Ponteiros

- Ponteiros para Ponteiros (continuação)

Na linguagem C podemos declarar ponteiros para ponteiros para ponteiros, ou então, ponteiros para ponteiros para ponteiros para ponteiros e assim por diante.

Para fazer isto basta aumentar o número de asteriscos na declaração.

Para acessar o valor desejado apontado por um ponteiro para ponteiro, o operador asterisco deve ser aplicado duas vezes, como mostrado anteriormente e no exemplo a seguir:

UNIVASF

Ponteiros

- Ponteiros para Ponteiros (continuação)

```c
#include <stdio.h>
int main()
{
    float pi = 3.1415, *pf, **ppf;
    pf = &pi;
    ppf = &pf;
    printf("\n%.4f", **ppf);
    printf("\n%.4f", *pf);
}
```

Ponteiros

Exercício:

Verifique o programa abaixo. Encontre o(s) seu(s) erro(s) e corrija-o(s) para que o mesmo escreva o número 10 na tela.

```
#include <stdio.h>
int main()
{
      int x, *p, **q;
      p = &x;
      q = &p;
      x = 10;
      printf("\n%d\n",**q);
}
```

Funções

Funções

Funções são as estruturas que permitem a usuário separar seus programas em blocos (subprogramas). Para fazermos programas grandes e complexos devemos construí-los bloco a bloco.

Uma função na linguagem C tem a seguinte forma geral:

```
tipo_de_retorno nome_da_função (declaração_de_parâmetros)
{
        corpo_da_função
}
```

Funções

O tipo-de-retorno é o tipo do valor que a função vai retornar. O default é o tipo **int**, ou seja, o tipo-de-retorno assumido por omissão. A declaração de parâmetros é uma lista com a seguinte forma geral:

tipo nome1, tipo nome2, ... , tipo nomeN

Observe que o tipo deve ser especificado para cada uma das N variáveis de entrada. É na declaração de parâmetros que informamos ao compilador quais serão as entradas da função (assim como informamos a saída no tipo-de-retorno).

É no corpo da função que as entradas são processadas, a saída é gerada ou outras operações são executadas.

Funções

- Comando return

Forma geral:

return valor_de_retorno; ou *return;*

Quando se executa uma declaração **return** a função é encerrada imediatamente e, se o valor de retorno é informado, a função retorna este valor. É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.

```c
#include <stdio.h>
int Square (int a)
{
    return (a*a);
}
int main ()
{
    int num;
    printf ("\nEntre com um numero: ");
    scanf ("%d",&num);
    num=Square(num);
    printf ("\n\n0 seu quadrado vale: %d\n",num);
}
```

```c
#include <stdio.h>
int Square (int a)
{
    return (a*a);
}
int main ()
{
    int num;
    printf ("\nEntre com um numero: ");
    scanf ("%d",&num);


}
```

Funções

Observação:

Devemos nos lembrar que a função **main()** é uma função e como tal devemos tratá-la. A função **main()** retorna um inteiro. Isto pode ser interessante se quisermos que o sistema operacional receba o valor de retorno da função **main()**. Se assim o quisermos, devemos nos lembrar da seguinte convenção: se o programa retornar zero, significa que ele terminou normalmente, e, se o programa retornar um valor diferente de zero, significa que o programa teve um término anormal.

UNIVASF

```c
#include <stdio.h>
int EPar (int a)
{
     if (a%2)
          return 0;
     else
          return 1;
}
int main ()
{
     int num;
     printf ("Entre com numero: ");
     scanf ("%d",&num);
     if (EPar(num))
          printf ("\n\nO numero e par.\n");
     else
          printf ("\n\nO numero e impar.\n");
     return 0;
}
```

```c
return (!(a%2));
```

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**.

```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);
    printf("\n\"a\"%seh divisivel por \"b\"",
    EDivisivel(a,b)?" ":" nao ");
    if (EDivisivel(a,b))
        printf("\n\"a\" eh divisivel por \"b\"");
    else
        printf("\n\"a\" nao eh divisivel por \"b\"");
}
```

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.

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

Funções

- Escopo de variáveis

O escopo de variáveis é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa.

Veremos agora três tipos de variáveis, no que se refere ao escopo:

Funções

- Variáveis locais

Variáveis locais são aquelas que só têm validade dentro do bloco no qual são declaradas. Podemos declarar variáveis dentro de qualquer bloco.

Só para lembrar: um bloco começa quando abrimos uma chave e termina quando fechamos a chave. A declaração de variáveis locais é a primeira coisa que devemos colocar num bloco.

```
func1 (...)
{
      int abc,x,z;
      ...
}
func (...)
{
      int z;
      ...
}
main ()
{
      int x,y;
      if (...)
      {
            float A,B,C,x;
            ...
      }
      ...
}
```

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.

```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.