Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

Estruturas em C

Material preparado pela profa
Silvana Maria Affonso de Lara

2º semestre de 2010

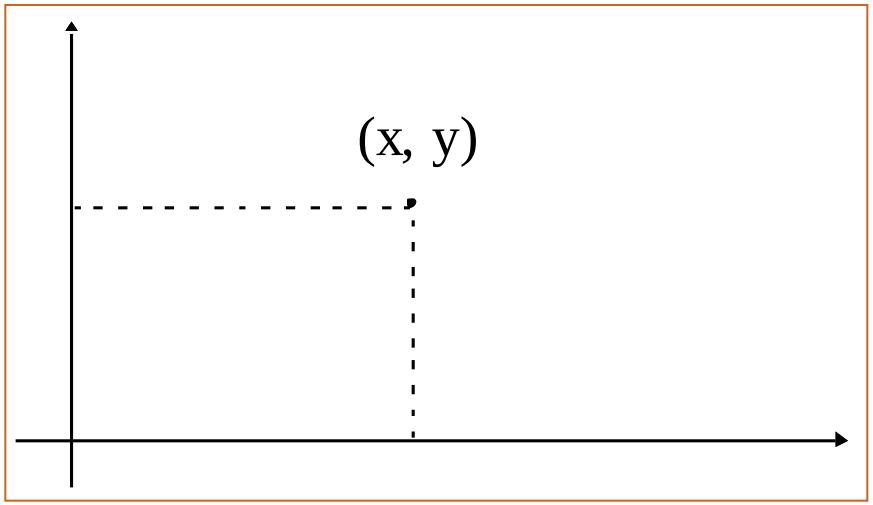
Roteiro da Aula

- Definição e Declaração de Estruturas
- Nomeando uma estrutura
- Atribuição de estruturas
- Composição de estruturas
- Estruturas como parâmetros
- Operações
- Ponteiros para estrutura
- Arrays de estruturas
- Classes de armazenamento

Estruturas

- `struct` são coleções de dados heterogêneos agrupados em uma mesma estrutura de dados

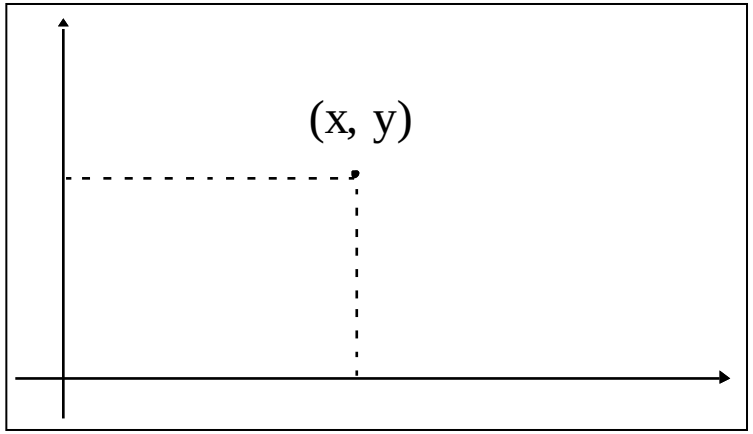
- Ex: armazenar as coordenadas (x,y) de um ponto:



Estruturas

- Declaração:

```
struct {
        int x;
        int y;
      } p1, p2;
```



- a estrutura contém dois inteiros, *x* e *y*
- *p1* e *p2* são duas variáveis tipo *struct* contendo duas coordenadas cada.

Declaração

- Formato da declaração:

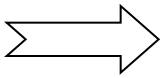
```
struct nome_da_estrutura {
                        tipo_1 dado_1;
                        tipo_2 dado_2;
                        ...
                        tipo_n dado_n;
                } lista_de_variaveis;
```

- A estrutura pode agrupar um número arbitrário de dados de tipos diferentes
- Pode-se nomear a estrutura para referenciá-la

Nomeando uma Estrutura

```
struct {
  int x;
  int y;
} p1;

struct {
  int x;
  int y;
} p2;
```

Repetição 

```
struct ponto {
        int x;
        int y;
};
struct ponto p1, p2;
```

struct ponto define um *novo tipo de dado*

- Pode-se definir novas variáveis do tipo **ponto**

Estruturas

- acesso aos dados:

struct-var.campo

Ex:

```
p1.x = 10;     /*atribuição */
p2.y = 15;
if (p1.x >= p2.x) &&
   (p1.y >= p2.y) ...
```

Atribuição de Estruturas

- Inicialização de uma estrutura:

  struct ponto p1 = { 220, 110 };

- Atribuição entre estruturas *do mesmo tipo*:

```
struct ponto p1 = { 220, 110 };
struct ponto p2;

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

- Os campos correspondentes das estruturas são automaticamente copiados da fonte para o destino

Atribuição de Estruturas

- Atenção para estruturas que contenham ponteiros:

```
struct aluno {
        char *nome; int idade;
} a1, a2;

a1.nome = "Alfredo";
a1.idade = 22;
a2 = a1;
```

Agora a1 e a2 apontam para o mesmo *string* nome:

a1.nome == a2.nome == "Alfredo"

Composição de Estruturas

```
struct retangulo {
        struct ponto inicio;
        struct ponto fim;
      };
      struct retangulo r = { { 10, 20 }, { 30 , 40 } };
```

Acesso aos dados:

```
r.inicio.x += 10;
r.inicio.y -= 10;
```

Estruturas como retorno de função

```
struct ponto cria_ponto (int x, int y) {
        struct ponto tmp;

        tmp.x = x;
        tmp.y = y;
        return tmp;
}

main () {
        struct ponto p = cria_ponto(10, 20);
}
```

Operações

- operações entre membros das estruturas devem ser feitas membro a membro:

```
/* retorna uma cópia de p1 = p1 + p2 */
        struct ponto soma_pts (struct ponto p1, struct
ponto p2)
        {
         p1.x += p2.x;
         p1.y += p2.y;

         return p1;   /* retorna uma copia de p1 */
        }
```

Ponteiros para Estruturas

- estruturas grandes são passadas como parâmetro de forma mais eficiente através de ponteiros

```
struct ponto *pp;
        struct ponto p1 = { 10, 20 };
        pp = &p1;
        printf("Ponto P1: (%d %d)\n", (*pp).x, (*pp).y};
```

- acesso via operador "->":

```
printf("Ponto P1: (%d %d)\n", pp->x, pp->y};
```

Arrays de Estruturas

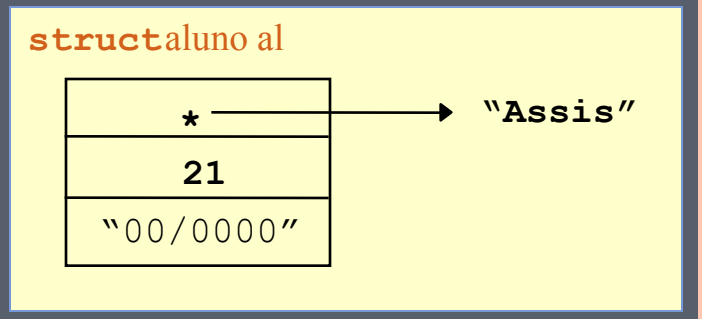
```
struct ponto arp[10];
/* cria um array de 10 pontos */
arp[1].x = 5; /*atribui 5 a coordenada x do 2º ponto */

struct jogador {
    char *nome;
    int idade;
};
struct jogador Brasil[11] = {
            "Robinho",        21,
            "Ronaldinho", 23, ...
};
```

Espaço Alocado para uma Estrutura

```
struct aluno {
    char *nome; /* ponteiro 4 bytes */
    short idade;            /* 2 bytes */
    char matricula[8];      /* array 8 bytes */
};

struct aluno al;
al.nome = "Assis";
al.idade = 21;
strcpy(al.matricula, "00/0001");
```



FUNÇÃO *SIZEOF(TIPO)*

- A função *sizeof(tipo)* retorna o tamanho em bytes ocupado em memória pelo tipo de dado passado como parâmetro

- Ex.:

```
sizeof(int)              => 4 bytes
sizeof(char)             => 1 byte
sizeof(struct ponto)  => 8 bytes
sizeof(struct ponto *) => 4 bytes
```

Classes de armazenamento

- Todas as variáveis e funções em C possuem dois atributos: tipo e classe de armazenamento

- As classes de armazenamento podem ser:

  `Auto`

  `Extern`

  `Register`

  `Static`

Classe auto

- Por *default*

```
auto int a, b, c;
```

- quando entra num bloco, o sistema aloca memória para as variáveis automáticas
- no bloco, essas variáveis são definidas e consideradas "locais" ao bloco
- quando sai do bloco, o sistema não reserva mais o espaço para essas variáveis.

Classe extern

- Método para se transmitir informações entre blocos e funções

- A declaração de variável **extern** possibilita que a variável seja considerada *global*

EXEMPLO

```
#include <stdio.h>
extern int a = 1, b = 2, c = 3;
int func(void);
 main() {
      printf("%3d\n", func());
      printf("%3d%3d%3d\n", a, b, c);
}
int func(void) {
      int b, c;
      a = b = c;
      a = b = c = 4;
      return (a + b + c);
}
```

Resultado: 12

4 2 3

```
// no arquivo p1.c
#include <stdio.h>
int a = 1, b = 2, c = 3;
int func(void);
 main() {
      printf("%3d\n", func());
       printf("%3d%3d%3d\n", a, b, c);
}
```

```
// no arquivo p2.c
int func(void) {
      extern int a; /* procura em algum outro lugar */
      int b, c;
      a = b = c;
      a = b = c = 4;
      return (a + b + c);
}
```

Classe `register`

- Indica ao compilador que as variáveis associadas devem ser armazenadas em registradores na memória de alta velocidade
- Basicamente, o seu uso é a tentativa de melhorar a velocidade de execução.

```
{     register int i;
      for (i = 0; i < LIMIT; ++i) {
       …
      }
}   // ao sair do bloco, o registrador será liberado
```

```
register i;     // equivale a
register int i;
```

Classe `static`

- As declarações **static** possuem dois usos distintos e importantes:
  - Permitir que uma variável local retenha seu valor anterior quando `entra´ num bloco novamente.
  - Estabelece ´privacidade´ na conexão com declarações externas.

```
void f(void)
{    static int cont = 0;
      ++ cont;
      if (cont % 2 == 0) {
           ... /* faz algo e imprime PAR */
      else
           ... /* faz diferente */
      }
```

- A primeira vez que a função é chamada, a variável **cont** é inicializada com zero
- MAS, ao sair da função o valor de cont é preservado na memória
- Quando a função é chamada novamente, **cont NÃO é** reinicializada e o seu valor é o que possuía na ultima chamada.

Exercício Prático

- Considere um cadastro de produtos de um estoque, com as seguintes informações para cada produto:
  - Código de identificação do produto: representado por um valor inteiro
  - Nome do produto: com ate 50 caracteres
  - Quantidade disponível no estoque: representado por um número inteiro
  - Preço de venda: representado por um valor real

(a) Defina uma estrutura em C, denominada produto, que tenha Os campos apropriados para guardar as informações de um produto, conforme descrito acima.

(b) Escreva uma função que receba os dados de um produto (código, nome, quantidade e preço) e retorne o endereço de um struct produto inicializado com os valores recebidos como parâmetros pela função. Essa função pode ter o seguinte protótipo:

```
void cria (int cod, char* nome, int quant, float preco, struct produto *p);
```

Solução

```
struct produto{
    int cod;
    char nome[51];
    int quant;
    float preco;
};

struct produto Produto;

void cria (int cod, char* nome, int quant, float preco, Produto
    *p) {
    p->cod = cod;
    strcpy(p->nome, nome); /* strcpy(destino,origem) */
    p->quant = quant;
    p->preco = preco;
}
```

Solução

```
int main(void){

    int codigo, quantidade;
    float preco;
    char nomep[51];
    Produto *paux;
    Produto prod;

    printf("Digite o cod: ");
    scanf("%d",&codigo);
    printf("\nDigite o nome: ");
    scanf("%s",&nomep);
```

Solução

```
printf("\nDigite a quantidade: ");
 scanf("%d",&quantidade);
 printf("\nDigite o preco: ");
 scanf("%f",&preco);
 paux=&prod;
 cria(codigo,nomep,quantidade,preco,paux);
printf("\n\nA estrutura criada foi:");
 printf("\nCod: %d", paux->cod);
 printf("\nNome: %s", paux->nome);
 printf("\nQuant: %d", paux->quant);
 printf("\nPreco: %6.2f", paux->preco);
getch();
}
```

Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

Estruturas em C

Material preparado pela profa
Silvana Maria Affonso de Lara

2º semestre de 2010