FACULDADE CESGRANRIO

ALGORITMO E PROGRAMAÇÃO

PROFESSORA FLÁVIA BALBINO DA COSTA
FLAVIA.BALBINO@YAHOO.COM.BR

INTRODUÇÃO À PROGRAMAÇÃO

- Um algoritmo é uma seqüência lógica de ações a serem executadas para se executar uma determinada tarefa.

- Um programa é a formalização de um algoritmo em uma determinada linguagem de programação, segundo suas regras de sintaxe e semântica, de forma a permitir que o computador possa entender a seqüência de ações.

LINGUAGEM DE PROGRAMAÇÃO:

- Uma linguagem de programação é um conjunto de símbolos (comandos, identificadores, caracteres ASCII, etc. ...) e regras de sintaxe que permitem a construção de sentenças que descrevem de forma precisa ações compreensíveis e executáveis para o computador.
- Uma linguagem de programação é uma notação formal para descrição de algoritmos que serão executados por um computador.

LINGUAGEM DE PROGRAMAÇÃO = SÍMBOLOS + REGRAS DE SINTAXE

TIPOS DE LINGUAGENS DE PROGRAMAÇÃO

- Linguagem de máquina;
- Linguagem de baixo nível; e
- Linguagem de alto nível.

LINGUAGENS DE MÁQUINA E BAIXO NÍVEL

- **Linguagem de máquina:** Única compreendida pelo computador. Específica de cada computador.

- **Linguagens de baixo nível:** São linguagens voltadas para a máquina, isto é, são escritas usando as instruções do microprocessador do computador (mnemônicos). São genericamente chamadas de linguagens *Assembly*.

LINGUAGENS DE ALTO NÍVEL

- **Linguagens de alto nível:** São linguagens voltadas para o ser humano. Em geral utilizam sintaxe estruturada tornando seu código mais legível. Necessitam de ***compiladores*** ou ***interpretadores*** para gerar instruções do microprocessador.
- **Interpretadores** fazem a interpretação de cada instrução do programa fonte executando-a dentro de um ambiente de programação, Basic e AutoLISP por exemplo.

LINGUAGENS DE ALTO NÍVEL

- **Compiladores** fazem a tradução de todas as instruções do programa fonte gerando um programa executável. Estes programas executáveis (*.exe) podem ser executados fora dos ambientes de programação, **C** e **Pascal,** por exemplo.

- As linguagens de alto nível podem se distinguir quanto a sua aplicação em **genéricas** como C, Pascal e Basic ou **específicas** como Fortran (cálculo matemático), GPSS (simulação), LISP (inteligência artificial) ou CLIPPER (banco de dados).

EDIÇÃO → **COMPILAÇÃO** → **LINK-EDIÇÃO**

ALGORITMO CÓDIGO-FONTE CÓDIGO-OBJETO PROGRAMA EXECUTÁVEL

POSSIBILIDADES DE ERROS NO PROGRAMA:

- **Erros de Compilação** : erros de digitação e de uso da sintaxe da linguagem.
- **Erros de Link-Edição** : erro no uso de bibliotecas de sub-programas necessárias ao programa principal.
- **Erros de Execução** : erro na lógica do programa (algoritmo).

LINGUAGEM C

- Foi desenvolvida no início da década de 70, inicialmente para Unix – isso permitiu que a linguagem fosse largamente utilizada no meio acadêmico, já que até meados da década de 90, tal comunidade utilizava basicamente computadores de grande porte, cujo sistema operacional era o Unix. A conseqüência deste fato é que a maioria dos algoritmos apresentados em livros ou encontrados na Internet está em C. Também podemos encontrar diversas bibliotecas de programação gratuitas na Internet escritas em C.

LINGUAGEM C

- Possui recursos de alto e baixo nível – isso faz com que a maioria dos sistemas operacionais (Windows e Linux, por exemplo), jogos e drivers, entre outros, sejam desenvolvidos em C.
- É uma linguagem que possui um conjunto de comandos e funções padronizados (o padrão ANSI) e possui compiladores para todos os sistemas operacionais – assim, uma aplicação escrita em ANSI C compilará em Unix, Linux, Windows, etc.

LINGUAGEM C

- Sua sintaxe foi utilizada como base para diversas outras linguagens desenvolvidas mais recentemente. Certamente você encontrará familiaridade entre C, com linguagens como C++, Java e JavaScript.
- A linguagem C é **genérica**, portanto, com ela podemos desenvolver qualquer tipo de aplicação.

CARACTERÍSTICAS DA LINGUAGEM

- A principal característica da linguagem C que se pode notar logo nos primeiros exemplos a serem testados, é que esta é uma linguagem “**sensitive case**”, ou melhor, “**sensível ao contexto**”.
- Na prática isto quer dizer que identificadores definidos com letras maiúsculas devem ser utilizados ao longo do código em letras maiúsculas e os definidos em letras minúsculas, utilizados em letras minúsculas.

CARACTERÍSTICAS DA LINGUAGEM

- Outra característica da linguagem, que muitas vezes assustam principalmente aqueles que já conhecem outra linguagem, é que, em alguns casos, os programas podem se tornar muito "**simbólicos**", isto é, a sintaxe da linguagem C em alguns momentos se utiliza mais de símbolos do que palavras. Por exemplo, para abrir e fechar um bloco de comandos em C utilizamos, respectivamente, { e }, enquanto que em Pascal, por exemplo, utilizamos as palavras begin e end.
- .

BOAS PRÁTICAS DE PROGRAMAÇÃO

- Uma boa prática de programação é a colocação de **comentários** no código em desenvolvimento. Comentários em C são iniciados por /* e finalizados por */. Podemos colocar em qualquer parte do código para descrever o seu objetivo.

BOAS PRÁTICAS DE PROGRAMAÇÃO

- Outra boa prática de programação, e talvez, a mais importante, e a **endentação do código**. Tal prática deve ser tornar um costume desde o primeiro exemplo a ser digitado, pois deixar para fazê-la após a conclusão do programa poderá se tornar uma tarefa extremamente difícil à medida que os seus programas vão se tornando cada vez maiores.

EXEMPLO DE COMENTÁRIOS E ENDENTAÇÃO

```
#include <stdio.h>          /* declaração da biblioteca */
#define PI  3.1415926       /* declaração de constante */
#define AreaCirc(r) PI*r*r  /* declaração de uma macro */

void main() {               /* início do programa principal */
   float raio, area;        /* declaração de variáveis */

   printf("Informe o raio do círculo: ");
   scanf("%f", &raio);
   área = AreaCirc(raio);
   printf("A área do círculo é igual a %.2f\n", raio);
} /* fim do programa principal */
```

FORMATO BÁSICO DE UM PROGRAMA EM C

- Antes de especificar tal formato, é importante citar que a linguagem C foi desenvolvida com o conceito de liberdade para programação. Portanto, a formato especificado a seguir não é algo extremamente rígido.

FORMATO BÁSICO DE UM PROGRAMA EM C

- Declaração das Bibliotecas
- Declaração das Constantes
- Declaração dos Tipos Estruturados
- Declaração das Variáveis Globais
- Declaração das Funções
- Programa Principal

DECLARAÇÃO DE BIBLIOTECAS

- Uma biblioteca é um conjunto de funções definidas em um outro arquivo (normalmente com a extensão “.h”) para utilizarmos sempre que precisarmos.

- A linguagem C possui diversas bibliotecas definidas pelo padrão ANSI, além de permitir que um programador crie suas próprias bibliotecas.

DECLARAÇÃO DE BIBLIOTECAS

- Quando desejarmos utilizar as funções definidas em uma biblioteca, devemos declará-la do seguinte modo, no início do programa:

#include <biblioteca>

ou

#include “biblioteca”

DECLARAÇÃO DE BIBLIOTECAS

- Utilizamos o primeiro caso quando é uma biblioteca instalada no diretório ou nos diretórios especificados na configuração do compilador (geralmente as bibliotecas que já são instaladas junto com o compilador ou bibliotecas instaladas por um outro programa).

- O segundo caso é utilizado quando a biblioteca foi desenvolvida pelo próprio programador e está no mesmo diretório do arquivo fonte do programa.

DECLARAÇÃO DE CONSTANTES

- Uma constante em C pode ser definida do seguinte modo:

- #define IDENTIFICADOR VALOR

DECLARAÇÃO DE VARIÁVEIS

- Podemos pensar em um programa de computador como um conjunto de comandos que agem sobre valores de algum modo obtidos.

- Tais valores ficam armazenados, quando um programa está em funcionamento, na memória principal do computador.

DECLARAÇÃO DE VARIÁVEIS

- Cada posição da memória é identificada por um endereço (um número). Uma variável é na verdade uma referência a uma posição de memória (a determinação de tal posição é papel do sistema operacional).

- Portanto, sempre que precisarmos armazenar algum valor em um programa (portanto uma posição da memória principal), usaremos uma variável.

DECLARAÇÃO DE VARIÁVEIS

- Toda variável possui um identificador (um nome – assim não precisamos nos referenciar ao endereço da memória) e um tipo.

- Na linguagem C, como em várias outras linguagens, todas as variáveis que utilizarmos devem ser antes declaradas. É na declaração que especificamos o identificador e o tipo de cada variável.

DECLARAÇÃO DE VARIÁVEIS

- A sintaxe para declaração de uma variável é a seguinte:

tipo IDENTIFICADOR;

- Se tivermos mais de uma variável do mesmo tipo, podemos declará-las na mesma linha, separadas por vírgula, como no modelo a seguir:

tipo IDENTIFICADOR_1, IDENTIFICADOR_2, ...,

DECLARAÇÃO DE VARIÁVEIS

- O identificador é definido pelo programador. Devemos apenas lembrar que todo identificador deve começar por uma letra ou pelo símbolo de "_", seguido ou não de letras, dígitos ou do símbolo de "_".

- NUNCA UTILIZAR ESPAÇO EM BRANCO NO IDENFICADOR, POIS ASSIM, PASSAREMOS A TER DOIS IDENTIFICADORES.

DECLARAÇÃO DE VARIÁVEIS

- Quanto ao tipo, a linguagem possui alguns tipos básicos. Os principais são:

Tipo	Descrição	Intervalo
int	Inteiro de 2 bytes	-32.728 a 32.727
long	Inteiro de 4 bytes	-2.147.483.647 a 2.147.483.648
float	Real com 6 dígitos de precisão (4 bytes)	3,4E-38 a 3,4E+38
double	Real com 10 dígitos de precisão (8 bytes)	1,7E-308 a 1,7E+308
char	Caracter	Caracteres com código de 0 a 255

DECLARAÇÃO DE VARIÁVEIS

Observações muito importantes:

- Não possui um tipo para guardar valores booleanos explicitamente. Na verdade, podemos usar os próprios valores numéricos como valores booleanos, pois em C, o valor 0 (zero) é considerado FALSO e qualquer outro valor diferente de zero é considerado VERDADEIRO.

DECLARAÇÃO DE VARIÁVEIS

Observações muito importantes:

- Não existe o tipo string. Isso porque, na verdade, strings são definidas explicitamente como vetores de caracteres. Veremos mais detalhes sobre strings mais tarde, mas já adiantando, a forma de se declarar uma string é:

char IDENTIFICADOR[TAMANHO]

Onde TAMANHO é um valor inteiro que corresponde a quantidade máxima de caracteres que o vetor IDENTIFICADOR poderá armazenar.

DECLARAÇÃO DE VARIÁVEIS

Observações muito importantes:

- Devemos lembrar que no momento que uma variável é declarada não podemos determinar qual o seu valor inicial (normalmente dizemos que o valor inicial de uma variável é “lixo”). No entanto, em C, podemos atribuir um valor para a variável no momento em que esta é declarada, como nos exemplos a seguir:
- int i = 0;
- float x = 1.0;

PROGRAMA PRINCIPAL

- O programa principal é o bloco de comandos que determina ao computador por onde deve ser iniciada a execução de um programa.

- Portanto, a execução de um programa começa pelo primeiro comando do programa principal, que, evidentemente, deve ser único.

PROGRAMA PRINCIPAL

- O programa principal em C é tratado com sendo uma função. Para diferenciar das funções que o programador poderá a vir definir em seu programa, o nome do programa principal deve ser **main**.

- O tipo de retorno pode ser inteiro (int) ou, simplesmente, não retornar valor algum (void).

PROGRAMA PRINCIPAL

- Nos exemplos que utilizaremos a função main possuirá o seguinte formato:

```
int main () {
        ...
}
```

UM EXEMPLO DE UM PROGRAMA EM C

```
#include <stdio.h>          /* declaração da biblioteca */
#define PI  3.1415926       /* declaração de constante */
#define AreaCirc(r) PI*r*r  /* declaração de uma macro */

int main() {                /* início do programa principal */
   float raio, area;        /* declaração de variáveis locais*/

   printf("Informe o raio do círculo: ");
   scanf("%f", &raio);
   area = AreaCirc(raio);
   printf("A área do círculo é igual a %.2f\n", raio);
} /* fim do programa principal */
```

OPERADORES DO C

Operador de Atribuição

- Em C, a atribuição é definida como um operador e não como um comando. Seu objetivo é o mesmo de qualquer outra linguagem: atribuir um valor ou o resultado de uma expressão à variável do lado esquerdo. A sintaxe da operação de expressão é a seguinte:

variável = valor ou expressão

OPERADORES DO C

Operador de Atribuição

- A conseqüência de ser uma operação é a de possuir um resultado que é o valor que está sendo atribuído. Isso permite, por exemplo, que diversas variáveis recebam um certo valor de uma só vez, como no exemplo a seguir:

```
x = y = z = 0;
```

OPERADORES DO C

Operadores Aritméticos

- Operadores aritméticos são utilizados para realização de cálculos. A linguagem possui **operadores binários** (necessitam de dois operandos) e **unários** (necessitam apenas de um operando). A seguir são listados cada um deles, juntamente com exemplos.

OPERADORES DO C

Operadores Aritméticos

- Operador binário de soma: +

x + y

- Operador binário de subtração: –

x - y

- Operador binário de multiplicação: ✶

x ✶ y

OPERADORES DO C

Operadores Aritméticos

- Operador binário de divisão: /

x / y

- Operador binário de resto da divisão: %

x % y

- Operador unário de mudança de sinal: –

-x

OPERADORES DO C

Operadores Aritméticos

- Operador unário de incremento: ++

x++

- Operador unário de decremento: - -

x - -

OPERADORES DO C

Operadores Aritméticos

- Operadores binários aritméticos de atribuição

```
x += y   /* o mesmo que x = x + y */
x -= y   /* o mesmo que x = x - y */
x *= y   /* o mesmo que x = x * y */
x /= y   /* o mesmo que x = x / y */
x %= y   /* o mesmo que x = x % y */
```

OPERADORES DO C

Operadores Relacionais

- São operadores utilizados para comparação entre variáveis e entre variáveis e valores ou expressões, retornando, portanto, resultados de VERDADEIRO ou FALSO.

OPERADORES DO C

Operadores Relacionais

- Operador de igualdade: ==

x == y

- Operador de diferença: !=

x != y

- Operador de maior: >

x > y

OPERADORES DO C

Operadores Relacionais

- Operador de maior ou igual: >=

x >= y

- Operador de menor: <

x < y

- Operador de menor ou igual: <=

x <= y

OPERADORES DO C

Operadores Lógicos

São operadores utilizados na combinação de testes.

- Operador E: &&

x == 0 && y == 0

- Operador OU: ||

x == 0 || y == 0

- Operador de negação: !

!(x == 0 && y == 0)

OPERADORES DO C

Operadores Lógicos

- E / AND - Uma expressão AND (E) é verdadeira se todas as condições forem verdadeiras.
- OR/OU - Uma expressão OR (OU) é verdadeira se pelo menos uma condição for verdadeira.
- NOT - Um expressão NOT (NÃO) inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa.

OPERADORES DO C

- A tabela abaixo mostra todos os valores possíveis criados pelos três operadores lógicos (AND, OR e NOT):

1º Valor	Operador	1º Valor	Resultado
T	AND	T	T
T	AND	F	F
F	AND	T	F
F	AND	F	F
T	OR	T	T
T	OR	F	T
F	OR	T	T
F	OR	F	F
T	NOT		F
F	NOT		T

ATRIBUIÇÃO *X* COMPARAÇÃO DE IGUALDADE

- Alguns programadores ao aprender C, por já conhecerem outra(s) linguagem(ns), comumente cometem um pequeno erro: trocar o operador de comparação de igualdade (==) pelo operador de atribuição (=) ao realizar um teste.

- Vejamos a conseqüência em um pequeno exemplo.

ATRIBUIÇÃO *X* COMPARAÇÃO DE IGUALDADE

```
#include <stdio.h>

void main() {
   ...
   if (x = 0)
   printf("O valor de x é zero");
   else
   printf("O valor de x é diferente de zero");
}
```

ATRIBUIÇÃO *X* COMPARAÇÃO DE IGUALDADE

- O que seria de imaginar que o valor de x sendo igual a zero o primeiro printf seria executado, enquanto se o valor de x for diferente de zero, o segundo printf seria executado.
- Mas não é isso que acontece: este trecho sempre executará o segundo printf. Isso acontece porque estamos realizando uma operação de atribuição (o valor de x passará, portanto, a ser igual a zero!).

ATRIBUIÇÃO *X* COMPARAÇÃO DE IGUALDADE

- O resultado é o próprio valor atribuído, portanto, zero, e zero em C é considerado falso. Portanto, seja qual for o valor de x antes da execução do comando if, seu valor passará a ser igual a zero e o segundo printf sempre será executado.

ATRIBUIÇÃO *X* COMPARAÇÃO DE IGUALDADE

- Para funcionar como imaginamos no início deste parágrafo, o programa deveria ser escrito do seguinte modo:

```
#include <stdio.h>

void main() {
   ...
   if (x == 0)
   printf("O valor de x é zero");
   else
   printf("O valor de x é diferente de zero");
}
```

COMANDOS DE ENTRADA E SAÍDA

- Para utilização da maioria destes, devemos declarar, no início do código fonte, a biblioteca **stdio.h**.

Comandos de Entrada

- Devemos lembrar, antes de tudo, que ao utilizarmos comandos de entrada em um programa, alguns dados devem ser informados pelo usuário, via teclado.

COMANDOS DE ENTRADA

SCANF(string_formato, lista_endereços_variáveis);

- O scanf é um comando para ler qualquer tipo de informação, mas que devido a algumas particularidades, utilizaremos basicamente para leitura de valores numéricos.

COMANDOS DE ENTRADA

SCANF(string_formato, lista_endereços_variáveis);

- O primeiro parâmetro deste comando, STRING_FORMATO, serve para especificar o formato da, e através da, utilização de códigos especiais de formatação, como os valores digitados pelo usuário deverão ser interpretados.

COMANDOS DE ENTRADA

SCANF(string_formato, lista_endereços_variáveis);

- Os códigos especiais de formatação mais utilizados são os seguintes:

Código de Formatação	Tipo da Interpretação
%d	int
%ld	long
%f	float
%lf	double
%c	char
%s	string (vetor de caracteres)

COMANDOS DE ENTRADA

SCANF(string_formato, lista_endereços_variáveis);

- OBS: Lembrando que string não é um tipo básico da linguagem C. Porém, já que muitas vezes no programa temos que ler um texto, a possibilidade de se ler um vetor de caracteres foi incorporada ao comando.

COMANDOS DE ENTRADA

SCANF(string_formato, lista_endereços_variáveis);

- O segundo parâmetro, LISTA_ENDEREÇOS_VARIÁVEIS, especifica uma lista de endereços de uma ou mais variáveis, de acordo com a quantidade de valores que serão lidos de uma só vez, ao se executar o comando.
- Para especificar o endereço de uma variável em C, esta deve ser precedida pelo símbolo & (exceto para vetores de caracteres - strings).

COMANDOS DE ENTRADA

SCANF(string_formato, lista_endereços_variáveis);

- Veja exemplos da utilização deste comando:

```
int i, dia, mês, ano;
float x;
char op, nome[80];

        scanf(“%d”, &i);
        scanf(“%f”, &x);
        scanf(“%d%f”, &i, &x);
        scanf(“%d/%d/%d”, &dia, &mês, &ano);
        scanf(“%c”, &op);
        scanf(“%s”, nome);
```

COMANDOS DE ENTRADA

SCANF(string_formato, lista_endereços_variáveis);

- A **leitura de strings** com o comando **scanf** possui uma pequena particularidade: quando o texto digitado pelo usuário contém mais de uma palavra, somente a primeira é lida. Portanto, como na maioria dos casos queremos ler um texto até que seja pressionada a tecla ENTER, independente de quantas palavras tal texto possui, utilizaremos outro comando, específico para leitura de strings.

COMANDOS DE ENTRADA

GETS(lista_variáveis)

- Este é o comando da linguagem C específico para leitura de strings, lendo o texto digitado pelo usuário até que este pressione a tecla ENTER. Veja o exemplo de como utilizá-lo:

```
gets(nome);
```

COMANDOS DE ENTRADA

```
getchar();
getch();
```

- Antes de explicar para que servem, getchar() e getch() na verdade são funções e não (simplesmente) comandos. Isso significa que os utilizaremos atribuindo seus resultados a variáveis. Tais funções servem para leitura de caracteres.

COMANDOS DE ENTRADA

getchar();
getch();

- Devemos utilizá-los do seguinte modo, como mostra o exemplo a seguir:

```
char op;

op = getchar();
op = getch();
```

COMANDOS DE ENTRADA

getchar();
getch();

- A diferença das duas funções é a seguinte: na primeira, o caracter que o usuário digitar aparecerá na tela e o processamento da função só é realizado quando o usuário pressiona a tecla ENTER. Já a segunda, o caracter que o usuário digitar não aparecerá na tela e não é preciso que o usuário pressione a tecla ENTER para a função terminar o seu processamento.

COMANDOS DE ENTRADA

getchar();

getch();

- Obs.: Na verdade, a função getch() pertence à biblioteca conio.h. No entanto, alguns compiladores à incorporaram na biblioteca stdio.h. Portanto, mesmo colocando esta última biblioteca, se o compilador não reconhecer a função getch(), declare também a biblioteca conio.h.

COMANDOS DE SAÍDA

- Comandos a serem utilizados pelo usuário quando se deseja que o programa informe algum tipo de informação.

COMANDOS DE SAÍDA

printf (STRING_TEXTO, LISTA_VARIÁVEIS);

- A sintaxe deste comando é bem similar a do comando de leitura scanf. O primeiro parâmetro, STRING_TEXTO, é onde definimos, colocando entre aspas, a mensagem que deve ser informada na tela do computador.

COMANDOS DE SAÍDA

printf (STRING_TEXTO, LISTA_VARIÁVEIS);

- Se a mensagem for composta por valores contidos em variáveis, devemos colocar no local da mensagem onde o valor deverá aparecer o código especial de formatação (os mesmos utilizados no scanf), de acordo com o tipo da variável.

COMANDOS DE SAÍDA

printf (STRING_TEXTO, LISTA_VARIÁVEIS);

- As variáveis que terão valores informados na mensagem, são especificadas na LISTA_VARIÁVEIS, separadas por vírgulas, se mais de uma. Neste caso, ao contrário do comando scanf, a variável não deve ser precedida do símbolo &, utilizando-se apenas o seu identificador.

COMANDOS DE SAÍDA

printf (STRING_TEXTO, LISTA_VARIÁVEIS);

- Além de variáveis, também podem ser especificados resultados de expressões. Veja exemplos a seguir:

COMANDOS DE SAÍDA - PRINTF

Exemplos:

```
int i, dia, mês, ano;
float n1, n2;
char nome[80];

printf(“Informe a sua idade: ”);
printf(“%d”, i);
printf(“O aluno %s nasceu no dia %d/%d/%d\n”, nome, dia, mês, ano);
printf(“O aluno %s de notas %f e %f ficou com media %f\n”,
        nome, n1, n2, (n1+n2)/2);
```

COMANDOS DE SAÍDA - PRINTF

- Alguns caracteres especiais podem ser utilizados para compor a STRING_TEXTO, para formatar a mensagem. O mais utilizado é o ‘\n’ para indicar uma quebra de linha.

COMANDOS DE SAÍDA - PRINTF

Obs.: Formatação Numérica de Saída

- Ao se utilizar, por exemplo, %f para escrever o valor de uma variável float, tal valor será escrito com 6 casas decimais. Tal quantidade pode ser controlada.

COMANDOS DE SAÍDA - PRINTF

Obs.: Formatação Numérica de Saída

- Podemos utilizar os códigos especiais de formatação para números do seguinte modo:

%M.Nf

%Md

- onde M corresponde ao total de casas que o número deverá ocupar e N corresponde ao número de casas decimais após o ponto. Se o valor de M for especificado e o valor da variável ocupar menos casas que o valor de M, ao ser escrito na tela, o valor da variável será precedida de espaços.

COMANDOS DE SAÍDA - PRINTF

Obs.: Formatação Numérica de Saída

Exemplo:

```
float x = 2.5;
int   i = 10;

    printf("%f\n", x);
    printf("%.3f\n", x);
    printf("%7.3f\n", x);
    printf("%d\n", i);
    printf("%5d\n", i);
```

A saída será:

```
2.500000
2.500
  2.500
10
   10
```

COMANDOS DE SAÍDA

puts(STRING);

- Comando para escrever na tela ou uma mensagem ou o valor de uma variável string. Exemplos:

```
puts(“Informe uma mensagem: ”);
puts(nome);
```

COMANDOS DE SAÍDA

putchar(CARACTER);

- Comando para escrever na tela ou um valor caracter ou o valor de uma variável caracter. Exemplos:

  putchar(‘A’);

  putchar(op);

EXERCÍCIOS:

1) Faça um programa que leia a string “Faculdade Cesgranrio” utilizando a função scanf() e a imprima na tela.
2) Faça um programa que leia a string Faculdade Cesgranrio” utilizando a função gets() e a imprima na tela.
3) Faça um programa que leia um inteiro, um caracter e um float, e depois os imprima na ordem contrária à lida.

EXERCÍCIOS:

4) Faça um programa que leia um valor de conta de restaurante, representando o gasto realizado pelo cliente e imprima o valor total a ser pago, considerando que o restaurante cobra 10% para o garçom.
5) Faça um programa que leia o valor de um horário e informe quantos minutos se passaram deste o início do dia.
6) Faça um programa que leia o nome e duas notas de