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.

```c
#include <stdio.h>
#define n 10
int vetor[n];
void inicializar_vetor()
{
    int i;
    for(i='\0';i<n;++i)
    {
        printf("\nEntre com o elemento v[%d]: ",i+1);
        scanf("%d",vetor+i);
    }
}
```

```c
void imprimir_vetor()
{
    int i;
    for(i='\0';i<n;++i)
        if(!i)
            printf("vetor = { %d, ", vetor[i]);
        else
            if(i==n-1)
                printf("%d }", vetor[i]);
            else
                printf("%d, ", vetor[i]);
}
```

```c
int main()
{
    inicializar_vetor();
    imprimir_vetor();
}
```

Funções
Passagem de Parâmetros

UNIVASF

Funções

- Passagem de parâmetros por valor e passagem por referência

Vimos que quando chamamos uma função os parâmetros formais desta recebem uma cópia dos valores dos parâmetros que são passados para a mesma. Isto quer dizer, que alterações nos parâmetros formais, que ocorrem dentro da função, não geram alterações nos valores dos parâmetros externos à função.

UNIVASF

Funções

- Passagem de parâmetros por valor e passagem por referência (continuação)

Este tipo de passagem de parâmetros é denominado passagem por valor. Isto ocorre, porque são passados para a função apenas os valores dos parâmetros e não os próprios parâmetros. Vejamos o exemplo a seguir:

UNIVASF

```c
#include <stdio.h>
float sqr (float num)
{
    num=num*num;
    return num;
}
int main ()
{
    float num,sq;
    printf ("Entre com um numero: ");
    scanf ("%f",&num);
    sq=sqr(num);
    printf ("\n\nO numero original e: %f\n",num);
    printf ("O seu quadrado vale: %f\n",sq);
}
```

Funções

- Passagem de parâmetros por valor e por referência (continuação)

Outro tipo de passagem de parâmetros para uma função ocorre quando alterações nos parâmetros formais, dentro da função, alteram os valores dos parâmetros que foram passados para a função. Este tipo de chamada de função tem o nome de "passagem por referência".

Funções

- Passagem de parâmetros por valor e por referência (continuação)

A linguagem C só permite passagem por valor.

Isto é bom quando queremos usar os parâmetros formais à vontade dentro da função, sem termos que nos preocupar em estar alterando os valores das variáveis que tiveram seus valores passados aos parâmetros da função.

Mas, isto também pode ser ruim, porque, às vezes, podemos desejar que mudanças nos valores dos parâmetros formais reflitam nas variáveis que tiveram seus valores copiados para os parâmetros formais da função.

UNIVASF

Funções

- Passagem de parâmetros por valor e passagem por referência (continuação)

Quando queremos alterar, dentro das funções, as variáveis que tiveram seus valores passados para uma função, nós podemos declarar os parâmetros formais como sendo *ponteiros*.

O ponteiro é a "referência" que precisamos para poder alterar dentro da função uma variável externa à função. O único inconveniente é que, quando usarmos uma função, que se utiliza deste artifício, teremos de lembrar de colocar um **&** na frente da variável que estivermos passando sua referência para a função.

Veja um exemplo:

UNIVASF

```c
#include <stdio.h>
void incrementa (int *a,int b)
{
    *a+=b;
}
int main (void)
{
    int num;
    num=100;
    printf ("\nValor de num antes da chamada %s%d",
    "da funcao:  ", num);
    incrementa (&num,50);
    printf ("\nValor de num depois da chamada %s%d",
    "da funcao:  ", num);
}
```

Funções

Exercício:

Construa um programa em ‘C’ que possua uma função responsável por fazer a troca de valores entre duas variáveis. O programa deve receber dois valores inteiros, através da entrada padrão, armazená-los em variáveis, com a utilização da função mencionada trocar o conteúdo das variáveis e, depois, retorná-las na saída padrão.

UNIVASF

```c
#include <stdio.h>
void Swap (int *a,int *b)
{
     int temp;
     temp=*a;
     *a=*b;
     *b=temp;
}
main ()
{
     int num1,num2;
     printf("\nEntre com o primeiro valor: ");
     scanf ("%d",&num1);
     printf("\nEntre com o segundo valor: ");
     scanf ("%d",&num2);
     Swap (&num1,&num2);
     printf ("\n\nEles agora valem %d  %d\n",
     num1,num2);
}
```

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. **Obs. o programa não pode possuir variáveis globais.**

UNIVASF

```c
#include <stdio.h>
#define tamanho 10
void inicializar(int *v)
{
    int i;
    for(i=0;i<tamanho;i++)
    {
        printf ("\nEntre com v[%i]: ",i+1);
        scanf("%i",v+i);
    }
}
```

```c
void imprimir(int *v)
{
    int i;
    for(i=0;i<tamanho;i++)
        if (!i)
            printf ("\nvetor[ %i, ",v[0]);
        else
            if (i==tamanho-1)
                printf ("%i ]",v[i]);
            else
                printf ("%i, ",v[i]);
}
main ()
{
     int vetor[tamanho];
     inicializar(vetor);
     imprimir(&vetor[0]);
}
```

Protótipos de Funções
Arquivos Cabeçalhos

UNIVASF

Funções

- Protótipos de funções

Protótipos são declarações de funções. Ist
você declara uma função que irá usar. O compilador toma então conhecimento do formato daquela função antes de compilá-la. Possibilitando assim, antes da compilação da função, a validação da utilização da mesma.

Um protótipo tem o seguinte formato:

TipoDeRetorno NomeDaFunção (DeclaraçãoDeParâmetros);

onde o *TipoDeRetorno*, o *NomeDaFunção* e a *DeclaraçãoDeParâmetros* são os mesmos que você pretende usar quando realmente escrever a função.

UNIVASF

```c
#include <stdio.h>
float Square (float a);
int main ()
{
    float num;
    printf ("Entre com um numero: ");
    scanf ("%f",&num);
    num=Square(num);
    printf ("\n\nO seu quadrado vale: %f\n",num);
    return 0;
}
float Square (float a)
{
    return (a*a);
}
```

Funções

- Arquivos cabeçalhos

Arquivos cabeçalhos ou arquivos headers, são aqueles que temos mandado o compilador incluir no início de nossos programas e que sempre terminam em **.h**. Estes arquivos não contêm os códigos das funções. Eles só contêm *protótipos* de funções.

Funções

O corpo das funções cujos protótipos estão no arquivo-cabeçalho, no caso das funções do próprio C, já estão compilados e normalmente são incluídos no programa no instante da "linkagem". Este é o instante em que todas as referências a funções cujos códigos não estão nos arquivos fontes são resolvidas, buscando este código nos arquivos de bibliotecas.

UNIVASF

Funções

- Arquivos cabeçalhos (continuação)

Se você criar algumas funções que queira aproveitar em vários programas futuros, você pode escrever arquivos cabeçalhos e incluí-los também.

Vamos supor que a função 'int EPar(int a)', vista anteriormente, seja importante em vários programas, e desejemos declará-la num módulo separado.

UNIVASF

Funções

- Arquivos cabeçalhos (continuação)

No arquivo de cabeçalho chamado por exemplo de “funcao.h” teremos a seguinte declaração:

```
int EPar(int a);
```

Funções

- Arquivos cabeçalhos (continuação)

O código da função será escrito num arquivo a parte. Vamos chamá-lo de “funcao.c”. Neste arquivo teremos a definição da função:

```
int EPar (int a)
{
   return (a%2?0:1);
}
```

Funções

- Arquivos cabeçalhos (continuação)

Por fim, no arquivo do programa principal teremos a função main(). Vamos chamar este arquivo aqui de “principal.c”.

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

Funções

- Arquivos cabeçalhos (continuação)

Este programa poderia ser compilado usando a seguinte linha de comando para o gcc:

```
gcc principal.c funcao.c -o saida
```

onde “saida” seria o nome do arquivo executável gerado.

Funções

Exercício:

Escreva um programa que faça uso da função EDivisivel(int a, int b), criada anteriormente. Além desta função o programa deverá usar: Max(int a, int b), que retorna o maior dos parâmetros; VMedio(int a, int b), que retorna o valor médio (inteiro) entre a e b, você deverá escrever estas funções. Organize o seu programa em três arquivos: o arquivo exercicio.c, conterá o programa principal que deve se utilizar das funções descritas; o arquivo func.c conterá o corpo das funções; o arquivo func.h conterá os protótipos das funções. Compile os arquivos e gere o executável.

UNIVASF

```c
/*conteúdo do arquivo func.h*/

int EDivisivel(int a, int b);
int Max(int a, int b);
int VMedio(int a, int b);
```

```c
/*conteúdo do arquivo exercicio.c*/
#include <stdio.h>
#include "func.h"
main()
{
   int a,b,opcao;
   printf ("\n\nEntre com o valor inteiro para \"a\": ");
   scanf("%d",&a);
   printf ("\nEntre com o valor inteiro para \"b\": ");
   scanf("%d",&b);
   do
   {
      printf("\n\nOpcoes:\n1-Para saber se a eh %s", "divisivel por b");
      printf("\n2-Para saber qual o maior valor\n");
      printf("3-Para saber o valor medio\n");
      printf("4-Para sair\nEntre com sua opcao: ");
      scanf("%d",&opcao);
```

```
    switch (opcao)
    {
       case 1:  if(EDivisivel(a,b))
                   printf("\n\"a\" eh divisivel por \"b\"");
                else
                   printf("\n\"a\" nao eh divisivel por \"b\"");
                break;
       case 2:  printf("\nO maior valor eh %d",Max(a,b));
                break;
       case 3:  printf("\nO valor medio eh %d",VMedio(a,b));
                break;
       case 4:  break;
       default: printf("\nOpcao invalida!");
    }
   }while(opcao!=4);
}
```

```c
/*conteudo do arquivo func.c*/

int EDivisivel(int a, int b)
{
   return(a%b?0:1);
}
int Max(int a, int b)
{
   return(a>b?a:b);
}
int VMedio(int a, int b)
{
   return((a+b)/2);
}
```

Funções

linha de comando para compilacao no gcc:

gcc exercicio.c func.c –o exercicio

Argumentos argc e argv

UNIVASF

Funções

- Os Argumentos argc e argv

A função **main()** pode ter parâmetros formais. Mas, o programador não pode escolher quais serão eles.

A declaração mais completa que se pode ter para a função **main()** é:

```
int main (int argc, char *argv[]);
```

Funções

Os parâmetros **argc** e **argv** dão ao programador acesso à linha de comando com a qual o programa foi chamado.

O **argc** (argument count) é um inteiro e possui o número de argumentos com os quais a função **main()** foi chamada na linha de comando.

O **argv** (argument values) é um vetor de strings. Cada string deste vetor é um dos parâmetros da linha de comando. **argc** é utilizado para sabermos quantos elementos temos em **argv**.

UNIVASF

Funções

- Os Argumentos argc e argv (continuação)

Exemplo: O programa a seguir faz uso dos parâmentros *argv* e *argc*. O programa recebe da linha de comando o dia, mês e ano correntes, e imprimi a data em formato apropriado. Veja o exemplo, supondo que o executável se chame “data”:

data 19 04 06

O programa imprimirá:

19 de abril de 2006

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
   int mes;



   if(argc == 4)
   {
      mes = atoi(argv[2]); /*poderia ter usado sscanf()*/
      if (mes<1 || mes>12)
         printf("Erro!\nMes invalido!");
      else
         printf("\n%s de %s de 20%s", argv[1],
      nomemes[mes-1], argv[3]);
   }
   else
      printf("Erro!\nUso: data dia mes ano, todos inteiros");
   return 0;
}
```

Funções

Exercício:

Construa um programa que receba da linha de comando, com a qual o programa é executado, um número natural, e retorne seu fatorial na saída padrão.

UNIVASF

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
   int num, fat=1;
   if(argc == 2 && atoi(argv[1])>=0)
   {
      for (sscanf(argv[1],"%d",&num);num>1;num--)
         fat*=num;
      printf ("\nO fatorial de  %s eh %d\n",argv[1],fat);
   }
   else
      printf("Erro!\nUso: exercicio numero, %s",
      "numero dever representar um natural");
   return 0;
}
```