

Estrutura de Dados

Linguagem C++

Estudando...

Ponteiros

- & → Endereço onde uma variável está guardada na memória
- %p → Exibe um endereço

```
int main(int argc, char *argv[]) {  
    int A, B, C;  
  
    A = 100;  
    B = 200;  
    C = 300;  
  
    printf("Endereco de A: %p \n", &A);  
    printf("Endereco de A: %p \n", &B);  
    printf("Endereco de A: %p \n\n", &C);  
  
    printf("Valor de A: %d \n", A);  
    printf("Valor de B: %d \n", B);  
    printf("Valor de C: %d \n", C);  
  
    system("Pause");  
    return 0;  
}
```

Ponteiros

Memória

0xAAFF01	
0xAAFF02	
0xAAFF03	
0xAAFF04	
0xAAFF05	
0xAAFF06	
0xAAFF07	
0xAAFF08	
0xAAFF09	
0xAAFF0A	
0xAAFF0B	
0xAAFF0C	
0xAAFF0D	
0xAAFF0E	
0xAAFF0F	

Ponteiros

0xAAFF05 = 300
0xAAFF06 = 1.70
0xAAFF07 = 2000
0xAAFF08 = 8.5

Memória

0xAAFF01	
0xAAFF02	
0xAAFF03	
0xAAFF04	
0xAAFF05	300
0xAAFF06	1.70
0xAAFF07	2000
0xAAFF08	8.5
0xAAFF09	
0xAAFF0A	
0xAAFF0B	
0xAAFF0C	
0xAAFF0D	
0xAAFF0E	
0xAAFF0F	

Ponteiros

CodAluno = 300
DataNasc = 1.70
Altura = 2000
MediaFinal = 8.5

Memória

0xAAFF01	
0xAAFF02	
0xAAFF03	
0xAAFF04	
0xAAFF05	300
0xAAFF06	1.70
0xAAFF07	2000
0xAAFF08	8.5
0xAAFF09	
0xAAFF0A	
0xAAFF0B	
0xAAFF0C	
0xAAFF0D	
0xAAFF0E	
0xAAFF0F	

Ponteiros

CodAluno = 300
DataNasc = 2000
Altura = 1.70
MediaFinal = 8.5

Memória

0xAAFF01	
0xAAFF02	
0xAAFF03	
0xAAFF04	
0xAAFF05	300
0xAAFF06	2000
0xAAFF07	1.70
0xAAFF08	8.5
0xAAFF09	
0xAAFF0A	
0xAAFF0B	
0xAAFF0C	
0xAAFF0D	
0xAAFF0E	
0xAAFF0F	

Ponteiros

Memória

CodAluno
DataNasc
Altura
MediaFinal



0xAAFF01	
0xAAFF02	
0xAAFF03	
0xAAFF04	
0xAAFF05	300
0xAAFF06	2000
0xAAFF07	1.70
0xAAFF08	8.5
0xAAFF09	
0xAAFF0A	
0xAAFF0B	
0xAAFF0C	
0xAAFF0D	
0xAAFF0E	
0xAAFF0F	

Ponteiros

Memória

int X;
int Y;



0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	10
0xA AFF05	
0xA AFF06	
0xA AFF07	10
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros

Memória

```
int X;  
int Y;
```

```
X = 10;  
Y = X;
```



0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	10
0xA AFF05	
0xA AFF06	
0xA AFF07	10
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros

Memória

```
int X;  
int Y;
```

```
X = 10;  
Y = X;
```

```
X = 500;  
Y = 100;
```

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	500
0xA AFF05	
0xA AFF06	
0xA AFF07	100
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros

Memória

int X;

int Y;



0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	
0xA AFF05	
0xA AFF06	
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros



Ponteiros são um tipo especial de variáveis que permitem armazenar endereços de memória ao invés de dados numéricos (como os tipos **int**, **float** e **double**) ou caracteres (como o tipo **char**).

Ponteiros

- Variável: é um espaço reservado de memória usado para guardar um valor que pode ser modificado pelo programa;
- Ponteiro: é um espaço reservado de memória usado para guardar um endereço de memória.



Na linguagem C, um ponteiro pode ser declarado para qualquer tipo de variável (char, int, float, double, etc), inclusive para aquelas criadas pelo programador (struct, etc).

Ponteiros

Memória

int X; 

int *Y; 

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	10
0xA AFF05	
0xA AFF06	???
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros

Memória

int X;



int *Y;

Y = &X;



0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	10
0xA AFF05	
0xA AFF06	0xA AFF04
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	



Ponteiros

Memória

```
int main(int argc, char *argv[]) {  
  
    int A;  
    int *B;  
  
    A = 100;  
    B = &A;  
  
    printf("Endereco de A: %p \n", &A);  
    printf("Conteudo de A: %d \n\n", A);  
  
    printf("Endereco de B: %p \n", &B);  
    printf("Endereco apontado por B: %p \n", B);  
    printf("Conteudo de B: %d \n\n", *B);  
  
    system("Pause");  
    return 0;  
}
```

0xAAFF01	
0xAAFF02	
0xAAFF03	
0xAAFF04	500
0xAAFF05	
0xAAFF06	0xAAFF04
0xAAFF07	
0xAAFF08	
0xAAFF09	
0xAAFF0A	
0xAAFF0B	
0xAAFF0C	
0xAAFF0D	
0xAAFF0E	
0xAAFF0F	



Ponteiros

Memória

```
int X;
```

```
int *Y;
```

```
X = 10;
```

```
Y = &X;
```



0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	10
0xA AFF05	
0xA AFF06	
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros

Memória

```
int X;
```

```
int *Y;
```

```
X = 10;
```

```
Y = &X;
```

```
*Y = 200
```

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	200
0xA AFF05	
0xA AFF06	
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros

Memória

```
int X;
```

```
int *Y;
```

```
X = 10;
```

```
Y = &X;
```

```
*Y = 200
```

```
X = 500
```



0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	500
0xA AFF05	
0xA AFF06	
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros

Memória

```
int X;
```

```
int *Y;
```

```
X = 10;
```

```
Y = &X;
```

```
*Y = 200
```

```
X = 500
```

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	500
0xA AFF05	
0xA AFF06	0xA AFF04
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	



Declaração Ponteiros

tipo_do_ponteiro *nome_do_ponteiro;



É o operador *asterisco* (*) que informa ao compilador que aquela variável não vai guardar um valor, mas sim um endereço de memória para aquele tipo especificado.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(){
4     //Declara um ponteiro para int
5     int *p;
6     //Declara um ponteiro para float
7     float *x;
8     //Declara um ponteiro para char
9     char *y;
10    //Declara uma variável do tipo int e um
        ponteiro para int
11    int soma, *p2,;
12
13    system( ' 'pause' ' );
14    return 0;
15 }
```

Ponteiros



Apesar de usarem o mesmo símbolo, o operador *
(**multiplicação**) não é o mesmo operador que o *
(**referência de ponteiros**).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(){
4     int x = 3, y = 5, z;
5     z = y * x;
6     int *p;
7
8     system( ' 'pause' ' );
9     return 0;
10 }
```

Inicialização de Ponteiros

int X;



int *Y;



???

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	
0xA AFF05	
0xA AFF06	
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Inicialização de Ponteiros

```
int X;
```



```
// Nenhum lugar na memória
```

```
int *Y = NULL;
```

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	
0xA AFF05	
0xA AFF06	
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Inicialização de Ponteiros

int X;

int *Y;

Y = &X;

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	10
0xA AFF05	
0xA AFF06	0xA AFF04
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	



Ponteiros – Operador * e &

Ao se trabalhar com ponteiros, duas tarefas básicas serão sempre executadas:

- acessar o endereço de memória de uma variável;
- acessar o conteúdo de um endereço de memória;

Para realizar essas tarefas, iremos sempre utilizar apenas dois operadores: o operador “*” e o operador “&”.

Operador “*” versus operador “&”	
“*”	Declara um ponteiro: int *x; Conteúdo para onde o ponteiro aponta: int y = *x;
“&”	Endereço onde uma variável está guardada na memória: &y

Ponteiros

```
int main(int argc, char *argv[]) {  
  
    int A;  
    int *B;  
  
    A = 100;  
    B = &A;  
  
    printf("Conteudo de B: %d \n\n", *B);  
  
    *B = 200;  
  
    printf("Conteudo de B: %d \n\n", *B);  
  
    system("Pause");  
    return 0;  
}
```

Exercício

1. Quais serão os valores de x, y e p ao final do trecho de código abaixo?

```
int x, y, *p; y = 0;
```

```
p = &y;
```

```
x = *p;
```

```
x = 4;
```

```
(*p)++;
```

```
--x;
```

```
(*p) += x;
```

Exercício

1. O método `misterio(&i,&j)` tem um problema. Qual é? Antes da chamada do método, temos a seguinte linha de comando: `int i=6, j=10;`

```
void misterio(int *p, int *q){  
    int *temp;  
    *temp = *p;  
    *p = *q;  
    *q = *temp;  
}
```

2. O que as linhas abaixo fazem?

```
int i=99, j;  
int *p;  
p = &i;  
j = *p + 100;
```

3. O que as linhas abaixo fazem?

```
int a=5, b=12;  
int *p;  
int *q;  
p = &a;  
q = &b;  
int c = *p + *q;
```

Ponteiros - Funções

```
void Alterar(int *B);


int main(int argc, char *argv[]) {

    int A;

    A = 100;

    printf("Conteudo de A: %d \n\n", A);

}
```



0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	100
0xA AFF05	
0xA AFF06	
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros - Funções

```
void Alterar(int *B);

int main(int argc, char *argv[]) {

    int A;

    A = 100;

    printf("Conteudo de A: %d \n\n", A);

    Alterar(&A);

}

void Alterar(int *B) {
```

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	100
0xA AFF05	
0xA AFF06	0xA AFF04
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	



Ponteiros - Funções

```
void Alterar(int *B);

int main(int argc, char *argv[]) {

    int A;

    A = 100;

    printf("Conteudo de A: %d \n\n", A);

    Alterar(&A);

}

void Alterar(int *B) {
    *B = 200;
}
```

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	200
0xA AFF05	
0xA AFF06	0xA AFF04
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Ponteiros - Funções

```
void Alterar(int *B);

int main(int argc, char *argv[]) {

    int A;

    A = 100;

    printf("Conteudo de A: %d \n\n", A);

    Alterar(&A);

    printf("Conteudo de A: %d \n\n", A);

}

void Alterar(int *B) {
    *B = 200;
}
```

0xA AFF01	
0xA AFF02	
0xA AFF03	
0xA AFF04	200
0xA AFF05	
0xA AFF06	0xA AFF04
0xA AFF07	
0xA AFF08	
0xA AFF09	
0xA AFF0A	
0xA AFF0B	
0xA AFF0C	
0xA AFF0D	
0xA AFF0E	
0xA AFF0F	

Exercício

- O que os programas abaixo imprime na tela?
- Tente responder e explicar sua resposta sem executar o programa.

```
void Somar(int x){  
    x = x + 5;  
}  
  
int main(void){  
    int x = 50;  
  
    Somar(x);  
  
    printf("%d \n",x);  
    return 0;  
}
```

```
void Somar(int * x){  
    *x = *x + 5;  
}  
  
int main(void){  
    int x = 50;  
  
    Somar(&x);  
  
    printf("%d \n",x);  
    return 0;  
}
```

Exercício

1. Reescreva o código abaixo utilizando notação de ponteiros dentro do laço for.

```
int x[5], i;  
for (i= 0; i < 5; i++)  
    x[i]= i*10;
```

2. Escreva um programa que leia 5 inteiros e os armazene em um vetor. A partir disso, utilizando notação de ponteiros para navegar no vetor, mostre na tela o maior valor lido.

Aritmética Com Ponteiros

- Apenas duas operações aritméticas podem ser utilizadas nos endereços armazenados pelos ponteiros: adição e subtração.

Aritmética Com Ponteiros

- Tipo **int** equivale a 4 bytes

```
int main(int argc, char *argv[]) {  
    int A;  
    int *B;  
  
    B = &A;  
    printf("B = Hexadecimal: %p \t Decimal: %d \n", B, B);  
  
    B = B + 1;  
    printf("B = Hexadecimal: %p \t Decimal: %d \n", B, B);  
  
    B = B + 1;  
    printf("B = Hexadecimal: %p \t Decimal: %d \n\n", B, B);  
  
    B = B - 1;  
    printf("B = Hexadecimal: %p \t Decimal: %d \n\n\n", B, B);  
  
    system("Pause");    return 0;  
}
```

Operação Relacionais Com Ponteiros

A linguagem C permite comparar os endereços de memória armazenados por dois ponteiros utilizando uma expressão relacional. Por exemplo, os operadores `==` e `!=` são usado para saber se dois ponteiros são iguais ou diferentes.



Dois ponteiros são considerados iguais se eles apontam para a mesma posição de memória.

Operação Relacionais Com Ponteiros

```
int main(int argc, char *argv[]) {  
    int X, Y;  
    int *B, *C;  
  
    B = &X;  
    C = &Y;  
  
    if (B == C)  
    {  
        printf("Os ponteiros apontam para o MESMO lugar. \n\n");  
    }  
    else  
    {  
        printf("Os ponteiros apontam para DIFERENTES lugares.\n\n");  
    }  
  
    system("Pause");  
    return 0;  
}
```


Exercícios

- Escreva um programa que contenha duas variáveis inteiras.
- Compare seus endereços e exiba o maior endereço

Exercícios

```
int main(int argc, char *argv[]) {
    int A, B;
    int *End_A, *End_B;

    A = 20;
    B = 50;

    End_A = &A;
    End_B = &B;

    printf("Endereco de A = %d\nEndereco de B = %d\n\n", &A, &B);
    printf("End ponteiro A = %d\nEnd ponteiro B = %d\n\n", End_A, End_B);

    if (End_A > End_B) {
        printf("\n\nEndereco %d de A eh maior do que de B: \n\n", End_A);
    }
    else {
        printf("\n\nEndereco %d de B eh maior do que de A \n\n", End_B);
    }

    system("Pause"); return 0;
}
```

Ponteiros Genéricos

Normalmente, um ponteiro aponta para um tipo específico de dado. Porém, pode-se criar um ponteiro *genérico*. Esse tipo de ponteiro pode apontar para todos os tipos de dados existentes ou que ainda serão criados.

```
void *nome_do_ponteiro;
```



Sempre que se trabalhar com um ponteiro genérico é preciso converter o ponteiro genérico para o tipo de ponteiro com o qual se deseja trabalhar antes de acessar o seu conteúdo.

Casting

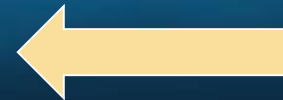
- Efetua a conversão de tipo.

```
int main(int argc, char *argv[]) {  
  
    int Vlr1, Vlr2;  
    float Resultado;  
  
    Vlr1 = 17;  
    Vlr2 = 5;  
  
    Resultado = (float) Vlr1 / Vlr2;  
    printf("Resultado: %.2f\n\n", Resultado);  
  
    system("Pause");  
    return 0;  
}
```

```
int main(int argc, char *argv[]) {  
  
    int Vlr1, Vlr2;  
    float Resultado;  
  
    Vlr1 = 17;  
    Vlr2 = 5;  
  
    Resultado = Vlr1 / Vlr2;  
    printf("Resultado: %.2f\n\n", Resultado);  
  
    system("Pause");  
    return 0;  
}
```

Ponteiros Genéricos

```
int main(int argc, char *argv[]) {  
  
    int A;  
    int *B;  
    void *C;  
  
    A = 100;  
  
    B = &A;  
    printf("Endereco de B: %p \n", &B);  
    printf("Conteudo de B: %d \n\n", *B);  
  
    C = &A;  
    printf("Endereco de C: %p \n", &C);  
    //printf("Conteudo de C: %d \n", *C); //INCORRETO  
    printf("Conteudo de C: %d \n\n", *(int *)C); //CORRETO  
  
    system("Pause");  
    return 0;  
}
```



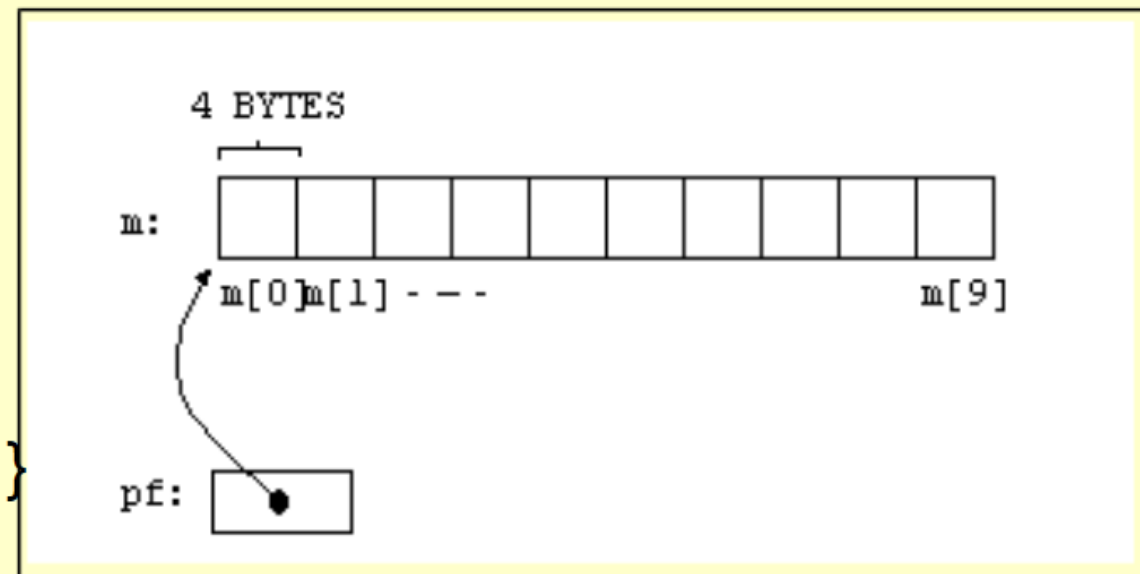
Ponteiros e Vetores

em **float** **m[10]** *m* é uma constante que endereça o primeiro elemento do vetor

portanto, não é possível mudar o valor de *m*

Ex:

```
void main ( ) {  
    float m[10], n[10];  
    float *pf;  
    m = n; //erro: m é constante!  
    pf = m; //ok  
}
```



Ponteiros e Vetores

- O nome do vetor representa seu endereço na memória

```
//PMATRIZ.CPP
//Imprime os valores dos elementos de uma matriz
#include <iostream.h>
void main()
{
    int M[5]={92,81,70,69,58};

    for(int i=0; i<5; i++)
        cout << "\n" << *(M+i);
}
```

Ponteiros e Vetores

```
//PMATRIZ.CPP
//Imprime os valores dos elementos de uma matriz
#include <iostream.h>
void main()
{
    int M[5]={92,81,70,69,58};

    for(int i=0; i<5; i++)
        cout << "\n" << *(M+i);
}
```

A expressão `*(M+i)` tem exatamente o mesmo valor de `M[i]`.

Ponteiros e Vetores

- Ponteiro CONSTANTE

```
cout << "\n" << *(M+i);
```

≠

```
cout << "\n" << *(M++); //ERRADO
```



```
x = 3++; //ERRADO
```

- O nome da matriz é um ponteiro constante

Ponteiros e Vetores

- Ponteiro VARIÁVEL

```
//PMATRIZ.CPP
//Imprime os valores dos elementos de uma matriz
#include <iostream.h>
void main()
{
    int M[5]={92,81,70,69,58};
    int *p=M;

    for(int i=0; i<5; i++)
        cout << "\n" << *(p++);
}
```

Ponteiros e Vetores

- Ponteiro VARIÁVEL

Exemplo: acessando arrays utilizando ponteiros.

Usando Array

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main () {
4     int vet[5] =
5         {1,2,3,4,5};
6     int *p = vet;
7     int i;
8     for (i = 0; i < 5; i++)
9         printf("%d\n", p[i]);
10    system('pause');
11    return 0;
12 }
```

Usando Ponteiro

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main () {
4     int vet[5] =
5         {1,2,3,4,5};
6     int *p = vet;
7     int i;
8     for (i = 0; i < 5; i++)
9         printf("%d\n", *(p+i));
10    system('pause');
11    return 0;
12 }
```

Ponteiros e Vetores

Pode-se referenciar os elementos de um vetor através de ponteiros:

Ex:

```
#include <stdio.h>

void main () {
    float m[] = { 1.0, 3.0, 5.75, 2.345 };
    float *pf;
    pf = &m[2];
    printf("%f", *pf);    /* ==> 5.75 */}
```

```
//PMEDIA.CPP
//Mostra passagem de matrizes para funções como argumento
#include <iostream.h>

int media(int *lista, int tamanho);

void main()
{
    const MAXI=20;
    int notas[MAXI];

    for(int i=0; i<MAXI ; i++)
    {
        cout << "Digite a nota do aluno " <<(i+1)<<": ";
        cin  >> *(notas+i);

        if(*(notas+i) < 0 ) break;
    }

    int m = media(notas,i);
}

int media(int *lista, int tamanho)
{
    int m=0;

    for(int i=0; i < tamanho ; i++) m += *lista++;

    return ( m/tamanho);
}
```

Exercícios

- Escreva uma função que troca o valor de duas variáveis do tipo inteiro. A função deve possuir o seguinte protótipo:

`void swap(int* x, int* y);`

Escreva um programa que imprima um vetor de inteiros na ordem inversa endereçando os elementos com um ponteiro

Exercícios

Escreva uma função que receba três números reais (float) e reordene-os de forma decrescente. Teste sua implementação.

Sugestão para protótipo da função:

```
void ordena(float *a, float *b, float *c);
```


Exercícios

- Elabore um algoritmo que inicialize dois vetores e em seguida passe os para uma função. A função deve realizar a soma de dois vetores, A e B, de números reais e de tamanho 5 no formato de ponteiros.
- Escreva o código fonte de um programa 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 por uma função e, em seguida enviada a outra função para encontrar e retornar o maior valor contido no vetor. Por fim deve ser impresso na tela.

Pensamento

"O pensamento lógico pode levar você de A a B, mas a imaginação te leva a qualquer parte do Universo."

(Albert Einstein)

FIM

Prof. Me Ricardo Luis Balieiro