28. Registros, ponteiros, sub-rotinas

DIM0321

2015.1

Sumário

Lembretes

- Registros são apenas um **outro tipo de dados**.
- Ponteiros de registros podem ser declarados e usados como foi o caso para inteiros, pontos flutuantes...
- e como será o caso

Sintaxe e uso

```c
int main (void)
{
    typedef struct {
        unsigned int dia;
        unsigned int mes;
        unsigned int ano;
    } data;

    data hoje = { 21, 12, 2014 };
    data *p_data;
    data *p_data2 = &hoje;

    p_data = &hoje;

    (*p_data).dia = 6;      /* os parênteses envolvendo *p_data são necessários */
    (*p_data).mes = 5;      /* devido ao operador ., de seleção de campo de registro, */
    (*p_data).ano = 2013;


    p_data->dia = 6;      /* os parênteses envolvendo *p_data são necessários */
    p_data->mes = 5;      /* devido ao operador ., de seleção de campo de registro, */
    p_data->ano = 2013;
}
```

Alocação

- Registros são alocados em memória **sequencialmente**.
- Podemos usar os operadores de referenciamento & e dereferenciamento *.

Uso de ->

- Existe um outro operador de acesso indireto aos campos de um registro, >
- Ele permite substituir o uso simultâneo dos operadores * e . no caso dos ponteiros.

Exemplo

```c
#include <stdio.h>
typedef struct {
    int dia;
    int mes;
    int ano;
} data;

int main(void)
{
    data hoje;
    data *p_data;

    p_data = &hoje;
    p_data->dia = 6;
    p_data->mes = 5;
    p_data->ano = 2013;

    printf("data de hoje: %.2i/%.2i/%.4i\n", hoje.dia, hoje.mes, hoje.ano);
    return 0;
}
```

Ponteiros como campo de registro

Atenção à sintaxe

Exemplo

```
1  typedef struct {
2      int i;
3      int *ptr;
4  } registro;
5
6  registro reg = { 10, &reg.i };
7  registro *reg_ptr = &reg;
8
9     reg.i;          /* variável comum com acesso direto a um campo comum */
10   *reg.ptr;        /* variável comum com acesso direto a um campo ponteiro + indireção */
11  (*reg_ptr).i;     /* ponteiro com acesso indireto a um campo comum */
12    reg_ptr->i;     /* ponteiro com acesso indireto a um campo comum */
13 *(*reg_ptr).ptr;   /* ponteiro com acesso indireto a um campo ponteiro + indireção */
14   *reg_ptr->ptr;   /* ponteiro com acesso indireto a um campo ponteiro + indireção */
```

Em resumo

```c
#include <stdio.h>
typedef struct {
    int *ptr1;
    int *ptr2;
} registro;

int main(void)
{
    int i1, i2 = 100;
    registro reg, *reg_ptr;
    reg.ptr1  = &i1;
    reg.ptr2  = &i2;
    reg_ptr   = &reg;
    *reg.ptr1 = 35;
    *(*reg_ptr).ptr1 = *reg.ptr1 / 7;
    *reg_ptr->ptr1   = *reg_ptr->ptr1 + 5;
    *reg_ptr->ptr2  -= 50;

    printf("%i, %i, %i, %i\n", i1, *reg.ptr1, *(*reg_ptr).ptr1, *reg_ptr->ptr1);
    printf("%i, %i, %i, %i\n", i2, *reg.ptr2, *(*reg_ptr).ptr2, *reg_ptr->ptr2);
    return 0;
}
```

Uso nas sub-rotinas

- Registros, com0 tipos compostos heterogêneos, também podem ser passados como parâmetros nas sub-rotinas.
- Diferentemente dos arranjos que são ponteiros e não copiam dados, um registro **copia** todos os seus dados na passagem de parâmetro tal qual ocorre com os tipos básicos.
- Para evitar a cópia dos dados, podemos novamente fazer uso explícito de ponteiros.

Passagem como parâmetro

Modo

- Pode-se passar:
  - um registro inteiro
  - apenas um campo,
  - ou o endereço de um deles como parâmetro de sub-rotina.
- Todas as passagens são **por valor** i.e. cópias.

Retornar um registro

Observação

- Sub-rotinas podem retornar registros, porque eles são estruturas de dados.
- Só arranjo (i.e ponteiros) não precisam ser retornados porque eles podem ser modificados **inplace** (no lugar)

Exemplo

```c
#include <stdio.h>

typedef struct {
    int codigo;
    char descricao[50];
    float valor;
    int estoque;
} produto;

void imprime_produto(produto prod)
{
    printf("%s (cod. %i): R$%.2f\n", prod.descricao, prod.codigo, prod.valor);
}

produto altera_preco(produto prod, float novo_valor)
{
    prod.valor = novo_valor;
    return prod;
}

int main(void)
{
    produto produto = { 104, "caneta", 3.25, 1598 };
    imprime_produto(produto);
    produto = altera_preco(produto, 2.19);
    imprime_produto(produto);
    return 0;
}
```

Alinhamento em memória

```c
#include <stdio.h>

typedef struct {
     char c;
     int i;
} data1;

typedef struct {
     int i;
     char c;
} data2;

int main(void)
{
     data1 d1;
     data2 d2;

     printf("char: %p, int: %p\n", &d1.c, &d1.i);
     printf("char: %p, int: %p\n", &d2.c, &d2.i);
     d2.c = 2;
     printf("%i\n", d2.c);
     *(&d2.i + 1) = 4;
     printf("%i\n", d2.c);
     return 0;
}
```

Referências

Precisões

[KR88] 6

[Bac13] 8.1

André Backes, *Linguagem C completa e descomplicada*, Elsevier, 2013.

Brian W. Kernighan and Dennis M. Ritchie, *The (ANSI) C Programming Language*, 2nd ed., Prentice Hall Professional Technical Reference, 1988.

Perguntas ?

http://dimap.ufrn.br/~richard/dim0321