Declaração de variáveis

```
#include <stdio.h>

/*protótipos */
void usaLocal(void);
void usaStaticLocal(void);
void usaGlobal(void);

int x=1;    /* var global */

int main()
{
  int x = 5; /* var local */

   printf("\n%d\n", x);

  {   int x = 7;
      printf("\n%d\n", x);
  }
```

```
  printf("\n%d\n", x);

  usaLocal();

  usaStaticLocal();

  usaGlobal();

  usaLocal();

  usaStaticLocal();

  usaGlobal();

  printf("\n%d\n", x);

  return 0;
}
```

Declaração de variáveis

```
void usaLocal(void)
{
     int x = 25;
     printf("\n%d\n", x);
     x++;
     printf("\n%d\n", x);
}

void usaStaticLocal(void)
{
     static int x = 50;
     printf("\n%d\n", x);
     x++;
     printf("\n%d\n", x);
}
```

```
void usaGlobal(void)
{
  printf("\n%d\n",x);
  x *= 10;
  printf("\n%d\n", x);
}
```

Apontadores – Alocação dinâmica

- Operadores unários:
  - & (endereço de)
  - * (conteúdo de)

- Alocação dinâmica:

- Biblioteca ***stdlib.h*** contem uma série de funções pré-definidas para tratar alocação dinâmica de memória e constantes predefinidas (**NULL**).

- Funções:

  - void * **malloc**(int num_bytes **);**

  - void **free(** void * p **);**

Alocação dinâmica

- Função **malloc**
  - recebe como parâmetro o número de bytes a serem alocados.
  - retorna um ponteiro genérico para o endereço inicial da área de memória alocada ou o endereço NULL se não houver espaço livre.

- Função **free**:
  - recebe como parâmetro um apontador com o endereço da memória a ser liberada..
  - **Só** pode ser liberada memória que tenha sido **alocada dinamicamente.**

- O operador **sizeof** retorna o número de bytes ocupado por um tipo..

Exemplo I

```
#include <stdio.h>
#include <stdlib.h>
float media(int, float *);
int main ( void )
{  float *d;   float med;   int i,n;
   scanf("%d",&n);
   d = (float *) malloc(n*sizeof(float));
   if (d==NULL){   printf("sem memoria\n");   exit(1); }
   for ( i = 0; i < n; i++ )
        scanf("%f", &d[i]);
   med = media(n,d);
   printf ( "Media = %f \n", med);   free(d);
   system("PAUSE");return 0;
}
float media(int n, float *v){
     int i=0; float m = 0.0;
     for (  ; i < n ; i++)
         m  += *(v + i); m /= n; return m;
}
```

Exemplo II

- Suponha que a variável ***a*** é armazenada na posição de memória 60000. O que é impresso pelo seguinte programa?

```
#include <stdio.h>
int main()
{
    int a;
    int *aPtr;
    a = 7;
    aPtr = &a;
    printf("O endereço de a é: %p \n", &a);
    printf("O valor do apontador é: %p \n", aPtr);
    printf("O valor de a é: %d \n", a);
    printf("O valor de a é: %d \n", *aPtr);
    printf("O valor de *&aPtr é: %p \n", *&aPtr);
    printf("O valor de &*aPtr é: %p \n", &*aPtr);
    return 0;
}
```

Formas de Referenciar um array

```
#include <stdio.h>
int main()
{
     int b[] = {0, 10, 20, 30};
     int *bPtr = b;    int i;      int offset;
     printf("\n Impressão usando índices \n");
     for (i = 0; i <4 ; i++){
            printf("b[%d] = %d\n", i, b[i]);
     }
     printf("\n Impressão usando apontador e offset\n");
     for (offset = 0; offset <4 ; offset++){
            printf("*(b + %d) = %d\n", offset, *(b + offset));
     }
     printf("\n Impressão usando bPtr e índice \n");
     for (i= 0; i <4 ; i++){
            printf("bPtr[ %d] = %d\n", i, bPtr[i] );
     }
     printf("\n Impressão  usando bPtr e offset \n");
     for (offset = 0; offset <4 ; offset++){
            printf("*(bPtr + %d) = %d\n", offset,*(bPtr +offset));
     }
}
```

Chamada de Funções e Apontadores (Ia)

```
#include  <stdio.h>
#include <stdlib.h>
int main()
{
            int i, n, *x;
    void ordena(int n, int *x);
    printf( "\nQuantos números? ");
    scanf("%d", &n);
    printf("\n");
    x = (int *) malloc( n * sizeof(int));
    for (i = 0; i < n; ++i) {
          printf("i =  %d  x = ",  i+1);
          scanf("%d", x + i);
    }
    ordena(n, x);
    printf("\n Lista Ordenada: \n");
    for (i =0; i < n ; ++i)
          printf("i = %d  x = %d \n", i + 1, *(x+ i));
  }
```

Chamada de Funções e Apontadores (Ib)

```
void ordena( int n, int *x)
{
   int i,  k, temp;
   for ( k = 0; k < n -1 ; ++k )
        for (i = k +1; i <  n; ++i)
             if ( *(x + i) < *(x + k)) {
                       temp = *(x + k);
                       *(x+k)= *(x + i);
                       *(x+ i) = temp;
              }

   return;
}
```

Chamada de Funções e Apontadores (II)

```
#include <stdio.h>
#include <ctype.h>
void converteMaiuscula(char *sPtr);
int main()
{
     char string[] =”abc123A”;
     printf(“String inicial: %s\n”, string);
     converteMaiuscula(string);
     printf(“String em maiúscula: %s\n”, string);
     return 0;
}
void converteMaiuscula(char *sPtr)
{
     while(*sPtr != ‘\0’)
     {    if   (islower (*sPtr) {*sPtr = toupper(*sPtr);}
          ++sPtr;
     }
 }
```

Chamada de Funções e Apontadores(III)

```
#include <stdio.h>
void imprime(const char *sPtr);
int main()
{
   char string[] = {'a','b','c','d','e','\0'}
   imprime(string);
   return 0;
}
void imprime(const char *sPtr)
{
   for ( ;*sPtr != '\0' ; sPtr++){
       printf("%c", *sPtr);
   }
}
```

Chamada de Funções e Apontadores(IV)

```
#include <stdio.h>

void f(const int *xPtr);

int main()
{
   int y;
   f(&y);
   return 0;
}

void f(const int *xPtr){
   *xPtr = 100; /*  erro  */
}
```

Chamada de Funções e Apontadores (V)

```
#include <stdio.h>
int main()
{ int x;
  int y;
  int * const ptr = &x;
  *ptr = 7;
  ptr = &y; /*  erro  */
  return 0;
}
#include <stdio.h>
int main()
{ int x =5;
  int y;
  const  int * const ptr = &x;
  printf ("%d \n", *ptr);
  *ptr = 7;        /*  erro  */
  ptr = &y;      /*  erro  */
  return 0;
}
```