Vetores

Declaração de Vetores

<tipo> <nome> [<tamanho>]

Ex.: char mensagem [10]

A indexação começa com zero.

As posições de um vetor quando criadas guardam valores desconhecidos ("lixo").

A inicialização pode ser feita dentro do programa ou na própria declaração.

Ex.: int vet [3] = {10, 20, 30};

int vet [] = {10, 20, 30};

vet[0] = 10; vet[1] = 20; vet[2] = 30;

Vetores

Vetores de caracteres

```
char vet [5] = {'a', 'b', 'c'};
vet [0] = 'a'; vet [1] = 'b'; vet [2] = 'c';
vet [3] = '\0'; vet[4] = '\0';
```

```
char vet [5] = "abc";
```

'a', 'b', 'c', '\0'

Vetores

Vetores multidimensionais

char nomes [4][10];

nomes [0][0] = ‘a’;
printf (“nome = %s”, nomes [3]);

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

int main()
 { int i;
   char nomes [4][10] = {"Adriana", "Bia",
"Carla", "Nadja"};

   for (i=0;i<4;i++)
    if (strcmp("Nadja", nomes[i])==0)
        printf ("nome = %s esta na %d
posicao", nomes [i], i);

   return 0;}
```

Funções importantes:
strcpy (a, b) – copia a cadeia de caracteres de *b* para *a.*
strcmp (a, b) – compara as cadeias *a* e *b*, retornando três possíveis valores: 1) maior que zero se *a* for maior que *b*
2) igual a zero se ambas forem iguais
3) menor que zero caso *b* seja maior que *a*

Vetores

Exercício: Escrever um programa em C que leia 10 números inteiros distintos e os ordene do menor para o maior.

```
#include <stdio.h>

#define N 10

int vetor [N];
int i, j, aux;

int main()
{
// Entrada dos números
for (i=0;i<N;i++)
 {
    printf ("\nEntre com o numero: ");
    scanf ("%d", &vetor[i]);
 }
 // Ordenação dos números
  for (i=0;i<(N-1);i++)
   for (j=(i+1);j<N;j++)
      if (vetor[j]<vetor[i])
        {
          aux = vetor[i];
          vetor[i]=vetor[j];
          vetor[j]=aux;
        }
// Impressão do vetor ordenado
printf ("O vetor ordenado e´:\n ");
for (i=0;i<N;i++)
 {
  printf ("%d ", vetor[i]);
  }
 return 0;
}
```

Apontadores

É um ponteiro para uma posição de memória.

<tipo> *<nome>;

Ex.: char *pc; /* *pc* é um ponteiro para uma variável do tipo char */

Atribuição de valores para ponteiros

Para se referenciar diretamente o endereço de uma variável, usa-se o operador **&** antes do nome da variável.

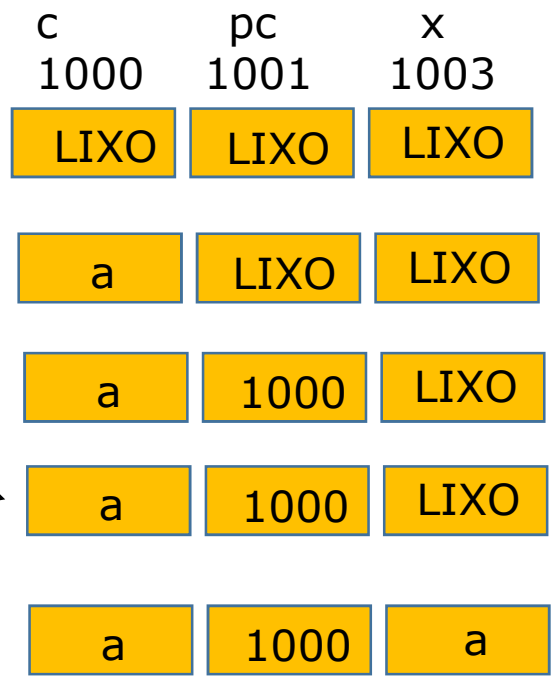
Para se obter o valor armazenado em um determinado endereço da memória utiliza-se o operador *****, que trata seu operando como o endereço de uma variável.

Ex.: int x, *px;

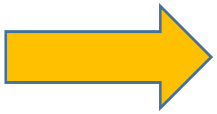
px = &x; /* o conteúdo de *px* será o endereço da variável *x* */

Apontadores

```
int main()
 { char c, *pc, x;

   c = 'a';
   pc = &c;
   printf ("%c", *pc);
   x = *pc;

   return 0;

 }
```




```
pc = &c;
x = *pc;
```



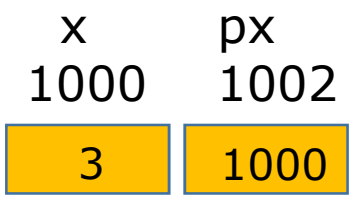
```
x = c;
```

Apontadores

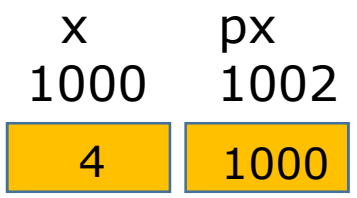
Aritmética com Ponteiros

```
int x; *px;
px = &x;
*px = 3;
```

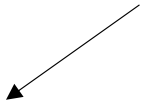
x com valor igual a 3



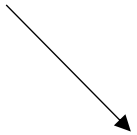

```
*px += 1;
```



(*px)++ é diferente *px++



Incrementa o conteúdo apontado por px



Incrementa o conteúdo de px

Apontadores e Vetores

Vetores e ponteiros têm conceitos equivalentes, pois qualquer operação executada em vetores pode ser feita utilizando ponteiros.

```
int  vet [10], *pv;

pv = &vet[0];
```

Funções

É um trecho de programa com atribuições bem definidas.

Sintaxe:

[<tipo>]<nome>([<tipo> <parâmetros>])]

{

  [<declaração de variáveis>]

  [<comandos>]

}

Funções

```
#include <stdio.h>

void pot (int x, int y);

int a, b;

int main ()
{
  a=10; b=2;
  pot (a, b);
}

void pot (int x, int y)
{
  int i, potencia = 1;
  for (i=0;i<y;i++)
   {
     potencia *=x;
     printf ("%d\n", potencia);
   }
}
```

```
#include <stdio.h>

int a, b;

void pot (int x, int y)
{
  int i, potencia = 1;
  for (i=0;i<y;i++)
  {
    potencia *=x;
    printf ("%d\n", potencia);
  }
}

int main ()
{
  a=10; b=2;
  pot (a, b);
}
```

Retorno de Função e Declaração de Tipo

Em C uma função pode ou não retornar valor ou retornar um e apenas um valor à função que a chamou. Este retorno é feito pelo comando *return* segundo a sintaxe abaixo:

return (<expressão>); /* retorna valor */

return (); /* não retorna valor */

return; /* não retorna valor */

```
#include <stdio.h>

int a, b;

int pot (int x, int y)
{
  int i, potencia = 1;
  for (i=0;i<y;i++)
   {
     potencia *=x;
     printf ("%d\n", potencia);
  }
  return (potencia);
}

int main ()
{
  a=10; b=2;
  printf ("\n%d ", pot (a, b));

}
```

Passagem de Parâmetros

Há dois tipos de passagem de parâmetros:

- Por valor – onde se passa para dentro da função uma cópia do parâmetro real. Qualquer alteração no conteúdo do parâmetro formal só vale dentro da função e não afeta o conteúdo fora dela.

- Por referência – onde se passa para dentro da função o próprio conteúdo do parâmetro real, isto é, seu endereço. Qualquer alteração no conteúdo do parâmetro formal altera o conteúdo do parâmetro real fora da função.

Passagem de Parâmetros

Por Valor

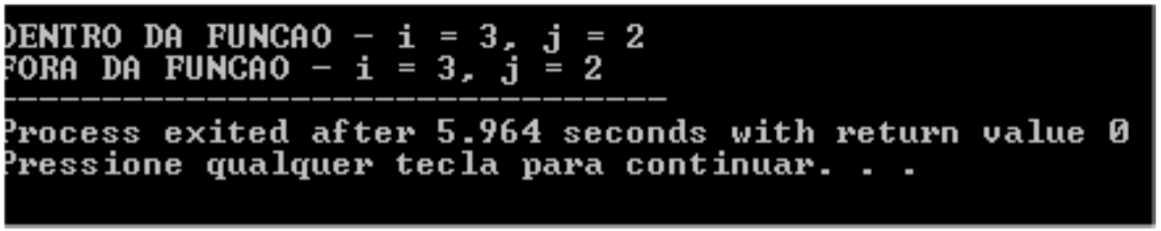
```
DENTRO DA FUNCAO - i = 3, j = 2
FORA DA FUNCAO - i = 2, j = 3
--------------------------------

Process exited after 5.556 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

```
#include <stdio.h>

int  i = 2, j = 3;

void troca (int i, int j);

int main ()
{
  troca (i, j);
  printf ("\nFORA DA FUNCAO - i = %d, j = %d", i, j);
}

void troca (int i, int j)
{
  int temp;

  temp = i;
  i = j;
  j = temp;
  printf ("\nDENTRO DA FUNCAO - i = %d, j = %d", i, j);
}
```

Passagem de Parâmetros

Por Referência

```
#include <stdio.h>

int  i = 2, j = 3;

void troca (int *i, int *j);

int main ()
{
  troca (&i, &j);
  printf ("\nFORA DA FUNCAO - i = %d, j = %d", i, j);
}

void troca (int *i, int *j)
{
  int temp;

  temp = *i;
  *i = *j;
  *j = temp;
  printf ("\nDENTRO DA FUNCAO - i = %d, j = %d", *i, *j);
}
```



Passagem de Parâmetros

Com Vetores

Na passagem de parâmetros com vetores, a função não recebe uma cópia do vetor, mas o endereço da sua primeira posição.

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

char nomes [4][10] = {"Adriana", "Bia",  "Carla", "Nadja"};

void buscaStr (char nome[]);

int main()
 {

   char nome [10];

   printf ("Entre com o nome: ");
   scanf ("%s", nome);
   buscaStr (nome);

   return 0;
}

void buscaStr (char nome[])
{
 int i;

 for (i=0;i<4;i++)
    if (strcmp(nome, nomes[i])==0)
        printf ("nome = %s esta na %d  posicao", nomes [i], i);
}
```

Passagem de Parâmetros

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

char nomes [4][10] = {"Adriana", "Bia",  "Carla", "Nadja"};

void buscaStr (char *nome);

int main()
 {

   char nome [10];

   printf ("Entre com o nome: ");
   scanf ("%s", nome);
   buscaStr (nome);

   return 0;
}

void buscaStr (char *nome)
{
 int i;

 for (i=0;i<4;i++)
    if (strcmp(nome, nomes[i])==0)
        printf ("nome = %s esta na %d  posicao", nomes [i], i);

}
```

Recursividade

Uma função é dita recursiva quando existe dentro da função uma chamada para ela mesma.

```
#include<stdio.h>

long fatorial (int n);

int main ()
{
  int n = 5;
  printf ("%ld", fatorial(n));


}

long fatorial (int n)
 {
   long res;

   if (n==0)
    {
      res = 1L;
      return (res);
    }
    res = n*fatorial(n-1);
    return (res);
 }
```

Exercício

Escrever um programa em C que calcule a combinação simples de dois números.

$$C_s^n = \binom{n}{s} = \frac{n!}{s! \cdot (n-s)!}$$

```
#include<stdio.h>

long fatorial (int n);

int main ()
{
  int n, s;
  float  comb;

  printf ("Entre com o valor de n: ");
  scanf ("%d", &n);
  printf ("\nEntre com o valor de s: ");
  scanf ("%d", &s);
  comb = fatorial (n)/(fatorial(s)*fatorial(n-s));
  printf ("%.2f", comb);

}

long fatorial (int n)
 {
   long res;

   if (n==0)
    {
      res = 1L;
      return (res);
    }
    res = n*fatorial(n-1);
    return (res);
 }
```