Algoritmo

NOTAS DE AULA

Rafael Dias Ribeiro
rdribeiro@processware.com.br

PROCESSWARE
Consultoria e Treinamento

Algoritmo

É uma sequência de passos (instruções) finitos para a resolução de um problema.

Ex: Algoritmo de como fritar ovo

1- pegar frigideira, óleo, ovo e sal;
2- colocar óleo na frigideira;
3- acender fogo;
4- colocar frigideira no fogo;
5- esperar óleo esquentar;
6- colocar ovo;
7- reirar quando pronto.

OBS: Para o mesmo problema podem existir diversas soluções válidas, como podem existir diversos algoritmos diferentes que solucionam o mesmo problema.

Um programa de computador é um algoritmo escrito em uma determinda linguagem para computadores (linguagem de programação), como:

- C;
- C++;
- Pascal;
- Basic;
- Java;
- Delphi;

e diversas outras linguagens existentes no mercado.

Em nosso curso, usaremos a linguagem C++ para exemplificação dos algoritmos.

Linguagem C++

A linguagem C++ foi desenvolvida inicialmente por Bjarne Stroustrup na AT&T, de 1979 a 1983, à partir da linguagem C, tendo como idéia principal a de agregar o conceito de classes, de orientação à objetos, àquela linguagem.

A partir da primeira versão de 1983, a linguagem foi sendo revisada e evoluindo, tornou-se disponível fora da AT&T em 1985, e após um longo processo foi padronizada pela ISO no final de 1997, pelo padrão ISO/IEC 14882.

De forma geral, é possível obter muito material de referência da linguagem através de páginas na Internet. Tente por exemplo pesquisar com a expressão *"C++ reference"* no Google.

Os compiladores do projeto GNU também podem ser obtidos livremente através da Internet. O ambiente BloodShed Dev-C++ roda sobre Windows e utiliza os compiladores gcc e g++.

É possível baixá-lo de: http://www.bloodshed.net/devcpp.html.

Há também inúmeros fóruns e listas de discussão sobre aspectos técnicos da linguagem. Então, use e abuse da Internet para aprendê-la e resolver seus problemas.

Variável

É um objeto (uma posição, freqüentemente localizada na memória) capaz de reter e representar um valor ou expressão. Enquanto as variáveis só "existem" em tempo de execução, elas são associadas a "nomes", chamados identificadores, durante o tempo de desenvolvimento.

Uma variável consiste em um elemento ao qual lhe damos um nome e lhe atribuímos um determinado tipo de informação.

Em diversas situações precisamos armazenar um determinado valor para uma futura utilização.

Tipos de Variáveis

Uma variável armazena um determinado conteúdo. Este conteúdo pode ser um número inteiro, um número real, um texto, um booleano ou um outro tipo de conteúdo válido.

Assim, precisamos informar para a máquina qual o tipo de conteúdo que irá ser armazenado naquela variável e qual o nome da variável (para uma futura referência ao valor armazenado).

Ex:

Tipos de Variáveis

A tabela abaixo apresenta os tipos de dados básicos da linguagem, exemplos de valores literais, e intervalos de valores aceitos.

Cada tipo de dado possui representação de valores literais adequados, tanto para inicialização quanto para atribuições. Em geral usa-se *bool*, *char, int*, *long* e *double* mais freqüentemente, onde *long* é uma abreviação para *long int*.

bool	Valores booleanos	True false	True false
char	Caracteres simples	'a' 'A' '+' '\t' '\n'	0 a 255, isto é, 8 bits
int	Números inteiros	100 0x3F	16 bits ou mais, 32 normal
float	Números reais	1.2f .23f 1.f 1.5e-15f	
double	Reais dupla precisão	1.2 .23 1. 1.5e-15	

Tipos de Variáveis

Há quatro modificadores que alteram a representação desses tipos básicos. Os modificadores ***signed*** e ***unsigned*** alteram o significado dos dados para representação de números negativos. Os modificadores ***short*** e ***long*** alteram a quantidade de bits com que o dado é representado. A linguagem não padroniza completamente o tamanho da representação binária de números, e esse é um aspecto que pode comprometer a portabilidade.

Assim como C, C++ é uma linguagem que permite programação em baixo nível. Assim, a linguagem não possui propriamente caracteres, mas o tipo *char* é apenas um número de no mínimo 8 bits, e os literais (*'a','X'*) podem ser atribuídos a qualquer variável numérica.

Operadores Aritméticos

As cinco operações aritméticas suportadas pela linguagem são:

+ Adição
- Subtração
* Multiplicação
/ Divisão
% Módulo

Operações de adição, subtração, multiplicação, e divisão não são um desafio de entendimento para você, já que elas correspondem literalmente com seus respectivos operadores matemáticos.

O único que pode não ser conhecido por você é o **módulo**, especificado com o sinal de porcentagem (%). Módulo é a operação que dá o resto de uma divisão de dois valores inteiros.

Por exemplo, se escrevermos **a = 11 % 3;**, a variável **a** irá conter **2** como resultado já que **2** é o resto da divisão de 11 por 3.

Operador de Atribuição

O operador de atribuição serve para atribuir um valor a uma variável.

```
x = 5;
```

Atribui o valor inteiro **5** para a variável **x**.

A parte da esquerda do operador = é conhecida como *lvalue* (left value – valor da esquerda) e a da direita como *rvalue* (right value – valor da direita).

lvalue precisa sempre ser uma variável, enquanto o lado direito pode ser uma constante, uma variável, o resultado de uma operação ou qualquer combinação entre eles. É preciso enfatizar que a operação de atribuição sempre funciona da direita para a esquerda e nunca o oposto.

```
x = y;
```

Atribui para a variável **x** (*lvalue*) o valor contido na variável **y** (*rvalue*) independentemente do valor que estiver guardado em **x** nesse momento.

Operador de Atribuição

Considere também que estamos apenas atribuindo o valor de **y** para **x** e que uma mudança futura em **y** não afetará o valor de **x**.

Por exemplo:

```
int x, y;      // x:?  y:?
x = 10;        // x:10 y:?
y = 4;         // x:10 y:4
x = y;         // x:4 y:4
y = 7;         // x:4 y:7
```

Isso nos dará o resultado que o valor contido em **x** é **4** e o valor contido em **y** é **7**. A modificação final de **y** não afetou **x**, mesmo sabendo que havíamos declarado **x = y;** (regra da direita-para-esquerda).

Operador de Atribuição

Uma propriedade que C++ tem sobre outras linguagens de programação é que a operação de atribuição pode ser usada como *rvalue* (ou parte de um *rvalue*) para outra atribuição. Por exemplo:

```
x = 2 + (y = 5);
```

é equivalente a:

```
y = 5;
x = 2 + b;
```

Primeiro atribui **5** para a variável **y** e então atribui para **x** o valor **2** mais o resultado a atribuição anterior de **y** (que é 5), deixando **x** com um valor final de **7**.

Assim, a seguinte expressão também é válida em C++:

```
x = y = w= 5;
```

atribui 5 às três variáveis: **x**, **y** e **w**.

Operadores de Atribuição Compostos

Uma característica de atribuição em C++ que contribui para sua fama de boa escrita de linguagem são os operadores de atribuição compostos (**+=**, **-=**, ***=** e **/=** entre outros), que permitem modificar o valor de uma variável com um dos operadores básicos:

value += increase; é equivalente a **value = value + increase;**

x -= 5; é equivalente a **x = x- 5;**

x /= b; é equivalente a **x = x / b;**

price *= units + 1; é equivalente a **price = price * (units + 1);**

e o mesmo para todas as outra operações.

Incremento (++) e Decremento (--)

Eles aumentam ou reduzem 1 ao valor guardado na variável. Por exemplo:

```
X++;
X+=1;
X=X+1;
```

são todos equivalentes e suas funcionalidades são incrementar 1 no valor de X.

Uma característica desse operador é que pode ser usado tanto como *prefixo* ou como *sufixo*. Isso significa que pode ser escrito antes do identificador da variável (++X) ou depois (X++).

Incremento (++) e Decremento (--)

Em operações o qual o resultado da operação de *incremento* ou *decremento* é avaliado como outra expressão eles podem ter uma diferença importante em seus significados:

Em caso do operador de incremento ser usado como *prefixo* (++a), o valor é incrementado antes de a expressão ser avaliada e, sendo assim, o valor incrementado é considerado na expressão;

Em caso de ser usado como *sufixo* (a++), o valor guardado em a é incrementado depois de ter sido avaliada e, sendo assim, o valor guardado antes da operação de incremento é avaliada na expressão. Note a diferença:

```
Z=3;
X=++Z;

// X é 4, Z é 4
```

```
Z=3;
X=Z++;

// X é 3, Z é 4
```

Operadores Relacionais (==, !=, >, <, >=, <=)

Para que seja possível avaliar uma comparação entre duas expressões, podemos usar os operadores relacionais. Conforme especificado no padrão ANSI-C++, o resultado de uma operação relacional é um valor **bool** que pode ser somente **true** ou **false**, de acordo com o resultado da comparação.

Nós podemos querer comparar duas expressões, por exemplo, para saber se elas são iguais ou se uma delas é maior que a outra. Aqui está uma lista com os operadores relacionais que podem ser utilizados no C++:

==	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Operadores Relacionais (==, !=, >, <, >=, <=)

Aqui você tem alguns exemplos:

(7 == 5) Retornaria **false**.
(5 > 4) Retornaria **true**.
(3 != 2) Retornaria **true**.
(6 >= 6) Retornaria **true**.
(5 < 5) Retornaria **false**.

é claro que, ao invés de usar somente constantes numéricas, podemos usar qualquer expressão válida, incluindo variáveis. Suponha que **X=2**, **Y=3** e **Z=6**,

(X == 5) Retornaria **false**.
(X*Y >= Z)Retornaria **true** já que (2*3 >= 6) é verdadeiro.
(Y+4 > X*Z)Retornaria **false** já que (3+4 > 2*6) é falso.
((Y=2) == X) Retornaria **true**.

Operadores Lógicos

O operador **!** é equivalente à operação boleana NOT, possui somente um operando localizado à direita, e a única coisa que faz é inverter o valor desse operando, gerando **false** se o operando for **true** e **true** se o operando for **false**.

É o mesmo que dizer que retorna o resultado oposto da avaliação do operando. Por exemplo:

!(5 == 5) Retorna **false** porque a expressão à direita (5 == 5) seria **true**.
!(6 <= 4) Retorna **true** porque (6 <= 4) seria **false**.
!true Retorna **false**.
!false Retorna **true**.

Operadores Lógicos

Os operadores lógicos **&&** (and) e **||** (or) são usados ao avaliar duas expressões para obter um resultado único.

Eles correspondem às operações lógicas boleanas *AND* e *OR* respectivamente. Seus resultados dependem da relação entre seus dois operandos:

Operando 1	Operando 2	Operando 1 && Operando 2	Operando 1 || Operando 2
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

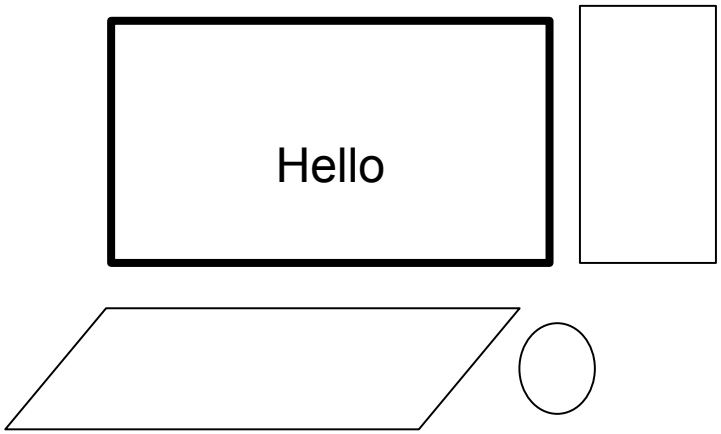
Por exemplo:

((5 == 5) && (3 > 6)) retorna **false** (*true* && *false*).

((5 == 5) || (3 > 6)) retorna **true** (*true* || *false*).

Primeiro Programa em C++

```
#include <iostream>

 using namespace std;

 int main() {

//última atualização: 12/05/07

cout << "Hello\n";

// O cout é uma instrução
 }
```



Primeiro Programa em C++

#include

Serve para fazer com que o compilador inclua como parte desse código outro arquivo de código fonte.
Em geral, esse recurso é usado para incluir definições de dados e código que serão utilizados por nosso programa, mas já foram compilados e estão disponíveis em uma biblioteca.

É função da etapa de ligação montar o executável final incluindo o código compilado para o nosso programa com o código compilado dessas bibliotecas.

Primeiro Programa em C++

Namespace

O ***namespace*** é um recurso utilizado para evitar que, quando se constróem grandes programas, nomes de variáveis, classes e funções conflitem.

Servem para separar espaços de nomes em módulos e bibliotecas inteiras, evitando que nomes de variáveis, estruturas, funções e classes conflitem.

Os *namespaces* são abertos, isto é, podem ser declarados várias vezes, cada qual acrescentando novos elementos dentro do mesmo *namespace*.

Primeiro Programa em C++

Função MAIN( )

Todo programa em C++, assim como em C, pode ser visto como um conjunto de funções.

As funções serão apresentadas mais adiante, mas é necessário saber primeiro que todos os comandos da linguagem devem estar dentro de funções, e que, portanto, um programa deve ter no mínimo uma função.

Essa função principal tem o nome de *main*, sendo esse nome o que identifica por onde a execução inicia.

Comentários

Comentários são pedaços de código-fonte descartados do código pelo compilador. Eles não fazem nada.

O objetivo deles é somente permitir que o programador insira notas ou descrições dentro do código-fonte.

O C++ suporta duas maneiras para inserir comentários:

// comentário de linha

/* comentário de bloco */

O primeiro deles, o comentário de linha, descarta tudo desde onde o par de barras (//) é achado até o final daquela mesma linha.

O segundo deles, o comentário de bloco, descarta tudo entre os caracteres /* e a próxima aparição dos caracteres */, com a possibilidade de incluir diversas linhas.

Comandos de Entrada e Saída

Comando de Saída (COUT <<)

O comando cout é usado em conjunto com o operador sobrecarregado << (um par de sinais de *"menor que"*).

```
cout << “Texto de saida"; // imprime texto de saida na tela
cout << 50;               // imprime o número 50 na tela
cout << x;                // imprime o conteúdo da variável x na tela
```

O operador << é conhecido como *operador de inserção*, pois insere o dado que vem a seguir no comando que o precede. No exemplo acima, insere a string constante Texto de saida, a constante numérica 50 e a variável x no comando de saída cout.

Note que a primeira das três frases está entre aspas duplas (") porque é uma string de caracteres. Sempre que quisermos usar strings de caracteres constantes, precisamos colocá-las entre aspas duplas (") para que possam ser claramente distinguidas de variáveis.

Comandos de Entrada e Saída

Comando de Saída (COUT <<)

Por exemplo, essas duas frases são bastante diferentes:

```
cout << "Ola";      // imprime Ola na tela
cout << Ola;        // imprime o conteúdo da variável Ola na tela
```

O operador de *inserção* (<<) pode ser usado mais de uma vez em uma mesma frase:

```
cout << "Ola, " << "eu sou " << "uma frase de C++";
```

Essa última frase imprimiria a mensagem Ola, eu sou uma frase de C++ na tela.

A utilidade em repetir o operador de inserção (<<) é demonstrada quando queremos imprimir uma combinação de variáveis e constantes ou mais de uma variável:

```
cout << "Ola, eu tenho " << age << " anos de idade e meu cep e " << zipcode;
```

Se supusermos que a variável age contém o número 24 e a variável zipcode contém 90064, a saída da frase anterior seria:

```
Ola, eu tenho 24 anos de idade e meu cep e 90064
```

Comandos de Entrada e Saída

Comando de Saída (COUT <<)

É importante notar que cout não adiciona uma quebra de linha depois de sua saída a não se que indiquemos isso explicitamente, sendo assim, as seguintes frases:

```
cout << "Essa e uma frase.";
cout << "Essa e outra frase.";
```

serão exibidas em seguida na tela:

Essa e uma frase.Essa e outra frase.

Mesmo se as tivéssemos escrito em duas chamadas diferentes de cout. Então, para adicionar uma quebra de linha na saída, precisamos escrever isso explicitamente inserindo um caractere de nova linha, que em C++ pode ser escrito como \n:

```
cout << "Primeira frase.\n ";

cout << "Segunda frase.\nTerceira frase.";
```

Produz a seguinte saída:

Primeira frase.

Segunda frase.

Terceira frase.

Comandos de Entrada e Saída

Comando de Saída (COUT <<)

Para adicionar uma nova linha, você também pode usar o manipulador endl.

Por exemplo:

```
cout << "Primeira frase." << endl;
cout << "Segunda frase." << endl;
```

imprimiria:
Primeira frase.
Segunda frase.

O manipulador endl tem um comportamento especial quando usado com comandos que utilizam buffer: eles são descarregados. Mas de qualquer maneira, cout não usa buffer por padrão.

Você pode usa tanto o caractere de escape \n quanto o manipulador endl para especificar uma quebra de linha no cout.

Comandos de Entrada e Saída

Comando de Saída (COUT <<)

Barra invertida ("\") é usada em literais para criar caracteres especiais, por exemplo:

"\n" é um *newline*
"\b" é um *backspace*
"\t" é um *tab*
"\a" é um *alert*
"\v" é um *tab vertical*

Comandos de Entrada e Saída

Comando de Entrada (CIN >>)

O trabalho com a entrada padrão no C++ é feito aplicando-se o operador sobrecarregado de *extração* (>>) no comando cin.

Isso precisa ser seguido pela variável que irá guardar o dado que será lido.

Por exemplo:
```
int age;
cin >> age;
```

Declara a variável age como um int e então espera por uma entrada do cin (teclado) para que possa guardá-la em uma variável inteira.

cin só pode processar a entrada do teclado depois que a tecla ENTER for pressionada. Sendo assim, mesmo que você peça um único caractere, o cin não irá processar a entrada até que o usuário pressione ENTER depois que o caractere tenha sido digitado.

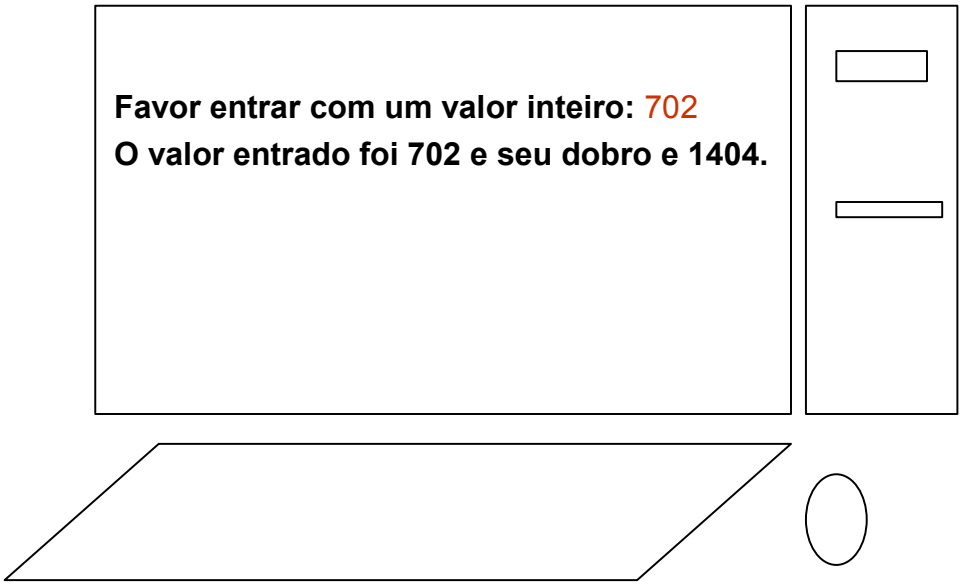
Comandos de Entrada e Saída

Comando de Entrada (CIN >>)

Você precisa sempre considerar o *tipo* da variável que você está usando para guardar o valor extraído pelo cin.

Se você pedir um inteiro, você receberá um inteiro, se você pedir um caractere, você receberá um caractere, e se você pedir uma string de caracteres, você receberá uma string de caracteres.

```
#include <iostream.h>

using namespace std;

main ()
{
int i;
cout << "Favor entrar com um valor inteiro: ";
cin >> i;
cout << "O valor informado foi " << i;
cout << " e seu dobro e " << i*2 << ".\n";
}
```



Comandos de Entrada e Saída

Comando de Entrada (CIN >>)

O usuário de um programa pode ser uma das razões que provocam erros mesmo quando o mais simples dos programas é usado com **cin** (como o que nós acabamos de ver). Isso porque se você requisitar um valor inteiro e o usuário entrar com um nome (que é uma string de caracteres), o resultado pode fazer com que seu programa funcione de maneira incorreta, pois não é o que se esperava do usuário.

Então quando você usar a entrada de dados provida pelo **cin**, você precisará confiar que o usuário de seu programa irá cooperar totalmente e que ele não irá entrar com um nome quando um valor inteiro for requisitado.

Você também pode usar **cin** para requisitar mais que um dado do usuário:
cin >> a >> b;

é equivalente a:
cin >> a;
cin >> b;