16. Ponteiros

DIM0321

2015.1

Outline

Organização da memória

- Memória ≈ vetor de células numeradas e consecutiva
- As células podem ser manipuladas
  - individualmente
  - em grupos contínuos (≈ intervalo de memória)

- 1 célula = 1 byte (`char`)
- 2 células consecutivas = `short`
- 4 células consecutivas = `long`

Relação variável / memória

Definição

Variável = bloco de dados com

- endereço na memória (`0xff879877`)
- um nome (`x`)
- um conteúdo / um valor (42)

Operadores

Definição

- `*` retorna o conteúdo no endereço de uma variável apontada
  - Esta operação é chamada de **dereferenciamento**
- `&` permite acessar o endereço de uma variável (**referenciamento**)
  - Só aplicável aos objetos em memória : variáveis, vetores, funções
  - Não pode ser aplicado a constantes ou expressões

```
1 int pos = 0;
2 int pos2;
3 int *pos_p;
4 pos_p = &pos;
5 pos2 = *pos_p;
```


Precedências

- Os operadores `*` e `&` têm prioridade sobre operadores aritméticos binários
- Mesmo nível de prioridade do que `++` ou `--`

Exemplo

```
1 int *ip, *iq;
2 *ip = 2;
3 y = *ip + 1; // == (*ip) + 1
4 *ip += 1; // == *ip = (*ip) + 1
5 ++*ip; // == (*ip = *ip + 1, *ip)
6 *ip++; // == (*ip, ip = ip + 1)
7 (*ip)++; // == (*ip, *ip = *ip + 1)
8 iq = ip;
```


Sentidos de *

Observação

`*` tem 3 usos:

- multiplicação,
- declaração um ponteiro
- derefenciamento.

Exemplo

```c
#include <stdio.h>
int main(void)
{
    int posicao = 12;
    int *posicao_p = NULL;
    posicao_p = &posicao;

    printf("& posicao_p: %p\n", &posicao_p);
    printf("& posicao: %p\n", &posicao);
    printf("posicao_p: %p\n", posicao_p);
    printf("posicao: %i\n", posicao);
    printf("* posicao_p: %i\n\n", *posicao_p);
    posicao += 10;
    printf("posicao: %i\n", posicao);
    printf("* posicao_p: %i\n\n", *posicao_p);
    *posicao_p += 10;
    printf("posicao: %i\n", posicao);
    printf("* posicao_p: %i\n\n", *posicao_p);
    int n_posicao = 50;
    posicao_p = &n_posicao;
    printf("& n_posicao: %p\n", &n_posicao);
    printf("posicao_p: %p\n", posicao_p);
    printf("n_posicao: %i\n", n_posicao);
    printf("* posicao_p: %i\n", *posicao_p);
    printf("posicao: %i", posicao);
    return 0;
}
```

Saída

```
& posicao_p:   0x7ffcb5204620
& posicao:     0x7ffcb520462c
posicao_p:     0x7ffcb520462c
posicao:               12
* posicao_p:           12

posicao:               22
* posicao_p:           22

posicao:               32
* posicao_p:           32

& n_posicao:   0x7ffcb520461c
posicao_p:     0x7ffcb520461c
n_posicao:             50
* posicao_p:           50
posicao:               32
```


Exercício

1. O que será impresso no seguinte programa?
2. Qual é o significado de cada asterisco?

```c
#include <stdio.h>

int main(void)
{
    int i = 2;
    int j = i * i;    // j = 4
    int *k = &i;      // *k = 2
    int m = *k * *k; // m = 4
    *k = j * *k * m; // *k = 4 * 2 * 4 = 32

    printf("%i %i %i %i", i, j, *k, m);
    return 0;
}
```


Exercício

- O que será impresso no seguinte programa?

```c
#include <stdio.h>

int main(void)
{
   int x = 5;      // x =5
   int *y = &x;    // *y = 5
   int z = *y;     // z = 5
   printf("%i, %i, %i\n", x, *y, z);

   x = 7;          // x = 7
   printf("%i, %i, %i\n", x, *y, z);

   *y = 2;
   printf("%i, %i, %i\n", x, *y, z);
   return 0;
}
```


Exercício

- O que será impresso no seguinte programa?

```c
#include <stdio.h>

int main(void)
{
    int i = 0, j = 1;
    int *p = &i;
    int *q = &j;

    printf("%i %i %p %p\n", i, j, &i, &j);
    printf("%p %p %p %p\n", p, q, &p, &q);
    printf("%i %i\n", *p, *q);

    *p = *q;  // *p = 1, *p == i
    printf("%i %i %p %p\n", i, j, &i, &j);
    printf("%p %p %p %p\n", p, q, &p, &q);
    printf("%i %i\n", *p, *q);
    return 0;
}
```

Tipagem

Seja **T** um tipo:

- Se ‘x‘ é do tipo ‘T‘, ‘&x‘ é do tipo ‘T*‘
- Se ‘x‘ é do tipo ‘T*‘, ‘*x‘ é do tipo ‘T‘

Exemplo

```
1 int i;
2 int *p;
```

Elemento	Tipo
i	int
&i	int*
*i	inválido
p	int*
&p	int**
*p	int

Caso especial `void *`

- Um ponteiro aponta sempre para um certo tipo dado
- Um exceção: a declaração `void *` designa um ponteiro para **qualquer** tipo de dado
- Um ponteiro `void *` não pode ser dereferenciado. Ele precise ser convertido para um tipo de dado específico antes de pode ser dereferenciado.

Exercício

- O que será impresso no seguinte programa?

```c
#include <stdio.h>

int main(void)
{
    int x, y = 0, *p = NULL;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    --x;
    (*p) += x;

    printf("%i %i %i", x, y, *p);
    return 0;
}
```

Exercício

- O que será impresso no seguinte programa?

```c
#include <stdio.h>

int main(void)
{
    int i = 5;
    int *p = &i;

    printf("%i %i %i %i %i", p, *p+2, **&p, 3**p, **&p+4);
    return 0;
}
```


Referências

Precisões

[KR88] 5

[Bac13] 2.1.3, 2.1.4, 3.5, 3.8, 14.3.3, 15.1

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