MC-102 — Aula 17
Ponteiros para Registros e Exemplos

Instituto de Computação – Unicamp

17 de Maio de 2012

Roteiro

Ponteiros para Registros

Ao criarmos uma variável de um tipo **struct**, esta é armazenada na memória como qualquer outra variável, e portanto possui um endereço.

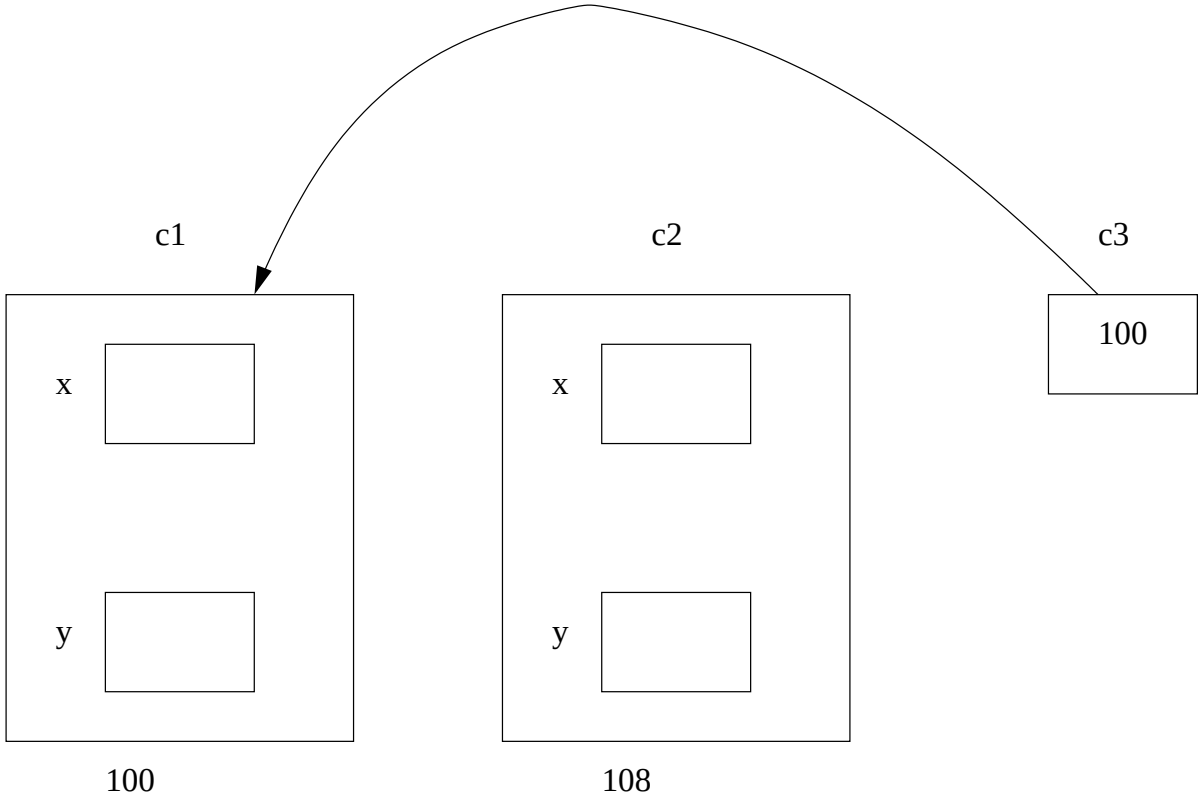
É possível então criar um ponteiro para uma variável de um tipo **struct**!

```
#include <stdio.h>

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


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

Ponteiros para Registros



Ponteiros para Registros

```
#include <stdio.h>
struct Coordenada{
  double x;
  double 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);
}
```

O que será impresso??

Ponteiros para Registros

Para acessarmos os campos de uma variável **struct** via um ponteiro, podemos utilizar o operador * juntamente com o operador . como de costume:

```
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 acessar 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 acessar campos de estruturas via ponteiros use um dos dois:

- `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);
}
```

O que será impresso??

Exemplo com Registros

Vamos criar uma pequena aplicação para manter um cadastro de frutas com as seguintes informações:

- Nome, peso médio, número de calorias.

Além disso nosso programa deverá ter opções para incluir/excluir uma fruta do cadastro.
Usaremos a seguinte estrutura:

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

Usaremos um vetor como uma base de dados para cadastro das frutas.

- O campo **usado** 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 leFruta(Fruta *f);: lê dados de uma única fruta passada como ponteiro. (Por quê como ponteiro?)

void imprimeFruta(Fruta f); Imprime dados de uma 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[]); Remove uma fruta pelo nome **se esta estiver cadastrada!**

Exemplo com Registros

```
//Le dados de uma unica fruta
void leFruta(Fruta *f){
  printf(" ------ Lendo Fruta -------\n");

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

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

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

Exemplo com Registros

Note o uso do campo **usado** na função para imprimir todo o cadastro!

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

//funcao que imprime todas as frutas de um vetor
void imprimeFrutas(Fruta vet[], int tam){
  int i;
  for(i=0; i<tam; i++){
    if(vet[i].usado == 1){ //se posicao i conter algo valido entao imprime
      imprimeFruta(vet[i]);
    }
  }
}
```

Exemplo com Registros

Note o uso do campo **usado** nesta função!

```
//retorna 1 ou 0 dependendo se foi possível ou não cadastrar a fruta
int insereFruta(Fruta vet[], int tam, Fruta f){
  int i;

  for(i=0; i<tam; i++){
    if(vet[i].usado == 0){ //se posição i estiver vaga
      vet[i] = f;
      vet[i].usado = 1; //posição i passa a estar usada
      return 1;
    }
  }
  return 0; //cadastro está cheio
}
```

Exemplo com Registros

Note o uso do campo **usado** nesta função!

```
//retorna 1 ou 0 dependendo se foi possível 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 2 strings parametros
    //forem iguais
    if( strcmp(vet[i].nome , nome) == 0){
      vet[i].usado = 0; //posicao i fica vaga
      return 1;
    }
  }
  return 0; //fruta não está cadastrada
}
```

Exemplo com Registros

Como no inicio do programa todo o cadastro está vago, criamos uma função para deixar os campos **usado** consistentes com este fato!

```
//faz com que todas posições do vetor fiquem vagas
void iniciaCadastro(Fruta vet[], int tam){
  int i;
  for(i=0; i<tam; i++)
    vet[i].usado = 0;
}
```

Olhem o programa completo em **cadastroFrutas.c**!