

Informática básica – Parobé

Introdução a Programação:

1. Arquivos de cabeçalho ou Bibliotecas:

São arquivos .h com códigos úteis para implementar algumas funções.

Exemplos:

`<stdio.h>` biblioteca para entradas e saídas (printf e scanf);

`<math.h>` Funções matemáticas;

`<stdlib.h>` Funções do sistema;

Dentre outras (google it).

Sintaxe:

São incluídas antes do início do programa principal

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

```
int main ()
{

}
```

O `#include` significa incluir no programa as funções encontradas nesta biblioteca.

2. Programa principal:

É o programa em si, o corpo do programa deve ser escrito entre `{ }` (chaves) para indicar seu início e fim. É nele que as variáveis locais são criadas e possivelmente alteradas.

Sintaxe:

```
int main()
{
    int matricula;

    printf("Digite seu numero de matricula");
    scanf("%d", &matricula);

    printf("Seu numero de matricula e: %d");
}
```

➤ Programa cria variável (numero inteiro) de nome 'matricula', mostra na tela a mensagem 'digite seu numero de matricula' e em seguida lê a informação digitada pelo usuário no teclado e armazena na variável 'matricula'. Após, mostra na tela o valor armazenado na variável 'matricula'.

Estrutura do programa até agora:

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

```
int main ()
{

}
```

3. Variáveis:

As variáveis são espaços de memória reservados pelo sistema operacional, que serão usadas no programa.

Podem ser de 5 tipos, porém vamos ver só 4 agora:

INT – numero inteiro; `%d`

FLOAT – numero decimal (4bytes); `%f`

DOUBLE – numero decimal de alta precisão (8bytes); `%lf`

CHAR – caractere. `%c`

Exemplos:

```
int idade; idades são números naturais inteiros;
float peso; o peso pode ser um numero decimal;
double peso; caso o peso tenha que ser muito preciso;
char letra; pode armazenar a letra inicial do seu nome.
```

Sintaxe:

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

```
int main ()
{
    int idade;
    float altura;
    double peso;
    char letra;

    printf("Digite a sua idade");
    scanf("%d", &idade);
    printf("Digite sua altura");
```

```
scanf("%f", &altura);
printf("Digite o seu peso");
printf("Digite a primeira letra do seu nome");
scanf("%c", &letra);
```

```
}
```

4. Entradas / Saídas:

➤ Printf

É uma função da biblioteca `<stdio.h>`, portanto, para utilizar esta função é necessária a inclusão desta biblioteca no início do programa.

Esta função ‘imprime’ (print) na tela uma mensagem pré-definida pelo programador.

Exemplo:

```
printf("sua mensagem aqui");
```

➤ Scanf

É também uma função da biblioteca `<stdio.h>`. Não é preciso incluir duas vezes a mesma biblioteca para executar mais de uma de suas funções.

Esta função ‘lê’ as informações digitadas no teclado pelo usuário e armazena em uma variável declarada. Deve-se prestar atenção aos tipos de variáveis declaradas.

Exemplos:

```
scanf("%d", &idade);
scanf("%f", &altura);
scanf("%lf", &peso);
scanf("%c", &letra);
```

5. Operadores e operações:

➤ Operadores matemáticos básicos:

Operador	Operação	Aplicação
=	Atribuição	$z = 2;$
+	Soma	$z = x + y;$
-	Subtração	$z = x - y;$
*	Multiplicação	$z = x * y;$
/	Divisão	$z = x / y;$
%	Resto Div Inteira	$z = x \% y;$

➤ Operadores de comparação:

Operador	Operação	Aplicação
==	Igual a	$b == b$
!=	Diferente de	$b != a$
>	Maior que	$b > a$
<	Menor que	$a < b$
>=	Maior ou igual a	$b >= a$
<=	Menor ou igual a	$a <= b$

6. Motivação:

Porque usar C?

A linguagem é uma das mais simples e poderosas, em eletrônica, a maioria dos microcontroladores e microprocessadores utilizam esta linguagem. Ela é executada de forma sequencial de cima para baixo, desde que não exista nenhum laço de repetição, ou instrução para que execute outras partes do código.

O código gerado é conhecido como código fonte.

Estrutura do programa até agora

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

```
int main ()
{
```

```
    int a, x, y;
    float b;
    double z;
    char letra;
```

```
    printf("Digite a letra inicial do seu nome");
    scanf("%c", &letra);
    printf("digite um valor inteiro para a");
    scanf("%d", &a);
    printf("digite um valor inteiro para x");
    scanf("%d", &x);
    printf("digite um valor inteiro para y");
    scanf("%d", &y);
    printf("digite um valor decimal para b");
    scanf("%f", &b);
    printf("digite um valor decimal para z");
    scanf("%lf", &z);
    printf("%c%d%d%d%f%lf", letra, a, x, y, b, z);
    a = b + a;
    x = y - x;
    letra = 'a';
    printf("a = %f, x = %d, letra = %c", a, x, letra);
```

```
}
```

7. Identação:

A identação é importante para a interpretação do programa posteriormente, lembrando que a linguagem c é repleta de sinais e blocos que devem ser bem claros quanto a seu inicio, comandos e fim.

É exatamente este o conceito da identação, utilizando espaços podemos indicar mais facilmente o inicio, comandos e fim de cada bloco. O bloco é uma sequencia de comandos que podem ser de entradas, saídas, operações, declaração de variáveis, laços, dentre outros.

Exemplo:

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

int main()
{
    int var1, var2, soma;

    printf("Atribua um valor para a variável 1:");
    scanf("%d", &var1);
    printf("Atribua um valor para a variável 2:");
    scanf("%d", &var2);

    soma = var1 + var2;

    printf("Soma = %d", soma);
}
```

Ainda podemos deixar o código mais 'bonito', ou legível, para podermos entender melhor o que nosso programa faz.

Utilizamos então os Comentários.

Eles não são compilados com o programa, ou seja, não interagem com o código, servem apenas para explicar o que parte de um código, ou ele todo, faz. Podem ser feitos de duas formas:

➤ //

O // é um comentário de linha, após o ;.

Exemplo:

```
int a; // cria variável a
a = 2; // atribui o valor 2 à variável a
```

➤ /* */

Este tipo é utilizado para mais de uma linha de comentários, ou um bloco de comentários.

Exemplo:

```
printf("Hello World");
/*este é um programa clássico em c, conhecido
também como o mundo, geralmente feito como primeiro
programa em c*/
```

Desta forma, e com a identação, nosso programa fica muito mais fácil de entender.

Estrutura do programa até agora

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

int main()
{
    /*nome: IC Salgado
    Tec. Eletronica
    Grad. Engenharia de Computação
    local: Sec. Instituto de Química UFRGS
    data: 08/05/2013*/

    //variaveis
    int var1, var2, var3,

    //entradas
    printf("digite o valor da variável 1:");
    scanf("%d", &var1);

    //operações
    var2 = 3;
    var3 = var1 + var2;

    //saídas
    printf("o valor da variável 2 eh: %d", var3);
}
```

8. Vetores

Um problema que podemos observar é que nas variáveis do tipo 'char', podemos armazenar apenas uma letra! E se quisermos informar um nome?... Outra situação problemática é, e se quisermos armazenar as notas de uma turma com 30 alunos? Podemos criar 30 variáveis para cada aluno, mas isso não é uma boa pratica de programação!

Para estes e outros problemas usamos vetores.

O vetor consiste em armazenar mais de um valor em uma mesma variável.

Exemplo:

```
int var1 [3];
```

Neste exemplo criamos a variável 'var1' como um vetor de 3 posições ou seja:

```
var1 = 

|   |
|---|
| 0 |
|---|



|   |
|---|
| 1 |
|---|



|   |
|---|
| 2 |
|---|


```

Assim temos o vetor 'var1', com 3 posições {0, 1, 2}.

Sintaxe:

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

```
int main()
{
    int var1[3];
    var1[0] = 3;
    var1[1] = 2;
    var1[2] = 1;
```

```
}
```

Neste caso var1 =

3

2

1

9. Strings:

Strings são vetores de 'char', ou seja, podemos criar um vetor com uma variável do tipo 'char' para solucionar o problema de armazenamento de nomes por exemplo. Funciona da mesma forma de um vetor de números inteiros, porém, os números são substituídos por caracteres.

Exemplo:

```
char nome[10];
```

Neste exemplo criamos a variável tipo 'char' nome como um vetor de 10 posições.

Assim temos o vetor string 'nome', com 10 posições {0,1,2,3,4,5,6,7,8,9,}

Sintaxe:

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

```
int main()
{
    char nome[10];
    nome = "usuário";
}
```

Neste caso o vetor string 'nome' recebe a sequência de caracteres 'usuario' sendo assim:

```
nome = [u] [s] [u] [a] [r] [i] [o] [ ] [ ] [ ]
```

A string é sempre finalizada por \0, que indica o fim do vetor string e para atribuir um nome, este deve sempre estar entre “ ”.

10. Manipulação de strings:

Para a manipulação de strings é necessária a inclusão da biblioteca específica para isto, <string.h>

As principais funções dessa biblioteca são:

➤ Strlen:

strlen retorna o tamanho, em caracteres, de uma string dada.

Exemplo:

```
tam_string = strlen(nome);
```

Deve ser criada uma variável inteira para armazenar o tamanho da string (tam_string).

➤ Strcpy:

strcpy copia o conteúdo de uma string para outra.

Exemplo:

```
strcpy (destino, origem);
```

A variável de destino deve sempre ser um vetor maior que a de origem, para poder receber a sequência de caracteres da mesma.

➤ Strcmp:

strcmp serve para comparar o conteúdo de duas strings.

Exemplo:

```
strcmp (nome1, nome2);
```

Esta função retorna um valor inteiro:

Menor que zero se 'nome1' for menor que 'nome2'

Igual a zero se 'nome1' for igual a 'nome2'

Maior que zero se 'nome1' for maior que 'nome2'

Existem várias outras funções da biblioteca <string.h> que podem ser encontradas na internet (google it).

11. Leitura e Armazenamento de strings:

Para ler uma informação do teclado vimos que podemos utilizar o comando `'scanf'` da biblioteca `<stdio.h>`. Para strings podemos também utilizar esta função, porém encontraremos alguns problemas.

Exemplo:

```
scanf("%s", nome);
```

Neste exemplo é lido do teclado a sequência de caracteres digitadas pelo usuário e armazenada na variável tipo `'char'` `'nome'`. Observamos que, para armazenar uma string utilizamos `%s`, onde antes se utilizava o `%c`, isto por que o `%c` armazena somente 1 caractere. Outra observação que podemos fazer é que não é obrigatório o uso do `&` para direcionar os dados digitados para a variável `'nome'`

➤ Primeiro problema (scanf):

O problema do `scanf` para ler strings é que ele captura somente até encontrar o primeiro espaço em branco, ou seja, se for digitado `'Marco Antonio'`, a variável que receber esta string será composta apenas de `'Marco'`.

Para solucionar este problema podemos utilizar a função `'gets'` da biblioteca `<stdio.h>`

Exemplo:

```
gets(nome);
```

Neste exemplo o `'gets'` funciona praticamente da mesma forma que o `'scanf'`, respeitando sua sintaxe.

➤ Segundo problema (gets):

Na função `'gets'` o principal problema é que ela não delimita a string que será armazenada, sendo assim, o usuário pode digitar uma string com uma quantidade de caracteres maior que a do vetor declarado, ocasionando

uma invasão de memória, que em casos mais extremos pode acessar dados restritos da memória, ou até mesmo apaga-los.

Para solucionar este problema utilizaremos então a função `'fgets'`, também da biblioteca `<stdio.h>`.

Exemplo:

```
fgets(string, tamanho, stdin);
```

Onde: `string` representa o nome do vetor e `tamanho` representa a quantidade máxima de caracteres.

Estrutura do programa até agora

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

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

```
#include <string.h>
```

```
int main()
```

```
{
```

```
    char nome [20];
```

```
    int tamanho; //quantidade de caracteres do vetor nome
```

```
    printf("Digite seu nome completo:");
```

```
    fgets(nome, 20, stdin);
```

```
    tamanho = strlen(nome); //retorna quant. Caracteres
```

```
    printf("Seu nome e: %s", nome);
```

```
    printf("Seu nome possui %d caracteres", tamanho);
```

```
}
```

Este programa recebe e armazena o nome do usuário e retorna o tamanho da string, incluindo o ENTER para continuação do programa.

12. Matrizes:

Matrizes são possíveis em C somente se usarmos vetores.

Uma matriz de duas dimensões é composta pelo número de linhas e número de colunas. Em C o uso desta matriz é simples:

Exemplo:

```
int matriz_2d [n_linhas][n_colunas];
```

Neste exemplo, podemos substituir `n_linhas` e `n_colunas` por números inteiros, assim teremos uma matriz com o número de linhas e número de colunas pré-definidos.

Muita atenção na declaração dos índices, em matrizes do tipo tabela, sempre se indica primeiro o número de linhas e depois o número de colunas.

Exemplo:

```
int matriz_2d [3][4];
```

Neste exemplo criamos uma matriz tabela de duas dimensões, com 3 linhas e 4 colunas, ou seja:

```
matriz_2d [3][4] =
```


Para inserir um valor qualquer a um dos campos da matriz, devemos indicar os índices (linha/coluna) que desejamos utilizar.

Exemplo:

```
matriz_2d [1][2] = 4;
```

Neste exemplo colocamos o número 4 no índice linha 1 e coluna 2 então:

```
matriz_2d [3][4] = 0
```

1		4		
2				
	0	1	2	3

13. Comando condicional IF/ELSE:

Em português estruturado este comando é o Se(então faça)..Senão(faça). Português Estruturado é uma forma de escrever um programa em português com a estrutura de um programa em C, que é todo em inglês, ele facilita o entendimento do programa pelo programador ou usuário.

Uma aplicação do IF/ELSE é um menu de opções, por exemplo, se o usuário digitar o número 1, o programa executa uma sequência de comando e se digitar qualquer outro número o programa executa outros comandos.

Exemplo:

```
scanf("%d", &num);
if (num == 1)
{
    printf("O numero digitado foi 1");
}else
{
    printf("O numero digitado foi %d", num);
}
```


Os comandos atribuídos ao IF devem estar dentro de { } (chaves), apenas se for um comando as chaves são dispensadas, porém, para uma leitura mais clara do código é recomendado que sempre se use as chaves. (lembrando também de utilizar a indentação).

14. IFs Aninhados:

Para reproduzir uma condição dentro da outra, podemos colocar um IF dentro do outro. Isto se chama aninhar ifs. O uso do ELSE não é obrigatório, mas para o entendimento pode ser utilizado.

Exemplo:

```
scanf("%d", &num);
if(num == 1)
{
    printf("O numero digitado foi 1");
    scanf("%d", &dia);
    if(dia == 15)
    {
        printf("O dia digitado foi 15");
    }else
    {
        printf("O dia digitado foi %d", dia);
    }
}
else
{
    printf("O numero digitado foi %d", num);
}
```

A indentação é de grande importância no uso do IF para sabermos onde inicia e onde termina o bloco de cada condição.

15. IFs Encadeados:

Para criarmos um menu de opções com mais de duas opções podemos utilizar IFs encadeados, isso significa, colocar um IF (uma condição) depois do outro.

Exemplo:

```
scanf("%d", &num);
if(num == 1)
{
    printf("O numero digitado foi 1");
}
if(num == 2)
{
    printf("O numero digitado foi 2");
}
if(num == 3)
{
    printf("O numero digitado foi 3");
}
if(num == 4)
{
    printf("O numero digitado foi 4");
}
else
{
    printf("Nenhuma das condições foi satisfeita");
}
```

16. Comando condicional SWITCH/CASE:

O SWITCH/CASE funciona como uma sequencia de IFs encadeados, cabe ao programador saber quando utilizar este comando pois alguns detalhes devem ser preservados.

Este comando é útil para a elaboração de menus com mais de uma opção, sendo assim pode substituir o encadeamento de IFs.

Exemplo:

```
scanf("%d", &op);
switch (op)
{
    case 1:
        printf("operacao 1 = adicao");
        x = a + b;
        break;
    case 2:
        printf("operacao 2 = subtracao");
        x = a - b;
        break;
    case 3:
        printf("operacao 3 = multiplicacao");
        x = a * b;
        break;
    case 4:
        printf("operacao 4 = divisao");
        x = a / b;
        break;
    default:
        printf("entrada invalida");
}
}
```

O comando `'break'` serve para que, quando um caso for satisfeito o programa sai do `'switch'`. Alguns detalhes devem ser observados, o `'switch'` é uma função executada dentro um bloco com condicionais dentro (`case`), no final de cada expressão condicional utilizamos dois pontos (`case 1:`) e no final da dos comando condicionais usamos o `break`.

17. Laço de repetição WHILE:

Traduzindo o comando, podemos dizer que um determinado bloco de comandos é executado ENQUANTO uma determinada condição for atendida.

Exemplo:

```
x = 5;
while( x > 0)
{
    printf("x = %d", x);
}
```

Neste programa temos um problema, a condição para execução do bloco sempre será verdadeira, ou seja, é um laço infinito. Em alguns casos o laço infinito será utilizado, mas por enquanto queremos um programa que tenha fim. Então devemos sempre colocar um comando que altere a variável ou que faça a condição ser falsa ao menos uma vez para terminar o laço.

Exemplo:

```
x = 5;
while( x > 0)
{
    printf("x = %d", x);
    x = x - 1;
}
```

Neste caso, a variável `x` será alterada cada vez que o programa entrar no laço, assim, quando ela for menor que 0 o programa sai do laço.

18. Laço de repetição DO-WHILE:

No comando `while` dependendo da variável o programa pode nunca entrar no laço, ou seja, a condição é falsa logo na primeira verificação. Caso seja preciso que o programa entre ao menos uma vez no laço e execute um bloco de comandos utilizaremos o `do-while`, que traduzindo seria 'faça enquanto a condição for verdadeira'. Também neste laço temos a possibilidade de sempre ser satisfeita a condição, ou seja, um laço infinito, então, será utilizado o mesmo artifício que no `while`.

Exemplo:

```
x = 5;

do
{
    printf("x = %d", x);
    x = x - 1;
}while( x > 0 );
```

Observamos que, primeiramente, o `while` vai no fechamento do bloco iniciado no `do`, outro detalhe é que neste caso o `while` precisa do ; (ponto e virgula) no final, diferentemente do comando `while` do tópico 17

19. Observações quanto ao WHILE / DO-WHILE:

- Precisamos inicializar a variável antes do início do bloco, ou seja, atribuir um valor à variável;
- Dentro do bloco devemos incrementar ou decrementar a variável para que a condição seja falsa em algum momento;

- No laço `WHILE` a condição vem no início do bloco, que pode nunca ser executado caso a condição seja sempre falsa;
- No laço `DO-WHILE` a condição vai no final do bloco então, o bloco é executado ao menos uma vez;
- Ambos os laços são utilizados quando não sabemos precisar quantas vezes o bloco de comandos deverá ser executado.

20. Operadores de Incremento e Decremento:

Operador	Função	O que faz	Sintaxe
<code>++</code>	Incremento	Soma 1	<code>var = var++</code> ou <code>var = ++var</code>
<code>--</code>	Decremento	Subtrai 1	<code>var = var--</code> ou <code>var = --var</code>

21. Laço de repetição FOR:

Este laço é mais simples que os outros dois, porém cabe ao programador saber quando utiliza-lo, isto é, existem situações que é de melhor prática a utilização de um laço `while` ou `do-while`, e em outras que o `for` é o mais recomendado, mas nada impede que se utilize um ou outro laço.

Em português estruturado o laço `for` é representado pelo comando `para`, ou seja, para uma sequência de condições, execute um bloco de comandos.

Exemplo:

```
int x;
for( x = 0; x < 5; x = x++ )
{
    printf("x = %d", x);
}
```

Neste programa é feita a verificação se a variável é menor que 5, se for, o bloco de comando é executado e a variável incrementada.

22. Observações quanto ao comando FOR:

- Não é preciso inicializar a variável antes do início do bloco e nem dentro dele, ela é inicializada nas condições do comando: `(for (x = 0; ;))`;
- A condição é colocada logo após a inicialização da variável: `(for (x = 0; x < 5;))`;
- Para que o laço tenha fim, a variável é incrementada ou decrementada logo após a condição: `(for (x = 0; x < 5; x = x++))`;
- O for é recomendado para situações em que sabemos o número de vezes que o bloco deve ser utilizado, e comumente aplicado a verificação de índices e contadores.

23. Observações quanto aos laços de repetição:

- Todos eles podem ser encadeados e aninhados;
- Pode-se utilizar um comando dentro do outro, respeitando-se a sintaxe de cada um;
- Cada um dos 3 são diferentes entre si, então devemos prestar atenção na correta aplicação dos mesmos.

24. Observações quanto aos comandos condicionais:

- O comando switch não é um comando muito recomendado, porém, pode facilitar a elaboração de menus;
- Ao utilizar if aninhado ou encadeado, deve-se ter atenção com o início e fim de cada bloco, caso contrário o programa pode não executar da forma desejada.

25. FIM

Na verdade não é o fim, ainda existem várias outras funções e comandos em C, mas acredito que nesta disciplina não serão abordadas. Espero que ajude!

Obrigado.

Iuri de Carvalho Salgado