INSTITUTO FEDERAL SUL-RIO-GRANDENSE
UNIVERSIDADE ABERTA DO BRASIL
Programa de Fomento ao Uso das
TECNOLOGIAS DE COMUNICAÇÃO E INFORMAÇÃO NOS CURSOS DE GRADUAÇÃO - TICS

PROGRAMAÇÃO ESTRUTURADA

Fernanda Lopes Guedes

Ministério da Educação
GOVERNO FEDERAL BRASIL PAÍS RICO É PAÍS SEM POBREZA

Apostila de Programação Estruturada

GUEDES, Fernanda Lopes

2012/1

Produzido pela Equipe de Produção de Material Didático da
Universidade Aberta do Brasil do Instituto Federal Sul-rio-grandense

EQUIPE DE PRODUÇÃO DE MATERIAL DIDÁTICO – UAB/IFSUL

Lisiane Corrêa Gomes Silveira
GESTORA DA EQUIPE DE DESIGN

Denise Zarnottz Knabach
Felipe Rommel
Helena Guimarães de Faria
Lucas Quaresma Lopes
Tabata Afonso da Costa
EQUIPE DE DESIGN

Catiúcia Klug Schneider
GESTORA DE PRODUÇÃO DE VÍDEO

Gladimir Pinto da Silva
PRODUTOR DE ÁUDIO E VÍDEO

Marcus Freitas Neves
EDITOR DE VÍDEO

João Eliézer Ribeiro Schaun
GESTOR DO AMBIENTE VIRTUAL DE APRENDIZAGEM

Giovani Portelinha Maia
GESTOR DE MANUTENÇÃO E SISTEMA DA INFORMAÇÃO

Anderson Hubner da Costa Fonseca
Carlo Camani Schneider
Efrain Becker Bartz
Jeferson de Oliveira Oliveira
Mishell Ferreira Weber
EQUIPE DE PROGRAMAÇÃO PARA WEB

SUMÁRIO

GUIA DIDÁTICO 9

UNIDADE A - A INTRODUÇÃO A LINGUAGEM 13
1. Introdução da programação estruturada 14
2. Histórico da linguagem 14
3. Processo de compilação de um programa 14
4. Utilizando variáveis e estruturas de controle 18
5. Funções de entrada e saída 19
6. Comentários 21

UNIDADE B - TIPOS, OPERADORES E EXPRESSÕES 25
1. Variáveis e constantes 26
2. Tipos de dados e tamanhos 26
3. Declaração de inicialização de variáveis 27
4. Operadores aritméticos 28
5. Conversores de tipos 29
6. Operadores de incremento e decremento 29
7. Operadores lógicos 30
8. Operadores e expressões de atribuições 31
9. Expressões condições 31
10. Precedência e ordem de avaliação 31

UNIDADE C - ESTRUTURA DE FLUXO 35
1. IF-else 36
2. Switch 37
3. While 38
4. For 38
5. Dowhile 39
6. Break e continue 40
7. Stings 41

UNIDADE D - FUNÇÕES 51
1. Conceitos gerais 52
2. Parâmetros e argumentos 52
3. Variáveis externas 57
4. Regras de escapo 57
5. Inicialização de variáveis 58
6. Recursividade 58

UNIDADE E - VETORES E APONTADORES 63
1. Vetores 64
2. Vetores multidimensionais 65
3. Inicialização de vetores 66
4. Apontadores 67
5. Declaração de apontadores 67

UNIDADE F - MANIPULAÇÃO DE ARQUIVOS 73
1. Entrada / saída padrão 74

GUIA DIDÁTICO GD

APRESENTAÇÃO

Prezado(a) aluno(a),

bem-vindo(a) ao estudo da Disciplina de Programação Estruturada.

Nesta disciplina, aprenderemos a construir algoritmos utilizando a Linguagem de Programação C, como é o funcionamento desta linguagem, seus principais comandos e suas principais características.

Nas unidades serão abordados os seguintes assuntos: breve introdução à Linguagem, destacando o software utilizado para a compilação dos programas (DevC++); quais os principais tipos de dados, operadores e expressões utilizados na Linguagem; como é a estrutura de um comando de seleção e de repetição; como criamos funções em C; como trabalhamos com vetores e apontadores e como ocorre a manipulação de arquivos na Linguagem.

Desejamos que com a apresentação dos conteúdos dos materiais e da realização das atividades propostas, você possa obter subsídios para aprender a trabalhar com a Linguagem de Programação C.

Objetivos

Objetivo Geral

Ao final desta disciplina, você será capaz de construir algoritmos estruturados que sejam solução de um dado problema e que manipulem os dados adequadamente, utilizando a Linguagem de Programação C.

Habilidades

- Definir e utilizar variáveis e constantes no desenvolvimento de programas.
- Compreender e utilizar operadores aritméticos, relacionais e lógicos no desenvolvimento de programas.
- Compreender e utilizar as estruturas básicas de controle na implementação de programas.
- Traduzir soluções algorítmicas encontradas, para a linguagem de programação C.
- Utilizar conceitos de modularidade na construção de soluções de problemas.

Metodologia

A disciplina será desenvolvida em 60h através do Ambiente Virtual de Aprendizado Moodle, onde serão disponibilizados materiais para subsidiar a aprendizagem. Os recursos tecnológicos para interação serão os seguintes: Fórum e Chat de Dúvidas, E-mail, Textos, Exercícios.

Avaliação

O rendimento dos alunos será avaliado através das atividades propostas no curso e do instrumento de avaliação que ocorrerá em encontro presencial.

Programação

Primeira semana:

As atividades a serem desenvolvidas na primeira semana são:

1. Fórum: Apresentação do professor, da disciplina e questões gerais.
2. Fórum com relato de experiência na utilização de Linguagens de Programação, principalmente estruturadas.
3. Fórum com atividade de Pesquisa: elaborar pesquisa sobre a importância da Programação estruturada e um histórico da Linguagem C(como surgiu, importância).
4. Leitura das instruções sobre como instalar o software Dev C++(item Instalação do Compilador DevC++ da Unidade A).
5. Leitura de Texto sobre as principais funções de entrada e saída utilizada pelo C e como são feitos comentários nesta linguagem (Unidade A- Introdução a Linguagem).
6. Resolução de exercícios de múltipa escolha sobre Unidade A - Introdução à linguagem

Segunda semana:

As atividades a serem desenvolvidas na segunda semana são:

1. Leitura de Texto sobre os principais tipos e operadores utilizados na linguagem. (Unidade B- Tipos, operadores e expressões).
2. Resolução de exercícios sobre a Unidade B - Tipos, operadores e expressões.

Terceira semana:

As atividades a serem desenvolvidas na terceira semana são:

1. Leitura de Texto sobre como devem ser utilizados os comandos if e switch. (Unidade C– Estrutura de Fluxo, itens If-else e Switch).
2. Resolução de 10 algoritmos sobre if e switch.

Quarta semana:

As atividades a serem desenvolvidas na quarta semana são:

1. Leitura de Texto sobre como devem ser utilizados os comandos if e switch. (Unidade C– Estrutura de Fluxo, itens If-else e Switch).
2. Resolução de 10 algoritmos sobre if e switch.

Quinta semana:

As atividades a serem desenvolvidas na quinta semana são:

1. Leitura de Texto sobre como devem ser utilizados os comandos While, For e Do-while. (Unidade C– Estrutura de Fluxo, itens While, For e Do-while).
2. Resolução de 10 algoritmos sobre While, For e Do-while.

Sexta semana:

As atividades a serem desenvolvidas na sexta semana são:

1. Leitura de Texto sobre como se deve trabalhar com strings e algumas de suas funções de manipulação. (Unidade C– Estrutura de Fluxo, item Strings).
2. Resolução de 10 algoritmos sobre Strings.

Sétima semana:

As atividades a serem desenvolvidas na sétima semana são:

1. Leitura de Texto sobre quais são as características que devem ser observadas para construir uma função em C (Unidade D- Funções).
2. Resolução de 5 algoritmos sobre funções.

Oitava semana:

As atividades a serem desenvolvidas na oitava semana são:

1. Leitura de Texto sobre quais são as características que devem ser observadas para construir uma função em C (Unidade D- Funções).
2. Resolução de 5 algoritmos sobre criação de funções para utilizarmos juntamente com strings.

Nona semana:

As atividades a serem desenvolvidas na nona semana são:

1. Leitura de Texto sobre quais são as características que devem ser observadas para construir uma função recursiva em C (Unidade D- Funções, item Recursividade).
2. Resolução de 3 algoritmos sobre a criação de funções utilizando recursividade.

Décima semana:

As atividades a serem desenvolvidas na décima semana são:

1. Leitura de Texto sobre como construir um vetor em C e como utilizar ponteiros (apontadores) (Unidade E- Vetores e Apontadores).
2. Resolução de 5 algoritmos sobre a criação de vetores.

Décima primeira semana:

As atividades a serem desenvolvidas na décima primeira semana são:

1. Leitura de Texto sobre como construir um vetor em C e como utilizar ponteiros (apontadores) (Unidade E- Vetores e Apontadores).
2. Resolução de 5 algoritmos sobre a criação de vetores e apontadores.

Décima segunda semana:

As atividades a serem desenvolvidas na décima segunda semana são:

1. Leitura de Texto sobre como construir e manipular arquivos em C. (Unidade F- Manipulação de arquivos).
2. Resolução de 6 algoritmos sobre criação e manipulação de arquivos.

Currículo do Professor-Autor

Fernanda Lopes Guedes

Possui graduação em Ciência da Computação pela Universidade de Passo Fundo (2001) e Mestrado em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul / RS (2004). Atualmente é professora do Ensino Básico, Técnico e Tecnólogico do Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense. Tem experiência na área de Ciência da Computação, com ênfase em Educação a Distância, atuando principalmente nos seguintes temas: internet, educação a distância, ambientes de ensino e informática na educação.

Outras informações do currículo lattes em:

http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4700033J9

Referências

ARAÚJO, Jário. **Dominando a linguagem C**. Rio de Janeiro: editora Ciência Moderna Ltda, 2004.

SCHILDT, Herbert. **C, completo e total**. São Paulo: Pearson Makron Books, 1997.

SCHILDT, Herbert. **C++: guia para iniciantes**. Rio de Janeiro: Editora Ciência Moderna, 2002.

Complementares

AHO, Alfred. V.; LAM, Monica S.;SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores: princípios, técnicas e ferramentas**. 2 ed. São Paulo: Pearson Addison-Wesley, 2008.

ASCENCIO, Ana Fernanda Gomes. CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores**. 2 ed. São Paulo: Pearson, 2007

LÓGICA DE PROGRAMAÇÃO. Disponível em <http://www.slideshare.net/aislan/aula-03-lgica-de-programao>. Acesso em: 15 jul. 2011.

MANZANO, José Augusto N. G. **Estudo dirigido de Linguagem C**. 12. ed. São Paulo : Érica, 2009.

MANZANO, José Augusto; OLIVEIRA, Jayr Figueiredo. **Algoritmos: Lógica para Desenvolvimento de Programação de Computadores.** 15.ed. São Paulo: Erica, 2003.

MEDINA, Marco; FERTIG, Cristina. **Algoritmos e Programação: Teoria e Prática**. São Paulo: Novatec, 2005.

MORAES, Celso Roberto. **Estrutura de dados e algoritmos: uma abordagem didática**. São Paulo: Futura, 2003.

ORTH, Afonso Inácio. **Algoritmos e programação com resumo das linguagens Pascal e C**. Porto Alegre: AIO, 2001. 175p.

SALIBA, Walter Luiz Caram. **Técnicas de Programação: uma abordagem estruturada**. São Paulo: Makron Books, 1993.

SILVA, Osmar Quirino da. **Estrutura de dados e algoritmos usando C: fundamentos e aplicações**. Rio de Janeiro: Ed. Ciência Moderna, 2007.

SOUZA, Patrícia Almeida de. **Linguagem C**. Disponível em <http://pt.scribd.com/doc/20287949/Linguagem-C>. Acesso em: 15 de julho de 2011.

A

INTRODUÇÃO À LINGUAGEM

Unidade A
Programação Estruturada

UNIDADE A

INTRODUÇÃO À LINGUAGEM

Importância da Programação Estruturada

A programação estruturada enfatiza-se nas rotinas praticadas em blocos estruturados, com procedimentos e funções por passagem de dados.

Segundo Schildt (1997), a característica especial de uma linguagem estruturada é o compartimentalização de código e de dados. Essa compartimentalização representa a separação de tarefas específicas em sub-rotinas. As sub-rotinas utilizam variáveis locais, desse modo como essas variáveis são temporárias, é possível realizar mudanças no conteúdo dessas variáveis sem refletir no restante do programa. Além disso, a utilização de sub-rotinas permite que programas escritos em C compartilhem facilmente seções de código. Com essa reutilização de código, programas levam menos tempo para ser produzidos, pois não precisamos reescrever algumas sub-rotinas novamente.

Uma linguagem estruturada suporta diversas construções de laços (*loops*), como *while, do-while* e *for*. Nesse tipo de linguagem, o uso de comandos como goto deve ser evitado, pois fará com que o programa perca sua legibilidade.

Histórico da linguagem

A linguagem C foi desenvolvida na década de 70 por Dennis Richie em um DEC PDP-11 que utiliza o sistema operacional UNIX. C é derivado da linguagem chamada B (criada por Ken Thompson), que foi construída tendo como base a linguagem BCPL, criada por Martin Richards, na Europa (SCHILDT, 1997).

O C é uma linguagem de programação genérica que é utilizada para a criação de programas diversos como processadores de texto, planilhas eletrônicas, sistemas operacionais, etc.

C é uma linguagem de médio nível para computadores, combinando elementos da linguagem de alto nível com a funcionalidade da linguagem de baixo nível (*Assembly*).

Por ser uma linguagem de nível médio, permite a manipulação de bits, bytes e endereços. Permite, ainda, a conversão entre tipos de dados, como números convertidos em letras.

O que é Endereço?

Endereço de memória é um lugar reservado para armazenamento de dados em nosso computador.

Processo de compilação de um programa

O processo de compilação de um programa na linguagem C é simples e envolve uma sequência de passos. A seguir, é apresentado um fluxograma indicando tais passos (Figura A.1).

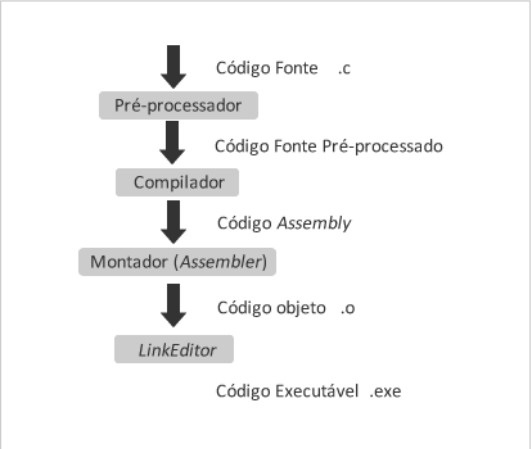


Figura A.1 - Compilação de um programa em C
Fonte: Adaptado de Lam, Sethi e Ullman (2008).

O **Pré-processador** realiza a análise léxica, procurando interpretar diretivas especiais, comentários do programa, entre outros detalhes a fim de que o programa objeto seja gerado (AHO; LAM; SETHI; ULLMAN, 2008).

Depois de realizada a etapa pelo pré-processador, o **Compilador** irá analisar o código gerado pelo pré-processador e transformar o código para a linguagem *Assembly*, efetuando a análise sintática e semântica dos dados.

O **Montador** *Asssembler* verifica o código e compila-o para código de máquina, produzindo, assim, o código objeto. O código objeto é geralmente armazenado em arquivos com a extensão .o .

Por fim, o ***LinkEditor*** faz a ligação com as bibliotecas externas, colocando as ligações e o código objeto em um único arquivo executável. Esse arquivo estará pronto para a execução e terá a extensão .exe.

O que é uma diretiva?

Segundo Schildt (1997), diretiva do pré-processador é uma instrução do compilador colocada no código fonte, inicia pelo símbolo #. As principais diretivas utilizadas em C e definidas pelo padrão ANSI são:

#if	#ifdef	#ifndef
#else	#elif	#endif
#include	#define	#undef

Revisão de termos

Código Fonte: é o texto criado para uma linguagem de programação, utilizando comandos próprios da linguagem.
Código Objeto: código de máquina que o computador pode ler e executar diretamente.
Código Executável: é o programa pronto para ser executado.

Instalação do Compilador DevC++

A primeira atividade a fazer antes de desenvolver programas usando a linguagem C é instalar um compilador. Para esta disciplina utilizaremos o ambiente de programação DevC++, disponível para download em <http://www.bloodshed.net/devcpp.html>. A opção por esse ambiente foi feita pelas seguintes razões:

- o DevC++ é um ambiente gratuito e livre, o que significa que toda e qualquer pessoa pode fazer o *download* e instalá-lo em sua máquina livremente.
- o DevC++ também é multiplataforma. Quer dizer, você pode utilizá-lo tanto em máquinas com sistema operacional *Windows* quanto *Linux*.

Passos para a instalação:

1. Fazer o download do programa DevC++.
2. Após baixar o programa, clique duas vezes sobre o ícone cujo nome corresponde a **devcpp4992.exe**. Se o arquivo executado foi baixado corretamente, você verá a seguinte tela (Figura A.2).

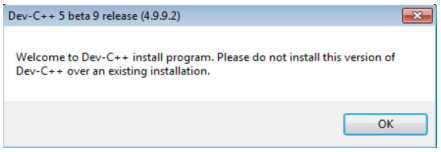


Figura A.2 – Tela de instalação do DevC++

3. Clique no botão de OK e em seguida aparecerá a tela (Figura A.3) de escolha do idioma.

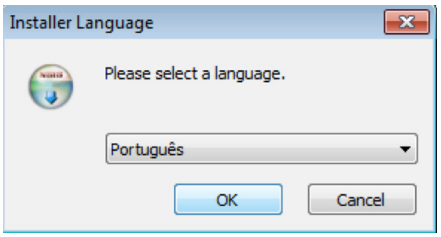


Figura A.3 – Tela de escolha do idioma de instalação do DevC++

4. É disparado então o procedimento de instalação. Será exibida uma janela com a licença de utilização do ambiente. Não se preocupe com isso, o software é livre e a licença diz exatamente isso. Leia a licença (se quiser) e clique no botão “Aceito”, indicando que concorda com os termos (Figura A.4).

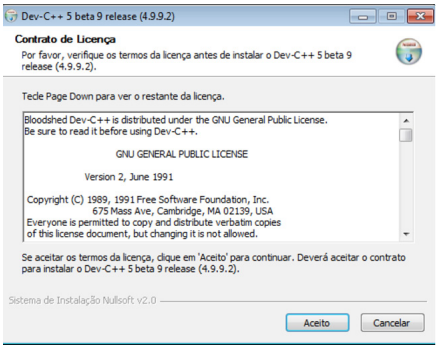


Figura A.4– Tela de contrato de licença do DevC++

5. A próxima tela (Figura A.5) solicita que você escolha quais os componentes do DevC++ que você quer utilizar. Por padrão, deixe todos marcados.

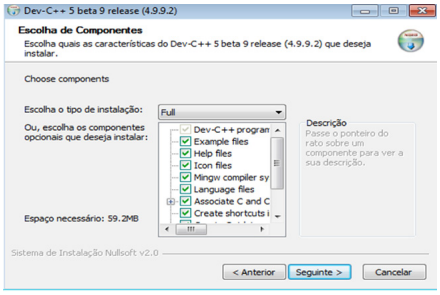


Figura A.5– Tela de escolha de componentes

6. Clicando no botão “Seguinte” aparecerá a tela para que você escolha o diretório onde o compilador deverá ser instalado. É aconselhável deixar no diretório sugerido (Figura A.6).

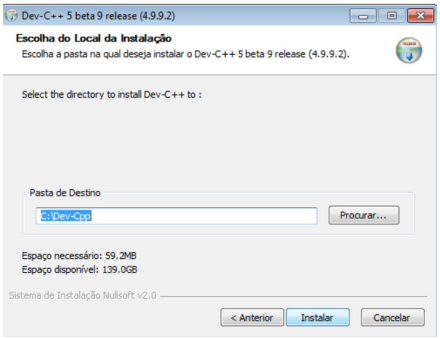


Figura A.6– Tela de escolha do local de instalação

7. Em seguida, após clicar em “Instalar”, aparecerá a tela (Figura A.7) de instalação.

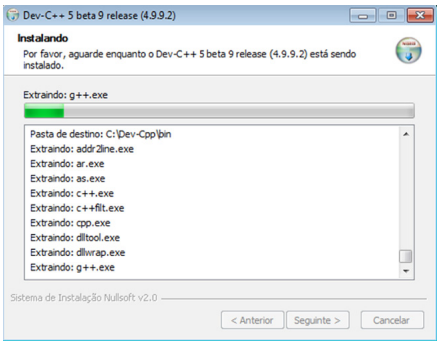


Figura A.7– Tela do progresso da instalação

8. Após a instalação clique em terminar e o software será executado.

Utilização do DevC++

1. Ao entrar no ambiente, você verá uma tela que é apresentada abaixo (Figura A.8). O primeiro passo é criar um projeto. Clique no botão indicado para criar um novo projeto (menu “Arquivo-Novo -> Projeto”).

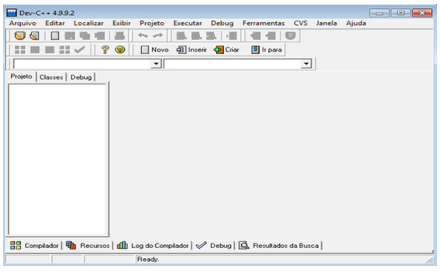


Figura A.8– Tela do ambiente DevC++

2. Aparecerá a tela da Figura A.9, solicitando o tipo de projeto. Clique na opção “Console application” (como indicado na figura abaixo), clique em Projeto C e, a seguir, no botão “OK”.

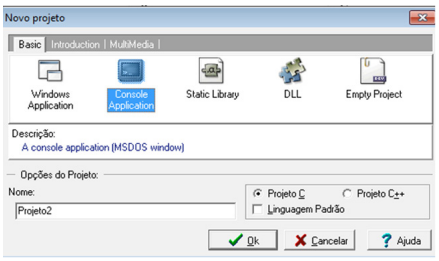


Figura A.9– Novo Projeto

3. A tela seguinte solicita o nome do projeto. Digite um nome adequado para ele e clique em “Salvar” (Figura A.10).

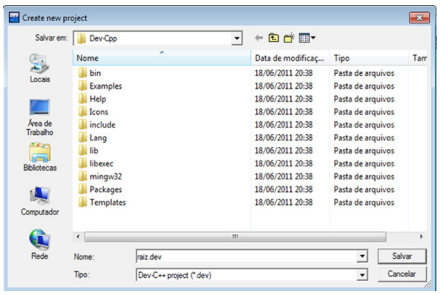


Figura A.10– Local para salvar o novo projeto

4. Após salvar o projeto, aparecerá uma tela com um trecho de código C permitindo que você inicie o seu trabalho. Salve o trecho de código com a extensão .c e bom trabalho (Figura A3.11).

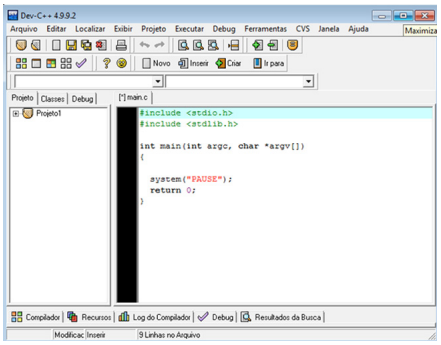


Figura A.11–Exemplo de arquivo dentro de um projeto

Utilizando variáveis e estruturas de controle

As estruturas de controle ou estruturas lógicas são designadas em: sequenciais, de decisão e de repetição. Porém, antes de iniciar com essas estruturas, veremos como devemos utilizar e escrever uma variável.

O uso de Variáveis

Variável pode ser definida como tudo aquilo que é sujeito a variações, que é incerto, instável ou inconstante. E quando trabalhamos com computadores teremos uma grande quantidade de informações que serão tratadas. Assim, essas informações a serem processadas tendem a ser variáveis (LÓGICA..., 2011).

Todo dado a ser armazenado na memória de um computador deve ser previamente identificado, ou seja, primeiro é necessário saber qual o seu tipo para depois fazer o seu armazenamento adequado. Estando armazenado o dado desejado, ele poderá ser utilizado e manipulado a qualquer momento.

Nome de Variáveis

Uma variável é formada por uma letra, "_" (*underline*), ou então, por uma letra seguida de outras letras ou dígitos. Não é permitido o uso de espaços em branco ou de qualquer outro caractere, que não seja letra ou dígito, na formação de um identificador.

Se utilizar palavras para compor o nome da variável utilize o "_" para separar as palavras.

Atenção

O C é "case sensitive", isto é, maiúsculas e minúsculas não são a mesma coisa. Significa que se declaramos uma variável chamada SOMA em maiúscula, essa mesma variável só existirá dessa forma. Caso alguma das letras tenha uma formação diferente, estaremos lidando com outra variável.

Funções de entrada e saída

As instruções são representadas pelo conjunto de palavras-chave (vocabulário) de uma determinada linguagem de programação, que tem por finalidade comandar em um computador o seu funcionamento e a forma como os dados armazenados deverão ser tratados. Em C, para realizarmos a entrada de um dado em um programa, utilizamos a função *scanf* e para a saída a função ***printf***.

Função *printf*

A função *printf* exibe na tela do monitor um dado, uma string ou mesmo um endereço de memória. O primeiro argumento da função é uma string que especifica o formato da impressão. A estrutura básica da função *printf*() é dada a seguir:

```
printf ("String de controle", Lista de argumentos);
```

onde:

String* de controle** pode ser tanto de caracteres para serem impressos na tela como códigos de formato, que especificam como apresentar o restante dos argumentos. Além de especificar caracteres dentro do primeiro parâmetro, pode-se incluir especificadores de formato (tabela A.1), que instruem ***printf() como imprimir os parâmetros indicados.

Código	Significado
%c	Exibe um caractere.
%d	Exibe um inteiro em formato decimal.
%i	Exibe um inteiro.
%e	Exibe um número em notação científica (com e minúsculo).
%E	Exibe um número em notação científica (com E maiúsculo).
%f	Exibe um ponto flutuante em formato decimal.
%g	Usa %e ou %f, o que for menor.
%G	O mesmo que %g, só que um E maiúsculo é usado se o formato %e for escolhido.
%o	Exibe um número em notação octal.
%s	Exibe uma string.
%u	Exibe um decimal sem sinal.
%x	Exibe um número em hexadecimal com letras minúsculas.
%X	Exibe um número em hexadecimal com letras maiúsculas.
%%	Exibe um sinal de %.
%p	Exibe um ponteiro.

Tabela A.1. String de controle. Fonte: Schildt (1997).

Lista de argumentos pode ser o conteúdo das variáveis a serem impressos.

Exemplo do comando printf:

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

int main(int argc, char *argv[])
{
    int idade = 30;
    float salario = 1820.50;
    char nome[3]="DEV";
    printf("Nome do usuario: %s idade: %d salario: %4.2f\n",
            nome,  idade, salario);/* será mostrado Nome do usuário:DEV idade:30
salario:1820.50*/

    printf("%d mais %d igual a %d\n", 1, 2, 1 + 2);/*será mostrado 1 mais 2 igual
a 3*/

    system("PAUSE");
    return 0;
}
```

Função scanf

A função scanf( ) é uma das funções de entrada de dados da Linguagem C, que pode ser usada para ler qualquer tipo de dado inserido por meio do teclado. A estrutura básica da função scanf é dada a seguir:

scanf ("String de controle", &Lista de argumentos);

onde:

String de controle: é uma string que especifica o formato da lista a ser lida, como visto no (tabela A.1).

Lista de argumentos: são os endereços das variáveis, onde os valores lidos serão armazenados. Deve ser precedido por & .

Exemplo do comando scanf:

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

int main(int argc, char *argv[])
{
   int a, b;
   double media;
   scanf ("%d %d", &a, &b); /* mostrará o prompt solicitando a digitação de dois
valores, vamos digitar 2 espaço 2*/
   media = (a + b) / 2.0;
   printf ("A média de %d e %d é %f\n", a, b, media); ); /* mostrará A média de 2
e 2 é 2.000000*/
```

```
    system("PAUSE");
    return 0;
}
```

Comentários

Os comentários servem principalmente para documentação do programa e são ignorados pelo compilador, portanto, não irão afetar o programa executável gerado.

Os comentários que envolvem mais de uma linha começam com os símbolos /* e se estendem até aparecerem os símbolos */.

Um comentário pode aparecer em qualquer ponto do programa, assim, temos também o símbolo // que corresponde ao comentário de somente uma linha.

```
#include <stdlib.h>

int main(int argc, char *argv[])
{
    //comentário de uma linha
   /* comentário
     de
     várias
    linhas
   */
   printf ("comentários");
   system("PAUSE");
   return 0;
}
```

Referências

AHO, Alfred. V.; LAM, Monica S.;SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores: princípios, técnicas e ferramentas**. 2 ed. São Paulo: Pearson Addison-Wesley, 2008.

LÓGICA DE PROGRAMAÇÃO. Disponível em <http://www.slideshare.net/aislan/aula-03-lgica-de-programao>. Acesso em: 15 jul. 2011.

SCHILDT, Herbert. **C, completo e total**. São Paulo: Pearson Makron Books, 1997.

SOUZA, Patrícia Almeida de. **Linguagem C**. Disponível em <http://pt.scribd.com/doc/20287949/Linguagem-C>. Acesso em: 15 de julho de 2011.

Atividades:

1. Com relação às características da Linguagem C.

I. **A linguagem C é uma linguagem orientada a objetos, que está em desuso.**

II. **A linguagem C é uma linguagem estruturada, de médio nível, permitindo a manipulação de bits, bytes e endereços.**

III. **Em linguagens estruturadas, como o C, o uso de comandos como goto deve ser evitado pois, caso contrário, produzirá um código macarrônico, fazendo com que o programa perca sua legibilidade.**

Estão corretas as alternativas:

a. Todas as alternativas.
b. II e III.
c. Somente II.
d. Somente III.
e. Todas estão erradas.

2. Quem inventou a Linguagem C? De quem ela é derivada?

a. Dennis Richie e Linguagem B.
b. Martin Richards e Linguagem BCPL.
c. Ken Thompson e Linguagem BCPL.
d. Martin Richards e Linguagem B.
e. Nenhuma das anteriores.

6. Com relação às etapas de compilação de um programa em C, marque a alternativa correta.

a. O pré-processador realiza a análise léxica, procurando interpretar diretivas especiais, comentários do programa, entre outros detalhes, a fim de que o programa objeto seja gerado.
b. O compilador irá interpretar diretivas especiais, comentários do programa, entre outros detalhes, a fim de que o programa objeto seja gerado.
c. O montador Asssembly verifica o código e compila-o para o código de máquina, produzindo assim o código objeto.
d. O arquivo executável terá a extensão .exe e o arquivo fonte a extensão .o.
e. O montador Assembler faz a ligação com as bibliotecas externas, colocando as ligações e o código objeto em um único arquivo executável.

4. Marque a alternativa que NÃO representa exemplos de diretivas em C.

a. #define
b. #include
c. #if
d. #else
e. #directive

5. Quais as regras que temos que seguir para criar variáveis em C? Marque a alternativa ERRADA:

a. Uma variável é formada por uma letra, "_" (underline), ou então, por uma letra seguida de outras letras ou dígitos.
b. Não é permitido o uso de espaços em branco ou de qualquer outro caractere que não seja letra ou dígito.
c. O nome de variável "seis vezes" é permitido;
d. O nome de variável "!isso" não é permitido;
e. O C é case sensitive, isto é, faz distinção entre letras maiúsculas e minúsculas na formação do nome de uma variável.

6. O que é Scanf?

a. É o nome de uma variável.
b. É uma das funções de entrada de dados da Linguagem C, que pode ser usada para ler qualquer tipo de dado inserido por meio do teclado.
c. É uma diretiva de C.
d. Exibe na tela do monitor um dado, uma string ou mesmo um endereço de memória.

e. É o comando para comentário em C.

7. O que é Printf?

a. É uma diretiva de C.

b. É uma das funções de entrada de dados da Linguagem C, que pode ser usada para ler qualquer tipo de dado inserido por meio do teclado.

c. Exibe na tela do monitor um dado, uma string ou mesmo um endereço de memória.

d. É o comando para comentário em C.

e. É o nome de um tipo de dado.

8. Marque a alternativa que não representa as strings de controle utilizadas nos comandos scanf e printf.

a. %x

b. %d

c. %s

d. %o

e. %h

9. O código a seguir apresenta quantos erros se observamos somente o comando scanf?

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

int main(int argc, char *argv[])
{
  char nome[40];
  int mes;
  printf(“Informe o seu nome:”);
  scanf(“s”,&nome);
  printf(“Informe o seu mes de aniversario”);
  scanf(&mes,”%d”);
   system(“PAUSE”);
  return 0;
}
```

a. 1

b. 2

c. 3

d. 4

e. Nenhuma das alternativas

10. Marque a alternativa em que o código equivale ao significado nas strings de controle utilizadas nos comandos scanf e printf.

a. %fd - exibe um inteiro em formato decimal.

a. %f – exibe um número em notação científica.

b. %s –exibe um caractere.

c. %h- exibe um número em hexadecimal.

d. %c –exibe um caractere.

B

TIPOS, OPERADORES E EXPRESSÕES

Unidade B
Programação Estruturada

UNIDADE B

TIPOS, OPERADORES E EXPRESSÕES

Variáveis e constantes

Segundo (SCHILDT, 1997), uma variável ocupa uma posição de memória, esta posição guarda seu valor que pode ser modificado pelo programa. Todas as variáveis devem ser declaradas antes de serem utilizadas. As regras para a criação dos nomes das variáveis já foram definidas no item **Nome de Variáveis** da unidade anterior. A declaração pode ser definida da seguinte maneira:

tipo lista_de_variáveis

Tipo deve ser um tipo de dado válido em C.

Exemplos:

```
int i, j, l;
short int si;
double balanco, salário, lucro;
```

Constantes em C são valores fixos que o programa não pode alterar. Elas podem ser de qualquer um dos cinco tipos de dados básicos.

Exemplos:

- 10 e -100 são constantes inteiras.
- 'a' e '%' são constantes tipo caractere.
- 11.123 e 12.34 são constantes em ponto flutuante.

Lembre-se

Utilizamos aspas simples para caracteres e aspas duplas para strings.
'a' é um caractere, enquanto "a" é uma string com somente uma letra.

Tipos de dados e tamanhos

Há cinco tipos básicos de dados em C: caractere, inteiro, ponto flutuante, ponto flutuante de precisão dupla e sem valor (char, int, float, double e void, respectivamente).

Tipo	Tamanho aproximado em bits	Faixa mínima
Char	8	-127 a 127
unsigned char	8	0 a 255
signed char	8	-127 a 127
Int	16	-32767 a 32767
unsigned int	16	0 a 65.535
signed int	16	O mesmo que int
short int	16	O mesmo que int
unsigned short int	16	0 a 65.535
signed short int	16	O mesmo que short int
long int	32	-2.147.483.647 a 2.147.483.647
signed long int	32	O mesmo que long int
unsigned long int	32	0 a 4.294.967.295
Float	32	Seis dígitos de precisão
Double	64	Dez dígitos de precisão
long Double	80	Dez dígitos de precisão

Tabela B.1- Tipos de dados definidos pelo padrão ANSI

Fonte: Schildt (1997).

Declaração e inicialização de variáveis

Em C é possível dar um valor à variável no momento em que elas são criadas ou declaradas, para fazer isso colocamos um sinal de igual e uma constante após o nome da variável. A inicialização pode ser definida da seguinte maneira:

tipo nome_da_variável= constante;

Exemplo:

```
char c = 'a';
int numero = 10;
float salario = 1500.49;
```

Lembre-se

Para declararmos uma variável devemos saber qual o seu tipo e utilizar nomes mnemônicos (nomes parecidos com a situação real, por exemplo, uma variável chamada sal ou salar ou salario poderá se referir ao salário).

Operadores aritméticos

A linguagem C trabalha basicamente com os quatro operadores aritméticos básicos (adição, subtração, multiplicação e divisão). O quadro a seguir representa esses operadores e outros que nos auxiliam na realização de operações aritméticas:

Operação	Operador aritmético
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto da divisão inteira	%
Decremento	--
Incremento	++

Tabela B.2- Operadores Aritméticos
Fonte: Schildt (1997).

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   int a, b, a1, b1;
   a=5;
   b=2;
   printf("%d",a/b); //mostrará 2
   printf("%d",a%b); //mostrará 1, o resto da divisão inteira
   a1=1;
   b1=2;
   printf("%d %d",a1/b1,a1%b1); //mostrará 0  1


   system("PAUSE");
   return 0;
     }
```

Conversores de tipos

Podemos fazer com que uma expressão se torne um determinado tipo usando o comando cast.

A forma de escrita deste comando é:

tipo (expressão)

onde tipo é qualquer tipo de dado válido em C.

Exemplo:
Se quisermos que a expressão i/2 resulte em float devemos escrevê-la assim:

(float)i/2

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   int i;
   i=1;
   printf("%d  %f",i,(float)i/2);//mostrará 1 e 0,5
   system("PAUSE");   return 0;
}
```

Operadores de incremento e decremento

O operador de incremento (operador ++) soma 1 ao seu operando, enquanto o operador de decremento (operador --) subtrai 1. Eles podem ser utilizados da forma prefixada ou sufixada (++x ou x++). Porém, há diferença quando eles são utilizados em uma expressão.

Exemplo:

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

int main(int argc, char *argv[])
{
   int a, b, c, d;
   a=1;
   c=1;
   b=a++; //atribui o valor de a para b e depois incrementa em 1 a variável a

   printf("%d %d",b,a); //mostrará 1 2

   d=++c; //incrementa a variável c em 1 e depois atribui o valor deste
incremento a d

   printf("%d %d",d,c); //mostrará 2 2
   system("PAUSE");
```

```
        return 0;
    }
```

Também podemos realizar incremento da seguinte forma:

x=x+1 ou x+=1

E decremento da seguinte forma:

x=x-1 ou x-=1

Operadores lógicos

Em c, verdadeiro é qualquer valor diferente de zero. Falso é zero. As expressões que usam operadores lógicos ou relacionais devolvem zero para falso e um para verdadeiro.

Operadores relacionais	
Operador	Relação
>	Maior que
>=	Maior que ou igual a
<	Menor que
<=	Menor que ou igual a
= =	Igual
!=	Diferente

Tabela B.3- Operadores relacionais
Fonte: Schildt (1997).

Operadores lógicos	
Operador	Operação lógica
&&	E
||	OU
!	NÃO

Quadro B.4- Operadores relacionais
Fonte: Schildt (1997).

Lembre-se

Nas expressões que envolvem operadores lógicos ou relacionais temos:
Verdadeiro(V) é 1.
Falso(F) é 0.

Operadores e expressões de atribuição

Em C, podemos utilizar o operador de atribuição em qualquer expressão válida de C. O operador de atribuição pode ser utilizado da seguinte forma:

nome_da_variaval=constante;

Exemplos:

i=10;
c='a';
s="casa";
O C permite, ainda, atribuição múltipla:
x = y = z = 0;

Expressões condições

C possui o operador ternário? que substitui certas sentenças da forma if-else (sentenças condicionais). O operador ternário tem a seguinte forma:

Exp1 ? *Exp2* : *Exp3*;

Funciona da seguinte forma: a Exp1 é avaliada, se ela for verdadeira então a Exp2 se torna o valor da expressão. Agora, se Exp1 é falsa, então Exp3 se torna o valor da expressão.

Exemplo:

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

int main(int argc, char *argv[])
{
   int i,y;
   i=10;
   y=i>9 ? 100 : 200;
   printf("%d",y);//mostrará 100

   system("PAUSE");
   return 0;
}
```

Precedência e ordem de avaliação

O quadro a seguir lista a precedência de todos os operadores de C do maior para o menor, fazendo a associação conforme mostrado.

Maior	Operador	Associatividade
	( ) [ ] -> .	esquerda-direita
	! ~ ++ -- - (tipo) * & sizeof	direita-esquerda
	* / %	esquerda-direita
	<< >>	esquerda-direita
	+ -	esquerda-direita
	< <= > >=	esquerda-direita
	== !=	esquerda-direita
	&	esquerda-direita
	^	esquerda-direita
	|	esquerda-direita
	&&	esquerda-direita
	||	esquerda-direita
	?:	direita-esquerda
	= += -= etc.	direira-esquerda
Menor	,	esquerda-direira

Quadro B.5- Precedência de operadores

Fonte: Adaptado de Schildt (1997).

Referências

LÓGICA DE PROGRAMAÇÃO. Disponível em <http://www.slideshare.net/aislan/aula-03-lgica-de-programao>. Acesso em: 15 jul. 2011.

SCHILDT, Herbert. **C, completo e total**. São Paulo: Pearson Makron Books, 1997.

SOUZA, Patrícia Almeida de. **Linguagem C.** Disponível em <http://pt.scribd.com/doc/20287949/Linguagem-C>. Acesso em: 15 de julho de

Atividades

1. Aponte qual o tipo em C de cada um dos valores: Exemplo: Nome de Rua char

a. Número de Casa
b. Idade de uma pessoa
c. Peso de uma pessoa
d. Sexo de uma pessoa
e. Valor do salário

f. Signo astrológico
g. Nome de carro
h. Preço do combustível
i. Número de pessoas em uma fila
j. Modelo de um carro
k. 'F'
l. "Feminio"
m. "Falso"
n. 5.84
o. "A*B!?"
p. "Amarelo"
q. "03 de agosto de 2004"
r. 2006
s. 03

2. Indique o valor da variável X do tipo inteiro em cada expressão aritmética. Considere A=10, B=5, C=6, D=7, E=2, F=3, em que A, B, C, D, E, F são variáveis inteiras.

a. X= A+B
b. X= A + B – C
c. X =A * B – C
d. X =A + B * C
e. X =A + B * C / F – E
f. X =A % B / E
g. X =C / E * F
h. X =A / B / E
i. X = C / E /F
j. X=A%C*F

3. Qual o valor final da variável Idade?

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

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

  idade=35;
  idade=79;
  idade=idade - 3;
  idade=idade/2;
  system("PAUSE");
  return 0;
}
```

Realize os seguintes algoritmos na Linguagem C:

4. Ler dois valores para as variáveis A e B, efetuar a troca dos valores de forma que a variável A passe a possuir o valor da variável B e que a variável B passe a possuir o valor da variável A. Apresentar os valores trocados.

5. Elaborar um algoritmo que tendo 2 valores (A,B) apresente como resultado as operações de: Adição, subtração, multiplicação e divisão de A por B.

6. Elaborar um algoritmo que tendo três valores (a, b e c) apresente como resultado final a soma dos quadrados dos três valores lidos.

7. Elaborar um algoritmo que tenho três notas(N1,N2,N3) de um determinado aluno, apresente como resultado a Média Final:

8. Escreva um algoritmo para calcular o comprimento de uma circunferência de raio 3 cm.

9. Escreva um algoritmo que lê um número e mostra seu sucessor e seu antecessor na tela.

10. Escreva um algoritmo que lê o raio de um círculo e mostre como saída o perímetro e a área.

C

ESTRUTURA DE FLUXO

Unidade C
Programação Estruturada

UNIDADE C

ESTRUTURA DE FLUXO

If-else

Comando de seleção que verifica se uma condição é verdadeira e executa o bloco ou comando que forma o **if**, caso contrário, o bloco ou comando que forma o else (se existir) será executado.

A forma geral da sentença **if**:

```
if (condição)
{
Comando ou bloco;
}
else
{
Comando ou bloco;
}
```

Lembre-se

Em C um valor verdadeiro é qualquer valor diferente de 0 (1). Enquanto um valor falso é 0.
Verdadeiro → 1
Falso → 0

Exemplo de utilização do comando if:

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

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

  printf("Informe idade:");
  scanf("%d",&idade);
  if (idade>=18)// se a idade for maior ou igual a 18 mostrará a mensagem
Maior de Idade
       printf("Maior de Idade\n");
  else // se a idade informada for menor que 18 mostrará a mensagem Menor de
Idade
     printf("Menor de Idade\n");
   system("PAUSE");
   return 0;
}
```

Switch

C tem um comando chamado **switch** de seleção múltipla. Ele permite testar uma constante ou um caractere com um valor de uma expressão, quando o valor coincide determinados comandos associados à constante ou ao caracter serão executados.

A forma geral da sentença **switch**:

```
switch  nome da variável{
        case valor 1:
                comando;
                break;
        case  valor 2:
                comando;
                break;
        case  valor n:
                comando;
                break;
        default:
        comando;}
```

O comando **default** será executado se nenhuma coincidência for detectada.

O comando **break** é um comando de desvio, e quando encontrado em um switch, a execução do programa "salta" para a linha de código seguinte ao comando switch.

Exemplo de utilização do comando **switch**:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  int numero;
  printf("Informe um numero:");
  scanf("%d",&numero);
  switch (numero){
    case 1:
        printf("um"); //mostrará um se 1 for digitado
     break;
    case 2:
     printf("dois"); //mostrará dois se 2 for digitado
     break;
    default:
     printf("nenhum"); //mostrará nenhum se for um valor diferente de 1 e 2
   system("PAUSE");
   return 0;
}
```

While

O comando while é um comando de iteração (também chamado de laço) que permite executar o mesmo comando diversas vezes. A forma geral do comando while é:

while(condição){

Comandos;

}

A condição pode ser qualquer expressão relacional. O laço se repete enquanto a condição for verdadeira.

Exemplo do comando:

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

int main(int argc, char *argv[])
{
  int i;
  //irá imprimir a mensagem “Adoro C” 10 vezes
  i=1;
  while(i<=10)
  {
     printf(“%d Adoro C!  \n”, i);
     i++;
  }
   system(“PAUSE”);
   return 0;
}
```

For

O comando for também é um comando de iteração. A forma geral do comando **for** é:

for(variável=inicialização; condição ; incremento){

Comandos;

}

Na **inicialização** colocamos um comando de atribuição, este comando irá colocar um valor na variável de controle do laço.

A **condição** é uma expressão relacional que determina quando o laço acaba.

O **incremento** define como a variável de controle do laço varia cada vez que o laço é repetido.

Exemplo do comando:

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

```
int main(int argc, char *argv[])
{
  int i;
  //irá imprimir a mensagem “Adoro C” 10 vezes
  for(i=1;i<=10;i++)
  {
     printf(“%d Adoro C!  \n”, i);
  }
   system(“PAUSE”);
   return 0;
}
```

Do-while

Outro comando de iteração é o comando do-while. A forma geral do comando **do-while** é:

do{

Comandos;

} while(condição);

A característica principal desta estrutura é que a instrução é executada **pelo menos uma vez**, independentemente do resultado da **condição**.

O comando do-while repete até que a condição se torne falsa.

Exemplo do comando:

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

int main(int argc, char *argv[])
{
  int i;
  //irá imprimir a mensagem “Adoro C” 10 vezes
  i=1;
  do
  {
     printf(“%d Adoro C!  \n”, i);
     i++;
  }while(i<=10);
   system(“PAUSE”);
   return 0;
}
```

Break e continue

Break

O comando **break** tem dois usos. O primeiro para terminar um **case** em um comando **switch**. O segundo para forçar o término de um laço.

Exemplo:

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

int main(int argc, char *argv[])
{
  int i;
  for(i=1;i<=10;i++)
  {
    if (i==4)
       break;//se i for 4 pula para o próximo comando após o for.
    else
        printf("%d\n",i); //imprime de 1 a 3
  }
  system("PAUSE");
  return 0;
}
```

Continue

Em vez de forçar o término do laço, ele força que ocorra a próxima iteração do laço.

Exemplo:

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

int main(int argc, char *argv[])
{
  int i;
  for (i = 0; i < 10; i++)
  {
      scanf("%d", &num);
      //lê 10 números inteiros, caso o número seja negativo um novo número é lido.
      if (num < 0) continue;
         printf("%d\n", num);
  }
```

```
    system("PAUSE");
    return 0;
}
```

Strings

Strings

Em C, uma string é definida como um vetor de caracteres que é terminada por um nulo (\0). Assim, precisamos declarar vetores de caracteres maiores que considerem o nulo. Por exemplo, para declararmos a variável frase que guarda uma string de 14 caracteres, escreveríamos:

char frase[15];

Isso reserva um espaço para o nulo no final da string.

Cuidados ao se trabalhar com strings:

- Por serem simples vetores char, as strings estão sujeitas a todas as regras pertinentes aos vetores normais.
- Todas as strings devem ser passadas por referência para funções, uma função não pode retornar strings, mas pode retornar o endereço de um string (char*).
- Strings não podem ser atribuídas com o operador de atribuição = (embora possam ser inicializadas).
- Strings não podem ser concatenadas com o operador +.
- Strings não podem ser comparadas com o operador de comparação ==.

Funções de Manipulação de String

As funções a seguir utilizam a biblioteca string.h (#include <string.h>)

Função strcpy

*char *strcpy(char *destino,char* *origem)

Copia o conteúdo de origem em destino. Copia uma string para outra. Retorna um ponteiro para destino.

Exemplo:

O fragmento de código seguinte copia alo na *string* str.

char str[80];

strcpy(str,"alo");

Função strncpy

*char *strcnpy(char* *destino, *char* *origem,*unsigned count)*

Copia até *count* caracteres da *string* apontada por origem na string apontada por destino. Retorna destino. Não coloca \0 no final de destino.

Exemplo:

O pedaço de código seguinte copia no máximo 79 caracteres de str1 para str2, garantindo, assim, que não ocorrerá nenhum estouro de limite de vetor.

char str1[128], str2[80];

gets(str1);

strncpy(str2,str1,79);

Função strcmp

int strcmp(*char* *str1, *char* *str2)

Compara duas strings e retorna um inteiro baseado no resultado, como mostrado a seguir:

Valor	Significado
<0	str1 é menor que str2
0	str1 é igual a str2
>0	str1 é maior que str2

Exemplo que retornará **Str1 Maior**:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[])
{
    char str1[4]="alo",str2[4]="al";
    if (strcmp(str1,str2)==0)
       printf("Igual");
    if (strcmp(str1,str2)<0)
       printf("Str1 Menor");
    if (strcmp(str1,str2)>0)
       printf("Str1 Maior");
    system("PAUSE");
    return 0;
}
```

Função strncmp

int strncmp(char *str1, char *str2,unsigned count)

Compara não mais que count caracteres das duas strings terminadas com nulo e retorna um inteiro baseado no resultado, como mostrado a seguir:

Valor	Significado
<0	str1 é menor que str2
0	str1 é igual a str2
>0	str1 é maior que str2

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[])

{char str1[4]="ab",str2[4]= "ab";
```

```
    if (strncmp(str1,str2,2)==0)
      printf("Igual");//MOSTRARÁ ESTA MENSAGEM
    if (strncmp(str1,str2,2)<0)
      printf("Str1 Menor");
    if (strncmp(str1,str2,2)>0)
      printf("Str1 Maior");
    system("PAUSE");
    return 0;
  }
```

Função strcat

*char *strcat*(*char* *destino,*char* *origem)

Concatena duas strings. Retorna a string concatenada (destino). É de responsabilidade do programador garantir que destino seja suficientemente grande para armazenar seu conteúdo original e o de origem.

Exemplo:

O programa seguinte acrescenta a primeira string lida à segunda. Por exemplo, supondo que o usuário tenha digitado alo e aqui, o programa escreve aquialo.

```
#include <stdio.h>
#include <string.h>
int main()
{char s1[80],s2[80];
  gets(s1);  //digitamos alo
  gets(s2); //digitamos aqui
  strcat(s2,s1);
  printf("%s",s2);//escreverá aquialo
  system("PAUSE");
  return 0;
}
```

Função strlen

unsigned strlen(*char* *str)

Retorna o número de caracteres da string, sem contar o '\0'.

Exemplo:

Este fragmento de código escreve 3 na tela.

printf("%d",*strlen*("alo"));

Função strupr

*char *strupr(char* *str)

Converte todos os caracteres da *string* em maiúsculo.

Exemplo:

```
#include <stdio.h>
#include <string.h>
int main()
{char s1[5]=”alo”;
  strupr(s1);
  printf(“%s”,s1);//escreverá ALO

  system(“PAUSE”);
  return 0;
}
```

Função strlwr

*char *strlwr(char* *str)

Converte todos os caracteres da string em minúsculo.

Exemplo:

```
#include <stdio.h>
#include <string.h>
int main()
{char s1[5]=”ALO”;

  strlwr(s1);
  printf(“%s”,s1);//escreverá alo
  system(“PAUSE”);
  return 0;
}
```

As funções a seguir utilizam a biblioteca *stdio*.h (#include <stdio.h>)

Função gets

*char *gets(char* *str)

Lê uma string a partir do teclado. Retorna um ponteiro para a string lida.

Exemplo:

char s1[10];

gets(s1);

printf(“%s\n”,s1); //se digitarmos alo aparecerá alo

Função puts

int *puts(char *str)

Escreve uma string na tela seguida de \n (nova linha). Retorna EOF em caso de erro. É semelhante a printf("%s\n",string);

Exemplo:

char s1[10]="alo";

puts(s1);

printf("%s\n",s1); //imprimirá alo

Funções de Manipulação de Caracter

As funções a seguir utilizam a biblioteca ctype.h (#include <ctype.h>).

int isalnum(int caracter)

Retorna um valor diferente de zero se o argumento for uma letra ou dígito. Se o caracter não for alfanumérico, isalnum() devolverá zero.

int isalpha(int caracter)
Retorna um valor diferente de zero se o argumento for uma letra do alfabeto. Caso contrário, devolverá zero.

int isdigit(int caracter)
Retorna um valor diferente de zero se o caracter for um dígito (isto é, de 0 a 9). Caso contrário, devolve zero.

int islower(int caracter)
Retorna um valor diferente de zero se o caracter for uma letra minúscula. Caso contrário, devolve zero.

int isupper(int caracter)
Retorna um valor diferente de zero se o caracter for uma letra maiúscula. Caso contrário, devolve zero.

int isspace(int caracter)
Retorna um valor diferente de zero se o caracter for um espaço branco. Caso contrário, devolve zero.

int tolower(int caracter)
Retorna o equivalente minúsculo do caracter se o caracter é uma letra; caso contrário, o caracter é devolvido sem alteração.

int toupper(int caracter)
Retorna o equivalente maiúsculo do caracter se o caracter é uma letra; caso contrário, o caracter é devolvido sem alteração.

Referências

LÓGICA DE PROGRAMAÇÃO. Disponível em <http://www.slideshare.net/aislan/aula-03-lgica-de-programao>. Acesso em: 15 jul. 2011.

SCHILDT, Herbert. **C, completo e total**. São Paulo: Pearson Makron Books, 1997.

SOUZA, Patrícia Almeida de. **Linguagem C**. Disponível em <http://pt.scribd.com/doc/20287949/Linguagem-C>. Acesso em: 15 de julho de 2011.

Atividades

1. A nota de uma disciplina é composta pela nota de três provas. A primeira prova tem peso 2, a segunda tem peso 3 e a terceira tem peso 5. Considerando que a média para aprovação é 6.0, Faça um algoritmo para calcular a média final de um aluno desta disciplina e dizer se o aluno foi aprovado ou não.

2. Faça um algoritmo que diga se um número fornecido como pelo usuário é PAR ou ÍMPAR.

3. Faça um algoritmo que leia um número, e se ele for maior do que 50, imprimir a metade desse número.

4. Faça um algoritmo que leia dois números inteiros e efetue a adição; caso o resultado seja maior que 20, imprima-o.

5. Faça um algoritmo que leia um número e, se ele for positivo, imprima a metade desse número, caso contrário imprima o número ao quadrado.

6. Construa um algoritmo que leia um número e imprima uma das mensagens: é múltiplo de 5, ou, não é múltiplo de 5.

7. Construa um algoritmo que leia dois números e responda se a divisão do primeiro pelo segundo é exata (o resto da divisão deve ser igual a 0). Se for, o algoritmo deve imprimir a mensagem “A divisão de (1º numero) por (2º número) é exata”.

8. Construa um algoritmo que leia o nome e o peso de duas pessoas e imprima os dados da pessoa mais gorda.

9. Construa um algoritmo que indique se um número digitado está compreendido entre 30 e 80, ou não.

10. Construa um algoritmo que leia a altura e o sexo de uma pessoa, calcule e imprima seu peso ideal, utilizando as seguintes fórmulas.

Para homens	(72,7* altura)-58
Para mulhers	(62,1* altura)-=44,7

Atividades

1. Um comerciante comprou um produto e quer vendê-lo com um lucro de 35% se o valor da compra for menor que R$ 15,00; caso contrário, o lucro será de 20%. Construa um algoritmo que leia o valor do produto e imprima o valor de venda para o produto.

2. Elabore um algoritmo que leia um número. Se o número for positivo armazene-o em A, se for negativo, em B. No final mostrar o resultado

3. A Turma C é composta de 60 alunos, e a turma D de 20 alunos. Escreva um algoritmo que leia o percentual de alunos reprovados na turma C, o percentual de aprovados na turma D, calcule e escreva:

- **a.** O número de alunos reprovados na turma C.
- **b.** O número de alunos reprovados na turma D.
- **c.** A percentagem de alunos reprovados em relação ao total de alunos das duas turmas.

4. Escreva um algoritmo para ler o nome de 2 times e o número de gols que cada time marcou em uma determinada partida. Escrever o nome do vencedor. Caso não haja vencedor deverá ser impresso a palavra EMPATE.

5. Escreva um algoritmo para ler o número de lados de um polígono regular, e a medida do lado. Calcular e imprimir o seguinte:

- **a.** Se o número de lados for igual a 3 escrever TRIÂNGULO e o valor do seu perímetro.
- **b.** Se o número de lados for igual a 4 escrever QUADRADO e o valor da sua área.
- **c.** Se o número de lados for igual a 5 escrever PENTÁGONO.

6. Escreva um algoritmo para ler 3 valores e escrevê-los em ordem crescente.

7. Escreva um algoritmo para ler 3 valores A, B e C representando as medidas dos lados de um triângulo, e escrever se formam ou não um triângulo. Para formar um triângulo o valor de cada lado deve ser menor que a soma dos outros 2.

8. Faça um algoritmo para ler cinco números e imprimir o cubo e o quadrado de cada um deles.

9. Elabore um algoritmo para gerar uma tabela de conversão entre milhas e Km, iniciando em 0 Km e finalizado em 1000 Km, e varie de 100 Km em 100 Km, sabendo-se que : 1 Milha = 1609 m.

10. Para eleição de representantes de classe de uma universidade há três candidatos. Os votos são informados através de código: 1, 2 ou 3, votos para os respectivos candidatos, 5 votos nulo e 6 votos em branco. Faça um algoritmo que calcule e escreva:

- **a.** A porcentagem e o total de votos para cada candidato
- **b.** Total de votos nulos
- **c.** Total de votos em branco
- **d.** Percentual de votos em brancos e nulos
- **e.** Classificação dos candidatos
- **f.** Total de votos

Atividades

Realize os exercícios de 1 a 3 utilizando o comando While:

1. Apresente todos os valores numéricos inteiros ímpares situados na faixa de 0 a 20. Para verificar se o número é ímpar, efetuar dentro do laço a verificação lógica desta condição com a instrução se, perguntando se o número é ímpar; sendo, mostre-o; não sendo, passe para o próximo passo.

2. Apresente o total da soma obtido dos cem primeiros números inteiros (1+2+3+4+5..+98+99+100).

3. Apresente os resultados de uma tabuada de um número qualquer. Ela deve ser impressa no seguinte formato:

Considerando como exemplo o fornecimento do número 2:

2X1=2
2X2=4
2X3=6
2X4=8
2X5=10
...
2X10=20

Realize os exercícios de 4 a 6 utilizando o comando For:

4. Apresente os números inteiros de 1 a 10 em ordem crescente.

5. Elabore um programa que apresente os valores de conversão de graus Celsius em Fahrenheit, de 10 em 10, iniciando a contagem em 10 graus Celsius finalizando em 100 graus Celsius. O programa deve apresentar o valor das duas temperaturas.

6. Elabore um programa que efetue a leitura de 10 valores numéricos e apresente no final o somatório e a média dos valores lidos.

Realize os exercícios de 7 a 10 utilizando o comando Do-While:

7. Elabore um programa que efetue a leitura sucessiva de valores numéricos e apresente no final o somatório, a média e o total de valores lidos. O programa deve fazer as leituras dos valores enquanto o usuário estiver fornecendo valores positivos. Ou seja, o programa deve parar quando o usuário fornecer um valor negativo (menor que zero).

8. Elabore um programa que apresente os resultados da soma e da média aritmética dos valores pares situados na faixa numérica de 50 a 70.

9. Escreva um algoritmo que leia 2 valores (se o segundo valor informado for zero deve ser lido um novo valor) e imprimir o resultado da divisão do primeiro pelo segundo. Acrescente uma mensagem de Valor Inválido caso o segundo valor informado seja 0. Acrescente uma mensagem de Novo Cálculo (S/N)? ao final deste exercício. Se for respondido 'S' deve retornar e executar um novo cálculo caso contrário deverá encerrar o algoritmo.

10. Escreva um algoritmo para ler as notas da 1ª e 2ª avaliações de um aluno, calcule e imprime a média semestral. Só deve aceitar valores válidos (0 a 10) para cada nota.

Atividades

Exercícios de Strings

1. Construa um algoritmo que receba uma string com o valor Sistemas para Internet e retire os espaços em branco desta palavra.

2. Construa um algoritmo que receba seu primeiro nome, seu nome do meio e seu sobrenome, e que concatene esses nomes com espaços em branco entre eles retornando o nome completo em uma única variável.

3. Construa um algoritmo que leia uma variável do tipo STRING e retorne a mesma em maiúscula.

4. Construa um algoritmo que leia uma variável do tipo STRING e um caractere e mostre como resultado o número de vezes que esse caractere aparece na variável STRING.

5. Construa um algoritmo que leia uma variável do tipo STRING e um caractere, e transforme todos os caracteres iguais ao caractere enviado em MAIÚSCULO.

6. Desenvolva um algoritmo que retorne o valor inverso de uma palavra que é informada. Por exemplo: o inverso da palavra 'jabuticaba' é igual a 'abacitubaj'.

7. Construa um algoritmo que leia duas variáveis do tipo STRING de tamanho 40 e diga se são iguais ou não.

8. Construa um algoritmo que leia duas variáveis do tipo STRING de tamanho 40 e diga quantas vezes a letra a e a letra r aparecem nas palavras.

9. Construa um algoritmo que tenha uma variável do tipo STRING inicializada com COMPUTAÇÃO e troque o ÃO por @@, mostre a palavra transformada.

10. Construa um algoritmo que leia uma variável do tipo STRING e mostre a palavra lida e seu tamanho.

D

FUNÇÕES

Unidade D
Programação Estruturada

UNIDADE D

FUNÇÕES

Conceitos Gerais

Um algoritmo pode ser escrito em pedaços que podem ser executados tantas vezes quantas forem necessárias. Facilitando a detecção de erros, a documentação de sistemas, a manutenção de software e a reutilização de funções já implementadas.

Parâmetros e argumentos

Uma função possui o seguinte formato:

tipo_retorno nome_da_funcao (tipo < **parametro1** >, tipo < **parametro2** >, . . . , tipo <**parametron** >)

{

comandos;

return(**valor de retorno**);

}

Temos que:

- nome_da _funcao é o nome que é dado a função.
- tipo < **parametro1** >, tipo < **parametro2** >, . . . , tipo <**parametron** > são os parâmetros da função. Consiste na definição das variáveis locais à função. Uma função pode não ter parâmetros, basta não informá-los.
- **tipo** < parametro1 >, tipo < parametro2 >, . . . , tipo <parametron > são os tipos de dados dos parâmetros da função.
- <**tipo_retorno**> é o tipo de dado do valor retornado pela função ao algoritmo chamador.
- <**comandos**> é o conjunto de instruções do corpo da função.
- <**return**> é o valor retornado, o seu tipo deve ser o mesmo definido no <**tipo_retorno**>. O **return** é sempre o último a ser executado por uma função, e nada após ele será executado.

Declarando uma função

As funções devem ser declaradas fora do programa principal (main()).

Para a sua declaração basta escrever o cabeçalho da função:

tipo_retorno nome_da_funcao (tipo < **parametro1** >, tipo <**parametron** >)

Chamando uma função

Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor a uma variável:

variavel = funcao (parametros);

Os parâmetros passados pela função não necessariamente possuem os mesmos nomes que os parâmetros que a função espera.

Esses parâmetros serão mantidos intocados durante a execução da função.

Exemplo:

```
#include <stdlib.h>
#include <stdio.h>
float quadrado(float w); //declaração da função
int main()
{  float x,y;
       printf("Info nro:");
       scanf("%f",&x);
       y=quadrado(x); //chamando uma função
       printf("%3.2f",y);
       system("PAUSE");
       return 0;
}
float quadrado(float w)/*calcula o quadrado do número passado como parâmetro*/
{   float  z;
     z=w*w;
     return(z);
}
```

Função sem retorno na Linguagem C

São funções do tipo void. Por exemplo, abaixo temos uma função que imprime o número que for passado para ela como parâmetro.

Exemplo:

```
#include <stdlib.h>
#include <stdio.h>
void pede_numero();//declaração da função
int main()
{  int x;
        pede_numero(); //chamada da função
       scanf("%d",&x);
       printf("Nro informado:%d",x);
       system("PAUSE");
       return 0;
}
void pede_numero()//criação da função
{    printf("Info nro:");

}
```

Para chamar esta função , basta chamar seu nome e seu parâmetro(se houver): **nome_função (parametros);**

Argumentos de função

Há duas formas de passagem de parâmetros em C:

- **Passagem por valor (ou por cópia)**: quando o que é passado para a função é o valor da variável usada como parâmetro.
- **Passagem por referência**: quando o que é passado não é o valor de uma variável, mas o endereço desta variável.

Exemplos:

Passagem por valor

```
#include <stdio.h>
#include <stdlib.h>
void proc(int y);
void main()
{
   int x;
   x=1;
   printf("Antes x=%d\n",x); //x será 1
   proc(x);
   printf("Depois x=%d\n",x); //x será 1
   system("Pause");
}
void proc(int y)
{
   y=y+1;
   printf("Durante y=%d\n",y); //y será 2
}
```

O resultado do algoritmo fornece o seguinte resultado:

Antes x= 1
Durante y= 2
Depois x= 1

Passagem por referência

```
#include <stdio.h>
#include <stdlib.h>
void proc(int *y);
void main()
{
int x;
x=1;
printf("Antes x=%d\n",x); //x será 1
```

```
proc(&x); //com o & passamos o endereço da variável
printf("Depois x=%d\n",x); // x será 2
system("Pause");
}
void proc(int *y) /*o y recebe o endereço, assim se modificarmos y modificaremos
a variável para a qual y está apontando, no caso x.*/
{
*y=*y+1;
printf("Durante y=%d\n",*y); // y será 2
}
```

O resultado do algoritmo fornece o seguinte resultado:

Antes x= 1
Durante y= 2
Depois x= 2

Chamando funções com vetores

Quando passamos vetores como argumentos de uma função, apenas o endereço deste vetor será passado para a função. Assim a função receberá um ponteiro para o primeiro elemento no vetor.

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
void mostra (int num[10]);
int main(int argc, char *argv[])
{
   int v[10],i;
   for(i=0;i<10;i++){
     v[i]=i;
   }
   mostra(v);
   system("Pause");
}
void mostra(int num[10])//imprime os números de 0 a 9.
{
   int i;
   for(i=0;i<10;i++)
     printf("%d \n",num[i]);

}
```

Outra forma de declarar um parâmetro do tipo vetor é como um ponteiro, assim substituímos o int num[10] por int *num:

```
void mostra(int *num)
{
   int i;
   for(i=0;i<10;i++)
     printf("%d \n",num[i]);

}
```

Lembre-se

Em C, um nome de vetor sem qualquer índice é um ponteiro para o primeiro elemento no vetor.

Chamando funções com strings

Quando passamos strings como argumentos de uma função, apenas o endereço desta string será passado para a função. Assim a função receberá um ponteiro para o primeiro elemento na string.

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
void mostra (char *string);
int main(int argc, char *argv[])
{
   char s[40];
   gets(s);//devemos digitar uma string
   mostra(s);
   system("Pause");
}
void mostra(char *string)//imprime a string em maiúscula.
{
   int i;
   for(i=0;string[i]!='\0';i++)
   {
     //transforma cada caracter da string em maiúscula.
     string[i]=toupper(string[i]);

     //imprime caracter por caracter da string
     printf("%c",string[i]);
   }
}
```

Variáveis externas

Uma variável externa pode ser compartilhada por funções que estão localizadas em arquivos fontes distintos. Para isso, usamos o comando **extern**:

```
extern int a;
extern int b;
```

Regras de escopo

Variáveis Globais são reconhecidas pelo programa inteiro. Estas variáveis são visíveis (isto é, podem ser usadas) no algoritmo principal e por todas as funções.

Variáveis Locais são aquelas definidas dentro de uma função e, portanto somente visíveis (utilizáveis dentro da mesma).

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int a;//variável global
int b; //variável global
void troca();
int main()
{
       printf("a:");
       scanf("%d",&a);
       printf("b:");
       scanf("%d",&b);
       printf("Antes %d-%d\n",a,b);/* se informarmos a=1 e b=2 teremos a=1 e
b=2*/
       troca();
       printf("Depois %d-%d\n",a,b); );/* se informarmos a=1 e b=2* teremos
a=2 e b=1*/

       system("Pause");
       return 0;
}
void troca()
{int x; //variável local
        x=a;
        a=b;
        b=x;
  }
```

Inicialização de variáveis

Variáveis globais são inicializadas apenas no começo do programa. Variáveis locais são inicializadas cada vez que o bloco no qual estão declaradas for inserido. Variáveis locais que não são inicializadas possuem valores desconhecidos antes de ser efetuada a primeira atribuição a elas. Variáveis globais não inicializadas são inicializadas com zero.

Recursividade

Em C, uma função pode chamar a si mesma. Quando uma função faz uma chamada a si mesma, dizemos que fez uma chamada recursiva.

Um exemplo simples de função recursiva é o cálculo do fatorial de um número. Para isso temos a seguinte função de recorrência:

0! =1 (Base)
n!= n * (n-1)! para n>0;

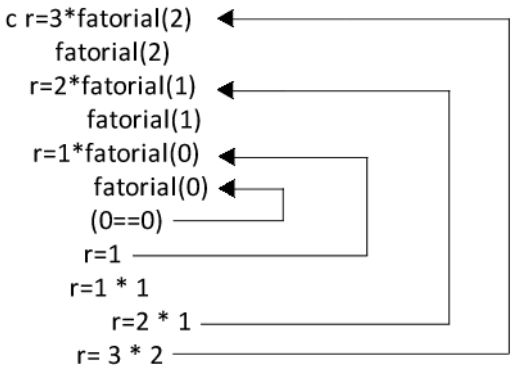
Segundo MEDINA e FERTIG (2005), uma função de recorrência é uma função que é expressa em função de si mesma. Assim temos um valor inicial (base) e depois prova-se que se o teorema é válido para qualquer valor de n, será também para o próximo valor(n+1). Este princípio é muito similar a resolução de funções recursivas.

Agora com a utilização da função de recorrência fica mais fácil a criação da função recursiva do fatorial:

```
#include <stdio.h>
#include <stdlib.h>
int fatorial (int n);
int main(int argc, char *argv[])
{
    int n, fat=1;
    printf("Digite um numero inteiro:");
    scanf("%d", &n);
    fat=fatorial(n);
    printf("\n\nO fatorial de %d e: %d ", n, fat);
    system("Pause");
}
int fatorial (int n)
{
    int r;
    if (n == 0) //BASE e também a condição de parada
       r=1;
    else //para n>0
        r=n*fatorial(n-1);
    return r;
}
```

Se informarmos que queremos o fatorial de 3 teremos o seguinte:

fatorial(3)



Retornará ao final 6

Referências

MANZANO, José Augusto; OLIVEIRA, Jayr Figueiredo. **Algoritmos: Lógica para Desenvolvimento de Programação de Computadores**. 15.ed. São Paulo: Erica, 2003.

MEDINA, Marco; FERTIG, Cristina. **Algoritmos e Programação: Teoria e Prática**. São Paulo: Novatec, 2005.

ORTH, Afonso Inácio. **Algoritmos e programação com resumo das linguagens Pascal e C**. Porto Alegre: AIO, 2001. 175p.

SALIBA, Walter Luiz Caram. **Técnicas de Programação: uma abordagem estruturada**. São Paulo: Makron Books, 1993.

SCHILDT, Herbert. **C, completo e total**. São Paulo: Pearson Makron Books, 1997.

Atividades

Realize os seguintes algoritmos utilizando funções:

1. Faça um algoritmo e nele faça uma função que recebe por parâmetro o raio de uma esfera e calcula o seu volume (v = 4/3.P.R3)

2. Escreva um algoritmo que recebe 3 notas por parâmetro. Em seguida faça uma função que calcule a média aritmética das notas e outra que calcule a média ponderada (pesos: 5, 3 e 2).

3. Faça um algoritmo e nele faça uma função que recebe a idade de uma pessoa em anos, meses e dias e retorna essa idade expressa em dias (Considere que os anos possuem 365 dias, e que os meses possuem 30 dias).

4. Faça um algoritmo e nele faça uma função que recebe um valor inteiro e verifica se o valor é positivo ou negativo. A função deve retornar 1 se for positivo e 0 se for negativo.

5. Faça um algoritmo que possua um vetor de tamanho 10, neste algoritmo crie uma função que mostre os números pares deste vetor.

Atividades

Realize os seguintes algoritmos utilizando funções:

1. Construa uma função que receba a palavra Sistemas para Internet e retire os espaços em branco desta palavra.

2. Construa uma função que receba seu primeiro nome, seu nome do meio e seu sobrenome, e que concatene esses nomes com espaços em branco entre eles retornando o nome completo em uma única variável.

3. Construa uma função que receba como parâmetros uma variável do tipo STRING e retorne a mesma em maiúsculo.

4. Construa uma função que receba como parâmetros uma variável do tipo STRING e um caractere e retorne como resultado o número de vezes que esse caractere aparece na variável STRING.

5. Construa uma função que receba como parâmetros uma variável do tipo STRING e um caractere, e transforme todos os caracteres iguais ao caractere enviado em MAIÚSCULO.

Atividades

Realize os seguintes algoritmos utilizando funções recursivas:

1. Construa uma função recursiva que retorne a multiplicação de dois números por meio de adição. Considerando que temos como função de recorrência:

x*y=x+x*(y-1) para n>0,
x*0=0 (base).

2. Crie uma função recursiva que realize a série de Fibonacci:

f(n)=f(n-1)+f(n-2) para n>1,
f(0)=0 e f(1)=1 (base).

3. Faça uma função recursiva que realize o cálculo do somatório:

$$\sum_{i=1}^{n}(2*i^2+2*i)$$

E

VETORES E APONTADORES

Unidade E
Programação Estruturada

UNIDADE E

VETORES E APONTADORES

Vetores

Vetor é uma coleção de valores que são referenciados por um nome comum. Um elemento específico em um vetor é acessado através de um índice.

Para declarar uma variável do tipo vetor, utilizamos a seguinte sintaxe:

tipo nome_da_variável[tamanho];

Onde:

<**tipo**>: é o tipo de dado do vetor.

<**nome_da_variável**>: é o nome dado ao vetor.

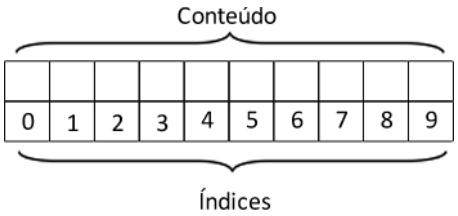
<[tamanho]>: é o tamanho do vetor.

Exemplos:

double salario[100];

int x[10];

Em C, todo vetor tem 0 como índice do seu primeiro elemento. Assim se escrevermos int x[10], teremos um vetor de tamanho 10 e que vai do índice 0 ao índice 9, conforme a tabela abaixo:



Inserindo valores em cada parte de um vetor

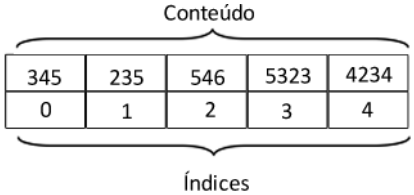
Seja o seguinte vetor:

```
int n[5];
```

Após executarmos as seguintes atribuições:

```
n[0]= 345;
n[1]= 235;
n[2]= 546;
n[3]= 5323;
n[4]= 4234;
```

Teremos um vetor com os seguintes elementos:



Lembre-se

Uma string é definida como um vetor de caracteres que é terminada por um nulo (\0).
char frase[15];

Exemplo de um programa que lê 10 números e os coloca em um vetor de tamanho 10 do tipo inteiro:

```
#include <stdlib.h>
#include <stdio.h>
int main()
{  int numero[10];
    int cont;

    for(cont=0;cont<=9;cont++)
    {
            printf("Digite um numero:");
            scanf("%d",&numero[cont]);
            printf("nro digitado: %d",numero[cont]);
     }
   system("PAUSE");
   return 0;
}
```

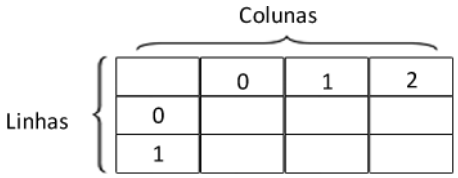
Vetores Multidimensionais

Vetores multidimensionais permitem trabalharmos com mais de uma dimensão ao mesmo tempo. A forma mais simples é o vetor bidimensional, também conhecido por matriz.

Para declarar um vetor bidimensional de inteiros de tamanho 2, 3, utilizamos a seguinte sintaxe:

int vetor[2][3];

Assim, teremos um vetor de 2 linhas por 3 colunas:



Inserindo valores em cada parte de um vetor

Seja o seguinte vetor:

int n[2][3];

Após executarmos as seguintes atribuições:

n[0][0]= 11;
n[0][1]= 12;
n[0][2]= 13;

```
n[1][0]= 14;
n[1][1]= 15;
n[1][2]= 16;
```

Teremos um vetor com os seguintes elementos:

	0	1	2
0	11	12	13
1	14	15	16

Exemplo de um programa que lê e escreve um conjunto de 6 números inteiros e armazena-os em um vetor bidimensional (matriz) de 2 linhas por 3 colunas.

```
#include <stdlib.h>
#include <stdio.h>
int main(){
    int numero[2][3];
    int i,j;
    for(i=0;i<=1;i++)  //for para a linha
    {
        for(j=0;j<=2;j++) //for para a coluna
        {
            printf("Digite um numero:");
            scanf("%d",&numero[i][j]);
            printf("nro digitado:%d\n",numero[i][j]);
        }
    }
    system("PAUSE");
    return 0;
}
```

A matriz número ficará com a seguinte configuração se digitados os números 1,2,3,4,5,6, um após o outro:

	0	1	2
0	1	2	3
1	4	5	6

Inicialização de vetores

C permite a inicialização de vetores na declaração. A forma de declaração é semelhante à de outras variáveis:

int i[10]={1,2,3,4,5,6,7,8,9,10}

Já para vetores bidimensionais (matrizes), a inicialização é feita da seguinte forma:

int n[2][3]={{11,12,13},{14,15,16}}

E para inicializarmos strings fazemos da seguinte maneira:

char s[17]="C é tudo de bom!".

Embora a frase tenha 16 caracteres o vetor deve ter um caracter a mais por causa do '\0'.

Podemos inicializar strings, também, conforme exemplo a seguir:

char s[17]={'C',' ','é',' ','t','u','d','o',' ','d','e',' ','b','o','m','!','\0'};

Apontadores

Apontadores são também conhecidos como ponteiros. Um ponteiro é uma variável que contém um endereço de memória. Esse é o endereço ocupado por outra variável na memória.

Declaração de Apontadores

Para uma variável ser ponteiro, ela deve ser declarada como tal:

tipo *nome;

onde:

<**tipo**>: corresponde ao tipo de dado do apontador.

<***nome***>: nome do ponteiro. O asterisco indica que estamos trabalhando com ponteiro.

Operadores de ponteiros

Existem dois operadores que são utilizados para trabalharmos com ponteiros: * e &.

- O & indica o endereço de memória onde está armazenada a variável na memória.
- O * indica o valor armazenado no endereço que o ponteiro está apontando.

Exemplo:

int i;

int *p; //declaração de um variável ponteiro do tipo inteiro

p=&i; //atribuição do endereço da variável i

Lembre-se

Sempre devemos inicializar um ponteiro com o endereço de uma variável.

```
int i, *p;
p=&i;
```

Ilustrando o funcionamento dos ponteiros:

```
#include <stdlib.h>
#include <stdio.h>
int main()
{   int i,*p;
    i=10;
```

```
    p=&i;
//o p está apontando para a variável i, assim terá conteúdo 10

    printf("A variavel item tem conteudo: %d  e o ponteiro %d",i,*p); //
mostrará 10 para i e 10 para *P
    printf("Endereco do ponteiro %p",p);
//através do %p mostrará o endereçco de p
    system("PAUSE");
   return 0;
}
```

Agora, se modificarmos o conteúdo da variável para a qual o ponteiro aponta:

```
#include <stdlib.h>
#include <stdio.h>
int main()
{   int i,*p;

    i=10;
    p=&i;
//o p está apontando para a variável i, assim terá conteúdo 10

   printf("A variavel item tem conteudo: %d  e o ponteiro %d\n",i,*p);
 //mostrará 10 para i e 10 para *P

   printf("Endereco do ponteiro %p\n",p);
//através do %p mostrará o endereço de p

    i=20;
   printf("A variavel item tem conteudo: %d  e o ponteiro %d\n",i,*p); //
mostrará 20 para i e 20 para *P

    *p=30;
//como o p está apontando para a variável i, o conteúdo de i também será
modificado.
   printf("A variavel item tem conteudo: %d  e o ponteiro %d\n",i,*p); //
mostrará 30 para i e 30 para *P

   system("PAUSE");
   return 0;
}
```

Utilizando vetores e ponteiros.

```
#include <stdlib.h>
#include <stdio.h>
int main()
{   int *p;
    int v[3]={5,0,2};

    p=v;
//ou p=&v[0], significa que o ponteiro receberá a primeira posição do vetor

    printf("Ponteiro: %d  e 1a posicao do vetor: %d\n",*p,v[0]);
//mostrará 5 para *p e 5 para v[0]
    p++;
//incrementa o ponteiro, assim irá apontar para o próximo elemento do vetor.
    printf("Ponteiro: %d  e 2a posicao do vetor: %d\n",*p,v[1]);
//mostrará 0 para *p e 0 para v[1]
    p++;
//incrementa o ponteiro, assim irá apontar para o próximo elemento do vetor.
    printf("Ponteiro: %d  e 3a posicao do vetor: %d\n",*p,v[2]);
//mostrará 2 para *p e 2 para v[2]

    system("PAUSE");
    return 0;
}
```

Referências

MEDINA, Marco; FERTIG, Cristina. **Algoritmos e Programação: Teoria e Prática**. São Paulo: Novatec, 2005.

MORAES, Celso Roberto. **Estrutura de dados e algoritmos: uma abordagem didática**. São Paulo: Futura, 2003.

SCHILDT, Herbert. **C, completo e total**. São Paulo: Pearson Makron Books, 1997.

Atividades

1. Preencher um vetor V de 10 elementos com os números inteiros 1,2,3,4,5,...,10. Escrever o vetor V após o seu total preenchimento.

2. Preencher um vetor número de 20 elementos com 1 se o índice do elemento for ímpar, e 0 se for par. Escrever o vetor número após o seu total preenchimento.

3. Seja o seguinte vetor v={1,-4,5,-7,9,-8,3,6,12,-15}. Trocar todos os valores negativos do vetor por 0. Escrever o vetor modificado.

4. Ler um vetor nome de 10 nomes e uma variável A que contenha o nome de uma pessoa. Escrever a mensagem ACHEI se o nome armazenado em A estiver no vetor nome, e NÃO ACHEI caso contrário. Obs: a mensagem deverá aparecer somente uma vez.

5. Ler um vetor de 10 elementos. Conte e escreva quantos são múltiplos de 5.

Atividades

1. Ler um vetor que contenha o nome de cinco lojas. Em seguida verifique se a loja Lar existe no vetor e imprima a mensagem Lar existe no vetor.

2. Faça um programa que leia um vetor de seis elementos numéricos inteiros. Calcule e mostre a quantidade de números pares e quantidade de números ímpares.

3. Ler um vetor que contenha as notas de 10 alunos. Criar uma função que calcule a média da turma e outra que conte quantos alunos obtiveram nota acima desta média calculada. Escrever a média da turma e o resultado da contagem.

4. Ler um vetor X de 10 elementos. A seguir copiar todos os valores negativos do vetor X para um vetor Y, sem deixar elementos vazios entre os valores copiados. Escrever o vetor X e o vetor Y.

5. Ler um vetor P de 10 posições (aceitar somente números positivos). Escrever a seguir o valor do maior elemento de P e a respectiva posição que ele ocupa no vetor.

F

MANIPULAÇÃO DE ARQUIVOS

Unidade F
Programação Estruturada

UNIDADE F

MANIPULAÇÃO DE ARQUIVOS

Entrada/Saída padrão

O ponteiro do arquivo

Para trabalharmos com arquivos utilizamos a variável ponteiro do tipo FILE. O comando deve ser utilizado da seguinte forma:

FILE *fp;

Abrindo um arquivo

FILE *fopen(char *nomearq, char *modo_abertura)

Onde:

<nomearq> é um ponteiro para o nome de um arquivo.

<modo_abertura> determina como o arquivo será aberto.

A string modo_abertura pode conter alguns dos seguintes caracteres:

Modo	Significado
r	Abre um arquivo-texto para leitura.
w	Cria um arquivo-texto para escrita.
a	Anexa a um arquivo-texto.
rb	Abre um arquivo-binário para leitura.
wb	Cria um arquivo-binário para escrita.
ab	Anexa a um arquivo binário.
r+	Abre um arquivo-texto para leitura/escrita.
w+	Cria um arquivo-texto para leitura/escrita.
a+	Anexa ou cria um arquivo-texto para leitura/escrita.
r+b	Abre um arquivo binário para leitura/escrita.
w+b	Cria um arquivo binário para leitura/escrita.
a+b	Anexa um arquivo binário para leitura/escrita.

```
FILE *fp;
if(fopen("teste","w")==NULL)
{       printf("Arquivo nao pode ser aberto\n");
                exit(1);
}
```

Fechando um arquivo

int fclose(FILE *arquivo)

Fecha um arquivo. Retorna 0 se OK, qualquer outro valor indica erro.

fclose(arquivo);

Escrevendo um caractere

int fputc(FILE *arquivo)

Escreve um caracter no arquivo. Retorna o caracter lido ou EOF em caso de erro ou final de arquivo.

if (fputc('x',arq)==EOF)

printf("Nao foi possivel gravar");

Lendo um caractere

int fgetc(FILE *arquivo)

Lê um caracter do arquivo. Retorna o caracter lido ou EOF em caso de erro ou final de arquivo. A seguir, um exemplo que utiliza os comandos fopen, fgetc e fclose.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fp;
    int c;
   if((fp=fopen("teste.txt","rt"))==NULL)
    {
        printf("O arquivo nao pode ser aberto");
        system("Pause");
        exit(1);
    }
    do{//lê cada caracter do arquivo e imprime
        c=fgetc(fp);
        if (c==EOF) //se é final de arquivo
     break;
        printf("%c",c);
     }
     while(1);
     fclose(fp);
     system("Pause");
  }
```

Usando feof()

int feof(FILE *fp)

Retorna verdadeiro se o final de arquivo foi atingido; caso contrário, devolve 0.

Trabalhando com strings

int fputs(char *str, FILE *fp)

Escreve uma string em um arquivo, retorna o último caracter gravado ou EOF em caso de erro.

char *fgets(char *str, int tamanho, FILE *fp)

Lê uma linha de texto do arquivo, retorna um ponteiro para a string lida ou NULL em caso de erro ou final de arquivo.

Usando Rewind

void rewind(FILE *fp)

O ponteiro é apontado para o início do arquivo, é como se o arquivo fosse rebobinado.

Usando fputs, fgets, rewind e feof. O exemplo a seguir lê strings do teclado até ser pressionada a tecla enter, em seguida as grava no arquivo e mostra o conteúdo do arquivo.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
    FILE *fp;
    char s[100];
    if((fp=fopen("teste.txt","w+"))==NULL)
    {   printf("O arquivo nao pode ser aberto");
        system("Pause");
        exit(1);
    }
    do{
        printf("Digite uma string (enter para sair)!\n");
        gets(s);
        strcat(s,"\n"); //adiciona uma nova linha a string
        fputs(s,fp) ; //coloca a string no arquivo
      }
     while(*s!='\n');
     rewind(fp);
     while(!feof(fp)) //testa se não é final de arquivo
     {
         fgets(s,99,fp); //pega a string do arquivo
         printf("%s",s);
     }
```

```
        fclose(fp);
        system("Pause");
        return 0;
    }
```

Apagando Arquivos

int remove(char *filename)

Apaga o arquivo especificado. Ele devolve 0, caso consiga, e um valor diferente de 0, caso contrário.

O exemplo tenta apagar o arquivo "teste.txt", caso consiga imprime mensagem mostrando o fato:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
       if(remove("teste.txt"))
        {
            printf("O arquivo nao pode ser apagado!\n");
            system("Pause");
            exit(1);
        }
      else
         printf("Arquivo apagado!\n");


        system("Pause");
        return 0;
}
```

Usando fread() e fwrite()

Essas funções permitem a leitura e escrita de blocos de qualquer tipo de dado. Elas são utilizadas quando trabalhamos com arquivos binários.

A função **fread** devolve o número de itens lidos (não o número de bytes).

size_t fread(void *buffer, size_t num_bytes, size_t count, FILE *fp)

onde:

- **<buffer>**: Ponteiro para uma região da memória, geralmente uma estrutura ou um vetor, onde serão armazenados os dados lidos do arquivo.
- **<num_bytes>**: Número de bytes a ler. Geralmente sizeof(buffer).
- **<count>**: Número de itens a ler. Geralmente 1.
- **<fp>**: Ponteiro FILE para um arquivo aberto anteriormente por fopen().

A função **fwrite** devolve o número de itens gravados.

size_t fwrite (void *buffer, size_t num_bytes, size_t count, FILE *fp)

onde:

- **<buffer>:** Ponteiro para uma região da memória, geralmente uma estrutura ou um vetor, onde estão armazenados os dados a serem escritos no arquivo.
- **<num_bytes>:** Número de bytes a escrever. Geralmente sizeof(buffer).
- **<count>:** Número de itens a escrever. Geralmente 1.
- **<fp>:** Ponteiro FILE para um arquivo aberto anteriormente por fopen().

Exemplo da utilização do fread e fwrite:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *pf;
float pi = 3.1415;
float pilido;
if((pf = fopen("arquivo.bin", "wb")) == NULL) /* Abre arquivo binário para
escrita */
{
    printf("Erro na abertura do arquivo");
    system("Pause");
    exit(1);
}
if(fwrite(&pi, sizeof(float), 1,pf) != 1)     /* Escreve a variável pi */
    printf("Erro na escrita do arquivo");
fclose(pf);                                    /* Fecha o arquivo */
if((pf = fopen("arquivo.bin", "rb")) == NULL) /* Abre o arquivo novamente para
leitura */
{
    printf("Erro na abertura do arquivo");
    system("Pause");
    exit(1);
}
if(fread(&pilido, sizeof(float), 1,pf) != 1)  /* Lê em pilido o valor da
variável armazenada anteriormente */
    printf("Erro na leitura do arquivo");
printf("\nO valor de PI, lido do arquivo e': %f", pilido);
fclose(pf);
system("Pause");
return(0);
}
```

Referência

SCHILDT, Herbert. **C, completo e total**. São Paulo: Pearson Makron Books, 1997.

ATIVIDADES

Resolver os seguintes algoritmos utilizando arquivos.

1. Fazer um programa que leia uma string e a grave em um arquivo chamado palavra.txt. A palavra deverá ser gravada caracter a caracter (utilizar a função fputc).

2. Fazer um programa que leia uma string e a grave em um arquivo chamado palavra2.txt. Utilizar a função fputs.

3. Construir um programa que leia 10 palavras do teclado e as guarde em um arquivo. Em seguida, utilizar a função fgets para mostrar as strings armazenadas.

4. Construir um programa que lê uma string e verifica se ela existe em um arquivo de texto. Imprimir “String tal encontrada no arquivo!” em caso de encontrá-la e, em caso contrário, “String tal não encontrada no arquivo!”.

5. Escrever um programa que leia um arquivo origem e copie seu conteúdo para um arquivo destino.

6. Escrever um programa que lê n números inteiros e os armazena em um arquivo binário. Em seguida, o programa deve ler o arquivo e imprimir os números armazenados.