Algoritmos e Estruturas de Dados – I

Aula 2 - Ponteiros

Ana Cláudia Martinez

Definições

- Variáveis : endereçam uma posição de memória que contem um determinado valor dependendo do seu tipo (char, int, float, double, ...)

```
void main() {
   int a=5;
   float i=10;
   char b='x';
}
```

Endereço	Valor	
1000	5	a
1001		
1002	10	
1003		i
1004		
1005		
1006	x	b

Definições

- Ponteiros: são variáveis cujo conteúdo é um endereço de memória.
  - Assim, um ponteiro endereça uma posição de memória que contém valores que são na verdade endereços para outras posições de memória.

```
void main() {
   int a=5;
   float i=10;
   char b='x';
   int *Ptr =&a;
}
```

Endereço	Valor	
1000	5	a
1001		
1002	10	
1003		i
1004		
1005		
1006	x	b
1007	1000	Ptr
1008	1001	

Operadores de ponteiros

- Para trabalharmos com ponteiro devemos conhecer alguns operadores básicos de ponteiros, são eles:

  & - colocado antes de uma variável qualquer (inclusive ponteiros) retornará não o conteúdo de uma variável, mas sim o endereço de memória onde esta está alocada.

  * - por sua vez, já tem uma função bem mais simples, ou seja, sempre que se colocar um * antes de um ponteiro, isso indicará que estaremos acessando o valor que está no endereço de memória contido em p;

Ponteiros

- Os ponteiros são usados para:
  - Manipular elementos de matrizes;
  - Receber argumentos em funções que necessitem modificar o argumento original;
  - Passar strings de uma função para outra;
  - Criar estruturas de dados complexas, como listas encadeadas e árvores binárias, onde um item contém referências a outro;
  - Alocar e desalocar memória do sistema.

Declaração de Ponteiros

- Para declararmos um ponteiro, basta utilizar o operador *(asterisco) antes do nome da variável.
- Exemplo:

```
int *p;
```

- Ponteiros são tipados, ou seja, devem ter seu tipo declarado e somente podem apontar para variáveis do mesmo tipo.

Operadores para Ponteiros

```
void main() {
   int a=5;
   float i=10;
   char b='x';
   int *aPtr =&a;
   cout<<*aPrt<<endl;
  cout<<aPrt<<endl;
  cout<<&aPrt;
}
```

- O que será impresso na tela?
  - 5
  - 1000
  - 1007

Endereço	Valor	
1000	5	a
1001		
1002	10	i
1003		
1004		
1005		
1006	x	b
1007	1000	Ptr
1008	1001	

Alguns exemplos... (1)

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

Alguns exemplos... (2)

```
#include <stdio.h>
main ()
{
   int num,*p;
   num=55;
   p=&num; /* Pega o endereco de num */
   printf ("Valor inicial: %d\n",num);
   *p=100; /* Muda o valor de num de uma maneira indireta */
   printf ("\nValor final: %d\n",num);
}
```

Operadores para Ponteiros

- Igualando ponteiros:

```
int *p1, *p2;
p1=p2;
```

  - Repare que estamos fazendo com que p1 aponte para o mesmo lugar que p2.

- Fazendo com que a variável apontada por p1 tenha o mesmo conteúdo da variável apontada por p2

```
*p1=*p2;
```

Alguns exemplos... (3)

```
#include <iostream.h>
main ()
{
   int num,*p1, *p2;
   num=55;
   p1=&num; /* Pega o endereco de num */
   p2=p1; /*p2 passa a apontar para o mesmo endereço apontado por p1 */
   cout<<"Conteúdo de p1: "<<p1<<endl;
   cout<<"Valor apontado por p1: "<<*p1<<endl;
   cout<<"Conteúdo de p2: "<<p2<<endl;
   cout<<"Valor apontado por p2: "<<*p2<<endl;
}
```

Alguns exemplos... (4)

```
#include <iostream.h>
main ()
{
   int num,*p1, *p2;
   num=55;
   p1=&num; /* Pega o endereco de num */
   *p2=*p1; /* p2 recebe o valor apontado por p1 */
   cout<<"Conteúdo de p1: "<<p1<<endl;
   cout<<"Valor apontado por p1: "<<*p1<<endl;
   cout<<"Conteúdo de p2: "<<p2<<endl;//  erro
   cout<<"Valor apontado por p2: "<<*p2<<endl; // erro
}
```

Operadores para Ponteiros

- Incremento/Decremento:
  - Apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta:

```
int *aPtr, a=5, b,c,d;
aPtr=&a;
aPtr++;
```

Endereço	Valor	
1000	5	a
1001		
1002	10	b
1003		
1004	15	c
1005		
1006	20	d
1007		
1008	1000	aPtr
1009	1001	

Operadores para Ponteiros

- Qual será o valor endereçado por aPtr++ ??
  - Se aPtr é int, como o int ocupa 2 bytes, aPtr irá apontar para o endereço 1002
  - Este é o principal motivo que nos obriga a definir um tipo para um ponteiro!!!

Endereço	Valor	
1000	5	a
1001		
1002	10	b
1003		
1004	15	c
1005		
1006	20	d
1007		
1008	1002	aPtr
1009	1003	

Alguns exemplos... (5)

```
#include <iostream.h>
main ()
{
   int num;
   int *p;
   num=55;
   p=&num;
   cout<<"Conteúdo de p: "<<p<<endl;
   cout<<"Valor apontado por p: "<<*p<<endl;
   ++(*p);
   cout<<"Valor apontado por p incrementado: "<<*p<<endl;
   cout<<"Conteúdo de num: "<<num<<endl;
}
```

Vetores como ponteiros

- O C enxerga vetores como ponteiros
- Quando declaramos um vetor, o C aloca memória para todas as posições necessárias conforme seu tipo:
  - int vet[10];
- O nome do vetor pode ser atribuído a um ponteiro. Neste caso o ponteiro irá endereçar a posição 0 do vetor:
  - int *p; p=vet; ou
  - int *p; p=&vet[0];

Alguns exemplos... (7)

```
    main ()
    {
      int vet [4];
      int *p;
      p=vet;
      for (int count=0;count<4;count++)
      {
        *p=0;
        p++;
      }

  for (int i=0;i<4;i++)
    cout<<vet[i]<<" - ";
}
```

Vetores como ponteiros

- **Importante**: um ponteiro é uma variável, mas o nome de um vetor não é uma variável
- Isto significa, que não se consegue alterar o endereço que é apontado pelo "nome do vetor"
- Diz-se que um vetor é um ponteiro constante!
- Condições inválidas:

```
int vet[10], *p;
vet++;
vet = p;
```

Ponteiros como vetores

- Quando um ponteiro está endereçando um vetor, podemos utilizar a indexação também com os ponteiros:
- Exemplo:

```
int vetor [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int *p;
p=vetor;
cout<<"O terceiro elemento do vetor e: "<<p[2];
```

- Neste caso p[2] equivale a *(p+2)

Porque inicializar ponteiros?

- Observe o código:

```
main () /* Errado - Não Execute */
{
  int x,*p;
  x=13;
  *p=x; //posição de memória de p é indefinida!
}
```

- A não inicialização de ponteiros pode fazer com que ele esteja alocando um espaço de memória utilizado, por exemplo, pelo S.O.

Porque inicializar ponteiros?

- No caso de vetores, é necessário sempre alocar a memória necessária para compor as posições do vetor.
- O exemplo abaixo apresenta um programa que compila, porém poderá ocasionar sérios problemas na execução. Como por exemplo utilizar um espaço de memória alocado para outra aplicação.

```
main() {
  char *pc; char str[] = "Uma string";
  strcpy(pc, str);// pc indefinido
}
```

Exercícios

01 - Considere a seguinte estrutura de memória e um ponteiro p para inteiro situado na posição de memória 3998 apontando para o endereço 4000.

4000	4001	4002	4003	4004
20		60		*NULL*

a. qual o valor de p?
b. qual o valor de *p?
c. qual o valor de &p?
d. qual o valor de *(p+1)?
e. qual o valor de (p+2)?
f. qual o valor de *(p+2)?

Exercícios

01 - Considere a seguinte estrutura de memória e um ponteiro p para inteiro situado na posição de memória 3998 apontando para o endereço 4000.

4000	4001	4002	4003	4004
20		60		*NULL*

a. qual o valor de p? 4000
b. qual o valor de *p? 20
c. qual o valor de &p? 3998
d. qual o valor de *(p+1)? 60
e. qual o valor de (p+2)? 4004
f. qual o valor de *(p+2)? NULL

Exercícios

02 - Qual o valor de y no final do programa? Tente primeiro descobrir e depois verifique no computador o resultado.

```
int main()
{
    int y, *p, x;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x++;
    (*p) += x;
    cout << y ;
    return(0);
}
```

Exercícios

02 - Qual o valor de y no final do programa? Tente primeiro descobrir e depois verifique no computador o resultado.

```
int main()
{
    int y, *p, x;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x++;
    (*p) += x;
    cout << y ;
    return(0);
}
```

VALOR 6

Exercícios

03 - Explique a diferença entre p++; (*p)++;

- p++;
- (*p)++;

Exercícios

03 - Explique a diferença entre p++; (*p)++;

- p++; //Incrementa o endereço de p
- (*p)++; //Incrementa o valor da variável apontada por p

Exercícios – Para implementação

- 01 - Faça um programa que tenha um vetor com 10 posições e um ponteiro que aponta para este vetor. Preencha este vetor com valores de 1 a 10 pelo ponteiro.
- 02 – Faça um programa que tenha um vetor com 50 posições e um ponteiro que aponte para este vetor. Mostre o contéudo pelo ponteiro da terceira posição do vetor.