MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2014

Roteiro

Tipos enumerados

- Para criar uma variável para armazenar um determinado mês de um ano (de janeiro a dezembro), uma das soluções possíveis é utilizar o tipo inteiro e armazenar um número associado àquele mês. Assim, janeiro seria o mês número 1, fevereiro o mês número 2, e assim sucessivamente.
- Entretanto, o código seria mais claro se pudéssemos escrever algo como:

```
mes = jan;
```

Tipos enumerados

- O comando `enum` cria um tipo enumerado. Com ele podemos usar nomes/identificadores para um conjunto finito de valores inteiros.
- A sintaxe do comando `enum` é a seguinte:

```
enum <tipo> {identificador_1, identificador_2, ···,
identificador_N};
```

- Exemplo:

```
/* Criando um novo tipo enumerado chamado mes */
enum mes {jan, fev, mar, abr, mai, jun,
          jul, ago, set, out, nov, dez};
```

Usando um tipo enumerado

- O compilador associa o número 0 ao primeiro identificador, 1 ao segundo, etc.
- Variáveis do novo tipo criado são, na realidade, variáveis inteiras.
- Tipos enumerados são usados para deixar o código mais legível.

```
#include <stdio.h>

/* Criando um novo tipo enumerado */
enum mes {jan, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez};

int main() {
  enum mes a, b; /* Aqui criamos 2 variaveis enumeradas do tipo mes */

  a = jan;
  b = jun;
  if (a != b) {
    /* Sera impresso "0 eh um mes diferente de 5" */
    printf("%d eh um mes diferente de %d\n", a, b);
  }
  return 0;
}
```

Usando um tipo enumerado

- Note que o primeiro identificador recebeu o valor zero, enquanto os demais identificadores receberam valores em sequência.
- Podemos alterar o valor inicial dos identificadores e, assim, os valores de todos os demais identificadores.

```
#include <stdio.h>

/* Criando um novo tipo enumerado */
enum mes {jan = 1, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez};

int main() {
  enum mes a, b; /* Aqui criamos 2 variaveis enumeradas do tipo mes */

  a = jan;
  b = jun;
  if (a != b) {
    /* Sera impresso "1 eh um mes diferente de 6" */
    printf("%d eh um mes diferente de %d\n", a, b);
  }
  return 0;
}
```

Outros exemplos de tipos enumerados

- Tipo para armazenar respostas binárias:

```
enum resposta {falsa, verdadeira};
```

- Tipo para armazenar as estações do ano:

```
enum estacao {primavera, verao, outono, inverno};
```

- Tipo para armazenar o sexo de uma pessoa:

```
enum sexo {masculino, feminino};
```

- Tipo para armazenar o naipe de uma carta de baralho:

```
enum naipe {ouros, espadas, copas, paus};
```

- Tipo para armazenar a direção de navegação:

```
enum direcao {N, NE, E, SE, S, SO, O, NO};
```

Registros

- Um registro é um mecanismo da linguagem C para agrupar diversas variáveis, que inclusive podem ser de tipos diferentes, mas que, dentro de um contexto, fazem sentido serem tratadas conjuntamente.
- Exemplos de usos de registros:
  - Registro de alunos para guardar dados como nome, RA, médias de provas e médias de laboratórios.
  - Registro de pacientes para guardar dados como nome, endereço e histórico de doenças.
  - Registro de clientes para guardar dados como nome, endereço, CPF, operadora e número do cartão de crédito.

Declarando um registro

- Para criarmos um novo tipo de registro, usamos a palavra reservada `struct` da seguinte forma:

```
struct nome_do_tipo_do_registro {
  tipo_1 nome_1;
  tipo_2 nome_2;
  ...
  tipo_n nome_n;
};
```

- Cada `nome_i` é um identificador de um campo do registro que será do tipo `tipo_i`.
- Exemplo:

```
struct Aluno {
  char nome[45];
  int idade;
  char sexo;
};
```

Declarando um registro

- A declaração do registro pode ser feita dentro de uma função (como `main`) ou fora dela. Usualmente, ela é feita fora de qualquer função, como no exemplo abaixo:

```
#include <stdio.h>

/* Declare aqui os registros do seu programa */

void funcao1() {
...
}

int main () {
...
}
```

Declarando um registro

- A próxima etapa é declarar uma variável registro do tipo definido, que será usada pelo seu programa, como no exemplo abaixo:

```
#include <stdio.h>

struct Aluno {
  char nome[45];
  int idade;
  char sexo;
};

int main() {
  /* Declarando variaveis registros do tipo Aluno */
  struct Aluno a, b;
  ...
}
```

Utilizando os campos de um registro

- Podemos acessar individualmente aos campos de uma determinada variável registro como se fossem variáveis comuns. A sintaxe é:

  `variável_registro.nome_do_campo`

- Os campos individuais de um variável registro têm o mesmo comportamento de qualquer variável do tipo do campo.
  - Isto significa que todas operações válidas para variáveis de um tipo são válidas para um campo do mesmo tipo.

Utilizando os campos de um registro

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

struct Aluno {
  char nome[45];
  int idade;
  char sexo;
};

int main() {
  struct Aluno a, b;

  strcpy(a.nome, "Helen");
  a.idade = 18;
  a.sexo = 'F';
  strcpy(b.nome, "Dilbert");
  b.idade = 34;
  b.sexo = 'M';

  printf("a.nome = %s, a.idade = %d, a.sexo = %c\n", a.nome, a.idade, a.sexo);
  printf("b.nome = %s, b.idade = %d, b.sexo = %c\n", b.nome, b.idade, b.sexo);
  return 0;
}
```

Lendo e escrevendo registros

- A leitura dos campos de um registro deve ser feita campo a campo, como se fossem variáveis independentes.
- O mesmo vale para a escrita, que deve ser feita campo a campo.

```
...

int main() {
  struct Aluno a;

  printf("Digite o nome: ");
  fgets(a.nome, 45, stdin);
  printf("Digite a idade: ");
  scanf("%d% ", &a.idade);
  printf("Digite o sexo: ");
  scanf(" "); /* Descarta espacos em branco */
  scanf("%c", &a.sexo);

  printf("a.nome = %s, a.idade = %d, a.sexo = %c\n",
         a.nome, a.idade, a.sexo);
  return 0;
}
```

Atribuição de registros

- Podemos atribuir um registro a outro diretamente:

  `var1_registro = var2_registro;`

- Neste caso é feita uma cópia de todos os campos do registro.

...

```
int main() {
  struct Aluno a, b;

  printf("Digite o nome: ");
  fgets(a.nome, 45, stdin);
  printf("Digite a idade: ");
  scanf("%d%", &a.idade);
  printf("Digite o sexo: ");
  scanf(" "); /* Descarta espacos em branco */
  scanf("%c", &a.sexo);

  b = a;

  printf("b.nome = %s, b.idade = %d, b.sexo = %c\n",
         b.nome, b.idade, b.sexo);
  return 0;
}
```

Vetor de registros

- Podemos declarar um vetor de registros quando necessitamos de várias instâncias de um mesmo tipo de registro, por exemplo, para cadastrar todos os alunos de uma mesma turma.
- Como declarar um vetor de registros:

```
struct nome_do_tipo_do_registro nome_do_vetor[MAX];
```

- Como acessar um campo de um registro de um vetor:

```
nome_do_vetor[indice].campo
```

Vetor de registros

```
#include <stdio.h>

struct Aluno {
  int ra;
  double nota;
};

int main () {
  struct Aluno turma[10];
  int i;
  double nota = 0.0;

  for (i = 0; i < 10; i++) {
    printf("Digite o RA do aluno numero %d: ", i + 1);
    scanf("%d", &turma[i].ra);
    printf("Digite a nota do aluno numero %d: ", i + 1);
    scanf("%lf", &turma[i].nota);
  }
```

Vetor de registros

```
  ...

  /* Calcula a nota media da turma */
  for (i = 0; i < 10; i++)
    nota = nota + turma[i].nota;

  printf("A media da turma eh: %f\n", nota / 10);

  return 0;
}
```

Redefinido um tipo

- Para facilitar a compreensão e melhorar a organização dos códigos, pode-se declarar um tipo próprio, que possui a mesma função de outro tipo já existente.
- Por exemplo, em um programa onde manipulamos médias de alunos, as variáveis relacionadas com o campo nota poderiam ser declaradas como `nota` e não `double`.

O comando typedef

- A forma de se fazer isso é utilizando o comando `typedef`, seguindo a sintaxe abaixo:

```
typedef <tipo_ja_existente> <tipo_novo>;
```

- Usualmente, fazemos essa declaração fora da função `main()`, embora também seja permitido declarar no novo tipo dentro da função.
- Exemplo:

  ```
  typedef float nota;
  ```

  Cria um novo tipo, chamado `nota`, cujas variáveis desse tipo serão pontos flutuantes.

Exemplo de uso do typedef

```
#include <stdio.h>

typedef double nota;

int main() {
  nota p1;

  printf("Digite a nota: ");
  scanf("%lf", &p1);

  printf("A nota digitada foi: %f", p1);

  return 0;
}
```

Exemplo de uso do `typedef`

- Os usos mais comuns para o comando `typedef` são para redefinições de tipos enumerados e registros.
- Poderíamos redefinir o tipo `enum mes` como simplesmente `mes`:
  - `typedef enum mes mes;`
- Assim como poderíamos redefinir o tipo `struct Aluno` como simplesmente `Aluno`:
  - `typedef struct Aluno Aluno;`

Exemplo de uso do typedef

```
#include <stdio.h>

/* Definindo o tipo enumerado mes */
enum mes {jan, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez};

/* Redefinindo o tipo "enum mes" como "mes" */
typedef enum mes mes;

int main() {
  mes a, b;

  a = jan;
  b = jun;

  if (a != b) {
    /* Sera impresso "0 eh um mes diferente de 5" */
    printf("%d eh um mes diferente de %d\n", a, b);
  }
  return 0;
}
```

Exemplo de uso do typedef

```
#include <stdio.h>

/* Definindo o tipo registro Aluno */
struct Aluno {
  int ra;
  double nota;
};

/* Redefinindo o tipo "struct Aluno" como "Aluno" */
typedef struct Aluno Aluno;
```

Exemplo de uso do typedef

```
int main () {
  Aluno turma[10];
  int i;
  double nota = 0.0;

  for (i = 0; i < 10; i++) {
    printf("Digite o RA do aluno numero %d: ", i + 1);
    scanf("%d", &turma[i].ra);
    printf("Digite a média do aluno numero %d: ", i + 1);
    scanf("%lf", &turma[i].nota);
  }

  for (i = 0; i < 10; i++)
    nota = nota + turma[i].nota;

  printf("A media da turma eh: %f\n", nota / 10);
  return 0;
}
```

Exemplo de uso de registros

- Suponha que queremos imprimir as informações dos alunos de uma turma (RA e nome) formada por n alunos, onde o valor de n não é conhecido a priori.
- Além da ordem original dos alunos, deseja-se também imprimir as informações ordenadas de outras duas formas:
  - Em ordem crescente dos números dos RAs.
  - Em ordem lexicográfica dos nomes.

Exemplo de uso de registros

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

/* Definicao do tipo Aluno */
struct Aluno {
  int ra;
  char nome[52];
};
typedef struct Aluno Aluno;

void imprime_turma(Aluno turma[], int n) {
  int i;

  printf("*** Turma ***\n");
  for (i = 0; i < n; i++)
    printf("%06d %s", turma[i].ra, turma[i].nome);
  printf("\n");
}
```

Exemplo de uso de registros

```
void ordena_por_ra(Aluno turma[], int n) {
  int i, j;
  Aluno aux;

  for (i = 1; i < n; i++) {
    aux = turma[i];
    j = i - 1;

    while ((j >= 0) && (turma[j].ra > aux.ra)) {
      turma[j + 1] = turma[j];
      j--;
    }

    turma[j + 1] = aux;
  }
}
```

Exemplo de uso de registros

```
void ordena_por_nome(Aluno turma[], int n) {
  int i, j;
  Aluno aux;

  for (i = 1; i < n; i++) {
    aux = turma[i];
    j = i - 1;

    while ((j >= 0) && (strcmp(turma[j].nome, aux.nome) > 0)) {
      turma[j + 1] = turma[j];
      j--;
    }

    turma[j + 1] = aux;
  }
}
```

Exemplo de uso de registros

```
int main () {
  Aluno *turma;
  int n, i;

  printf("Quantos alunos? ");
  scanf("%d", &n);

  turma = malloc(sizeof(Aluno) * n);

  for (i = 0; i < n; i++) {
    printf("* Aluno numero %d:\n", i + 1);
    printf("RA: ");
    scanf("%d", &turma[i].ra);
    printf("Nome: ");
    scanf(" "); /* Descarta espacos em branco */
    fgets(turma[i].nome, 52, stdin);
  }
  ...
```

Exemplo de uso de registros

```
  ...
  /* Turma na ordem inicial */
  imprime_turma(turma, n);

  ordena_por_ra(turma, n);

  /* Turma ordenada por ra */
  imprime_turma(turma, n);

  ordena_por_nome(turma, n);

  /* Turma ordenada por nome */
  imprime_turma(turma, n);

  free(turma);

  return 0;
}
```

Exemplo de uso de registros

```
*** Turma ***
152034 Ronald Bilius Weasley
161212 Hermione Jean Granger
149826 Harry James Potter

*** Turma ***
149826 Harry James Potter
152034 Ronald Bilius Weasley
161212 Hermione Jean Granger

*** Turma ***
149826 Harry James Potter
161212 Hermione Jean Granger
152034 Ronald Bilius Weasley
```

Ponteiros para registros

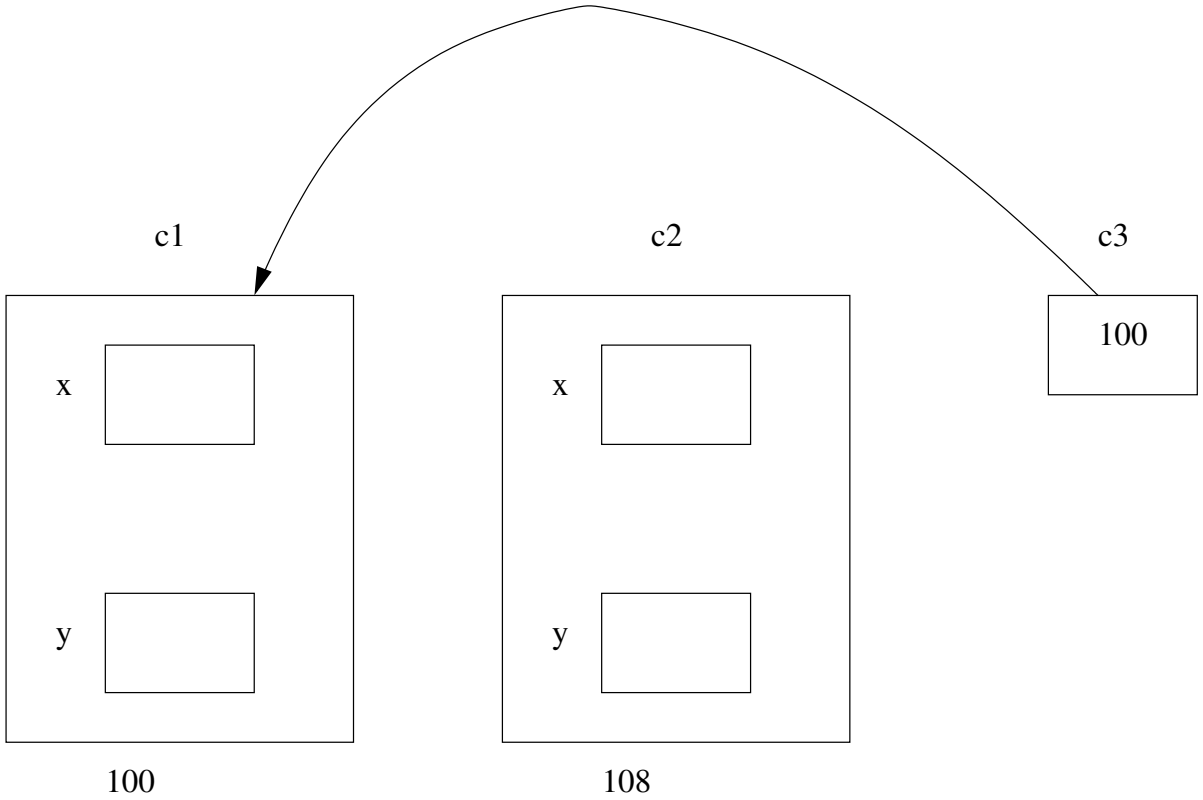
- Ao criarmos uma variável do tipo `struct`, esta é armazenada na memória como qualquer outra variável e, portanto, possui um endereço.
- Como vimos anteriormente, é possível então criar um ponteiro para uma variável de um tipo `struct`.

```
#include <stdio.h>

struct Coordenadas {
  float x, y;
};
typedef struct Coordenadas Coordenadas;

int main() {
  Coordenadas c1, c2, *c3;
  c3 = &c1;
  ...
```

Ponteiros para registros



Ponteiros para registros

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

int main() {
  Coordenadas c1, c2, *c3;

  c3 = &c1;
  c1.x = -1;
  c1.y = -1.5;
  c2.x = 2.5;
  c2.y = -5;
  *c3 = c2;

  printf("Coordenadas de c1: (%0.2f, %0.2f)\n", c1.x, c1.y);
  return 0;
}
```

O que será impresso pelo programa?

```
Coordenadas de c1: (2.50, -5.00)
```

Ponteiros para registros

- Para se ter acesso aos campos de uma variável `struct` via um ponteiro, podemos utilizar o operador '*' juntamente com o operador '.', como usualmente:

```
Coordenadas c1, *c3;
c3 = &c1;
(*c3).x = 1.5;
(*c3).y = 1.5;
```

- Em C, também podemos usar o operador ->, que também é usado para ter acesso aos campos de um registro via um ponteiro. Podemos obter o mesmo resultado do exemplo anterior:

```
Coordenadas c1, *c3;
c3 = &c1;
c3->x = 1.5;
c3->y = 1.5;
```

- Resumindo: para ter acesso aos campos de registros via ponteiros, podemos usar uma destas duas formas:
  - `ponteiroRegistro->campo`
  - `(*ponteiroRegistro).campo`

Ponteiros para registros

```
#include <stdio.h>

struct Coordenadas {
  float x, y;
};
typedef struct Coordenadas Coordenadas;

int main() {
  Coordenadas c1, c2, *c3, *c4;

  c3 = &c1;
  c4 = &c2;

  c1.x = -1;
  c1.y = -1.5;

  c2.x = 2.5;
  c2.y = -5;

  ...
```

Ponteiros para registros

```
  ...

  (*c3).x = 1.5;
  (*c3).y = 1.5;

  c4->x = -1;
  c4->y = -1;

  printf("Coordenadas de c1: (%0.2f, %0.2f)\n", c1.x, c1.y);
  printf("Coordenadas de c2: (%0.2f, %0.2f)\n", c2.x, c2.y);

  return 0;
}
```

O que será impresso pelo programa?

```
Coordenadas de c1: (1.50, 1.50)
Coordenadas de c2: (-1.00, -1.00)
```

Exemplo com registros

- Vamos criar uma pequena aplicação para manter um inventário de estoque com as seguintes informações:
  - Nome do item, quantidade e preço.
- Além disso, nosso programa deverá ter opções para incluir/excluir um item do estoque.
- Usaremos o seguinte registro para representar um item do estoque:

```
struct Item {
  char nome[52];
  int quantidade;
  double preco;
  int emUso;
};
typedef struct Item Item;
```

- Usaremos um vetor para armazenar o estoque.
  - O campo `emUso` de Item servirá para indicar se no vetor uma determinada posição está em uso (1) ou não (0).

Exemplo com registros

Vamos criar as seguintes funções:

- `void inicializaEstoque(Item estoque[], int tam);`
  Inicializa o vetor usado para armazenar os itens do estoque.
- `void cadastraItem(Item *item);`
  Cadastra um item para posteriormente ser incluído no estoque.
- `void imprimeItem(Item item);`
  Imprime os dados de um item.
- `void imprimeEstoque(Item estoque[], int tam);`
  Imprime os dados de todos os itens do estoque.
- `int insereItem(Item estoque[], int tam, Item item);`
  Insere um novo item no estoque, se houver espaço.
- `int removeItem(Item estoque[], int tam, char nome[]);`
  Dado o nome, remove o item, se ele estiver no estoque.

Exemplo com registros

```
void inicializaEstoque(Item estoque[], int tam) {
  int i;

  /* Inicialmente todas as posicoes do estoque estao vagas */
  for (i = 0; i < tam; i++)
    estoque[i].emUso = 0;
}
```

Exemplo com registros

```
void cadastraItem(Item *item) {
  printf("----- Cadastrando um Item -----\n");

  printf("Digite o nome do item: ");
  scanf(" "); /* Descarta espacos em branco */
  fgets(item->nome, 52, stdin);

  printf("Digite a quantidade do item: ");
  scanf("%d", &(item->quantidade));

  printf("Digite o preco do item: ");
  scanf("%lf", &(item->preco));
}
```

Exemplo com registros

```
void imprimeItem(Item item) {
  printf("----- Imprimindo Item -----\n");
  printf("Nome: %s\n", item.nome);
  printf("Quantidade: %d\n", item.quantidade);
  printf("Preco: R$%0.2f\n", item.preco);
}

void imprimeEstoque(Item estoque[], int tam) {
  int i;

  for (i = 0; i < tam; i++)
    /* Se a posicao i estiver em uso, imprime o item */
    if (estoque[i].emUso)
      imprimeItem(estoque[i]);
}
```

Note o uso do campo `emUso` na função `imprimeEstoque`.

Exemplo com registros

```
int insereItem(Item estoque[], int tam, Item item) {
  int i;

  for (i = 0; i < tam; i++)
    /* Se a posicao i estiver vaga... */
    if (estoque[i].emUso == 0) {
      estoque[i] = item;
      estoque[i].emUso = 1; /* ... a posicao i passa a estar em uso */
      return 1; /* Item inserido com sucesso */
    }

  return 0; /* Nao foi possivel inserir o item */
}
```

Note, novamente, o uso do campo `emUso` nesta função.

Exemplo com registros

```
int removeItem(Item estoque[], int tam, char nome[]) {
  int i;

  for (i = 0; i < tam; i++)
    /* Se achou o item no estoque... */
    if ((estoque[i].emUso) &&
        (strcmp(estoque[i].nome, nome) == 0)) {
      estoque[i].emUso = 0; /* ... remove o item do estoque */
      return 1; /* Item removido com sucesso */
    }

  return 0; /* Nao foi possivel remover o item */
}
```

Note, novamente, o uso do campo emUso nesta função.

Exercícios

- Implemente uma função como o seguinte protótipo:
  `int buscaItem(Item estoque[], int tam, char nome[]);`
  Sua função deve verificar se existe um item cadastrado com o nome dado. Se existir, deve retornar o índice do item no vetor que representa o estoque. Caso contrário, deve retornar `-1`.
- Altere a função `insereItem` de tal forma a garantir que nunca existirão dois itens no estoque com o mesmo nome.
- Altere as funções `insereItem` e `imprimeItem` (e por consequência, também a função `imprimeEstoque`) de tal forma que estas funções recebam um ponteiro para o item de interesse e não uma cópia do mesmo, como nas implementações atuais destas funções.
- Implemente uma função que dado o vetor que representa o estoque, seu tamanho, um nome e um inteiro x (não necessariamente positivo), incremente a quantidade de itens com aquele nome registrado no estoque em x unidades.

Exercícios

- Escreva uma função que, dados dois registros do tipo `data` (com campos dia, mês e ano), retorne o número de dias entre as duas datas.
- Escreva uma função que, dado um polinômio $P(x)$ (representado através de um registro) e um ponto x, retorne o valor do polinômio no ponto dado.
- Escreva uma função que, dado um registro `triangulo` (formado por 3 registros do tipo `ponto`), determine se o triângulo é equilátero, isósceles ou escaleno.