Estruturas de Dados I

Estruturas de Dados - Structs

Prof. Dr. Lidio Mauro Lima de Campos

limadecampos@gmail.com

Universidade Federal do Pará – UFPA
ICEN

Agenda

- Tipo estrutura.
- Definição de novos tipos.
- Aninhamento de estruturas.
- Vetores de estruturas.

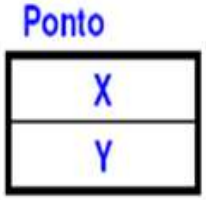
Introdução

- Normalmente trabalhamos com tipos básicos disponibilizados pela linguagem C: char, int, float, double.
- Desenvolvimento de Programas mais complexos, precisa-se trabalhar de uma maneira mais abstrata para representar dados.

  - **Ex:Dados Compostos por diversas informações.**
    - Ponto no espaço bidimensional
    - **Abstração** : Ponto é representado pelas coordenadas (x,y);
      - C oferece mecanismos para agrupar as duas coordenadas (x,y) no mesmo contexto, para que seja possível tratar o ponto como um único objeto.

Motivação

- **Manipulação de dados compostos ou estruturados.**
- **Exemplo 1:** Ponto no espaço Bidimensional.



- Representado por duas coordenadas (x,y).
- Mas tratado como um único objeto ou tipo.

Motivação

- **Manipulação de dados compostos ou estruturados.**
- **Exemplo 2:** Dados Associado a um aluno.

- Aluno representado pelo seu nome, número de matrícula, endereço.
- Estruturados em um único objeto ou (tipo)

Definição em C

- Um tipo estruturado é definido como segue:

```
struct <nome_var>
{
 <lista_tipos_simples>
};
```

- Podemos agrupar os dados de um ponto (x,y) da seguinte forma:

```
struct ponto
{
  float x;
  float y;
};
```

Definição em C

- A definição de uma variável é de forma usual.

```
struct ponto
{
 float x;
 float y;
};

struct ponto p;
```

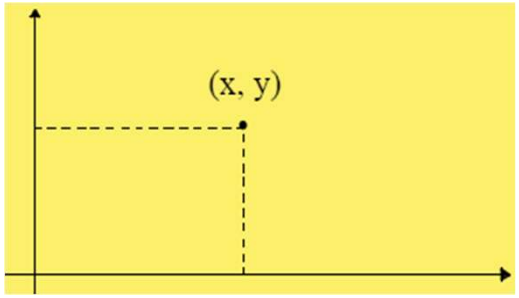
- Para acessar os campos, usamos o operador ''ponto'' (.).
  - **p.x = 10.0;**
  - **p.y = 5.0;**

Definição em C

- Ao invés de representarmos os pontos (x1,y1) e (x2,y2):
  - **float x1; float y1;**
  - **float x2; float y2;**
- Podemos utilizar a representação a seguir:
  - **struct ponto**

```
{
 float x;
 float y;
};
```

- **struct ponto p1, p2;**



Exemplo – Estrutura Ponto

- **acesso aos dados:**
- **struct_var.campo**

Ex:

- p.x = 10; /*atribuição */
- p.y = 15;
- if (p.x >= p.x) &&(p.y >= p.y) ...

- Ver exemplo:
- (struct1.c) e (struct2.c)

Exemplo – Estrutura Ponto

```
#include<stdio.h>
struct ponto
{  float x; float y; };

int main(void)
{ struct ponto p;
  printf("Digite as coordenadas do pontos");
  scanf("%f",&p.x);
  scanf("%f",&p.y);
  printf("o ponto fornecido foi:(%.2f,%.2f)\n",p.x,p.y);
 return 0;
}
```

- A passagem de variáveis do tipo estrutura para funções funciona de maneira análoga à de variáveis simples .

```
void imprime(struct ponto p)
{
 printf("Ponto(%.2f,%.2f)\n:",p.x,p.y);
}
```

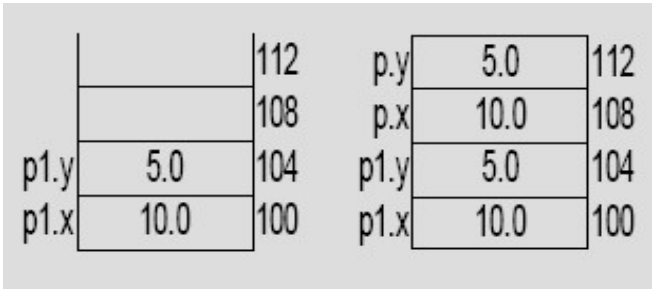
- O valor da variável é copiado para a pilha de execução.

Passagem de Estruturas para Funções – Passagem por Valor

```
struct ponto
{ float x; float y;};
void imprime(struct ponto p)
{ printf("o ponto fornecido foi:(%.2f,%.2f)\n",p.x,p.y);}

int main(void)
{
  struct ponto p;
  printf("Digite as coordenadas do pontos");
  scanf("%f",&p.x);
  scanf("%f",&p.y);
  imprime(p);
  getche();
  return 0;
}
```

struct2pag101.cpp



Inicialização de Estruturas

- **Inicialização de uma Estrutura**

  struct ponto p = { 220, 110 };

- Atribuição entre estruturas *do mesmo tipo*:

  struct ponto p1 = { 220, 110 };

  struct ponto p2;

  p2 = p1; /* p2.x = p1.x e p2.x = p1.y */

- Os campos correspondentes das estruturas são automaticamente copiados do destino para a fonte.

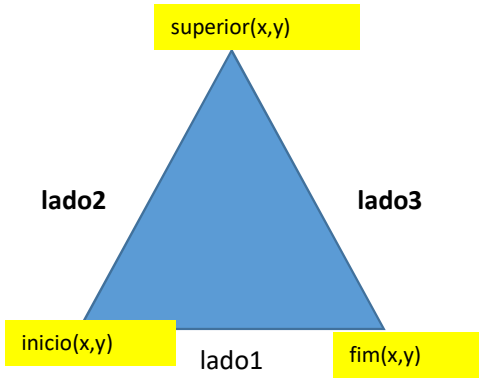
Exercícios

- **1)Representar um circulo por um struct, funções imprime, desloca.**
  - **struct3Ex1PII.cpp**

- **2)Representar um retangulo por um struct, funções imprime, calcula_diagonal, calcula_area.**
  - Considerar que retangulo é definido por um **ponto inicio (canto inferior esquerdo)** e por um **ponto fim (canto superior direito).**
  - **struct4Ex2PII.cpp**

Exercícios

- **3)**Representar um triangulo por um s**truct,** funções **imprime, calcula_area.** Considerar que retangulo é definido por um três pontos : ponto **inicio (canto inferior esquerdo)** e por um ponto **fim (canto inferior direito), ponto cima canto superior)**
  - **struct4Ex3PII.cpp**



- Fórmula da área : AREA = SQRT(T(T - L1)(T - L2)(T - L3))
- T=(L1+L2+L3)/2;

Ponteiros para Estruturas

- Acesso ao valor de um campo x de uma variável estrutura p, p.x ;
- Acesso ao valor de um campo x de uma variável ponteiro pp , pp->x;
- Acesso ao endereço de um campo x de uma variável ponteiro pp, &pp->x;

- Ex: struct ponto *pp;

  (*pp).x=12.0;

  ou

  pp->x=12.0;

Passagem de Estruturas para funções – por referência.

```
#include<stdio.h>
struct ponto
{float x; float y;};

void imprime(struct ponto *pp)
{ printf("o ponto fornecido foi:(%.2f,%.2f)\n",pp->x,pp->y);}

int main(void)
{struct ponto p;
  printf("Digite as coordenadas do pontos");
  scanf("%f",&p.x); scanf("%f",&p.y);
  imprime(&p);
  return 0;
}
```

- struct5Ex1.cpp

Passagem de Estruturas para funções – por referência.

```
#include<stdio.h>
struct ponto {  float x;  float y; };

void imprime(struct ponto *pp)
{ printf("o ponto fornecdido foi:(%.2f,%.2f)\n",pp->x,pp->y); }

void captura(struct ponto* pp)
{ printf("Digite as coordenadas do pontos");
   scanf("%f",&pp->x); scanf("%f",&pp->y);}

int main(void)
{ struct ponto p;
  captura(&p);
  imprime(&p);
 return 0;
} struct5Ex1.cpp
```

Alocação Dinâmica de Estruturas

- Tamanho do espaço de memória alocado é dado pelo operador sizeof aplicado sobre o tipo estrutura.

- Função malloc retorna o endereço do espaço alocado, que é convertido para o tipo ponteiro da estrutura.

```
struct ponto* p;
p=(struct ponto*)malloc(sizeof(struct ponto));
 if(p==NULL)
 { printf("memoria insuficiente");
 exit(1);}
free(p);
```

Alocação Dinâmica de Estruturas

```
#include<stdlib.h>
#include<stdio.h>
struct ponto
{   float x;float y;
};

void imprime(struct ponto *pp)
{  printf("o ponto fornecido foi:(%.2f,%.2f)\n",pp->x,pp->y);  }

void captura(struct ponto* pp)
{ printf("Digite as coordenadas do pontos");
  scanf("%f",&pp->x);  scanf("%f",&pp->y);
}
```

Alocação Dinâmica de Estruturas

```
int main(void)
{
  struct ponto* p;
  p=(struct ponto*)malloc(sizeof(struct ponto));

  if(p==NULL)
  {
    printf("memória insuficiente");
    exit(1);
  }
  captura(p);
  imprime(p);
  free(p);
  return 0;
}
```

struct5Ex3AD.cpp

Similaridades com Orientação a Objetos

- Estruturas similaridades com Orientação a Objetos – Métodos Construtores.

```
struct ponto cria_ponto (float x, float y)
{
  struct ponto tmp;
  tmp.x = x;
  tmp.y = y;
  return tmp;
}
```

structEx6construt

Similaridades com Orientação a Objetos

```
#include<stdio.h>
#include<conio.h>

struct ponto
{ float x; float y;};

struct ponto cria_ponto (float x, float y)
{
 struct ponto tmp;
 tmp.x = x;
 tmp.y = y;
 return tmp;
}

void imprime(ponto p)
{
  printf("O Ponto Fornecido foi:(%.2f,%.2f)\n",p.x,p.y);
}
```

structEx6construt

Similaridades com Orientação a Objetos

```
int main (void)
{
 struct ponto p = cria_ponto(10, 20);
 imprime(p);
 getch();
}
```

structEx6construt

Definição de Novos Tipos

- **A linguagem C permite criar nomes de tipos**

```
typedef float Real;
typedef struct ponto Ponto;
typedef struct ponto *PPonto;
```

- **Após essa definição, podemos declarar:**

```
Real r = 10.5;
Ponto p; p.x = 10.0;
PPonto pp = &p;
pp->x = 12.0;
```

Definição de Novos Tipos

- Podemos definir uma estrutura e associar mnemônicos para ela em um mesmo comando

```
typedef struct ponto
{
 float x;
 float y;
} Ponto;
A partir daí, utiliza-se
Ponto p;
p.x = 10.0;
```

Aninhamento de Estruturas

- Os campos de uma estrutura podem ser outras estruturas previamente definidas.
- Um círculo é formado por um ponto (x,y) e o raio r.

```
struct circulo
  {
   float x, y;
   float r;
  };
  struct circulo
  {
   Ponto p;
   float r;
  };
```

Exercício

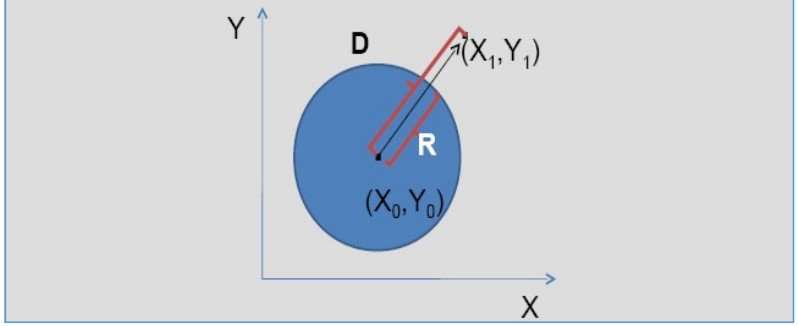
- Ex4)Fazer um programa em C que testa se um ponto é interior a um circulo.
- Sugestão usar estruturas

```
typedef struct ponto
{
  float x;
  float y;
}Ponto;

typedef struct circulo
{
 Ponto p;
 float r;
}Circulo;
```

- **Struct7pag102.c**

```
int interior(Circulo *c, Ponto *p);
```



Vetor de Estruturas

- Uso de vetores para agrupar elementos dos tipos básicos (vetores de inteiros, por exemplo).
- Nesta seção, discutiremos o uso de vetores de estruturas, isto é, vetores cujos elementos são estruturas.
- **Exemplo:** cálculo do centro geométrico de um conjunto de pontos.
  - **Entrada**:vetor de estruturas definindo o conjunto de pontos.
  - **Saída**:centro geométrico, dado por:

$$\bar{x} = \frac{\sum x_i}{n} \qquad \bar{y} = \frac{\sum y_i}{n}$$

Vetor de Estruturas

```
#include<stdio.h>
#include<conio.h>
struct ponto
{  float x;   float y;
};
typedef struct ponto Ponto;

Ponto centro_geom(int n,Ponto* v)
{
  int i;
  Ponto p={0.0f,0.0f};
  for(i=0;i<n;i++)
  {
    p.x+=v[i].x;
    p.y+=v[i].y;
  }
  p.x/=n;
  p.y/=n;
  return p;
}
```

Vetor de Estruturas

```
int main(void)
{
  Ponto p[3]={{1.0,1.0},{5.0,1.0},{4.0,3.0}};
  printf("Digite as coordenadas do pontos\n");
  Ponto p1=centro_geom(3,p);
  printf("%f\n",p1.x);
  printf("%f\n",p1.y);
  return 0;
}
```

Vetor de Ponteiros para Estruturas

- **1.Armazenar um vetor de estruturas.**
  –Ocupa grande quantidade de memória, porém o acesso é mais rápido.
  –Quando o número alocado é sempre Utilizado.
- **2.Armazenar um vetor de ponteiros para estruturas.**
  – Ocupa pouca memória, porém o acesso é mais lento.
  – Quando o número alocado não é sempre utilizado.

Vetor de Ponteiros para Estruturas

- Da mesma forma que podemos declarar vetores de estruturas, podemos usar vetor de ponteiros para estruturas.
- O uso de vetores de ponteiros é útil quando temos de tratar um conjuntos de elementos complexos.
- **Exemplo : Cadastro de Alunos, podemos organizá-los em um vetor. Para cada aluno são necessárias as seguintes informações :**

  - **Matrícula** : número inteiro;
  - **Nome** : cadeia de até 80 caracteres;
  - **Endereço** : cadeia com até 120 caracteres;
  - **Telefone** : cadeia com até 20 caracteres;

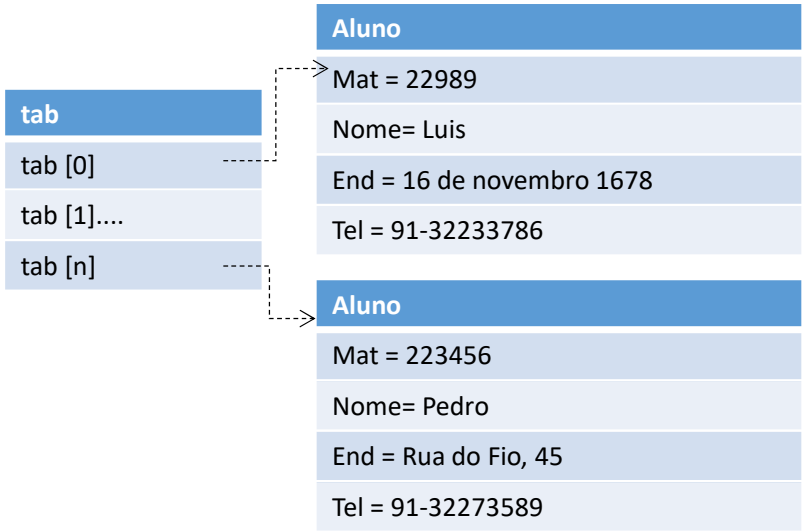
Vetor de Ponteiros para Estruturas

- **Exemplo: Cadastro de Alunos.**
- Para estruturar esses dados, pode-se definir um tipo que representa os dados de um aluno.

```
struct aluno
{
  int mat; char nome[81]; char end [121]; char tel[21];
};
typedef struct aluno Aluno;
```

- Vamos montar a tabela de alunos usando um vetor com um número máximo de alunos.

```
#define MAX 100
Aluno tab[MAX];
```



Vetor de Ponteiros para Estruturas

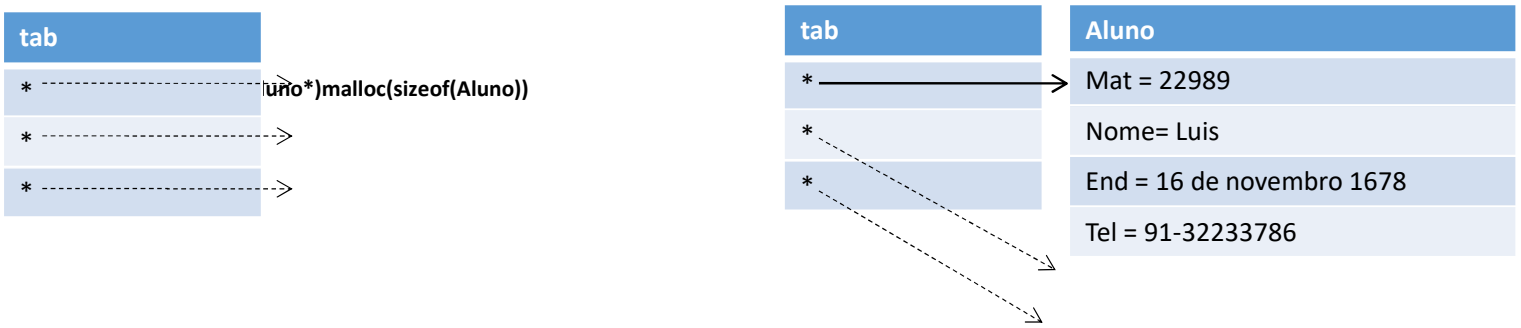
- **Exemplo: Cadastro de Alunos.**
- Tab[i].mat=9912222;
- **Desvantagens:**
  - O tipo Aluno definido ocupa pelo menos 227 (=4+81+121+21) bytes.
  - A declaração de um vetor dessa estrutura representa um desperdício significativo de memória!.
- Para contornar esse problema podemos trabalhar com um vetor de ponteiros.

```
#define MAX 100
Aluno* tab[MAX];
```

Vetor de Ponteiros para Estruturas

- **Exemplo: Cadastro de Alunos.**
- Para contornar esse problema podemos trabalhar com um vetor de ponteiros.

```
#define MAX 100
Aluno* tab[MAX];
```



Ver Cadastro_Alunos.cpp

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Algoritmos – Teoria e Prática, Tradução da Segunda Edição. Campus, 2016.
- Ziviani, N. Projeto de Algoritmos Com Implementações em Pascal e C, Pioneira Thomson Learning, 4ed. 2009.
- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados, Editora Campus* (2004)