DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Estruturas

K&R: Capítulo 6

IAED

Estruturas

- Introdução às Estruturas
- Estruturas e Funções
- Vectores de Estruturas
- Typedef
- Exemplo

Motivação

- Definir uma representação agregada na linguagem C para manipular inscrições de alunos a disciplinas
- Numa inscrição é necessário representar:
  - Número de aluno
  - Sigla da disciplina (máx. 6 caracteres)
  - Nota obtida

36933 IAED 16

- Como representar em C ?
  - Vector? Todos os dados precisam de ser do mesmo tipo!
  - 3 Vectores do mesmo tamanho?
- Solução: **estruturas** para representar dados agregados de tipos diferentes

Exemplos de Estruturas

- As estruturas permitem definir estruturas de dados sofisticadas, as quais possibilitam a agregação de diferentes tipos de declarações
- Exemplos:

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

```
struct data {
  int dia;
  int mes;
  int ano;
};
```

```
struct inscricao {
  int numero;
  char disciplina[7];
  int nota;
};
```

Estruturas – Declaração de Variáveis

- Declararar variáveis do tipo estrutura

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

struct ponto centro;
```

- Manipular os campos de variáveis do tipo estrutura

```
centro.x = 12.3;
centro.y = 5.2;
```

Estruturas – Declaração de Variáveis

- Definição da estrutura: introduz um novo tipo de dados

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

- Declaração de variável: declara variável como estrutura

```
struct ponto centro;
```

- Inicialização: `<tipo> <variavel> = { <valores> };`

```
struct ponto centro = { 12.3, 5.2 };
```

Manipulação de Estruturas

- Manipulação: `<variavel>.<membro>`

```
centro.x = 12.3;
centro.y = 5.2;
```

- Estruturas podem incluir outras estruturas

```
struct rectangulo {
  struct ponto a, b;
};

struct rectangulo rect;
rect.a.x = 3.4;
rect.b.y = 6.1;
```

Manipulação de Estruturas

- Estruturas ***não*** podem ser comparadas usando operador de comparação de igualdade
  - ***Para determinar se duas estruturas são iguais, é necessário comparar cada campo da estrutura***

Estruturas e Funções

- Funções podem receber e retornar estruturas

```
struct ponto adicionaPonto(struct ponto p1, struct ponto p2) {
  struct ponto res;
  res.x = p1.x + p2.x;
  res.y = p1.y + p2.y;
  return res;
}
```

- Função retorna uma *cópia* da estrutura `res`
- Passagem de argumentos feita por valor
  - Chamada **`adicionaPonto(pa, pb)`** não altera valores de **`pa`** nem de **`pb`**

Estruturas e Funções: exemplo

- Como a passagem de argumentos é feita por valor, podemos redefinir a função:

```
struct ponto adicionaPonto(struct ponto p1, struct ponto p2) {
  p1.x += p2.x;
  p1.y += p2.y;
  return p1;
}

int main() {
  struct ponto pa = {1, 3}, pb = {5, 2};
  struct ponto pc = adicionaPonto(pa, pb);
  ...
}
```

- Ao invocar a função `adicionaPonto(pa, pb)`, os valores de `pa` e `pb` não são alterados

Vectores de Estruturas

- Permitem representar conjuntos de dados agregados

```
struct ponto {
  double x;
  double y;
};
struct ponto figura [100];
```

- Estamos a declarar a variável **`figura`** como um vector de 100 pontos

Vectores de Estruturas: inicialização

- Inicialização

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

struct ponto figura [] = {
  { 1.2, 3.4 },
  { 4.5, 12.6 },
  { 1.2, 10.8 }
};
```

Desta forma estamos
•a declarar um vector de "pontos",
•de dimensão 3,
•inicializando os 3 pontos do vector, atribuindo um valor a cada um dos campos de cada um dos "pontos"

Definição de novos tipos usando `typedef`

- **`typedef`** permite associar um nome a um tipo de dados já existente
  - Permite um grau adicional de abstracção

```
typedef int Inteiro;

int main() {
  Inteiro i;
...
}
```

- Formato

```
typedef <tipo> <nome>;
```

Definição de novos tipos usando `typedef`

- `typedef` permite associar um nome a um tipo de dados já existente
  - Permite um grau adicional de abstracção

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

Ponto a;
```

Ao definirmos um novo tipo, já não necessitamos de usar a `struct ponto` quando declaramos um `Ponto`

- Formato

```
typedef <tipo> <nome>;
```

Exercício : Números complexos

- Implemente a função

```
Complexo soma (Complexo a, Complexo b)
```

Que recebe dois números complexos (definidos como uma estrutura) como argumentos e devolve a soma dos dois números complexos.

```
typedef struct {
  float real;
  float imag;
} Complexo;

Complexo k; /*exemplo de declaração de variavel */
```

Exercício : Números complexos

- Implemente a função

```
Complexo soma (Complexo a, Complexo b)
```

Que recebe dois números complexos (definidos como uma estrutura) como argumentos e devolve a soma dos dois números complexos.

```
typedef struct complexo {
  float real;
  float imag;
} Complexo;

Complexo k; /*exemplo de declaração de variavel */
```

Também posso fazer como anteriormente... Tenho apenas de ter cuidado com o nome

Exercício : Números complexos

- Implemente a função

```
Complexo soma (Complexo a, Complexo b)
```

Que recebe dois números complexos (definidos como uma estrutura) como argumentos e devolve a soma dos dois números complexos.

```
typedef struct complexo {
  float real;
  float imag;
} Complexo;

Complexo k1; /*exemplo de declaração de variavel */
struct complexo k2; /*exemplo equivalente ao anterior */
```

Também posso fazer como anteriormente... Tenho apenas de ter cuidado com o nome

Exercício : Números complexos

```
Complexo somaComplexo(Complexo a, Complexo b)
{
        Complexo soma;
        soma.real = a.real + b.real;
        soma.imag = a.imag + b.imag;
        return soma;
}

Complexo leNumeroComplexo()
{
        Complexo a;
        char i;
        scanf("%f%f%c", &a.real, &a.imag, &i);
        return a;
}

void escreveNumeroComplexo(Complexo a)
{
        if (a.img >= 0) printf("%f+%fi", a.real, a.imag);
        else printf("%f %fi", a.real, a.imag);
}
```

Exercício : Números complexos

```
int main()
{
        Complexo a, b, soma;

        a = leNumeroComplexo();
        b = leNumeroComplexo();

        soma = somaComplexo(a,b);

        escreveNumeroComplexo(soma);

        return 0;
}
```

Exemplo 1: Inscrições e Notas a Disciplinas

- Objectivo:
  - Programa para manipular as inscrições de alunos a disciplinas;
    - Dado um número de aluno, mostra as notas do aluno às disciplinas onde está inscrito
    - Os dados de entrada são todas as inscrições às disciplinas
    - Suponha que numa inscrição representamos o número de aluno, a disciplina e a nota do aluno na disciplina
    - Define-se um limite máximo de inscrições no programa

Exemplo 1: Inscrições e Notas a Disciplinas

- Objectivo:
  - Programa para manipular as inscrições de alunos a disciplinas;
    - Dado um número de aluno, mostra as notas do aluno às disciplinas onde está inscrito
    - Os dados de entrada são todas as inscrições às disciplinas
    - Suponha que numa inscrição representamos o número de aluno, a disciplina e a nota do aluno na disciplina
    - Define-se um limite máximo de inscrições no programa
  - Exemplo dos dados de entrada:
    - 1ª linha com o número total de inscrições (N)
    - N linhas com o formato: Número aluno, nota, sigla disciplina

```
3
36933 16 IAED
12345 14 IAED
23456 8 AC
```

Exemplo 1: Inscrições e Notas a Disciplinas

- tipo “Inscricao”

```
typedef
```

struct inscrição

- **Número do aluno inscrito**
- **Código da disciplina**
- **Nota do aluno na disciplina**

```
Inscricao ;
```

Exemplo 1: Inscrições e Notas a Disciplinas

- Definição dos limites e tipo inscricao

```
#include <stdio.h>

#define MAX_COD_DISC 7
#define MAX_INSCRICOES 10000

typedef struct {
    int numero; /* número do aluno */
    int nota;   /* nota à disciplina */
    char disciplina[MAX_COD_DISC]; /* nome/código da disciplina*/
} Inscricao;
```

Exemplo 1: Inscrições e Notas a Disciplinas

- Protótipos e Main

```
int leTodasInscricoes (Inscricao v[]);
void mostraNotasAluno(Inscricao v[], int n, int aluno);

int main() {
  Inscricao insc[MAX_INSCRICOES];
  int numInscricoes = 0, aluno;

  numInscricoes = leTodasInscricoes(insc);
  scanf("%d", &aluno);
  while (aluno > 0) {
    mostraNotasAluno(insc, numInscricoes, aluno);
    scanf("%d", &aluno);
  }
  return 0;
}
```

Esta função vai ler as inscrições e guardar toda a informação lida no vector `insc`

Exemplo 1: Inscrições e Notas a Disciplinas

- Protótipos e Main

```
int leTodasInscricoes (Inscricao v[]);
void mostraNotasAluno(Inscricao v[], int n, int aluno);

int main() {
  Inscricao insc[MAX_INSCRICOES];
  int numInscricoes = 0, aluno;

  numInscricoes = leTodasInscricoes(insc);
  scanf("%d", &aluno);
  while (aluno > 0) {
    mostraNotasAluno(insc, numInscricoes, aluno);
    scanf("%d", &aluno);
  }
  return 0;
}
```

Loop c/ 2 passos:
1.Lê um nº de aluno
2.Procura aluno e mostra notas

Exemplo 1: Inscrições e Notas a Disciplinas

- Funções para ler e mostrar dados

```
int leTodasInscricoes (Inscricao v[]) {
  int n, i;
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d%d%s", &v[i].numero, &v[i].nota, v[i].disciplina);
  return n;
}

void mostraNotasAluno (Inscricao v[], int n, int aluno) {
  int i;
  for (i = 0; i < n; i++)
    if (aluno == v[i].numero)
      printf("%s %d\n", v[i].disciplina, v[i].nota);
}
```

Exemplo 1: Inscrições e Notas a Disciplinas

- Leitura de strings

```
int leTodasInscricoes (Inscricao v[]) {
  int n, i;
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d%d%s", &v[i].numero, &v[i].nota, v[i].disciplina);
  return n;
}
```

- Recordar: Usar **`%s`** para ler uma sequência de caracteres até encontrar um espaço, tabulação ou fim de linha
  - Coloca `'\0'` no fim do texto
  - Não é necessário usar operador `&` para ler uma string
  - Para ler um nº arbitrário de "palavras", poderia usar o código de leitura de uma linha inteira dado na aula passada

Exercício – procura de strings (código de disciplina)

- Conseguem pensar numa função que procura e mostra todas as notas de uma disciplina?

```
#include <string.h>

void mostraNotasDisciplina (Inscricao v[], int n, char disc[]) {
  int i;
  for (i = 0; i < n; i++)
    if (strcmp(v[i].disciplina, disc) == 0)
      printf("%d %d\n", v[i].numero, v[i].nota);
}
```

- Usar `int strcmp(char a[], char b[])` para comparar duas strings
  - devolve 0 se strings são iguais
  - necessário incluir **`string.h`**

Exercício – procura de strings (código de disciplina)

- Conseguem pensar numa função que procura e mostra todas as notas de uma disciplina?

```
#include <string.h>

void mostraNotasDisciplina (Inscricao v[], int n, char disc[]) {
  int i;
  for (i = 0; i < n; i++)
    if (!strcmp(v[i].disciplina, disc))
      printf("%d %d\n", v[i].numero, v[i].nota);
}
```

- Usar `int strcmp(char a[], char b[])` para comparar duas strings
  - devolve 0 se strings são iguais
  - necessário incluir **`string.h`**