Alocação de Memória

Programação e Desenvolvimento de Software I - DCC/UFMG

Thiago Melo de Oliveira

Jun/2019

Variáveis

Ao declararmos uma variável **`x`** como abaixo:

```
int x = 100;
```

Temos associados a ela os seguintes elementos:

- Um nome (x);
- Um endereço de memória ou referência (0xbfd267c4);
- Um valor (100).

O operador address-of (&)

Para acessarmos o endereço de uma variável, usamos o operador &:

```
printf("Endereço de x: %p\n", &x);
```

programa1.c

Ponteiros

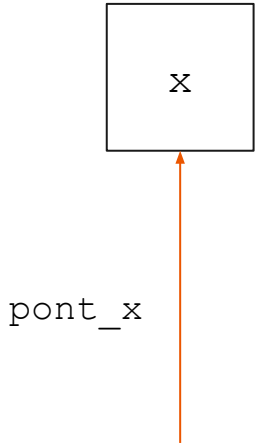
Existem tipos de dados para armazenar o endereços de variáveis:

```
int x;
x = 10;
int *pont_x; /* ponteiro para inteiros */
pont_x = &x; /* pont_x aponta para x */
```

Ponteiros

Existem tipos de dados para armazenar o endereços de variáveis:

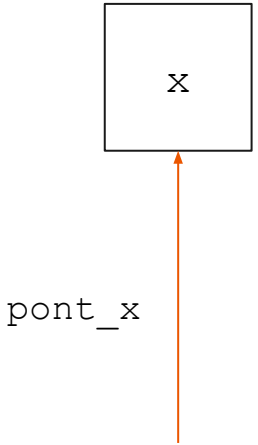
```
int x;
x = 10;
int *pont_x; /* ponteiro para inteiros */
pont_x = &x; /* pont_x aponta para x */
```



Ponteiros

Para declarar uma variável do tipo ponteiro utilizamos o operador: *

```
int *p_int;
char *p_char;
float *p_float;
double *p_double;
```



Ponteiros

Cuidado ao declarar vários apontadores em uma única linha.

O operador * deve preceder o nome da variável e não suceder o tipo que o apontador apontará.

```
int *pont_1, *pont_2, *pont_3;
```

Exercício: A declaração abaixo declara quantos inteiros e quantos apontadores para inteiro?

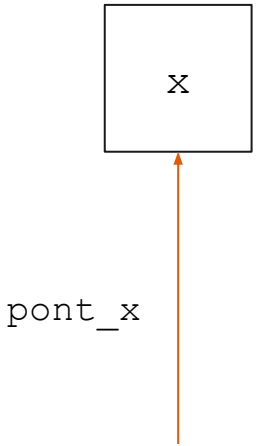
```
int *p1, p_ou_int1, p_ou_int2;
```

Ponteiros

Para acessarmos o valor de uma variável apontada por um endereço, também usamos o operador *.

Ao precedermos um apontador com este operador, obtemos o equivalente a variável armazenada no endereço em questão.

`*pont_x` pode ser usado em qualquer contexto que a variável **`x`** seria.



Referenciando e Desreferenciando

Referenciando (*referencing*): significa pegar o endereço de uma variável existente (usando &) para definir uma variável de ponteiro. Para ser válido, um ponteiro deve ser definido para o endereço de uma variável **do mesmo tipo** que o ponteiro, sem o asterisco.

```
int  x = 5;
int *y;
y = &x; /* y referencia x */
```

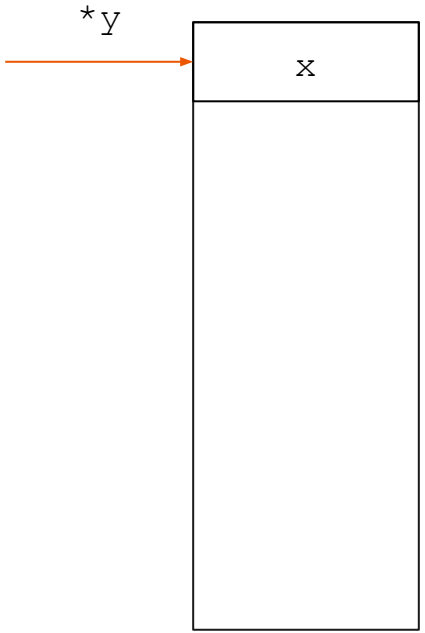
Desreferenciando

Desreferenciando (*dereferencing*): desreferenciar um ponteiro significa usar o operador * para recuperar o valor do endereço de memória apontado pelo ponteiro. O valor armazenado no endereço do ponteiro deve ser um valor **do mesmo tipo** que o tipo de variável que o ponteiro "aponta".

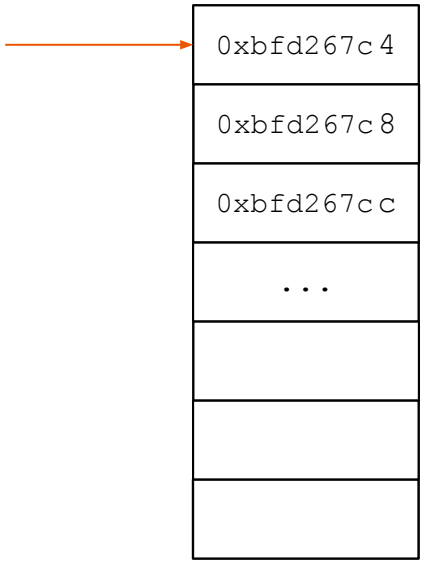
```
int  x = 5;
int *y;
y = &x;
int z;
z = *y; /* z desreferencia y */
```

programa2.c

Memória



Memória



Função *sizeof*

Este operador permite saber o número de bytes ocupado por um determinado tipo de variável.

Sintaxe: `sizeof(tipo);`

Exemplo:

A expressão `sizeof(float)` retorna o número de bytes ocupado por um float. Como o operador sizeof retorna um valor inteiro podemos visualizar esse valor usando %d.

programa3.c

Passagem de parâmetros por valor e referência

Ao passarmos argumentos para uma função, estes são copiados como variáveis locais da função. Isto é chamado passagem por valor.

Existe uma forma de alterarmos a variável passada como argumento, fazendo uma passagem por referência.

O artifício corresponde a passarmos como argumento o endereço da variável, e não o seu valor. Ou seja, o mecanismo usado na linguagem C para fazer chamadas por referência corresponde a passarmos ponteiros para as variáveis que queremos alterar na função.

programa4a.c

e

programa4b.c

e

programa4c.c


pass by reference
cup =
fillCup( )
pass by value
cup =
fillCup( )
www.penjee.com

Ponteiros e Vetores

Uma variável que representa um vetor é implementada por um ponteiro constante para o primeiro elemento do vetor. A operação de indexação corresponde a deslocar este ponteiro ao longo dos elementos alocados ao vetor. Isto pode ser feito de duas formas:

- Usando o operador de indexação: **v[4]**
- Usando aritmética de endereços: **programa5.c**

Este dupla identidade entre ponteiros e vetores é a responsável pelo fato de vetores serem sempre passados por referência e pela inabilidade da linguagem em detectar acessos fora dos limites de um vetor.

programa5.c

Alocação Dinâmica

Além de reservarmos espaços de memória com tamanho fixo na forma de variáveis locais, podemos reservar espaços de memória de tamanho arbitrário e acessá-los através de apontadores. A esse método damos o nome de alocação dinâmica de memória.

Desta forma podemos escrever programas mais flexíveis, pois nem todos os tamanhos devem ser definidos aos escrever o programa. A alocação e liberação destes espaços de memória é feito por duas funções da biblioteca `<stdlib.h>`:

- `malloc()`: Aloca um espaço de memória.
- `free()`: Libera um espaço de memória.

Função *malloc()*

Aloca um bloco consecutivo de bytes na memória e retorna o endereço deste bloco.

Para determinarmos o tamanho necessário, devemos usar a função `sizeof()`.

O espaço alocado por esta função pode ser usado para armazenar qualquer tipo de dados, logo devemos converter o tipo retornado `(void*)` para o tipo que iremos usar.

```
int *p;
p = (int*) malloc(100 * sizeof(int));
```

Função *free()*

Libera o uso de um bloco de memória, permitindo que este espaço seja reaproveitado.

Deve ser passado para a função free() exatamente o mesmo endereço retornado por uma chamada da função malloc() .

A determinação do tamanho do bloco a ser liberado é feita automaticamente.

```
int *p;
p = (int*) malloc(100 * sizeof(int));
free(p);
```

Vantagens de Alocação Dinâmica

A principal vantagem de usar a alocação dinâmica de memória é evitar o desperdício de memória.

Isso ocorre porque quando usamos alocação de memória estática, muita memória é desperdiçada, porque nem sempre toda a memória alocada será utilizada.

Assim, a alocação de memória dinâmica nos ajuda a alocar memória como e quando necessário e, portanto, economiza memória.

programa6.c

e

programa7.c

Dúvidas?

Obrigado! =)

thiagomelo@dcc.ufmg.br