Funções

Exercício

Construa um programa que possua a função "EDivisivel(int **a**, int **b**)", escrita por você. A função deverá retornar 1 se **a** for divisível por **b**. Caso contrário, a função deverá retornar zero. O programa deve ler dois números fornecidos pelo usuário (**a** e **b**, respectivamente), e utilizar a função EDivisivel para retornar uma mensagem dizendo se **a** é ou não divisível por **b**.

UNIVASF

```c
#include <stdio.h>
EDivisivel(int a, int b)
{
    return(a%b?0:1);
}
int main() {
    int a,b;
    printf ("\nPrograma que retorna se \"a\" eh divisivel por \"b\"");
    printf ("\n\nEntre com o valor de \"a\": ");
    scanf("%d",&a);
    do {
        printf ("\nEntre com o valor de \"b\": ");
        scanf("%d",&b);
    }while(!b);
    if (EDivisivel(a,b))
        printf("\n\"a\" eh divisivel por \"b\"");
    else
        printf("\n\"a\" nao eh divisivel por \"b\"");
}
```

```c
printf("\n\"a\"%seh divisivel por \"b\"",
EDivisivel(a,b)?" ":" nao ");
```

Funções

- O tipo void

Em inglês, **void** quer dizer vazio e é isto mesmo que o **void** significa. Ele nos permite fazer funções que não retornam nada:

void nome_da_função (declaração_de_parâmetros){...}

Numa função, como a demonstrada acima, não temos valor de retorno na declaração **return**. Aliás, neste caso, o comando **return** não é necessário na função. Contudo, podemos utilizá-lo em pontos onde desejamos que a função finalize sua execução.

UNIVASF

Funções

Conforme podemos observar nas funções *main* dos programas feitos até o momento uma função pode não ter parâmetros.

Logo, podemos fazer funções como a presente no programa a seguir:

```c
#include <stdio.h>
void Mensagem (void)
{
    printf ("Ola! Eu estou vivo.\n");
}
int main ()
{
    Mensagem();
    printf ("\tDiga de novo:\n");
    Mensagem();
    return 0;
}
```

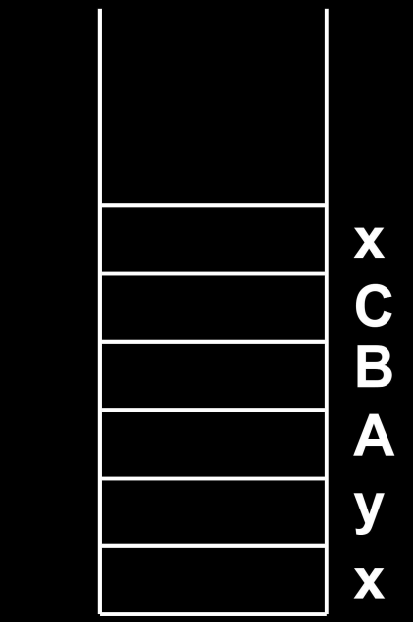
Funções
Escopo de Variáveis

UNIVASF

```
func1 (...)
{
      int abc,x,z;
      ...
}
func (...)
{
      int z;
      ...
}
main ()
{

      if (...)

            float A,B,C,x;
            ...
      }
      ...
}
```



Pilha de Variáveis

Funções

- Parâmetros formais

Os parâmetros formais são declarados como sendo as entradas de uma função. Um parâmetro formal é uma “espécie de variável local da função”. É possível alterar o valor de um parâmetro formal, destacando, que esta alteração não terá efeito na variável que foi passada à função.

Isto tem sentido. Pois, quando passamos parâmetros para uma função, na linguagem C, são passadas apenas cópias das variáveis. Isto é, os parâmetros formais existem independentemente das variáveis que foram passadas para a função. Eles tomam apenas uma cópia dos valores passados para a função.

UNIVASF

```c
#include <stdio.h>
void func (int a, int b)
{
      ...
      a=5;
}
main ()
{
      int x=1,y=2;
      ...
      func (x,y);
      printf ("%d", x);
      ...
      func (-1, -2);
      ...
}
```

Funções

Exercício:

Escreva o código fonte de um programa, na linguagem C, que manipule um vetor de inteiros com dez elementos. O programa deve possuir uma função que receba o vetor e retorne o maior valor contido no mesmo. As seguintes manipulações devem ser feitas: o vetor deve ser inicializado e, por meio da utilização da função mencionada, o maior valor contido no vetor deve ser impresso na saída padrão.

UNIVASF

```c
#include <stdio.h>
#define n 10
int maior_valor (int vetor[n]) (int *vetor)
{
   int i,maior_v=vetor[0];
   for(i=1;i<n;++i)
      if(maior_v<vetor[i])
         maior_v=vetor[i];
   return maior_v;
}
main()
{
   int i,v[n];
   for(i='\0';i<n;++i)
   {
      printf("\nEntre com o elemento v[%d]: ",i+1);
      scanf("%d",v+i);
   }
   printf("\nO maior valor contido no vetor eh: %d", maior_valor(v));
   return 0;
}
```

Funções

- Variáveis globais

Variáveis globais são declaradas fora de todas as funções do programa. Elas são conhecidas e podem ser alteradas por todas as funções do programa que seguem sua declaração.

Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local. Vamos ver um exemplo. Observação: O mesmo vale para parâmetros formais.

```
int z,k;
func1 (...)
{
    int x,y;
    ...
}
void func2 (...)
{
    int x,y,z;
    ...
    z=10;
    ...
}
main ()
{
    int count;
    z=7;
    func2(...);
    ...
}
```

Exercício:

Analise o programa abaixo e indique o que será impresso na saída padrão.

Precedência (Hierarquia nas operações)

Hierarquia	Operação
1	Parênteses
2	Função
3	++,--
4	- (menos unário)
5	*, /, %
6	+, -

```c
#include <stdio.h>
int num;
int func(int a, int b)
{
   a = (a+b)/2;  = (0+50)/2  = 25
   num -= a+1;  = num – (a+1)  = 10 – (25+1)  = – 16
   return a;
}
int main()
{
   int first = 0, sec = 50;
   num = 10;              10                  0                  50
   printf("\nnum antes = %d\tfirst antes = %d\tsec antes = %d", num, first, sec);
   num += func(first, sec); num  +  func(0, 50)   ou seja   -16 +  25   =   9
   printf("\nnum depois = %d\tfirst depois = %d\tsec depois = %d\n", num, first, sec);
}
```

num antes = 10 | first antes = 0 | sec antes = 50

num depois = 9 ? first depois = 0 | sec depois = 50

Funções

Exercício:

Escreva o código fonte de um programa, na linguagem C, que manipule um vetor de inteiros com dez elementos. O programa deve possuir uma função responsável pela inicialização do vetor e outra função que efetue a impressão do vetor na saída padrão. Por meio das funções mencionadas, o programa deve inicializar o vetor e em seguida retorná-lo no monitor.