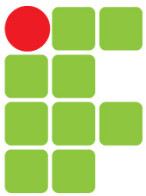
Estruturas e tipos abstratos de dados

Programação de computadores II

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

8 de março de 2017



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Tópicos da aula

Tópico

Objetivos

- Aprender a utilizar novos tipos de dados
- Criar dados personalizados
- Utilizar o conceito de “dados abstratos”

Tipos dados

A Linguagem C permite que o programador crie formas adicionais para manipulação de dados:

Tipos padrões de dados

int, char, long, float, ...

Estruturas (*struct*)

Estruturas são aglomerados de dados reunidos pelo um mesmo nome

Campos de bit (bit field)

Tipo especial de estrutura que permite acesso individual de bits

Tipos dados

União (union)

Tipo especial de estrutura que a mesma porção de memória seja compartilhada por dois ou mais membros

Enumeração (enum)

Lista numerada de símbolos

Definição de novo tipo (*typedef*)

Cria um novo tipo de dado, ou renomeia um já existente

Tópico

Estruturas

- Estruturas são uma coleção de dados reunidos pelo um mesmo nome
- Permite integrar, agrupar e relacionar dados logicamente
- Exemplo: Endereço de um cidadão
  - nome;
  - rua;
  - cidade;
  - estado;
  - CEP;

Estruturas

- Estruturas são uma coleção de dados reunidos pelo um mesmo nome
- Permite integrar, agrupar e relacionar dados logicamente
- Exemplo: Endereço de um cidadão
  - nome;
  - rua;
  - cidade;
  - estado;
  - CEP;
- Qual a diferença entre um array e uma estrutura? Ambos não agrupam dados?

Estruturas

Declaração de estruturas:

```
struct nome_estrutura
{
  tipo nome_membro;
  tipo nome_membro;
};
```

Obs: a declaração não é uso

Estruturas

Declaração de estruturas com possível utilização:

```
struct <nome_estrutura>
{
  tipo <nome_membro_1>;
  tipo <nome_membro_2>;
}  <nome_variavel_deste_tipo>;
```

Para acessar cada membro, utiliza-se o operador “.”:

```
<nome_variavel_deste_tipo>.<nome_membro_1> = <numero, texto, ... > ;
```

Exemplo – Endereço

```
struct endereco
{
  char nome[30];
  char rua[50];
  char cidade[20];
  char estado[3]
  unsigned int cep;
};
```

```
struct endereco end1;

end1.nome = "Renan";
end1.rua = "Mauro Ramos";
end1.cidade = "Florianopolis";
```

Como fica na memória?

Exemplo – Ponto espaço 3D

```c
struct Point3D
{
  int x;
  int y;
  int z;
};
```

```c
struct Point3D meu_ponto = {3, 1, 0};

printf("x: %d\n", meu_ponto.x);
```

Atribuição

- Pode-se copiar e reatribuir valores com o operador "="
- **Ponteiros: copia-se o endereço não o dado**
- Todos os campos são copiados, em array deve-se faze item por item

```
struct Point3D a = {3, 1, 0};

struct Point3D b = {5, 8, -1};


b = a;  // b recebeu 3 em x, 1 em y e 0 em z
```

Exemplo – struct global

```c
#include <stdio.h>

struct {
   int a;
   int b;
} x, y;

void main(void)  {

   x.a = 10;

   y = x;

   printf("%d\n", y.a);
}
```

Exemplo – struct local

```c
#include <stdio.h>

void main(void)  {

   struct {
      int a;
      int b;
      } x, y;

   x.a = 10;

   y = x;

   printf("%d\n", y.a);
}
```

Exemplo – struct tipo global, declaração local

```c
#include <stdio.h>

struct p{
  int a;
  int b;
};

void main(void)  {

  struct p x;
  struct p y;

  x.a = 10;

  y = x;

  printf("%d\n", y.a);
}
```

Exemplo – struct funções

```c
#include <stdio.h>

struct p { //declaracao de p
    int a;
    int b;
  };

int sum(struct p p1, struct p p2);  //header de sum

void main(void)  {

   struct p x;  //variaveis tipo struct p
   struct p y;

   x.a = 10;

   y = x;

   printf("%d\n", y.a);

   printf("%d\n", sum(x,y));

}

int sum(struct p p1, struct p p2) { return p1.a + p2.a; } //implementacao
```

Exemplo – struct funções com ponteiros

```c
#include <stdio.h>

struct p {
  int a;
  int b;
};

int sum(struct p *p1, struct p *p2);

void main(void)  {

  struct p x;
  struct p y;

  x.a = 10;

  y = x;

  printf("%d\n", y.a);

  printf("%d\n", sum(&x,&y));

}

int sum(struct p *p1, struct p *p2) { return p1->a + p2->a; }
```

Exemplo – struct vetores

```c
#include <stdio.h>

struct p {
  int x;
  int y;
};


void main(void)  {

  int i = 0;

  struct p pontos[10];

  for (i=0; i < 10; i++) {
      pontos[i].x = rand();
      pontos[i].y = rand();
  }

  for (i=0; i < 10; i++) {
      printf("x: %d\n", pontos[i].x);
      printf("y: %d\n ————————\n", pontos[i].y);
  }

}
```

Tópico

Campos de bit

Campos de bit (bit field)

Tipo especial de estrutura que permite acesso individual de bits

Bit fields permitem

- economizar memória
- definir tamanhos personalizados de dados
- operações aritméticas corretas com dados não padrões

Declaração

```
struct status {
  unsigned     :    4; // campo de 4 bits sem sinal
  unsigned cts :    1; // campo de 1 bit
  unsigned dsr :    1; // campo de 1 bit
};
```

Observação quanto a fragmentação de memória: tamanho total da estrutura será múltipla de bytes

Exemplo e operador sizeof

```c
#include <stdio.h>

struct example1
{
    int isMemoryAllocated;
    int isObjectAllocated;
};

struct example2
{
    int isMemoryAllocated : 1;
    int isObjectAllocated : 1;
};

int main(void)
{
    printf("\n sizeof example1 eh [%u] bytes, sizeof example2 eh [%u] bytes\n",
           sizeof(struct example1), sizeof(struct example2));

    return 0;
}
```

```
   sizeof example1 eh [8] bytes, sizeof example2 eh [4] bytes
```

Tópico

Uniões

União (union)

Tipo especial de estrutura que a mesma porção de memória seja compartilhada por dois ou mais membros

- uniões são quase como estruturas
- **mas**, o tamanho da união é igual ao tamanho do seu maior membro, não a soma deles

Exemplo e operador sizeof

```
struct str_char_and_ascii
{
   char ch;
   unsigned int ascii_val;
};


union un_char_and_ascii
{
   char ch;
   unsigned int ascii_val;
};
```

Tamanhos:

- struct: sizeof(struct st_char_and_ascii) = sizeof(char) + sizeof(int) = 1 + 4 = 5 bytes
- union: sizeof(union un_char_and_ascii) = 4 bytes

Exemplo union

```c
#include <stdio.h>

union char_and_ascii
{
    char ch;
    unsigned short ascii_val;
};

int main (void)
{
    union char_and_ascii obj;
    obj.ascii_val = 0;

    obj.ch = 'A';

    printf("\n character = [%c], ascii_value = [%u]\n", obj.ch, obj.ascii_val);

    return 0;
}
```

```
  character = [A], ascii_value = [65]
```

Exemplo union

```c
#include <stdio.h>

union pw
{
    int i;
    char byte[4];
};
```

O que posso fazer com esta union?

Tópico

Enumerações

Enumeração (enum)

Lista numerada de símbolos

- Tipo definido pelo usuário contando uma lista de constantes com NOME DEFINIDO

```
enum tipo_nome { valor1, valor2, ..., valorN };
```

Exemplo enum

```c
#include <stdio.h>

// com definicao de numeros
enum cardsuit {
    CLUBS    = 1,
    DIAMONDS = 2,
    HEARTS   = 4,
    SPADES   = 8,
    FIRST = 10,
    SECOND,     //igual a 11
    THRID,        //igual a 12
};

enum semana{ domingo, segunda, terca, quarta, quinta, sexta, sabado};

int main(){
    enum semana hoje;

    hoje=segunda;

    printf("%d dia",hoje+1);

    return 0;
}
```

Tópico

Definição de novo tipo (*typedef*)

Cria um novo tipo de dado, ou renomeia um já existente

- typedef é uma palavra reservada
- utilizada para dar um nome ou criar um tipo personalizado de dado

```
typedef <tipo> <novo_nome>
```

Exemplo typedef

```c
typedef unsigned char byte;
//ou
typedef unsigned char BYTE;


int main(void)
{
  BYTE  b1, b2;
  //ou
  byte b1, b2;
  ...
```

Exemplo typedef

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

typedef struct Livros {
   char titulo[50];
   char autor[50];
   char assunto[100];
   int livro_id;
} Livro;

int main( ) {

   Livro livro_01;

   strcpy( livro_01.titulo, "Programacao C");
   strcpy( livro_01.autor, "Nuha Ali");
   strcpy( livro_01.assunto, "Tutorial de Programacao C");
   livro_01.livro_id = 6495407;

   printf( "Titulo : %s\n", livro_01.titulo);
   printf( "Autor : %s\n", livro_01.autor);
   printf( "Assunto : %s\n", livro_01.assunto);
   printf( "ID : %d\n", livro_01.livro_id);

   return 0;
}
```

Exercício

- Considerando o conteúdo sobre estruturas de dados, alocação dinâmica e as implementações anteriores:
  - Obter velhinos.csv do **Moodle**.
  - Criar a estrutura que comporte o dados deste arquivo (utilize typedef).
  - Criar uma função que leia o conteúdo deste arquivo e armazene em um vetor de estruturas alocado dinamicamente. Deve-se retornar um ponteiro do vetor de estruturas e também o número de dados lidos.

```
struct pessoa {
  char *nome;
  int idade;
  char sexo;
};
```

  - Criar uma função que busque determinado nome e retorne a idade e o sexo
  - **Requisitos:**
    - As funções devem ser implementadas em um arquivo separado do **main.c**: pessoa.c pessoa.h
    - Utilize a macro que evita multi-inclusão de arquivos de cabeçalhos