Aula – Programação II (INF 09330)

UFES
Informática

Ponteiros

Prof. Thiago Oliveira dos Santos
Departamento de Informática
Universidade Federal do Espírito Santo

2015

Visão Geral da Aula

UFES
Informática

- Introdução
- Definição
- Utilização de ponteiros
- Expressões com ponteiros
- Problemas com ponteiros

Introdução

UFES
Informática

Características

- Ponteiro é uma ferramenta muito poderosa
  - Permite acesso as posições de memória
- C é altamente baseada em ponteiros
- Dominar ponteiros é importante para dominar C
- Uso descuidado de ponteiros é perigoso

Principais Razões para Uso de Ponteiros

- Permitem passagem argumentos por referencia em C
- Aumentam eficiência em certos casos
- Permitem alocação dinâmica (Próximo semestre)

Definição

UFES
Informática

O que são ponteiros?

- São variáveis especiais
  - Armazenam endereço para algo com um tipo definido
- Não alocam memória para o dado
  - Alocam memória para o endereço do dado
- Variáveis ponteiro possuem tipo
  - Ponteiro para int é diferente de ponteiro para float

Utilização de Ponteiros

UFES
Informática

Operadores

- Existem dois operadores especiais
  - * e &
  - Significado deles dependem do contexto em que são utilizados

Declaração de Variáveis

- Uma variável do tipo ponteiro é declarada com o operador *
- Exemplo

```
int * iP;
float * fP;
char * cP;
```

Utilização de Ponteiros

UFES
Informática

Inicialização

- É extremamente importante inicializar os ponteiros
  - Valor inicial é lixo
  - Pode apontar para algum lugar inacessível
  - Uso sem inicialização causa problemas
  - Inicialização com zero (NULL) também causa problemas
    - Mas é recomendada quando não se sabe o valor
- Inicialização requer uma posição válida de memória
  - Posição de memória deve ser do tipo apontado pelo ponteiro
  - Para obter o endereço de uma posição de memória usa-se &
  - Exemplo

```
int i;
int * iP = &i;
```

Utilização de Ponteiros

UFES
Informática

Atribuição

- Ponteiros armazenam endereços
  - Endereços podem vir de
    - Variáveis do tipo apontado (usando &)
    - Conteúdo de variáveis do tipo ponteiro
    - Exemplos

```
int i = 10;
int * iP = &i;
int * qP;
qP = iP;
```

Utilização de Ponteiros

UFES
Informática

Acessando Valores

- Ponteiros armazenam endereços que apontam para um dado
- É possível acessar o valor do dado usando-se
  - Operador de derreferenciamento *
  - Exemplo

```
int i = 10;
int * iP = &i;
printf("%d", *iP);
```

- Observar que
  - Impressão sem o operador * acessa o endereço da memória
  - Para imprimir ponteiros usa-se %p
  - Exemplo

```
int i = 10;
int * iP = &i;
printf("%d %p", iP, iP);
```

Utilização de Ponteiros

UFES
Informática

Alterando Valores

- Dados apontados pelos ponteiros podem ser modificados
- Exemplo

```
int i = 10;
int * iP = &i;
*iP = 20;
printf("%d %d", i, *iP);
```

Utilização de Ponteiros

UFES
Informática

Aritmética de Ponteiros

- Modifica o valor da posição de memória apontada
- Considera o tamanho do tipo apontado
- Somar 1 ao ponteiro é equivalente a
  - Apontar para a próxima posição de memória com o tipo apontado
- Exemplo

```
char * p = "ola";

while (*p) {
    printf("%c", *p);
    p = p + 1;
}
```

Utilização de Ponteiros

UFES
Informática

Ponteiro para Ponteiro

- Funciona da mesma forma que o ponteiro
- Variável do tipo ponteiro que aponta para um tipo ponteiro
- Declaração é feita usando-se **
- O operador pode ser usado quantas vezes for necessário
- Exemplo

```
int i = 10;
int * iP = &i;
int ** iPP = &iP;
printf("%d %d %d ", i, *iP , **iPP);
```

Utilização de Ponteiros

UFES
Informática

Atenção

- O operador * possui diferentes significados
  - Declaração do tipo ponteiro
  - Multiplicação
  - Acesso do dado apontado pelo ponteiro
  - Exemplo

```
int i = 10;
int * iP = &i;
i = i * *iP;
printf("%d", i);
```

Utilização de Ponteiros

UFES
Informática

Atenção

- O operador & possui diferentes significados
  - Obtenção do endereço de uma variável
  - “E” lógico
  - “E” lógico bit a bit (não visto no curso)

Utilização de Ponteiros

UFES
Informática

Tipos de Passagens de Parâmetros

- Por valor
- Por referência

Por Valor

- Copia o valor do argumento para a sub-rotina
  - Parâmetros alterados dentro da sub-rotina
    - Não alteram o valor do argumento fora da sub-rotina

Por Referência

- Copia o endereço do argumento para a sub-rotina
  - Endereço referencia o argumento de fora da sub-rotina
  - Parâmetros alterados dentro da sub-rotina
    - Alteram o valor do argumento fora da sub-rotina

Utilização de Ponteiros

UFES
Informática

Passagens de Parâmetros por Referência

- Permite retornar mais de um valor por função
- Torna passagem de dados complexos mais eficiente
  - Ao invés de copiar tudo da estrutura
  - Copia somente o endereço

Utilização de Ponteiros

UFES
Informática

Exemplo de Passagens de Parâmetros

```
void PassagemPorValor(int num){
    num++;
    printf("Num dentro: %d\n", num);
}
void PassagemPorReferencia(int *num){
    (*num)++;
    printf("Num dentro: %d\n", *num);
}
int main()
{
   int x = 0;
   PassagemPorValor(x);
   printf("Num fora por valor: %d\n", x);
   PassagemPorReferencia(&x);
   printf("Num fora por referencia: %d\n", x);
   return 0;
}
```

Utilização de Ponteiros

UFES
Informática

Exemplo de Passagens de Parâmetros

```
void LeIntervalo(int * a, int * b){
   printf("Informe o inicio:");
   scanf("%d", a);
   printf("Informe o fim:");
   scanf("%d", b);
}

int main()
{
   int i, m, n;
   LeIntervalo(&m, &n);

   for(i = m; i <= n; i++){
      printf("%d ", i);
   }
   printf("\n");

   return 0;
}
```

Ponteiros para Estruturas

UFES
Informática

- Permite passar estruturas por referencia para funções
  - Evita alocação desnecessária de memória para estruturas grandes
  - Evitar estourar a memoria da pilha de funções
  - Mais eficiente
  - Permite implementar um novo tipo sem alocações desnecessárias

Ponteiros para Estruturas

UFES
Informática

- Funciona como para variáveis normais
  - Operador “->” permite acesso direto aos atributos

```
#include <stdio.h>
#include <stdlib.h>
typedef struct{
   int x;
   int y;
} tPonto;

int main(){

   tPonto p = {1, 2};
   tPonto * pp = &p;

   printf("(%d %d)\n", (*pp).x, (*pp).y);
   printf("(%d %d)\n", pp->x, pp->y);

   return 0;
}
```

Ponteiros para Estruturas

UFES
Informática

- Permite passar estruturas por referencia para funções
  - Exemplo de algumas funções do tipo tData
    - OBS: tData é usado como exemplo mesmo sendo estrutura pequena

```
typedef struct {
   int dia;
   int mes;
   int ano;
} tData;

void InicializaDataParam( tData *data, int a_dia, int a_mes, int a_ano);

void LeData( tData *data );

void ImprimeData( tData *a_data );
```

Ponteiros para Estruturas

UFES
Informática

```
void InicializaDataParam( tData *data, int a_dia, int a_mes, int a_ano)
{
   int qtdDiasNoMes;
   data->ano = a_ano;

   if (a_mes > 12){
      a_mes = 12;
   } else if (a_mes < 1){
      a_mes = 1;
   }
   data->mes = a_mes;
   qtdDiasNoMes = InformaQtdDiasNoMesMA(a_mes, a_ano);
   if ( a_dia > qtdDiasNoMes ){
      a_dia = qtdDiasNoMes;
   } else if (a_dia < 1){
      a_dia = 1;
   }
   data->dia = a_dia;
}
```

Ponteiros para Estruturas

UFES
Informática

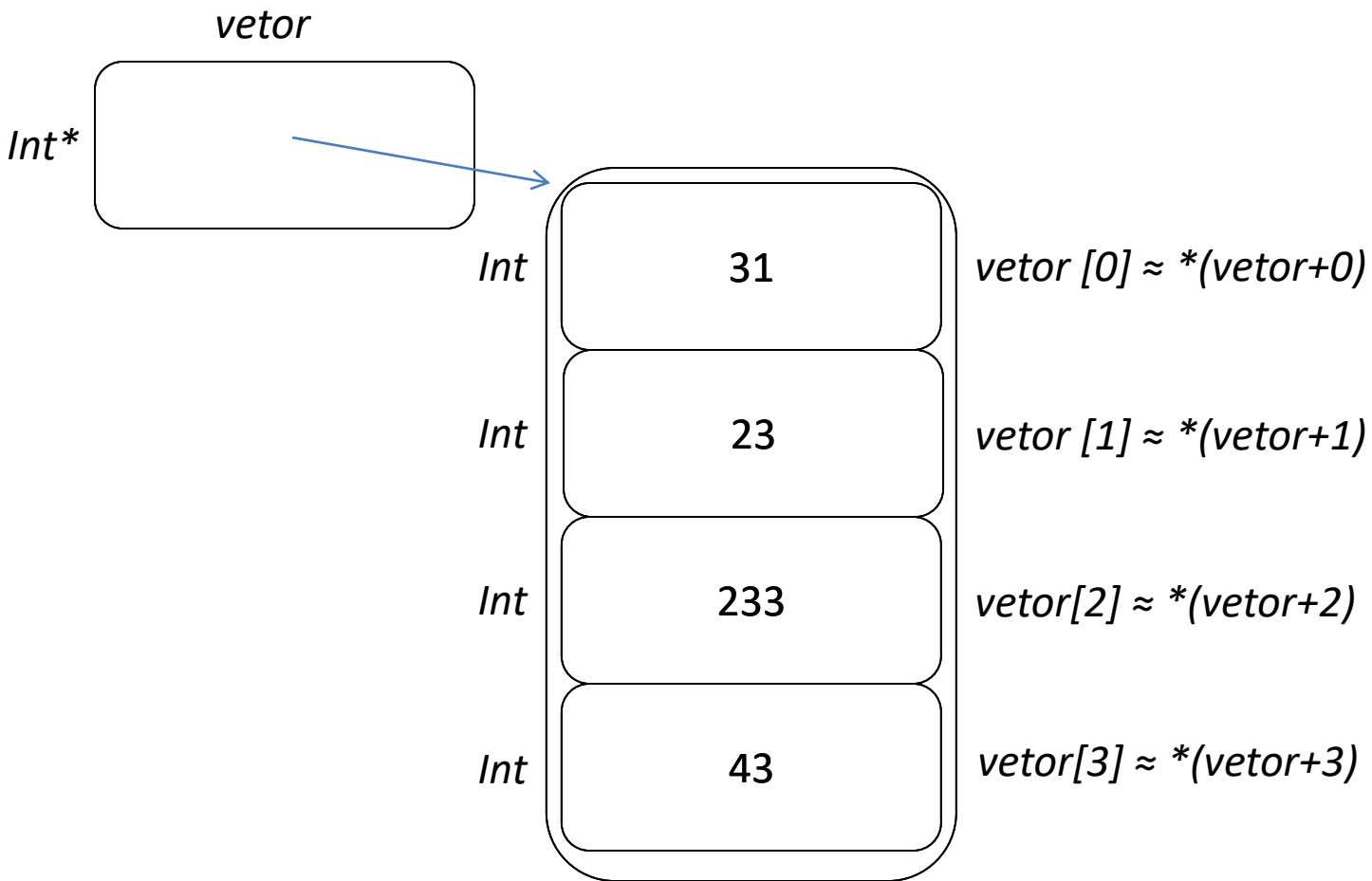
```
void LeData( tData *data )
{
   int d, m, a;
   //Le a data do teclado
   //OBS:Se nao for para o BOCA, pode ter um printf pedindo para digitar a data
   if ( scanf("%d %d %d", &d, &m, &a) != 3 ){
      printf("ERRO: Formato de entrada nao compativel!\n");
      exit(1);
   }

   //Inicializa a data e garante q eh vailida
   InicializaDataParam(data, d, m, a );
}
```

Variáveis Indexadas com Ponteiros

UFES
Informática

- *int vetor[4]*



Variáveis Indexadas com Ponteiros

UFES
Informática

- Identificador da variável é do tipo ponteiro
- Exemplo

```
int i;
float vetor[6] = { 1.3, 4.5, 2.7, 4.1, 0.0, 100.1 };
float *ponteiro = vetor;
for (i = 0; i < 6; i++) {
    printf("%.1f\n", *ponteiro);
    ponteiro = ponteiro + 1;
}
```

Variáveis Indexadas com Ponteiros

UFES
Informática

- O operador de acesso *[idx]* é equivalente
  - Ao apontado pelo endereço do identificador mais *idx*

```
int i;
float vetor[6] = { 1.3, 4.5, 2.7, 4.1, 0.0, 100.1 };
float *ponteiro = &vetor[0];

for (i = 0; i < 6; i++)
   printf("%.1f\n", ponteiro[i] );

printf("\n");

for (i = 0; i < 6; i++)
   printf("%.1f\n", *(vetor+i) );
```

Vetores como Argumento de Função

UFES
Informática

- Permite abstrair e encapsular a manipulação dos vetores
- Permite criar tipos mais complexos
  - Para manipular vetores de estruturas

Vetores como Argumento de Função

UFES
Informática

- void func(int* vet);

```
void PreencheVetor(int *v, int tam){
    int i;
    printf("Informe %d numeros inteiros:\n", tam);
    for (i = 0; i < tam; i++)
        scanf("%d", &v[i]);
}
void ImprimeOrdemInversa(int *v, int tam){
    int i;
    printf("Ordem inversa:\n");
    for (i = tam-1; i >= 0; i--) {
        printf("%d\n", v[i]);
    }
}
```

Vetores como Argumento de Função

UFES
Informática

- Exemplo de um tipo tDatas
  - Encapsula a manipulação de vetores de datas

```
//Acha a menor data em um intervalo [a,b] de um vetor de datas
int AcharMenorEntreAeB(tData * vet, int a, int b);

//Ordena um vetor de datas em ordem crescente
void OrdeneDatasCrescente(tData * vet, int qtd);

//Apresenta valores de um vetor de datas
void ApresentaDatas(tData * vet, int qtd);
```

Vetores como Argumento de Função

UFES
Informática

- Exemplo de um tipo tDatas
  - Uso do tDatas no main

```
tData datas[TAM];

//Le e apresenta as datas
LeData(&datas[0]);
LeData(&datas[1]);
LeData(&datas[2]);
LeData(&datas[3]);
LeData(&datas[4]);
ApresentaDatas(datas, TAM);

//Ordena e apresenta as datas
OrdeneDatasCrescente(datas, TAM);
ApresentaDatas(datas, TAM);
```

Vetores de Ponteiros

UFES
Informática

- Evita ter que copiar itens muito grandes na memoria
  - Por exemplo para fazer uma ordenação de um vetor de estruturas
  - Exemplo com o tipo tData (apesar de tData ser pequena)

```
//Acha a maior data em um intervalo [a,b] de um vetor de ponteiros p datas
int AcharMaiorEntrePAePB(tData ** vet, int a, int b);

//Ordena um vetor de ponteiros p datas
void OrdenePDatasDecrescente(tData ** vet, int qtd);

//Apresenta valores de um vetor de ponteiros p datas
void ApresentaPDatas(tData ** vet, int qtd);

//Copia os enderecos das datas de um vetor de datas p um vetor de ponteiros p
datas
int IniciaVetorPDatas(tData ** vetOut, tData * vetIn, int qtd);
```

Vetores de Ponteiros

UFES
Informática

- Exemplo de um tipo tDatas
  - Uso do vetor de ponteiros do tDatas no main

```
tData datas[TAM];
LeData(&datas[0]);
LeData(&datas[1]);
LeData(&datas[2]);
LeData(&datas[3]);
LeData(&datas[4]);

ApresentaDatas(datas, TAM);
OrdeneDatasCrescente(datas, TAM);
ApresentaDatas(datas, TAM);

tData* datasP[TAM];
IniciaVetorPDatas(datasP, datas, TAM);
OrdenePDatasDecrescente(datasP, TAM);
ApresentaPDatas(datasP, TAM);
ApresentaDatas(datas, TAM);
```

Vetores de Ponteiros

UFES
Informática

```
void ApresentaPDatas(tData ** vet, int qtd){
   int i;

   for(i = 0; i < qtd; i++)
   {
      ImprimeData( vet[i] );
      printf("\n");
   }
   printf("\n");
}

int IniciaVetorPDatas(tData ** vetOut, tData * vetIn, int qtd){
   int i;

   for(i = 0; i < qtd; i++){
      vetOut[i] = &(vetIn[i]);
   }
}
```

Vetores de Ponteiros

UFES
Informática

```
void OrdenePDatasDecrescente(tData ** vet, int qtd){
    int i, idxMaior;
    tData *aux;

    for(i = 0; i < qtd-1; i++)
    {
      idxMaior = AcharMaiorEntrePAePB(vet, i+1, qtd-1);
      if ( EhMenorDataAqDataB(vet[i], vet[idxMaior]) ){
        aux = vet[idxMaior];
        vet[idxMaior] = vet[i];
        vet[i] = aux;
      }
    }

}
```

Vetores de Ponteiros

UFES
Informática

```
int AcharMaiorEntrePAePB(tData ** vet, int a, int b){
    int i, idx;
    tData *maior;

    maior = vet[a];
    idx = a;
    for(i = a+1; i <= b; i++)
    {
        if ( EhMenorDataAqDataB(maior, vet[i]) ){
            maior = vet[i];
            idx = i;
        }
    }
    return idx;
}
```

Problemas com a Utilização de Ponteiros

UFES
Informática

- Ponteiros não inicializados
  - Conteúdo inicial do ponteiro é lixo
    - Faltou garantir que a variável apontasse algo válido
- Programação macarrônica
  - Muitos referências para uma mesma região de memória
  - Dificulta leitura do código e aumenta chances de erros

Perguntas???

UFES
Informática