P3 – Memória

Ponteiros:

- Ponteiros facilitam a solução de dois problemas: habilita partes diferentes do código a compartilhar rapidamente informações e habilita estrutura de dados complexas como lista e árvores binárias.

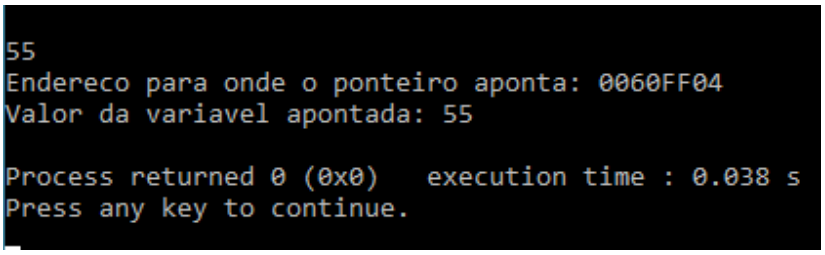
Exemplos:

1) Ponteiro simples:

Código

```c
#include <stdio.h>
int main ()
{
int num,valor;
int *p;
num=55;
p=&num; /* Pega o endereco de num */
valor=*p; /* Valor e igualado a num de uma maneira indireta */
printf ("\n\n%d\n",valor);
printf ("Endereco para onde o ponteiro aponta: %p\n",p);
printf ("Valor da variavel apontada: %d\n",*p);
return(0);
}
```

Saída:



2) Codifique um programa que inicializa variáveis do tipo inteiro. iDado1 e iDado2 devem receber os valores 122 e 312. Declare os ponteiros ptrPonteiro1 e ptrPonteiro2: inicialize ptrPonteiro1, apontando para iDado1 e prtPonteiro2, apontando para iDado2. Faça a troca dos dados de iDado1 e iDado2 usando acesso direto e acesso indireto. Depois, troque os valores dos ponteiros ptrPonteiro1 e ptrPonteiro2 e imprima os valores indiretamente acessados pelos ponteiros.

Código:

```c
#include <stdio.h>
int main(){
int iDado1, iDado2, valor1, valor2;
int *ptrPonteiro1, *ptrPonteiro2;
///OBS.: *prtPonteiro1 = iDado1 e prtPonteiro1 ≠ iDado1
iDado1 = 122;
iDado2= 312;

ptrPonteiro1= &iDado1; ///ptrPonteiro1 recebendo o valor da memória iDado1
ptrPonteiro2= &iDado2; ///ptrPonteiro2 recebendo o valor da memória iDado2


printf("Variavel: %d \n""   Local de memoria: %p \n        O local de memoria aponta para: %d \n", iDado1,ptrPonteiro1,*ptrPonteiro1);
printf("Variavel: %d \n""   Local de memoria: %p \n        O local de memoria aponta para: %d \n", iDado2,ptrPonteiro2,*ptrPonteiro2);


//Indireta
valor1=*ptrPonteiro1; /// valor1 recebendo o valor da variável iDado1
*ptrPonteiro1=*ptrPonteiro2;
*ptrPonteiro2=valor1;
printf("Variavel: %d \n""   Local de memoria: %p \n        O local de memoria aponta para: %d \n", iDado1,ptrPonteiro1,*ptrPonteiro1);
printf("Variavel: %d \n""   Local de memoria: %p \n        O local de memoria aponta para: %d \n", iDado2,ptrPonteiro2,*ptrPonteiro2);

//Direta
valor1  = iDado1;
iDado1 = iDado2;
iDado2 = valor1;
printf("Variavel: %d \n""   Local de memoria: %p \n        O local de memoria aponta para: %d \n", iDado1,ptrPonteiro1,*ptrPonteiro1);
printf("Variavel: %d \n""   Local de memoria: %p \n        O local de memoria aponta para: %d \n", iDado2,ptrPonteiro2,*ptrPonteiro2);

return 0;
}
```

Saída:

```
Variavel: 122
   Local de memoria: 0060FF00
         O local de memoria aponta para: 122
Variavel: 312
   Local de memoria: 0060FEFC
         O local de memoria aponta para: 312
Variavel: 312
   Local de memoria: 0060FF00
         O local de memoria aponta para: 312
Variavel: 122
   Local de memoria: 0060FEFC
         O local de memoria aponta para: 122
Variavel: 122
   Local de memoria: 0060FF00
         O local de memoria aponta para: 122
Variavel: 312
   Local de memoria: 0060FEFC
         O local de memoria aponta para: 312

Process returned 0 (0x0)   execution time : 0.038 s
Press any key to continue.
```

Passagem de parâmetros:

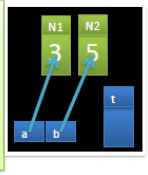
- Há duas maneiras de se realizar uma passagem de parâmetros: por valor ou por referência.
- Passagem de parâmetros **por valor:** a função recebe uma cópia da variável que é fornecida quando é invocada. Todas as alterações feitas dentro da função não vão afetar as variáveis globais:

```
#include<stdio.h>
main()
{
 int N1=3, N2=5;
 troca(N1,N2);
}
```


```
void troca(int a, int b)
{
  int t;
  t=a;
  a=b;
  b=t;
}
```

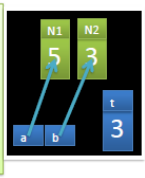
- Passagem de parâmetros **por referência:** a função recebe uma referência da variável (ponteiro) que é fornecida quando invocada. Todas as alterações feitas dentro da função vão afetar as variáveis globais:

```
#include<stdio.h>
main()
{
 int N1=3, N2=5;
 troca(&N1,&N2);
}
```




```
void troca(int *a, int *b)
{
  int t;
  t=*a;
  *a=*b;
  *b=t;
}
```

```
#include<stdio.h>
main()
{
 int N1=3, N2=5;
 troca(&N1,&N2);
}
```




```
void troca(int *a, int *b)
{
  int t;
  t=*a;
  *a=*b;
  *b=t;
}
```

Exemplo: escrever uma função que retorne o quadrado de x, e indiretamente, armazena o cubo de x.

Código:

```c
#include <stdio.h>

double quadradoCubo(double y, double *ptr1, double *ptr2) ///função secundária
{
/// Obs.: onde há x (na função principal), irá se comportar como y
/// Obs.: onde há &cubo (na função principal) , irá se comportar como *ptr1
/// Obs.: onde há &w (na função principal), irá se comportar como *ptr2

    *ptr1=y*y*y;
    y = y*y;
    *ptr2 = y* y;

    return y;
}


int main() ///função principal
{
    double x, quadrado, cubo, w;
    printf("***Retorna os valores direta e indiretamente***\n");

    x=3;

    quadrado = quadradoCubo (x, &cubo, &w); ///invoca a função secundária

    printf("x=%.2f, quadrado=%.2f, cubo=%.2f, quarta pot = %.2f",x, quadrado, cubo, w);
    return 0;
}
```

Saída:

```
***Retorna os valores direta e indiretamente***
x=3.00, quadrado=9.00, cubo=27.00, quarta pot = 81.00
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Alocação Dinâmica:

- Alocação dinâmica de memória: alocar a memória em tempo de execução. Ou seja, durante o programa, define-se o tamanho a ser armazenado (que pode ser dado pelo usuário).

```c
#include <stdio.h>
#include <stdlib.h>
void funcao (int *pPint, int tam){
    int x;
    for (x=0; x<tam; x++)
        scanf ("%d", &pPint[x]);
}

int main(){

int i, *pPint, y;

scanf ("%d", &i);///recebe o valor i
pPint =(int*) malloc(i* sizeof (int)); ///aloca-se a memória (dinâmica) pro pPint com i casas do tipo inteiro

funcao(pPint, i);

    for (y=i-1; y>=0; y--){
        printf("%d ", pPint[y]);
    }

    printf("\n");
    free(pPint); /// libera a memória alocada
    pPint = NULL; /// convenção mesmo, serve pra pn

return 0;
}
```

Structs:

- É uma coleção de informações que podem ter diferentes tipos;

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

/// L05Ex05 - Defini□□o dos tipos tPessoa e tAgenda
typedef struct tPessoa
{
    char nome[51],cpf[12],tel_movel[12];
    int ano_nascimento,id;
}tPessoa;
typedef struct tAgenda
{
    tPessoa *pessoas;
    int qtde,max_pessoa;
}tAgenda;
```