Programando em C++

Joaquim Quinteiro Uchôa

joukim@comp.ufla.br

DCC-UFLA, 2002

Histórico da Linguagem C

- Linguagem C: 1972 - Laboratório Bells, por Dennis Ritchie, a partir da linguagem B de Ken Thompson
- PDP-11: primeira implementação
- UNIX e C: união forte
- 1978: Ritchie & Thompson lançam "The C Programming Language"
- Padronização ANSI em 1989

Histórico da Linguagem C++

- Iniciou-se em 1980 nos Laboratórios Bell por Bjarne Stroustrup
- Inicialmente chamado de "C com classes"
- Orientação a Objetos
- Padronização ANSI (bem recente)
- Compatibilidade com Linguagem C

Porque Usar C++?

- portabilidade
- modularidade
- compilação separada
- recursos de "baixo"e "alto"nível
- geração de código eficiente
- confiabilidade
- regularidade
- simplicidade
- facilidade de uso

Como Usar?

Compilador escolhido: Gnu C Compiler (gcc) - versões Linux, Cygwin ou Mingw32

Use um editor de sua preferência

Compile em linha de comando:

```
g++ arquivo.cpp -o executavel
```


Hello World

```
#include <iostream>

// tradicional "hello world" em c++
/* escreve "Hello World" na tela
   do computador */

int main( ){
    cout << "Hello World" << endl;
    return 0;
}
```

Comentários

Comentários de fim de linha:

- são inseridos com “`//`”
- valem de onde começam até o fim da linha
- deve ser o estilo preferido

Comentários tipo C:

- são inseridos entre “`/*`” e “`*/`”
- tudo entre “`/*`” e “`*/`” fica sendo comentário

Entrada e Saída (i)

Operador “<<” significa “coloca”:

- é operador de saída
- escreve

`cout`:

- objeto da classe iostream
- representa a saída padrão (geralmente o monitor)
- pode receber o encadeamento de várias saídas
- saída pode ser redirecionada

Entrada e Saída (ii)

Operador “>>” significa “recebe”:

- é operador de entrada
- lê

`cin`:

- objeto da classe iostream
- representa a entrada padrão (geralmente o teclado)
- pode atribuir o encadeamento de várias entradas
- entrada pode ser redirecionada

Entrada e Saída (iii)

`cerr`:

- objeto da classe iostream
- representa a saída padrão de erros (geralmente o monitor)
- pode receber o encadeamento de várias saídas
- saída pode ser redirecionada

Entrada e Saída (iv)

Exemplos de uso do cout:

```
cout << "Isto   um teste" << endl;
cout << "Outro teste\n";
cout << "Um teste" << " mais "
     << "complicado" << "!\n";
```

Exemplo de uso do cin:

```
int x;
cin >> x;
int z;
int w;
cin >> z >> w;
```


Declaração de Variáveis

A forma de declaração de variáveis é:

```
tipo nome;
```

O nome deve ser iniciado por letras do alfabeto inglês, seguido de outras letras, sublinhado “_” ou números.

Exemplos:

```
int x;
string nomeDoAluno;
float nota_do_aluno;
double xpqp1;
char* nomeDoProfessor;
minhaClasse meuObjeto;
```

Tipos de Variáveis - Inteiros

- char (1 byte) - também armazena caracteres
- int (4 bytes)
- short = short int (2 bytes)
- long = long int (4 bytes)
- signed = com sinal
- unsigned = sem sinal

Tipos de Variáveis - Flutuantes

- float
- double
- long double

Tipos de Variáveis - Outros

- Booleanas: bool (true, false)
- Ponteiros: *
- Arrays: [ ]
- Enumerações
- Classes (string, vector, etc.)

Classes

```
class date{

  private:
    int dia;
    int mes;
    int ano;

  public:
    date(int d, int m, int a);
    print();
    void set(int d, int m, int a);
};
```

Tamanho de Tipos

- Pode-se usar o operador `sizeof( )`
- No caso de strings do tipo C, deve-se usar `strlen( )`
- No caso da classe string, deve-se usar a função `length( )`
- **Não deve-se usar `sizeof( )` em ponteiros!!!**

Estruturas de Controle

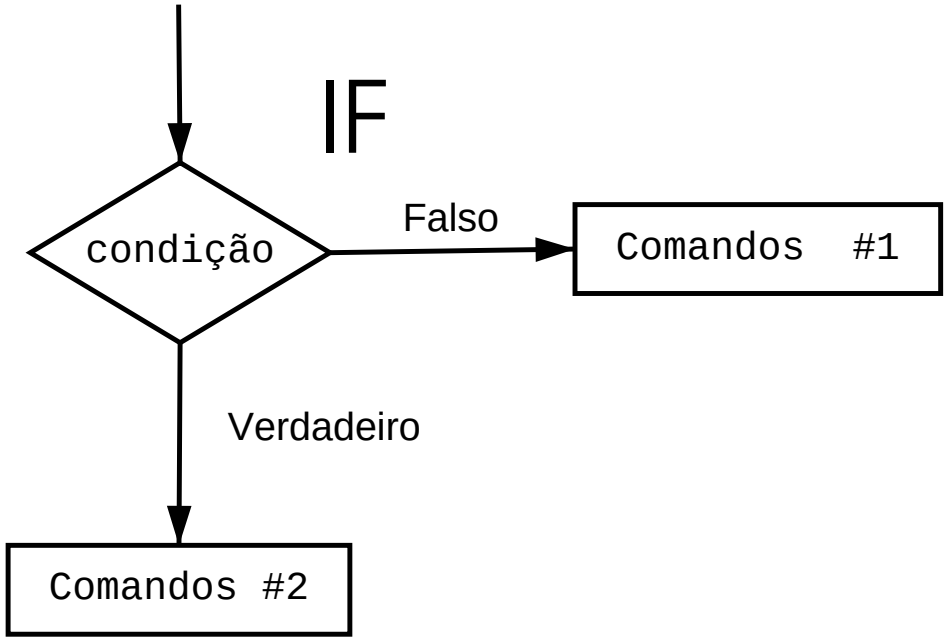
Estruturas condicionais:

- if
- if ... else
- switch ... case

Estruturas de repetição:

- while
- do ... while
- for

Estrutura condicional `if`



Uso do `if`

```
if (condicao)
  comando;
```

ou

```
if (condicao) {
  comandos
}
```

Uso do `if ... else`

```
if (condicao)
  comando;
else
  comando;
```

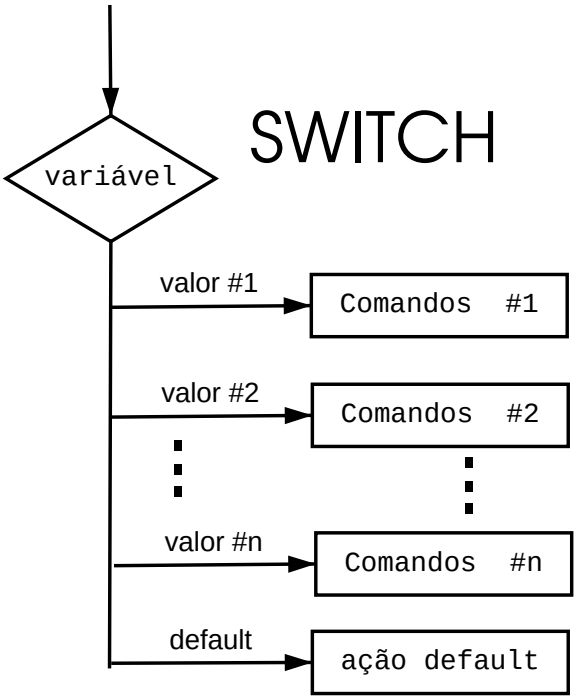
ou

```
if (condicao){
  comandos
}
else {
  comandos;
}
```


Exemplo de uso do `if`

```
if (x < 0) {
  cout << "menor";
  z++;
}
else if (x > 0){
  cout << "maior";
  z--;
}
else { // (x = 0){
  cout << "igual";
  i++;
}
```

Estrutura condicional `switch`



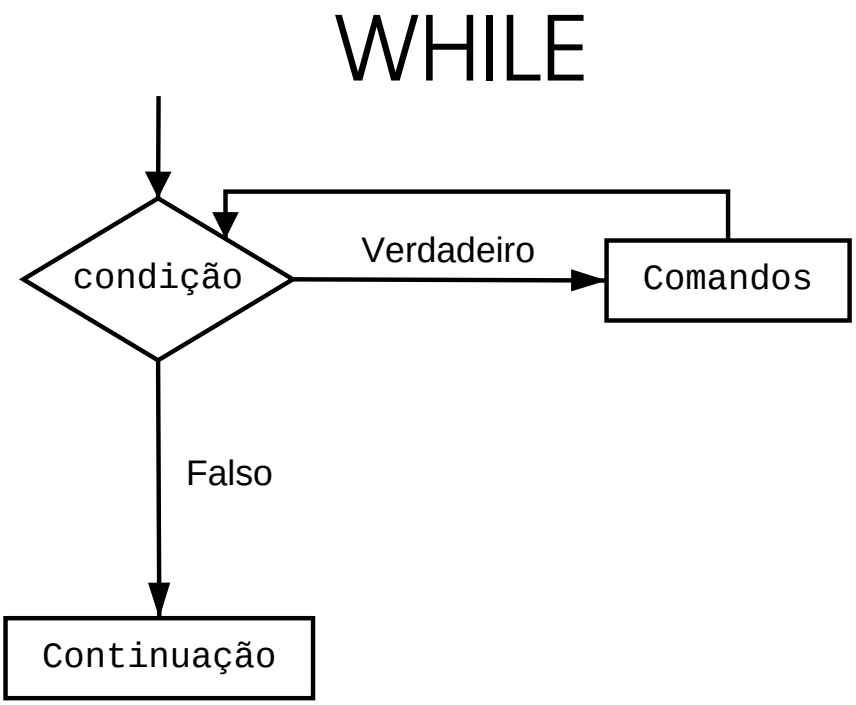
Uso do `switch`

```
switch (escolha) {
case valor1 :
    comandos1;
    break;
  ⋮
case valorn:
    comandosn;
    break;
default:
    acao_default;
}
```

Exemplo de uso do `switch`

```
switch (escolha) {
  case 1:
    cout << "Voce digitou 1\n";
    break;
  case 2:
    cout << "Voce digitou 2\n";
    break;
  default:
    cout << "escolha incorreta!\n";
}
```


Estrutura de repetição `while`



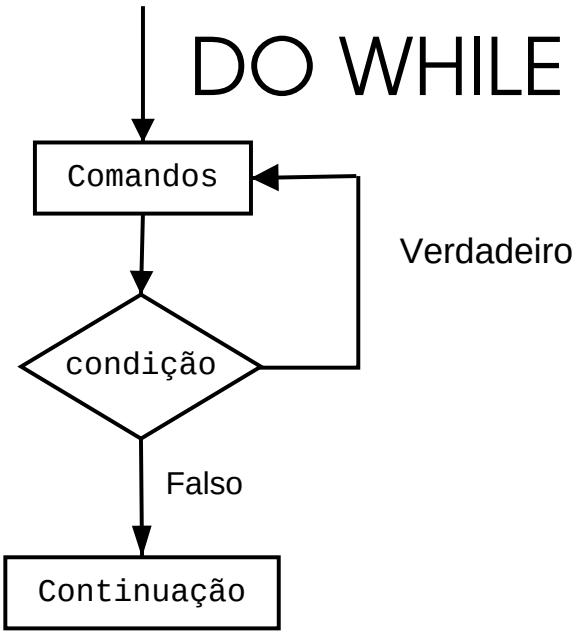
Uso do `while`

```
while (condicao)
  comando;

 ou

while (condicao){
  comandos
}
```


Exemplo de uso do `while`

```
int i = 0;

while (i < 10) {
  cout << i << " ";
  i++;
}
```

Estrutura de repetição do ... while



Uso do do ... while

```
do
  comandos;
while (condicao)
```

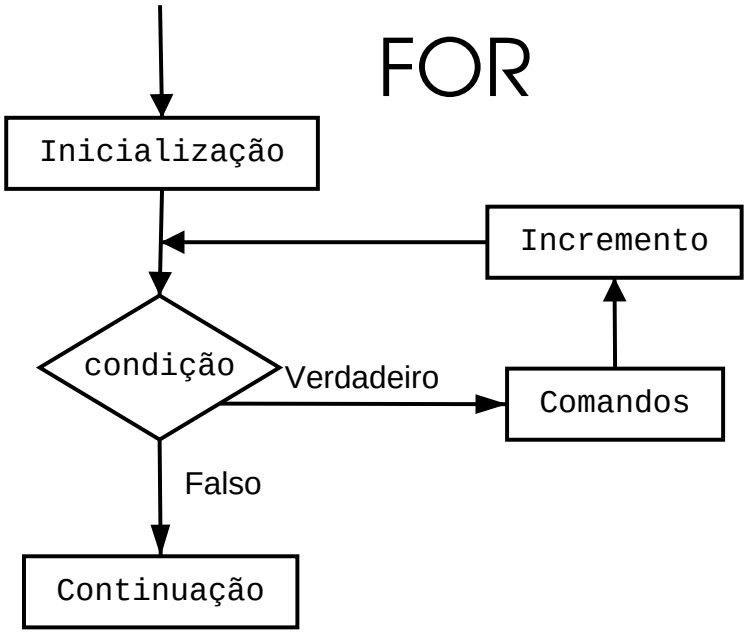
ou

```
do {
  comandos;
} while (condicao)
```

Exemplo de uso do `do ... while`

```
int i = 0;

do {
  cout << i << " ";
  i++;
} while (i < 10)
```


Estrutura de repetição `for`



Uso do `for`

```
for (inicializacao; condicao; incremento)
  comando;
```

ou

```
for (inicializacao; condicao; incremento){
  comandos;
}
```


Exemplo de uso do `for`

```
int j;

for (j= 0; j <= 10; j++) {
  cout << j << " ";
}


for (int i = 0; i <= 10; i++) {
  cout << i << " ";
}
```

Funções

Uma função em c++ é declarada com a seguinte sintaxe:

```
tipo nome(parametros);
```

e é implementada como

```
tipo nome(parametros){
    comandos;
}
```

parâmetros é uma lista do tipo

```
tipo1 var1, tipo2 var2,..., tipon varn
```


Exemplos de funções

```
double triplo(double x){
  return 3*x;
}

void (string nome="anonimo"){
  cout << "Ol, " << nome << "!";
}

inline double quadrado(double x){
  return x*x;
}
```

Ponteiros

- Aponta para uma variável
- Armazena o endereço desta variável
- Possibilita alocação dinâmica de memória
- Possui a forma “`tipo *nome`” ou “`tipo* nome`”
- Deve ser utilizado com cuidado!

Ponteiros