Ponteiros

Prof. John Lenon C. Gardenghi

1 Introdução

A memória RAM (= random access memory) de qualquer computador é uma sequência de bytes. A posição (0, 1, 2, 3, etc.) que um byte ocupa na sequência é o endereço do byte. Se e é o endereço de um byte então $e + 1$ é o endereço do byte seguinte.

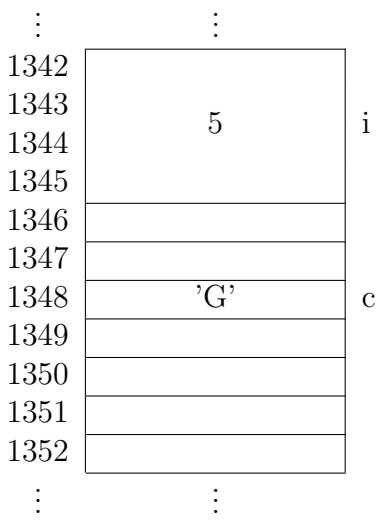
As variáveis de um programa ocupam uma certa quantidade de bytes na memória, de acordo com seu tipo. Para saber o tamanho que um dado em C ocupa na memória, existe a função `sizeof`.

Exemplo 1. *Implemente o código do Slide 1, e diga qual o tamanho que cada variável ocupa na memória.*

Cada posição da memória tem um endereço. Na maioria dos computadores, o endereço de uma variável é o endereço de memória que ocupa seu primeiro byte. O compilador é que controla do local de armazenamento destas variáveis em memória. Por exemplo, após as declarações

```
int i = 5;
char c = 'G';
```

as variáveis podem assumir os seguintes endereços:



O operador unário `&` é o **operador de endereço**. Retorna o endereço que uma variável ocupa na memória. Em nosso exemplo,

```
&i;  /* Contém 1342 */
```

Todavia, os endereços de memória costumam ser representados em notação *hexadecimal*.

Exemplo 2. *Implemente o código no Slide 2 e observe os endereços de memória.*

Ponteiros são variáveis que armazenam endereços de memória. Uma variável contém um valor, um ponteiro, por sua vez, contém o endereço de uma variável, que contém um valor. Diz-se que uma variável referencia um valor *diretamente*, enquanto um ponteiro referencia um valor *indiretamente*. Por isso, chama-se o uso do ponteiro, muitas vezes, de **indireção**.

Como qualquer outra variável, um ponteiro deve ser declarado. O formato de declaração de ponteiros é:

```
tipo *ptr;
```

onde `tipo` são os tipos de variáveis em C. Um ponteiro pode ter o valor `NULL`, que é o valor inválido de ponteiros.

Se um ponteiro p armazena o endereço de uma variável i, dizemos que "`p` aponta para `i`". Se um ponteiro p é diferente de `NULL`, então

```
*p
```

é o valor da variável apontada por p.

Exemplo 3. *Rode o código do Slide 3 e observe sua saída.*

Como exemplo de aplicação de ponteiros, escreva uma função `troca` que receba dois valores inteiros `a` e `b` e troque os valores de cada.

Exemplo 4. *Slide 4.*

Há duas formas de passar argumentos para funções: por **valor** e por **referência**. No primeiro, cópia das variáveis originais são criadas nos parâmetros da função, e a função lida apenas com cópias das variáveis originais do processo chamador. No segundo, ao invés de criar uma cópia, a própria variável original no processo chamador é passada ao procedimento chamado. Em nosso exemplo, a função troca recebe argumentos por **referência**.

Exemplo 5. *Faça uma função em C que receba um inteiro n e retorne n^2 e $\sqrt{n}$.*

Ponteiros podem ser usados de forma múltipla, isto é, um ponteiro pode apontar para outro ponteiro. Por exemplo,

```
int x = 10;
int *p = &x;
int **q = &p;
```

Chamamos isto de **indireção múltipla**.

Exemplo 6. *Slide 5.*

Discussão:

- Faz sentido usar o operador `*` várias vezes? E o `&`?