Fundamentos de Programação
Incremento/decremento, operadores com atribuição, macros

Dainf - UTFPR

Profa. Leyza B. Dorini
Prof. Bogdan T. Nassu

Operadores de incremento e decremento

Operadores de incremento e decremento

A linguagem C possui dois operadores especiais: `++` e `--`

- `foo++` e `++foo` equivalem a `foo = foo+1`.
- `foo--` e `--foo` equivalem a `foo = foo-1`.

IMPORTANTE

Estes operadores são muito usados – entenda-os!

*Curiosidade: o nome C++ vem destes operadores.

Operadores de incremento e decremento

Se o operador estiver antes do nome da variável, ela é testada/usada antes de ser incrementada.

Se o operador estiver depois do nome da variável, ela é testada/usada depois de ser incrementada.

Exemplo: dada uma variável `foo` com valor 5...

- `foo++ > 5` retornará 0 (falso).
- `++foo > 5`retornará 1 (verdadeiro).

Em ambos os casos, o valor de `foo` após o teste será 6. Não esqueça que o valor da variável mudará com o incremento!

Quando usar estes operadores?

Para os programas que fizemos até agora, estes operadores não têm muita utilidade. Mas eles vão fazer *muito* sentido em breve!

Operadores com atribuição

Operações com atribuição

Expressões do tipo `a = a + b` são bastante frequentes. Por isso, a linguagem C oferece uma forma mais compacta de escrever este tipo de atribuição...

Expressão	Equivalente
`a = a + b`	`a += b`
`a = a - b`	`a -= b`
`a = a * b`	`a *= b`
`a = a / b`	`a /= b`
`a = a % b`	`a %= b`
`a = a << b`	`a <<= b`
`a = a >> b`	`a >>= b`
`a = a & b`	`a &= b`
`a = a | b`	`a |= b`
`a = a ^ b`	`a ^= b`

Macros

Macros

Em C, o caractere # indica uma diretiva de pré-processamento.

- Estas diretivas são interpretadas antes da compilação.
- Já vimos que `#include` "cola" o conteúdo de um arquivo.
- A diretiva `#define` define uma *macro*.

Sintaxe

```
#define <NOME> <CÓDIGO>
```

Cada ocorrência da macro no código é substituída pelo que estiver à direita do seu nome.

Exemplo

```
1  #define FAIXA1 1499.80
2  #define FAIXA2 2700.30
3
4  int main()
5  {
6    float salario, imposto);
7    scanf("%f", &salario);
8
9    if(salario < FAIXA1)
10     imposto = 0;
11   else if(salario < FAIXA2)
12     imposto = (salario - FAIXA1)*0.075;
13   else
14     imposto = (salario-FAIXA2)*0.15 + FAIXA2-FAIXA1*0.075;
15
16   printf("Imposto: %f", imposto);
17 }
```

O `define` pode estar fora de qualquer bloco. Na verdade, ele não é afetado pelo escopo dos blocos! Mas uma macro precisa ser declarada "antes" (acima) das linhas onde ela é é usada.

no código, usamos as macros

Atenção!

```
1  #define FAIXA1 1499.80
2  #define FAIXA2 2700.30
3
4  int main()
5  {
6    (...)
7    if(salario < FAIXA1)
8      imposto = 0;
9    (...)
10 }
```

note que não temos o ; aqui

Se tivéssemos, o ponto e vírgula na definição da macro, as aparições de `FAIXA1` seriam substituídas por `1499.80;` (e não por `1499.80`).

Usando o #define

Você pode colocar qualquer coisa à direita do nome. A expansão da macro funciona como um copiar/colar.

- `#define FOO foo`
  Qualquer ocorrência de `FOO` no código será substituída por `foo` (que pode ser um identificador, por exemplo).
- `#define BOOL char`
  Damos um "apelido" para um tipo de dados.
- `#define PI 3.1416`
  Podemos definir valores padrão para constantes numéricas...
- `define AUTHOR ``Bogdan Tomoyuki Nassu''`
  ... ou strings...
- `#define FOO`
  Podemos inclusive substituir `FOO` por nada! Isso simplesmente define que `FOO` é um símbolo que existe

Preste atenção à convenção!

Convenção

Nomes de macros são definidos com `LETRAS_MAIUSCULAS`.

Quando usar macros?

- Macros são boas ferramentas para "limpar" o código, evitando a presença de "números mágicos". Isto torna o código mais legível e, especialmente em programas grandes, facilita a depuração.
- Macros também facilitam a alteração de valores-padrão, mensagens de erro, etc.
- Mas cuidado!
  - O código é expandido sem qualquer teste.
  - Cuidado com coisas como ponto e vírgula, identificadores e palavras reservadas dentro de macros.