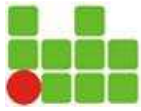


LP1

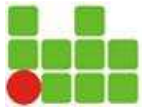
Prof. Luciano Bernardes de Paula

Estrutura básica de um programa em C



diretivas

```
tipo nomeFunc(declaração de parâmetros)
{
    declaração de variáveis;
    instrução_1;
    instrução_2;
    ...
    instrução_n;
    retorno valor;
}
```



Menor programa C

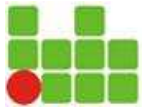
```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    return 0;
```

```
}
```



Menor programa C

```
#include <stdio.h>

int main()
{
    printf("Ola, mundo!");

    return 0;
}
```



Função de escrita na tela (saída)

```
printf(“frase a ser escrita na tela...”);
```

```
printf(“bla bla bla”);
```



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

```
int main()
```

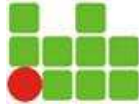
```
{
```

```
    printf("Ola, \nmundo!\n");
```

```
    return 0;
```

```
}
```

Alguns caracteres especiais para o



printf

\n – nova linha

\t - tabulação

\f – salto de página

\a – sinal sonoro

\r – retorna o cursor no início da linha

\\ - barra invertida

\0 – caracter nulo

\' – aspas simples

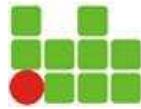
\” – aspas duplas



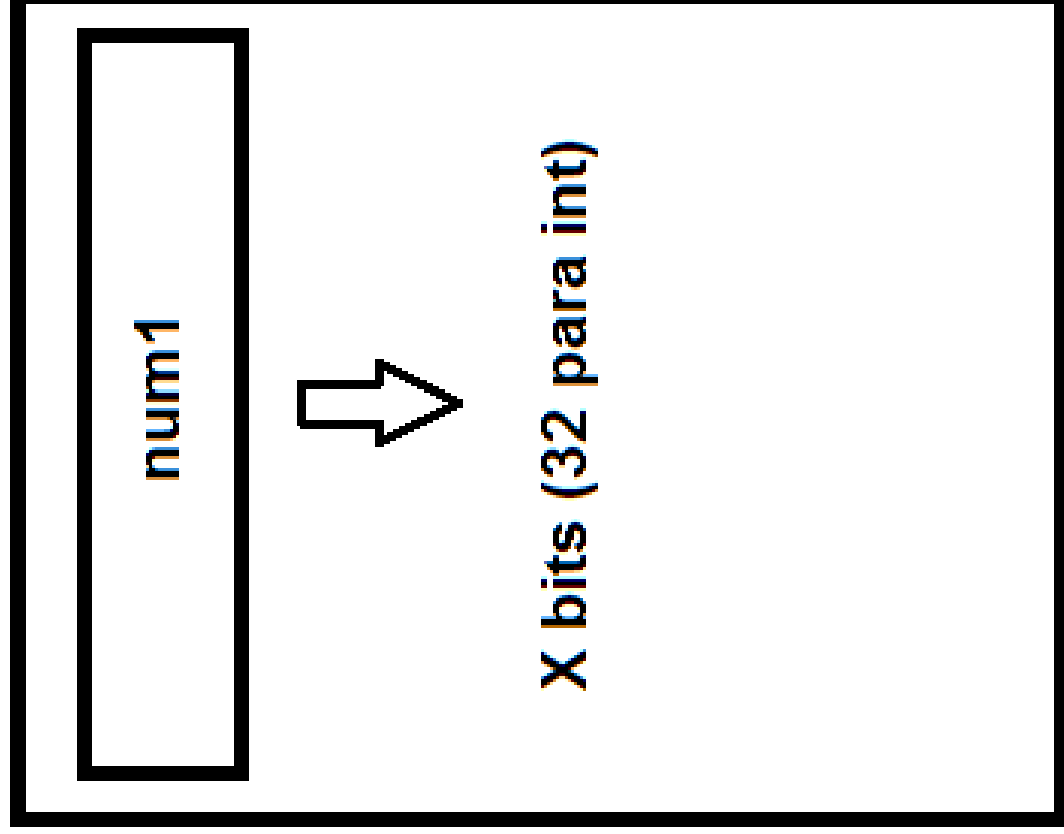
Variáveis

Para que um programa use valores (para fazer cálculos, apresentar informações na tela, etc), é preciso utilizar *variáveis*.

Variáveis são elementos que, como o próprio nome diz, terá valor variável.



endereço de
memória





Atribuição de valores

É feita da direita para esquerda.

Exemplos

```
num = 5;  
altura = 1.80;
```



Exemplo

```
#include <stdio.h>

int main()
{
    int num;

    num = 5;

    printf(“Valor de num = %d\n”, num);

    return 0;
}
```



Exemplo

```
#include <stdio.h>

int main()
{
    float num;

    num = 5.5;

    printf("Valor de num = %f\n", num);

    return 0;
}
```



Apresentando várias variáveis em um único *printf*

```
printf("var1 = %d; var2 = %d; var3 = %d", var1, var2, var3);
```



Exemplo

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

```
int main()
```

```
{
```

```
    int num1;
```

```
    float num2;
```

```
    num1 = 5;
```

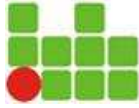
```
    num2 = 7.2;
```

```
    printf("num1 = %d, num2 = %f\n", num1, num2);
```

```
    return 0;
```

```
}
```

Operadores com variáveis



Binários (usados com dois valores)

+ soma

- subtração

* multiplicação

/ divisão

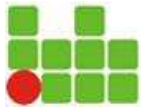
% módulo (resto da divisão)



O operador de módulo (%) divide o primeiro valor pelo segundo e retorna o resto da divisão.

Exemplo:

resto = num1 % 2;



Comentários de programa

São usados para explicar o código e para “desabilitar” trechos.



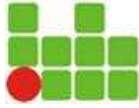
Comentário de linha

```
#include <stdio.h>

int main()
{
    // Declaração de variável e inicialização com valor
    int num = 5;

    // Saída para a tela
    printf("Valor de num = %d\n", num);

    // Finaliza o programa
    return 0;
}
```



Comentário de bloco

#include <stdio.h>

```
int main()
{
    int num1, num2, soma;
    /*
    printf("Entre com o valor de num1: ");
    scanf("%d", &num1);

    printf("Entre com o valor de num2: ");
    scanf("%d", &num2);

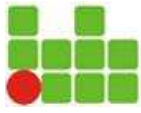
    soma = num1 + num2;

    printf("soma = %d\n", soma);
    */
    return 0;
}
```



Operadores unários

- subtração unário (altera o sinal)
- + adição unário (altera o sinal)
- & endereço da variável
- ++ incremento (pós e pré fixado)
- decremento (pós e pré fixado)



Exemplos

```
num = -num;
```

```
valor++;
```

```
++valor;
```

(ambas equivalem a **valor = valor + 1;**)

```
--valor;
```

```
valor--;
```

(ambas equivalem a **valor = valor - 1;**)



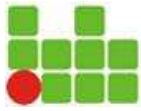
A diferença entre o pré-fixado e o pós-fixado para os operadores ++ e -- é qual o momento que a operação é feita.

Exemplos:

```
valor = num1 + num2++;
```

Ou

```
valor = num1 + ++num2;
```



Variáveis constantes

São variáveis que não terão seus valores alterados durante a execução do programa.

Exemplo:

```
const float pi = 3.141592;
```

Se for preciso mais casas decimais, pode-se usar **double**.

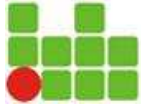
Outra forma de fazer isso é por meio de **define**, que veremos mais à frente na disciplina.



Exemplo: Escreva um programa que o volume de um cilindro com altura 10 cms e raio 5 cms, usando a fórmula:

Volume de um cilindro = $3,141592 * \text{raio} * \text{raio} * \text{altura}$


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



```
int main(){
```

```
    float raio, altura, volume;
```

```
    const float pi = 3.141592;
```

```
    raio = 5.0;
```

```
    altura = 10.0;
```

```
    volume = pi * raio * raio * altura;
```

```
    printf("O volume do cilindro é: %f", volume);
```

```
    return 0;
```

```
}
```



Conversão implícita de tipos de variáveis

Se os dois operandos de uma operação aritmética são do mesmo tipo, o resultado será desse tipo.

```
int n1 = 3;  
int n2 = 5;  
int res;
```

```
res = n1 + n2;
```

O valor a ser atribuído a **res** terá o tipo **int**.



```
float n1 = 3.0;  
float n2 = 5.0;  
float res;  
  
res = n1 + n2;
```

O valor a ser atribuído a **res** terá o tipo **float**.



Ao usar números, coloque casas decimais para ter um resultado em float.

Exemplos

`float num;`

`num = 5 / 2;`

num terá o valor 2, pois o resultado será um **int**.



```
float num;
```

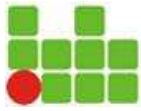
```
num = 5.0 / 2.0;
```

num terá o valor 2.5, pois o resultado será um **float**.



Ao se misturar os tipos em uma expressão, há uma ordem de conversão que é executada.

char → int → long → float → double



Exemplo

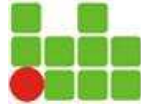
$3.5 + 1$ resulta em um float, 4.5

$4 * 2.5$ resulta em um float, 10.0



Conversão explícita (cast)

É possível determinar o tipo do resultado de uma expressão de forma explícita usando ***cast*** de tipo.



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

```
int main(){
```

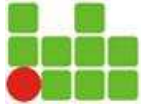
```
    float n;
```

```
    n = (float) 5 / 2;
```

```
    printf("%f", n);
```

```
    return 0;
```

```
}
```



```
#include <stdio.h>

int main(){

    float n;
    int var1 = 5, var2 = 2;

    n = (float) var1 / var2;

    printf("%f", n);

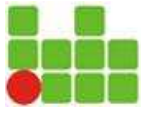
    return 0;

}
```



Entrada de dados pelo usuário

Função scanf



Função de leitura de dados (entrada)

`scanf(“expressão de controle”, argumentos);`

`scanf(“%d”, &num);`



Formatação do scanf

`%c` – caracter simples
`%d` – inteiro decimal com sinal
`%i` - inteiro decimal, hexadecimal ou octal
`%e` – notação científica
`%f` – ponto flutuante
`%s` – string de caracteres
`%x` - hexadecimal

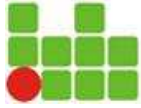
Etc...



Outra forma de usar o **scanf**:

```
scanf("%d %d %d", &num1, &num2, &num3);
```

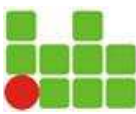
O caracter delimitador é o espaço, e assim deve ser a entrada do usuário.



```
scanf("%d:%d", &hora, &min);
```

Neste exemplo, o caracter “:” é o delimitador e indica que ambos valores devem ser informados entre “:” (poderia ser qualquer outro caracter).

Exemplo



```
#include <stdio.h>

int main()
{
    int num1, num2, soma;

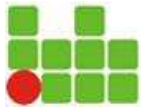
    printf("Entre com o valor de num1: ");
    scanf("%d", &num1);

    printf("Entre com o valor de num2: ");
    scanf("%d", &num2);

    soma = num1 + num2;

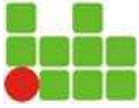
    printf("soma = %d\n", soma);

    return 0;
}
```

Refazendo o exemplo do volume do cilindro para
calcular o volume de qualquer cilindro.

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



```
int main(){
```

```
    float raio, altura, volume;
```

```
    const float pi = 3.141592;
```

```
    printf("Entre com o tamanho do raio do cilindro: ");
```

```
    scanf("%f", &raio);
```

```
    printf("Entre com o tamanho da altura do cilindro: ");
```

```
    scanf("%f", &altura);
```

```
    volume = pi * raio * raio * altura;
```

```
    printf("O volume do cilindro é: %f", volume);
```

```
    return 0;
```

```
}
```



Exercícios!