemplos - Divisão de dois números se possível

5 Sobre narrativas
5.2.3 Exemplo de narrativa: divisão de dois números, se possível

Problema: escreva um algoritmo para mostrar o resultado da divisão de dois números, caso seja possível.
Solução:
Receber os números que serão divididos.
Se o segundo número for igual a zero, não poderá haver divisão, pois não existe divisão por zero; caso contrário, dividir o primeiro número informado pelo segundo.
Mostrar o resultado desta divisão.

(60) Exemplos - Atribuicoes de variáveis

7 O que é um programa?
7.3.1 Exemplos de atribuições de variáveis:

x = 25
y = x + 15 – 3
z = y – x + rad(x) – pot(y,2)

(61) Exemplos - Pseudocodigo - Atribuicoes de variáveis

7 O que é um programa?
7.3.2 Exemplos de atribuições de variáveis em pseudocódigo:

```
x <- 25
y <- x + 15 – 3
z <- y – x + rad(x) – pot(y,2)
```

(62) Exemplos - Pseudocodigo - Aplicacao de atribuicoes usando variaveis

7 O que é um programa?
7.3.3 Aplicação de atribuições usando variáveis

```
inicio
\\ Declaracao de variaveis
   variavel real a, valor1, valor2, matriz[4][10]
   variavel texto nome_do_estudante
   variavel logico sinalizador
\\ Calculos
   a <- 3
   Valor1 <- 1.5
   Valor2 <- Valor1 + a
   matriz[3][9] <- a/4 - 5
   nome_do_estudante <- "José da Silva"
   sinalizador <- FALSO
\\ Mostrar resultados
   escrever a
   escrever "\n",Valor1
   escrever "\n",Valor2
   escrever "\n",matriz[3][9]
   escrever "\n",nome_do_estudante
   escrever "\n",sinalizador
fim
```

(63) Exemplos - Constantes numéricos

7 O que é um programa?
7.3.4 Exemplos de constantes numéricos

```
8734
60
13
```

(64) Exemplos - Caracteres constantes

7 O que é um programa?
7.3.5 Exemplos de caracteres constantes

```
'A'
'3'
'x'
'\n'
'\t'
```

(65) Exemplos - Cadeia de caracteres constantes

7 O que é um programa?
7.3.6 Exemplos de cadeia de caracteres constantes

```
"Alo mundo"
"\nEstou entendendo"
```

(66) Exemplos - Pseudocodigo - Aplicacao de caracteres especiais

7 O que é um programa?
7.3.7 Exemplo - pseudocódigo: aplicação de caracteres especiais

```
inicio
   escrever "Estou \n entendendo"
fim
```

(67) Exemplos - Constantes no C

7 O que é um programa?
07.3.8 Exemplo - Constantes no C

```
const double pi;
```

Pode-se declarar vários dados constantes em C numa mesma linha se eles forem todos do mesmo tipo.
Observação: no momento da declaração de um dado constante em C, pode-se também inicializar a constante. Exemplo:

```
const double pi = 3.14;
```

(68) Exemplos - Alo mundo

7 O que é um programa?
7.4.1 Alô mundo!

Escreva um programa que mostre na tela a frase "Alo mundo!"
Solução:

```
#include <stdio.h>
void main()
{
   printf("Alo mundo!");
}
```

(69) Exemplos - Pseudocodigo - Alo mundo

7 O que é um programa?
7.4.2 Pseudocódigo - Alô mundo!

Exercício: Escreva um programa que mostre na tela a frase "Alo mundo!"
Solução:

```
inicio
   escrever "Alo mundo!"
fim
```

(70) Exemplos - Entrar com inteiro e mostrar

7 O que é um programa?
7.4.3 Exemplo - Entrar com número inteiro e mostrar o valor digitado

```
#include < stdio.h >
int main()
{
 int a;
 printf(""Entre com um numero inteiro\n"");
 scanf(""%d"", &a);
 //trecebe um inteiro do usuario
 printf(""O n[umero que voce entrou foi %d\n"", a);
 return 0;
}
```

(71) Exemplos - Trocar 2 numeros

7 O que é um programa?
7.4.4 Exemplo - Trocar dois números

```
#include <stdio.h>
int main()
{
int x, y, temp;
printf("Entre com os valores de x e y:\n");
scanf("%d%d", &x, &y);
printf("Antes da troca\nx = %d\ny = %d\n",x,y);
//usando temp para trocar:
temp = x; //armazenar x em temp
x = y; //armazenar y em x
```

```
y = temp; //mover temp para y
printf("Apos a troca\nx = %d\ny = %d\n",x,y);
return 0;
}
Resultado na tela:
Entre com os valores de x e y:
4
9
Antes da troca
x = 4
y = 9
Apos a troca
x = 9
y = 4
```

(72) Exemplos - Funcoes string – imprimir

```
7 O que é um programa?
7.5.1 Exemplos de funções de strings - imprimir

#include <stdio.h>
int main()
{
   char texto[100];
   printf("Escreva algo\n");
   scanf("%s", texto);
/* %s é usado para permitir a entrada de texto pelo usuário*/
   printf("Voce escreveu %s\n",texto);
   return 0;
}
```

(73) Exemplos - Funcoes string – copiar opção 1

```
7 O que é um programa?
7.5.2 Exemplos de funções de strings - copiar

#include <stdio.h>
#include <string.h>
main()
{
  char origem[] = "Linguagem C";
  char destino[50];
  strcpy(destino, origem);
  printf("Texto de origem: %s\n", origem);
  printf("Texto de destino: %s\n", destino);
  return 0;
}
```

(74) Exemplos - Funcoes string – copiar opção 2

```
7 O que é um programa?
7.5.2 Exemplos de funções de strings - copiar

#include <stdio.h>
#include <string.h>
main()
{
  char origem[] = "Linguagem C";
  char destino[50];
  strcpy(destino, origem);
/*strcpy copiará o texto de origem para destino*/
  printf("Texto de origem: %s\n", origem);
  printf("Texto de destino: %s\n", destino);
  return 0;
}
```

(75) Exemplos - Funcoes string – comprimento

```
7 O que é um programa?
7.5.3 Exemplos de funções de strings - Comprimento

#include <stdio.h>
#include <string.h>
int main()
{
  char a[100];
  int comprimento;
  printf("Digite uma string para que seja calculado o seu
comprimento\n");
  gets(a);
  comprimento = strlen(a);
/* A função strlen(string) informa o comprimento da string*/
  printf("O comprimento da string digitada eh:
%d\n",comprimento);
  return 0;
}
```

(76) Exemplos 07.5.4 Funcoes string – reverter

```
7 O que é um programa?
7.5.4 Exemplos de funções de strings - Reverter

#include <stdio.h>
#include <string.h>
  int main()
{
  char arr[100];
  printf("Digite uma string para ser revertida\n");
  gets(arr);
  strrev(arr);
/*strrev(string) reverte a string dada */
  printf("A reversa da string digitada é \n%s\n",arr);
  return 0;
}
```

(77) Exemplos - Funcoes string – concatenar

```
7 O que é um programa?
7.5.5 Exemplos de funções de strings - Concatenar

#include <stdio.h>
#include <string.h>
int main()
{
  char a[100], b[100];
  printf("Entre com a primeira string\n");
  gets(a);
  printf("Entre com a segunda string\n");
  gets(b);
  strcat(a,b);
/*strcat acrescentará a string b na string a*/
  printf("A string obtida na concatenacao eh %s\n",a);
  return 0;
}
```

(78) Exemplos - Funcoes string – comparar

```
7 O que é um programa?
7.5.6 Exemplos de funções de strings - Comparar

#include <stdio.h>
#include <string.h>
int main()
{
  char a[100], b[100];
  printf("Entre com a primeira string\n");
  gets(a);
  printf("Entre com a segunda string\n");
  gets(b);
/*strcmp(string, string) retorna 0 se as strings forem iguais, caso
contrário, elas não são iguais*/
  if( strcmp(a,b) == 0 )
  printf("\nAs strings digitadas sao iguais.");
```

```
  else
  printf("\nAs strings digitadas nao sao iguais.");
  return 0;
}
```

(79) Exemplos - Variáveis no C

8 Dados, variáveis e operadores
8.1.1 Exemplo - Variáveis no C

```
int x;
```

Pode-se declarar vários dados variáveis em C numa mesma linha se eles forem todos do mesmo tipo. Exemplo:

```
char letra, tom, nome;
```

Obs.: no momento da declaração de um dado variável em C, pode-se também inicializar a variável. Exemplo:

```
double y = 5;
```

(80) Exemplos - Quatro operações básicas

8 Dados, variáveis e operadores
8.2.3 Exemplo - Quatro operações básicas

```
#include <stdio.h>
int main()
{
int num1, num2, soma, subtracao, multiplicacao;
float divisao;
printf("Entre dois inteiros\n");
scanf("%d%d", &num1, &num2);
soma = num1 + num2;
subtracao = num1 - num2;
multiplicacao = num1 * num2;
divisao = num1 / (float)num2;
//typecasting
printf("Soma = %d\n",soma);
printf("Diferenca = %d\n",subtracao);
printf("Multiplicacao = %d\n",multiplicacao);
printf("Divisao = %.2f\n",divisao);
return 0;
}
Resultado na tela:
Entre dois inteiros
4
3
Diferenca = 1
Multiplicacao = 12
Divisao = 1.0
```

(81) Exemplos - Pseudocódigo - Área do circulo

8 Dados, variáveis e operadores
8.2.4 Exemplo - Pseudocódigo - Área do círculo

```
inicio
   real pi <- 3.142
   real raio , area
   escrever "Entre com o raio do circulo: \n"
   ler raio
   area <- pi * raio * raio
   escrever "Area do circulo = \n" , area
fim
Resultado:
Entre com o raio do circulo:
12
Area do circulo =
452.448
```

(82) Exemplos - Área do circulo

Dados, variáveis e operadores
08.2.5 Exemplo - Área do círculo

```
#include <stdio.h>
#include <math.h>
#define PI 3.142
void main()
{
float raio, area;
printf("Entre com o raio do circulo: \n");
scanf("%f", &raio);
area = PI * pow(raio, 2);
printf("Area do circulo = %5.2f\n", area);
}
Resultado:
Entre com o raio do circulo:
12
Area do circulo =
452.448
```

(83) Exemplos - Decisao simples

9 Estrutura de controle – Decisão
9.1.1 Exemplo - Decisão simples

Programa que solicita um número inteiro ao usuário, e o exibe somente se for positivo.

```
inicio
   inteiro num
   ler num
   se (num > 0) entao
      escrever num
   fimse
fim
```

(84) Exemplos - Pseudocódigo - Divisão de dois números

9 Estrutura de controle – Decisão
9.1.2 Exemplo - Divisão de dois números

Programa que solicita um número inteiro ao usuário, e o exibe somente se for positivo.

```
inicio
   inteiro num
   ler num
   se (num > 0) entao
      escrever num
   fim se
fim
```

(85) Exemplos - Maior de três números

9 Estrutura de controle – Decisão
9.1.3 Exemplo - Maior de três números

```
#include <stdio.h>
void main()
{
int a,b,c;
printf("Entre tres numeros quaisquer:\n");
scanf("%d%d%d",&a, &b, &c);
if(a>b&&a>c)
printf("O maior numero eh: %d",a);
else if(b>c)
printf("O maior numero eh: %d",b);
else
printf("O maior numero eh: %d",c);
}
```

(86) Exemplos - Par ou impar

9 Estrutura de controle – Decisão

9.1.4 Exemplo - Par ou ímpar

```
#include <stdio.h>
main()
{
 int n;
 printf("Entre um numero inteiro\n");
 scanf("%d",&n);
/*Se o numero for divisivel por 2 mostra par, senao n eh impar*/
 if ( n%2 == 0 )
  printf("Par\n");
 else
  printf("Impar\n");
 return 0;
}
```

(87) Exemplos - Ano bissexto

9 Estrutura de controle – Decisão
9.1.5 Exemplo - Ano bissexto

```
#include <stdio.h>
int main()
{
 int ano;
 printf("Entre um ano para verificar se eh bissexto\n");
 scanf("%d", &ano);
 if ( ano%400 == 0)
 printf("\n%d eh ano bissexto.", ano);
 else if ( ano%100 == 0)
 printf("\n%d nao eh ano bissexto.", ano);
 else if ( ano%4 == 0 )
 printf("\n%d eh ano bissexto.", ano);
 else
 printf("\n%d nao eh ano bissexto.", ano);
 return 0;
}
```

(88) Exemplos - Média de duas notas

9 Estrutura de controle – Decisão
9.1.6 Exemplo - Média de duas notas

O programa solicita duas notas ao usuário, e calcula a média. Se for maior ou igual a 60, mostra uma mensagem parabenizando pela aprovação:

```
inicio
   real nota1
   real nota2
   real media
   ler nota1
   ler nota2
   media <- ( nota1 + nota2 ) / 2
   se media >= 60 entao
      escrever "Parabéns! Aprovado!"
   fimse
fim
```

Como no programa anterior, solicita duas notas ao usuário e calcula a média. Mas neste caso, se não atender a condição (senão), faz outra coisa, apresenta uma mensagem de reprovação.

```
inicio
   real nota1
   real nota2
   real media
   ler nota1
   ler nota2
   media <- ( nota1 + nota2 ) / 2
   se media >= 60 entao
      escrever "Parabéns! Aprovado!"
   senao
      escrever "Reprovado!"
   fimse
fim
```

(89) Exemplos - Pseudocódigo - (SE) numero maior que 0 ou não

9 Estrutura de controle – Decisão
9.2.1 Exemplo - Pseudocódigo - (SE) número maior que 0 ou não

```
inicio
   inteiro num
   escrever "Entre com um inteiro: "
   ler num
   se (num >= 0) entao
      escrever num, " eh maior ou igual a zero"
   senao
      escrever num, " eh menor que zero"
   fimse
fim
```

(90) Exemplos - Pseudocódigo - (SE composta) numero maior que 0 ou não

9 Estrutura de controle – Decisão
9.2.2 Exemplo - Pseudocódigo - (SE composta) número maior que 0 ou não

```
inicio
   inteiro num
   ler num
   se ( num > 0 ) entao
      escrever num , " é maior que zero"
   senao
      escrever num , " é menor ou igual a zero"
   fimse
fim
```

(91) Exemplos - Pseudocódigo - (SE composta) numero maior menor ou igual a zero

9 Estrutura de controle – Decisão
9.2.3 Exemplo - Pseudocódigo - (SE composta) número maior, menor ou igual a zero

```
inicio
   inteiro num
   ler num
   se ( num > 0 ) entao
      escrever num , " é maior que zero"
   senao
      se ( num < 0 ) entao
         escrever num , " é menor que zero"
      senao
         escrever num , " é igual a zero"
      fimse
   fimse
fim
```

(92) Exemplos - Pseudocódigo - 4 operações

9 Estrutura de controle – Decisão
9.2.4 Exemplo - Pseudocódigo - Quatro operações

```
inicio
   inteiro num_1 , num_2
   caracter operacao
   ler num_1
   ler num_2
   ler operacao
```

```
   se ( operacao = "+" ) entao
      escrever ( num_1 + num_2 )
   senao
      se ( operacao = "-" ) entao
         escrever ( num_1 - num_2 )
      senao
         se ( ( operacao = "*" ) ou ( operacao = "X" ) ) entao
            escrever ( num_1 * num_2 )
         senao
            se ( operacao = "/" ) entao
               escrever ( num_1 / num_2 )
            senao
               escrever ( "Operação inválida" )
            fimse
         fimse
      fimse
   fimse
fim
```

(93) Exemplos - Pseudocódigo - Ano bissexto

9 Estrutura de controle – Decisão
9.2.5 Exemplo - Ano bissexto

```
#include <stdio.h>
int main()
{
 int ano;
 printf("Entre um ano para verificar se eh bissexto\n");
 scanf("%d", &ano);
 if ( ano%400 == 0)
 printf("\n%d eh ano bissexto.", ano);
 else if ( ano%100 == 0)
 printf("\n%d nao eh ano bissexto.", ano);
 else if ( ano%4 == 0 )
 printf("\n%d eh ano bissexto.", ano);
 else
 printf("\n%d nao eh ano bissexto.", ano);
 return 0;
}
```

(94) Exemplos - Pseudocódigo - 4 operações

9 Estrutura de controle – Decisão
9.3.1 Exemplo - Pseudocódigo - Quatro operações

```
inicio
   inteiro num_1 , num_2
   caracter operacao
   ler num_1
   ler num_2
   ler operacao
   escolhe operacao
      caso "+":
         escrever ( num_1 + num_2 )
      caso "-":
         escrever ( num_1 - num_2 )
      caso "*" , "X":
         escrever ( num_1 * num_2 )
      caso "/":
         escrever ( num_1 / num_2 )
      defeito:
         escrever ( "Operação inválida" )
   fimescolhe
fim
```

(95) Exemplos - Checar vogal

9 Estrutura de controle – Decisão
9.3.2 Exemplo - Checar vogal

```
#include < stdio.h >
int main()
{
 char ch;
 printf("Input a character\n");
 scanf("%c", &ch);
 switch(ch)
 {
 case 'a':
 case 'A':
 case 'e':
 case 'E':
 case 'i':
 case 'I':
 case 'o':
 case 'O':
 case 'u':
 case 'U':
 printf("%c is a vowel.\n", ch);
 break;
 /*if ch matches any case then it prints & breaks the execution
*/
 default:
 printf("%c is not a vowel.\n", ch);
/*if the ch is not from the cases then prints ch is not a vowel */
 }
 return 0;
}
```

(96) Exemplos - Pseudocódigo - x menor que 3

10 Estrutura de controle – Laços (repetição)
10.1.1 Exemplo - Pseudocódigo - x menor que 3

```
inicio
   inteiro x
   enquanto x<3 faz
      escrever ""\nO valor de x é: "",x
      x<-x+1
   fim enquanto
fim
```

(97) Exemplos - Somar dígitos

10 Estrutura de controle – Laços (repetição)
10.1.2 Exemplo - Somar dígitos

```
#include <stdio.h>
int main()
{
   int n, sum = 0, remainder;
   printf("Enter an integer\n");
   scanf("%d",&n);
   while(n != 0)
   {
     remainder = n % 10;
    /*stores unit place digit to remainder*/
     sum = sum + remainder;
     n = n / 10;
    /*dividing no to discard unit place digit*/
   }
   printf("Sum of digits of entered number = %d\n",sum);
   return 0;
}
```

(98) Exemplos - Quadrado dos números de 0 a 20

10 Estrutura de controle – Laços (repetição)
10.1.3 Exemplo - Quadrado dos números de 0 a 20

```
#include <stdio.h>
```

```
void main()
{
  int val, ind=0;
  printf("Quadrados dos numeros inteiros\n");
  while (ind<=20){
        val=ind*ind;
        printf("%d*%d=%d\n",ind, ind, val);
        ind++;
  }
}
```

(99) Exemplos - Pseudocódigo - Quadrado dos números 0 a 20

10 Estrutura de controle – Laços (repetição)
10.1.4 Exemplo - Pseudocódigo - Quadrado dos números de 0 a 20

```
inicio
inteiro val, ind
    ind <- 0
    escrever "Quadrados dos numeros inteiros\n"
    enquanto ind <= 20 faz
        val <- ind*ind
        escrever ind, " * ", ind," = ",val, "\n"
        ind <- ind + 1
    fimenquanto
fim
```

(100) Exemplos – Pseudocódigo - x menor que 3

10 Estrutura de controle – Laços (repetição)
10.2.1 Exemplo - Pseudocódigo - x menor que 3

```
inicio
   inteiro x
   repete
      escrever ""\nO valor de x é: "",x
      x <- x+1
   ate x>=3
fim
```

(101) Exemplos - Pseudocódigo - Mostra x de 0 a 2

10 Estrutura de controle – Laços (repetição)
10.2.2 Exemplo - Pseudocódigo - Mostra x de 0 a 2

Escreva o valor de x e incremente x até que x seja maior que 2.

```
x<-0
repita
escreval("O valor de x é: ",x)
x<-x+1
ate (x>2)
```

(102) Exemplos - x de 0 a 3

10 Estrutura de controle – Laços (repetição)
10.3.1 Exemplo - x de 0 a 3

Exemplo:
Neste programa, x vale de 0 até 2. Isso quer dizer que da primeira vez que a estrutura for repetida, x valerá 0; na segunda, x valerá 1, e assim por diante, até que x seja igual a 2.
Adicionalmente, este comando permite definir o passo, ou seja, de quanto em quanto o x será incrementado.
Neste exemplo o incremento é de 1, mas poderia ser de 2 em 2, de 3 em 3, e assim por diante.

```
#include <stdio.h>
void main()
{
  int x;
  for (x=0; x<3; x++){
     printf(""\nO valor de x eh: %d"",x);
  }
}
```

(103) Exemplos - Pseudocódigo - x de 0 a 3

10 Estrutura de controle – Laços (repetição)
10.3.2 Exemplo - Pseudocódigo - x de 0 a 3

Exemplo:
Neste programa, x vale de 0 até 2. Isso quer dizer que da primeira vez que a estrutura for repetida, x valerá 0; na segunda, x valerá 1, e assim por diante, até que x seja igual a 2.
Adicionalmente, este comando permite definir o passo, ou seja, de quanto em quanto o x será incrementado.
Neste exemplo o incremento é de 1, mas poderia ser de 2 em 2, de 3 em 3, e assim por diante.

```
inicio
   inteiro x
   para x de 0 ate 2 passo 1
      escrever ""\nO valor de x eh: "",x
   proximo
fim
```

(104) Exemplos - Matemática - Conjuntos numéricos

11 Vetores e matrizes
11.1 Exemplo - Matemática - conjuntos numéricos

Conjuntos numéricos, na matemática:
Naturais: N = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...}
Naturais não nulos: N* = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...}
Inteiros: Z = {... -4, -3, -2, -1, 0, 1, 2, 3, 4, ...}
Inteiros não negativos: Z+ = {0, 1, 2, 3, 4, 5, 6, ...}
Inteiros não positivos: Z- = {..., -5, -4, -3, -2, -1, 0}
Inteiros não negativos e não nulos: Z*+ = {1, 2, 3, 4, 5, 6, 7, ...}
Inteiros não positivos e não nulos: Z*- = {... -4, -3, -2, -1}
Outros conjuntos:
Decimais finitos (Ex.: 743,8432)
Decimais infinitos periódicos ou dízimas periódicas (Ex.: 12,050505…)
Racionais (inteiros, decimais finitos + decimais infinitos periódicos).
Irracionais (decimais infinitos não periódicos). Ex. PI (3,14159265…) e todas as raízes não exatas (como a raiz de 2 = 1,4142135...)
Fonte: http://www.infoescola.com/matematica/conjuntos-numericos/

(105) Exemplos - Vetores - Entrada de dados

11 Vetores e matrizes
11.1.1 Exemplo - Vetores - Entrada de dados

```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
float notas[10];
int indice;
printf (“Lendo as notas:\n”);
for (indice=0; indice<10; indice++){
printf (“Digite a nota do proximo estudante:”);
scanf (“%f”, &notas[indice]);
}
printf (“Exibindo as notas digitadas:\n”);
for (indice=0; indice<10; indice++){
printf (“A nota %f foi armazenada na posicao %d do vetor.\n”,notas[indice], indice);
}
system (“pause”);
```

```
}
```

(106) Exemplos - Vetores - Ordenação por seleção

11 Vetores e matrizes
11.1.2 Exemplo - Vetores - Ordenação por seleção

```
#include < stdio.h >
int main()
{
  int array[100], n, c, d, position, swap;
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for ( c = 0 ; c < n ; c++ )
  scanf("%d", &array[c]);
  for ( c = 0 ; c < ( n - 1 ) ; c++ )
{
  position = c;
/* for all array, from position c selecting smallest element in array and swap with c position*/
  for ( d = c + 1 ; d < n ; d++ )
{
  if ( array[position] > array[d] )
  position = d;
}
  if ( position != c )
{
  swap = array[c];
  array[c] = array[position];
  array[position] = swap;
}
}
  printf("Sorted list in ascending order:\n");
  for ( c = 0 ; c < n ; c++ )
  printf("%d\n", array[c]);
  return 0;
}
```

(107) Exemplos - Matrizes - Entrada de dados

11 Vetores e matrizes
11.2.1 Exemplo - Matrizes - Entrada de dados

```
#include <stdio.h>
#include <stdlib.h>
int main ( ){
float notas[5][2];
int lin,col;
printf (“INICIANDO O LOOP DE LEITURA \n \n”);
for (lin=0;lin<5;lin++)
for (col=0;col<2;col++) {
printf (“Digite a nota %d do estudante %d: “, col, lin);
scanf (“%f”, &notas[lin][col]);}
printf (“\nINICIANDO O LOOP DE EXIBICAO \n \n”);
for (lin=0;lin<5;lin++)
for (col=0;col<2;col++)
printf (“nota = %.2f \n”,notas[lin][col]);
system (“pause”);}
```

EXERCÍCIOS

(108) Exercício - Identificar etapas algoritmo opção 1

4 Algoritmos
4.1 Exercício - Identificar etapas algoritmo

Identifique no algoritmo, para cada passo, os dados de entrada (E), processamento (P) e saída (S), e responda a alternativa correta:
1. Receba código da peça.
2. Receba valor da peça.
3. Receba Quantidade de peças.
4. Calcule o valor total da peça (Quantidade * Valor da peça).
5. Mostre o código da peça e seu valor total.
() 1-E, 2-E, 3-E, 4-P, 5-S. (correta)
() 1-S, 2-S, 3-S, 4-S, 5-E.
() 1-P, 2-P, 3-P, 4-P, 5-S.
() 1-E, 2-E, 3-E, 4-S, 5-P.
() 1-P, 2-P, 3-P, 4-E, 5-S.

(109) Exercício - Receita de bolo

4 Algoritmos
04.2 Exercício - Receita de bolo

Fazendo uma analogia do algoritmo com uma receita de bolo, relacione os elementos da primeira linha com os da segunda linha.
(x) Ingredientes (y) Bolo pronto (z) Modo de preparo
(1) Saída (2) Dados de entrada (3) Processamento
Responda: qual é a correspondência correta?
x-1, y-2, z-3.
x-2, y-3, z-1.
x-2, y-1, z-3. (correta)
x-1, y-3, z-2.

(110) Exercício - Ordenar sequencia algoritmo

4 Algoritmos
4.3 Exercício - Ordenar sequencia algoritmo

Leia as sentenças a seguir e ordene conforme a necessidade na criação de algoritmos:

(w) Construir o algoritmo.
(x) Entender o problema.
(y) Testar o algoritmo.
(z) Determinar os dados de entrada, processamento e saída.

Qual é a ordem correta das sentenças?
w – x – y – z.
z – x – y – w.
x – y – z – w.
x – z – w – y. (correta)

(111) Exercício - Identificar etapas algoritmo opção 2

4 Algoritmos
4.4 Exercício - Identificar etapas algoritmo

Identifique no algoritmo, para cada passo, os dados de entrada (E), processamento (P) e saída (S), e responda a alternativa correta:
1. Receba código da peça.
2. Receba valor da peça.
3. Receba Quantidade de peças.
4. Calcule o valor total da peça (Quantidade * Valor da peça).
5. Mostre o código da peça e seu valor total.

1-E, 2-E, 3-E, 4-P, 5-S. (correta)
1-S, 2-S, 3-S, 4-S, 5-E.
1-P, 2-P, 3-P, 4-P, 5-S.
1-E, 2-E, 3-E, 4-S, 5-P.
1-P, 2-P, 3-P, 4-E, 5-S.

(112) Exercício (Narrativa de seleção) Divisão de 2 números

5 Sobre narrativas
5.1 (Narrativa de seleção) Divisão de dois números

Faça um algoritmo para mostrar o resultado da divisão
de dois números:
Passo 1 – Receber dois números que serão divididos;
Passo2 – Se o segundo número for igual a zero, não poderá haver divisão, pois não existe divisão por zero; caso contrário, dividir os números e mostrar o resultado da divisão.

(113) Exercício - Somar 2 números

5 Sobre narrativas
5.2.1 Exercício - Somar 2 números

Escreva um algoritmo para somar dois números.

--
Resposta:
1. Receber os dois números.
2. Somar os dois números.
3. Mostrar o resultado obtido.

(114) Exercício - Somar conteúdos retângulos opção 1

5 Sobre narrativas
5.2.2 Exercício - Somar conteúdos de retângulos

Considere três retângulos, A, B e C. Faça um algoritmo que escreva números dentro de cada retângulo, de forma que atenda a expressão A+B=C.

--
Resposta:

1. Escreva o primeiro número no retângulo A
2. Escreva o segundo número no retângulo B
3. Some o número do retângulo A com número do retângulo B e coloque o resultado no
retângulo C.

[ ]A + [ ]B = [ ]C

Obs.: os caracteres [ ] acima representam um retângulo.

(115) Exercício - Somar 3 números

5 Sobre narrativas
5.2.3 Exercício - Somar 3 números

Escreva um algoritmo para somar três números.

--
Resposta:
1. Receber os três números.
2. Somar os três números.
3. Mostrar o resultado obtido.

(116) Exercício - Somar conteúdos retângulos opção 2

5 Sobre narrativas
5.2.4 Exercício - Somar conteúdos de retângulos

Considere dois retângulos, A e B. Faça um algoritmo que escreva números dentro de cada retângulo, de forma que atenda a expressão A=A+B.

--
Resposta:

1. Escreva o primeiro número no retângulo A
2. Escreva o segundo número no retângulo B
3. Some o número do retângulo A com número do retângulo B e coloque o resultado no retângulo A.

[ ]A = [ ]A + [ ]B

Obs.: os caracteres [ ] acima representam um retângulo.

(117) Exercício - Criação de algoritmos

5 Sobre narrativas
5.2.5 Exercício - Criação de algoritmos

Pergunta: Leia as sentenças a seguir e ordene conforme a necessidade na criação de algoritmos:

(w) Construir o algoritmo.
(x) Entender o problema.
(y) Testar o algoritmo.
(z) Determinar os dados de entrada, processamento e saída.

Qual é a ordem correta das sentenças?
w – x – y – z.
z – x – y – w.
x – y – z – w.
x – z – w – y.

(118) Exercício - Pseudocódigo - Teste dos códigos especiais

7 O que é um programa?
7.1 Exercício - Pseudocódigo - Teste dos códigos especiais

1. Digite o programa:

```
inicio
   escrever "Estou \n entendendo"
fim
```

(119) Exercício - Teste dos códigos especiais

7 O que é um programa?
7.2 Exercício - Teste dos códigos especiais

1. Digite o programa:

Pseudocódigo:

```
inicio
   escrever "Estou \n entendendo"
fim
```

Linguagem C:

```
#include <stdio.h>
   void main()
   {
      printf("Estou \n entendendo");
   }
```

2. Altere o programa usando os demais códigos especiais da função printf. Altere as frases, espaços ou posições dos códigos dentro dos argumentos. Observe os resultados a cada alteração.

(120) Exercicio - String

7 O que é um programa?
7.3 Exercício - String

Uma string de texto é uma sequência de caracteres composta de linhas, cada linha composta por nenhum, um ou mais caracteres, acrescentada de um caractere identificador de nova linha.

Verdadeiro. (correta)
Falso.

Verdade, cada linha deve conter nenhum, um ou mais caracteres finalizadas por um caractere de nova linha.

(121) Exercício – programa

7 O que é um programa?
7.4 Exercício - Programa

Sobre o conceito de programa, indique a alternativa INCORRETA:

() é uma lista de eventos que o computador analisa para escolher qual deles o usuário vai utilizar. (correta)
() é a codificação de um algoritmo numa linguagem de programação específica.
() é um conjunto de instruções que descrevem uma tarefa a ser realizada por um computador.
()refere-se ao código fonte, escrito em alguma linguagem de programação, ou ao arquivo executável deste código fonte.

(122) Exercício - Avalie Alo mundo opção 1

7 O que é um programa?
7.5 Exercício - Avalie Alo mundo opção 1

Exercício:
Por que na estrutura do pseudocódigo não aparece o bloco de declaração de variáveis?

--
Resposta:
Por que não há variáveis a declarar. Como o programa apenas mostra na tela uma string (sequência de caracteres entre aspas), nenhuma variável foi necessária para esta ação.

(123) Exercício - Avalie Alo mundo opção 2

7 O que é um programa?
7.6 Exercício - Avalie Alo mundo opção 2

Exercício:
Na estrutura da linguagem C, por que surgiu um elemento desconhecido logo no início do programa (#include <stdio.h>)?

--
Resposta:
Por que um programa em C não tem todas as funções embutidas previamente, assim como o printf, e por isso precisa que seja incluída esta informação (include) que está dentro de um arquivo chamado biblioteca (stdio.h), juntamente com algumas outras funções semelhantes.

(124) Exercício – Saída

7 O que é um programa?
7.7 Exercício – Saída

Na função escrever (printf), o especificador de formato %s pode ser usado para escrever um caractere em letras maiúsculas.

Verdadeiro.
Falso. (correta)

Explicação: O especificado de formato diz ao compilador que a informação fornecida é uma string de caracteres.

(125) Exercício - Tipos de operadores aritméticos

8 Dados, variáveis e operadores
8.1 Exercício - Tipos de operadores aritméticos

Na classe de “Operadores Aritméticos” temos alguns tipos especiais. Identifique-os abaixo na 1ª coluna, e relacione-o com o exemplo que melhor o representa na 2ª coluna (obs.: operadores baseados no VisuALG). Em seguida responda a única alternativa que corresponde à combinação correta.

potenciação (1)
radiciação (2)
resto da divisão (3)
quociente da divisão (4)

(a) Ex.: raiz quadrada de 9, ou raizq(9)
(b) Ex.: 27 mod 5
(c) Ex.: 2 elevado a 3, ou 2^3
(d) Ex.: 9 div 4

a) 1-a, 2-c, 3-b, 4-d.
b) 1-c, 2-a, 3-b, 4-d.
c) 1-c, 2-a, 3-d, 4-b.
d) 1-c, 2-a, 3-b, 4-d. (correta)

(126) Exercício - Dados do tipo inteiro

8 Dados, variáveis e operadores
8.2 Exercício - Dados do tipo inteiro

Exercício:
Marque os dados do tipo inteiro:

1000 (correta)
"0"
"-900"
VERDADEIRO
-456 (correta)
34 (correta)
"Casa 8"
0 (correta)
FALSO
-1.56

(127) Exercício - Operadores - Soma de dois números

8 Dados, variáveis e operadores
8.3 Exercício - Operadores - Soma de dois números

Exercício:

Faça um programa que solicite dois números inteiros ao usuário, some os dois valores e armazene o resultado em uma terceira variável. Ao final, o programa deve exibir a operação que foi realizada, os valores entrados e o resultado desta operação.

Respostas:

Pseudocódigo:

```
inicio
   variavel real primeiro_num, segundo_num, x
   escrever "Digite o primeiro numero: "
   ler primeiro_num
   escrever "Digite o segundo numero: "
   ler segundo_num
   x <- primeiro_num + segundo_num
   escrever "\nA soma eh: "
   escrever x
fim
```

Linguagem C:

```
#include <stdio.h>
int main(){
   float primeiro_num, segundo_num, x;
   printf("Digite o primeiro numero: ");
   scanf("%f",&primeiro_num);
   printf("Digite o segundo numero: ");
   scanf("%f",&segundo_num);
   x = primeiro_num + segundo_num;
   printf("\nA soma eh: ");
   printf("%f",x);
}
```

(128) Exercício - (aritméticos) área do retângulo

8 Dados, variáveis e operadores
8.4 Exercício - (aritméticos) área do retângulo

Exercício:

Sabendo-se que a área de um retângulo é a medida do comprimento multiplicada pela sua largura, desenvolva um programa em que o usuário informe as medidas necessárias, e retorne o valor da área do retângulo.

--
Resposta:

Pseudocódigo:

```
inicio
  inteiro comprim, largura, area
  escrever "\nEntre o comprimento do retangulo: "
  ler comprim
  escrever "\nEntre a largura do retangulo: "
  ler largura
  area <- comprim * largura
  escrever "\nArea do retangulo: ", area
fim
```

Linguagem C:

```
#include <stdio.h>
#include <conio.h>
  int main() {
  int comprim, largura, area;
  printf("\nEntre o comprimento do retangulo: ");
  scanf("%d", &comprim);
  printf("\nEntre a largura do retangulo: ");
  scanf("%d", &largura);
  area = comprim * largura;
  printf("\nArea do retangulo: %d", area);
  return (0);
}
```

Resultado na tela:

Entre o comprimento do retangulo: 4

Entre a largura do retangulo: 3

Area do retangulo: 12

(129) Exercício - (aritméticos) área do triangulo

8 Dados, variáveis e operadores
8.5 Exercício – (aritméticos) área do triangulo

Informe qual foi a fórmula utilizada para desenvolver a principal função do programa a seguir:

Pseudocódigo:

```
inicio
 inteiro a, b
 real resp
 escrever "Entre com os valores de a e b\n"
 ler a, b
 resp <- 0.5 * a * b
 escrever "O resultado eh ",resp
fim
```

Linguagem C:

```
#include <stdio.h>
void main()
{
 int a, b;
 float resp;
 printf("Entre com os valores de a e b\n");
 scanf("%d %d",&a, &b);
 resp = 0.5 * a * b;
 printf("O resultado eh %f",resp);
}
```

Assinale a alternativa que apresenta a fórmula mais corrreta:
() resp=(b*a)/2 (correta)
() resp = b*a
() 0.5*b*a
() "resultado %f", resp

(130) Exercício - (aritméticos) volume do cilindro

8 Dados, variáveis e operadores
8.6 Exercício - (aritméticos) volume do cilindro

Analisando o programa a seguir:

```
#include <stdio.h>
void main()
{
  float vol,pie=3.14;
  float r,h;
  printf("ENTER THE VALUE OF RADIOUS :- ");
  scanf("%f",&r);
  printf("ENTER THE VALUE OF HEIGHT :- ");
  scanf("%f",&h);
  vol = pie * r * r * h;
  printf("ANSWER:- %3.2f ",vol);
}
```

Informe qual é a função deste programa:

() Cálculo do volume de um cilindro.
() Cálculo da área de uma circunferência.
() Cálculo do valor de ANSWER.
() Não calcula nada, apenas mostra "ANSWER:- %3.2f " ao final.

(131) Exercício - (aritméticos) volume do cubo

8 Dados, variáveis e operadores
8.7 Exercício - (aritméticos) volume do cubo

Olhe bem para o programa a seguir, que calcula a área e volume de um cubo. Ele contém alguns erros:

```
#include <stdio.h>
void main()
{
```

```
  float a, b
  float surface_area,volume;
  printf("Entre com o valor do lado do cubo: ");
  scanf("%d",&a);
  surface_area = 6 * (a * a * a);
  volume = a * a * a
  printf("A area da superfície do cubo eh: %.3f",srf_area);
  printf("\nO volume do cubo eh: %.3f",volume);
}
```

Agora responda: dentre as alternativas abaixo, quais as que são realmente erros:

() falta de ";". (correta)
() Cálculo incorreto da área. (correta)
() cálculo incorreto do volume.
() Variável não declarada.
() Variável escrita incorretamente. (correta)
() Variável declarada e não usada.
() código de formatação errado. (correta)

(132) Exercício - Operadores - condição existência triângulo

8 Dados, variáveis e operadores
8.8 Exercício - Operadores - condição existência triângulo

O algoritmo abaixo testa a condição de existência do triângulo, sabendo que cada um dos lados é menor que a soma dos outros dois. O usuário deve entrar com três valores, referentes às medidas de cada lado. Sendo assim, complete as lacunas abaixo de modo que funcione, e a seguir responda a alternativa correta.

```
inicio
   variável real lado1, lado2, lado3
   escrever("Digite os valores dos 3 lados.\n")
   _____ lado1
   _____ lado2
   ler _____
   se (lado1 + lado2 > _____) e (_____ + lado3 > lado1) e (lado1
+ lado3 > lado2) entao
      escrever("Triangulo possivel.")
   senao
      escrever ("_________________")
   fim _____
fim
```

ler, ler, lado3, lado2, Triangulo impossivel, se. (correta)
escrever, escrever, lado3, lado2, Triangulo impossivel, se.
ler, ler, lado2, lado3, Triangulo impossivel, se.
escrever, escrever, lado3, lado2, Triangulo possivel, se.
ler, ler, lado3, lado2, Triangulo impossivel, senao.

(133) Exercício - salário bruto-líquido

8 Dados, variáveis e operadores
8.9 Exercício - salário bruto-líquido

Um analista desenvolveu um algoritmo para ajudar a calcular o salário líquido dos colaboradores da empresa em que trabalha. As regras (fictícias) para o cálculo serão as seguintes:
1) salário líquido = salário bruto - %IR - %INSS
2) salário bruto menor que 1,55 mil, sem desconto de IR e 8% de INSS.
3) salário a partir de 1,55 mil, desconta 7,5% de IR e 9% de INSS.
Ao final deverá ser exibido o salário líquido a ser pago ao colaborador.

```
inicio
   ____ bruto, liquido, inss, ir
   _________ "Digite o salario\n"
   ___ bruto
   se _____ < 1550 entao
      inss <- _____ * 0.08
      ir <- ___
   senao
      ____ <- bruto * 0.09
      ir <- bruto * ____
   fimse
   liquido <- bruto - ____ - ___
   ________ "O valor do salário líquido eh: ", _______
fim
```

--
Respostas (na sequência):

real
escrever
ler
bruto
bruto
0
inss
0.075
inss
ir
escrever
liquido

(134) Exercício - Torre de Hanói

8 Dados, variáveis e operadores
8.10 Exercício - Torre de Hanói

No jogo “Torre de Hanoi” podemos nos basear em um algoritmo para resolver o quebra-cabeça. Além disso, por ter movimentos lógicos e previsíveis, é possível calcular a quantidade mínima de movimentos dos N discos entre suas três hastes necessários para sua solução.

Observe o pseudocódigo abaixo, e assinale a alternativa que responde corretamente à pergunta: o que faz este programa?

```
inicio
   inteiro n
   real x
   escrever " Informe o numero de discos: "
   ler n
   se n>=3 entao
      x<-(2^n-1)
      escrever "O numero de movimentos sera: ",x
   senao
      escrever "O numero de discos é insuficiente"
   fimse
fim
```

()Calcula a quantidade de movimentos para solução, em função do número de hastes informado pelo usuário. (correta)
() Calcula a quantidade de discos necessários para movimentar as 3 hastes existentes.
() Calcula a quantidade de movimentos de 3 discos entre as 3 hastes existentes.
() Calcula a quantidade de movimentos para solução, em função do número de discos informado pelo usuário.

(135) Exercício - Condição - nota maior igual a 7

9 Estrutura de controle - Decisão
9.1 Exercício - Condição - nota maior ou igual a 7

Desenvolva um programa que solicita que o usuário preencha

suas duas notas, tire a média das duas e, caso a média seja maior ou igual a 6, apresente uma mensagem parabenizando-o pela aprovação.

--
Resposta:

```
inicio
   real nota1, nota2, media
   ler nota1, nota2
   media <- (nota1 + nota2)/2
   se media >= 7 entao
      escrever "Parabens! Aprovado!"
   senao
      escrever "Estudante Reprovado!"
   fim se
fim
```

--

(136) Exercício - numero par impar pos neg nulo

9 Estrutura de controle - Decisão
9.2 Exercício - número par, ímpar, positivo, negativo ou nulo

O programa abaixo solicita um número inteiro ao usuário e diz se ele é par ou ímpar:

```
#include <stdio.h>
main()
{
 int n;
 printf("Entre um numero inteiro\n");
 scanf("%d",&n);
 if ( n%2 == 0 )
   printf("Par\n");
 else
   printf("Impar\n");
 return 0;
}
```

Implemente este mesmo programa para que ele indique se o mesmo número é positivo, negativo ou nulo (zero).

--
Resposta:

```
#include <stdio.h>
main()
{
        int n;
        printf("Entre um numero inteiro\n");
        scanf("%d",&n);
        if ( n%2 == 0 )
                printf("Par\n");
        else
                printf("Impar\n");
        if (n > 0)
                printf("Positivo\n");
        else
                if (n<0)
                        printf("Negativo\n");
                else
                        printf("Nulo\n");
        return 0;
}
```

(137) Exercício - idade_atleta x categoria

9 Estruturas de controle - Decisão
9.3 Exercício - idade_atleta x categoria

Observe o problema abaixo:

Escreva um programa em que o usuário entre com a idade de um atleta e mostre a categoria em que ele se enquadra, de acordo com estes parâmetros:
de 5 a 10 anos: categoria infantil
de 11 a 17 anos: categoria juvenil
de 18 a 30 anos: categoria profissional
acima de 30 anos: categoria sênior

Agora, responda qual destas estruturas de controle seria a mais adequada para desenvolver a função principal deste programa:

() escolha(caso). (correta)
() repita-até (do-while).
() para (for).
() enquanto-faça (while)

(138) Exercício - número é par ou impar

9 Estruturas de controle - Decisão
9.4 Exercício - número é par ou impar

Escreva um algoritmo que leia um número inteiro e diga se ele é par ou ímpar.
Dica: utilize o operador % (resto da divisão inteira).

--
Resposta:

```
#include <stdio.h>
main()
{
 int n;
 printf("Entre um numero inteiro\n");
 scanf("%d",&n);
 if ( n%2 == 0 )
   printf("Par\n");
 else
   printf("Impar\n");
 return 0;
}
```

(139) Exercício - while e do-while

10 Estrutura de controle - Laços (repetição)
10.1 Exercício - while e do-while

Marque duas diferenças entre o enquanto-faça (do-while) e o repita-até (while).

() Na estrutura enquanto-faça (do-while), a ação poderá nunca ser executada. (correta)
() Na estrutura repita-até (while), a ação é executada pelo menos uma vez.
() Na estrutura enquanto-faça (do-while), a condição é verificada no início do laço.
() Na estrutura repita-até (while), a condição é verificada ao final do laço.

(140) Exercício - Mesa de testes 1

10 Estrutura de controle - Laços (repetição)
10.2 Exercício - Mesa de testes 1

Analisando a “mesa de testes” do programa a seguir, qual será o último valor assumido pela variável s ao término do da execução?

```
inicio
   inteiro i,s
   i <- 1
   s <- 0
   enquanto(i<=10) faz
      s <- s+i
      escrever i,"\t",s,"\n"
      i <- i+1
   fim enquanto
fim
```

Mesa de testes:

i	s
1	0
1	1
2	3
3	.
4	.
5	.
6	.
7	.
8	.
9	.
10	.

Marque a resposta correta:

() 36.
() 45.
() 55. (correta)
() 66.

(141) Exercício - Mesa de testes 2

10 Estrutura de controle - Laços (repetição)
10.3 Exercício - Mesa de testes 2

Analise o programa abaixo, e responda a questão a seguir:

```
inicio
   inteiro i,s
   i <- 1
   s <- 0
   enquanto (i<=10) faz
      s <- s+i
      escrever i,"\t",s,"\n"
      i <- i+1
   fim enquanto
fim
```

Mesa de testes:

i	s
1	0
1	1
2	3
3	.
4	.
5	.
6	.
7	.
8	.
9	.
10	.

Se formos converter a estrutura do programa da questão anterior para a estrutura REPITA... ATÉ, de forma a mostrar o mesmo resultado no final, indique a alternativa que representa o algoritmo correto:

() (correta)

```
   i <- 1
   s <- 0
   repete
      s <- s + i
      escrever i , "\t" , s , "\n"
      i <- i + 1
   ate (i > 10)
```

()

```
   i <- 0
   s <- 1
   repete
      s <- s + i
      escrever i , "\t" , s , "\n"
      i <- i + 1
   ate (i > 10)
```

()

```
   i <- 1
   s <- 0
   repete ate (i > 10)
      s <- s + i
      escrever i , "\t" , s , "\n"
      i <- i + 1
   fim repita
```

()

```
   i <- 1
   s <- 0
   repete
      s <- s + i
      i <- i + 1
      escrever i , "\t" , s , "\n"
   ate (i > 10)
```

(142) Exercício - mostra pares de 2 a 20

10 Estrutura de controle - Laços (repetição)
10.4 Exercício - mostra pares de 2 a 20

Abaixo temos uma aplicação de algoritmo com uma estrutura Para (for). Observe as variáveis e responda:

```
inicio
   inteiro a,b,c,d
   escrever "Qual o valor inicial? "
   ler b
   escrever "Qual o valor final? "
   ler c
   escrever "Qual o passo? "
   ler d
   para a de b ate c passo d
      escrever "\n",a
   proximo
fim
```

Que valores devemos inserir para as variáveis b, c e d, respectivamente, para que o resultado na tela seja exatamente uma lista dos números pares de 2 a 20?

() 2, 20, 2. (correta)
() 1, 20, 2.
() 2, 20, 1.
() 1, 20, 1.

(143) Exercício - Matrizes - Compara palavras

11 Vetores e matrizes
11.1 Exercício - Matrizes - Compara palavras

Faça um programa que lê duas palavras do teclado e diz se elas

são iguais ou diferentes. O programa deve dizer ainda se alguma das palavras digitadas é igual a “papagaio”.

Resposta:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
 char p1[30],p2[30];
 //captura palavras
 printf("Informe palavra 1: ");
 gets(p1);
 printf("Informe palavra 2: ");
 gets(p2);
 //verifica se sao iguais
 if(strcmp(p1,p2)==0)
 printf("\nPalavras sao iguais.");
 if(strcmp(p1,"papagaio")==0)
 printf("\nPalavra 1 igual papagaio.");
 if(strcmp(p2,"papagaio")==0)
 printf("\nPalavra 2 igual papagaio.");
 return 0;
}
```

(144) Exercício - Matrizes - Entrada valores

11 Vetores e matrizes
11.2 Exercício - Matrizes - Entrada valores

Desenvolva um algoritmo que recebe 25 valores numéricos inteiros numa matriz 5x5 e mostra estes números.

Resposta:

```
#include <stdio.h>
void main()
{
	int Mat [5][5],i,j;
	for(i=0; i<5; i++)
	{
		for(j=0; j<5; j++)
		{
			printf("Digite um valor inteiro:
");
			scanf("%d",&Mat[i][j]);
			printf("%d",Mat[i][j]);
		}
	}
return 0;
}
```

(145) Exercicio - Matrizes - Palavras em ordem inversa

11 Vetores e matrizes
11.3 Exercício - Matrizes - Palavras em ordem inversa

Faça um programa que lê três palavras do teclado e imprime as três palavras na ordem inversa.

Resposta:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
 int i;
 char palavras[3][30];
 //captura palavras
 for(i=0;i<3;i++){
 printf("Informe palavra %d: ",i+1);
 gets(palavras[i]);
 }
 //EXIBE EM ORDEM INVERSA
 printf("\n::: Palavras em ordem inversa :::\n");
 for(i=2;i>=0;i--)
 printf("%s\n",palavras[i]);
 return 0;
}
```

(146) Exercício - Vetores - Inverte ordem

11 Vetores e matrizes
11.4 Exercício - Vetores - Inverte ordem

Faça um programa que cria um vetor com 5 elementos inteiros, lê 5 números do teclado, armazena os números no vetor e imprime o vetor na ordem inversa.

Resposta:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
 int i, v[5];
 //captura os elementos
 for(i=0;i<5;i++){
 printf("Elemento[%d]= ",i);
 scanf("%d",&v[i]);
 }
 //EXIBIR VALORES ORIGINAIS
 printf("\n::: Valores originais :::\n");
 for(i=0;i<5;i++)
 printf("%d\n",v[i]);
 //EXIBIR VALORES ORIGINAIS
 printf("\n::: Valores na ordem inversa :::\n");
 for(i=4;i>=0;i--)
 printf("%d\n",v[i]);
 return 0;
}
```

MATERIAIS COMPLEMENTARES

(147) Materiais 01.1 Video - Seja bem-vindo

Vídeo de boas vindas:
https://drive.google.com/open?id=0B8wKF3o5SDwnMEItbV9XazhMV1E

(148) Materiais 01.2 Creditos

Equipe de Desenvolvimento

Conteúdo do curso:
Lucio Vasconcelos dos Santos
Ana Klock
Prof. Dra. Isabela Gasparini

Ilustrações:
Vitor Mulazani dos Santos

Coordenação:
Prof. Dra. Isabela Gasparini

Agradecimentos
...
...
...

(149) Materiais 02.1 Papo BJPnet

Da lógica à programação
Sugestão - Papo BJPnet

Você pode aprender muito sobre algoritmos e programação (assim como qualquer outro assunto de seu interesse) ouvindo podcasts disponíveis na internet, como este:
Link: Podcast: Papo BJPnet 27
Fonte: BJPnet (o que é um programa de computador)

(150) Materiais 02.2 Jogo Light-bot

Da lógica à programação
Sugestão - Jogo Light-bot

Jogo Light-bot - para "aquecer" no aprendizado da lógica de programação, neste jogo devem ser criadas sequencias de passos para que o personagem possa resolver um problema, inclusive usando as estruturas que vamos aprender neste curso:

Link: Jogo Light-bot

(151) Materiais 03.3.1 Outros interpretadores de pseudolinguagem

Outras ferramentas de aprendizagem
Sugestão - Outros interpretadores de pseudolinguagem
G-Portugol
MACP Compilador Portugol
VisuAlg
Portugol online
Webportugol
Projeto AMBAP

(152) Materiais 03.3.2 Outros compiladores de linguagem C

Outras ferramentas de aprendizagem
Sugestão - Outros compiladores de linguagem C
Microsoft Visual Studio
Code::Blocks
Codepad.org
Ideone.com
TutorialsPoint
CppDroid (para mobile)

(153) Materiais 04.1.1 Passos para criar algoritmo

Algoritmos
Passos para criação de um algoritmo

Entender o problema;
Determinar os dados de entrada;
Determinar como os dados serão processados;
Determinar os dados de saída;
Construir o algoritmo;
Testar o algoritmo.

(154) Materiais 05.1.1 Narrativa sequencial

Sobre narrativas
Narrativa Sequencial

É a descrição de uma ação que é constituída por um número variável de sequencial (segmentos narrativos com princípio, meio e fim). Como exemplo de narrativa sequencial, vamos descrever de forma simples como fazer um café:

1. Pegar o bule.
2. Colocar o coador de plástico sobre o bule.
3. Colocar o coador de papel no coador de plástico.
4. Colocar pó de café no coador de papel.
5. Colocar água quente no coador de papel.

Observe que as ações foram dispostas de tal modo que não é possível trocar as ordens das frases.

(155) Materiais 05.1.2 Narrativa de selecao

Sobre narrativas
Narrativa de Seleção

Digamos que no algoritmo anterior queiramos detalhar o momento correto de se colocar a água sobre o café. Precisamos fazer então um teste para saber se a água já está fervendo. Se isto for verdadeiro, ou seja, SE a água estiver fervente, ENTÃO neste momento pode-se colocar a água sobre o café.
Pegar o bule.
Colocar o coador de plástico sobre o bule.
Colocar o coador de papel no coador de plástico.
Colocar pó de café sobre o coador de papel.
Se a água estiver fervente, então colocar água sobre o café.
Vamos aprimorar o nosso algoritmo, dando uma certa "inteligência" a ele. Digamos que ou possa acompanhar o processo de aquecimento da água. Vou então repetindo testes ou observações, ATÉ QUE a água esteja fervente. Daí segue então o meu processo (colocar água sobre o café), que agora tem o aspecto de uma narrativa de repetição.

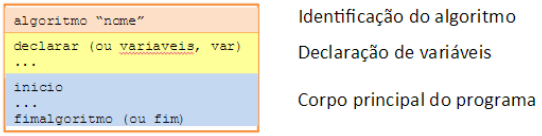
(156) Materiais 05.1.3 Narrativa de repetição

Sobre narrativas
Narrativa de Repetição

Descrição de narrativa de repetição usando a estrutura de repetição. Para não termos que ficar testando indefinidamente a temperatura da água, e repetindo este texto indefinidas vezes no meu algoritmo, posso ESTRUTURAR o meu texto de forma adequada e clara.
Ou seja, escrevo em apenas uma frase a mesma coisa que quis dizer em várias linhas no algoritmo anterior.
Pegar o bule.
Colocar o coador de plástico sobre o bule.
Colocar o coador de papel sobre o coador de plástico.
Colocar pó de café sobre o coador de papel.
Enquanto a água não estiver fervente, faça aquecer a água.
Colocar água sobre o café.

(157) Materiais 06.1.1 Animacao - estrutura do pseudocódigo

Sobre pseudocódigos
Animação - estrutura do pseudocódigo



(158) Materiais 07.1.1 Animacao - Comparativo estrutura pseudo e C

Programa
Comparativo das estruturas básicas de pseudocódigo e C
Observe as estruturas básicas de um programa em pseudocódigo e em linguagem C, lado a lado. As cores definem as funções semelhantes em cada uma.

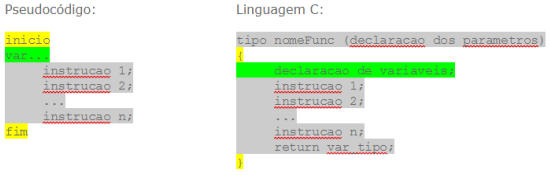


Figura - Comparativo das estruturas básicas de um programa

em pseudocódigo e em C.

(159) Materiais 07.1.2 Alo mundo

Programa
Primeiro programa - Alô mundo!

```
#include <stdio.h>
void main()
{
   printf("Alo mundo!");
}
```

(160) Materiais 07.1.3 Pseudocodigo - Alo mundo

Programa
Primeiro programa em pseudocódigo - Alô mundo!

```
inicio
   escrever "Alo mundo!"
fim
```

(161) Materiais 07.1.4 Teste de mesa

Programa
Teste de Mesa

Após desenvolver um algoritmo, é importante checar seu funcionamento para saber se está se comportando conforme o esperado. O TESTE DE MESA é um recurso preciso e eficiente, que nos permite acompanhar o valor de todas as variáveis envolvidas a cada passo de execução do programa.
Alguns programas compiladores ou interpretadores já possuem o teste de mesa integrado, o que facilita esta análise.
Vamos tomar como exemplo o cálculo da média anual de um estudante, a partir de suas notas (B1 a B4) em cada um dos 4 bimestres.
1. Entrar com a B1.
2. Entrar com a B2.
3. Entrar com a B3.
4. Entrar com a B4.
5. Media = (B1 + B2 + B3 + B4) / 4
6. Mostrar Media.

Teste de mesa:

Passo n	B1	B2	B3	B4	Media
Inicial	0	0	0	0	0
Passo 1	5	0	0	0	0
Passo 2	5	5,5	0	0	0
Passo 3	5	5,5	8	0	0
Passo 4	5	5,5	8	7,5	0
Passo 5	5	5,5	8	7,5	6,5
Passo 6	5	5,5	8	7,5	6,5

(162) Materiais 07.3.1 Pseudocodigo - Comando de atribuição

Programa
Comando de atribuição em pseudocódigos

Em pseudocódigos, por motivos didáticos, o comando de atribuição é representado por uma seta para a esquerda (<-). Ele é utilizado da seguinte forma:
variavel <- valor
Do seu lado esquerdo fica a variável à qual está sendo atribuído o valor, e à sua direita pode-se colocar qualquer expressão (constantes, variáveis, expressões numéricas), desde que seu resultado tenha tipo igual ao da variável.
Observe os exemplos a seguir. Tente compreender a estrutura do comando, mesmo que não entenda alguns termos da expressão (não é necessário resolver os cálculos).

```
x <- 25
y <- x + 15 – 3
z <- y – x + rad(x) – pot(y,2)
```

(163) Materiais 07.4.1 Pseudo x C - Comandos de saída

Programa
Comparativo Pseudocódigo x Linguagem C - Comandos de Saída

Pseudocódigo	Linguagem C
`escrever "Mostrando o resultado."`	`printf("Mostrando o resultado");`

Neste exemplo, "escrever" (ou "printf") é a função, e o argumento é o que está entre aspas "Mostrando o resultado".

(164) Materiais 07.4.2 Tabela de códigos especiais

Programa
Tabela de Códigos Especiais em C

Código	Significado
\n	Nova linha.
\t	tabulação.
\b	Retrocesso (usado para impressora).
\f	Salto de página de formulário.
\a	Beep no alto-falante.
\r	Retorna cursor para o início da linha.
\\	Barra invertida.
\0	Zero.
\'	aspas simples.
\"	Aspas duplas.
\xdd	Representação hexadecimal.
\ddd	Representação octal.

(165) Materiais 07.4.2.1 Pseudocodigo - Funcao de entrada

Programa
Pseudocódigo - Função de entrada

```
inicio
   variavel inteiro x
   escrever "Entre com o valor de x: "
   ler x
   escrever "x = ", x
fim
```

(166) Materiais 07.4.2.2 Funcao de entrada

Programa
Função de entrada

```
#include <stdio.h>
   void main()
   {
      int x;
            printf("Entre com o valor de x: ");
      scanf("%d",&x);
      printf("x = %d",x);
   }
```

(168) Materiais 07.4.2.3 Pseudocodigo - Entrada de dados

Programa
Pseudocódigo - Entrada de dados

```
inicio
```

```
 inteiro a
 escrever "Entre um numero inteiro:\n"
 ler a //recebe um numero inteiro do usuario
 escrever "O numero inteiro que voce digitou eh ",a,".\n"
fim
```

(169) Materiais 07.4.2.4 Entrada de dados

Programa
Pseudocódigo - Entrada de dados

```
#include <stdio.h>
int main()
{
 int a;
 printf("Entre um numero inteiro:\n");
 scanf("%d", &a); //recebe um numero inteiro do usuario
 printf("O numero inteiro que você digitou eh %d.\n", a);
 return 0;
}
```

(170) Materiais 07.4.3 Tabela de codigos de formatação

Programa
Tabela de códigos de formatação

Códigos especiais para printf()	Significado
%c	Caractere simples.
%d	Inteiro decimal com sinal.
%i	Inteiro decimal com sinal.
%e	Notação científica (e minúsculo).
%E	Notação científica (E maiúsculo).
%f	Ponto flutuante em decimal.
%g	Usa %e ou %f, o que for menor.
%G	Usa %E ou %f, o que for menor.
%o	Inteiro octal sem sinal.
%s	String de caracteres.
%u	Inteiro decimal sem sinal.
%x	Inteiro hexadecimal sem sinal (letras minúsculas).
%X	Inteiro hexadecimal sem sinal (letras maiúsculas).
%p	Ponteiro (endereço).
%n	Ponteiro inteiro.
%%	Imprime um caractere%.

(171) Materiais 07.5.1 Tabela das funcoes de string

Programa
Tabela das funções de String

Função	Significado
strcat ( )	Concatena str2 ao final da str1.
strncat ( )	Acrescenta uma parte da string à outra.
strcpy ( )	Copia a str2 dentro da str1.
strncpy ( )	Copia um dado número de caracteres de uma string para outra.
strlen ( )	Informa o comprimento de str1.
strcmp ( )	Retorna 0 se str1 é igual str2. Retorna <0 se str1<str2. Retorna >0 se str1>str2.
strcmpi ( )	Mesmo que strcmp(), mas esta não diferencia maiúsculas e minúsculas.
strchr ( )	Retorna o ponteiro à primeira ocorrência de char em str1.
strrchr ( )	Localiza a última ocorrência de um dado caractere na string.
strstr ( )	Retorna o ponteiro para a primeira ocorrência de str2 em str1.
strrstr ( )	Retorna o ponteiro para a última ocorrência de str2 em str1.
strdup ( )	Duplica a string.
strlwr ( )	Converte a string em minúsculas.
strupr ( )	Converte a string em maiúsculas.
strrev ( )	Reverte a string.
strset ( )	Muda todos os caracteres em uma string para um dado caractere.
strnset ( )	Muda parte dos caracteres em uma string para um dado caractere.
strtok ( )	Sinaliza ("tokeniza") a string dada usando delimitador.

(172) Materiais 08.1.1 Pseudocodigo - Atribuicoes com variáveis

Dados, variáveis e operadores
Pseudocodigo - Atribuicoes com variaveis

Exemplos de atribuições usando variáveis:

```
a <- 3
Valor1 <- 1.5
Valor2 <- Valor1 + a
vet[1] <- vet[1] + (a * 3)
matriz[3,9] <- a/4 - 5
nome_do_estudante <- "José da Silva"
sinalizador <- FALSO
```

```
inicio
\\ Declaracao de variaveis
   variavel real a, valor1, valor2, matriz[4][10]
   variavel texto nome_do_estudante
   variavel logico sinalizador
\\ Calculos
   a <- 3
   Valor1 <- 1.5
   Valor2 <- Valor1 + a
   matriz[3][9] <- a/4 - 5
   nome_do_estudante <- "José da Silva"
   sinalizador <- FALSO
\\ Mostrar resultados
   escrever a
   escrever "\n",Valor1
   escrever "\n",Valor2
   escrever "\n",matriz[3][9]
   escrever "\n",nome_do_estudante
   escrever "\n",sinalizador
fim
```

Observe que variável do tipo lógico mostra o resultado na tela como VERDADEIRO ou FALSO, enquanto que na linguagem C o resultado é 1 ou 0.

(173) Materiais 08.1.2 Atribuicoes com variáveis

Dados, variáveis e operadores
Atribuições com variáveis

Exemplos de atribuições usando variáveis:

```
a <- 3
Valor1 <- 1.5
Valor2 <- Valor1 + a
vet[1] <- vet[1] + (a * 3)
matriz[3,9] <- a/4 - 5
nome_do_estudante <- "José da Silva"
sinalizador <- FALSO
```

Aplicação com atribuições de variáveis:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
//Declaracao de variaveis
   float a, valor1, valor2, matriz[4][10];
   char nome[]="Jose da Silva";
   int sinalizador;
// Calculos
   a = 3;
   valor1 = 1.5;
   valor2 = valor1 + a;
   matriz[3][9] = a/4 - 5;
   sinalizador = a > valor1;
// Mostrar resultados
   printf ("%f", a);
   printf ("\n%f",valor1);
   printf ("\n%f",valor2);
   printf ("\n%f",matriz[3][9]);
   printf ("\n%s",nome);
   printf ("\n%d",sinalizador);
}
```

Observe que variável do tipo lógico mostra o resultado na tela como 1 ou 0, enquanto que no pseudocódigo o resultado é VERDADEIRO ou FALSO.

(174) Materiais 08.1.3 Tabela - Pseudo x C - Tipos de dados

Dados, variáveis e operadores
Tabela - Pseudocódigo x Linguagem C - Tipos de dados

Pseudocódigo	Linguagem C	Descrição	Exemplos
inteiro	int	Os dados do tipo inteiro são todo e qualquer número inteiro, sejam eles negativos ou não.	-1 0 10
real	float, double	Os dados do tipo real são todos aqueles pertencentes ao conjunto dos números reais (todo número inteiro é também um número real). Para representá-los utilizaremos um ponto no lugar da vírgula ().	1 1.1 3/4 2
caracter	char	Os dados do tipo caractere são todos aqueles compostos por um único caracter alfanúmerico. Sempre escreveremos os caracteres dentro de aspas duplas.	'a' 'A' 'b' '\n'
texto	char	Os dados de do tipo texto são todos aqueles compostos por quantos caracteres alfanúmericos desejarmos. Se quisermos um texto que contenha aspas, devemos escapá-las, ou seja, escrever o código \".	"Seja bem vindo(a) ao minicurso! " "Olá, mundo" "123" "a" "" "Um texto com \"aspas\"""
lógico	boolean	Os dados do tipo lógico aceitam apenas dois valores: verdadeiro ou falso (sem fazer a distinção de maiúsculas e minúsculas, ou seja, podemos escrever VERDADEIRO ou verdadeiro e o interpretador irá reconhecer o valor).	VERDADEIRO FALSO

(175) Materiais 08.4.1 Tabela - Operadores relacionais

Dados, variáveis e operadores
Tabela - Operadores relacionais em C

Na tabela abaixo observamos os operadores relacionais em C:

Operador	Função	Exemplo	Resultado
>	maior que	2>3	Falso
<	menor que	2<3	Verdadeiro
>=	maior ou igual a	10>=5+3	Falso
<=	menor ou igual a	-5<=10-15	Verdadeiro
==	igual a	8==6	Falso
!=	diferente de	8!=6	Verdadeiro

(176) Materiais 11.1.1 Declaracao de vetores

Vetores e matrizes
Exemplo - Declaração de vetores

Tipo nome_vetor[tamanho];

Também é possível inicializar o vetor no momento de sua declaração:
Tipo nome_vetor[tamanho]={lista_de_valores};
Exemplo:
int vetor_exemplo[9]={0,1,2,3,4,5,6,7,8,9};

Entrada de dados do vetor pelo usuário:
Exemplo:
scanf (“%d”, &vetor_exemplo[5]);

(177) Materiais 11.2.1 Declaracao de matrizes

Vetores e matrizes
Exemplo - Declaração de matrizes

Exemplo: matriz "B" de 3 linhas por 3 colunas:
Declaração da matriz:
int B[2][2];

inicialização da matriz:
int B[2][2] = {{1,2}, {3,4}};

Acessando a matriz:
B[0][0] = 1;
B[0][1] = 2;
B[1][0] = 3;
B[1][1] = 4;

LINKS DE APOIO

(178) 03.2.1 Links - Aplicativo - DevCpp

http://www.bloodshed.net/devcpp.html

(179) 03.2.2 Links - Download DevCpp 5.4.0

http://sourceforge.net/projects/orwelldevcpp/files/Setup%20Releases/Dev-Cpp%205.4.0%20TDM-GCC%20x64%204.7.1%20Setup.exe/download

(180) 03.3.1 Links - Aplicativo - G-Portugol

http://sourceforge.net/projects/gpt.berlios/

(181) 03.3.2 Links - Aplicativo – MACP

http://portugol.sourceforge.net/

(182) 03.3.3 Links - Aplicativo – VisuAlg

http://www.apoioinformatica.inf.br/produtos/visualg

(183) 03.3.4 Links - Aplicativo - Portugol online

https://www.vivaolinux.com.br/artigo/Portugol-Online-Software-livre-para-facilitar-o-estudo-de-algoritmos

(184) 03.3.5 Links - Aplicativo – Webportugol

http://www.univali.br/webportugol

(185) 03.3.6 Links - Aplicativo – AMBAP

https://sites.google.com/site/ldsicufal/softwares/projeto-ambap

(186) 03.3.7 Links - Aplicativo - Microsoft Visual Cpp

https://www.visualstudio.com/features/cplusplus

(187) 03.3.8 Links - Aplicativo – CppDroid

https://play.google.com/store/apps/details?id=name.antonsmirnov.android.cppdroid&hl=pt_BR

(188) 03.3.9 Links - Aplicativo – CodeBlocks

http://www.codeblocks.org/

(189) 03.3.10 Links - Pagina - codepad.org

http://codepad.org/

(190) 03.3.11 Links - Pagina - ideone.com

https://ideone.com/

(191) 03.3.12 Links - Pagina - tutorialspoint.com

http://www.tutorialspoint.com/compile_embedded_c_online.php

QUESTÕES DA AVALIAÇÃO

(192) Avaliação – QUESTÃO 01 (estrutura do programa)

Observe o programa na figura, e tente visualizar cada parte de sua estrutura (ENTRADA, PROCESSAMENTO, SAÍDA):

Pseudocódigo:

```
1
2
3   inicio
4       inteiro a, b
5       real resp
6       escrever "Entre com os valores de a e b\n"
7       ler a, b
8       resp <- 0.5 * a * b
9       escrever "O resultado eh ",resp
10  fim
```

Linguagem C:

```
1   #include <stdio.h>
2   void main()
3   {
4     int a, b;
5     float resp;
6    printf("Entre com os valores de a e b\n");
7     scanf("%d %d",&a, &b);
8     resp = 0.5 * a * b;
9     printf("O resultado eh %f",resp);
10  }
```

Qual o número da linha em que o programa realiza a etapa correspondente ao PROCESSAMENTO?

a) Linha 6.
b) Linha 7.
c) Linha 8. (correta)
d) Linha 9.

Justificativa:
A operação principal do programa é o cálculo de uma determinada fórmula. Verifica-se que tem uma ENTRADA de dados, pelo usuário (linhas 6 e 7), e a SAÍDA é a apresentação do resultado deste cálculo (linha 9). Portanto, o PROCESSAMENTO corresponde ao cálculo da fórmula (linha 8).

(193) Avaliação – QUESTÃO 02 (condição)

O que faz este programa?

Pseudocódigo:

```
inicio
    inteiro i , valorpago <- 100 , valorproduto <- 12 , troco , quantidade
    escrever "Valor pago = " , valorpago , " , Valor produto = " , valorproduto
    troco <- valorpago - valorproduto
    escrever "\nTroco = " , troco
    se ( troco >= 50 ) entao
        quantidade <- troco / 50
        troco <- troco % 50
        escrever "\nR$ 50 x " , quantidade
    fimse
    se ( troco >= 20 ) entao
        quantidade <- troco / 20
        troco <- troco % 20
        escrever "\nR$ 20 x " , quantidade
    fimse
    se ( troco >= 10 ) entao
        quantidade <- troco / 10
        troco <- troco % 10
        escrever "\nR$ 10 x " , quantidade
    fimse
    se ( troco >= 5 ) entao
        quantidade <- troco / 5
        troco <- troco % 5
        escrever "\nR$ 5 x " , quantidade
    fimse
    se ( troco >= 2 ) entao
        quantidade <- troco / 2
        troco <- troco % 2
        escrever "\nR$ 2 x " , quantidade
    fimse
    se ( troco >= 1 ) entao
        quantidade <- troco / 1
        escrever "\nR$ 1 x " , quantidade
    fimse
    escrever "\n\n"
fim
```

Linguagem C:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int i, valorPago=100, valorProduto=12, troco, quantidade;
    printf("Valor pago = %d, Valor produto = %d",valorPago, valorProduto);
    troco=valorPago-valorProduto;
    printf("\nTroco = %d", troco);
    if (troco >= 50){
        quantidade = troco/50;
        troco = troco % 50;
        printf ("\nR$ 50 x %d", quantidade);
    }
    if (troco >= 20){
        quantidade = troco/20;
        troco = troco % 20;
        printf ("\nR$ 20 x %d", quantidade);
    }
    if (troco >= 10){
        quantidade = troco/10;
        troco = troco % 10;
        printf("\nR$ 10 x %d", quantidade);
    }
    if (troco >= 5){
        quantidade = troco/5;
        troco = troco % 5;
        printf ("\nR$ 5 x %d", quantidade);
    }
    if (troco >= 2){
        quantidade = troco/2;
        troco = troco % 2;
        printf ("\nR$ 2 x %d", quantidade);
    }
    if (troco >= 1){
        quantidade = troco/1;
        printf ("\nR$ 1 x %d", quantidade);
    }
printf("\n\n");
system ("PAUSE");
return 0;
}
```

a) Calcula o troco, baseado em um valor pago informado pelo usuário, e informa quantas e quais cédulas ou moedas serão necessárias.
b) Calcula o troco, baseado em um valor pago pré-declarado (pré-existente) no programa, e informa quantas e quais cédulas ou moedas serão necessárias. (correta)
c) Calcula o troco, baseado nas cédulas ou moedas que o usuário informou, e mostra o valor pago.
d) Calcula o troco, baseado nas cédulas ou moedas pré-declaradas (pré-existentes) no programa, e mostra o valor pago.

Justificativa:
O programa auxilia a selecionar as cédulas e moedas em uma atividade de troco. A estratégia usada é baseada no cálculo do resto de uma divisão (operador %): iniciando pela cédula de maior valor, divide-se o valor desta cédula pelo valor do troco.

Se der um número exato, apenas verifica-se a quantidade de cédulas necessárias (divisão simples). Se sobrar resto, este valor será o novo troco, e compara-se da mesma forma com a cédula seguinte. No início do programa é feito o cálculo do troco, subtraindo o valor do produto do valor pago. Estes valores (valorproduto e valorpago) já estão declarados no código, não há entrada de dados pelo usuário. As cédulas e moedas também já estão estabelecidas no código (cédulas de 50, 20, 10, 5 e 2 Reais, e moeda de 1 real).

(194) Avaliação – QUESTÃO 03 (algoritmo/narrativa)

Um estudante tinha uma calculadora simples, de quatro operações [+] [-] [*] [/] (soma, subtração, multiplicação e divisão), mas as teclas de [*] e [/] quebraram. Como desafio, ele pensou em um algoritmo (em forma de narrativa) que fosse capaz de calcular as operações de [*] e [/] utilizando apenas somas ou subtrações. Dentre as alternativas, qual a única que descreve a narrativa POSSÍVEL que o estudante pensou? Observação: para simplificar, considere para a subtração apenas números positivos, e para a divisão apenas números maiores ou iguais a zero.

a) Algoritmo 1:

```
Entrar com o primeiro valor.
Entrar com o segundo valor.
Entrar com a operação.
Se a operação for soma, então
        primeiro valor [+] segundo valor.
        Mostrar o resultado da operação.
Se a operação for subtração, então
        primeiro valor [-] segundo valor.
        Mostrar o resultado da operação.
Se a operação for multiplicação, então
        primeiro valor [*] segundo valor.
        Mostrar o resultado da operação.
Se a operação for divisão, então
        Primeiro valor [/] segundo valor.
        Mostrar o resultado da operação
```

b) Algoritmo 2:

```
Entrar com o primeiro valor.
Entrar com o segundo valor.
Entrar com a operação.
Se a operação for soma, então
        primeiro valor [+] segundo valor.
        Mostrar o resultado da operação.
Se a operação for subtração, então
        primeiro valor [-] segundo valor.
        Mostrar o resultado da operação.
Se a operação for multiplicação, então
        primeiro valor [*] segundo valor.
        Mostrar o resultado da operação.
Se a operação for divisão, então
        Se o segundo valor for zero
                Informar erro ao usuário e
parar programa.
        Senão
                Primeiro valor [/] segundo valor.
                Mostrar o resultado da operação.
```

c) Algoritmo 3:

```
Entrar com o primeiro valor.
Entrar com o segundo valor.
Entrar com a operação.
Se a operação for soma, então
        primeiro valor [+] segundo valor.
        Mostrar o resultado da operação.
Se a operação for subtração, então
        primeiro valor [-] segundo valor.
        Mostrar o resultado da operação.
Se a operação for multiplicação, então
        Repetir
                primeiro valor [+] ele mesmo, pela
                quantidade de vezes representada pelo
                        segundo valor.
                Mostrar o resultado da operação.
Se a operação for divisão, então
        Repetir
                primeiro valor [-] ele mesmo, pela
                quantidade de vezes representada pelo
                segundo valor.
        Mostrar o resultado da operação.
```

d) Algoritmo 4: (correta)

```
        Entrar com o primeiro valor.
Entrar com o segundo valor.
Entrar com a operação.
Se a operação for soma, então
        primeiro valor [+] segundo valor.
        Mostrar o resultado da operação.
Se a operação for subtração, então
        primeiro valor [-] segundo valor.
        Mostrar o resultado da operação.
Se a operação for multiplicação, então
        Repetir
                primeiro valor [+] ele mesmo, pela
quantidade
        de vezes representada pelo segundo valor.
        Mostrar o resultado da operação.
Se a operação for divisão, então
                Se o segundo valor for zero
                Informar erro ao usuário e parar
programa.
        Senão
                Repetir
                        Segundo valor [-] primeiro
valor
                        Até que o resultado da
subtração seja menor                    que o
segundo valor.
                Contar o número de subtrações que
foram necessárias,
                e mostrar (este será o resultado da
divisão).
                Mostrar o resultado da última subtração
(será o resto).
```

Justificativa:
- Para a multiplicação, são feitas somas repetidas do primeiro valor, pela quantidade de vezes do segundo valor. Por exemplo: 4 * 7 = 28, soma-se o número 4 por sete vezes, ou 4 + 4 + 4 + 4 + 4 + 4 + 4. Vale a comutativa, ou seja, soma-se 4 vezes o número 7, ou 7 + 7 + 7 + 7.

(195) Avaliação – QUESTÃO 04 (compreensão do código)

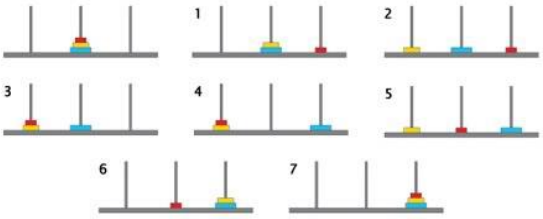
No jogo "Torre de Hanoi" podemos nos basear em um algoritmo para resolver o quebra-cabeça. Além disso, por ter movimentos lógicos e previsíveis, é possível calcular a quantidade mínima de movimentos necessários para sua solução. O código apresentado na figura foi desenvolvido justamente para realizar este cálculo.

Pseudocódigo:

```
inicio
    inteiro n, x
    escrever " Informe o numero de discos: "
    ler n
    se n>=3 entao
        x<-(2^n-1)
        escrever "O numero de movimentos sera: ",x
    senao
        escrever "O numero de discos é insuficiente"
    fimse
fim
```

Linguagem C:

```
#include <stdio.h>
void main()
{
    int n, x;
    printf("Informe o numero de discos: ");
    scanf("%d", &n);
    if (n>=3) {
        x = (pow(2,n)-1);
        printf("O numero de movimentos sera: %d",x);
    }
    else{
        printf("O numero de discos é insuficiente");
    }
return 0;
}
```



Avalie o seu funcionamento, e responda as afirmativas a seguir com Verdadeiro ou Falso.
a) A quantidade de movimentos é calculada em função do número de discos. (verdadeiro)
b) A variável x corresponde ao número de discos. (falso)
c) O cálculo só é realizado a partir de 3 discos. (verdadeiro)
d) A fórmula para calcular o número de movimentos é n>=3. (falso)
e) O usuário informa o número de movimentos, e o programa calcula o número de discos. (falso)
f) A variável n representa a informação fornecida pelo usuário. (verdadeiro)

Justificativa:
O jogo Torre de Hanoi tem, invariavelmente, 3 hastes, só se pode movimentar 1 disco por vez, e um disco menor nunca pode ficar sob um disco maior. Após um certo treino, pode-se prever os movimentos porque se comportam de forma lógica. A quantidade de movimentos aumenta se forem adicionados mais discos, e nota-se um valor exato e mínimo de movimentos até a solução, em função do número de discos. Trata-se de uma relação de Progressão Geométrica (PG), que pode ser calculada pela fórmula x=2^n+1 , onde x é o número de movimentos e n é o número de discos. Avaliando o jogo, não faz sentido realizar o cálculo com menos de 3 discos (daí a restrição no código).

(196) Avaliação – QUESTÃO 05 (estruturas de controle)

Considere um programa em que o usuário entre com a idade de um atleta e mostre a categoria em que ele se enquadra, de acordo com estes parâmetros:
de 5 a 10 anos: categoria infantil
de 11 a 17 anos: categoria juvenil
de 18 a 30 anos: categoria profissional
acima de 30 anos: categoria sênior

Agora responda, qual destas estruturas de controle seria a mais adequada para desenvolver a função principal deste programa?
a) repetir-ate (do-while)
b) se (if) (correta)
c) para (for)
d) enquanto-faz (while)

Justificativa:
Não faz sentido utilizar estruturas de repetição neste problema. No entanto, uma estrutura de condição (if, ou se) resolve bem: se idade é x, então categoria é y.

(197) Avaliação – QUESTÃO 06 (repetição)

Abaixo temos uma aplicação de algoritmo com uma estrutura Para (for). Observe as variáveis e responda:

Pseudocódigo:

```
inicio
     inteiro a,b,c,d
     escrever "Qual o valor inicial? "
     ler b
     escrever "Qual o valor final? "
     ler c
     escrever "Qual o passo? "
     ler d
     para a de b ate c passo d
          escrever "\n",a
     proximo
fim
```

Linguagem C:

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a, b, c, d;
    printf("Qual o valor inicial? ");
    scanf("%d", &b);
    printf("Qual o valor final? ");
    scanf("%d", &c);
    printf("Qual o passo?" );
    scanf("%d", &d);
    for(a=b; a<=c; a+=d){
        printf("%d\n",a);
    }
system ("PAUSE");
return 0;
}
```

Que valores devemos inserir para as variáveis b, c e d, respectivamente, para que o resultado na tela seja exatamente uma lista dos números pares de 2 a 20?
a) 2, 20, 2. (correta)
b) 1, 20, 2.
c) 2, 20, 1.
d) 1, 20, 1.

Justificativa:
Quando o usuário entra com os valores de b, c e d, estamos determinando os parâmetros do laço para (for). Então o comando ficará assim:
Pseudocódigo:
para a de b até c passo d
Linguagem C:
for (a = b; a <=c ; a+=d)
O valor que aparece na tela é o da variável a. Se queremos que a assuma os valores pares de 2 a 20, então então ele deve iniciar a primeira iteração valendo 2, e a cada nova iteração ser acrescido de 2. O laço só irá ser executado enquanto o a for menor ou igual a 20.
Em resumo: o laço a inicia em b = 2, faz iterações enquanto a for menor ou igual a c = 20, e a cada nova iteração é acrescido

de d = 2.

(198) Avaliação – QUESTÃO 07 (dados, variáveis, operadores)

Um analista desenvolveu um algoritmo para ajudar a calcular o salário líquido dos colaboradores da empresa em que trabalha. Deverá ser digitado o salário bruto, e ao final deverá ser exibido o salário líquido a ser pago ao colaborador. As regras (fictícias) para o cálculo serão as seguintes:
1) salário bruto menor que 1550, não tem desconto de IR, mas desconta 8% de INSS.
2) salário a partir de 1550, desconta 7,5% de IR, e 9% de INSS.
3) salário líquido = salário bruto - IR - INSS

Observe as lacunas na figura, identificadas por "____":

Pseudocódigo:

```
inicio
    real bruto, liquido, inss, ir
    escrever "Digite o salario ____: \n"
    ler bruto
    se ____ < 1550 entao
        inss <- ____ * 0.08
        ir <- ____
    senao
        ____ <- bruto * ____
        ir <- bruto * ____
    fimse
    liquido <- bruto - ir - inss
    escrever "Valor a ser pago ao colaborador: ", ____
fim
```

Linguagem C:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    float bruto, liquido, inss, ir;
    printf ("Digite o salario ____: \n");
    scanf ("%f", &bruto);
    if (____ < 1550) {
        inss = ____ * 0.08;
        ir = ____;
    }
    else{
        ____ = bruto * ____;
        ir = bruto * ____;
    }
    liquido = bruto - ir - inss;
    printf ("Valor a ser pago ao colaborador: %f", ____);
}
```

Identifique a única alternativa em que TODAS as opções preenchem corretamente as lacunas no programa:
a) bruto, bruto, bruto, 0, inss, 0.09, 0.075, liquido. (correta)
b) bruto, liquido, bruto, 0.075, ir, 0.09, 0.075, bruto.
c) bruto, bruto, liquido, 0.09, inss, 0.075, 0.09, bruto.
d) liquido, liquido, liquido, 0, ir, 0.075, 0.09, liquido.

Justificativa:
- bruto: o valor a ser entrado é o do salário bruto, conforme a linha seguinte do programa.
- bruto: o valor que é comparado a 1550 é o bruto.
- bruto: o valor de INSS para salário abaixo de 1550 é calculado sobre o valor bruto.
- 0: quando o salário é menor que 1550, não há desconto de IR. Por isso seu valor deve ser zero.
- inss: como a linha seguinte calcula o IR, supõe-se que esta calcule o INSS.
- 0.09: se esta linha calcula o INSS, então a alíquota sobre o bruto é de 9%, ou bruto * 0,09.
- 0.075: se esta linha calcula o IR, então a alíquota sobre o bruto é de 7,5%, ou bruto * 0,075.
- liquido: deve ser mostrado na tela o valor líquido, calculado na linha anterior.

(199) Avaliação – QUESTÃO 08 (dados, variáveis, operadores)

O programa da figura testa a condição de existência do triângulo, sabendo que cada um dos lados é menor que a soma dos outros dois. O usuário deve entrar com três valores, referentes às medidas de cada lado. Sendo assim, complete as lacunas abaixo, representadas por "____", de modo que esta regra funcione, e a seguir responda a alternativa correta.

Pseudolinguagem:

```
inicio
    real lado1, lado2, lado3
    escrever("Digite os valores dos 3 lados.\n")
    ler lado1
    ler lado2
    ler lado3
    se (lado1 < lado2 + ____) e (____ < lado1 + lado3)
        e (lado3 < ____ + lado2) entao
        escrever("Triangulo ____.")
    senao
        escrever ("Triangulo ____.")
    fim se
fim
```

Linguagem C:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    float lado1, lado2, lado3;
    printf("Digite os valores dos 3 lados.\n");
    scanf("%f", &lado1);
    scanf("%f", &lado2);
    scanf("%f", &lado3);
    if (lado1 < lado2 + ____ && ____ < lado1 + lado3
    && lado3 < ____ + lado2) {
        printf("Triangulo ____.");
    }
    else{
        printf("Triangulo ____.");
    }
return 0;
}
```

a) lado1, lado3, lado2, impossivel, possivel.
b) lado2, lado1, lado3, possivel, impossivel.
c) lado1, lado2, lado3, impossivel, possivel.
d) lado3, lado2, lado1, possivel, impossivel. (correta)

Justificativa:
- lado3: lado1 deve ser menos que lado2 + lado3.
- lado2: lado2 deve ser menos que lado1 + lado3.
- lado1: lado3 deve ser menos que lado1 + lado2.
- possivel: se todas as condições forem verdadeiras, o triângulo será possível.
- impossivel: se alguma das condições não for verdadeira, o triângulo será impossível.

(200) Avaliação – QUESTÃO 09 (repetição)

Em relação às diferenças entre as estruturas enquanto-faz (do-while) e repete-até (while), verifique se as afirmativas são verdadeiras ou falsas.
I. Na estrutura repete-até (do-while), a ação poderá nunca ser executada.
II. Na estrutura enquanto-faz (while), a ação é executada pelo menos uma vez.
III. Na estrutura repete-até (do-while), a condição é verificada no início do laço.
IV. Na estrutura enquanto-faz (while), a condição é verificada ao final do laço.
Assinale a alternativa correta:
a) As afirmativas I e III são verdadeiras.
b) Todas as afirmativas são falsas. (correta)
c) As afirmativas II e IV são verdadeiras.

d) Todas as afirmativas são verdadeiras.

Justificativa:
Observe as estruturas de cada tipo:
Pseudolinguagem:

repete ... até (condição)	enquanto (condição) faz ... fim enquanto

Linguagem C:

do{
...
} while (condição)
	while (condição){
	...
	}

Independente de ser em pseudolinguagem ou linguagem C, na estrutura da esquerda a condição é testada ao final do bloco, e na estrutura da direita a condição é testada no início.

(201) Avaliação – QUESTÃO 10 (matrizes)

Este programa realiza a soma de 12 valores inseridos pelo usuário, dispostos em uma matriz 3x4. Logo abaixo, observa-se uma representação da ORDEM de preenchimento da matriz (figura a. Original).
Digamos, que, por algum motivo, a ordem de entrada dos dados seja importante, e sabemos que para preencher a matriz de uma forma alternativa (figura b. Alternativa), basta uma pequena alteração no código. Indique a resposta que mostre como deve ser esta alteração.

Pseudocódigo:

```
inicio
    inteiro MAT[3][4], i, j, soma <- 0
    para i de 0 ate 2
        para j de 0 ate 3
            escrever "Digite um valor inteiro para [",i,"][",j,"]: "
            ler MAT[i][j]
            soma <- soma + MAT[i][j]
        proximo
    proximo
    escrever "\nA soma dos valores digitados eh: ",soma,".\n"
fim
```

Linguagem C:

```
#include <stdio.h>
void main()
{
    int MAT[3][4],i,j,soma = 0;
    for(i=0; i<3; i++)
    {
        for(j=0; j<4; j++)
        {
            printf("Digite um valor inteiro para [%d][%d]: ",i,j);
            scanf("%d",&MAT[i][j]);
            soma = soma + MAT[i][j];
        }
    }
    printf("\nA soma dos valores digitados eh: %d.\n",soma);
    system ("PAUSE");
}
```

Figura a. Original

1º	2º	3º	4º
5º	6º	7º	8º
9º	10º	11º	12º

Figura b. Alternativa

1º	4º	7º	10º
2º	5º	8º	11º
3º	6º	9º	12º

a) Na declaração da matriz, trocar i com j (ficará: MAT[4][3]).
b) Trocar um laço de repetição (para / for) com o outro. (correta)
c) Trocar os índices apenas na leitura dos dados do usuário (ler / scanf).
d) Trocar os índices na declaração da Matriz e também na leitura de dados.

Justificativa:
Note que, no programa, um laço está dentro do outro (laços encadeados ou aninhados). Isso significa que a cada iteração do laço mais externo, é executado um ciclo completo de iterações do laço mais interno.

Pseudocódigo original:

```
para i de 0 ate 2
    para j de 0 ate 3
    ...
    Proximo
proximo
```

Linguagem C original:

```
for (i = 0 ; i < 3 ; i++){
    for (j = 0 ; j < 4 ; j++){
    ...
    }
}
```

Se o laço mais externo for a linha i, conforme apresentado na estrutura original, para cada iteração de i, o laço da coluna j irá variar de 0 a 3. Ou seja:
linha i = 0 ---- coluna j = 0
linha i = 0 ---- coluna j = 1
linha i = 0 ---- coluna j = 2
linha i = 0 ---- coluna j = 3
linha i = 1 ---- coluna j = 0
linha i = 1 ---- coluna j = 1
linha i = 1 ---- coluna j = 2
linha i = 1 ---- coluna j = 3
...
Na prática, a matriz será preenchida inicialmente toda a linha 0, depois toda a linha 1, e assim por diante.
Trocar os índices na declaração da matriz não resolveria, pois apenas mudaria a quantidade de elementos de linhas e colunas – não queremos uma matriz de 4 linhas por 3 colunas, e sim o contrário.
Trocar i com j na leitura dos valores (ler/scanf) poderia até funcionar, mas daria erro ao solicitar o elemento MAT[3][0], pois não existe linha 3 (apenas linhas 0, 1 e 2).
Trocando os índices na declaração da matriz e também na leitura dos dados seria uma solução, e até funcionaria, mas cairia no mesmo problema da matriz de 4 linha por 3 colunas.
Porém, se trocarmos um laço com o outro,ou seja, os laços i com j, o preenchimento da matriz será inicialmente todos os elementos da coluna 1, depois os da coluna 2, e assim por diante, que é o que se deseja na forma alternativa de preenchimento da matriz:

Pseudocódigo alternativo:

```
para j de 0 ate 3
    para i de 0 ate 2
    ...
    Proximo
proximo
```

Linguagem C alternativo:

```
for (j = 0 ; j < 4 ; j++){
    for (i = 0 ; i < 3 ; i++){
    ...
    }
}
```

APÊNDICE H **Intervenções da etapa de Implementação**

Intervenção 1 (qui 01/09/16)

Assunto: Início do Minicurso de Algoritmos e Linguagem de Programação - Edição 2016/2

Seja bem-vind@ ao Minicurso de Algoritmos e Programação!

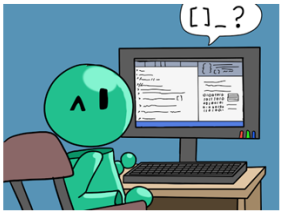
A partir de agora você já pode acessar o ambiente do minicurso através do sistema AdaptWeb.

Para ganhar o **certificado de participação**, você precisa:
- Estudar os conceitos, exemplos, exercícios e materiais complementares que achar importante, acessando o ambiente quando desejar (no período de 01/09 a 31/10);
- Fazer a Avaliação (que estará aberta no período de 01/11 a 11/11);
- Responder o Questionário de Satisfação (que estará aberto no período de 01/11 a 11/11).

Após completado esses itens, você ganhará o certificado de participação com carga horária de 20 horas que poderá ser validado como **Atividade Complementar**. Além disso, teremos um brinde surpresa como um Bônus de participação!

Teremos mensagens periódicas! Durante o curso, você receberá mensagens como esta, para lembrá-l@ de:
- Datas de início e término de atividades.
- Sugestões de estudo para que você não esqueça dos conteúdos abordados, e nem deixe acumular conteúdos para a última hora.
- Propostas de desafios de programação, baseados nos conteúdos sugeridos nas mensagens.

Sugerimos agora que você acesse e explore o ambiente. Aproveite este período inicial para tirar dúvidas e resolver eventuais problemas de acesso.



Obs.: Elaboramos o conteúdo deste minicurso com muito carinho e atenção, mas nem tudo é perfeito!

Caso encontre algum erro, discorde de alguma informação ou tenha sugestões de melhoria, por favor, não hesite em nos informar.
Sua participação é muito importante para nós.

Obrigad@!

Intervenção 2 (dom 04/09/16)

Assunto: Minicurso de Algoritmos e Linguagem de Programação - Semana 1

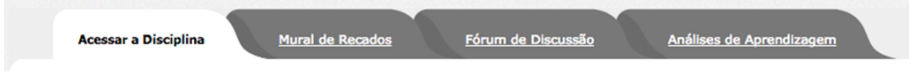
Vamos começar?

Iremos sugerir os conteúdos a serem estudados em cada semana, de forma que você chegue ao final dos 2 meses aprendendo todos os tópicos propostos. Porém, você pode estudar da maneira que desejar! Seguindo nossa sugestão, você poderá se organizar durante este período, se beneficiando também em sua disciplina presencial.

Para esta semana, sugerimos que você estude os seguintes tópicos e seus respectivos sub-tópicos:
- Da lógica à programação;
- Interpretadores e compiladores;
- Algoritmos;
- Sobre narrativas;
- Sobre pseudocódigos.

Acessando o minicurso, você poderá estudar os conceitos, ver os exemplos e resolver os exercícios, além de consultar os materiais complementares.

Aproveite também para explorar as outras opções do sistema:



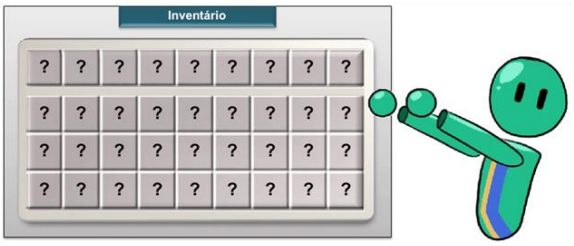
Acessar a Disciplina: você pode estudar todos os tópicos do minicurso
Mural de recados: você pode enviar e visualizar as mensagens recebidas
Fórum de discussão: você pode tirar dúvidas e colaborar com seus colegas
Análises de Aprendizagem: você pode acompanhar o andamento da sua aprendizagem.

Até a próxima!

Intervenção 3 (dom 11/09/16)

Hora de programar!!!

Para aprender Algoritmos, diferentes tópicos devem ser trabalhados. No nosso minicurso, estes tópicos podem ser considerados **Coleções de Conhecimentos** para serem usados mais adiante.



Vamos conquistar estes conhecimentos? Sugerimos então, para esta semana, o estudo do seguinte tópico:

O que é um programa?
- Estrutura de um programa (main, system, include);
- Constantes e comandos de atribuição;
- Comandos de entrada e saída (printf, scanf);
- Strings.

Até a próxima semana!

Intervenção 4 (dom 18/09/16)

Ah, a lógica matemática e o raciocínio abstrato… Uhul!

Vamos entendê-los juntos! (:p) Para esta semana, veja este tópico:

DADOS, VARIÁVEIS E OPERADORES:
- O que são dados e para que servem;
- Como declarar variáveis;
- Operadores aritméticos, relacionais e lógicos.

Estes itens podem ser úteis para o seu aprendizado:

Colocamos uma atividade no Fórum de Discussão! Participe! Você tem até 25/09 para responder. Dia 26/09 daremos a resposta!

Bom estudo!

Intervenção 5 (dom 25/09/16)

Muito bem, vamos colocar um pouco mais de emoção no nosso aprendizado!

Até agora, sugerimos os seguintes itens:

Com estes recursos, acreditamos que você já possa resolver alguns DESAFIOS! Estes desafios estão no Fórum de Discussão! Dia 03/10 daremos as respostas!

Boa sorte!

Intervenção 6 (dom 02/10/16)

Ensinando o computador a tomar decisões!

A partir de agora você pode aprofundar seus conhecimentos ao estudar Estruturas de Controle. Para esta semana, sugerimos o tópico:

Estruturas de Controle - Decisão
(if, if-else, switch-case).

Nossa coleção está assim:

Compartilhe seus conhecimentos com os colegas ou ajude alguém com os assuntos que você domina! Para isso, use o Fórum de Discussão. Aproveite para tirar dúvidas dos assuntos que você tem dificuldade. Afinal, contrariando as leis da Matemática, o conhecimento é a única riqueza que dividimos mas não perdemos nada!

Bons estudos!

Intervenção 7 (dom 09/10/16)

Está ficando cada vez mais interessante!

Deixe o computador trabalhar por você, As estruturas de repetição podem ajudar em diversas tarefas. Durante esta semana, sugerimos que você acesse o seguinte tópico:

Estruturas de Controle - Laços (repetição)
(while, do-while, for)

Veja o que conquistamos até agora:

Tenha uma ótima semana!

Intervenção 8 (dom 16/10/16)

Queimando neurônios...!

Este é um tópico bem interessante! Então, vamos aproveitar ao máximo! :D

Com mais estes conhecimentos, você já tem um vasto conjunto de informações, que podem ajudar a resolver e entender a maioria dos problemas de lógica de programação! Para isso propomos o seguinte tópico:

Vetores e Matrizes
- Vetores
- Matrizes

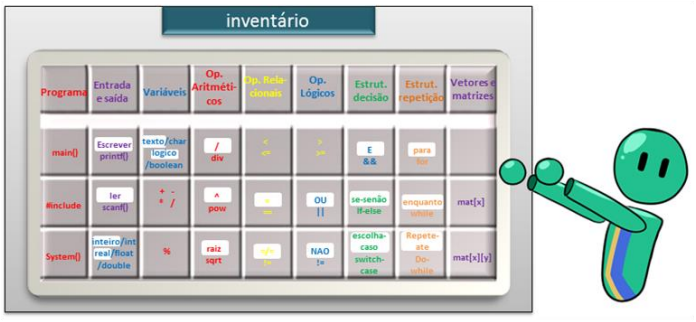
Nosso inventário, neste momento, é este:

Bom estudo!

Intervenção 9 (dom 23/10/16)

Eba! Mais um desafio!!!!

Pode parecer pouco, mas o que você viu neste minicurso fundamenta a maioria das linguagens de programação existentes no mercado!



Propomos agora mais um desafio no Fórum de Discussão! - esperamos que gostem e compartilhem suas dificuldades e soluções com os outros participantes por meio do Fórum. Dia xx/xx daremos a resposta.

Não tenha receio de expor suas ideias, tente colaborar com outros que estão com dificuldade. "Duas cabeças pensam melhor que uma", já dizia o velho ditado. Se você conseguir chegar em uma solução, conte-nos como conseguiu, que estratégia usou.

LEMBRE-SE: dia 31/10 será o último dia para acesso ao ambiente de aulas do minicurso! Aproveite para estudar ou revisar os conteúdos. Tire suas dúvidas pelo Fórum de Discussão e o Mural de Recados. Explore também outros recursos do ambiente AdaptWeb.

Intervenção 10 (dom 30/10/16)

Mas já está acabando????

Amanhã, dia 31/10, é o ultimo dia para acessar o ambiente de aula!

A partir de 01/11 a avaliação (prova) estará liberada juntamente com um questionário de satisfação, e o ambiente de aula estará bloqueado para acesso. Não perca, esta é a última oportunidade de estudar/revisar os conteúdos.

Ah, e ao final tem o CERTIFICADO e a possibilidade de validar as horas deste minicurso como ATIVIDADE COMPLEMENTAR, mas somente para quem realizar a avaliação e responder todo o Questionário de satisfação!

Participem!

Intervenção 11 (seg 31/10/16)

Hoje é o ÚLTIMO DIA para acessar o ambiente...

A partir de amanhã 01/11, a prova e o questionário de satisfação ficarão disponíveis até 11/11 às 23:59.

- Faça a prova e o questionário de satisfação para receber o seu certificado de conclusão.
- Não se esqueça que pode validar como atividade extra.
- Brinde surpresa para quem realizar a prova e o questionário de satisfação!

Esperamos ter ajudado. Agradecemos sua participação!

Intervenção 12 (dom 06/11)

Lembre-se, você só tem mais uns dias para realizar a prova e responder o questionário de satisfação!
Todos que concluírem estes dois processos poderão receber certificado de participação!

Qualquer dúvida ou informação vocês podem entrar em contato com a professora Isabela Gasparini
por email: isabela.gasparini@udesc.br ou diretamente em sua sala: Bloco F, 2o andar, sala 21.

Obrigad@ a tod@s pela participação!

APÊNDICE I **Desafios da etapa de Implementação**

DESAFIO 1:

Qual é a resposta correta para as equações abaixo?

a) 6 / 2 * (1 + 2) = ?
b) 7 + 8 * 0 - 2 = ?
c) 2 + 5 x 3 + 4 = ?
d) 2 + 2 + 2 * 0 = ?
e) 7 + 7 / 7 + 7 * 7 - 7 = ?
f) 12 / 2 * (6 - 7 + 4) * 2 = ?

Por quê?
--
RESPOSTAS:

Para qualquer um destes exercícios, a forma que o computador resolve é a mesma da Matemática: são adotadas as mesmas prioridades entre as operações.

a) 6 / 2 * (1 + 2) = 6 / 2 * (3) = 3 * (3) = 9
Justificativa: resolve-se primeiro o que está entre parênteses, então a fração (6/2), e finalmente a multiplicação.

b) 7 + 8 * 0 - 2 = 7 + 0 – 2 = 5
Justificativa: resolve-se primeiro a multiplicação, depois a soma ou subtração.

c) 2 + 5 x 3 + 4 = 2 + 15 + 4 = 21
Justificativa: idem ao anterior.

d) 2 + 2 + 2 * 0 = 2 + 2 + 0 = 4
Justificativa: idem aos anteriores.

e) 7 + 7 / 7 + 7 * 7 - 7 = 7 + 1 + 7 * 7 – 7 = 7 + 1 + 49 – 7 = 7 + 50 – 7 = 50
Justificativa: primeiro resolve-se a fração, depois multiplicação, e ao final as somas e subtrações.

f) 12 / 2 * (6 - 7 + 4) * 2 = 12 / 2 * (3) * 2 = 6 * (3) * 2 = 36
Justificativa: inicialmente resolve-se o que está entre parênteses, a fração, e depois as multiplicações.

DESAFIO 2:

(Prazo até 02/10) Observe a sequência: 2, 5, 10, 17, 26, ... Escreva um programa que forneça o n-ésimo elemento desta sequência. Sabemos que não existe uma única forma de se resolver, e que cada pessoa pensa de maneira diferente. Compartilhe com os outros participantes, através do fórum de discussões, como você desvendou a sequência, e qual estratégia você usou para resolvê-la.
--
DISCUSSÕES:
Compreensão do problema:
Uma sequência de números geralmente obedece a um determinado padrão. Há uma ordem lógica entre seus termos, de forma que se possa calcular o elemento seguinte.
É comum utilizar-se a variável n para identificar cada termo de uma sequência: “A1” para o primeiro termo ou n = 1, “A2” para o segundo ou n = 2, “A20” para o vigésimo ou n = 20. Daí o termo “n-ésimo” (ou An), que serve referenciar qualquer um dos termos desta sequência.

Estratégia sugerida:
1. Desvendar a sequência.
2. Desenvolver um algoritmo que gere qualquer elemento da sequência.
3. Criar um programa que receba o n do usuário e execute este algoritmo.

1. Desvendando a sequência:
(Opção 1) Uma possibilidade é elevar cada número natural (0, 1, 2, 3, ...) ao quadrado, e a cada vez, adicionar 1.
(Opção 2) Uma segunda forma seria adicionar um número ímpar, na ordem (3, 5, 9, 11, ...), ao termo anterior da sequência.
2. Algoritmo:
(Opção 1)
1. Receber n do usuário.
2. An = (n^2) + 1
3. Mostrar An. (“O termo n da sequência é An”)
(Opção 2)
1. A1 = 2 (primeiro termo da sequência).
2. x = 2 (variável auxiliar, será incrementado de 2 até n).
3. i (número ímpar, calculado: 2 * x - 1).
4. Receber n do usuário.
5. Repetir:
i = 2 * x - 1 (próximo número ímpar).
An = A1 + i (adiciona termo anterior ao número ímpar).
A1 = An (O termo atual An será o novo termo anterior).
x = x + 1 (incrementa x).
até que x > n.
6. Mostrar An. (“O termo n da sequência é An”)

3. Programa:
(Opção 1)

```
inicio
   inteiro n , an
   escrever "Entre com o número do termo: "
   ler n
   an <- n ^ 2 + 1
   escrever an
fim
```

```
#include <stdio.h>
main()
{
        int n, an;
        printf("Entre com o numero do termo: ");
        scanf("%d",&n);
        an=pow(n,2)+1;
        printf("O valor eh %d",an);
}
```

(Opção 2)

```
inicio
   inteiro a1 <- 2 , n , i , an , x <- 2
   escrever "Entre com o número do termo: "
   ler n
   repete
      i <- 2 * x - 1
      an <- a1 + i
      a1 <- an
      x <- x + 1
   ate ( x > n )
   escrever an
fim


#include <stdio.h>
main()
{
        int n, an, i, a1 = 2, x = 2;
        printf("Entre com o numero do termo: ");
        scanf("%d",&n);
        do
        {
                i = (2 * x) - 1;
                an = a1 + i;
                a1 = an;
                x = x + 1;
        }
        while (x > n);
        printf("O valor eh %d",an);
}

--
```

DESAFIO 3:

(Prazo até 02/10) Crie uma calculadora de tempo. Por exemplo, calcular a diferença entre 3:15:33 (3 horas : 15 minutos : 33 segundos) e 17:28:22. A resposta também deve ser em horas:minutos:segundos. Compartilhe através do fórum a sua solução, e qual estratégia usou para escrever o programa.

--
POSSÍVEIS DISCUSSÕES:

Como não foi especificado mais nada além de se calcular simplesmente a diferença entre dois horários, podemos pensar em várias alternativas para resolver. As observações a seguir podem parecer muito óbvias para você, mas pense "friamente" sobre cada uma delas:
Obs.2: no exemplo foram mostrados dois horários, mas não foi especificada nenhuma ordem entre eles;
Obs.3: o primeiro horário mostrado é menor do que o segundo.
Obs.4: não foi dito se o usuário entrará com os horários.

Algumas questões podem surgir:
- Os horários correspondem a horas de um dia (Ex.: 0 horas até 24 horas) ou são horas "corridas" (Ex.: 3 horas, 17 horas, 39 horas, 112 horas)? Resposta: horas corridas. De fato, pode-se pensar que seriam horas do dia, já que foi usada a palavra "horário". Mas, para isso, seria necessário usar ao menos a variável "dia" para possibilitar a resolução, ou então limitar o valor de entrada.

- Poderá se mostrar hora negativa na resposta? Resposta: Não. Geralmente, quando se fala de horas, não usamos valores negativos. Além do mais, o enunciado pede a "diferença" entre dois valores, e não algo como "o primeiro menos o segundo" ou "o maior menos o menor". Assim, vamos considerar que não existirá ordem de entrada dos horários.

- Como será a entrada de dados de cada horário, tudo junto, como no exemplo (no formato hora:minuto:segundo) ou separado (primeiro a hora, depois minuto, e depois segundo)? Resposta: separado. Somente para simplificar o algoritmo. Se formos usar de outra forma, precisaremos tratar a entrada como string, o que complica um pouco o desenvolvimento. O nosso foco aqui não é a estrutura do código, e sim a estratégia da resolução.

Estratégia sugerida:
1. Entender como se faz cálculo com horas;
2. Criar um algoritmo que faça este cálculo com dois horários quaisquer;
3. Escrever um programa em que o usuário entre com dois horários, e o programa mostra na tela a diferença entre eles.

1. Fazendo cálculo com horas:
Estamos usando horas, minutos e segundos.
Começando pelo menor, podemos dizer que 60 segundos é igual a 1 minuto, e que 60 minutos é igual a 1 hora.
Podemos calcular a diferença por, ao menos, 3 formas diferentes: (a) fazer as contas diretamente em horas; (b) converter tudo em decimais da maior unidade, que é a hora, fazer as contas diretas com estes decimais, e ao final voltar para h:m:s; (c) converter tudo para a menor unidade, que é segundos, fazer as contas normalmente, e depois voltar para h:m:s.
Seguindo o exemplo:
(a) Contas diretamente em horas:

Para não dar número negativo, subtraimos o maior do menor:
17:28:22
03:15:33 –
??:??:??

Começamos pelos segundos: como não dá para subtrair 22 de 33, emprestamos 1 min dos 28 min, que vira 27 min, e o 1 min (ou 60 seg) soma com os 22 seg. Então:
17:27:82
03:15:33 –

Agora sim, a conta é possível, e não precisa emprestar mais nada.
17:27:82
03:15:33 –
14:12:49

(b) Convertendo tudo em horas:

3:15:33 = 3 h + 15 min + 33 seg
Se 1 h = 60 min, então 15 min = 0,25 h (por regra de três)
Se 1 h = 60 min * 60 seg = 3600 seg, então 33 seg = 0,0091666...h
Ou seja, 3h + 0,25 h + 0,0091666...h = 3,259166...h

Da mesma forma: 17:28:22 = 17 h + 0,4666...h + 0,006111...h = 17,472777...h

Portanto, a diferença entre 3,259166...h e 17,472777...h é 14,2136111...h

Mas se tirarmos a parte inteira do número (14 h inteiras), teremos 0,2136111...h
ou em minutos: 0,2136111...h * 60 min = 12,81666...min

E se tirarmos a parte inteira deste número (12 min inteiros) teremos 0,81666...min
ou em segundos: 0,81666...min * 60 seg = 49 seg

Juntando as partes inteiras (em destaque):
14 h 12 min 49 s ou 14:12:49

(c) Convertendo tudo em segundos:

3:15:33 = 3 h + 15 min + 33 seg = (3 * 60 * 60) seg + (15 * 60) seg + 33 seg =
= 10800 + 900 + 33 seg = 11733 seg

17:28:22 = 17 h + 28 min + 22 seg = (17 * 60 * 60) seg + (28 * 60) seg + 33 seg =
= 61200 + 1680 + 22 seg = 62902 seg

Subtraindo: (62902 – 11733) seg = 51169 seg

Revertendo:
(51169 / 60 / 60) h = 14,21361111 h ou 14 h inteiras + 0,21361111 h

O restante é igual ao método (b):

Em minutos: 0,2136111...h * 60 min = 12,81666...min

E se tirarmos a parte inteira deste número (12 min inteiros) teremos 0,81666...min
ou em segundos: 0,81666...min * 60 seg = 49 seg

Juntando as partes inteiras (em destaque):
14 h 12 min 49 s ou 14:12:49

2. Criar algoritmo:
(a) Contas direto em horas:
- Receber h1.
- Receber min1.
- Receber seg1.
- Receber h2.
- Receber min2.
- Receber seg2.
- Se (seg1 – seg2) > 0, então segFinal = seg1 – seg2.
- Senão, segFinal = (seg1 + 60) – seg2, e min1 = min1 – 1.
- Se (min1 – min2) > 0, então minFinal = min1 – min2.
- Senão, minFinal = (min1 + 60) – seg2, e h1 = h1 – 1.
- Se (h1 – h2) > 0, então hFinal = h1 – h2.
- Senão, inverter sinal de hFinal.
- Mostrar hFinal + ":" + minFinal + ":" + segFinal.

(b) convertendo tudo em horas:
- Receber h1.
- Receber min1.
- Receber seg1.
- Receber h2.
- Receber min2.

- Receber seg2.
- Horário1 em horas: T1 = h1 + (min1/60) + (seg1/60/60).
- Horário2 em horas: T2 = h2 + (min2/60) + (seg2/60/60).
- Diferença em horas: Dif = T1 – T2 (se for negativo, inverter sinal de Dif).
- Revertendo: HoraFinal = parte inteira de h.
- m = casas decimais de h * 60.
- MinutosFinal = parte inteira de m.
- SegundosFinal = casas decimais de m * 60.
- Mostrar HoraFinal + ":" + MinutoFinal + ":" + SegundosFinal.

(c) convertendo tudo em segundos:

- Receber h1.
- Receber min1.
- Receber seg1.
- Receber h2.
- Receber min2.
- Receber seg2.
- Horário1 em segundos: T1 = (h1*60*60) + (min1*60) + seg1.
- Horário2 em segundos: T2 = (h2*60*60) + (min2*60) + seg2.
- Diferença em segundos: Dif = T1 – T2 (se for negativo, inverter sinal de Dif).
- Revertendo: h = Dif / 60 / 60.
- HoraFinal: parte inteira de h.
- m = casas decimais de h * 60.
- MinutosFinal = parte inteira de m.
- SegundosFinal = casas decimais de m * 60.
- Mostrar HoraFinal + ":" + MinutoFinal + ":" + SegundosFinal.

DESAFIO 4:

Proposta 1:
O que você prefere? Receber 1 milhão de reais de uma só vez ou receber 1 centavo hoje, e a cada dia, durante 30 dias, receber o dobro do dia anterior? Elabore um algoritmo que demonstre qual opção é mais vantajosa.

--
Possível solução da proposta 1:

Se fizer as contas, recebendo em um dia o dobro do valor do dia anterior, chega-se a conclusão que começar com 1 centavo vale mais a pena. Ao final de 30 dias você terá recebido mais de 10 milhões de reais.

--
Proposta 2:
O torneio de Wimbledon é uma competição mundial de tênis que envolve mais de quase 1000 atletas em diversas categorias, e ocorre em um período relativamente curto. Como isso é possível? E se os participantes fossem a população do mundo todo, digamos, 8 bilhões de pessoas, em quantas rodadas o torneio terminaria?

--
Possível solução da proposta 2:
Agrupam-se todos os participantes, 2 a 2, na 1ª rodada. Os ganhadores são agrupados novamente, 2 a 2, na 2ª rodada, e assim por diante. A cada rodada, a metade dos participantes é eliminada. O torneio de Wimbledon tem seu campeão em 10 rodadas. Calculando-se para 8 bilhões de participantes, o torneio terminaria em 33 rodadas.

--
Proposta 3:
Suponha que você tenha 100 amigos no Facebook, e cada um deles tenham mais 100 amigos. Quantas interações/conexões são necessárias para que você tenha ligação com qualquer pessoa no mundo?
--

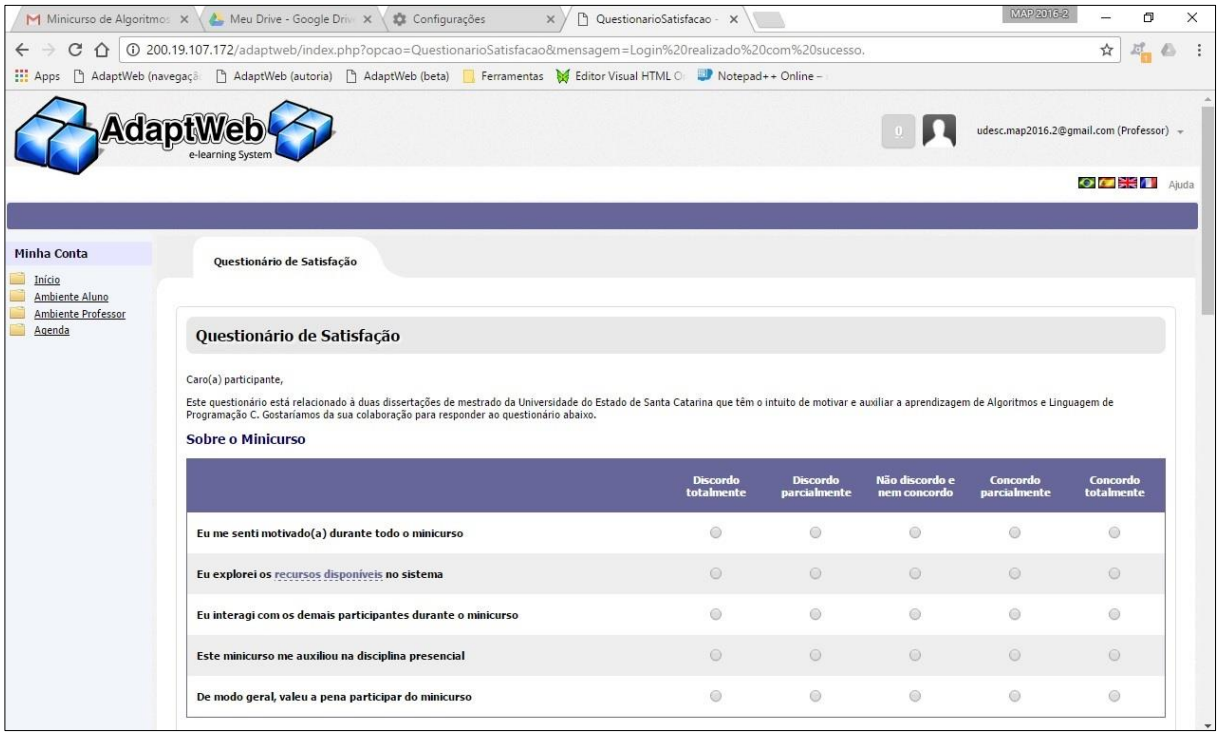
Possível solução da proposta 3:
Você tem 100 amigos. Na 1ª interação, seus amigos tem 100*100 = 10.000 amigos. Os amigos dos amigos têm, na 2ª interação, 100*100*100 = 1.000.000 amigos, e assim por diante. Em 5 interações, já seriam mais de 10 bilhões de pessoas interligadas, sendo que o mundo tem cerca de 8 bilhões de pessoas.

--
Proposta 4:
Pensando o Jogo da velha, entre dois jogadores, como uma matriz 3x3, como identificar que alguém ganhou o jogo?

Possível solução da proposta 4:
No jogo clássico, se um jogador completar uma linha, coluna ou diagonal com a sua marcação, ele ganha. No computador, é necessário analisar, A CADA JOGADA, os elementos da matriz. Podem existir várias estratégias. Duas delas são:
a) Para cada jogador, e a cada jogada, verificar se há a combinação de caracteres iguais na sequência. Por exemplo: verificar se na linha 1 da matriz tem ”X”, “X”, “X”, repete, para linhas 2 e 3. Verificar se na coluna 1 tem ”X”, “X”, “X”, repete, para colunas 2 e 3. As diagonais são duas: verificar, para cada uma delas, se ocorre os mesmos ”X”, “X”, “X”. Depois, fazer o mesmo para “O”, “O”, “O”, e repetir tudo a cada nova jogada.
http://duvidainsana.blogspot.com.br/p/apostilas.html
b) Atribuir +1 para o elemento “X” do jogador1, e -1 para o elemento “O” do jogador2, e 0 para elemento vazio. A cada jogada, se a soma da linha, da coluna ou da diagonal der 3, o jogador1 ganha; se der -3, o jogador2 ganha.
http://www.cprogressivo.net/2013/03/Codigo-jogo-da-velha-em-C.html

APÊNDICE J **Questionário de Satisfação**

Figura 14 – Questionário de satisfação – Tela 1 de 3



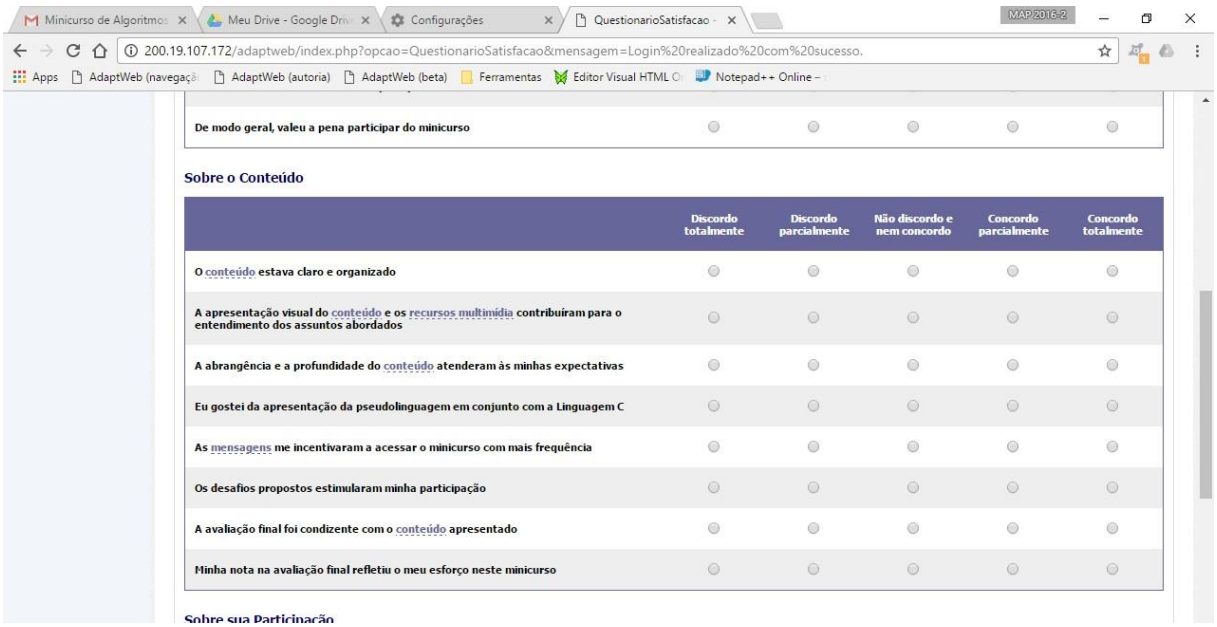
Fonte: elaborada pelo autor, 2016.

Transcrição do texto na Figura 14:

Sobre o minicurso (resposta em escala de Likert, de 1 a 5):

- Eu me senti motivado(a) durante todo o minicurso
- Eu explorei os recursos disponíveis no sistema
- Eu interagi com os demais participantes durante o minicurso
- Este minicurso me auxiliou na disciplina presencial
- De modo geral, valeu a pena participar do minicurso

Figura 15 – Questionário de satisfação – Tela 2 de 3



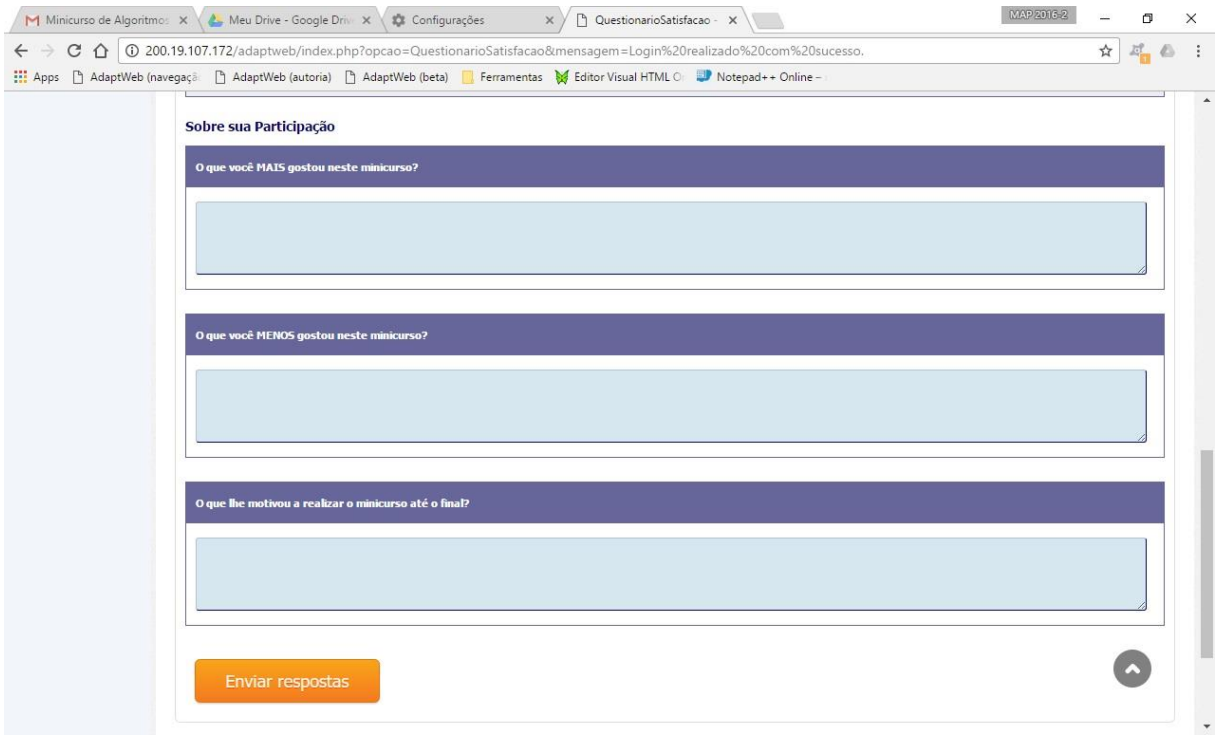
Fonte: elaborada pelo autor, 2016.

Transcrição do texto na Figura 15:

Sobre o conteúdo (resposta em escala de Likert, de 1 a 5):

- O conteúdo estava claro e organizado
- A apresentação visual do conteúdo e os recursos multimídia contribuíram para o entendimento dos assuntos abordados
- A abrangência e profundidade do conteúdo atenderam às minhas expectativas
- Eu gostei da apresentação da pseudolinguagem em conjunto com a Linguagem C
- As mensagens me incentivaram a acessar o minicurso com mais frequência
- Os desafios propostos estimularam minha participação
- A avaliação final foi condizente com o conteúdo apresentado
- Minha nota na avaliação final refletiu o meu esforço neste minicurso

Figura 16 – Questionário de satisfação – Tela 3 de 3



Fonte: elaborada pelo autor, 2016.

Transcrição do texto na Figura 16:

Sobre sua participação:

- O que você MAIS gostou neste minicurso?
- O que você MENOS gostou neste minicurso?
- O que lhe motivou a realizar o minicurso até o final?