Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2013

Roteiro

Ponteiros para Registros

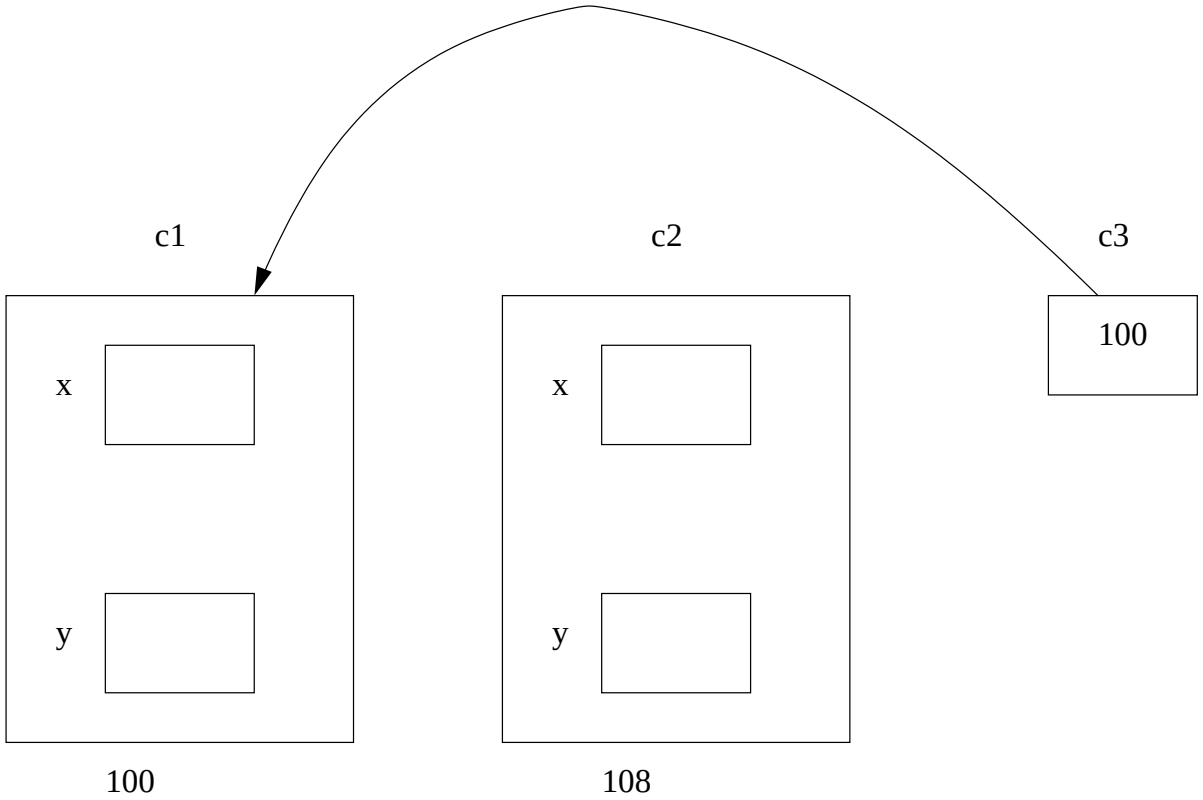
- Ao criarmos uma variável do tipo `struct`, esta é armazenada na memória como qualquer outra variável e, portanto, possui um endereço.
- É possível então criar um ponteiro para uma variável de um tipo `struct`.

```
#include <stdio.h>

struct Coordenada {
  float x;
  float y;
};
typedef struct Coordenada Coordenada;

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

Ponteiros para Registros



Ponteiros para Registros

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

int main() {
  Coordenada 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: (%lf, %lf)\n", c1.x, c1.y);
  return 0;
}
```

O que será impresso por este programa?

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:

```
Coordenada 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 uma estrutura via um ponteiro. Podemos obter o mesmo resultado do exemplo anterior:

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

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

Ponteiros para Registros

```
...
int main() {
  Coordenada c1, c2, *c3, *c4;
  c3 = &c1;
  c4 = &c2;

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

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

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

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

  printf("Coordenadas de c1: (%lf, %lf)\n", c1.x, c1.y);
  printf("Coordenadas de c2: (%lf, %lf)\n", c2.x, c2.y);
  return 0;
}
```

O que será impresso por este programa?

Exemplo com Registros

- Vamos criar uma pequena aplicação para manter um cadastro de frutas com as seguintes informações:
  - Nome, peso médio, número médio de calorias.
- Além disso, nosso programa deverá ter opções para incluir/excluir uma fruta do cadastro.
- Usaremos a seguinte estrutura para representar uma fruta:

```
struct Fruta {
  char nome[80];
  double peso;
  double calorias;
  short emUso; /* apenas para indicar se está sendo usado ou não */
};
typedef struct Fruta Fruta;
```

- Usaremos um vetor como uma base de dados para cadastro das frutas.
  - O campo `emUso` de Fruta serve para indicar se no vetor uma determinada posição está em uso (1) ou não (0).

Exemplo com Registros

Vamos criar as seguinte funções:

- `void cadastraFruta(Fruta *f);`
  Cadastra uma fruta.
- `void imprimeFruta(Fruta f);`
  Imprime os dados de uma dada fruta.
- `void imprimeFrutas(Fruta vet[], int tam);`
  Imprime dados de um cadastro inteiro de frutas.
- `int insereFruta(Fruta vet[], int tam, Fruta f);`
  Insere uma nova fruta no cadastro, se houver espaço.
- `int removeFruta(Fruta vet[], int tam, char nome[]);`
  Dado o nome, remove a fruta, se ela estiver cadastrada.
- `void inicializaCadastro(Fruta vet[], int tam);`
  Inicializa o vetor usado para armazenar as frutas.

Exemplo com Registros

```
/* Cadastra uma fruta */
void cadastraFruta(Fruta *f) {
  printf("----- Cadastrando Fruta -----\n");

  printf("Digite o nome da fruta: ");
  scanf("%[^\n]", f->nome);
  getchar();

  printf("Digite o peso medio da fruta: ");
  scanf("%lf", &(f->peso));

  printf("Digite a quantidade de calorias da fruta: ");
  scanf("%lf", &(f->calorias));
}
```

Exemplo com Registros

Note o uso do campo emUso na função para imprimir todo o cadastro.

```
/* funcao que imprime uma unica fruta */
void imprimeFruta(Fruta f) {
  printf("----- Imprimindo Fruta -----\n");
  printf("Nome: %s\n", f.nome);
  printf("Peso medio: %lf\n", f.peso);
  printf("Calorias: %lf\n", f.calorias);
}

/* funcao que imprime todas as frutas do cadastro */
void imprimeFrutas(Fruta vet[], int tam) {
  int i;

  for(i = 0; i < tam; i++)
    /* se a posicao i estiver em uso, imprime a fruta */
    if (vet[i].emUso)
      imprimeFruta(vet[i]);
}
```

Exemplo com Registros

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

```
/* retorna 1 ou 0 dependendo se foi possivel ou nao inserir a fruta */
int insereFruta(Fruta vet[], int tam, Fruta f) {
  int i;

  for (i = 0; i < tam; i++)
    if (vet[i].emUso == 0) { /* se posicao i estiver vaga */
      vet[i] = f;
      vet[i].emUso = 1; /* posicao i passa a estar em uso */
      return 1;
    }

  return 0; /* cadastro esta cheio */
}
```

Exemplo com Registros

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

```
/* retorna 1 ou 0 dependendo se foi possivel ou nao remover a fruta */
int removeFruta(Fruta vet[], int tam, char nome[]) {
  int i;

  for (i = 0; i < tam; i++)
    /* strcmp retorna 0 se as duas strings forem iguais */
    if ((vet[i].emUso) &&
        (strcmp(vet[i].nome, nome) == 0)) {
      vet[i].emUso = 0; /* posicao i fica vaga */
      return 1;
    }

  return 0; /* fruta nao esta cadastrada */
}
```

Exemplo com Registros

Como no início do programa todo o cadastro está vago, criamos uma função para deixar os campos `emUso` consistentes com este fato.

```
/* faz com que todas posicoes do vetor fiquem vagas */
void inicializaCadastro(Fruta vet[], int tam) {
  int i;

  for(i = 0; i < tam; i++)
    vet[i].emUso = 0;
}
```

Exercícios

- Implemente uma função:
  `int buscaFruta(Fruta vet[], int tam, char nome[]);`
  Sua função deve verificar se existe um fruta cadastrada com o nome dado. Se existir, deve retornar o índice da fruta no vetor. Caso contrário, deve retornar `-1`.
- Altere a função `insereFruta` previamente definida de tal forma a garantir que nunca existam duas frutas cadastradas com o mesmo nome.