Linguagem de programação

Structs e unions

Structs

Introdução

- **Structs** (estruturas) são conjuntos de variáveis agrupadas que possuem um mesmo *nome*
  - as variáveis são as partes de um todo (Struct)
  - podem conter variáveis de vários tipo: `int`, `char`, `double`, `int*`, `char[10]`...
- Um “tipo” de variável criado pelo programador
- São utilizadas na criação de estruturas mais complexas de dados, como:
  - listas encadeadas;
  - filas;
  - pilhas; e
  - árvores

Definições

- Considere a seguinte definição de struct:

```
struct carta {
 char face[25];
 char naipe[25];
};
```

- **`struct`**: palavra reservada para se criar uma estrutura
- **`carta`**: nome da struct criada. Usada para declarar variáveis do tipo da struct
- **`face[25]`** e **`naipe[25]`** são membros da struct

Obs.: cada definição deve ser encerrada com ; (ponto-e-vírgula)

Declaração

- Uma variável pode ser declarada ou juntamente com a struct

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

- ou como em tipos básicos do C

```
struct ponto {
  int x, y;
};
int main() {
  struct ponto p1; }
```

Inicialização

- Membros não pode ser inicializados junto com a declaração da struct

```
struct ponto {
  int x = 0;  // ERRO DE COMPILACAO
  int y = 0;  // ERRO DE COMPILACAO
};
```

  - não há memória alocada para a atribuição de valores

- Exemplo de declaração válida:
  - `struct ponto p1 = {0, 1};`

Estruturas Auto-Referenciais

- Uma estrutura não pode conter uma instância de si mesma
  - Por exemplo, uma variável do tipo struct Empregado não pode ser declarada na definição de struct empregado

- Um ponteiro para a estrutura de funcionário, no entanto, pode ser incluído. Por exemplo:

```
struct empregado {
   primeiroNome[20];
   sobrenome[20];
   unsigned int age;
   char genero;
   double salario;
   struct empregado pessoa; // erro
   struct empregado *ePtr; // ponteiro
}; //fim da struct empregado
```

Acesso a membros da *struct*

- Membros de estruturas são acessadas utilizando o operador ponto (.)

```
struct ponto{
  int x, y;
};
 int main() {
  struct ponto p1 = {0, 1};
 // Acessando membros do ponto p1
  p1.x = 20;
  printf ("x = %d, y = %d", p1.x, p1.y);
  return 0;
}
```

- Saída:
  - x = 20, y = 1

Vetores de structs

- Assim como outros tipos primitivos, podemos criar vetores de *structs*

```c
struct ponto {
  int x, y;
};
int main() {
  // Cria um vetor de Ponto
  struct ponto arr[10];
   // Acessando os membros do 1ª posição do vetor
  arr[0].x = 10;
  arr[0].y = 20;
  printf("%d %d", arr[0].x, arr[0].y);
return 0; }
```

- Saída:
  - 10 20

Ponteiro de struct

- Como em tipos primitivos, podemos ter ponteiros para *struct*
  - nesse caso, os membros são acessados pelo operador **(->)**
  - **OU** pelo operador (.), desde que faça (*ptr).membro
- No exemplo anterior:
  - **`(*p2).x`** equivale a **`p2->x`**

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

int main() {
 struct ponto p1 = {1, 2};

 // p2 é um ponteiro para struct p1
 struct ponto *prtP1 = &p1;

 // acessando membros da struct usando ponteiro
 printf("%d %d", prtP1->x, prtP1->y);
 return 0; }
```

Typedef

- É um comando que cria um “sinônimo” ou “apelido” para tipos de dados existentes
- Renomeia um tipo de dado, que pode facilitar a organização e o entendimento do código
- Sintaxe: **typedef** <nome do tipo de dado> <novo nome>;

Typedef: exemplo

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

typedef struct ponto
Ponto;

int main() {
 Ponto p1 = {1, 2};
return 0;
}
```

OU

```c
typedef struct{
 int x, y;
}Ponto;

int main() {
 Ponto p1 = {1, 2};
return 0;
}
```

- Não se faz necessário o uso de `struct` toda vez que for usar a estrutura

Exercício

1. Crie uma struct para representar uma Pessoa, com nome (até 100 caracteres) e telefone; em seguida, crie uma agenda para armazenar o contato de até **3** pessoas.

2. Crie uma função

   `void addPessoa(Pessoa *agenda, Pessoa pessoa, int indice)` que adicione uma pessoa à agenda, por vez;

1. Use a função para adicionar **3** pessoas e mostre o conteúdo da agenda na main

Exercício

- A partir do exercício anterior, crie uma função para buscar uma pessoa na agenda (pelo nome) e retornar o número do seu telefone.

Unions

Uniões

- Uma **união** é um tipo derivado de dados (como uma *struct*)
    - Porém, os membros compartilham o mesmo espaço de armazenamento
- Por que usá-las?
    - Às vezes algumas variáveis podem não ser apropriadas, mas outras, são;
    - portanto, uma união compartilha o espaço em vez de desperdiçar armazenamento em variáveis que não estão sendo usadas.
- Os membros de uma união podem ser de qualquer tipo.
- O Nº de bytes usado para armazenar uma união deve ser pelo menos o suficiente para conter o maior membro.

Uniões

- Apenas um membro e, portanto, um tipo de dado, pode ser referenciado por vez.
- É sua responsabilidade assegurar que os dados de uma união sejam referenciados com o tipo apropriado.

Declaração

- Uma definição de *union* tem o mesmo formato que uma definição de *struct*

```
union number {
 int x;
 double y;
}; // end union number
```

Inicialização

- Em uma declaração, uma união *pode ser inicializada com um valor do mesmo tipo do primeiro membro da união.*
  - `union number value = {10};`

- Se fizéssemos `union number value = {1.43};`
  - a declaração truncaria a parte de ponto flutuante e seria produzido um aviso do compilador

Exemplo de utilização

```c
union number {
 int x;
 double y;
};

int main(){
  union number value; // definição da variavel union

  value.x=100; //colocando um inteiro na union

  printf("x=%d y=%lf", value.x,value.y);

  value.y=100.0;//colocando um double na mesma union
  printf("\nx=%d y=%lf", value.x,value.y);

  return 0; }
```

Exercício

- Qual será a saída do seguinte programa em C?

```c
typedef union{
  float salary;
  int workerNo;
} Job;

int main() {
  Job j;
  j.salary = 12.3;
  j.workerNo = 100;
  printf("Salario = %.1f\n", j.salary);
  printf("Nº de trabalhadores = %d", j.workerNo);
  return 0; }
```

Exercício

- Qual será a saída do seguinte programa em C?

R =

Salario = 0.0

Nº de trabalhadores = 100

```c
typedef union{
  float salary;
  int workerNo;
} Job;

int main() {
  Job j;
  j.salary = 12.3;
  j.workerNo = 100;
  printf("Salario = %.1f\n", j.salary);
  printf("Nº de trabalhadores = %d", j.workerNo);
  return 0; }
```