Algoritmos e Programação

Prof. Dr. Josenalde Barbosa de Oliveira

josenalde@eaj.ufrn.br

Aulas: 2T345, 4T345 – 6 CRDS – 90h

Atendimento presencial: 3 e 4: e-mail, discord

Análise e Desenvolvimento de Sistemas – UFRN

Variáveis e constantes

A regra para nomear variáveis em C++ é que só podem iniciar com letras ou o símbolo de underscore (_) e não devem conter caracteres especiais (ç, ~, ´, ‘, `, “, %, &, (,), { , }, [,]) nem operadores como +, -, *, /

```
float 2y; // ERRADO
float _y; // PERMITIDO
char p34; // PERMITIDO
short z*; // ERRADO
```

Constantes não podem ter o valor alterado ao longo do código e são criadas de duas formas: com a diretiva **#define** e com a palavra reservada **const**

```
#define PI 3.1416
int main() {
       double s = 10 + PI;
....
```

```
int main() {
   const double PI = 3.1416;
   double s = 10 + PI;
    ...
```

Conversão de Tipos (cast)

- É possível, em tempo de execução, converter temporiamente um valor de um tipo para outro, o denominado *typecast (casting)*

```
cout.precision(8);
cout << std::fixed;
int x = 5, y = 2;
cout << x/y << endl; // divisão inteira
double z = 204.34567890;
int zI; // parte inteira de z
double zF; // parte fracionaria de z
zI = (int)z;
zF = z - zI;
cout << "zI: " << zI << " zF: " << zF << endl;
cout << "zI+zF: " << zI+zF << endl;
```

Conversão de Tipos (cast)

- Atenção aos tipos float e double: por exemplo

```
float 204.34561 sendo exibido com 5 casas decimais – OK

float 204.345615 sendo exibido com 6 casas decimais – ERRO na
sexta casa em diante = 204.345612

double 204.345615 com 06 casas – OK

double 204.345615123456789 (15 casas) – OK até a casa 14
(precisão) = 204…80, 81, ou seja, precisão até a casa 14.

Avaliar na prática limites!! Pode depender do compilador
```

- Funções de conversão de tipos (cstdlib)
  - itoa (int para string – cadeia de caracteres), atoi (string para inteiro), atol, atof, ltoa
- Funções para teste de variáveis (cctype)
  - isdigit(int c), isalpha(int c), islower(), isupper, isspace(), tolower(), toupper() – CONVERSÕES DE CARACTERES, etc.
    #include <cctype> //for isalpha, isdigit, isalnum...

Tipo especial booleano

- É muito usado para definir testes e FLAGs ao longo de um Código fonte.

```
#include <iostream>
int main() {
     bool x = true, y;
     y =  !x;
  // ! é o operador de negação lógico
     cout << "x: " << x << "\t" << "y: " << y << endl;
     return 0;
}
```

Palavras-Chave de C++

- Também denominadas de Palavras Reservadas da linguagem C++.
- Não podem ser utilizadas para definições do usuário, de **variáveis**, **constantes**, **nomes de função, estruturas, tipos** etc.

Categoria	**Palavras-chave**
Tipos de dados	char, int, float, double, void
Modificadores de tipo	long, short, signed, unsigned
Modificadores de tipo de acesso	const, volatile
Classes de armazenamento	auto, extern, static, register
Tipos definidos pelo usuário	struct, enum, union, typedef
Comandos condicionais	if, else, switch, case, default
Comandos de laços	while, for, do
Comandos de desvio	break, goto, return, continue
Operador	sizeof

Constantes numéricas

- Uma constante tem valor fixo e inalterável. Números constantes em C++ podem ser escritos nas bases:

- DECIMAL: escreve-se de forma simples, como a que estamos acostumados. Exemplos: 10, 1020, 5, 84. Os números em decimal não podem estar entre aspas.
- HEXADECIMAL: os números na base 16 devem ser precedidos de 0x. Exemplos: 0x41, 0x1afb, 0x54CA.
- OCTAL: Os números escritos na base 8 devem ser precedidos por um zero. Exemplos: 041, 010, 0754. Escrever 10 não é a mesma coisa que escrever 010. 010 é o número 8 em decimal.
- CARACTERE: A numeração caractere só está definida para números entre 0 e 255 (1 byte). Caracteres constantes são escritos entre aspas simples (apóstrofos). Exemplos: o número 65 pode ser escrito como ‘A’. Exemplos: ‘5’, ‘a’, ‘\n’, ‘\t’.

Constantes numéricas

- Qual será o resultado para:

```
int main()
{
        int x = 0x0F;
        int y = x + 0x16;
        cout << std::hex << “x+16 (hex) =" << y << std::endl;
        cout << std::dec << “y em dec: "<< y << endl;
        cout << "y em dec: " << y << endl;
}
```

Aqui, temos a soma de um número HEXADECIMAL com outro HEXADECIMAL, e formatamos a saída para HEXADECIMAL. O compilador faz a operação no contexto HEXA e dá 25 como saída (HEX), o qual convertido para DEC dá 37.

Já neste caso:

```
int x = 0x0F;
int y = x + 16;
```

Neste caso, temos a soma de um número HEXADECIMAL com outro DECIMAL, e formatamos a saída para HEXADECIMAL. O compilador transforma F para decimal (**15**), soma a **16**. O resultado é **31** (decimal), mas como a saída é HEXADECIMAL, temos **1F** como resposta.

Operadores Aritméticos

Os operadores aritméticos em C++ são:

BINÁRIOS (operam sobre dois operandos)

+	Soma
-	Subtração
*	Multiplicação
/	Divisão (**se operandos são inteiros, retorna quociente inteiro)**
%	Módulo

O operador módulo opera somente com operandos inteiros e dá como resultado o resto da divisão do inteiro à sua esquerda pelo inteiro à sua direita. Um exemplo de uso é para determinar se um número é PAR ou ÍMPAR.

Como um número PAR é aquele divisível por 2, basta checar se o resto da divisão do número por 2 é 0, ou seja, X % 2 == 0 ?
Já 17 % 2, por exemplo, resulta 1 != 0, logo é ÍMPAR.

Operadores Aritméticos

Exemplo de uso: dado um valor inteiro pelo usuário, exibir quantas notas de R$ 100, 50, 20, 10, 5, 2 e moedas de R$ 1,00 são necessárias para compor o valor fornecido

```
Algoritmo comporValor
Início
   Leia(valor)
   N100 = valor / 100;
   R100 = valor % 100;
   N50 = R100 / 50;
   R50 = R100 % 50;
   N20 = R50 / 20;
   R20 = R50 % 20;
   N10 = R20 / 10;
   R10 = R20 % 10;
   N5 = R10 / 5;
   R5 = R10 % 5;
   N2 = R5 / 2;
   R2 = R5 % 2;
   M1 = R2;
   Exiba(N100,N50,N20,N10,N5,N2,M1)
Fim
```

```
Execução do Algoritmo comporValor
Seja valor = 673
N100 = 6
R100 = 73
N50  = 1
R50  = 23
N20  = 1
R20  = 3
N10  = 0
R10  = 3
N5   = 0
R5   = 3
N2   = 1
R2   = 1
M1   = 1
```

OBS: o operador resto da divisão % (ou mod) é implementado da seguinte forma:

Seja dois inteiros A e B:
Se A>B então R = A%B (resto)
Senão R = A

Operadores aritméticos

UNÁRIO (opera sobre um operando) :Menos unário: trocar valor algébrico

X = -8; X = -X; // X assume o valor 8 positivo.

PRECEDÊNCIA DE OPERADORES:

Indica qual operador deverá ser executado primeiro. Operadores tem Uma regra de precedência e de associatividade que determinam Exatamente como a expressão é resolvida, mas podemos usar parênteses para mudar a ordem de avaliação. Seja z = 2, y = 5, x = 2

a = z + y * x = 2 + 5 * 2 = 12
b = (z + y) * x = (2 + 5) * 2 = 20

Os operadores + e – tem a mesma precedência, e a regra é da "esquerda para a direita".

x – y + 5

Operadores aritméticos

PRECEDÊNCIA DE OPERADORES:

Os operadores *, / e % tem a mesma precedência e associatividade da esquerda para a direita.

Os operadores *, / e % tem precedência sobre os operadores + e -.

Operadores aritméticos de atribuição

```
+=, -=, *=, /=, %=

x += 2; // x = x + 2;
y *= x; // y = y * x;
z %= 10; // z = z % 10;
d /= 2; // d = d / 2;
```

Operadores aritméticos

Incremento ++ e Decremento --, pré-fixado e pós-fixado

O operador ++ equivale a somar uma unidade
O operador – equivale a subtrair uma unidade

Contudo, se usado numa operação de atribuição,
a localização pode gerar resultados diferentes. No uso sem atribuição, o resultado é o mesmo, indiferente. Exemplo:

```
x = 2; x++; // resulta x = 3
x = 2; ++x; // resulta x = 3
```

```
Pós-fixado
int x = 2, y;
y = x++;
// y = x;
// x += 1;
// No fim: y = 2, x = 3
```

```
Pré-fixado
int x = 2, y;
y = ++x;
// x += 1;
// y = x;
// No fim: y = 3, x = 3
```

Operadores relacionais

<	Menor relacional
<=	Menor ou igual relacional
>	Maior relacional
>=	Maior ou igual relacional
==	Igual relacional
!=	Diferente relacional

Utilizados em critérios de parada, testes lógicos para desvios etc.

Envolvidos em instruções que são avaliadas como true ou false e podem ser combinados com os operadores lógicos

Operadores lógicos

&& **E lógico (and)**

A	B	S
F	F	F
F	V	F
V	F	F
V	V	V

|| **OU lógico (or)**

A	B	S
F	F	F
F	V	V
V	F	V
V	V	V

! **NÃO lógico (not)**

A	S
F	V
V	F

Operadores lógicos

^ **OU exclusivo**

A	B	S
F	F	F
F	V	V
V	F	V
V	V	F

!^ **coincidência**

A	B	S
F	F	V
F	V	F
V	F	F
V	V	V

```
bool a = false;
bool b = false;
bool s = a^b;
cout << a << b << s;
a = false;
b = true;
s = a^b;
cout << a << b << s;
a = true;
b = false;
s = a^b;
cout << a << b << s;
a = true;
b = true;
s = a^b;
cout << a << b << s;
```

```
x = 5, z = 2, y = 10;
(((y > x) && (x % z != 0)) ^ !(z!=2))
```

V ou F?

Variáveis e operador de endereços de memória &

O operador & atua sobre o nome de uma variável e resulta o seu endereço na memória. É utilizado por exemplo na função SCANF. Um endereço é a referência que o computador usa para localizar variáveis. Toda variável ocupa uma certa localização na memória, e o seu endereço é o do primeiro byte ocupado por ela. Se você declarou uma variável Inteira, nomeou-a X e atribuiu a ela o valor 2, quando X for referida, obteremos 2. Entretanto se você usar **&X**, o resultado será o endereço do primeiro byte (byte menos significativo) ocupado por X.

```
int x = 2;
cout << "valor x: " << x << endl;
cout << "#x: " << &x << endl;
```

O resultado irá depender da máquina e memória do equipamento.
Exemplo: valor = 2, endereço = 61FF0C