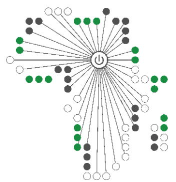


Universidade Virtual Africana

INFORMÁTICA APLICADA: CSI 1304

PRINCÍPIOS DE PROGRAMAÇÃO

Elisabeth Andrade

Prefácio

A Universidade Virtual Africana (AVU) orgulha-se de participar do aumento do acesso à educação nos países africanos através da produção de materiais de aprendizagem de qualidade. Também estamos orgulhosos de contribuir com o conhecimento global, pois nossos Recursos Educacionais Abertos são acessados principalmente de fora do continente africano.

Este módulo foi desenvolvido como parte de um diploma e programa de graduação em Ciências da Computação Aplicada, em colaboração com 18 instituições parceiras africanas de 16 países. Um total de 156 módulos foram desenvolvidos ou traduzidos para garantir disponibilidade em inglês, francês e português. Esses módulos também foram disponibilizados como recursos de educação aberta (OER) em oer.avu.org.

Em nome da Universidade Virtual Africana e nosso patrono, nossas instituições parceiras, o Banco Africano de Desenvolvimento, convido você a usar este módulo em sua instituição, para sua própria educação, compartilhá-lo o mais amplamente possível e participar ativamente da AVU Comunidades de prática de seu interesse. Estamos empenhados em estar na linha de frente do desenvolvimento e compartilhamento de recursos educacionais abertos.

A Universidade Virtual Africana (UVA) é uma Organização Pan-Africana Intergovernamental criada por carta com o mandato de aumentar significativamente o acesso a educação e treinamento superior de qualidade através do uso inovador de tecnologias de comunicação de informação. Uma Carta, que estabelece a UVA como Organização Intergovernamental, foi assinada até agora por dezenove (19) Governos Africanos - Quênia, Senegal, Mauritânia, Mali, Costa do Marfim, Tanzânia, Moçambique, República Democrática do Congo, Benin, Gana, República da Guiné, Burkina Faso, Níger, Sudão do Sul, Sudão, Gâmbia, Guiné-Bissau, Etiópia e Cabo Verde.

As seguintes instituições participaram do Programa de Informática Aplicada: (1) Université d'Abomey Calavi em Benin; (2) Université de Ougagadougou em Burkina Faso; (3) Université Lumière de Bujumbura no Burundi; (4) Universidade de Douala nos Camarões; (5) Universidade de Nouakchott na Mauritânia; (6) Université Gaston Berger no Senegal; (7) Universidade das Ciências, Técnicas e Tecnologias de Bamako no Mali (8) Instituto de Administração e Administração Pública do Gana; (9) Universidade de Ciência e Tecnologia Kwame Nkrumah em Gana; (10) Universidade Kenyatta no Quênia; (11) Universidade Egerton no Quênia; (12) Universidade de Addis Abeba na Etiópia (13) Universidade do Ruanda; (14) Universidade de Dar es Salaam na Tanzânia; (15) Universite Abdou Moumouni de Niamey no Níger; (16) Université Cheikh Anta Diop no Senegal; (17) Universidade Pedagógica em Moçambique; E (18) A Universidade da Gâmbia na Gâmbia.

Bakary Diallo

O Reitor

Universidade Virtual Africana

Créditos de Produção

Autor

Elisabeth Andrade

Par revisor(a)

Arlindo Oliveira da Veiga

UVA - Coordenação Académica

Dr. Marilena Cabral

Coordenador Geral Programa de Informática Aplicada

Prof Tim Mwololo Waema

Coordenador do módulo

Jules Degila

Designers Instrucionais

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Equipa Multimédia

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

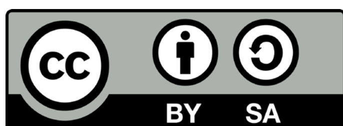
Kefa Murimi

Gerisson Mulongo

Direitos de Autor

Este documento é publicado sob as condições do Creative Commons
http://en.wikipedia.org/wiki/Creative_Commons

Attribution http://creativecommons.org/licenses/by/2.5/



Modelo de módulo é licenciado Universidade Virtual Africana licenciada sob uma Licença Internacional Creative Commons Atribuição-Partilha 4.0 Internacional. CC-BY, SA

Apoiado por

Projecto Multinacional II da UVA financiado pelo Banco Africano de Desenvolvimento.

Índice

Prefácio 2

Créditos de Produção 3

Direitos de Autor 4

Apoiado por 4

Unidade 1. Introdução às Linguagens de programação e paradigmas de programação 7

Introdução à Unidade 7

Objetivos da Unidade 7

Termos-chave 7

Actividades de Aprendizagem 8

Actividade 1.1 - Introdução à programação 8

Introdução 8

Programas de sistema (softwares de sistema): 9

Programas de aplicação (programas aplicativos): 9

Avaliação 10

Conclusão 10

Actividade 1.2 - Níveis de Linguagens de programação 10

Introdução 10

Conclusão 12

Avaliação 13

Actividade 1.4 - Paradigmas de Linguagens de programação 14

Introdução 14

Conclusão 15

Avaliação 15

Resumo da Unidade 15

Avaliação 16

Avaliação da Unidade 16

Critérios de Avaliação 16

Leituras e outros Recursos 17

Unidade 2. Algoritmia: fluxograma e pseudocódigo 18

Introdução à Unidade 18

Objectivos da Unidade. 18

Termos-chave 18

Actividades de Aprendizagem 19

Actividade 1.1 - Pseudocódigo 19

Introdução 19

Introdução a vectores 29

Conclusão 30

Avaliação 30

Actividade 1.2 - Fluxograma (diagrama de fluxo) 31

Introdução 31

Conclusão 32

Avaliação 33

Resumo da Unidade 33

Avaliação da Unidade 34

Critérios de Avaliação 34

Avaliação 34

Leituras e outros Recursos 35

Unidade 3. Introdução à Linguagem de programação C 36

Introdução à Unidade 36

Objetivos da Unidade 36

Termos-chave 36

Actividades de Aprendizagem 37

Actividade 1.1 - Tipos de dados, variáveis e operadores 37

Introdução 37

Funções Printf() e Scanf() 42

Tipos de dados 43

Variáveis 44

Constantes 46

Tipo de Operadores em C 47
Actividades Hands-on 50
Conclusão 51
Avaliação 51
Actividade 1.2 - Estruturas de Controle e Funções 53
Introdução 53
Estrutura de controle condicional 53
Loops/estruturas de controle de repetição. 55
Outras Instruções 57
Funções 58
Escopo de regras 60
Hands-on atividade (prática) 61
Conclusão 62
Avaliação 62
Actividade 1.3 - Vectores(array) e strings 63
Introdução 63
Strings 66
Conclusão 67
Avaliação 68
Resumo da Unidade 68
Avaliação da Unidade 69
Avaliação 71
Leituras e outros Recursos 73
Unidade 4. Métodos de programação e princípios de programação modular 74
Introdução à Unidade 74
Objectivos da Unidade. 74
Termos-chave 74
Actividades de Aprendizagem 75
Actividade 1.1 - Os métodos de programação 75
Introdução 75

Conclusão 77

Avaliação 78

Actividade 1.2 - Programação modular em C 79

Introdução 79

Definição de uma função 79

Passagem de parâmentros a uma função 80

O corpo da função 83

Instrução return 83

Funções que retornam um valor 83

Procedimentos - o tipo void 84

Variáveis Globais 85

Variáveis Locais 86

Função recursiva 88

Os arquivos de cabeçalho em C 90

Conclusão 90

Avaliação 91

Resumo da Unidade 91

Avaliação da Unidade 92

Critérios de Avaliação 92

Avaliação 92

Leituras e outros Recursos 93

Avaliação do Curso. 93

Critérios de avaliação 93

Avaliação 93

Referências do Curso 95

Outros Links Consultados 96

Unidade 1. Introdução às Linguagens de programação e paradigmas de programação

Introdução à Unidade

Esta unidade apresenta a parte da história das linguagens de programação, centrando-se nos diferentes níveis de linguagens de programação que existem, desde as linguagens de baixo nível e de alto nível, enfatizando as diferentes gerações destas linguagens. Também serãoá abordadas e discutidas as diferentes concepções de diversos paradigmas de programação.

Objetivos da Unidade

Apóós a conclusão desta unidade, deverá ser capaz de:

1. Identificar os marcos históricos da evolução das linguagens de programação
2. Caracterizar os diferentes momentos históricos da evolução das linguagens de programação
3. Identificar os diferentes níveis de linguagens de programação
4. Enfatizar as diferentes gerações das linguagens de programação
5. Discutir os diferentes paradigmas de programação

Termos-chave

Linguagens de Programação: é uma notação formal para descrever a execução de algoritmos em computador.

Sintaxe: conjunto de regras que estipulam o uso correcto dos termos para construir expressões válidas.

Semântica:conjunto de termos, palavras e sinais, que assumem determinados significados.

Actividades de Aprendizagem

Actividade 1.1 - Introdução à programação

Introdução

Esta atividade irá fazer um breve historial e introduzir os fundamentos da programação, especialmente definir alguns conceitos importantes como programas de sistema, programas de aplicação entre outros.

Detalhes da atividade

Uma linguagem de programação é um sistema de escrita formal para enunciar a execução de operações no computador.

Uma linguagem de programação é composta por:

Uma terminologia ou um conjunto de termos, palavras e sinais, que assumem determinados significados (semântica)

Um conjunto de regras que estipulam o uso correcto dos termos para construir expressões válidas (sintaxe).

As linguagens de programação têm como finalidade fundamental proporcionarem meios para a resolução de problemas, mediante um processamento computarizado .Existem várias linguagens de programação, cada uma com suas características próprias.

Exemplos: Payton, Java, Pascal, Clipper, C, Visual Basic, Delphi etc.

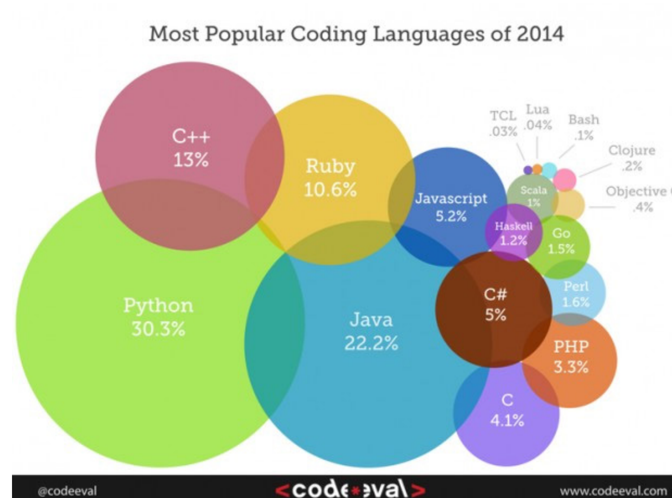


Figura 1:Linguagens de programação mais utilizadas em 2014:

Um programa, no contexto das linguagens de programação, é um conjunto de "frases", que utilizam os termos e as regras de determinada linguagem de programação, com vista a concretizar certos objectivos. Normalmente, um programa, é a tradução de um algorítmo numa linguagem de programção, para que possa ser entendido e executado pelo computador.

Um programa é uma seqüência de instruções executadas passo a passo para resolver um problema particular e fazer com que o computador para fazer uma tarefa específica.

Na literatura, existem geralmente dois tipos de programas.

Programas de sistemas

Os programas de aplicação

Programas de sistema (softwares de sistema):

Os programas de sistemas, também chamados de softwares de sistemas são os programas fundamentais para o funcionamento correcto do computador. Eles foram concebidos para facilitar a utilização do computador e torná-lo eficaz e rápido. Os Sistemas Operacionais (SO) e os drivers de dispositivo são exemplos de tais programas. Um sistema operacional é o programa de sistema fundamental pois é ele que controla e coordena a utilização do hardware durante a execução dos programas de aplicação e controla todos os seus recursos

Exemplos:

- Microsoft Windows Operating Systems (Windows 95, 98, 2000)
- Microsoft Windows XP
- Microsoft Windows 7, 8
- Unix
- Linux

Programas de aplicação (programas aplicativos):

Estes programas definem as formas de utilização dos recursos do sistema com o objectivo de resolver problemas dos utilizadores. Estes são os programas que não são fornecidos previamente pelo sistema operacional. O software aplicativo pode ser implementado em todas as áreas comuns, tais como bases de dados, comunicação, compatibilidade, gráficos, criação de vídeo, etc.

Exemplos:

- O Microsoft Word é um software de aplicação para processamento de texto.
- O Internet Explorer eb o Mozilla Firefox são o software escrito para uma aplicação de navegação na web.
- Os jogos de computador também são programas de aplicação.

Conclusão

Esta atividade foca alguns conceitos importantes de programação.

Avaliação

Exercicios:

1. Define um sistema operativo?
2. Lista cinco problemas da vida quotidiana que podem ser resolvidos pelo computador?
3. Aponta pelo menos duas situações que levaram ao desenvolvimento de programas.
4. Dê exemplos de dois programas de aplicação.
5. Dê exemplo de dois programas de sistemas.

Actividade 1.2 - Níveis de Linguagens de programação

Introdução

A programação pode ser definida como a utilização de uma linguagem de programação para escrever instruções que podem ser executadas pelo computador, de modo a resolver os problemas através da aplicação da sintaxe e de semântica da linguagem correcta.

Centenas de linguagens de programação, foram desenvolvidos e alguns estão em existência, enquanto outros não. Isto significa que o desenvolvimento da linguagem de programação já percorreu um longo caminho, assim, tem uma longa história, desde as primeiras linguagens como, Prolog, Fortran, C, Visual Basic.Net, etc. Algumas delas foram melhorados e ganhou mais popularidade do que outros.

Portanto, nesta actividade, você deve ler os tópicos sobre o desenvolvimento histórico da programação; como programação e começou a implementar o programa em suas decendance idiomas disponíveis. Influências sobre a evolução da concepção e desenvolvimento de idiomas.

As linguagens de programação podem ser agrupadas em dois grandes grupos: linguagens de baixo nível e linguagens de alto nível.

Esta atividade pretende mostrar as diferentes gerações de linguagens de programação, dando exemplos pertencentes a cada uma das gerações. Destacam-se também as vantagens e desvantagens de cada nível de linguagem.

Detalhes da atividade

Desde a sua criação, as linguagens de programação evoluíram e diversificaram. As Linguagens de programação existentes podem ser classificadas em dois níveis.

- Linguagens de baixo nível (cada instrução está associada com uma sequência de 0´s e 1,`s, designada por palavras).
- Linguagens de alto nível (é aquela cujo estilo e recursos a aproximam da língua natural do programador).

A linguagem de baixo nível é caracterizada por duas gerações:

Primeira geração:

Esta geração é também é chamada de linguagem máquina, onde cada instrução está associada com uma sequência de 0´s e 1,`s, designada por palavras.

A principal vantagem da linguagem de máquina que é escrito por um código de usuário pode correr muito rápido e assim eficaz, devido ao facto de o processador executa instruções directamente. O programador tem total controlo da máquina e desses recursos. No entanto, existem muitas desvantagens. Por um lado, existe um problema da portabilidade comparada com a diversidade de arquitectura que pode ter plataformas de entrega de programa. Por exemplo, para executar um código em duas arquiteturas diferentes de CPU, o programa deve ser totalmente reescrito. Diz-se que esta linguagem é dependente da máquina como instruções variam de acordo com a arquitetura do computador. Por outro lado, o código é difícil de escrever, editar e depurar.

Exemplo:

- Linguagem máquina: (01000100011100)-circuitos electrónicos -baixo nível

Segunda Geração:

Esta geração é também chamada de linguagem de montagem . O código é escrito com variáveis e símbolos. O código deve ser convertido para um formato compreensível pelo computador, por isso a linguagem de máquina antes de ser executada pelo computador. Este é o montador que é utilizado para efectuar esta conversão. Esta linguagem fornece uma facilidade óbvia no que diz respeito à velocidade, o mais próximo da linguagem de máquina, mas requer um grande esforço de programação e é difícil de usar de forma eficaz para grandes aplicações. Esta linguagem também é dependente da máquina.

Exemplo:

- Linguagem Assembly (add ax, es:(bx))-montagem

A linguagem de alto nível é caracterizada por três gerações:

Terceira Geração:

As linguagens de terceira geração também chamadas de linguagens de programação modernas ou estruturadas, são caracterizadas pela grande capacidade procedural e estrutural de seus dados foram desenvolvidos de 1974 a 1986.

As linguagens de terceira geração tiveram como principais características a possibilidade de criar sistemas distribuídos, incorporar recursos mais inteligentes, e exigir um hardware menos robusto.

Quarta Geração:

A quarta geração das linguagens de programação foram desenvolvidas a partir de 1986 e teve como características principais a geração de sistemas especialistas, o desenvolvimento de inteligência artificial e a possibilidade de execução dos programas em paralelo.

São também conhecidas como linguagens artificiais contém uma sintaxe distinta para representação de estruturas de controle e dos dados. Essas linguagens por combinarem características procedurais e não procedurais, representam estas estruturas com um alto nível de abstração, eliminando a necessidade de especificar algoritmicamente esses detalhes.

Quinta Geração:

A linguagem de 5ª geração foi criada na década de 90 , são baseadas na resolução de problemas utilizando restrições do problema, ao invés de usar o algoritmo escrito por um programador. Utiliza como novos recursos de línguagem a programação de eventos e utilização GUI entre outros novos recursos. Exemplos desses tipos de linguagens são: Visual Basic, Visual C ++, Small Talk, java.

Conclusão

Nesta unidade foi introduzida a classificação geral das linguagens de programação de computador, ou seja, as linguagens de alto nível e linguagens de baixo nível. Alguns exemplos foram apresentados.

A linguagem de máquina é a única linguagem entendida pelo computador e todos os outros idiomas devem ser traduzidas em linguagem de máquina por um tradutor antes de suas instruções são executadas.

Tabela 1: Relação das linguagens de programação com o ano em que foram desenvolvidas:

1957	FORTRAN	
1958	ALGOL	
1960	LISP	
1960	COBOL	
1962	APL	
1962	SIMULA	
1964	BASIC	
1964	PL/1	
1966	ISWIM	
1970	Prolog	
1972	C	

1975	Pascal	
1975	Scheme	
1977	OPS5	
1978	CSP	
1978	FP	
1980	dBase II	
1983	Smalltalk	80
1983	Ada	
1983	Parlog	
1984	Standard ML	
1986	C++	

1986	CLP(R)	
1986	Eiffel	
1988	CLOS	
1988	Mathematica	
1988	Oberon	
1990	Haskell	
1995	Delphi	
1995	Java	

Avaliação

Exercícios:

1. O que caracteriza as línguagens de 1ª, 2ª e 3ª geração?
2. Apresenta duas vantagens de linguagens de programação.
3. A principal vantagem das linguagens de alto nível é a abstração. A afirmação é correcta? Justifica a tua resposta.
4. O que caracteriza as linguagens de alto nível?
5. Como aos programas são traduzidos em código de máquina?
6. Classificação das língugens abaixo:

   FORTRAN, BASIC, C ++, Pascal
7. Para que serem ou servem os tradutores?
8. Apresenta dois exemplos de linguagens de 5ª geração?

Actividade 1.4 - Paradigmas de Linguagens de programação

Introdução

Nesta unidade será apresentado o conceito do paradigma de programação, como se traduz este significado para o contexto de programação e os tipos de paradigmas de programação (programação imperativa, orientada a objetos, funcional e lógica). Portanto, esta unidade vai permitir entender o porquê deste conceito.

Detalhes da atividade

Paradigma de programação é um modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas caraterísticas comuns.

Os paradigmas de programação podem ser classificados em:

- Programação imperativa (também denominado procedural)
- Programação Orientada a objectos
- Programação funcional
- Programação lógica

Programação imperativa (também denominado procedural)

Programação convencional, onde os programas são decompostos em "passos" de processamento que executam operações complexas. Rotinas são usadas como unidades de modularização para definir tais passos de processamento.Exemplos: Pascal e C.

Programação orientada a objectos

Enfatiza a definição de classes de objectos. Instâncias de classes são criadas atarvés dp programa conforme a necessidade, durante a execução de programas. Exemplos: C++, Java.

Programação Funcional

Estilo de programação que tem origem na teoria das funções matemáticas. Enfatiza o processamento de valores através do uso de expressões e Funções. Exemplo: LISP.

Programação Lógica

Neste tipo de programação, programa-se de forma declarativa, ou seja, especificando o que deve ser computado, não apresenta instruções explícitas e sequenciamento.Exemplo: prolog (Programming Logic).

Conclusão

Esta acividade destacou os principais paradigmas de programação.

Avaliação

Exercício: Classifica as linguagens de programação a seguir segundo a categoria dos paradigmas de programação apresentadas acima:

- Fortran,
- Pascal,
- "C"
- Smalltalk,
- "C++",
- Java
- LISP
- PROLOG

Resumo da Unidade

Na informática, o algoritmo é um projecto de software, ou seja, antes de se fazer um program na linguagem de programação desejada deve-se fazer antes o respectivo algoritmo.

Com esta unidade, o aluno terá uma visão clara do processo de introdução à programação de computadores, conhecendo as linguagens de programação e os diferente níveis existentes. Foi também apresentada uma introdução aos paradigmas de linguagens de programação a fim de o aluno poder compreender e conhecer as possíveis técnicas utilizadas na programação.

Avaliação da Unidade

Avaliação sobre as linguagens de programação e paradigmas de programação

Instruções

No final desta unidade serão apresentados dois grupos de exercícios sobre linguagens de programação e paradigmas de programação. Para além das informações já disponíveis na unidade o aluno também poderá consultar a internet como uma fonte de pesquisa para responder a todas as questões de avaliação.

Critérios de Avaliação

O exercício 1 será avaliado em 8 pontos atribuindo 2 pontos em cada alínea e o exercício 2 em 12 pontos em que cada alínea valerá 6 pontos por serem exercícios que requerem mais trabalho, organização de ideias e tempo.

Avaliação

Exercício 1:

1. Conceitue linguagem de programação e diferencie sintaxe de simântica?
2. Por que é útil para um programador ter alguma experiência no projecto de linguagens de programação?
3. Descreva, justificando pelo menos três critérios importantes que devem ser considerados na escolha de uma linguagem e programação?
4. Que linguagem de programação tem dominado a inteligência artificial nos últimos 50 anos?

Exercício 2:

1. Faça Uma pesquisa on-line sobre Paradigmas de programação e produza links para fontes de informação confiáveis de todos os paradigmas apresentados nesta unidade.
2. Escreva, com as tuas próprias palavras, um breve resumo das características diferenciais dos diferentes paradigmas de programação. Apresenta exemplos sempre que for possível.

Leituras e outros Recursos

As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.

Unidade 2. Algoritmia: fluxograma e pseudocódigo

Introdução à Unidade

A algoritmia é a base essencial para a programação porque estimula o raciocínio lógico e prepara os alunos para a resolução de problemas de programação. O conceito central da programação e da ciência da computação é o conceito de algoritmos, isto é, programar é basicamente construir algoritmos. Nesta unidade serão abordados o conceito de algoritmo e as suas formas de representação , o pseudocódigo e fluxograma.

Objectivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Elaborar um algoritmo
2. Formalizar o raciocínio lógico
3. Escrever pseudocódigo para a resolução de problemas
4. Fazer fluxogramas a partir de pseudocódigos

Termos-chave

Algoritmo: conjunto de regras (instruções) bem especificadas para a resolução de um problema específico.

Programa: algoritmo escrito numa linguagem computacional.

Fluxograma (diagrama de fluxo): são uma representação gráfica que utilizam formas geométricas padronizadas ligadas por setas de fluxo, para indicar as diversas instruções e decisões que devem ser seguidas.

Pseudocódigo: escrever um algorítmo por meio de regras predefinidas, os passos a serem seguidos para a sua resolução.

Actividades de Aprendizagem

Actividade 1.1 - Pseudocódigo

Introdução

Pseudocódigo é uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples (nativa a quem o escreve, de forma a ser entendido por qualquer pessoa) sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação. É, como o nome indica, um pseudo-código e, portanto, não pode ser executado num sistema real (computador) , antes tem de ser passado para uma linguagem de programação específica.

Algorítmo: um algorítmo é uma sequência de instruções ordenadas de forma lógica para a resolução de uma determinada tarefa ou problema.

Exemplo: somar números, fazer sanduíche, trocar lâmpada, tirar dinheiro no banco, etc.

Programa - um programa é um algoritmo escrito em uma linguagem de programação.

Etapas para o desenvolvimento de um programa:

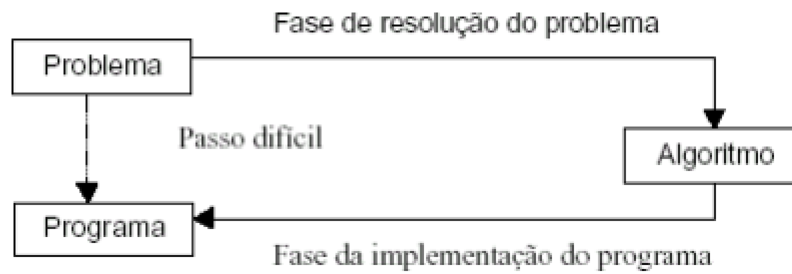
1-Análise- estuda o enunciado, 2-algoritmo-utiliza ferramentas para descrever o problema com suas soluções, 3-Codificação-implementação da solução numa linguagem de programação.

Detalhes da actividade

A seguir será apresentado um algorítmo não computacional cujo objectivo é usar um telefone público.

Exemplo 2:

1. Tirar o telefone do gancho
2. Ouvir o sinal de linha
3. Introduzir o cartão
4. Teclar o número de destino desejado
5. Se der o sinal de chamar
6. Conversar
7. Desligar
8. Retirar cartão
9. se não
10. repetir
11. fim



Em geral, os pseudocódigos são utilizados para descrever um algoritmo antes de passar para uma linguagem de programação específica. Isso contribui para o planejamento inicial do programa, criando a estrutura lógica e sequência de instruções que constituem o código. A seguir serão apresentados os conceitos básicos de algoritmos como: tipos de dados, variáveis e constantes, comandos de entrada/saída, operadores e operandos e instruções de controlos.

Exemplo 1: programa Número

```
Programa SeparaNum

Var num:real

Inicio

Escreva "Introduza um número"

Leia num

Se num = 0 então

Escreva "O número é nulo"

Se num > 0 então

Escreva num, " é um número positivo"

Senão

Escreva num, " é um número negativo"

Fim.
```

Exemplo 2:

```
programa Habilitação
var
idade:numérico
inicio
escreva ("informe idade:")
leia(idade)
se idade >= 18 então
escreva("pode tirar a carteira")
senão
escreva("não pode tirar a carteira")
 fimse
fimalgoritmo
```

Tipos de dados: os tipos básicos mais utilizados e conhecido são:

Inteiro ("int, short int ou long int"): : qualquer número inteiro, negativo, nulo ou positivo. Exemplo: -2, -1, 0...

Operações: soma(+), subtração(-), multiplicação(*), divisão inteira(/), resto(%) e comparações.

Real ("float ou double"): qualquer número real, negativo, nulo ou positivo. Exemplo: 2.5, 3.1

Operações: soma(+), subtração(-), multiplicação(*), divisão exata(/) e comparações.

Carácter ("char"): qualquer conjunto de caracteres alfanuméricos.

Ex: A, B, "ABACATE "

Operações: comparações

Texto ou cadeia de caracteres ("STRING"): uma variável deste tipo poderá armazenar uma cadeia de caracteres de qualquer tamanho. Caso seja imprescindível para o entendimento pode-se acrescentar, entre parênteses, a quantidade máxima de caracteres. (Exemplo: texto (10)).

Obs.: Os textos deverão ser representados sempre entre apóstrofes para que não se confundam com os valores numéricos. Veja que o inteiro 5, é diferente do texto '5'.

Lógico (“BOOLEAN”): tipo especial de variável que armazena apenas os valores V e F, onde V representa VERDADEIRO e FALSO. Ex: e, ou, não

Operações: Verdadeiro ou Falso

Variáveis e constantes

Na maioria das linguagens de programação, quando o computador está executando um programa e encontra uma referência a uma variável ou a uma constante qualquer, se esta não tiver sido previamente definida, ele não saberá o que fazer com ela. Da mesma forma, um programador que estiver implementando um algoritmo, em alguma linguagem de programação, terá o seu trabalho simplificado se todas as constantes e variáveis referenciadas no algoritmo tiverem sido previamente declaradas. As constantes são declaradas antes das variáveis. Vejamos os formatos da declaração e alguns exemplos.

Declaração:

Variável<identificadores> : tipo

Exemplos:

Inteiro X1; onde X1 é o nome de um local de memória que só pode conter valores do tipo inteiro.

real SOMA, MÉDIA;

caracter frase, nome;

Constantes são endereços de memória destinados a armazenar informações fixas, inalteráveis durante a execução de um programa.

A declaração de um constante pode reservar o espaço de memória para armazenamento de dados, cujo valor é fixo para a totalidade do algoritmo.

Declaração:

PI= 3.1416

Comandos de entrada e saída

Comandos simples:

É uma instrução simples.

leia(x);

Comandos composto:

Um grupo de comandos simples que executam alguma tarefa.

```
Início
 leia(x);
 y = 2*x;
fim.
```

Operadores e Operandos

Operadores:

Na solução da grande maioria dos problemas é necessário que as variáveis tenham seus valores consultados ou alterados e, para isto, devemos definir um conjunto de operadores, sendo eles:

Operador de atribuição

Identificador Expressão aritmética

Expressão lógica

Expressão literal

Ex: O valor da expressão é atribuído ao identificador (variável).

X = 2; y = 5-x;

Este comando permite que se forneça ou altere o valor de uma determinada variável, onde o tipo desse valor seja compatível ao tipo de variável na qual está sendo armazenado, de acordo com o especificado na declaração.

NUM = 8 {A variável NUM recebe o valor 8}

NOME = "Guilherme" {A variável NOME recebe o valor 'Guilherme'}

CONT = 0

AUXNOME = NOME {A variável AUXNOME recebe o conteúdo da variável NOME}

ACHOU = falso {A variável ACHOU recebe o valor falso}

Operadores aritméticos:

Os operadores aritméticos são utilizados na resolução de expressões matemáticas utilizando quaisquer valores, sejam eles constantes ou variáveis. Retorna um inteiro se todos os valores são inteiros ou um real se todos os valores são reais.

São eles:

+ = Adição / = Quociente da divisão de inteiros : 5/2=2

* = Multiplicação % = Resto da divisão de inteiros : 5%2=1

- = Subtração ou inverso do sinal. a**b = Exponenciação ab

/ = Divisão TRUNCA(A) - A parte inteira de um número fracionário.

Operadores relacionais:

São utilizados para relacionar variáveis ou expressões, resultando num valor lógico (Verdadeiro ou Falso), sendo eles:

== igual

!= diferente de

< menor

> maior

<= menor ou igual

>= maior ou igual

Exemplos:

NUM = 3

NOME = 'ELIZABETH'

NUM > 5 é falso (0) {3 não é maior que 5}

NOME < 'DENIZ' é verdadeiro (1) {DENISE vem antes de DENIZ}

(5 + 3) > = 7 é verdadeiro

NUM != (4 – 1) é falso {lembre-se que NUM "recebeu" 3}

Operadores lógicos:

São utilizados para avaliar expressões lógicas, sendo eles:

e (“and”) Ù : conjunção

ou (“or”) Ú :disjunção

não (“not”) Ø : negação

Exemplos:

ACHOU = falso

NUM =9

(4>5) e (5>3) =>falso e verdadeiro (1) = falso (0)

não ACHOU =>verdadeiro

Prioridade dos operadores:

Durante a execução de uma expressão que envolve vários operadores, é necessária a existência de prioridades, caso contrário poderemos obter valores que não representam o resultado esperado.

A maioria das linguagens de programação utiliza as seguintes prioridades de operadores :

1° - Efetuar operações embutidas em parênteses “mais internos”

2° - Efetuar Funções

3° - Efetuar multiplicação e/ou divisão

4° - Efetuar adição e/ou subtração

5° - Operadores Relacionais

6° - Operadores Lógicos

Ou seja,

Primeiro: Parênteses e Funções

Segundo: Expressões Aritméticas

1)+, - (Unitários)

2) **

3) *,/

4)+,- (binário)

Terceiro: Comparações

Quarto: Não

Quinto: e

Sexto: ou

Estrutura de controlo

Estrutura de Repetição ENQUANTO-FAÇA

Para além das decisões é muito frequente surgir a necessidade de repetir instruções. As repetições podem assumir diferentes formas, aqui consideramos:

ENQUANTO <Condição for verdadeira> FAÇA

<Comandos>

FIM ENQUANTO

Exemplo1 : Faça um algoritmo para ler e escrever o Nome de 20 pessoas.

ALGORITMO LeEscreve

VARIÁVEIS

Nome : CADEIA

Total : INTEIRO

INICIO

Total 0

ENQUANTO Total<20 FAÇA

LEIA(Nome)

ESCREVA (‘Nome=’, Nome)

Total Total + 1

FIM ENQUANTO

END.

Estrutura de Repetição REPITA-ATÉ

Outra forma:

REPITA

<Comandos>

ATE <Condição for verdadeira>

Exemplo2: Faça um ALGORITMO para ler e escrever o Nome de 20 pessoas.

ALGORITMO LeEscreve

VARIÁVEIS

Nome : CADEIA

Total : INTEIRO

INICIO

Total 0

REPITA

LEIA(Nome)

ESCREVA('Nome=',Nome)

Total Total + 1

ATÉ Total >=20

FIM

Estrutura de selecção

Estrutura SE…ENTÃO…SENÃO (IF…THEN…ELSE)

SE <Condição FOR verdade> ENTÃO

<Comandos>

[SENÃO

<Comandos>] Colchete indica que o comando é opcional

FIMSE

Exemplo 1: Dado dois valores A e B quaisquer, faça um algoritmo que imprima se A > B, ou A < B , ou A = B.

```
ALGORITMO Maior
VARIÁVEIS
A,B : INTEIRO
INICIO
ESCREVA('Digite os valores A e B');
SE A > B ENTÃO
ESCREVA('A é maior que B')
SENÃO
SE A < B ENTÃO
ESCREVA('A é menor que B')
SENÃO
ESCREVA('A é igual a B')
FIM
```

Estrutura case

```
ESCOLHA <Valor>
<Opções> : <Comandos>
...
<Opções> : <Comandos>
[ Senão
<Comandos>]
FIMESCOLHA
```

Exemplo usando o comando ESCOLHA:

```
ESCOLHA Idade
0..3 : ESCREVA('BEBÊ')
4..10 : ESCREVA('CRIANÇA')
11..18 : ESCREVA('ADOLESCENTE')
SENÃO
ESCREVA('ADULTO')
FIM ESCOLHA
```

Introdução a vectores

Vimos, no início desta unidade, ser possível dar um nome para uma posição de memória, sendo que a esta será associado um valor qualquer. No entanto, muitas vezes, esta forma de definição, ou de alocação de memória, não é suficiente para resolver certos problemas computacionais.

Suponhamos, que se pretendia desenvolver um programa que dadas as notas dos cerca de 4000 alunos de uma Universidade, calculasse o desvio de cada uma relativamente à média das notas.

Não seria uma tarefa simples, visto que para o cálculo do desvios é necessário o cálculo prévio da média, o que implica manter as notas após o cáculo da média, ou seja, guardar as notas em variáveis. Teríamos que definir 4000 variáveis do tipo STRING, como é mostrado abaixo:

nota_1, nota_2, nota_3, nota_4….nota_4000… seria impraticável!!

Seria perferível a possibilidade de definir as 4000 variáveis de uma só vez, por exemplo da seguinte forma:

nota (1 até 4000), em que nota(1) guardaria a nota do aluno1, nota(2) do aluno 2, e assim sucessivamente fazendo variar o valor do índice até 4000.

Esta solução é conhecida como por "Vetor". Uma variável uni-dimencional, como o próprio Nome já indica, possui apenas uma dimensão, sendo ser possível definir variáveis com quaisquer tipo de dados.

Um vector pode ser definido como um conjunto de tamanho fixo de elementos do mesmo tipo ocupando posições contíguas.

Declaração e sintaxe:

DIM nome_vector (inicio ATÉ fim)

Em que:

nome_vector é o nome do vector escolhido pelo programador

início é o valor inicio do índice

fim é o valor máximo do índice

Definição:

ALGORITMO Define

VARIÁVEIS

<Nome>: VETOR [INÍCIOV : FIMV] DE <tipo>

INÍCIO

<Comandos>

FIM

PROGRAM Define;

VAR

<Nome>: VECTOR[INICIO..FIM] OF <tipo>;

INICIO

<Comandos>;

FIM.

onde

a) "VECTOR" é uma palavra reservada do Pc

b) Os valores "INÍCIO" e "FIM" correspondem aos índices inicial e final

c) Uma variável indexada pode ser apenas de um tipo de dado

Conclusão

Esta atividade frisa diferentes conceitos para o desenvolvimento de um pseudocódigo simples eficaz e inequívoca, a fim de formalizar a solução para um problema específico, sem estar vinculado às restrições de sintaxe relacionados a uma linguagem de programação específica.

Avaliação

Exercícios:

1. Qual é a ordem de precedência dos vários operadores da seguinte expressão:

   ((3 * a) – x ^ 2) – (((c – d) / (a / b)) / d)

2. Faça um algoritmo que leia os valores A, B, C e diga se a soma de A + B é menor que C.
3. Faça um algoritmo que leia dois valores inteiros A e B se os valores forem iguais deverá se somar os dois, caso contrário multiplique A por B ao final do cálculo atribuir o valor para uma variável C.
4. Faça um algoritmo para ler base e altura de 50 triângulos e imprimir a sua área.
5. Leia 20 valores reais e escreva o seu somatório.

Actividade 1.2 - Fluxograma (diagrama de fluxo)

Introdução

Um fluxograma (diagrama de bloco) é uma forma de representar um algorítmo através de símbolos (representação gráfica). Um vantagem é a simplicidade que os elementos gráficos proporcionam para o entendimento e uma desvantagem é a necessidade de aprender a sua simbologia. São recomendáveis já que muitas vezes os símbolos substituem várias palavras.

Serão apresentados alguns exemplos e posteriormente as sugestões de actividades.

Detalhes da actividade

A seguir serão apresentados os símbolos mais utilizados nos fluxogramas:

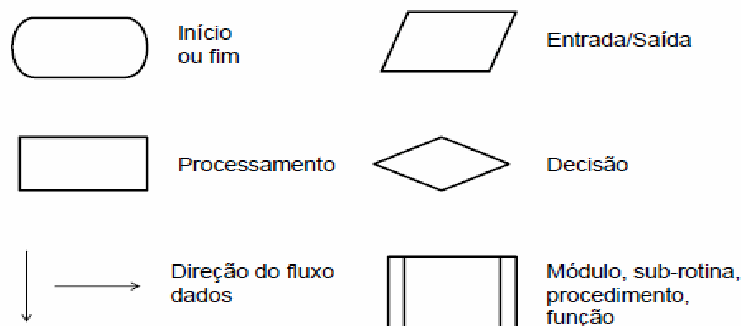
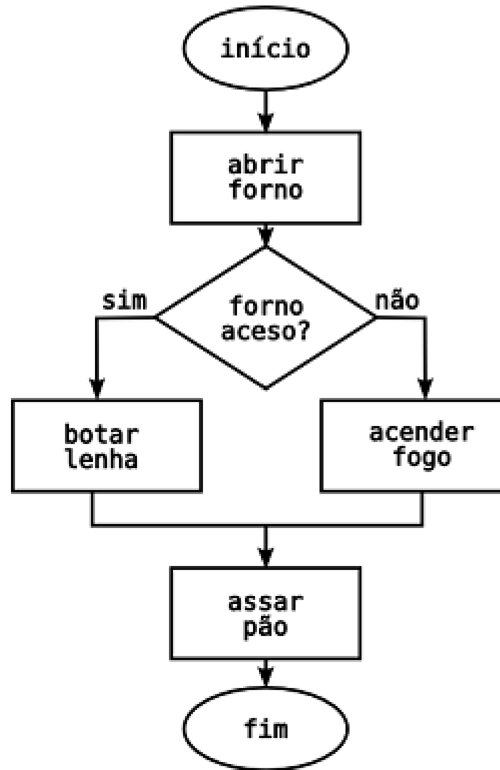


Figura 1: Simbologia fluxograma

Exemplo 1: Linguagem natural

- Obter as notas da primeira e da segunda prova
- Calcular a média aritmética entre as duas
- Se a média for maior ou igual a 7,
- o aluno foi aprovado, senão ele foi reprovado.

Exemplo 2: Fluxograma



Conclusão

Podemos concluir que os fluxogramas são úteis pois através da sua forma de representação é provável que todos os alunos possam ter uma visão mais completa do processo de criação de um algorítmo e uma melhor percepção e resultado final.

Avaliação

Execícios:

1. Faça os fluxogramas para os exemplos 1 e 2 dos pseudocódigos apresentados na página 1 e 2?
2. Faça um fluxograma que leia três valores e armazene nas variáveis IDADE1, IDADE2 e IDADE3, depois calcule a média das idades e armazene na variável IDADE_MÉDIA em seguida informe se a média das idades é maior ou menor que 20?
3. No algoritmo abaixo, mostre como ficaria a sua representação na forma do Fluxograma:

```
Programa idade
inteiro IDADE;
Inicio
escreva("Informe sua idade: ");
leia(IDADE);
se (IDADE < 16) então
escreva ("Você não pode votar");
senão
escreva("Você pode votar");
fim se;
Fim
```

Resumo da Unidade

Na informática, o algoritmo é um projecto de software, ou seja, antes de se fazer um programa na linguagem de programação desejada deve-se fazer antes o respectivo algoritmo.

Para a aprendizagem do algoritmo foi elaborado um conjunto de actividades que resumidamente compreende a definição do algoritmo e as etapas para a sua construção, o aluno terá uma visão geral do processode desenvolvimento de programas (softwares) bem como as linguagens de programação. Foi também apresentado as principais formas de representação dos algoritmos desenvolvendo a lógica de programação, como o fluxograma e o pseudocódigo como os mais utilizados.

Avaliação da Unidade

Verifique a sua compreensão!

Exame final sobre fluxograma e pseudocódigo

Instruções

Faça todos os exercícios a seguir apresentados.

Critérios de Avaliação

A avaliação da unidade consiste em uma análise de alguns dos conceitos abordados ao longo desta unidade, considerando importante o desenvolvimento de fluxogramas e pseudocódigos. A avaliação é classificada em 20 pontos, com 4 pontos para cada exercício.

Avaliação

Exercicio1 :Proponha um algoritmo que calcula o factorial de um número n. Indicação de n! = N (n-1) (n-2) ... * 1.

Exercicio 2 :Apresenta todas as razões para que o algorítmo abaixo esteja incorrecto:

```
Algoritmo Incorrecto
  x,y : inteiro
  z : real
  Inicio
  z ← x + 2
  y ← z
  x * 2 ← 3 + z
  y ← 5y + 3
  Fim.
```

Exercicio 3: Faça um algoritmo para resolver em R uma equação quadrática do segundo grau da forma: X2 + bx + c = 0

Exercicio 4 : Escreva em pseudocódigo um algoritmo que leia o nome de um vendedor, o seu salário fixo e o total de vendas efectuadas por ele no mês (em dinheiro). Sabendo que este vendedor ganha 15% de comissão sobre suas vendas efectuadas, informar o seu nome, o salário fixo e salário no final do mês.

Exercicio 5 :Faça o fluxograma correspondente ao pseudocódigo da questão 4.

Leituras e outros Recursos

As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.

Unidade 3. Introdução à Linguagem de programação C

Introdução à Unidade

Detalhes da programação podem ser diferentes em diferentes linguagens, mas algumas instruções básicas aparecem em quase todas as linguagens. Essas instruções incluem instruções de entrada (input) que são usadas para obter dados do teclado, de um ficheiro, ou algum outro dispositivo, instruções de saída (output) são usadas para exibir informações na tela ou enviar informações para um ficheiro ou outro dispositivo, instruções aritméticas para realizar operações aritméticas básicas, como adição e multiplicação, instruções de execução condicional que são usadas para verificar a existência de certas condições e executar a sequência apropriada de declarações e instruções de repetição para executar alguma ação repetidamente, normalmente com alguma variação. Portanto, esta unidade discute construções de programação como variável, tipos de dados, operadores, estruturas de controle, matrizes e sub-rotinas ou funções. Sub-rotinas ou funções irão reforçar o conceito de dividir um problema em problemas menores (técnicas de concepção top down e bottom up). Programas simples serão escritos usando a linguagem C para reforçar todos os conceitos aprendidos na unidade anterior.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Fazer download e instalar um programa.
2. Diferenciar os vários tipos de dados.
3. Fornecer soluções escrevendo programas simples em C

Termos-chave

Variáveis: variáveis são os nomes dados para a região da memória armazenada.O valor de uma variável altera durante a execução do programa.

Constante: representa como os valores são armazenados na memória e os valores fixos não mudam.

Declaração de variável: indica o tipo da variável e aloca memória para a variável.

Declaração de variável: indica o tipo da variável e aloca memória para a variável.

Tipo de Dados: tipos de dados definem o modo como os valores estão representados em um sistema.

Identificador: identificador é um nome de variável.

Constante: é um valor que não se altera durante a execução de um programa (fundamental).

Interpretador: traduz, instrução por instrução, à medida que o programa vai sendo lido e executado, ficando assim o programa dependente do software que efectua essa interpretação.

Compilador: traduz a totalidade das instruções de alto nível para um programa em código-máquina, o qual poderá ser executado autónoma.

Código Fonte - programa fonte: programa escrito numa linguagem de alto nível, enquanto não é convertido em executável.

Código objecto - programa objecto: programa depois de compilado (utilização de um compilado

Actividades de Aprendizagem

Actividade 1.1 - Tipos de dados, variáveis e operadores

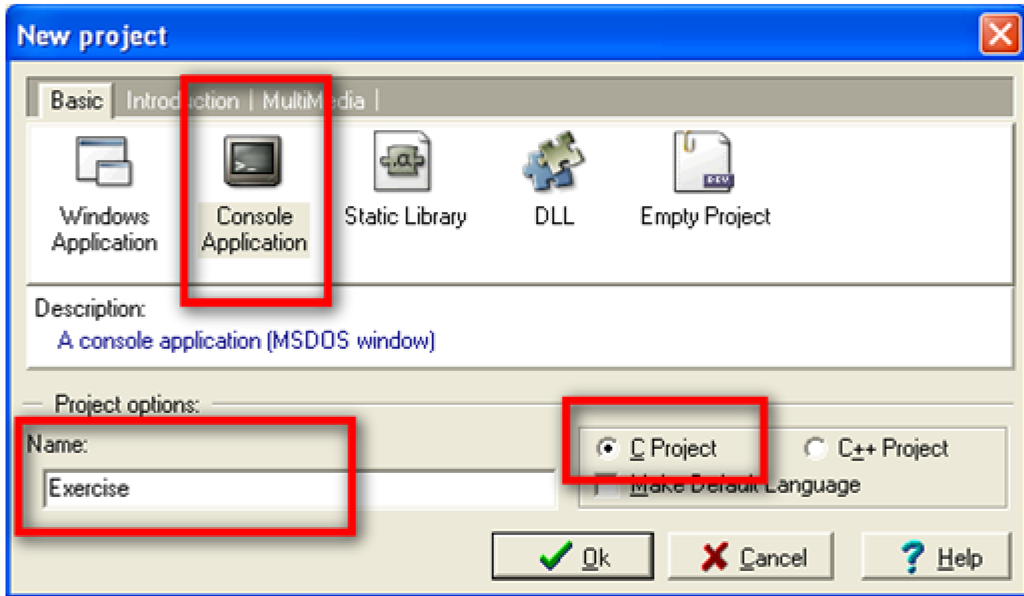
Introdução

Para discutir e ilustrar esses conceitos usando um programa C, primeiro temos que começar por nos familiarizar com o ambiente de linguagem (visão geral da linguagem C). Existem bons materiais de referência que dão um bom início de programação C, incluimos as referências fornecidas para esta unidade -programming and problem solving through 'C' languages by Harsha Priya, r. Ranjeet, Fundamentals of programming languages by Dipali P. Baviskarand Programming and PROB solving using C. by ISRD.

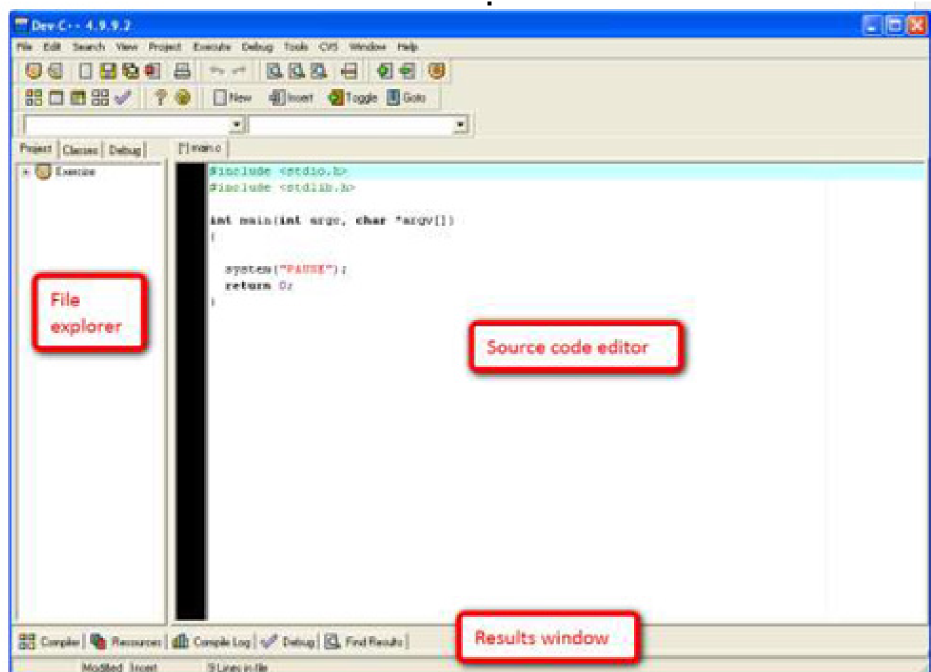
Existem muitos compiladores de C no mercado e foram concebidos ou desenvolvidos por diferentes grupos de desenvolvedores de programas. Portanto, a preferência de um compilador é deixado aqui neste unidade. Preferência deste curso é o compilador Dev C ++; podes baixá-lo, o recomendado pode ser encontrado http://liquidtelecom.dl.sourceforge.net (Dev-cpp 5.8.3 TDM-Gcc 4.8.1 setup.exe) ou a versão mais recente, atualizado, que pode ser executado na versão mais recente do Windows. Se você não conseguir solicita o seu instrutor

para ajudá-lo. O compilador lê C++ e isso não deve ser uma preocupação para você, porque vais aprender como compilar um programa em C renomeando os ficheiros para ter a extensão C (.c).

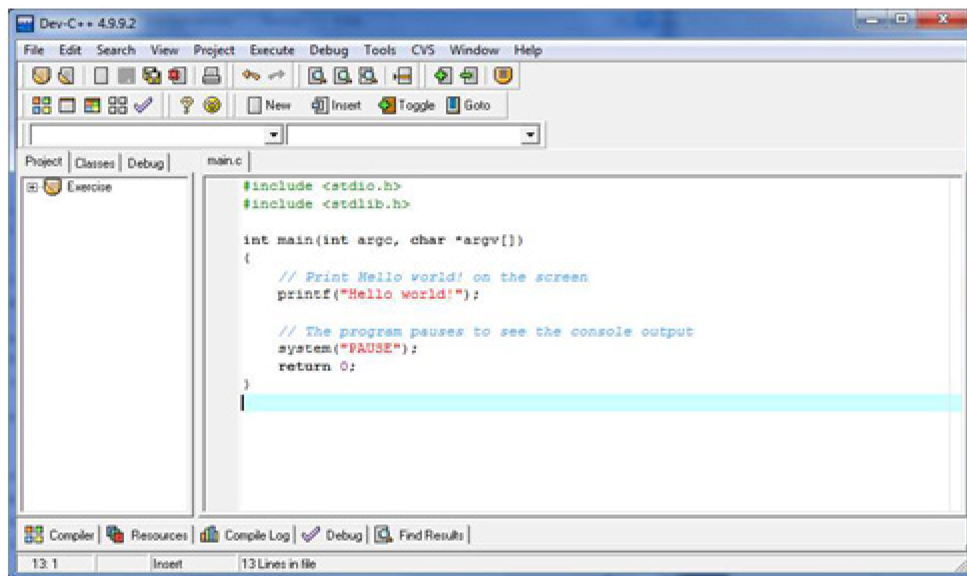
Antes de criar o ficheiro de código-fonte, é necessário criar um projeto (File> New> Project) e as opções incluem aplicativo de console, o nome do projeto e do projeto C para a linguagem C.



A próxima actividade é seleccionar a pasta para armazenar os ficheiros na próxima janela. E depois de indicar a pasta onde o ficheiro de configuração do projecto (.dev) será guardado, o IDE gera um ficheiro de origem base de código (por padrão, main.c). A janela IDE inclui três sub-janelas: Project Files Explorer, the Result Tabs, and the Source Code Editor. Estas janelas podem ser redimensionadas e minimizadas.



O sistema de declaração (“PAUSE”); está incluído automaticamente para interromper a execução do programa antes de fechar a janela do terminal, a fim de tornar possível a saída do programa. A janela File Explorer mostra o nome do projecto e os ficheiros incluídos. O Projecttab geralmente contém um único ficheiro com o código-fonte do programa. Neste painel, podemos encontrar dois guias adicionais: Classes e Debug. Guia Classes mostra as funções do programa. Separador de depuração mostra variáveis no processo de depuração assistiu. A janela de resultados é usado para apresentar os resultados das ações do IDE: compilação, erros de compilação, depuração directivas commands. A fonte editor de código mostra o código do programa. cComo o projeto já foi criado, nós podemos começar a escrever, em seguida, executar o nosso programa C. Por exemplo, um simples programa “Hello World”, a seguir:

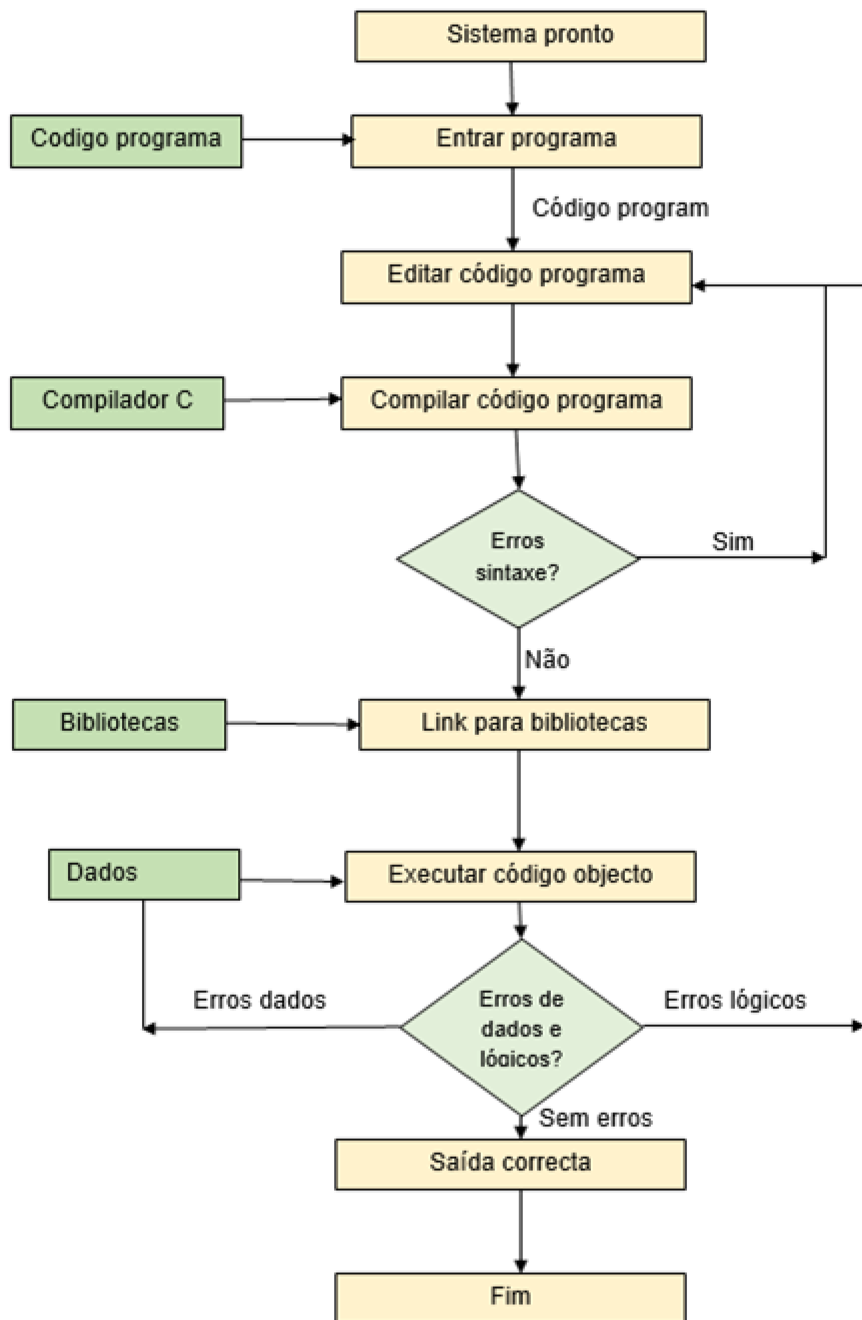


O processo de executar um programa em C envolve os seguintes passos:

1. Criação do programa.
2. Compilação do programa.
3. Vinculação do programa com funções que são necessárias a partir da biblioteca C.
4. Execução do programa.

Nota:

O conteúdo acima foi consultado em : http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf. Portanto, nós recomendamos que você visita este link para outras leituras. Introduz o Dev C ++ e de uma forma simples e clara (passo a passo), irá guiá-lo para facilmente iniciar. Não se preocupe tanto com a depuração porque esta parte iremos introduzi-lo na próxima unidade. Um resumo dessas etapas pode ser representado em um gráfico que se segue;



Processo de execução de um programa em C fonte: ISRD

Criando o programa: O programa que está a ser criado deve ser inserido em um ficheiro. O nome do ficheiro pode conter letras, números e caracteres especiais, seguido pela extensão. C. Exemplo de nomes de ficheiros válidos são;

```
Hello.c
```

```
Pract1.c
```

Compilar e ligar: Durante o processo de compilação as instruções do programa de origem são convertidos para uma forma que é adequada para a execução pelo computador. O processo de tradução verifica cada instrução para correcção e se não há erros em seguida, ele gera o código objecto.

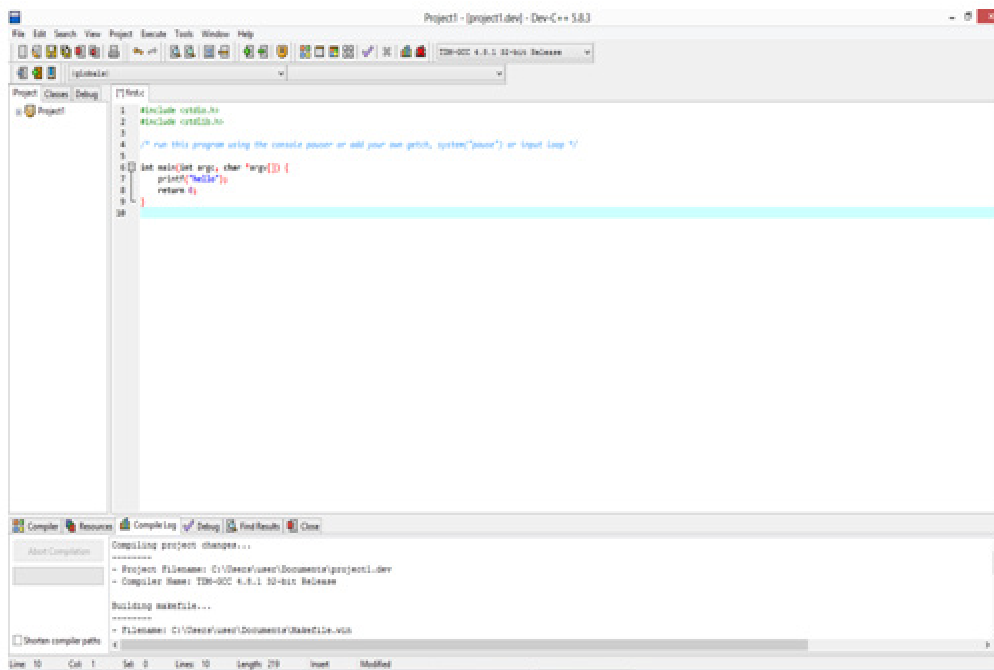
Durante a fase de ligação os ficheiros de programas e funções são colocados juntos e se tiverem erros de sintaxe e semântica da linguagem, estes são descobertos, são listados no processo de compilação terminam alí. Os erros devem ser corrigidos no programa de origem com a ajuda de um editor e a compilação é feita novamente.

A execução do programa: durante a execução, nós carregamos o código objecto executável na memória de computadores e executamos a instrução. Durante a execução, o programa pode solicitar alguns dados através do teclado. Um exemplo que descreve partes básicas de um programa em C usando Dev C ++;

```
#include <stdio.h>        }  <---- fihceiros cabeçalho
#include <stdlib.h>

Executar este programa usando console ou adiciona o getch,
system("pause") or input loop */          <---- comentário
                                          Função mais
int main(int argc, char *argv[]) {
        printf("hello");       }  <---- corpo do programa
        return 0;
                          contendo instruções executáveis
dentro de '{  }'.
}
```

Veja a figura abaixo para um programa usando o Dev C ++ versão 5.8.3 (TDM-GCC 4.8.1)



Funções Printf() e Scanf()

A função printf () é usada para escrever informações em um ficheiro, no écran ou qualquer outro dispositivo de saída. Enquanto a função scanf () é usada para ler os dados que o programa manipula ou escreve na tela. Estas funções são suportadas pelo arquivo de cabeçalho ou pré-processador diretiva <stdio.h>, ou seja, ficheiros de entrada e saída de cabeçalho padrão. Por exemplo, um programa que imprime um valor que já foi atribuído a uma variável e um valor (idade) lida para o sistema pelo usuário. O programa solicita que o usuário digita a idade;

```
#include <stdio.h>

#include <stdlib.h>

/* executa este programa usando a consola ou adiciona o getch,
system("pause") ou loop de entrada */

int main(int argc, char *argv[]) {

    int y=30;   //valor atribuído directamente a variável

    int idade;

    printf("Entra a idade\n");    //leitura da idade no prompt

    scanf("%d",&idade);                                   //lendo a
idade através do valor introduzido no teclado

    printf("y=%d\n",y);                               // valor
imprimido em y

    printf("idade=%d",idade);                         // imprimir
idade

    return 0;

}
```

Lembre-se de usar a formatação especifica quando lê e escreve valores. Além disso, use o endereço do operador '&' para a leitura dos valores. Coloca espaços nos caracteres quando aplicável para tornar o seu código mais legível.

Tipos de dados

Tal como as variávies o estudo dos tipos de dados é também importante. Os dados manipulados em C são digitados, ou seja, que para cada dado usado (nas variáveis, por exemplo) é preciso especificar o tipo de dado, o que permite conhecer a ocupação de memória (o número de bytes) do dado e sua representação.

A linguagem C, possui 5 tipos básicos de dados:

- char - caratere ASCII
- int - número inteiro
- float - número real de precisão simples
- void - nenhum valor
- double - número real de precisão dupla

Esta divisão deve-se basicamente ao número de bytes reservados para cada uma. Cada tipo e dado apresenta um intervalo de valores permitidos.

Os tipos char e int são inteiros e os tipos float e double são de ponto flutuante. A tabela 1 a seguir ilustra esses valores.

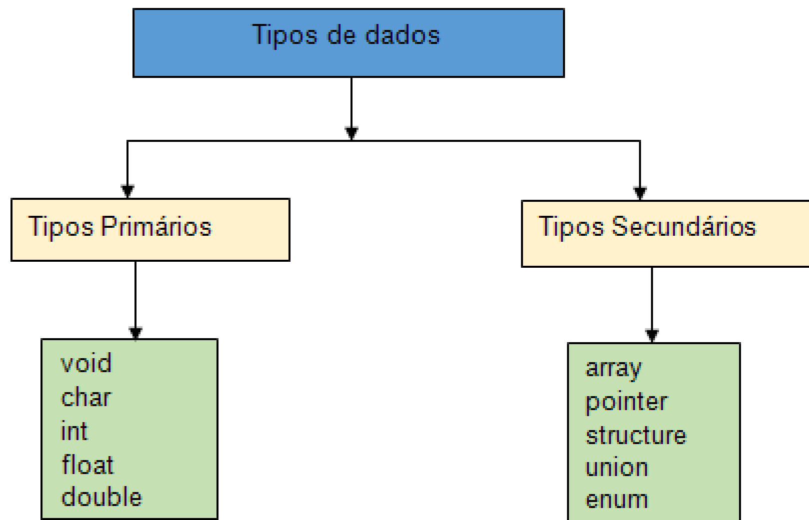
Tabela 1: Tamanho dos tipos de dados

Tipo	Tamanho em bits	Faixa de Números
char	8	-128 a 127
int	32	-2.147.483.647 a -2.147.483.64
float	32	seis dígitos de precisão
double	64	dez dígitos de precisão
void	0	nenhum valor

Tabela 2: Representação dos tipos de dados

Linguagem C	Formato	Tipo de dados
char	%c	caracter
int	%d	inteiro
float	%f	real
char []	%s	cadeia de carateres (string)

Para além destes tipos de dados, C suporta outros tipos de dados, conhecidos como tipos de dados secundários. A figura abaixo apresenta um resumo dos tipos de dados:



Os tipos de dados primários são também conhecidos como tipos de dados primitivos. Tipos de dados secundários podem ainda ser divididos em tipos de dados definidos pelo usuário e tipos de dados derivados. Por exemplo, a estrutura, union e enum são tipos de dados definidos pelo usuário, enquanto que array e pointer são tipos de dados derivados.

Variáveis

Em C, como na maioria das linguagens, as variáveis devem ser declaradas no inicio do programa, pois é através dela que é reservado um espaço na memória para a variável.

Uma variável é um objecto identificado pelo seu nome, podendo conter dados que poderão ser modificados durante a execução do programa.

As variáveis podem ser de vários tipos: int (inteiro), float (real de simples precisão), carácter e lógica. Veremos aqui como são declaradas as variáveis e usadas em um programa.

Para que se possa usar uma variável em um programa, é necessário fazer uma declaração dela antes. A declaração de variáveis simplesmente informa ao processador quais são os nomes utilizados para armazenar dados variáveis e quais são os tipos usados. Deste modo o processador pode alocar (reservar) o espaço necessário na memória para a manipulação destas variáveis. É possível declarar mais de uma variável ao mesmo tempo, basta separá-las por vírgulas (,).

Sintaxe: a sintaxe para a declaração de variáveis é a seguinte:

tipo variavel_1 [, variavel_2, ...] ;

Onde tipo é o tipo de dado e variavel_1 é o nome da variável a ser declarada. Se houver mais de uma variável, seus nomes são separados por vírgulas.

Exemplo 1: Declaração das variáveis:

```
int i;

int x,y,z;

char letra;

float nota_1,nota_2,media;
```

Exemplo 2: Observe o uso da declaração de variáveis no trecho de programa abaixo:

```
void main()

{

float raio, area; // declaracao de variaveis

raio = 2.5;

área = 3.14 * raio * raio;

}
```

Regras para o nome das variáveis:

- Um nome de variável deve necessariamente começar com uma letra;
- Um nome de variável não deve conter nenhum símbolo especial, excepto a sublinha (_) e nenhum espaço em branco;
- Um nome de variável não poderá ser uma palavra reservada a uma instrução de programa ou comando.

Quando se faz a declaração de uma variável está se determinando que tipo de dado ela vai receber. É possível, em C, declarar uma variável e já armazenar nela um valor inicial. Chamamos este procedimento de inicialização de uma variável.

Sintaxe para inicialização: a sintaxe para a inicialização de variáveis é:

tipo var_1 = valor_1 [, var_2 = valor_2, ...] ;

Onde tipo é o tipo de dado, var_1 é o nome da variável a ser inicializada e valor_1 é o valor inicial da variável.

Exemplo 3: Inicialização de variáveis:

```
int i = 0, j = 100;

float num = 13.5;

char  titulo = " Programa Teste ";
```

As variáveis são na memória do computador armazenadas através de um endereço. Daí devem ser declaradas no início de um bloco de código antes de serem utilizadas. Podemos concluir que é importante conhecer os seus diferentes tipos e as respectivas regras a fim de podê-los utilizar correctamente num programa.

Constantes

C suporta um número de constantes que incluem;

1. Constantes inteiras

   Por exemplo, 234, -78 etc consiste de um grupo de dígitos de 0 a 9 e pode ser representado como dígitos assinados utilizando o sinal + (positvo) ou - menos (valores negativos).

2. Constantes reais

   Por exemplo -0.98, 10.78 etc consiste em parte da fração

3. Constantes de caracteres

   Por exemplo 'v', 'j', 'y', 'n' etc colocado dentro de aspas simples ('.')

4. onstantes de string

   Por exemplo, "Elizabeth", "Estudante", "home" etc entre aspas duplas ("...").

Constantes podem ser definidas usando um operador de atribuição (=), ou uma palavra-chave ou # definir directiva.

Outras constantes de caracteres (sequência de espaços)

Há muitos outros caracteres não imprimíveis e C representa esses caracteres usando uma sequência de espaços (\). Por exemplo '\ n' representa nova linha. Ele avança informações para uma nova linha.

Tipo de Operadores em C

Um programa tem como característica fundamental a capacidade de processar dados, ou seja, realizar operações com este dados. Serão apresentados a seguir os operadores básicos utilizados em C.

As operações a serem realizadas com os dados podem ser determinadas por operadores ou funções.

Tipos de operadores básicos:

- Operadores de atribuição
- Operadores aritméticos
- Operadores incrementais
- Operadores relacionais
- Operadores lógicos

Uma expressão é um arranjo de operadores e operandos. A cada expressão válida é atribuído um valor numérico.

Exemplo 1:

A expressão matemática 1+2 relaciona dois operandos (os números 1 e 2) por meio de um operador (+) que representa a operação de adição/soma em que o valor é 3.

Operadores de atribuição:

É a operação mais simples do C.Consiste em atribuir valor de uma expressão a uma variável.

Sintaxe: identificador=expressão;

Exemplo 2:

```
a=1;

delta=b*b-4.*a*c;

i=j;
```

Operadores aritméticos:

Estes operadores se relacionam às operações aritméticas básicas e possuem prioridade diferentes conforme a figura a seguir:

Exemplo 3:

```
area=2* PI *raio;

delta=b*b - 4.*a*c;
```

Operadores Incrementais:

Em programação existem instruções muito comuns chamadas de incremento e decremento. Um instrução de incremento adiciona uma unidade ao conteúdo de uma variável e o de decremento subtrai. Os operadores são (++) e (--) respectivamente.

Exemplo 4:

```
x=x+1;, escreve-se x++;

x=x-1;, escreve-se x--;
```

Operadores relacionais:

Em C, existe um conjunto de seis operadores relacionais, os quais podem ser usados na avaliação das expressões. Seu objectivo consiste no estabelecimento de relações entre os operandos.

Tabela 1: Operadores relacionais em C

Operador	significado
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a
!=	diferente de
==	igual a

Uma expressão que contenha um operador relacional devolve sempre como resultado o valor lógico (verdade (1) ou FALSO (0).

Exemplo 5:

```
int a = 10, b = 1, c = 12;

a > b + c;
```

O resultado da expressão acima é: Falso

Operadores lógicos:

Os operados lógicos servem para interligar mais de uma operação relacional. E assim como os relacionais retornam zero para falso e um para verdadeiro.

Tabela 2: operadores lógicos

Operador	Descrição
&&	AND (conjunção)
||	OR (disjunção)
!	NOT (operador de negação)

Exemplo 6:

- 3 > 2 e 7 > 3: resulta em verdadeiro
- 3 < 2 e 7 > 3: resulta em falso
- 3 < 2 ou 7 > 3: resulta em verdadeiro
- não(3 < 2 e 7 > 3): resulta em verdadeiro

Os operadores indicam ao compilador a necessidade de se fazer manipulações matemáticas ou lógicas.Conclui-se que os operadores são indispensáveis em nossos programas e que a linguagem C possui um número muito grande de operadores.

Detalhes da actividade

Deves ler mais sobre as funções printf (), scanf (), variável, contantes, sequência de comandos e operadores. A actividade envolve a componente prática. Após a leitura, deves escrever programas simples que irão ajudá-lo a entender os principais conceitos de programação mencionados aqui. É aconselhável usar esta referência Fundamentals of programming languages by Dipali P. Baviskarand Programming and PROB solving using C. by ISRD. Estes livros vão orientá-lo a aprender e compreender variáveis, constantes e operadores. Podes também procurar por material relevante sobre estes temas a partir da internet. Depois de ler, faça breves notas sobre os tipos de dados, variáveis, constantes e operadores.

Actividades Hands-on

1. Baixa e instala o compilador DeV C++ em seu computador (caso ainda não o tiver). Podes pedir ajuda sobre esta actividade.

2. Familiarize-se com o ambiente de linguagem C.

   - Escreva um programa simples que apresenta o seu nome.

3. Aprenda a criar, definir e declarar identificadores e a seguir:

   i. declara uma variável para armazenar o inicial do seu nome.

   ii.declara uma variável para armazenar a sua idade.

   iii.declara uma variável para armazenar a sua altura.

   iv.declara uma constante que detém participação (0,12).

   v.declara uma variável que contém um montante de capital (número real).

4. Escreva programas simples que ajuda-lhe a aprender como usar variáveis, atribuir valores a variáveis, uso de operadores e resultados de saída.

   - Usando o operador de atribuição, inicializar variáveis I a V e imprimir os valores:

5. 5. Faça uma única instrução que;

   - Lê um número inteiro x

   - Lê um double y

   - Lê um caracter

   - Imprime inteiro x

   - Imprime double y

   - Imprime caráter i

   - Incrementa x por 1

6. Usando tipos de dados apropriados, identificadores/constantes e operadores, escreva um programa em C que lê dois valores, acrescenta-lhes valores, dá o produto de dois valores, divide a soma dos dois valores por 3, Aumenta o valor obtido pela soma dois valores (usando o método de prefixo), diminui o valor obtido a partir do produto dos dois valores por 2, em seguida, emite os resultados na tela.

7. Escreva um programa que implementa o seguinte código. Mostra a saída

```
int a, b, c, d=5;

a=++d;

b=a++;

c=b--;

        printf("%d %d %d %d %d", a, b, ++c, d, --d);
```

Conclusão

Esta actividade ensinou-lhe como executar um programa usando o Dev C ++, recomendamos usar a versão mais recente. Você também conheceu as formas de definir e declarar variáveis e utilizar de forma adequada os vários tipos de dados e os operadores mencionadas nesta unidade.

Avaliação

1. Defina os seguintes termos;

   i. Tipo de dados

   ii. Variável

   iii. Operador

   iv. Identificador

   v. Caractere

   vi. Tipo de dados derivado

   vii. Arquivo de cabeçalho

2. Nomeia as regras que segues quando usas um identificador.
3. Por que é importante entender como usar os tipos de dados?
4. Várias linguagens de programação incluíndo o C suportam um número de operadores. Liste estes operadores, dê exemplos, sua associatividade e como eles são aplicados.
5. O que é um operador ternário? Mostre a sua sintaxe.
6. Lista cinco tipos de dados primários quaisquer e quatro tipos de dados secundários.

7. Qual é a diferença entre o operador de incremento e um operador de decremento?

8. Discuta os quatro tipos de constantes.

9. Descreva três maneiras de definir constantes.Usa a linguagem C, escreva um programa para ilustrar como as constantes são declaradas com a utilização de três métodos

Actividade 1.2 - Estruturas de Controle e Funções

Introdução

Um programa não se limita apenas ao fluxo sequencial de controle, porque em algum momento, um programa pode decidir usar um caminho particular de instruções ou pode repetir uma declaração um determinado número de vezes, enquanto a condição é verdadeira. Portanto, outros tipos de manipulação ou representação de controle de fluxo na programação incluem; ramificação e looping. Estruturas de ramificação são também conhecidas como estruturas condicionais ou de decisão. Estruturas de loop repetem um comando, enquanto a condição for verdadeira. Estas duas estruturas de controle são parte da programação estruturada (discutido na unidade 2. Lembre-se!) Elementos que fazem um programa mais legível e compreensível.

Estrutura de controle condicional

Ao lidar com ramificação ou estruturas condicionais também conhecidas como instruções de seleção, o programador necessita especificar uma condição que precisa ser avaliada pelo programa, juntamente com as instruções a serem executadas pelo programador. Se a condição for avaliada como verdadeira a instrução if ou bloco de instruções são executadas. Outra estrutura de seleção se if/else (ver figura A). If/else declaração avalia uma condição e se ela for verdadeira executa o / else if ou bloco de instruções, mas caso a condição for avaliada como falsa ele executa outra declaração (Veja a figura B).

Figura A - Se(if)

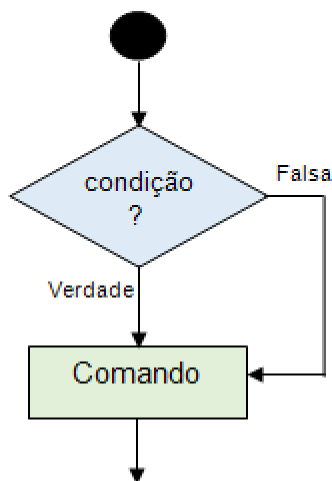
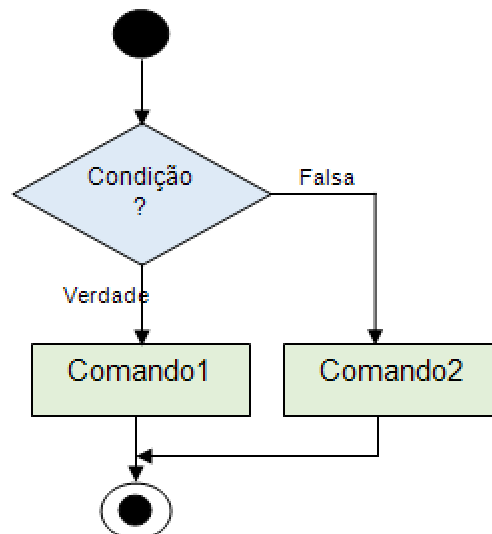


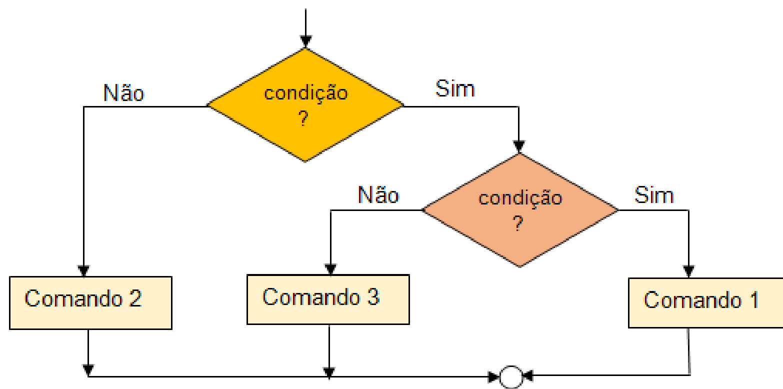
Figura B - Senão (if/else)



Ao contrário de figura A, a figura B diz o que é que acontece se a condição for falsa. Na figura A, se a condição for falsa, o controle é transferido para a instrução abaixo em dar outra opção. Há também o if aninhado if onde você pode comunicar por dentro (executado por outra instrução if), outro caso declaração por else if (ver figura C). A sua sintaxe a seguir:

```
se (condição)    {
se(condição) {
comando/s
else
comando/s
}
else
comando/s}
```

Figura C - if aninhados



Comando switch é uma estrutura de decisão multi-controlo que permite que os valores sejam testados de uma lista de valores até encontrar uma correspondência que retorna resultados. Esta instrução é outro modo para simular o uso de varias instruções if e pode somente verificar uma relação de igualdade. Todos os tipos primitivos (int, string, decimal, etc) podem ser usados nas instruções switch/case. Sua sintaxe:

```
switch(expressao) {
case 1:
case 2:
comando/s;
break;
comando/s;
break;


.default:
comando/s;
}
```

O operador?: (Operador condicional). Este operador opera como o if / else. Fizemo-lo na primeira actividade desta unidade. A sintaxe é como mostrado abaixo:

exp 1 ? exp 2 : exp 3;

Se exp 1 é verdadeiro o exp 2 é executado, mas se exp 1 for falsa, então exp 3 é executado.

Nota: if não é encerrado com (;) porque ele está executando a instrução/s abaixo dela.
Além disso, se a declaração for executar mais do que uma instrução, então é importante apresentá-la como um bloco de código usando o {}. Por exemplo:

```
if(condição)
{
comando;
comando;
}
```

Loops/estruturas de controle de repetição.

Há situações em que uma instrução precisa ser executada repetidamente, ou um certo número de vezes que desejar, execução loop. Os seguintes ciclos de instruções podem ser usados para executar repetidamente uma instrução.

Comando While: o valor de condição é calculado como verdadeiro ou falso.

Se a condição for verdadeira o comando é executado e se a condição for falsa, então o while é encerrado. Veja a Figura D. A sintaxe:

```
while(condição)
{
comando/s;
}
```

Por exemplo:

```
#include<stdio.h>
int main() {
int x=1;
while( x<=10)
printf("Facil de usar!");
}
```

Comando For: o laço for é uma estrutura de repetição muito utilizada nos programas em C.É muito útil quando se sabe de antemão quantas vezes a repetição deverá ser executada. Este laço utiliza uma variável para controlar a contagem do loop, bem como seu incremento.Veja a Figura E. A sintaxe:

```
for(valor_inicial; condição_final; valor_incremento)
 {
   comando/s;
 }
```

Por exemplo:

```
#include<stdio.h>

int main() {

for(int x=1; x<=10; ++x)

printf("Quase como While!");

}
```

Comando Do While: no comando do…while o bloco de comandos é executado pelo menos uma vez de forma obrigatória, independente do resultado da expressão lógica.Veja a Figura F. A sintaxe:

```
do {

 comando/s

}

Por exemplo:

#include<stdio.h>

int main() {

int x=1;

    do   {

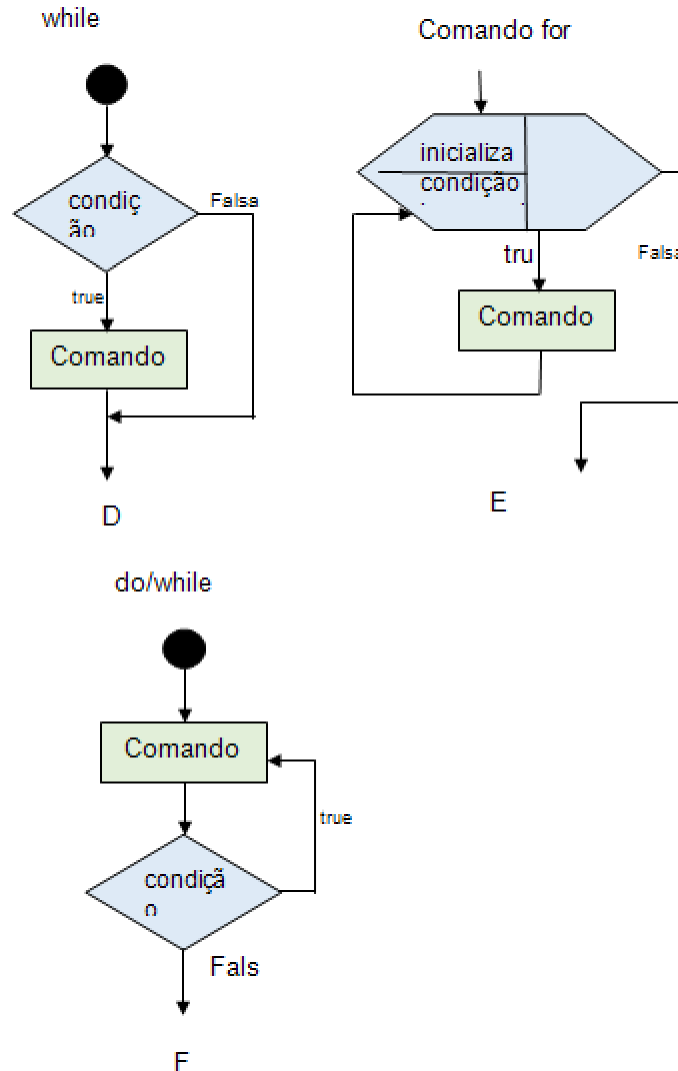
printf("%d", x);

++x;

} while( x<=10);

printf(facil de usar!");

}
```



Nota:

Há uma pequena diferença entre os comandos "Do" e "Do/while". O comando "Do" testa primeiro a condição antes de executar. Enquanto o "Do/while" executa as instruções pelo menos uma vez antes de avaliar a condição - o que significa que retorna um valor pelo menos uma vez, se a condição é verdadeira ou falsa. Mais uma vez, o "Do/while" é encerrado; simplesmente porque não executa as instruções abaixo dele.

Observação geral (! Fácil de usar !, quase como while e fácil de usar) todas as instruções abaixo de loop são impressos se a condição for verdadeira ou falsa.

Outras Instruções

Há duas outras instruções importantes na programação. Estes são; instrução de break (pausa) e instrução continue (continuar). A instrução break termina um loop enquanto que a instrução continue sai do loop atual, em seguida, executa o próximo ciclo ou iteração.

Funções

Na unidade anterior, discutimos a resolução de problemas que incluem as técnicas de top down e bottom up. Estas técnicas envolvem a divisão de um programa em sub-programas ou sub-rotinas e mais tarde todos os sub-programas são integrados para formar um programa. Você ainda se lembra dessas técnicas? Se não, re-le a unidade 2.

De qualquer modo, estas técnicas formam a programação modular ou criação de funções. Agora, podemos dizer que uma função é um grupo de instruções que executam uma determinada tarefa. Em termos simples, todas as instruções em um grupo específico trabalham juntos para alcançar um objectivo.

A maioria das linguagens de programação têm pelo menos uma função conhecida como função main () e este identifica onde a execução do programa começa.

Os programadores também são capazes de inventar ou acrescentar mais funções aos seus programas. Sempre que um programa encontra um nome de uma função, o controle é então passado para essa função específica, cujo nome foi encontrado. A função tem a seguinte forma:

```
tipo_retorno  nome-função(lista de parâmetros) //cabeçalho da
função

{

comando/s; //corpo ou definição da função

}
```

Tipo de retorno, define o tipo do valor que a função retorna dados. Outras funções não devolvem qualquer valor após a execução de uma instrução. Essas funções são definidas com o tipo de retorno usando a palavra chave "void". Assim como um identificador, nome da função é o nome real da função que é usada por outros programas sempre que eles precisam de serviços. Um parâmetro é um espaço reservado para os valores que são passados a partir do ambiente de entrada. Outras funções podem não possuir parâmetros e a lista de parâmetros pode ser definida como nula. Isto é, não está a receber os valores de outros programas. O corpo da função contém instruções que definem o que a função faz.

Por exemplo, uma função que retorna saudação para outra função;

```
void saudação (void)

{

printf ("Bom dia!");

}
```

Quando outra função precisa de uma instrução de saída ("Bom dia!"), então o programador inclui o nome da função (saudação ()), a fim de invocar ou chamar essa função. A função que chama outra função forma o ambiente de chamada. Agora veja o programa abaixo:

```
#include<stdio.h>

void      saudação(void)

{

printf ("Bom dia!");

}

main() {

saudacao();   //chamada da função

}
```

Nota:

Nós definimos a nossa função (saudação ()) antes de nossa função principal. Isso ocorre porque o compilador não será capaz de vê-lo se foi colocado depois que a função principal. Para evitar restringir-se ao local para definir uma função, você pode introduzir um protótipo de função. Um protótipo da função declara uma função e informa o compilador que existe uma determinada função no programa. Ele dá os detalhes dessa função; volor, tipo, nome e parâmetros.

```
#include<stdio.h>

void      saudação(void);   //declaração da função (prototipo)

main() {

saudacao();   //chamada da função

}

void      saudação(void)

{

printf ("Bom dia!");

}
```

Escopo de regras

Variáveis podem ser declaradas como variáveis locais ou globais. As variáveis locais são definidas dentro o do bloco de código de uma função particular. Essas variáveis podem ser usadas apenas por declarações que estão dentro dessa função ou bloco de código, simplesmente porque eles não são conhecidos por variáveis fora das funções. Variáveis globais são definidas fora de todas as outras funções do programa. Como convenção, elas são declaradas na parte superior do programa após a directiva de pré-processamento, mas antes que a função main (). Eles detêm valores ao longo da vigência do programa, e podem ser acessados por outras funções que estão definidas para o programa. Vamos aprender o escopo de variáveis usando este exemplo obtido no tutorial do site;

```
#include <stdio.h>
/* declaração de variável global */
int a = 20;
int main ()
{
 /* declaração de variável local na função main */
  int a = 10;
  int b = 20;
  int c = 0;
  printf (“valor de a no main() = %d\n”,  a);
  c = sum( a, b); //parâmetros actuais (a, b)
  printf (“valor de c no main() = %d\n”,  c);
  return 0;
}
/* função para somar dois inteiros */
int sum(int a, int b)   //parâmetros formais (int a, int b)
{
    printf (“valor de a em sum() = %d\n”,  a);
    printf (“valor de b em sum() = %d\n”,  b);
    return a + b;
}
```

Detalhes da atividade

Assim como na primeira actividade, nesta unidade, você é obrigado a ler mais sobre estruturas de controle e funções. Esta actividade também envolve prática. Após a leitura, você vai ser obrigado a escrever programas simples que irá ajudá-lo a entender os principais conceitos de programação. É aconselhável usar Fundamentals of programming languages by Dipali P. Baviskar, Programming and PROB solving using C. by ISRD and Programming and problem solving through 'C' languages by Harsha Priya, r. Ranjeet.

Podes também obter informações relevantes a partir da internet. Por exemplo; http://www.tutorialspoint.com/cprogramming/index.htm. Depois de ler os referidos temas, esta actividade que exige você:

Escreva notas breves que descrevem a diferença entre estruturas de ramificação de controle e estruturas repetitivasl.

Descrever a importância das estruturas de ramificação e de looping.

De acordo com seu próprio entendimento, dando exemplos e, provavelmente, ilustrando, explica as instruções de salto e suas aplicações. Discute a importância de aprender e entender como usar estas declarações.

Faça breves notas sobre as vantagens de se usar funções em um programa.

Hands-on atividade (prática)

Usando um programa(de preferência Microsoft Word), desenha um fluxograma para executar os passos indicados nas demonstrações de 1 a X, em seguida, dar a declaração C correspondente. Suponha que sejam devidamente declaradas.

1. Se a idade é igual a 20, idade incremento por 1. Output "no próximo ano você vai ter___" (saída da idade incrementado na linha em branco).
2. Se a nota for maior que 70, imprimir a declaração de "excelente".
3. Se a nota for menor ou igual a 60, imprimir a declaração "Média", caso contrário, a saída "Get serious nyana!".
4. Se a altura é maior ou igual a 53,9 metros, peso decremento por 20,4 kg.

   Caso contrário imprimir "manter o peso em 55 kg".
5. Enquanto x é menor ou igual a 15, imprima a soma dos valores.
6. Converter declaração 5 para uma instrução do.
7. Converter declaração 5 a um do / while
8. Faça programas simples para execução do projeto (fluxogramas 1-7), assumir que são obrigados a declará-los.
9. Além disso, escreva um programa que lê em um pequeno número inteiro n, passa para uma função que repete n vezes retornando "Deus é bom".

Conclusão

Esta é uma unidade muito interessante onde você aprendeu a implementar alguns dos princípios básicos de programação. Até agora você deve ter apreciado os conceitos de programação, como tipos de dados, variáveis, operadores, estruturas de controle e funções. Estas construções são enfatizadas porque são comuns à maioria das linguagens de programação.

Avaliação

1. Mostra a forma geral para:

   i. if

   ii. if / else

   iii. if aninhado

   iv. if / else aninhados

   v. declaração condicional

   vi. switch

   vii. while

   viii. do/ while

   ix. uma instrução

   x. uma função

2. Em algumas situações, as estruturas de controle executam mais do que uma instrução. Por que é importante bloquear instruções ao executar mais de uma instrução?
3. Explica a diferença entre os comandos do / while e while.
4. Normalmente, estruturas de controle por exemplo, se (x> y) não são terminados no final. Por quê?
5. Qual é a diferença entre um parâmetro formal e um parâmetro real?
6. O que é um protótipo de função? Qual é o seu significado em um programa?
7. Diferencia as variáveis locais e globais.
8. Quais são algumas das vantagens do uso de variável global num programa?
9. O que é um cabeçalho da função.
10. Dê as vantagens de usar funções em um programa.

Actividade 1.3 - Vectores(array) e strings

Introdução

Vector (Array) é uma estrutura de dados que armazena os itens de dados do mesmo tipo. Por exemplo, para armazenar cinco inteiros na memória, somos obrigados a declarar cinco variáveis do tipo int (por exemplo, int a, b, c, d, e); mas um array vem a calhar para este caso. Usando um array, criando um identificador, em seguida, definir o número de elementos ou o tamanho do vector. Assim como todas as outras variáveis, um vector deve ser declarado antes do uso. Sua forma geral de uso:

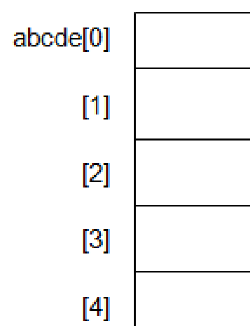
```
tipo nome_vector[tamanho];
```

O tipo especifica o tipo de elementos que serão armazenados no vector que inclui dados; int, float, char, double e.t.c. Enquanto o tamanho indica o número máximo de elementos que podem ser armazenados no vector. Por exemplo:

```
int abcde[5];

float balanca[10];
```

Elementos do vector são acessados usando o nome do vector e seu índice. Elementos do vector são armazenados de um modo contíguo e o primeiro elemento do vector começa com um índice zero (0), o último elemento do vector termina com um tamanho do elemento menos um ([elemento-1]). No nosso exemplo, o primeiro elemento é abcde [0] .... [4]. Veja a figura abaixo:



Uma matriz pode ser inicializada ou valores atribuídos como;

Tipo de nome array [] = {} valores;

Por examplo;

```
int abcde[5] = {10, 20, 30, 40, 50};

or;

int abcde[5];

abcde[0]=10;

abcde[1]=20;

abcde[2]=30;
```

```
abcde[3]=40;

abcde[4]=50;
```

Veja a figura;

abcde[0]	10
[1]	20
[2]	30
[3]	40
[4]	50

Este comando é usado para manipular o conteúdo de um vector. Para ilustrar isso, vamos escrever este programa simples;

```
#include<stdio.h>

    main ()

{

int abcde[5]={10, 20, 30,40,50}, x;

for(x=0; x<5; ++x)  {

printf(“abcde[%d] =%d\n”, x, abcde[x]);

}

printf(“imprimi os elementos do meu vector\n”);

}
```

Saída:

```
abcde[0]=10;

abcde[1]=20;

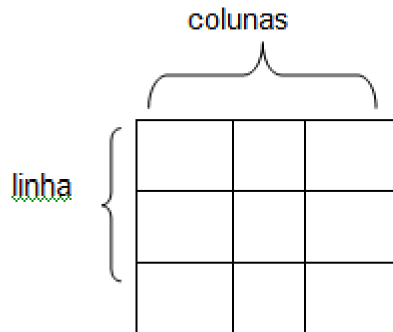
abcde[2]=30;

abcde[3]=40;

abcde[4]=50;
```

Aqui, x é a nossa variável de controle que usamos para acessar ou manipular o conteúdo do array.

Temos outros tipos de vectores conhecidos como matrizes multidimensionais. Matriz multidimensional é um array de arrays. Por exemplo, uma matriz bidimensional é uma matriz multidimensional que tenta representar elementos e informar sobre as mesmas. Ela tem dois índices; um para a linha e a outra para a coluna. Por exemplo, a seguir é uma tabela de três por três para armazenar nove valores (3x3 = 9).



Forma geral de uma matriz bidimensional;

```
tipo_dados  nome_matriz[tamanho_linha];
```

Por exemplo, my_table é declarada como uma matriz com 4 linhas e 5 colunas. O primeiro elemento da matriz é variavel [0] [0] e último elemento é my_table [3] [4]. AFigura abaixo ilustra uma matriz bidimensional;

marca	0	1	2	3	4
0					
1					
2					
3					

Como aquela matriz dimensional acima, esta matriz pode inicializada como;

```
tipo_dados  nome_vector[ ]={valores}
```

Por exemplo;

```
float marcas[20]={45.8, 34.2,56.0..............78.5};
```

marcas	0	1	2	3	4
0	45.8	34.2	56.0	66.7	76.8
1	78.5	54.3	49.5	52.4	36.0
2	55.5	54.5	44.3	66.5	80.5
3	90.5	77.5	48.0	56.6	78.5

Um programa simples ilustrata a matriz dimensional

```
#include<stdio.h>
                        main ()
{
float markcas
[20]={45.8,34.2,55.0,66.7,76.8,78.5,54.3,49.5,52.4,36.0,55.5,54.5,44.3,
66.5,80.5,90.5,77.5,48.0,56.6,78.5}, x,y;
for(x=0; x<5; ++x)
for(y=0; y<5; ++y)    {
printf("marcas[%d][%d] =%d\n", x,y,marcas[x][y]);
}
printf("Imprimi os dois elementos da matriz\n");
}
```

Strings

Uma string é um vector de caracteres que podem ser terminados com um valor nulo '\0'. Aqui nós definimos uma string usando duas palavras que temos que encontrou nesta unidade; como uma matriz e caráter (char). Uma string é declarada usando o mesmo formato de um vector. Aqui:

tipo nome_string[tamanho];

O tipo é char porque é um array de caracteres, o nome indica o identificador que marca o local na memória para armazenar os caracteres. Tamanho contém o número de caracteres que podem ser contidos na string. Por exemplo;

"Elisandra" é uma cadeia fechada em aspas duplas, enquanto 'y' é um colocada em aspas simples. A string tem a seguinte forma:

char primeiro_nome[10];

char ultimo_nome[10];

A string pode ser inicializada como:

char primeiro_nome[10]= {'E', 'l', 'i', 's', 'a', 'n', 'd', 'r', 'a','\0'};

ou;

char primeiro_nome[10]="Elisandra";

Por exemplo;

```
#include <stdio.h>
int main ()
{
char primeiro_nome[10]= {'E', 'l', 'i', 's', 'a', 'n', 'd', 'r',
'a','\0'};
char ultimo_nome[10]="alves";
printf("Ola! : %s %s\n", primeiro_nome, char ultimo_nome);
   return 0;
}
```

Detalhes de actividade

Deves ler mais sobre vectores e strings. Após a leitura, você vai ser obrigado a escrever programas que ilustram o conceito de vectores e strings. Aconselha-se a usar Fundamentals of programming languages by Dipali P. Baviskar, Programming and PROB solving using C. by ISRD and Programming and problem solving through 'C' languages by Harsha Priya, r. Ranjeet. You can also obtain relevant information from the internet. Por exemplo; http://www.tutorialspoint.com/cprogramming/index.htm . Esta atividade exige que você faça uma pequena pesquisa. Lê os referidos temas, esta actividade requer, então, que você:

- Escreva um programa que localiza a soma dos valores armazenados em uma matriz unidimensional (vector). Mostra o tamanho do vector, o tipo de dados e valores.
- Escreva um programa que leia os valores para uma matriz bidimensional, em seguida, encontra a média. Cria a sua própria matriz, indicando o tipo de matriz, o tamanho e os seus valores. Imprima todos os elementos da matriz.
- Algumas vezes, é necessário limitar as matrizes de uso multidimensionais em nossos programas. (efeitos de matrizes multidimensional em nosso sistema). Investiga sobre este tópico.

Conclusão

Concluímos esta unidade com dois conceitos importantes na programação; vectores e strings. Como você pode ver, estas duas construções são quase inevitavelmente, ao desenvolver um programa ou projecto de uma solução. Nós sempre lidamos com uma série de itens de dados, provavelmente do mesmo tipo e dois usamos strings. Portanto, é importante que você entenda esses conceitos. Por fim, a prática vai ajudá-lo a apreciar a programação.

Avaliação

1. Define os seguintes termos:

   i. Um vector

   ii. Uma string

2. Por que nós usamos uma instrução aninhada quando se lida com uma matriz bidimensional?

3. O que está errado com a seguinte declaração:

   nome char [5] = {'n', 'o', 'e', 'l', 'a'};. Explique a tua resposta.

Resumo da Unidade

Esta unidade apresentou a parte teória e actividades hands-on de princípios de programação. O estudante foi introduzido para a resolução de problemas utilizando a linguagem C. Portanto, a unidade começou com ambiente C incluindo programas simples e o uso das funções de printf () e scanf (). Também focou os tipos de dados, as variáveis, as constantes, os operadores, as estruturas de controle, as funções, vectores e strings. Até agora, o aluno deve ser capaz de escrever programas usando todas as construções que foram ensinados nesta unidade.

Avaliação da Unidade

Verifica a sua compreensão

Avaliação formativa

1. Declara e inicializa uma variável inteira chamada total a 0.
2. Declara e inicializa um carácter chamado inicial para a letra K.
3. Usa uma constante para declarar uma constante PI e atribuir o valor 3,14.
4. Defina a constante PI que tem o valor de 3,14.
5. Atribua o valor da variável inteira para uma variável b.
6. Guarda a soma de duas variáveis C e E na variável soma.
7. Adiciona o valor de a e o valor b e coloca a resposta em a.
8. Divida o valor da soma por 9 e guarda a resposta na variável suma.
9. Leia valor inteiro para y.
10. Leia um carácter para c.
11. Escreva um de comando que compara o valor de um inteiro chamado produto com o valor de 15, e se ele é maior, imprima a sequência de texto "você excedeu".
12. Escreva um loop while para imprimir os valores de 1 a 15 na tela.
13. Escreva uma instrução loop para imprimir os valores 3 a 7.
14. Escreva uma instrução de loop que adiciona todos os valores entre 50 e 100 em uma variável chamada soma.
15. Escreva uma instrução que compara o valor de um inteiro chamado produto com o valor de 15, e se ele é maior, imprima a sequência de texto "você supera".
16. Se a variável x é igual a y, imprima o valor de x, senão imprima o valor de y.
17. Reescreva o código a seguir usando o comando switch.

```
if( c == 'A' )

sum = 0;

else if ( c == 'B' )

sum+= 1;

else if( c == 'Z' )

sum = 26;

else printf("caracter  desconhecido %c\n", c );
```

18. Converte a seguinte expressão para o comando if/else.

```
v=(a<b) ? a : b;
```

19. Declara um vector dimensional chamado limites que armazena seis inteiros.
20. Atribua 23,46,67,87,53,44 para os elementos do vector limites .
21. Atribua o valor 56 para o quarto elemento do vector limites.
22. Atribua o valor 34 para o quinto elemento do vector limites.
23. Impressão o primeiro elemento do vector.
24. Declara uma matriz baseada em caracteres chamado nomes de 20 elementos.
25. Atribua o carácter 'S' para o elemento 10 do vector.
26. Faça um comando para ler valores em um vector, limite (19).
27. Faça um comando para a leitura de um vector nome, limite (24).

Instruções

Você é obrigado a escrever instruções individuais para todas as perguntas de 1 a 27.

A perguntas cobrem tudo o que foi apresentado nesta unidade, a fim de avaliar a compreensão global. Responde-as com cuidado e se a sua pontuação estiver:

abaixo de 40%, refazer as leituras.

entre 40% e 60%, refazer as leituras em seu ponto fraco.

acima de 60%, você tem uma quantidade substancial de conhecimentos.

Avaliação

Critérios de Avaliação

Exercícios 20%

Teste de avaliação 10%

Exame final 70%

Feedback

- Na sua opinião, a unidade está bem desenvolvida?
- Quais são as áreas que achas necessário mais clareza?
- Houve termos que precisavam de mais explicações?
- Que sugestões você daria para melhorar o conteúdo?
- Você acha que exercícios (avaliação), foram adequados?

Respostas da avaliação formativa

1. `int total=0;`
2. `char initial='K';`
3. `float const PI = 3.14;`
4. `#define PI 3.14`
5.
```
int a, b;
b=a;
```
6. `sum=c+e;`
7. `a+=b; or a=a+b;`
8. `sum/=9;`
9. `scanf("%d", &y);`
10. `scanf("%c", &c);`
11. `balances=(float) total/5;`
12.
```
int values=1;
while (values<=15) {
printf("%d\n", values);
++values  }
```
13. `for(int values=3; values<=7; ++values)`

14. ```
for(int values=51, sum=0; values<100; sum+=values, ++values) or
int sum=0, values;
for(values=51; values<100; ++values)
sum+=values;
```

15. ```
if(product>15)
printf("you exceeded");
```

16. ```
if(x==y)
printf("%d", x);
else
printf("%d", y);
```

17. ```
switch (c) {
case 'A':
            sum = 0;
            break;
  case 'B':
            sum+= 1;
            break;
  case 'C':
            sum = 26;
            break;
  default:
                 printf("caracter  desconhecido %c\n", c
);   }
```

18. ```
if (a<b)
   v=a;
else
  v=b;
```

19. ```
int limits [6];
```

20. ```
limits [6]= {23,46,67,87,53,44};
```

21. ```
limits[4]= 56;
```

22. ```
limits[4]=34;
```
```

23. `printf("%d",limits[0]);`

24. `char names [20];`

25. `names[10]='S';`

26. `for (x=0; x<6; ++x)`

    `scanf("%d" &limits[x]);`

27. `scanf("%s", names);`

Leituras e outros Recursos

As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.

Unidade 4. Métodos de programação e princípios de programação modular

Introdução à Unidade

Esta unidade irá fornecer uma visão geral dos dois principais métodos de programação que são top-down e bottom-up. Em seguida, apresentamos a programação modular em C usando as funções. Vários exemplos são apresentados para ilustrar os diferentes conceitos.

Objectivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Explicar as técnicas utilizadas na programação estruturada
2. Desenvolver programas de computador usando um dos métodos de programação estruturada
3. Explicar os benefícios da programação modular
4. Implementar os vários módulos de um programa como funções ou grupos de funções C

Termos-chave

Programação modular: permite dividir uma grande aplicação em módulos, a fim de desenvolver de forma independente e reutilizada em outras aplicações.

programação top-down: é uma abordagem de programação que permite fazer um programa partindo do geral para o particular.

Programação bottom-up: abordagem contrária de top-down, o programa é feito do particular para o geral.

Função: as funções são partes do código-fonte que realizam o mesmo tipo de tratamento várias vezes ou em diferentes objetos.

Variável global: é uma variável comum definida fora de uma função e acessível a todas as funções.

Função recursiva: função que chama a sí mesma.

Actividades de Aprendizagem

Actividade 1.1 - Os métodos de programação

Introdução

Um programa de computador é projectado para resolver um problema específico após a execução de um número finito de instruções. No entanto, o computador não resolve os problemas sem os humanos. De facto, os seres humanos através da sua inteligência e intuição utilizam o computador para resolver problemas específicos, seguindo as instruções fornecidas pelos programadores. Esta unidade vai permitir-nos compreender os métodos básicos de "programação estruturada". O projecto de programação estruturada é uma ferramenta de programação desenvolvida na década de 1960 como um meio de definir os elementos de um problema e agora parece ser a melhor abordagem para definir as tarefas de programação de computadores.

Detalhes de actividade

O desenvolvimento de grandes programas informáticos exige que uma determinada altura haja um processo de decomposição de problemas ou tarefas complexas e muito abrangentes em tarefas mais pequenas e específicas, isto é, mais fáceis de lidar.

Esta técnica pode-se aplicar várias vezes de modo a se obter uma espécie de hierarquia ou árvore de tarefas em que na raíz reside o problema ou tarefa inicial e nas folhas as tarefas mais simples e implementáveis em sub-rotinas.

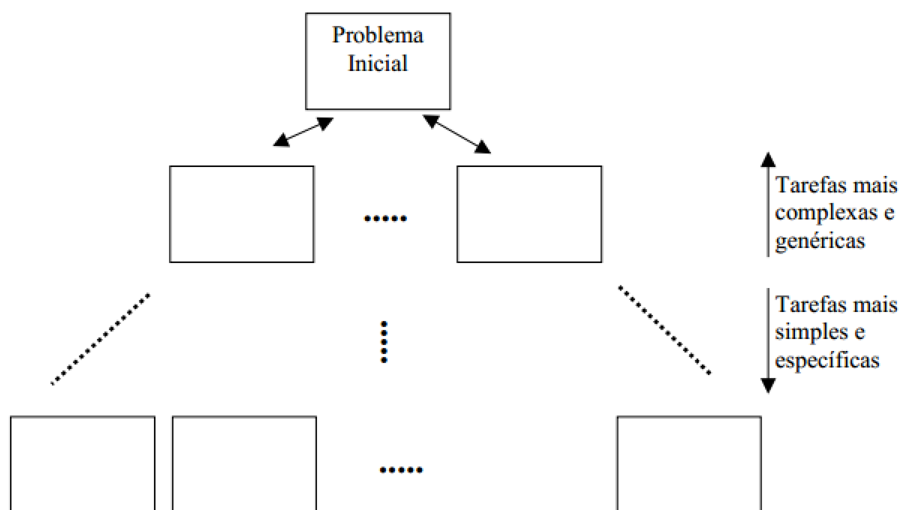


Figura 1: Ilustração decomposição de tarefas

O princípio de programação estruturada é a decompor-se um módulo de programa. Além de uma boa decomposição, este princípio proporciona um bom guia ao escrever o programa desde as estruturas de programação estruturada simples a mais complexas. Estas estruturas nomeadamente sequência, seleção e iteracção já foram abordadas durante o desenvolvimento da unidade 2. Esta unidade, portanto, incidirá sobre os métodos utilizados para definir e codificar os diferentes módulos de um programa.

Nesta atividade, serão apresentados os dois métodos usados no desenvolvimento de um programa estruturado. São os seguintes:

- A abordagem top-down (do geral para o específico)
- A abordagem bottom-up (do específico para o geral)

Programação descendente ou top-down (do geral para o específico)

Esta metodologia permite decompor um problema em partes mais pequenas claramente definidas e com mecanismos de interacção claros e bem limitados. A construção ou desenvolvimento do programa é feita implementando cada uma das partes.

Tarefas genérica/complexas->Tarefas específicas/simples

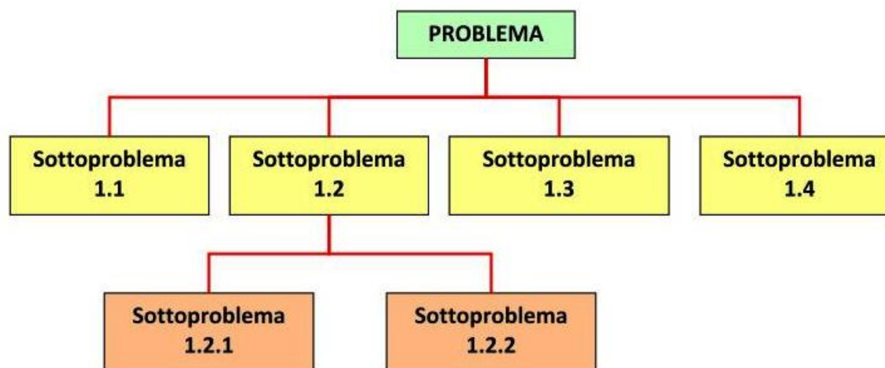


Figura 2: Ilustração Top-Down

Vantagens método top-down:

- Normalmente é usada na fase inicial do projecto
- Mais rápido para ser informado ao cliente
- Útil para medir o interesse do cliente no projecto
- Utilizada em projecto quando não temos informações detalhadas

Desvantagens método top-down:

- Baseado em experiências anteriores
- Deve ser realizado por uma especializada
- Possuir grande risco ao estimar o custo do projecto

Programação ascendente ou bottom-up (do específico para o geral)

A programação buttom-up usa a lógica contária do top-down, isto é, identifica num programa complexo, tarefas bem definidas e que forma reconhecidas como necessárias ao desenvolvimento. Depois da implementação dessas partes o programador irá reuní-las de modo a obter a solução para o problema mais complexo.

Tarefas específicas/simples->Tarefas genéricas/complexas

De um modo geral o desenvolvimento de um programa poderá adoptar uma abordagem mista "top-down" e "bottom-up".

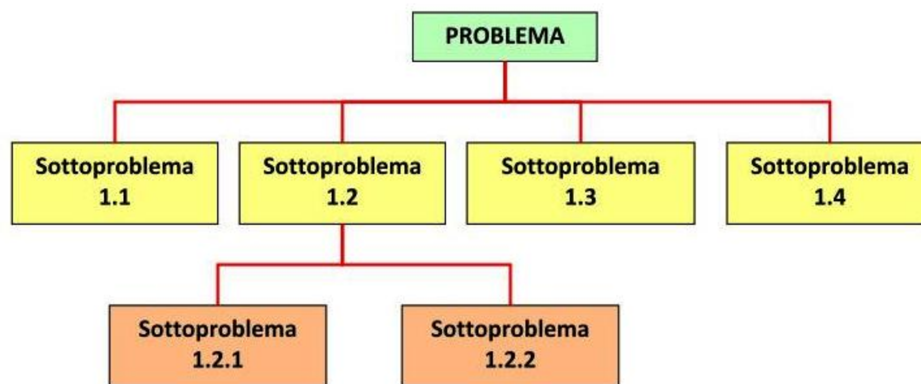


Figura 2: Ilustração Bottom-up

Vantagens método bottom-down:

- Maior precisão
- Desenvolve o comprometimento da equipa
- Baseado na análise detalhado do projeto
- Fornece a base para monitoramento e controlo do projecto

Desvantagens método bottom-down:

- Maior esforço de tempo, custo e recursos para desenvolver a estimativa
- Requer que o projecto esteja bem definido e entendido

Conclusão

Esta actividade apresentou dois métodos comuns para decompor um sistema em módulos. Foram apresentadas as principais características de cada um dos métodos. As vantagens e desvantagens de cada método também foram destacadas.De um modo geral o desenvolvimento de um programa poderá adoptar uma abordagem mista, ou seja, utilizam as duas abordagens "top-down" e "bottom-up".

Avaliação

Exercício:

1. Faça uma comparação da abordagem top-down e bottom-up?
2. Apresenta duas vantagens e duas desvantagens de cada um desses métodos de programação?

Actividade 1.2 - Programação modular em C

Introdução

É fundamental que um programador de C domine completamente a escrita de programas de forma modular, através de procedimentos e funções .

Neste actividade vamos aprender como escrever uma função, como passar parâmentros e como devolver algum valor como resultado.

Detalhes da actividade

O que é programação modular?

A programação modular permite a divisão de um programa em módulos (subprogramas), cada um dos quais pode ser desenvolvido separada e independentemente dos outros, podendo estes módulos, posteriormente, serem integrados num único programa.

As linguagens de alto nível fornecem dois processos distintos de produzir módulos, dando um deles origem à criação de funções e o outro à criação de procedimentos.

Como vantagens da sua utilização temos:

- Programas mais fáceis de escrever
- Programas mais fáceis de ler
- Programas, em geral, mais curtos
- Programas mais fáceis de modificar
- Abstracção
- Reutilização

Existem dois tipos de subprogramas:

- Os que retornam um valor (funções)
- Os que executam acções (procedimentos)

Definição de uma função

Conjunto de comandos agrupados em um bloco que recebe um nome e através deste pode ser activado.

Um programa em C é constituído por uma sequência de funções. Uma função contém instruções que especificam as operações de computação a executar. A execução de um programa começa numa função de nome main.

Uma função é composta por um cabeçalho, seguido do respectivo corpo entre chavetas.

A sintaxe de uma função é a seguinte:

```
tipo_retornado  nome_função (declarações de parâmetros)
{
declarações locais
instruções
}
```

Exemplo:

```
int media(n1,n2,n3,n4)
{
double aux;
aux=(n1+n2+n3+n4)/4.0;
return (int)aux; /*  chamada da funçao aux */
}
```

Passagem de parâmentros a uma função

A comunicação com uma funcão se faz através de argumentos que lhe são enviados e dos parâmetros presentes na função que os recebe.

O número de parâmetros de uma função pode ser 0,1,2,etc dependendo apenas das necessidades do programador. Cada função necessita, no entanto, saber qual o tipo de cada um dos parâmetros.

Os parametros de uma função são separados por vírgula, e é absolutamente necessário que para cada um deles seja indicado o seu tipo.

Exemplo:

```
funcao (int x,  char y,  float k)    /* exemplo correcto*/
```

Exemplo:

```
funcao (int x,  y,  k)    /* exemplo incorrecto*/
```

A linguagem C, suporta dois tipos de passagem de parâmetros formais:

- Passagem por valor
- Passagem por referência

Passagem por valor:

A função recebe uma cópia da variável que é fornecida quando é invocada. Todas as alterações feitas dentro da função não vão afectar os valores originais.

Exemplo: Função troca que recebe dois valores a e b, e troca os respectivos conteúdos.

```
# include<stdio.h>
void troca (int a, int b)
{
int temp;
temp=a;
a=b;
b=temp;
}
int main()
{
int a=2, b=3;
printf("Antes de chamar a função :\na=%d\nb=%d\n", a, b);
troca(a,b);
printf("Depois de chamar a função :\na=%d\nb=%d\n", a, b);
return 0;
}
```

Apesar da troca dentro da função, os valores originais não sofreram alterações.

Passagem por referência:

Neste caso o que é enviado para a função é uma referência às variáveis utilizadas, e não uma simples cópia, pelo que as alterações realizadas dentro da função irão certamente alterar os valores contidos nessas variáveis.

Exemplo: Função troca que recebe dois valores a e b, e troca os respectivos conteúdos. Em C só existe a passagem de parâmetros por valor (obrigatório o uso de apontadores, capítulo a ser introduzido no módulo seguinte).

```
# include<iostream>

void troca (int *a, int *b)

{

 int temp;

 temp=*a;

 a*=*b;

 b*=temp;

}

int main ()

{

int a=2, b=3;

printf("Antes de chamar a função :\na=%d\nb=%d\n", a, b);

troca(&a,&b);

printf("Depois de chamar a função :\na=%d\nb=%d\n", a, b);

return 0;

}
```

A troca dentro da função reflecte-se nos valores originais.

O corpo da função

O corpo de uma função é constituído por instruções de C de acordo com a sintaxe da linguagem. Tem que se seguir imediatamente ao cabeçalho da função, e é escrito entre chaves.

O cabeçalho de uma função NUNCA deve ser seguido de ponto-e-vírgula(;)

Sempre que uma função é invocada pelo programa, o corpo da função é executado, instrução a instrução, até terminar o corpo da função ou até encontrar a instrução return, voltando imediatamente ao programa que foi invocado.

O número de instruções que pode estar presente dentro de uma função não tem qualquer limite, deve, no entanto, ser relativamente pequeno e responsável por realizar uma única tarefa.

Instrução return

A instrução return permite terminar a execução de uma função e voltar ao programa que a invocou. A execução desta instrução na função main faz com que o programa termine.

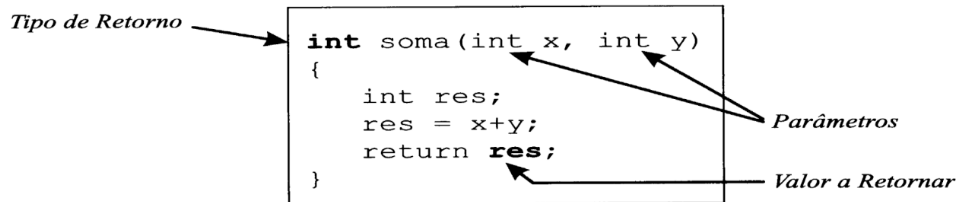
Exemplo:

```
#include <stdio.h>
main()
{
printf(“Hello”);     /*Escrever Hello
return;                /*Termina a função (e o programa)*/
printf(“World”);     /*Esta linha nunca é executada */
}
```

Funções que retornam um valor

É possível que uma função seja responsável por realizar uma determinada tarefa e que, uma vez terminada essa tarefa, devolva UM ÚNICO resultado. Esse resultado poderá ser armazenado numa variável ou aproveitado por qualquer instrução. A devolução de um resultado é feita através da instrução return, seguida do valor a ser devolvido.

Exemplo: a seguir uma função soma que recebe dois inteiros e terá que devolver um resultado do tipo inteiro que corresponderá à soma dos dois parâmetros recebidos pela função:



Exemplo: Função que calcula o maior de dois inteiros:

```
int max (int n1, int n2)

{

if (n1>n2)

return n1;

else

return n2;

}
```

Procedimentos - o tipo void

Qual a diferença entre uma função e procedimento?

Uma função tem sempre um tipo e um valor de retorno associados, enquanto um procedimento não devolve qualquer valor.

Procedimentos são estruturas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado. Na verdade, procedimentos são subprogramas que são executados quando o programa principal invoca o procedimento.

Em C, existe a palavra reservada, void, que permite indicar que uma função não devolve qualquer tipo.

Exemplo de procedimento em C:

```
#include < stdio . h>
void Ola ( ) /*declaraçao da função */
{
printf ("\n Ola ! ! ! \ n" ) ; /*Corpo da função */
}
/*-------- Programa P r i n c i p a l --------- */
int main ()
{
printf ("\n Programa Principal ! ! " ) ;
Ola ( ) ; /*Chamada da função */
printf ("\n Posso chamar a funcao de novo : " ) ;
Ola ( ) ; /*Chamada da função */
Ola ( ) ;
printf ("\n Final do Programa ! ! " ) ;
return 0 ;
}
```

Variáveis Globais

Uma variável global, a sua definição deve aparecer do lado de fora das funções que utilizam e deve preceder a sua definição. Diz-se que o seu âmbito é limitado à porção do programa seguinte a sua definição.

A definição de uma variável faz com que a atribuição de uma memória permanente para esta variável e, possivelmente, inicializar o seu valor. Quando o valor dessa variável é alterado no corpo de uma função, o seu novo valor será conhecido para todas as funções que o utilizam.

Variáveis Locais

As variáveis podem ser declaradas dentro do corpo de uma função. Essas variáveis são visíveis (isto é, conhecidas) apenas dentro da própria função, por isso são denominadas de variáveis locais.

A sua declaração deve ser realizada antes de qualquer instrução.

```
função (...)

{

declaração de variaveis

instruções

}
```

Exemplo:

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

//declaração de variáveis globais

// ---------------- Função main()-------------------
int main(void)
{
  //declaração das variáveis locais da main()

   return(0);
}
// -------------------------------------------------

void funcão1(variáveis locais de parâmetros)
{
   // declaração das variáveis locais da função1

   return;
}
```

Exemplo de um programa com variáveis globais e variáveis locais:

```
#include<stdio.h>

#include<conio.h>

//declaração de variáveis globais

float media, nota1, nota2;

//protótipo da função entrada

void entrada(void);

// ---------------- função main()-------------------

int main(void)

{

  //variável local

  char resposta;

  do

  {

    //chamada da função p/ entrada das notas

    entrada();

       //usando variáveis globais: média,nota1,nota2

    media = (nota1 + nota2) / 2;

       printf("\nMédia do aluno: %.2f\n", media);

         printf("\nDeseja calcular outra média? (s/n)");

    fflush(stdin);

    scanf("%c",&resposta);

  }

  while(resposta == 's');

         return(0);

}

// --------- fim da função main() -----


//função entrada de dados

//usa as variáveis globais nota1 e nota2
```

```
void entrada(void)
{
  printf("\nDigite a primeira nota: ");
  scanf("%f", &nota1);

  printf("Digite a segunda nota: ");
  scanf("%f", &nota2);

  return;
}
```

Função recursiva

Na linguagem C, assim como em muitas outras linguagens de programação, uma função pode chamar a sí própria. Uma função assim é chamada de RECURSIVA.

A técnica de recursividade permite a escrita de algoritmos claros e precisos, especialmente problemas de problemas que possuem natureza recursiva.

Por exemplo, para calcular o factorial de um número inteiro, a recursão pode ser utilizado. O fatorial de um número x é denotado por x! e é calculado como se segue:

```
x! X * = (x - 1) * (x-2) ...... 2 * 1
 x! = X * (n - 1)!
 onde (x-1)! = (X-1) * (x-2)!
```

O cálculo pode ser continuado até que recursivamente o valor 1.

Por definição, a função de forma recursiva calcula o factorial de um número inteiro positivo é: fact (n) = n * fact (n - 1)

Exemplo: Cálculo de factorial com função recursiva

```
#include <stdio.h>

#include <conio.h>

  //protótipo da função fatorial

double fatorial(int n);

  int main(void)

{

  int número;

  double f;

     printf("Digite o número que deseja calcular o factorial: ");

  scanf("%d",&numero);


  //chamada da função factorial

  f = fatorial(numero);

     printf("Factorial de %d = %.0lf",número,f);


  getch();

  return 0;

}

//Função recursiva que calcula o factorial

//de um numero inteiro n

double factorial(int n)

{


  double vfat;


  if ( n <= 1 )

     //Caso base: factorial de n <= 1 retorna 1

     return (1);
```

```
    else

    {

        //Chamada recursiva

        vfat = n * factorial(n - 1);

        return (vfat);

    }

}
```

Os arquivos de cabeçalho em C

As definições de função deve ser colocado no mesmo arquivo de origem como a principal função main (). Mas também podem ser colocados em uma separada que contém o ficheiro de fonte principal função. Por isso, é possível definir arquivos de extensão .h chamados arquivos de cabeçalho, em que os protótipos de funções definidas em um programa são combinados.

Header.h é um arquivo de cabeçalho.

A diretiva #include "header.h" deve ser colocado no início de cada arquivo-fonte para estabelecer o programa. É incorporar automaticamente o protótipo apropriado da função.

Conclusão

Essa atividade tornou-se familiarizado com o uso de funções que permitem a aplicação prática do conceito de linguagem de programação modular C. Os vários conceitos importantes, tais como a declaração, definição, chamadas de funções, recursividade, etc., foram discutidos em detalhe, acompanhada por exemplos ilustrativos. A seguir propõe-se um conjunto de exercícios para aplicação.

Avaliação

Exercícios:

1. Escreva uma função que solicita dois números ao utilizador e apresente na tela o resultado da sua soma e o dobro de cada um eles?
2. Faça uma função que recebe por parâmetro o raio de uma esfera e calcula o seu volume (v =4/3 π R3)?
3. Escreva uma função int_logic, a função deverá devolver o um valor lógico que indica se X é ou não igual a y2 , função do tipo int_logic(int x, int y)

Resumo da Unidade

Nesta unidade foi apresentada uma introdução sobre os métodos de programação top-down e bottom-up, suas vantagens e desvantagens e a introdução da programação modular e o porque da sua utilização pelos programadores. Também foram destacados os conceitos de funções e os procedimentos bem como a introduçao às funções recursivas. Frisou-se o formato genérico de uma função bem como as razões para a sua utilização.

A unidade foi ilustrada com vários exemplos para cada um dos pontos destacados de forma que o aluno possa ter uma visão clara de como utilizar estes conceitos na programação estruturada.

É notável a importância do desenvolvimento dessa unidade dado que as funções são uma ferramenta essencial ao desenvolvimento estruturado de aplicações hoje em dia e não só.

Avaliação da Unidade

Métodos de programação e programação modular em C

Instruções

Esta actividade permite fazer a avaliação de todos os pontos tratados nas diferentes actividades acima, desde as abordagens top-down e bottom-up até ao uso das funções. No final da unidade é apresentada a avaliação final do curso.

Critérios de Avaliação

Esta avaliação consiste em dois exercícios. O primeiro exercício tem 8 pontos com 2 pontos para cada resposta correcta e o segundo tem 12 pontos ,4 pontos a cada resposta correcta.

Avaliação

Exercício 1:Responda verdadeiro ou falso?

1. Na abordagem top-down para desenvolver os módulos de nível mais alto, não é preciso se preocupar com os pequenos detalhes.
2. Na abordagem bottom-up, os módulos de nível mais baixo são projectados tendo em conta os módulos de nível superior.
3. A principal vantagem da abordagem de cima para baixo é que ela permite ver o programa na sua totalidade.
4. A abordagem top-down torna mais difícil de monitorar e gerenciar o projeto.

Exercício 2: faça três funções recursivas que:

1. Calcula a potência de um número?
2. Pede um inteiro impar e positivo e multiplica todos os seus múltiplos inferiores ou iguais?
3. Calcule e retorne o N-ésimo termo da sequência Fibonacci. Alguns números desta sequência são: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89...

Leituras e outros Recursos

As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.

Avaliação do Curso

Desenvolvimento de um mini-projecto em C

Instruções

O mini-projecto aqui proposto tem como objectivo, aprofundar os seus conhecimentos adquiridos durante o módulo pondo em prática as instruções até aqui leccionados, desde de os conceitos de algoritmia, as estruturas condicionadas e ciclos de repetição até o estudo da programaçao modular as funções. Neste programa irás pôr em prática de uma forma resumida todos os conceitos considerados mais importantes para o curso.

O programa deverá ser elaborado no compilador Dev C++ e devidamente comentado. Para tal irás desenvolver um programa em C com o nome Mini_Proj_SeuNome_Curso.

Critérios de avaliação

Esta avaliaçao final do curso tem um peso de 30% na nota final e é um trabalho para ser realizado de forma individual e entregue no prazo estipulado, 15 dias.

Deverá ser entregue os ficheiros (código e executável) numa pasta zipada com o respectivo nome do aluno e do curso via plataforma moodle ou email???

Avaliação

O programa será desenvolvido deverá ter a seguinte estrutura e que responda a todas as questões a seguir.

Ao arrancar, o programa deve apresentar o seguinte menu de opções:

1. Área Triângulo
2. Múltiplo do Número
3. Fórmula Resolvente
4. Soma dos Quadrados
5. Senos e Cosenos
6. Sair

O utilizador escolhe entre uma destas opções:

1. Área Triângulo

Se no menu principal, o utilizador escolher a opção 1, o programa deverá calcular a área de um triângulo rectângulo, sendo pedido ao utilizador um dos catetos e a hipotenusa. Lembra-se que a soma dos quadrados dos catetos é igual a hipotenusa ao quadrado. Área triângulo =(Base*Altura/2).

2. Múltiplo do Número

Ao carregar na opção 2, o utilizador tem a oportunidade de introduzir um número inteiro qualquer e o programa deverá verificar se este número é múltiplo dos números 2, 3 ou de 5. Caso contrário aparecerá uma mensagem de erro.

3. Fórmula Resolvente

Se o utilizador escolher esta opção, o programa deverá fornecer as possíveis soluções para uma equação do segundo grau conhecendo os valores de A, B e C. Relembra-se que uma equação do segundo grau é descrita pela seguinte fórmula: Ax2+Bx+C. As soluções poderão ser 3, para diferentes valores de delta e Delta=B-4*A*C.

4. Soma dos Quadrados

Se o utilizador escolher esta opção, o programa deverá fazer a soma dos quadrados dos 10 números consecutivos, em que o limite inferior deverá ser introduzido pelo utilizador.

5. Senos e Cosenos

Se o utilizador escolher esta opção, o programa deverá preencher uma tabela com os valores de senos e cosenos de ângulos no intervalo [0°;180°] incrementados de 10°.

6. Sair

Se o utilizador escolher esta opção, deverá concluir e sair do programa.

Referências do Curso

1. Algoritmos -Teoria e Prática, Thomas H. Carmem ett all, Editora Campus, 2002.
2. C Completo e Total, Herbert Schildt, Editora Pearson Makron Books – 2006.
3. Concepts in programming Languages. John C. Mitchell. Cambridge University Press, 2003. Chapter, disponível: http://www.softpanorama.org/History/lang_history.shtml#General
4. C Completo e Total, SCHILDT H. "",Makron Books. SP, 1997.
5. Dicionário de Informática e novas tecnologias, José A. de Matos.
6. Fundamental da programação em C, Sampaio,Isabel e Sampaio, Alberto,FCA-Editora Informática.
7. Fundamentos de Programação usando C, Marques de Sá, J.P, Editora FCA- 2004, ISBN 972-722-475
8. Fundamentos da Programação de Computadores, Ascencio, Ana, Veneruchi,Edilene de Campos - 2ª edição – Editora Pearson Prentice Hall.
9. Lógica de Programação: A construção de algoritmos estruturas de dados", FORBELLONE, A. L. V. , Prentice Hall, SP, 2005.
10. Introdução à programação numa linguagem de alto nível, sem autor.
11. Introdução à informática e Algoritmia, Sampaio, Alberto e Sampaio, Isabel, Instituto Politécnico do Porto, Departamento de Eng.Informática.
12. Linguagem C, Luís Damas,Editora FCA, 1999, ISBN 972-722-136-4.
13. Lógica de Programação, Irenice F.Carboni, Editora Thomson.
14. Programming Languages & Paradigms,James B. Fenwick .
15. Programming in C, Stephen Kochan, (3rd edition). Sams Publishing.
16. The C Programming Language, Brian Kernighan and Dennis Ritchie. (2nd edition). Prentice Hall.
17. The Practice of Programming, Brian W. Kernighan et Rob Pike, 1999.
18. Tecnologias e Informação e Comunicação - o que são e para que servem?, Sérgio Sousa, 4ª Edição actualizada, Editora FCA.
19. "Treinamento em Linguagem C++ Módulo 1", MIZRAHI, V. V. , Makron Books, SP, 1995.
20. Utilizar o computador Depressa e Bem, Jorge Neves, Editora FCA.

Outros Links Consultados

TUTORIAIS E APOSTILAS

1. http://www.ime.usp.br/~kunio/devcpp/devcppintro.pdf
2. http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/Alg_Prog_I_2007_1/Aula_02.pdf
3. http://dcm.ffclrp.usp.br/~augusto/teaching/ici/Estruturas-de-Decisao.pdf
4. http://wurthmann.blogspot.com/2013/03/depuracao-debugger-no-dev-c.html
5. http://www.esj.eti.br/Apostilas/DicasDeDepuracaoDeCodigoUtilizandoDEVCPP_V10.pdf
6. http://www.esj.eti.br/Apostilas/DicasDeDepuracaoDeCodigoUtilizandoDEVCPP_V10.pdf
7. http://wiki.icmc.usp.br/images/e/e3/Manual_Dev_C.pdf
8. http://www.urisan.tche.br/~janob/Cap_1.html
9. http://pt.scribd.com/doc/27760045/Teste-de-Inf-Basica
10. http://www.proprofs.com/quiz-school/story.php?title=teste-diagnostico-tic

Sede da Universidade Virtual africana

The African Virtual University
Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

Escritório Regional da Universidade Virtual Africana em Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org

CC BY SA 2017 UVA