

Revisão de Tipos Estruturados





Tipos Estruturados

- Tipo Estruturado é uma abstração na representação de dados que nos permite desenvolver programas mais complexos.
- A linguagem C permite a construção de tipos de dados complexos, nas quais as informações são compostas por diversos campos mais simples.

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 p;
```

- Para acessar os campos, usamos o operador "ponto" (.).

```
p.x = 10.0;
```

```
p.y = 5.0;
```

Exemplo

- Ao invés de representarmos os ponto (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

```
#include<stdio.h>
struct ponto{
    float x;
    float y;
};
int main (void) {
    struct ponto p;
    printf("Digite as coordenadas do ponto(x,y):\n");
    scanf("%f %f",&p.x,&p.y);
    printf("Ponto(%.2f,%.2f):\n",p.x,p.y);
    return 0;
}
```

Passagem de Estrutura para Função

- 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);  
}
```

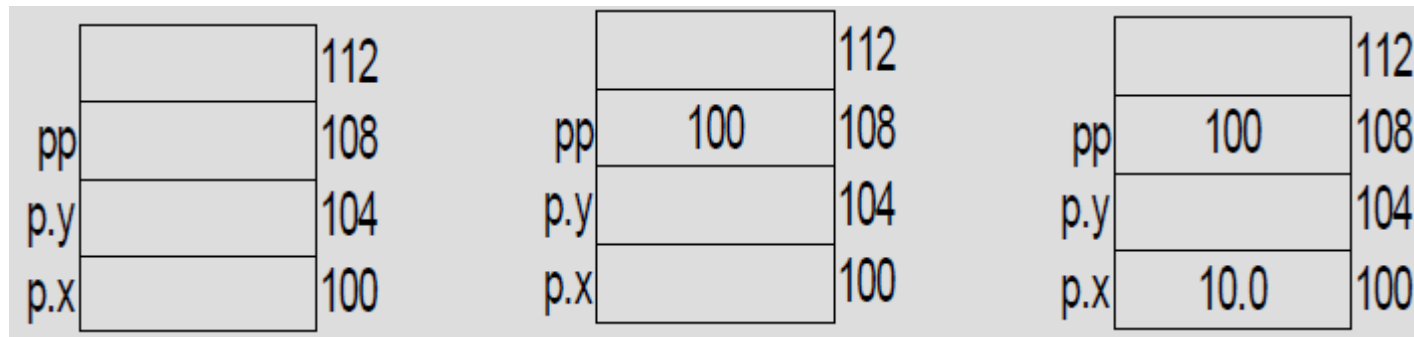
- O valor da variável é copiado para a pilha de execução

		112	p.y	5.0	112
		108	p.x	10.0	108
p1.y	5.0	104	p1.y	5.0	104
p1.x	10.0	100	p1.x	10.0	100

Passagem de Estrutura para Função

- Podemos utilizar variáveis ponteiros

```
struct ponto p;  
struct ponto *pp;  
pp = &p;  
(*pp).x = 10.0;
```



- Para facilitar o uso, define-se o operador (->)

```
pp->x = 10.0;
```


Exemplo

```
#include<stdio.h>
struct ponto{
    float x;
    float y;
};
void captura(struct ponto *pp){
    printf("Digite as coordenadas do ponto (x,y) \n:");
    scanf("%f %f", &pp->x, &pp->y);
}
void imprime(struct ponto *pp){
    printf("Ponto (%.2f, %.2f) : \n", pp->x, pp->y);
}
int main (void) {
    struct ponto p;
    captura(&p);
    imprime(&p);
    return 0;
}
```

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

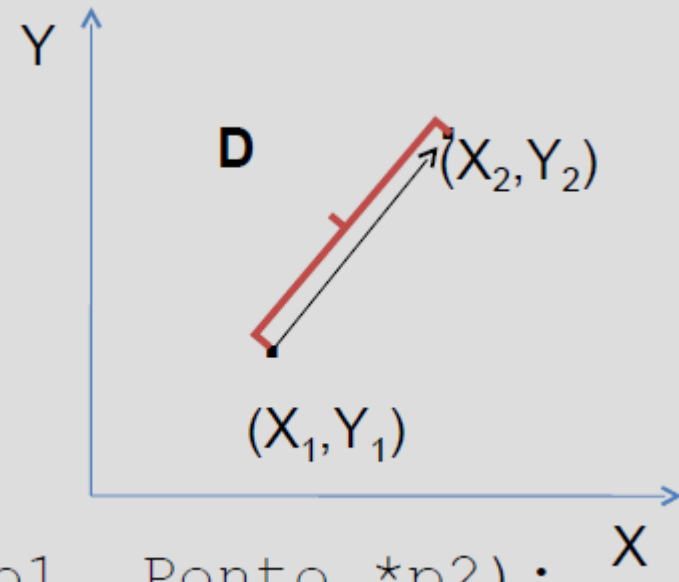
```
struct circulo{  
    float x, y;  
    float r;  
};
```

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

Exercício

- Faça uma função que calcule a distância entre dois pontos conforme o protótipo abaixo

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



```
float distancia(Ponto *p1, Ponto *p2);
```

- Faça um programa completo que capture dois pontos e imprima a distância entre eles

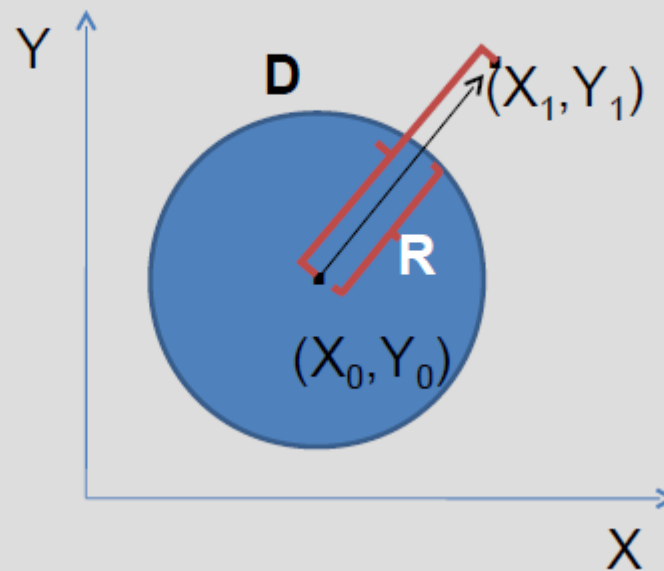
Solução (prog3)

```
#include<stdio.h>
#include<math.h>
typedef struct ponto{
    float x;
    float y;
} Ponto;
void captura(Ponto *pp); // Como definido em slide anterior
float distancia(Ponto *p1, Ponto *p2){
    return sqrt((p1->x - p2->x)*(p1->x - p2->x) +
        (p1->y - p2->y)*(p1->y - p2->y) );
}
int main (void) {
    Ponto p1, p2;
    captura(&p1);
    captura(&p2);
    printf("Distancia entre pontos %.1f\n",distancia(&p1,&p2));
    return 0;
}
```

Exercício

- Faça uma função, conforme protótipo abaixo, que retorne 1 se um ponto está dentro de um círculo e 0 caso contrário.

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




Solução (prog4)

```
#include<stdio.h>
typedef struct ponto{float x; float y;} Ponto;
typedef struct circulo{Ponto p; float r;} Circulo;
void capturaPonto(Ponto *pp);
void capturaCirculo(Circulo *pc);
float distancia(Ponto *p1, Ponto *p2);
int interior(Circulo *c, Ponto *p){
    float d = distancia(&c->p,p);
    return (d < c->r);
}
int main (void) {
    Ponto p; Circulo c;
    capturaPonto(&p); capturaCirculo(&c);
    if(interior(&c,&p))
        printf("Pertence ao circulo\n");
    else
        printf("Nao Pertence ao circulo\n");
    return 0;
}
```




Vetores de Estruturas

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



Exemplo - Alunos

- Tabela com dados de alunos
 - Nome: cadeia de 80 caracteres
 - Ira: número real no intervalo [0,100.00]
- Principais funções:
 - Inicializar o vetor dos alunos
 - Atualizar os dados de um aluno
 - Imprimir os dados de um aluno
 - Imprimir todos os dados dos alunos
 - Excluir os dados de um aluno
- Duas soluções:
 - Utilizar um vetor de alunos
 - Utilizar um vetor de ponteiro para alunos

Solução 1 para MAX alunos (prog5)

```
#include<stdio.h>
typedef struct aluno{
    char nome[81];
    float ira;
} Aluno;
#define MAX 100
#define SEMALUNO -1 // Valor qd aluno inexistente no vetor
void inicializa(int n, Aluno alunos[]);
void atualiza(int n, Aluno alunos[], int i);
void imprime(int n, Aluno alunos[], int i);
void imprime_todos(int n, Aluno alunos[]);
void exclui(int n, Aluno alunos[], int i);
int main (void){
    Aluno alunos[MAX];
    int i;
    inicializa(MAX, alunos);
    for(i=0; i<MAX; i++)
        atualiza(MAX, alunos, i);
    imprime_todos(MAX, alunos);
    return 0;
}
```

Solução 1 para MAX alunos

- Os elementos do vetor (estruturas de alunos) que não são alunos serão "marcados" pelo IRA com o valor SEMALUNO (-1)

```
void inicializa(int n, Aluno alunos[]){  
    int i;  
    for(i=0; i<n; i++)  
        alunos[i].ira = SEMALUNO;  
}
```

Solução 1 para MAX alunos

```
void imprime(int n, Aluno alunos[], int i){
    if(i<0||i>=n){
        printf("Indice fora do limite do vetor!!!\n");
        exit(1);
    }
    if(alunos[i].ira != SEMALUNO){
        printf("Nome: %s\n",alunos[i].nome);
        printf("Ira: %.2f\n",alunos[i].ira);
    }
}

void imprime_todos(int n, Aluno alunos[]){
    int i;
    printf("Listagem de Alunos\n");
    for(i=0;i<n;i++)
        imprime(n,alunos,i);
}
```

Solução 1 para MAX alunos

```
void atualiza(int n, Aluno alunos[], int i){
    float ira;
    if(i<0||i>=n){
        printf("Indice fora do limite do vetor!!!\n");
        exit(1);
    }
    printf("Entre com o nome do aluno\n");
    scanf(" %[^\\n]", alunos[i].nome);
    while(1){
        printf("Entre com o IRA do aluno\n");
        scanf(" %f", &ira);
        if(ira<0||ira>100)
            printf("IRA pertence ao intervalo [0.0, 100.0]\n");
        else
            break;
    }
    alunos[i].ira = ira;
}
```

Solução 1 para “n” alunos (prog6)

```
...
int main (void){
    Aluno *alunos;
    int n, i;
    printf("Entre com o numero de alunos:\n");
    scanf("%d", &n);
    alunos = (Aluno*)malloc(n*sizeof(Aluno));
    if(alunos == NULL){
        printf("Memoria Insuficiente!!!\n");
        exit(1);
    }
    inicializa(n, alunos);
    for(i=0; i<n; i++)
        atualiza(n, alunos, i);
    imprime_todos(n, alunos);
    free(alunos);
    return 0;
}
```


Solução 2 para MAX alunos (prog7)

```
#include<stdio.h>
typedef struct aluno{
    char nome[81];
    float ira;
} Aluno;
#define MAX 100
void inicializa(int n, Aluno **alunos);
void imprime(int n, Aluno **alunos, int i);
void imprime_todos(int n, Aluno **alunos);
void atualiza(int n, Aluno **alunos, int i);
void exclui(int n, Aluno **alunos, int i);
int main (void){
    Aluno* alunos[MAX];
    int i;
    inicializa(MAX,alunos);
    for(i=0;i<MAX;i++)
        atualiza(MAX,alunos,i);
    imprime_todos(MAX,alunos);
    for(i=0;i<MAX;i++)
        exclui (MAX,alunos,i);
    return 0;
}
```


Solução 2 para MAX alunos

- Os elementos do vetor (ponteiros para estruturas de alunos) que não são alunos serão "marcados" como NULL (nulos)

```
void inicializa(int n, Aluno **alunos) {  
    int i;  
    for (i=0; i<n; i++)  
        alunos[i] = NULL;  
}
```

Solução 2 para MAX alunos

```
void imprime(int n, Aluno **alunos, int i){
    if(i<0||i>=n){
        printf("Indice fora do limite do vetor!!!\n");
        exit(1);
    }
    if(alunos[i] != NULL){
        printf("Nome: %s\n",alunos[i]->nome);
        printf("Ira: %.2f\n",alunos[i]->ira);
    }
}

void imprime_todos(int n, Aluno **alunos){
    int i;
    printf("Listagem de Alunos\n");
    for(i=0;i<n;i++)
        imprime(n,alunos,i);
}
```

Solução 2 para MAX alunos

```
void atualiza(int n, Aluno **alunos, int i){
    float ira;
    if(i<0||i>=n){
        printf("Indice fora do limite do vetor!!!\n"); exit(1);
    }
    if(alunos[i]==NULL) {
        alunos[i] = (Aluno*)malloc(sizeof(Aluno));
        if(alunos[i]==NULL) {
            printf("Memoria insuficiente!!!\n"); exit(1);
        }
    }
    printf("Entre com o nome do aluno\n");
    scanf(" %[^\\n]", alunos[i]->nome);
    while(1){
        printf("Entre com o IRA do aluno\n");
        scanf(" %f", &ira);
        if(ira<0||ira>100)
            printf("IRA pertence ao intervalo [0.0, 100.0]\n");
        else
            break;
    }
    alunos[i]->ira = ira;
}
```

Solução 2 para MAX alunos

```
void exclui(int n, Aluno **alunos, int i){  
    if(i<0||i>=n){  
        printf("Indice fora do limite do vetor!!!\n");  
        exit(1);  
    }  
    if(alunos[i]!=NULL){  
        free(alunos[i]);  
        alunos[i] = NULL;  
    }  
}
```

Solução 2 para “n” alunos (prog8)

```
...
int main (void){
    Aluno** alunos;
    int n, i;
    printf("Entre com o numero de alunos:\n");
    scanf ("%d", &n);
    alunos = (Aluno**)malloc(n*sizeof(Aluno*));
    if(alunos==NULL) {
        printf("Memoria insuficiente!!!\n"); exit(1);
    }
    inicializa(n, alunos);
    for(i=0; i<n; i++)
        atualiza(n, alunos, i);
    imprime_todos(n, alunos);
    for(i=0; i<n; i++)
        exclui(n, alunos, i);
    free(alunos);
    return 0;
}
```

Resumo

- Definição de Estrutura
 - `struct ponto { float x; float y;};`
- Definição de nomes para Tipos Estruturados
 - `typedef struct ponto Ponto;`
 - `typedef struct ponto { float x; float y;} Ponto;`
- Definição de variáveis
 - `struct ponto a; Ponto a; Ponto *pa;`
- Acesso aos valores
 - `a.x = 10; pa = &a; pa->y = 12.0;`
- Soluções para Vetores de Estruturas

Vetor de	Memória	Acesso
Estruturas	+	+
Ponteiros p/ Estruturas	-	-