**Operações Básicas na Linguagem C**

A principal finalidade de qualquer programa de computador é a manipulação de dados. Assim, uma importante característica a ser estudada em uma linguagem de programação é a forma com que ela trabalha com as informações. Variáveis e constantes são os objetos de dados básicos manipulados em um programa. As declarações listam as variáveis a serem usadas e definem os tipos que as mesmas devem ter e talvez seus valores iniciais. Os operadores especificam o que deve ser feito com as variáveis. As expressões combinam variáveis e constantes para produzir novos valores.

***Ciclo de Desenvolvimento de um Programa***

Resumidamente tem-se abaixo um fluxograma das principais etapas para o desenvolvimento de um programa.

**Determinação e entendimento do problema :**

É sem dúvida a principal etapa deste ciclo, pois um problema mal entendido ou mal interpretado, levará a soluções erradas. Portanto deve-se entender exatamente o problema a ser solucionado, esquematizando a solução por meio um algoritmo, representado por algumas ferramentas como, por exemplo, o fluxograma ou pseudocódigo.

**1- Codificação:** Uma vez esquematizado a maneira como se pretende solucionar o problema, devemos iniciar a codificação do programa que nada mais é que escrever o programa em uma linguagem e usando regras que o compilador da linguagem entenda, gerando desta maneira um CÓDIGO FONTE (arquivo.c). Para isto, pode-se utilizar a maioria dos editores de texto. Em nosso caso, estaremos usando um editor de texto do próprio ambiente de programação.

**2- Compilação:** É o processo através do qual um compilador lê o programa inteiro codificado e converte-o em um CÓDIGO OBJETO (arquivo.obj), que é uma tradução do CÓDIGO FONTE do programa em uma forma que o computador possa executar diretamente. O CÓDIGO OBJETO é também conhecido como código binário ou código de máquina.

**3- Link-Edição:** Todo compilador C vem com uma biblioteca padrão de funções que realizam as tarefas necessárias mais comuns, como exemplo escrever no vídeo (*printf()*), ler um caracter do teclado (*getch()*), entre outras. O padrão ANSI C (*American National Standards Institute*) especifica o conjunto mínimo de funções que estará contido na biblioteca. No entanto seu compilador provavelmente conterá outras funções que podem não estar incluídas em um compilador de outro fabricante. O processo de link-edição é combinar o CÓDIGO OBJETO gerado pelo compilador que você escreveu, com o código objeto já encontrado na biblioteca padrão quando da chamada de uma função desta biblioteca, por exemplo *printf*() e *getch*(). O resultado deste processo é o CÓDIGO EXECUTÁVEL (arquivo.exe , no DOS).

**4- Depuração e Testes:** Uma vez gerado o código executável passa-se para a etapa de verificação de funcionamento do programa, entrando em detalhes como valores de variáveis, fluxo dos dados, etc..., isto é na realidade o que chamamos de Depuração, e o ambiente a ser utilizado possui uma ferramenta poderosa para tal, o DEBUG. Uma vez toda que toda a parte lógica do programa está certa, parte-se para os testes. Se nesta etapa qualquer problema é encontrado, devemos iniciar todo o processo, passando novamente pelas etapas 1, 2, 3 e 4.

***Estrutura básica de um programa em C***

Antes de explorar alguns detalhes de estrutura é necessário conhecer a estrutura básica de um programa na linguagem C.

O programa seguinte tem como objetivo único e exclusivo mostrar os diversos componentes de um programa em C.

**/\* Cabeçalho do programa \*/**

// Programa : **Modelo Geral**

// Programador : Fulano de Tal

// Data :

// Objetivo:

// Dados de entrada:

// Dados de saída:

**/\* Declaração das Bibliotecas \*/**

**#include** <stdio.h>

**#include** <conio.h>

**/\* Definição dos Tipos \*/**

**typedef enum{** FALSE, TRUE**}** boolean;

**/\*Definição das Constantes \*/**

**#define** MEU\_NOME “Marcos Junior”

**/\* Prototipação das Funções \*/**

**void** Linha( );

**int** Soma (int x, int y);

// **Aqui começa a função principal - programa propriamente dito.**

**main ( )**

**{**

**/\*Declaração das Variáveis da função principal \*/**

int idade;

float altura;

char nome[30],

sexo;

boolean estudante;

clrscr( ); // Limpa a tela

Linha( ); // Chamada da função Linha

printf(“ Meu nome e --> %s \n”, MEU\_NOME);

Linha( );

printf(“ Qual eh o seu nome ? ”);

gets(nome);

Linha( );

printf(“ Qual e o seu sexo - F (feminino) e M (masculino) ? ”);

sexo= getchar( );

Linha( );

printf(“ Qual a sua idade ?”);

scanf( “%d”, &idade);

Linha( );

printf(“ Nossas idades somam \_ %d \n” , Soma(20,idade));

Linha( );

estudante = TRUE;

printf(“Prazer em conhecê-lo (a)”)

getch( );

**}**

**// Definição do corpo de cada função prototipada**

**void** Linha ( )

**{**

int i ;

**for**( i = 0; i< 80 ; i++)

printf(“—” );

**}**

int Soma (int x, int y)

**{**

**return** (x + y);

**}**

Análise de cada parte do programa:

**Cabeçalho do Programa:** É uma descrição das informações principais do programa que ajudem a elucidar o funcionamento do mesmo, tais como nome, data, nome do programador, dados de entrada, saída, objetivo, entre outros.

**Declaração das Bibliotecas, Definições de Pré-Processamento ou Arquivos-Cabeçalhos:** Existem funções pré-definidas em C e para utilizá-las em um programa é necessário declarar a biblioteca a qual a função está definida. Quando é iniciada a compilação, a primeira tarefa do compilador é executar automaticamente o pré-processador, que consiste em ler o seu código fonte procurando por linha que iniciam com # e expande o seu programa fonte com o conteúdo do arquivo indicado.

**Definição dos Tipos**: em C existe cinco tipos de dados pré-definidos: inteiro, ponto flutuante (*float*), ponto flutuante duplo (*double*), caracter e sem valor. No entanto, pode-se definir novos tipos de dados nesta parte do programa. Todas as variáveis do programa devem ser declaradas no início da função em que ela será usada.

**Definição das Constantes:** uma constante simbólica é um nome que substitui uma seqüência de caracteres. Ela permite que um nome apareça no lugar de uma constante numérica, uma constante caracter ou uma constante *string*. Quando um programa é compilado, cada ocorrência de uma constante simbólica é substituída

pela sua seqüência de caracteres correspondente.

**Prototipação das Funções:** Nesta parte deve-se declarar as funções definidas pelo programador. Elas são chamadas durante a execução do programa pelos seus respectivos nomes. A linguagem C é baseada no conceito de blocos de construção. Os blocos de construção são chamados de funções.

Um programa em C é uma coleção de uma ou mais funções. Para escrever um programa, você primeiro cria funções e, então, coloca-as juntas. Em C, uma função é uma sub-rotina que contém uma ou mais declarações e realiza uma ou mais tarefas. Em programas bem escritos, cada função realiza somente uma tarefa.

Cada função tem um nome e uma lista de argumentos que receberá. Em geral, pode-se dar qualquer nome a uma função, com exceção de **main**.

**Função Principal ( main ):** É a função por onde é iniciado a execução de um programa em C. Todo programa em C deve possuir a função principal. Para delimitar a área de uma função usa-se chaves ( **{ }** ) . Entre as chaves está o corpo da função, chamado também de bloco.

**Corpo da Função Principal:** O corpo da função principal é dividido em duas partes:

**Declaração das Variáveis:** Nesta parte deve-se declarar as variáveis que serão utilizadas na função.

**Instruções:** Após definir as variáveis, inicia-se a seqüência de comandos que o programa deve executar.

**Corpo de cada Função:** Para cada função prototipada deve-se especificar suas variáveis e seus comandos como na função principal.

O nosso programa inicialmente apresentará as seguintes partes:

**/\* Cabeçalho do programa \*/**

// Programa: **Modelo Básico**

// Programador: Fulano de Tal

// Data:

// Objetivo:

// Dados de entrada:

// Dados de saída:

**/\* Declaração das Bibliotecas \*/**

**#include** <stdio.h> //funções para entrada e saída de dados

**#include** <conio.h> //funções para controle de vídeo

**/\*Definição das Constantes \*/**

**/\* Função principal \*/**

**main ( )**

**{**

**/\*Declaração das Variáveis da função principal \*/**

**/\*Instruções \*/**

}

***Comentários***

Comentários são textos que se introduz no meio do programa fonte com a intenção de torná-lo mais claro. Tudo que estiver entre os símbolos **/\*** e **\*/** são considerados comentários, o qual não tem sentido para o **compilador.**

As linhas de comentários são extremamente importantes para garantir uma boa compreensão do programa e, automaticamente, sua manutenção. Nunca poupe comentários.

Você sempre vai agradecer quando precisar mexer novamente no programa.

No programa, após duas barras **//** até o final da linha, o compilador considera como comentário.

**Exemplo:**

/\* Isto é um comentário \*/ - ignora toda a frase

teste = 1; // Isto é um teste - ignora a frase “Isto é um teste”

***Variáveis***

**Nome de Variáveis**

Cada linguagem possui suas regras para dar nome a uma variável.

Identificadores são os nomes escolhidos para representar constantes, variáveis, tipos, funções. Estes nomes obedecem as seguintes regras:

a) O identificador deve ter como primeiro caracter uma letra ou underscore ' \_ ';

b) Após a primeira letra o identificador só pode conter: letras, números ou underscore ' \_ ';

c) O identificador não pode conter espaços e caracteres especiais.

d) Não pode ser uma palavra reservada.

e) É uma prática tradicional do C, usar letras minúsculas para nomes de variáveis e maiúsculas para nomes de constantes. Isto facilita na hora da leitura do código;

f) Quando se escreve código usando nomes de variáveis em português, evita-se possíveis conflitos com nomes de rotinas encontrados nas diversas bibliotecas, que são em sua maioria absoluta, palavras em inglês.

Deve-se ressaltar que a linguagem C é "Case Sensitive", isto é, maiúsculas e minúsculas fazem diferença. Caso se declare uma variável com o nome **soma** ela será diferente de **Soma**, **SOMA**, **SoMa** ou **sOmA**.

Da mesma maneira, os comandos do C **if** e **for**, por exemplo, só podem ser escritos em minúsculas pois senão o compilador não irá interpretá-los como sendo comandos, mas sim como variáveis. mas o uso de palavras reservadas não é uma coisa recomendável de se fazer pois pode gerar confusão.

**Observações:**

- Utilize sempre nomes que lembrem a finalidade do que está sendo identificado.

- Embora seja permitido em algumas versões de compiladores, não é conveniente usar

acentuação em nomes de identificadores.

**Exemplos:**

Meu\_Nome - válido

MEU\_NOME - válido

\_Linha - válido

Exemplo23 - válido

2teste - não válido, começa com número

Exemplo 23 - não válido, tem um espaço

X ...Y - não válido

#max - não válido

**Constantes**

Uma constante é um identificador que recebe um nome e é usada para guardar um valor que **não** pode ser modificado durante a execução do programa.

A constante precisa ser inicialmente declarada, e isso é feito com a diretiva ***#define***, normalmente no início do programa ou em arquivos de cabeçalho (.h).

As variáveis constantes devem ser definidas de acordo com a sintaxe seguinte:

**SINTAXE: #define <*nome\_da\_constante* > <*valor\_da\_constante****>*

As constantes podem ser de qualquer tipo de dados básicos. O valor da constante é que define o tipo da constante, ou seja, o modo como cada constante é representada depende do seu tipo.

**Exemplo:**

// Programa : **Exemplo**

// Objetivo: **Mostrar a declaração de constantes e seus tipos**

// Dados de entrada:

// Dados de saída:

#define FALSO 0

#define MAX 1000

#define ERRO “Ocorreu um erro !!!”

#define VERDADEIRO ‘V’

#define PI 3.1415926

**main ()**

**{**

...

**}**

**Variáveis**

Uma variável é uma posição de memória que recebe um nome e é usada para guardar um valor que pode ser modificado durante a execução do programa.

Para ser utilizada, a variável precisa ser inicialmente declarada. A forma geral para declaração de variáveis é:

**SINTAXE:**

***<tipo de dado> variável\_1, variável\_2, ... , variável\_n;***

Ou,

***<tipo de dado> variável\_1,***

***variável\_2,***

***variável\_n;***

ou ainda;

***<tipo de dado> variável\_1;***

***<tipo de dado> variável\_2;***

***Tipos de Dados***

Inicialmente serão utilizados dados do tipo simples, ou seja, os tipos de dados já definidos pela linguagem.

Em C, existem cinco tipos de dados: Inteiro ( int ), Real (float), Duplo (double), caracter (char) e sem valor (void ).



Com exceção do tipo **void**, os tipos básicos de dados poderão ter **modificadores** precedendo-os. Um modificador é usado para alterar o significado do tipo base para adequá-los melhor às necessidades do programa, aumentando a capacidade de armazenamento das informações.

Os modificadores são: **signed (com sinal), unsigned (sem sinal), long (longo), short (curto)**. Assim, tem-se os seguintes tipos de dados.



**Observações:**

* O tamanho de cada tipo pode ser diferente em computadores ou compiladores C diferentes, portanto, se for necessário saber exatamente o tamanho, é importante testá-lo no ambiente em uso. Veja, posteriormente, a utilização do operador ***sizeof*** para determinar o tamanho de um tipo de dados.
* Na tabela tem-se o tipo de dados *char* que é o tipo utilizado para caracteres, no entanto, sua faixa de dados está indicada com números. A explicação é que a linguagem C converte os números em caracteres através da tabela ASCII.

**Exemplo:**

// Programa : **Exemplo**

// Objetivo: **Mostrar a declaração de variáveis e seus tipos**

**main ( )**

**{**

int X; // resultado do acréscimo

char resposta; // resposta para executar o programa 'S' ou 'N'

float preco ; // preço do produto a ser reajustado

**...**

**}**

***Operador de Atribuição***

O operador utilizado para atribuição na linguagem C é o **=** . A variável que recebe o valor é especificada a esquerda do operador e o valor atribuído é especificado a direita do operador.

**Exemplo**:

// Programa : **Exemplo**

// Objetivo: **Mostrar as atribuições de variáveis em C**

**main ()**

**{**

int x; // armazena resultado das atribuições

int y; // armazena um valor constante

y = 50; // a variável y recebe o valor 50

x = 10; // a variável x recebe o valor 10

x = y; // a variável x recebe o valor armazenado em y

x = y + 2; // x recebe o valor armazenado em y mais dois

**}**

***Inicialização de Variáveis***

É possível combinar uma declaração de variável com o operador de atribuição para que a variável tenha um valor ao mesmo tempo de sua declaração. Isto é chamado de inicialização de variáveis.

Isto é importante, pois quando o C cria uma variável ele não a inicializa. Isto significa que até que um primeiro valor seja atribuído à nova variável ela tem um valor indefinido e que não pode ser utilizado para nada. Nunca presuma que uma variável declarada vale zero ou qualquer outro valor.

**Exemplo:**

// Programa : **Exemplo**

// Objetivo: **Mostrar a inicialização de variáveis em C**

**main ( )**

**{**

int x = 5;

char ch = 'A';

float altura = 1.60;

**...**

**}**

***Expressões Aritméticas***

Para a construção de programas todas as expressões aritméticas devem ser linearizadas, ou seja, colocadas em linhas. É importante também ressalvar o uso dos operadores correspondentes da aritmética tradicional para a computacional.

****

**Operadores Aritméticos**

Para formar uma expressão matemática utiliza-se os operadores matemáticos. Os operadores matemáticos são representados da seguinte forma:

**Operação Símbolo**

****

**Observações:**

* Utilizam-se parênteses para indicar a prioridade das operações.
* O operador Resto da Divisão ( % ) é utilizado somente com variáveis do tipo inteiro
* O operador / (divisão) quando aplicado a variáveis inteiras, nos fornece o resultado da divisão inteira; quando aplicado a variáveis em ponto flutuante nos fornece o resultado da divisão "real".



// Programa : **Exemplo**

// Objetivo: **Mostrar a atribuição de expressões a variáveis em C**

**main ( )**

{

int x, y;

y = 2;

x = 10;

x = y + 2;

x = ((y+3)\*4) - 5; // x recebe o valor 15

x = (x % 2); // x recebe o resto da divisão, igual a 1

}

// Programa: **Exemplo**

// Objetivo: **Mostrar a atribuição de expressões a variáveis em C**

**main ( )**

{

int a = 17, b = 3;

int x, y;

float z = 17., z1, z2;

x = a / b; // o quociente da divisão: x é 5

y = a % b; // o resto da divisão: y é 2

z1 = z / b; // o quociente da divisão: z1 é 5.666667

z2 = a / b; // o quociente da divisão: z2 é 5.0

}

**Observação**:

Note que, na linha do exemplo 3a correspondente a z2, primeiramente é feita uma divisão inteira (pois os dois operandos são inteiros). Somente após efetuada a divisão é que o resultado é atribuído a uma variável *float*.

**Funções matemáticas pré-definidas**

Um programa, escrito em linguagem C, pode fazer uso das seguintes funções matemáticas já definidas na biblioteca **math.h .**



**Exemplo:**

// Programa : **Exemplo**

// Objetivo: **Mostrar as funções matemáticas básicas em C**

#include <math.h>

#include <stdio.h>

**main ( )**

**{**

double a , b, c, d ;

a = fabs (-3.5);

b = log (a);

c = sqrt (b);

d = pow ( a, b);

printf (“ a = %6.2f , b = %6.2f , c = %6.2f , d = %6.2f ”, a, b, c, d);

**}**

**A Função printf( )**

A função printf(),da biblioteca **stdio.h,** é uma função que permite efetuar a saída de informações no vídeo.

SINTAXE:

**printf ( “ expressão de controle ” ) ;**

**ou**

**printf ( “expressão de controle ” , lista de argumentos ) ;**

* Na expressão de controle tem-se uma descrição de tudo que a função vai colocar na tela. Ela mostra não apenas os caracteres que devem ser colocados na tela, mas também quais as variáveis e suas respectivas posições. Isto é feito usando-se os códigos de controle, que usam a notação %.Na expressão de controle indica-se quais, de qual tipo e em que posição estão as variáveis a serem apresentadas. Portanto, a expressão de controle pode conter: texto, especificadores ou códigos de formato, códigos de barra invertida, por exemplo.
* A lista de argumentos representa os dados de saída individuais, que podem ser constantes, variáveis, expressões ou funções.
* É muito importante que, para cada código de controle, tenha-se um argumento na lista de argumentos. Apresenta-se a seguir alguns dos códigos % e alguns códigos de barra invertida:





// Programa : **Exemplo**

// Objetivo: **Mostrar a função de saída printf( )**

#include <stdio.h>

**main ( )**

**{**

int num;

num = 1;

printf ("Eu sou um simples");

printf ("computador. \n");

printf ("Meu numero favorito eh %d porque ele eh o primeiro. \n", num);

**}**

// Programa : **Exemplo**

// Objetivo: **Mostrar a função de saída printf( )**

#include <stdio.h>

**main ( )**

**{**

printf ("%s esta a %d milhoes de milhas \n do Sol.", "Venus", 67);

**}**

// Programa : **Exemplo**

// Objetivo: **Mostrar a função de saída printf( )**

#include <stdio.h>

**main ( )**

**{**

printf ("Teste %% %%\n");

printf ("%f\n",40.345);

printf ("Um caracter %c e um inteiro %d\n",'D',120);

printf ("%s%d%%","Juros de ",10);

**}**

Na função *printf( )* é possível estabelecer o tamanho mínimo para a impressão de um campo. Veja o exemplo seguinte.

// Programa : **Exemplo**

// Objetivo: **Mostrar a função de saída printf( )**

#include <stdio.h>

**main ( )**

**{**

printf ("Os alunos são %2d. \n", 350);

printf ("Os alunos são %4d. \n", 350);

printf ("Os alunos são %5d. \n", 350);

**}**

Pode-se ainda usar tamanho de campos com números em ponto flutuante para se obter precisão e arredondamento, bem como alinhar à direita ou à esquerda.

// Programa : **Exemplo**

// Objetivo: **Mostrar a função de saída printf( )**

#include <stdio.h>

**main ( )**

**{**

printf ("%4.2f\n", 3456.78);

printf ("%3.2f\n", 3456.78);

printf ("%3.1f\n", 3456.78);

printf ("%10.3f\n", 3456.78);

**}**

**A Função scanf( )**

Para executar um comando de entrada de dados, utiliza-se a função ***scanf( )*** da biblioteca **stdio.h** .

**SINTAXE: scanf ( “ % t ” , & NOME\_DA\_VARIÁVEL ) ;**

* O NOME\_DA\_VARIÁVEL indica em qual variável o valor lido será armazenado
* O símbolo ‘**&**’ indica que o valor lido deve ser armazenado no endereço referente a variável especificada, ou seja, qual a localização, na memória do micro, da variável definida.
* **%t** é um conversor que indica o tipo de informação que deve ser lida. Deve-se substituir a letra **t** pelo caracter de acordo com a tabela:



// Programa : **Exemplo**

// Objetivo: **Mostrar a função de entrada scanf( )**

#include <stdio.h>

**main ( )**

**{**

int valor;

printf ("Digite um valor inteiro e tecle Enter : ");

scanf ("%i", &valor);

printf ("Valor em Decimal = %d \n", valor);

printf ("Valor em Hexadecimal = %x \n", valor);

printf ("Valor em Octal = %o \n", valor);

printf ("Valor em caracter (ASCII) = %c \n", valor);

**}**

// Programa : **Exemplo**

// Objetivo: **Mostrar a função de entrada scanf( )**

#include <stdio.h>

**main ( )**

**{**

char a;

printf ("Digite um caracter e veja-o em decimal, ");

printf ("octal e hexadecimal. \n");

scanf ("%c", &a);

printf ("\n%c=%d decimal, %o octal e %x hexadecimal \n", a, a, a, a);

**}**

// Programa : **Exemplo**

// Objetivo: **Mostrar a função de entrada scanf( )**

#include <stdio.h>

**main ( )**

**{**

int dia, mes, ano;

printf("Digite a data de hoje no formato dd-mm-aaaa ");

scanf(“%d-%d-%d”, &dia, &mes, &ano);

printf("Data: %d-%d-%d \n“, dia, mes, ano);

**}**

**As funções getche( ) e getch ( )**

A biblioteca **conio.h** oferece funções que lêem um caracter no instante em que é digitado, sem esperar **[enter].** Para isto as funções mais usadas, quando se está trabalhando em ambiente DOS ou Windows, são *getch()* e *getche().*

Ambas retornam o caractere pressionado. *getche()* imprime o caracter na tela antes de retorná-lo e *getch()* apenas retorna o caracter pressionado sem imprimi-lo na tela.

Geralmente estas funções não estão disponíveis em ambiente Unix (compiladores cc e gcc) e podem ser substituídas pela função *scanf()*, porém sem as mesmas funcionalidades.

**Exemplo:**

// Programa : **Exemplo**

// Objetivo: **Mostrar a função de entrada getch ( )**

#include <stdio.h>

#include <conio.h>

/\* Este programa usa conio.h . Se você não tiver a conio, ele não funcionará no Unix \*/

**main ( )**

**{**

char Ch;

Ch=getch();

printf ("Voce pressionou a tecla %c", Ch);

**}**

***Variáveis do tipo Cadeia de Caracteres***

Apesar de na Linguagem C não haver um tipo de dados *string*, pode-se trabalhar com uma cadeia de caracteres.

Na linguagem C uma *string* é um vetor de caracteres terminado com um caractere nulo. O caracter nulo é um caracter com valor inteiro igual a zero (código ASCII igual a 0). O terminador nulo também pode ser escrito usando a convenção de barra invertida do C como sendo '\0'.

Embora o assunto vetores seja discutido posteriormente, será visto aqui os fundamentos necessários para que seja possível utilizar as *strings*. Para declarar uma *string*, podemos usar o seguinte formato geral:

**SINTAXE: char <*nome\_da\_string*> [TAMANHO];**

Isto declara um vetor de caracteres (uma *string*) com número de posições igual a tamanho. Note que, como temos que reservar um caracter para ser o terminador nulo, declaramos o comprimento da *string* como sendo, no mínimo, um caractere maior que a maior *string* que se pretende armazenar. Supor que se declare uma *string* de 7 posições e se coloque a palavra João nela. Temos:



No caso acima, as duas células não usadas têm valores indeterminados. Isto acontece porque o C não inicializam variáveis, cabendo ao programador esta tarefa. Portanto, as únicas células que são inicializadas são as que contêm os caracteres 'J', 'o', 'a', 'o' e '\0' .

A função *scanf()*, não é muito adequada para pegar do teclado um seqüência de caracteres, pois um simples espaço a mesma enxerga como um terminador de *string* e a partir daí não pega mais nada do teclado. Além disso, você precisa pressionar **[enter]** depois da sua entrada para que a função termine a leitura. Para solucionar este problema, deve-se usar a função *gets()*, que será vista a seguir.

**A Função gets( )**

Caso se deseje ler uma *string* fornecida pelo usuário pode-se usar a função gets(). Características:

* Arquivo de cabeçalho a ser incluído: *stdio.h*
* Lê uma *string* de caracteres entrada pelo teclado e coloca-o no endereço apontado por seu argumento.
* Pode-se digitar caracteres até que seja pressionada a tecla **[Enter]**, quando então *gets()* retorna.
* Quando pressionada a tecla Enter é acrescentado um terminador nulo ao final da string.
* Pode-se corrigir erros de digitação usando a tecla <backspace>.

**Exemplo:**

// Objetivo: **Mostrar a função de entrada gets( )**

#include <stdio.h>

**main ( )**

**{**

char cadeia[100];

printf ("Digite uma cadeia de caracteres ( string) : ");

gets (cadeia);

printf ("\n Voce digitou %s", cadeia);

**}**

**Observação:**

* Neste programa, o tamanho máximo da *string* que você pode entrar é uma string de 99 caracteres. Se você entrar com uma *string* de comprimento maior, o programa irá aceitar, mas os resultados podem ser desastrosos. Será visto porque posteriormente.
* A função fflush(stdin), normalmente é usada quando se faz várias chamadas da função gets ( ). O objetivo é limpar o buffer, ou área de armazenamento temporário.