Aula 26 – Estruturas de Dados Heterogêneas - struct

Algoritmos e Programação de Computadores
Profs: Ronaldo Castro de Oliveira – ronaldo.co@ufu.br
Anilton Joaquim da Silva – anilton@ufu.br

Introdução

- Imagine que você deseje armazenar informações sobre um **funcionário** de uma empresa. Entre as informações necessárias tem-se por exemplo, nome, endereço, telefone, sexo, estado civil, cargo, setor e salário. Em um programa, isto seria representado por oito variáveis apenas para um funcionário. Se você desejar incluir mais informações, isto implicará em mais variáveis, aumentando a complexidade do programa. Em muitas aplicações, é importante a capacidade de tratar todas as informações de uma entidade (uma pessoa, por exemplo), como sendo uma única unidade de armazenamento, ou seja, uma única variável, mas que permita acesso a cada informação em separado. Neste tipo de informação não há homogeneidade quanto ao tipo de dados tratado. Por isso, deve haver um mecanismo para trabalhar com variáveis estruturadas heterogêneas, também conhecidas simplesmente como **estruturas** ou **structs**.
- Sintaxe:

```
struct identificador{
        tipo_campo_1   nome_campo_1;
        tipo_campo_2   nome_campo_2;
        ...
        tipo_campo_n   nome_campo_n;
        } variáveis_estrutura;
```

Exemplos

```
struct endereco
{
    char rua[30];
    int numero;
    char complemento[15];
    char bairro[30];
    char cidade[30];
    char estado[3];
    char  CEP[12];
};
...
struct endereco  end1, end2, end3;
```

```
struct funcionario
{
    char nome[30];
    struct endereco  end;
    char telefone [20];
    char sexo;
    char estadoCivil[20];
    char cargo[30];
    char  setor[30];
    float  salario;
};
```

OBS: a definição de uma struct é a mesma coisa que definir um novo tipo de dado, onde pode-se criar novas variáveis deste tipo.

typedef

- A palavra chave typedef permite que se defina novos tipos de dados a partir de tipos já existentes. Por exemplo, é possível definir um tipo “numero_real” ou um tipo “caractere”. A sintaxe para o uso typedef é a seguinte:

  typedef tipo_antigo tipo_novo;

- Os exemplos apenas dados são implementados assim:

  typedef float numero_real;

  typedef char caractere;

- Este recurso me permite especificar uma variável de valor real usando sem usar o termo float:

  numero_real PI = 3.1415;

  caracter nome[30];

Struct – forma compacta

```
typedef   struct{
    char rua[30];
    int numero;
    char complemento[15];
    char bairro[30];
    char cidade[30],;
    char estado[3];
    char  CEP[12];
} Tendereco ;
```

```
typedef  struct {
    char nome[30];
    Tendereco  end;
    char telefone [20];
    char sexo;
    char estadoCivil[20];
    char cargo[30];
    char  setor[30];
    float  salario;
} Tfuncionario;
```

Declarando a estrutura:

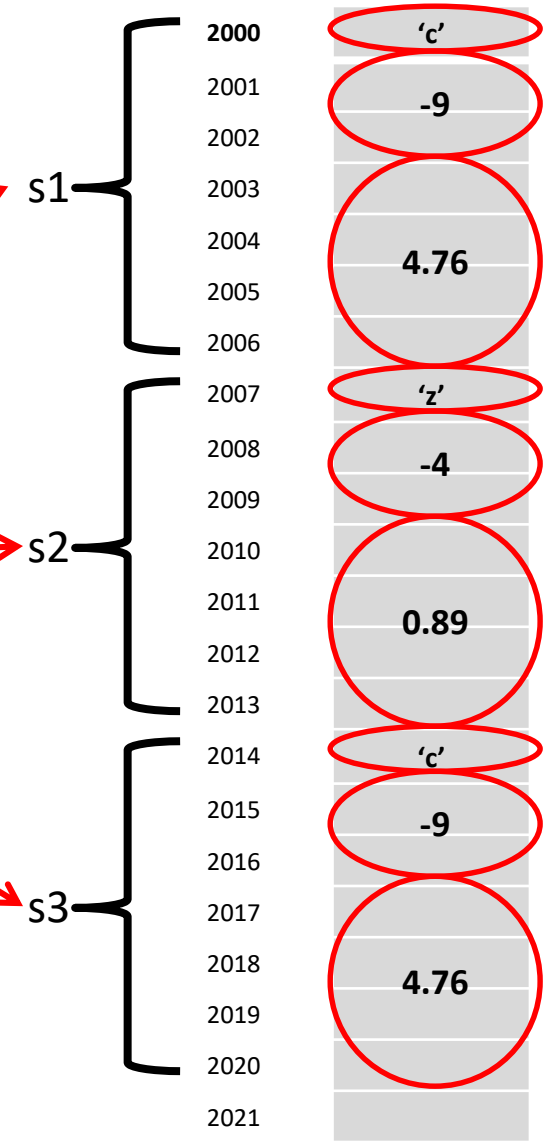
```
Tfuncionario  FUNC;
Tendereco   E1, E2, E3;
```

Struct - Acesso aos campos

- Embora as estruturas agreguem vários campos, com raras exceções, o manuseio da mesma deve ser feito campo a campo, como variáveis normais. Para acessar um dos campos de uma estrutura, usa-se o operador ’.’ (ponto). Diferentemente de variáveis indexadas, variáveis do tipo estrutura podem receber o valor de outra do mesmo tipo (desde que nenhum dos campos seja vetor).

Struct - Acesso aos campos

```
typedef struct {
        char L;
        int I;
        float B;
        } estrutura;
...
estrutura  s1 = {'c', -9, 4.76},
//Atribuicao na iniciacao.
estrutura  s2,  s3;
s2.L = 'z';          //Atribuicao...
s2.I = -4;           //... campo a ...
s2.B = 0.89;         //... campo.
s3 = s1;             //Copia de valores
                     //de todos os campos.
...
```



Exemplo:

```
#include <stdio.h>
#include <string.h>
typedef struct
{
      char rua [50];
      int numero;
      char bairro [20];
      char cidade [30];
      char estado [3];
      long int CEP;
} tipo_endereco;

typedef struct l
{
      char nome [50];
      long int telefone;
      tipo_endereco endereco;
} ficha_pessoa;
```

```
int main ()
{
      ficha_pessoa  ficha;

      strcpy (ficha.nome,"Luiz Osvaldo Silva");
      ficha.telefone=32341234;
      strcpy (ficha.endereco.rua,"Rua Flores");
      ficha.endereco.numero=10;
      strcpy (ficha.endereco.bairro,”Vila Velha");
      strcpy (ficha.endereco.cidade,"Belo Horizonte");
      strcpy (ficha.endereco.estado,"MG");
      ficha.endereco.CEP=31340230;

      return 0;
}
```

Matrizes de estruturas

- Um estrutura é como qualquer outro tipo de dado no C. Podemos, portanto, criar vetores e matrizes de estruturas. Vamos ver como ficaria a declaração de um vetor de 100 funcionários:

  T**funcionario FUNC [100];**

- Podemos então acessar os dados do terceiro funcionário com: FUNC[2], da mesma forma que acessamos um vetor:

  Atribuindo dados:

  strcpy (FUNC[2].nome, "Jose da Silva");

  FUNC[2].sexo = 'M';

  Lendo do teclado:

  cin.getline (FUNC[2].end.rua, 30);

  cin >> FUNC[2].salario;

- Podemos acessar a segunda letra da sigla de estado do décimo terceiro funcionário fazendo:

  FUNC[12].end.estado[1];

Exemplo: números complexos

```
#include <iostream>
using namespace std;

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

int main()
{
	int i, j;
	complexo A[2][3] = {
				{ {1.0, -0.1}, {2.0, -0.2}, {2.0, -0.2} },
				{ {4.0, -3.4}, {5.0, 4.1}, {6.0, -2.6} } };
	for ( i= 0; i < 2; i++)
	{
		for (j = 0; j < 3; j++)
		{
				cout << A[i][j].real << " + " << A[i][j].imag << "*i" << "   ";
		}
		cout << endl;
	}
	return 0;
}
```

Estruturas e funções

- Estruturas são utilizadas em funções da mesma forma que os tipos básicos que vimos (int, float, char, . . .).

```
#include<iostream>
#include <stdio.h>
#define tam 100

using namespace std;

typedef struct
{
    char  nome[40];
    int  idade;
    float  salario;
} pessoa;

void  le_pessoa(pessoa P[], int N)
{
   for (int i=0; i<N; i++)
   {
      cout << "Digite dados da pessoa "<<i+1<<": "<<endl;
      cout << "Nome: ";
      fflush(stdin);
      cin.getline (P[i].nome, 40);
      cout << "Idade: ";
      cin >> P[i].idade;
      cout << "Salario: ";
      cin >> P[i].salario;

   }
}
```

```
pessoa maisVelho(pessoa P[ ],int N)
{
   pessoa velho = P[0];
   for (int i=1; i<N; i++)
   {
      if (P[i].idade > velho.idade)
         velho = P[i];
    }
   return velho;
}
int main()
{
   pessoa PES[tam], mv;
   int N;
   cout << "Controle de pessoas"<<endl;
   cout << "Digite numero de pessoas: ";
   cin >> N;
   cout << endl <<"Digite os dados de cada pessoas: ";
   le_pessoa(PES, N);
   mv = maisVelho(PES, N);
   cout << "O mais velho eh " << mv.nome <<", com "<< mv.idade
      << " anos e salario de " << mv.salario << endl;
   return 0;
}
```

Exercício 1

- Faça um programa que leia informações sobre 15 pessoas. Essa informação deve ficar em um vetor de variáveis do tipo estruturado pessoa, o qual deve conter as seguintes informações:
  - Nome: string de tamanho 30;
  - Sexo: tipo enumerativo com os valores masc, fem;
  - Idade: valor inteiro;
  - Estado Civil: tipo enumerativo com os valores solteiro, casado, separado,
  - viúvo.
  - Salário: valor real.
- Em seguida, imprima o número de homens, número de mulheres e informações da pessoa com maior salário.

Exercício 1

```
#include <iostream>
#include <iomanip>
#include <stdio.h>
#define tam 100
using namespace std;

typedef struct {
	char nome[30];
	char sexo[5];
	int idade;
	char estadoCivil[10];
	float salario;
} pessoa;

void le_pessoas(pessoa P[], int N)
{
	for (int i=0; i<N; i++)
	{
		cout << "Pessoa "<<i+1<<": "<<endl;
		cout << "Nome: ";
		 fflush(stdin);  cin.getline (P[i].nome, 30);
		cout << "Sexo: ";
		fflush(stdin); cin.getline (P[i].sexo, 5);
		cout << "Idade: ";
		cin >> P[i].idade;
		cout << "Estato civil: ";
		fflush(stdin); cin.getline (P[i].estadoCivil, 10);
		cout << "Salario: ";
		cin >> P[i].salario;
	}
}

void mostra_pessoas(pessoa P[], int N)
{
	//cabeçalho da impressao
	cout << setw (7) << left <<"Pessoa";
	cout << setw (31) << left << "Nome";
	cout << setw (6) << left << "Sexo";
	cout << setw (6) << left << "Idade";
	cout << setw (13) << left << "Estato civil";
	cout << setw (15) << left << "Salario" <<endl;
	for (int i=0; i<N; i++)
	{
		cout << setw (7) << left << i+1;
		cout << setw (31) << left << P[i].nome;
		cout << setw (6) << left << P[i].sexo;
		cout << setw (6) << left << P[i].idade;
		cout << setw (13) << left << P[i].estadoCivil;
		cout << setw (15) << left << P[i].salario <<endl;
	}
}

int main()
{
	int N;
	pessoa PES[tam];
	cout << "Programa para controlar N pessoas" <<endl;
	cout << "Digitar o numero de pessoas: ";
	cin >> N;

	cout << "Entrar com dados das pessoas: "<< endl;
	le_pessoas(PES, N);
	cout << endl << "Pessoas cadastradas no sistema: " <<endl;
	mostra_pessoas (PES, N);
}
```

Variáveis tipo referência

- Agora que já conhecemos vetores e matrizes, usaremos uma analogia mais precisa para entendermos **referências**, ou como são mais popularmente conhecidas, os famigerados **ponteiros**.
- A memória é um grande vetor: Exemplos:
  - **char C;** → o compilador separa um byte da memória (Ex. byte 7542) , e passa a chamá-lo de C. Assim, toda vez que se referir à variável C em seu programa, o computador sabe que está se referindo ao byte 7542.
  - **float Z;** → o compilador reserva quatro bytes para você, digamos, os bytes 8012, 8013, 8014 e 8015. Tal variável tem passa a ter o endereço 8012
  - **int V[10];** → o compilador reserva um espaço de tamanho N x tipo_inteiro, que neste caso o vetor terá tamanho 20 bytes. O endereço desta variável é o endereço do primeiro byte da primeira célula. Assim, quando você, por exemplo, acessa a posição 5 de um vetor de inteiros cujo endereço é 1000, o computador faz o seguinte cálculo para determinar a posição de memória que você está acessando, sabendo que cada inteiro ocupa 2 bytes: 1000 + 5 * 2 = 1010.
  - **float M[4][3];** → similarmente o compilador reserva L x C x tipo_float, que no caso a matriz terá um tamanho de 48 bytes.

Variáveis tipo referência

- Em certas situações, referir a posições de memória pelo seus endereços em vez de por um nome de variável é muito útil.
- Por exemplo, imagine que precisemos calcular uma série de estatísticas sobre os dados em um vetor. Neste caso, gostaríamos de definir uma função que analisasse o vetor e desse como resultados a média, o desvio padrão, a variância, o intervalo de confiança etc. Entretanto funções em C(++) retornam um único resultado. Se, contudo, pudéssemos passar como parâmetro para a função as referências de onde gostaríamos que os resultados fossem colocados, então a função poderia colocá-los diretamente nestas posições de memória. Esse mecanismo é o que chamamos de **passagem de parâmetro por referência**.
- Declaração de uma variável tipo referência usando o operadore **'*':**

  tipo_dado *identificador;
- Assim, para se declarar variáveis de referência para uma posições de memória que armazenam um inteiro, um caractere, e um float, você usaria as seguintes declarações:

  int *ref_int;

  char *ref_char;

  float *ref_float;

Variáveis tipo referência

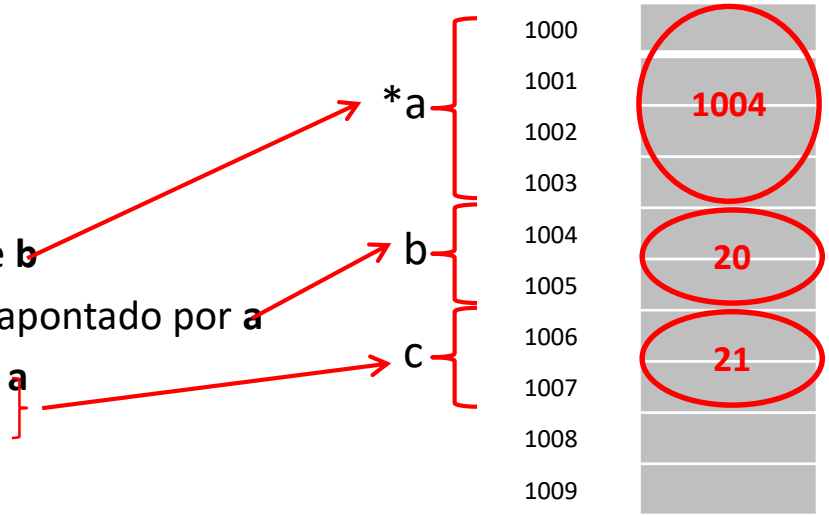
- Alem do operador **'*'** que define uma variável do tipo referência, podemos usar o operador **'&'** que se extraia o endereço (ou uma referência) de uma variável. Exemplo:

```
int *a;
int b;
...
a = &b;      // a passa a conter o endereço onde a
             //variável b foi alocada na memória
```

- **"Em raríssimas situações é necessário saber o endereço de uma variável. Na maior parte dos casos, basta saber que uma variável tem tal endereço."**
- Finalmente, o operador '*' também nos permite acessar o conteúdo de uma variável referenciada, em do valor da referência. Exemplo:

```
int *a;
int b, c;

a = &b;      // a passa a conter o endereço de b
*a = 20;     // o valor 20 é atribuído ao valor apontado por a
c = *a;      //  c recebe o valor apontado por a
c++;         // a variável c é incrementada
```



Passagem de Referências como Parâmetros

- Variáveis do tipo referência, como outros tipos de variáveis, podem ser passadas como parâmetro em chamadas de função. A grande vantagem desta técnica, também conhecida como passagem de parâmetros por referência, é a possibilidade de modificação da memória para qual a referência foi passada. Exemplo:

```
#include<cmath>

void sen_cos_tan(float val, float *seno, float *cosseno, float *tangente)
{
        *seno = sin(val);
        *cosseno = cos(val);
        *tangente = tan(val);
}

int main()
{
        float v = 0, s, c, t;

        sen_cos_tan(v, &s, &c, &t);
        cout << "seno: " << s << " cosseno: " << c << " tangente: "<< t;

        return 0;
 }
```

Exemplo

- Programa de uma loja de livros com as funções **novo_livro** que le os dados de um livro e armazena na memória e **lista_livros** que lista todos os livros cadastrados. Usar estruturas, ponteiros e menu de opções com switch – case.

```
#include <iostream>
#include <iomanip>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#define MAX 100

using namespace std;

typedef struct     //Declaração da Estrutura Livro
{
   char  TITULO[30];
   float PRECO;
} TLIVRO;

char MENU()       // Função Menu de Opções
{
   char C;
   cout <<"Escolha a Opcao Desejada:"<<endl;
   cout <<"1 - Inserir Novo Livro"<<endl;
   cout <<"2 - Listar Livros Cadastrados"<<endl;
   //cout <<"3 - Consultar Preco do Livro"<<endl;
   //cout <<"4 - Ordenar Livros por Titulo"<<endl;
   //cout <<"5 - Ajusta precos dos livros."<<endl;
   cout <<"6 - Sair do Programa"<<endl;
   C = getch();
   return(C);
}
```

```
//Função de cadastro deum livro
void LELIVRO(TLIVRO LIV[MAX], int *QT)
{
   cout <<endl<< "Digite os dados do livro " << *QT << ": "<<endl;
   cout <<"Titulo do Livro: ";
   fflush(stdin);
   cin.getline (LIV[*QT].TITULO, 30);
   cout <<"Preco do livro: ";
   cin >> LIV[*QT].PRECO;
   *QT = *QT + 1;
}

//Função para listar os livros cadastrados
void LISTALIVRO(TLIVRO LIV[MAX], int QT)
{
   cout <<endl<<"Lista de Livros cadastrados:"<<endl;
   cout <<"-------------------------------------------------"<<endl;
   cout <<"|Titulo do Livro               |Preco do Livro |"<<endl;
   cout <<"|------------------------------|---------------|"<<endl;
   for (int i=0; i<QT; i++)
   {
      cout << "| ";
      cout << setw(30) << left << LIV[i].TITULO;
      cout << " | ";
      cout << setw(14)<<left<<LIV[i].PRECO << "|" <<endl;
   }
   cout <<"-------------------------------------------------"<<endl<<endl;
}
```

Exemplo – CONTINUAÇÃO

- Programa de uma loja de livros com as funções **novo_livro** que le os dados de um livro e armazena na memória e **lista_livros** que lista todos os livros cadastrados. Usar estruturas, ponteiros e menu de opções com switch – case.

```
//Programa Principal
int main()
{
   TLIVRO LIVRO[MAX];
   int N=0;
   char OP;
   do
   {
      cout <<endl <<"-----<<<<<  Loja de Livros  >>>>>-----"<<endl;
      OP = MENU();
      switch(OP)
      {
         case '1': LELIVRO(LIVRO, &N); break;
         case '2': LISTALIVRO(LIVRO, N); break;
         //case '3': CONSULTALIVRO(LIVRO, N); break;
         //case '4': ORDENALIVRO(LIVRO, N); break;
         //case '5': AJUSTAPRECO(LIVRO, N); break;
         case '6': break;
         default: cout <<"Digite somente opcoes validas!!"<<endl<<endl;
      }
   }while (OP != '6');
}
```

Exercício 2

- Faça um programa que leia o **nome**, **duas notas** e **número de faltas** de **N** alunos. As informações desses alunos devem ser armazenadas em um vetor de variáveis do tipo estrutura do aluno, o qual deve conter as seguintes informações de cada aluno:
  - Nome: string de tamanho 30;
  - Média: número real resultado da média das duas notas lidas;
  - Situação: caractere representando situação, isto é, 'A' (Aprovado), se média maior ou igual a 60 e número de faltas menor que 18, e 'R' (Reprovado), caso contrário.
- Ao final os dados dos alunos devem ser impressos em ordem alfabética (usar método da bolha para ordenação).

Exercício 3

- No exemplo da loja de livros fazer as seguintes funções e opções de menu:
  - Função CONSULTALIVRO que deverá receber a lista de livros já cadastrados, solicitar que o usuário entre com um nome de um livro e verificar se o livro esta cadastrado, mostrando o Nome e o Preco do livro. Se não estiver cadastrado deve mostra um mensagem que o livro não existe.
  - Uma função ORDENALIVRO que deve ordenar todos os livros cadastrados em ordem alfabética pelo nome do livro usando o método de ordenação por inserção.
  - Uma função AJUSTAPRECO que recebe os livros cadastrados, solicita que o usuário entre com um valor de porcentagem de aumento dos preços dos livros e atualiza todos os preços dos livros cadastrados com este valor definido pelo usuário.