INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

ALGORITMOS II

Prof. Adilso Nunes de Souza

PONTEIRO

- Ponteiro é um endereço de memória.
- Indica em que parte da memória do computador uma variável está alocada, não o que está armazenado.
- Proporciona um modo de acesso a uma variável sem referi-la diretamente.
- Ponteiros são utilizados em situações em que o uso do nome de uma variável não é permitido, impossível ou indesejável.

PONTEIRO

- **Por que usar ponteiros:**

- Manipular elementos de array
- Receber argumentos em funções que necessitem modificar o valor original.
- Manipular estruturas de dados complexas, como listas encadeadas, árvores, grafos, em que um item deve conter referência a outro.
- Alocar memória dinamicamente
- Manipular string de uma função para outra.
- Manipular referências de endereços de memória entre funções.

PONTEIRO

- Seu valor indica ***onde*** uma variável está armazenada, não ***o que*** está armazenado.
- Para manipular ponteiros utilizamos o operador indireto (*) asterisco.
- Declarando uma variável do tipo ponteiro:

```
int *p;  //ponteiro int
int*  p; //ponteiro int
char *p;  //ponteiro char
int *p, *p1, *p2;  //vários ponteiros int
```

PONTEIRO

- Declarando uma variável do tipo ponteiro:

```
string *s1;  //ponteiro string

int *p = &x;

//declara o ponteiro e já define seu endereço, ou seja, p aponta para o endereço de x


int x, *p = &x, i;  //variáveis e ponteiro já endereçado
```

OBS: Neste caso observar a ordem de criação, pois se o ponteiro recebe o endereço de X então X deve ser criado primeiro.

PONTEIRO

- Todo o ponteiro após ter sido definido deve indicar a qual endereço de memória ele se refere. Para isso utiliza-se o “&”.

```
int *px, x;
x = 5;
px = &x;
```

- Desta forma pode-se dizer que:

  px recebe o endereço de x

  px aponta para o endereço em memória de x

PONTEIRO

- O operador “*” é complementar ao operador “&”, ele é um operador unário, que retorna ou manipula o valor da variável localizada no endereço apontado pelo ponteiro.
- Podemos utilizar o operador “*” para manipular a variável apontada pelo ponteiro (valor contido no endereço).

```
*px += 7; //isto modifica o valor da variável x
```

PONTEIRO

- Como todas as variáveis, os ponteiros têm um endereço e um valor.
- O operador &, precedendo o nome do ponteiro, resulta na posição de memória onde o ponteiro está localizado.
- O nome do ponteiro indica o valor contido nele, isto é, o endereço para o qual ele aponta.

PONTEIRO

- Exemplo:

```
int x = 5, *p = &x;
cout << "Endereço da variável x " << &x << endl;
cout << "Endereço do ponteiro p " << &p << endl;
cout << "Endereço para onde o ponteiro aponta: " << p << endl;
cout << "Conteúdo da variável x " << *p << endl;
```

Saída:

Endereço da variável x **0x61fe1c**

Endereço do ponteiro p **0x61fe10**

Endereço para onde o ponteiro aponta: **0x61fe1c**

Conteúdo da variável x 5

PONTEIRO

- Exibindo um ponteiro:

```
cout << "px = " << px;
    //mostra o endereço de memória para onde o ponteiro aponta
cout << "px = " << *px;
    //mostra o valor no endereço apontado
cout << "px = " << &px;
    //mostra o endereço de memória do ponteiro
cout << "px = " << **&px;
    //mostra o valor no endereço apontado
```

PONTEIRO

- A primeira coisa a ter em mente é que um ponteiro não está apontando para nenhum lugar até que atribuímos a ele o endereço de uma outra variável ou alocamos um endereço em memória para ele.
- Um programa entra em colapso absoluto se tentarmos acessar um ponteiro que aponta para um local de memória que já foi liberado novamente ao sistema.
- No caso menos grave, estaremos tentando acessar locais de memória inválidos ou reservados a outros programas ou tarefas do sistema operacional.

PONTEIRO

- Passagem de ponteiros para função:
  - Utilizando ponteiros é possível alterarmos valores de variáveis através de seus endereços, por exemplo: uma função não pode alterar diretamente valores de variáveis da função que fez a chamada. No entanto, se passarmos para uma função os valores dos endereços de memória onde suas variáveis estão armazenadas, a função pode indiretamente alterar os valores das variáveis.

PONTEIRO

```
void leitura()
{
  int n1, n2;
  cout << "Digite o valor 1:";
  cin >> n1;
  fflush(stdin);

  cout << "Digite o valor 2:";
  cin >> n2;
  fflush(stdin);

  ajuste(&n1, &n2);

  cout << "\n\nVALORES NA FUNCAO LEITURA: " << endl;
  cout << "N1: " << n1 << endl;
  cout << "N2: " << n2 << endl;
}
```

```
void ajuste(int *x, int *i)
{
  *x += 4;
  *i -= 2;
  cout << "\n\nVALORES NA FUNCAO AJUSTE: \n";
  cout << "X: " << *x << endl;
  cout << "I: " << *i << endl;
}
```

- **Os ponteiros X e I manipulam o endereço das variáveis N1 e N2**

PONTEIRO

- Ponteiro com struct:
  - Da mesma forma que podemos definir um ponteiro para uma variável “comum” podemos definir ponteiros para variáveis de tipos diferenciados, como é o caso das struct.
  - Para manipular uma variável do tipo struct o ponteiro deve ser definido do mesmo tipo;
  - Ao utilizar o ponteiro para referenciar as variáveis da struct o mesmo deve estar entre parênteses ou usar o apontador ->

PONTEIRO

- Ponteiro com struct

```
struct dados
{
    int numero;
    char nome[50];
};

main()
{
   dados cadastro;
   leitura(&cadastro);
   cout << "Numero informado: " << cadastro.numero;
   getchar();
}
```

```
void leitura(dados *x)
{
   cout << "informe o numero: ";
   cin >> x->numero;
   //cin >> (*x).numero;
   fflush(stdin);
}
```

PONTEIRO

- Manipulação de array com ponteiro
  - Um vetor é uma variável que é capaz de armazenar N vezes um determinado tipo.
  - Em se tratando de vetores, o nome do vetor representa o endereço de memória onde se inicia o armazenamento do vetor.
  - Portanto, em C, quando se refere a um vetor, sem especificar seu índice (somente com o nome da variável), se está referenciando o ENDEREÇO DE MEMÓRIA da 1ª posição de um vetor.

PONTEIRO

- Manipulação de array com ponteiro
  - Assim, em C

    &vetor[0] é o mesmo que vetor
  - Assim, como inicializo um ponteiro para um vetor ?

    Ex.:

```
int vetor[] = {0,1,2,4,6};
int  *pVetor;
pVetor = &vetor[0];
    OU
pVetor = vetor;
```

  - Por que não é preciso que o ponteiro seja um vetor?

PONTEIRO

- Manipulação de array com ponteiro
  - Pois apontando para o início da memória do vetor, posso percorrê-lo até o seu final indicando o DESLOCAMENTO (posição do vetor) que se quer acessar
  - Ex.: no ponteiro pVetor, acessando-se pVetor[2], se estará acessando a terceira posição do vetor.
  - No acesso ao valor contido na posição apontada, NÃO SE USA O CARACTERE * precedendo a variável.
  - Pois o uso do índice já indica o conteúdo partindo de um deslocamento.

PONTEIRO

- ## Manipulação de array com ponteiro

```
main()
{
  int vet[5];
  leitura(vet);
  mostra(vet);
  getchar();
}

void leitura(int *valores)
{
  int x;
  for(x = 0; x < 5; x++)
  {
    cout << "Informe o valor da posicao [" << x + 1 << "]: ";
    cin >> valores[x];
    fflush(stdin);
  }
}
```

```
void mostra(int *valores)
{
  int x;
  for(x = 0; x < 5; x++)
  {
    cout << valores[x] << ", ";
  }
}
```

PONTEIRO

- Manipulação de array com ponteiro
  - Como o nome do vetor sozinho, indica o endereço de memória onde ele inicia, é possível utilizar a sintaxe de ponteiro par percorrer o vetor. Exemplo:

```
int vet[5]={9, 3, 2, 6, 25};
for(int i = 0; i < 5; i++)
        cout << *(vet + i) << ", ";
```

PONTEIRO

- Manipulação de array com ponteiro
  - A expressão *(vet + i) tem exatamente o mesmo valor que vet[i], pois se vet é um ponteiro int e aponta para vet[0], assim, se somarmos 1 a vet, obteremos o endereço vet[1] e assim sucessivamente.
  - Abordaremos mais sobre este tópico na aula sobre aritmética de ponteiro.

REFERÊNCIAS

- ARAÚJO, Jáiro – Dominando a Linguagem C. Editora Ciência Moderna.
- PEREIRA, Silvio do Lago. Estrutura de Dados Fundamentais: Conceitos e Aplicações, 12. Ed. São Paulo, Érica, 2008.
- LORENZI, Fabiana. MATTOS, Patrícia Noll de. CARVALHO, Tanisi Pereira de. Estrutura de Dados. São Paulo: Ed. Thomson Learning, 2007.
- VELOSO, Paulo. SANTOS, Celso dos. AZEVEDO, Paulo. FURTADO, Antonio. Estrutura de dados. Rio de Janeiro: Ed. Elsevier, 1983 27ª reimpressão.