Structs

Programação Estruturada - Cris Sato

struct

- Permite agrupar vários tipos de dados em uma estrutura
- No exemplo, abaixo criamos um ponto, que consiste de duas coordenadas inteiras x e y:

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

- criando um ponto com coordenadas (2,3):

```
struct ponto p;
p.x = 2;
p.y = 3;
```

ou

```
struct ponto p = {2,3};
```

ou

```
struct ponto p = {.x=2,.y=3};
```

typedef

- É muito comum utilizar o comando typedef para renomear struct:

```
struct ponto_s {
    int x;
    int y;
};
typedef struct ponto_s ponto;
```

Agora ao invés de termos que digitar struct ponto_s, podemos usar o tipo ponto:

```
ponto p;
p.x = 2;
p.y = 3;
```

```
typedef struct ponto_s {
    int x;
    int y;
} ponto;
```

É mais comum fazer o typdef junto com a definição da struct!

A declaração

```
ponto p = {2,3};
```

funciona mesmo se houve tipos diferentes em ponto. Por exemplo:

```
typedef struct ponto_s {
  int x;
  int y;
  char* nome;
} ponto;
```

```
ponto p = {5,3,"casa"};
printf("x=%d\n
y=%d\n nome=%s\n",
p.x,p.y,p.nome);
```

```
x = 5
y = 3
nome = casa
```

```
ponto p = {5};
printf("x=%d\n
y=%d\n nome=%s\n",
p.x,p.y,p.nome);
```

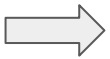
```
x = 5
y = 0
nome = (null)
```

structs dentro de structs

```
typedef struct ponto_s
{
    int x;
    int y;
} ponto;
```

```
typedef struct
circulo_s{
    ponto centro;
    double raio;
} circulo;
```

```
-- main --
ponto p = {1,2}
circulo c = {p, 0.4};
imprime_circulo(c);
```



```
Centro : x=1 y=2
Raio: 0.400000
```

```
void imprime_ponto(ponto p)
{
    printf("x=%d y=%d\n",
    p.x,p.y);
}
```

```
void imprime_circulo(circulo c) {
    printf("Centro : " );
    imprime_ponto(c.centro);
    printf("Raio: %lf\n", c.raio);
}
```

```
typedef struct ponto_s
{
    int x;
    int y;
} ponto;
```

```
typedef struct
circulo_s{
    ponto centro;
    double raio;
} circulo;
```

Para acessar/modificar a coordenada x do centro de um circulo c, basta usar:

```
c.centro.x
```

structs e ponteiros

- Funcionam do mesmo jeito que para outros tipos:

```
ponto *p;
p = malloc(sizeof(ponto));
(*p).x = 0;
(*p).y = 1;
imprime_ponto(*p);
```

x = 0
y = 1

Precisamos dos () por causa da precedência de operadores. Caso contrário, primeiro seria aplicado o .x e depois o *. O problema é que p não tem um x, mas *p tem um x

structs e ponteiros

- Mas temos a seguinte abreviação:

```
ponto *p;
p = malloc(sizeof(ponto));
p->x = 0;
p->y = 1;
imprime_ponto(*p);
```

x = 0
y = 1

p->x é exatamente o mesmo que (*p).x

Exemplo: ordenando structs

```
typedef struct
cadastro_s {
    char *nome;
    int idade;
    int peso;
} cadastro;
```

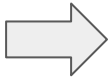
Um vetor com 3 cadastros:

```
cadastro pessoa[3] =
{{"Ana",12,40},{"Bob",4,15},{"Carlos",3,16}};
```

```
int compara_idade (const void *a,const void
*b){
    cadastro *a1 = (cadastro *)a;
    cadastro *b1 = (cadastro *)b;
    return a1->idade-b1->idade;
}
```

Os dados da struct são movidos juntos!

```
qsort(pessoa,3,sizeof(cadastro),compara_idade);
```



Carlos, 3, 16
Bob, 4, 15
Ana, 12, 40