

RECAPITULANDO A AULA ANTERIOR

Como vimos na aula anterior, temos que um algoritmo é representado por um código quando implementado e esse código é usado para resolver um problema. Contudo, para chegarmos no código primeiro é necessário entender o problema, elaborar um plano, executá-lo (implementando) e verificar se a solução satisfaz o necessário. Em um código apresentado também, temos a apresentação de 3 unidades básicas em um código em C: cabeçalho, bibliotecas e código principal.

IMPLEMENTAÇÃO

Escrevendo dados com printf. Um exemplo de um código em C foi dado na aula anterior, utilizando o comando chamado **printf**. Com ele, podemos imprimir mensagens na tela do usuário, tal como mostra o exemplo [1-ola.mundo.c](#), ele simplesmente imprime a mensagem **Ola mundo!** na tela:

```
1  /*    Codigo: Exemplificando o printf
2      Autor: Carlos Adir
3  */
4  #include <stdio.h>
5  int main()
6  {
7      printf("Ola mundo!\n");
8      return 0;
9  }
```

Existem algumas configurações adicionais que podemos utilizar junto com o **printf**, tais como utilizar o caracter **'\n'** no meio do texto.

```
1  printf("Primeira Linha\nSegundaLinha TerceiraLinha\nAh! Faltou um
      '\n'!\n");
```

Da mesma maneira que existem os caracteres **'\t'**, **'\n'** entre outros que podem ser encontrados no exemplo [2-mais-exemplos-com-printf.c](#)

OS TIPOS DE DADOS

Em C, existem tipos que são responsáveis pelo controle das ações no computador e na memória. Por ser um processo mais complicado e demandar tempo, não será explicado, contudo, existem tipos chamados de **int** (inteiro), **double** (real) e **char** (símbolo), e cada um possui suas operações e modo de interagir com o programa.

O inteiro, operações, leitura e escrita.

O inteiro. Existe o tipo inteiro, que representa os números inteiros. A maneira com que você declara um inteiro é utilizando uma variável, como exemplifica o código [3-declaracao.inteiro.c](#).

```
1  /*    Codigo: Declaracao de inteiro
2      Autor: Carlos Adir
3  */
4  #include <stdio.h>
5  int main()
6  {
7      int inteiro;
8      return 0;
9  }
```

Podemos ver que podemos atribuir um valor a um inteiro, que representa a mesma coisa:

```
1  int inteiro = 3;
```

ou

```
1  int inteiro;  
2  inteiro = 3;
```

Para economizar espaço, sempre será utilizado a primeira forma.

Escrita. Com o inteiro, podemos imprimir seu valor na tela, como mostra o arquivo [4-imprimir_inteiro.c](#)

```
1  int numero = 3;  
2  printf("O numero e: %d\n", numero);
```

Operações. Com os inteiros, podemos fazer operações básicas tais como soma, mostrado pelo arquivo [5-soma_inteiro.c](#):

```
1  int num1 = 2, num2 = 3, soma;  
2  soma = num1 + num2;  
3  printf("O resultado da soma e: %d", soma);
```

Podemos montar o resultado da soma sem armazenar na variável **soma**:

```
1  int num1 = 2, num2 = 3;  
2  printf("O resultado da soma e: %d", num1+num2);
```

Da mesma maneira, podemos utilizar os operadores de $-$, \times e \div que são representados pelos símbolos $-$, $*$ e $/$, como é representado pelo exemplo [6-outras_operacoes_com_inteiros.c](#).

```
1  int num1 = 40, num2 = 5;  
2  printf("O resultado da subtracao e: %d", num1-num2);  
3  printf("O resultado da multiplicacao e: %d", num1*num2);  
4  printf("O resultado da divisao e: %d", num1/num2);
```

Contudo, e se fizermos a operação $3/2$? Podemos verificar que o resultado é 1! Mas por que 1 e não 1.5? Essa é uma das operações com inteiros, sempre quando se faz uma divisão com dois inteiros, é esperado um outro inteiro e no caso de 1.5 não é inteiro. Assim, a linguagem trunca, isto é, sempre arredonda para o valor mais baixo e então se o resultado fosse 0.99, o resultado inteiro ainda sim seria 0.

Ora! Mas eu queria que desse 1.5 e não 1! Neste caso, utilizaremos um **double** que será explicado mais adiante. Por enquanto vamos nos concentrar no inteiro.

Existe uma quinta operação chamada de **resto** e é representada por **%**. Essa operação consiste em dividir um número pelo outro e verificar qual o seu resto, assim como se faz na divisão manual. Logo, se tivermos o código como indicado por [7-resto.c](#) teremos as respostas respectivamente 1, 2 e 7.

```
1  printf("7%%2 = %d\n", 7%2); /* 1 */  
2  printf("5%%3 = %d\n", 5%3); /* 2 */  
3  printf("7%%8 = %d\n", 7%8); /* 7 */
```

Leitura. Nas operações com inteiros, é sempre interessante que receba entrada do usuário, como por exemplo queremos saber o resultado de uma soma de números que usuário coloca no computador. Para isso, é necessário que se receba números e fazemos como indica o exemplo [8-scanf.c](#) e sempre é necessário que haja uma variável para armazenar o valor lido. Para utilizar o **scanf**, assim como no exemplo, é necessário colocar o símbolo **'&'** antes do nome da variável.

```
1  int num1;  
2  scanf("%d", &num1);  
3  printf("O numero lido e: %d", num1);
```

Desta maneira, podemos ler, escrever e realizar operações com inteiros e isso será muito útil daqui para frente!

O double, operações, leitura e escrita.

O double. O double representa os números decimais com vírgula. Embora o conjunto dos números reais seja maior que os inteiros, ainda sim existe os inteiros. Sempre que possível, utilize inteiros para calcular.

A declaração e atribuição é de semelhante maneira que o inteiro, assim:

```
1 double numero = 3.1415;
```

Leitura. Para a leitura de double, simplesmente utilizar o comando scanf, que no caso gera:

```
1 double num;  
2 scanf("%lf", &num);
```

Escrita. Para escrita na tela, utiliza-se o mesmo printf, mas altera-se para:

```
1 double num = 3.1415;  
2 printf("%lf", num);
```

As operações. As operações para o double são +, -, × e ÷ e não existe o %. Da mesma maneira que o inteiro, sempre que houver operação entre dois doubles, o resultado será um double e portanto $3.0/2 = 1.5$. O exemplo é dado pelo arquivo [9-double.c](#)

A palavra chave const. Como vimos, existem as variáveis que armazenam valores e como o nome já diz, elas variam. Contudo, às vezes queremos adicionar algumas constantes que não mudam ao longo do programa, como por exemplo:

```
1 double pi = 3.141593;
```

Assim, é interessante que ao longo de um código, não haja atribuição como a abaixo:

```
1 pi = 5.3;
```

Assim, utilizamos a palavra **const** para determinar se uma variável é constante ou não e é declarado da maneira:

```
1 const double pi = 3.141593;  
2 pi = 5.3 /* 0 programa vai acusar o erro que e mudar o valor de uma  
           constante */
```

Essa é uma tática para que ao longo de um programa maior, evitar de confundir o programador. Para aulas seguintes será importante essa constante na passagem para funções.