C
Estruturas

Adriano Cruz
adriano@nce.ufrj.br

Instituto de Matemática
Departamento de Ciência da Computação
UFRJ

1 de fevereiro de 2016

Section Summary

Bibliografia

1. Adriano Cruz. *Curso de Linguagem* C, Disponível em `http://equipe.nce.ufrj.br/adriano`

Bibliografia

1. Adriano Cruz. *Curso de Linguagem* C, Disponível em `http://equipe.nce.ufrj.br/adriano`
2. Ulysses de Oliveira. *Programando em* C, Editora Ciência Moderna.

O que é?

1. Uma estrutura é um conjunto de uma ou mais variáveis, que podem ser de tipos diferentes, agrupadas sob um único nome.

O que é?

1. Uma estrutura é um conjunto de uma ou mais variáveis, que podem ser de tipos diferentes, agrupadas sob um único nome.
2. Estruturas facilitam a manipulação dos dados nelas armazenados.

O que é?

1. Uma estrutura é um conjunto de uma ou mais variáveis, que podem ser de tipos diferentes, agrupadas sob um único nome.
2. Estruturas facilitam a manipulação dos dados nelas armazenados.
3. Um exemplo poderia ser uma estrutura que armazenasse as diversas informações sobre os alunos de uma Universidade.

O que é?

1. Uma estrutura é um conjunto de uma ou mais variáveis, que podem ser de tipos diferentes, agrupadas sob um único nome.
2. Estruturas facilitam a manipulação dos dados nelas armazenados.
3. Um exemplo poderia ser uma estrutura que armazenasse as diversas informações sobre os alunos de uma Universidade.
4. Nesta estrutura estariam armazenadas, sob o mesmo nome, informações do tipo: nome, registro, data de nascimento, data de ingresso, CPF, etc.

O que é?

1. Uma estrutura é um conjunto de uma ou mais variáveis, que podem ser de tipos diferentes, agrupadas sob um único nome.
2. Estruturas facilitam a manipulação dos dados nelas armazenados.
3. Um exemplo poderia ser uma estrutura que armazenasse as diversas informações sobre os alunos de uma Universidade.
4. Nesta estrutura estariam armazenadas, sob o mesmo nome, informações do tipo: nome, registro, data de nascimento, data de ingresso, CPF, etc.
5. Uma estrutura pode incluir outras estruturas além de variáveis simples.

Informações Demais

Section Summary

Definições Básicas

Estrutura: um conjunto de variáveis, de tipos diversos ou não, agrupadas sob um único nome.

Definições Básicas

Estrutura: um conjunto de variáveis, de tipos diversos ou não, agrupadas sob um único nome.

Membros: Variáveis que compõem a estrutura são os seus membros, elementos ou campos.

Definições Básicas

Estrutura: um conjunto de variáveis, de tipos diversos ou não, agrupadas sob um único nome.

Membros: Variáveis que compõem a estrutura são os seus membros, elementos ou campos.

Relação: Os elementos da estrutura podem ter alguma relação semântica. Por exemplo: alunos de uma universidade, discos de uma coleção, elementos de uma figura geométrica, etc.

Exemplo I

```
struct ALUNO {
    char nome[40];
    int registro;
    int ano_entrada;
    char curso[20];
};
```

Exemplo II

- A palavra chave **struct** inicia a declaração da estrutura, em seguida pode aparecer um identificador (`ALUNO`), que subseqüentemente pode ser usado como abreviação da definição da estrutura.

Exemplo II

- A palavra chave **struct** inicia a declaração da estrutura, em seguida pode aparecer um identificador (`ALUNO`), que subseqüentemente pode ser usado como abreviação da definição da estrutura.
- A declaração continua com a lista de declarações entre chaves e termina com um “;”.

Exemplo II

- A palavra chave **struct** inicia a declaração da estrutura, em seguida pode aparecer um identificador (`ALUNO`), que subseqüentemente pode ser usado como abreviação da definição da estrutura.
- A declaração continua com a lista de declarações entre chaves e termina com um “;”.
- Um membro da estrutura e uma variável não membro da estrutura podem ter o mesmo nome, já que é possível distingui-las por contexto.

Exemplo II

- A palavra chave **struct** inicia a declaração da estrutura, em seguida pode aparecer um identificador (`ALUNO`), que subseqüentemente pode ser usado como abreviação da definição da estrutura.
- A declaração continua com a lista de declarações entre chaves e termina com um ";".
- Um membro da estrutura e uma variável não membro da estrutura podem ter o mesmo nome, já que é possível distingui-las por contexto.
- Para definir estruturas deste tipo podemos usar a seguinte declaração:
  struct `ALUNO paulo, carlos, ana;`

Declarações

- É possível declarar ao mesmo tempo o modelo da estrutura e as variáveis do programa.

Declarações

- É possível declarar ao mesmo tempo o modelo da estrutura e as variáveis do programa.

```
struct ALUNO  {
    char nome[40];
    int registro;
    int ano_entrada;
    char curso[20];
} paulo, carlos, ana;
```

Referenciando elementos

- Para referenciar um elemento da estrutura usa-se o nome da variável do tipo da estrutura seguida de um ponto e do nome do elemento.

Referenciando elementos

- Para referenciar um elemento da estrutura usa-se o nome da variável do tipo da estrutura seguida de um ponto e do nome do elemento.
- `paulo.ano_entrada = 1999;`

Referenciando elementos

- Para referenciar um elemento da estrutura usa-se o nome da variável do tipo da estrutura seguida de um ponto e do nome do elemento.
- `paulo.ano_entrada = 1999;`
- Armazena o ano em que aluno `paulo` entrou na universidade.

Referenciando elementos

- Para referenciar um elemento da estrutura usa-se o nome da variável do tipo da estrutura seguida de um ponto e do nome do elemento.
- `paulo.ano_entrada = 1999;`
- Armazena o ano em que aluno `paulo` entrou na universidade.
- Para ler o nome do curso que paulo cursa pode-se usar o comando `gets(paulo.curso);`

Referenciando elementos

- Para referenciar um elemento da estrutura usa-se o nome da variável do tipo da estrutura seguida de um ponto e do nome do elemento.
- `paulo.ano_entrada = 1999;`
- Armazena o ano em que aluno `paulo` entrou na universidade.
- Para ler o nome do curso que paulo cursa pode-se usar o comando `gets(paulo.curso);`
- `paulo.curso` é um vetor de caracteres.

Estruturas dentro de estruturas

- Estruturas podem conter outras estruturas como membros. Por exemplo, vamos definir uma estrutura para armazenar uma data com a seguinte definição:

Estruturas dentro de estruturas

- Estruturas podem conter outras estruturas como membros. Por exemplo, vamos definir uma estrutura para armazenar uma data com a seguinte definição:

```
struct DATA {
    int dia, mes, ano;
};
```

Estruturas dentro de estruturas

- Agora vamos modificar a estrutura aluno de modo que ela inclua a data de nascimento do aluno.

Estruturas dentro de estruturas

- Agora vamos modificar a estrutura aluno de modo que ela inclua a data de nascimento do aluno.

```
struct aluno {
    char nome[40];
    int registro;
    int ano_entrada;
    char curso[20];
    struct DATA data_nascimento;
};
```

Estruturas dentro de estruturas

- Agora vamos modificar a estrutura aluno de modo que ela inclua a data de nascimento do aluno.

```
struct aluno {
    char nome[40];
    int registro;
    int ano_entrada;
    char curso[20];
    struct DATA data_nascimento;
};
```

- Para se referir ao mês de nascimento de uma variável `paulo` do tipo estrutura `aluno` usamos a declaração
  `paulo.data_nascimento.mes`

Section Summary

Definindo tipos!

- Uma forma mais conveniente de definição de estruturas é possível com o uso do comando **typedef**.

Definindo tipos!

- Uma forma mais conveniente de definição de estruturas é possível com o uso do comando **typedef**.
- Este comando permite dar a um tipo de dados um novo nome.

Definindo tipos!

- Uma forma mais conveniente de definição de estruturas é possível com o uso do comando **typedef**.
- Este comando permite dar a um tipo de dados um novo nome.
- Uma utilidade é aumentar a legibilidade do programa.

Definindo tipos!

- Uma forma mais conveniente de definição de estruturas é possível com o uso do comando **typedef**.
- Este comando permite dar a um tipo de dados um novo nome.
- Uma utilidade é aumentar a legibilidade do programa.
- Por exemplo, é possível usar o seguinte código

```
#define AMARELO 1
typedef int cores;
typedef int laranja;
typedef int manga;
...
laranja lima;
manga espada;
cores = AMARELO;
...
espada++;
```

Section Summary

Simplificando `struct`

- É comum o uso de **typedef** em conjunto com **struct**.

Simplificando `struct`

- É comum o uso de **typedef** em conjunto com **struct**.
- Considere a definição de uma estrutura para guardar horas, minutos e segundos.

Simplificando `struct`

- É comum o uso de **typedef** em conjunto com **struct**.
- Considere a definição de uma estrutura para guardar horas, minutos e segundos.
- Usando esta combinação, a definição é usualmente feita da seguinte maneira:

Simplificando `struct`

- É comum o uso de **typedef** em conjunto com **struct**.
- Considere a definição de uma estrutura para guardar horas, minutos e segundos.
- Usando esta combinação, a definição é usualmente feita da seguinte maneira:

```
struct _TEMPO {
    int hora, minuto, segundo;
};
typedef struct _TEMPO TEMPO;
...
TEMPO t1;
t1.hora = 10; t1.minuto = 15; t1.segundo = 0;
```

Simplificando

- Uma forma ainda mais abreviada, junta as duas definições, ficando a definição da estrutura da seguinte maneira:

Simplificando

- Uma forma ainda mais abreviada, junta as duas definições, ficando a definição da estrutura da seguinte maneira:

```
typedef struct _TEMPO {
    int hora, minuto, segundo;
} TEMPO;
...
TEMPO t1;
```

Mais simples ainda

- É possível dispensar o nome da estrutura (`_TEMPO`) e a definição fica ainda mais simples, com a seguinte forma:

Mais simples ainda

- É possível dispensar o nome da estrutura (_TEMPO) e a definição fica ainda mais simples, com a seguinte forma:

```
typedef struct  {
    int hora, minuto, segundo;
} TEMPO;
...
TEMPO t1;
```

Section Summary

Atribuições entre estruturas

```
#include <stdio.h>

typedef struct _EMPREGADO {
    char nome[40];
    float salario;
} EMPREGADO;

int main () {
    EMPREGADO temp, emp1;

    puts("Entre com nome.");
    gets(emp1.nome);
    puts("Qual o salario?"); scanf("%f", &emp1.↩
       salario);
    temp=emp1;
    printf("O salario de %s e %.2f\n",
        temp.nome, temp.salario);
    return 0;
}
```

Simplificando

- É possível atribuir o conteúdo de uma estrutura a outra estrutura do mesmo tipo, não sendo necessário atribuir elemento por elemento da estrutura.

Simplificando

- É possível atribuir o conteúdo de uma estrutura a outra estrutura do mesmo tipo, não sendo necessário atribuir elemento por elemento da estrutura.
- Esta é uma das grandes vantagens de estruturas já que o tamanho do código é reduzido e a clareza dos programas aumenta.

Simplificando

- É possível atribuir o conteúdo de uma estrutura a outra estrutura do mesmo tipo, não sendo necessário atribuir elemento por elemento da estrutura.
- Esta é uma das grandes vantagens de estruturas já que o tamanho do código é reduzido e a clareza dos programas aumenta.
- O comando `temp = emp1;` faz com que todos os dados armazenados na estrutura `emp1` sejam transferidos para a estrutura `temp`.

Section Summary

Matrizes de Estruturas

Estruturas aparecem freqüentemente na forma de matrizes. Por exemplo, a declaração `ALUNO turma[100];` define uma matriz de 100 estruturas do tipo **struct** `ALUNO`.

Exemplo de Matriz de Estrutura I

```c
#include <stdio.h>
#include <string.h>

typedef struct _ALUNO {
    char nome[40];
    float n1, n2, media;
} ALUNO;
```

Exemplo de Matriz de Estrutura II

```
int main (void) {
  ALUNO turma[4], temp;
  int jaOrdenados = 0, foraOrdem, i;

  for (i = 0; i < 4; i++) {
      gets(turma[i].nome);
      scanf("%f", &turma[i].n1);
      do {} while (getchar()!='\n');
      scanf("%f", &turma[i].n2);
      do {} while (getchar()!='\n');
      turma[i].media=(turma[i].n1+turma[i].n2)/2.0;
    }
```

Exemplo de Matriz de Estrutura III

```
  do {
      foraOrdem = 0;
      for (i = 0; i < 4 - 1 - jaOrdenados; i++) {
          if (turma[i].media > turma[i+1].media) {
              temp = turma[i];
              turma[i] = turma[i+1];
              turma[i+1] = temp; foraOrdem = 1;
          }
      }
      jaOrdenados++;
  } while (foraOrdem);
  for (i=0; i<4; i++) {
      printf("\nDados do aluno %d\n", i);
      printf("%s: %0.1f, %0.1f, %0.1f\n",
        turma[i].nome, turma[i].n1, turma[i].n2,
        turma[i].media);   }
  return 0;
}
```

Section Summary

Passando elementos da estrutura

Primeiro vamos considerar o caso de passar elementos da estrutura para funções.

Passando elementos da estrutura

Passando elementos da estrutura

Primeiro vamos considerar o caso de passar elementos da estrutura para funções.

Caso os elementos da estrutura sejam variáveis de um dos tipos já vistos, a passagem é efetuada da maneira normal.

Exemplo

```c
#include <stdio.h>

typedef struct _CIRCULO {
    float x, y, raio;
} CIRCULO;

float Area (float r) {
    return 3.141516 * r * r;
}

int main (void) {
    CIRCULO c;
    c.x = c.y = c.raio = 1.0;
    printf("%f\n", Area(c.raio));
    return 0;
}
```

Passando estruturas

- Estruturas, quando passadas para funções, se comportam da mesma maneira que as variáveis dos tipos que já estudamos.

Passando estruturas

- Estruturas, quando passadas para funções, se comportam da mesma maneira que as variáveis dos tipos que já estudamos.
- Ao passar uma estrutura para uma função estaremos passando os valores armazenados nos membros da estrutura.

Passando estruturas

- Estruturas, quando passadas para funções, se comportam da mesma maneira que as variáveis dos tipos que já estudamos.
- Ao passar uma estrutura para uma função estaremos passando os valores armazenados nos membros da estrutura.
- Como este tipo de passagem é feito por valor, alterações nos membros da estrutura não modificam os valores da estrutura na função que chamou.

Exemplo

```
#include<stdio.h>
#include<math.h>
typedef struct _PONTO {
    float x, y;
} PONTO;

float comp(PONTO p1, PONTO p2) {
    return sqrt(pow(p2.x-p1.x,2)+pow(p2.y-p1.y,2));
}
int main (void) {
  PONTO p1, p2;

  printf("x1 = ? "); scanf("%f", &p1.x);
  printf("y1 = ? "); scanf("%f", &p1.y);
  printf("x2 = ? "); scanf("%f", &p2.x);
  printf("y2 = ? "); scanf("%f", &p2.y);
  printf("Comprimento da reta = %f\n", comp(p1, p2))↩
      ;
  return 0;
```

Passando vetores de estruturas

Passando vetores de estruturas

A passagem de vetores de estruturas é similar a passagem de vetores de variáveis simples.

Passando vetores de estruturas

Passando vetores de estruturas

A passagem de vetores de estruturas é similar a passagem de vetores de variáveis simples.

Exemplo

Para ilustrar a passagem de vetores de estruturas para funções iremos, no exemplo anterior, substituir o trecho que ordena o vetor de alunos por uma função.

Passando vetores de estruturas

Passando vetores de estruturas

A passagem de vetores de estruturas é similar a passagem de vetores de variáveis simples.

Exemplo

Para ilustrar a passagem de vetores de estruturas para funções iremos, no exemplo anterior, substituir o trecho que ordena o vetor de alunos por uma função.

Exemplo

```
void Ordena(ALUNO turma[], int tam) {
    int i, foraOrdem, jaOrdenados = 0;
    ALUNO temp;
    do {
        foraOrdem = 0;
        for (i = 0; i < 4 - 1 - jaOrdenados; i++) {
            if (turma[i].media > turma[i+1].media) {
                temp = turma[i];
                turma[i] = turma[i+1];
                turma[i+1] = temp;
                foraOrdem = 1;
            }
        }
        jaOrdenados++;
    } while (foraOrdem);
}
```

Section Summary

Ponteiros para estruturas

Ponteiros para estruturas

Para definir ponteiros para estruturas a declaração é similar a declaração de um ponteiro normal.

```
struct aluno {
    char nome[40];
    int ano_entrada;
   float n1, n2, media;
} *ze;
```

Passando estruturas

- Ponteiros são uteis quando passamos estruturas para funções.

Passando estruturas

- Ponteiros são uteis quando passamos estruturas para funções.
- Ao passar apenas o ponteiro para estrutura economizamos tempo e memória.

Passando estruturas

- Ponteiros são uteis quando passamos estruturas para funções.
- Ao passar apenas o ponteiro para estrutura economizamos tempo e memória.
- O espaço de memória, é economizado por que se evita passar os dados que compõem a estrutura um por um.

Passando estruturas

- Ponteiros são uteis quando passamos estruturas para funções.
- Ao passar apenas o ponteiro para estrutura economizamos tempo e memória.
- O espaço de memória, é economizado por que se evita passar os dados que compõem a estrutura um por um.
- O tempo é economizado porque não é necessário gastar o tempo de empilhar e desempilhar todos os elementos da estrutura no processo de passagem para a função.

Passando estruturas

- Ponteiros são uteis quando passamos estruturas para funções.
- Ao passar apenas o ponteiro para estrutura economizamos tempo e memória.
- O espaço de memória, é economizado por que se evita passar os dados que compõem a estrutura um por um.
- O tempo é economizado porque não é necessário gastar o tempo de empilhar e desempilhar todos os elementos da estrutura no processo de passagem para a função.
- Para acessar elementos da estrutura apontada por um ponteiro usa-se o chamado operador seta (->).

Passando estruturas

- Ponteiros são uteis quando passamos estruturas para funções.
- Ao passar apenas o ponteiro para estrutura economizamos tempo e memória.
- O espaço de memória, é economizado por que se evita passar os dados que compõem a estrutura um por um.
- O tempo é economizado porque não é necessário gastar o tempo de empilhar e desempilhar todos os elementos da estrutura no processo de passagem para a função.
- Para acessar elementos da estrutura apontada por um ponteiro usa-se o chamado operador seta (->).
- Por exemplo para imprimir a média do aluno ze usaríamos o comando

```
printf ("A media vale %.1f", ze->media);
```

Alocando espaço para estruturas

```c
#include <stdio.h>

typedef struct _ALUNO {
    char nome[40];
    float n1, n2, media;
} ALUNO;

int main (void) {
    ALUNO *maria;

    maria = (ALUNO *) malloc (sizeof(ALUNO));
    if (!maria) exit(1);

    gets(maria->nome);
    scanf("%f %f", &(maria->n1), &(maria->n2));
    maria->media = (maria->n1 + maria->n2) / 2;
    printf("A media de %s vale %0.2f\n", maria->nome↩
        , maria->media);
    return 0;
```

Alocando espaço para vetores de estruturas I

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct _func {
    char nome[40];
    float salario;
} Tfunc ;

void le (Tfunc *cadastro, int funcionarios) {
    int i;
    char linha[40];
    for (i=0; i<funcionarios; i++) {
      fgets((cadastro+i)->nome, 39, stdin);
      puts ("Salario ?"); fgets(linha, 39, stdin);
      sscanf(linha, "%f", &((cadastro+i)->salario));
    }
}
```

Alocando espaço para vetores de estruturas II

```
float media(Tfunc *cadastro, int funcionarios) {
    float media=0.0;
    int i;

    for (i=0; i<funcionarios; i++) {
        media += (cadastro+i)->salario;
    }
    return media /= funcionarios;
}
```

Alocando espaço para vetores de estruturas III

```
int main (void) {
    Tfunc *cadastro;
    int funcionarios;
    char linha[40];
    puts("Quantos funcionarios?");
    fgets(linha, 39, stdin);
    sscanf (linha, "%d", &funcionarios);
    cadastro = (Tfunc *)
           malloc(funcionarios * sizeof (Tfunc));
    if (!cadastro) exit (1);
    le(cadastro, funcionarios);
    printf("Salario medio = %.2f\n",
            media(cadastro, funcionarios));
    return 0;
}
```

The End