C++ Boolean Expressions and Selection Statements

November 15, 2010

C++ Code Path Control

Programs that can “make decisions” are better than programs that cannot.

1. Decide whether user input is valid.
2. Decide if special calculations are necessary ($\sqrt{-4} = 2i$).
3. Execute a sequence of instructions a specific number of times.

All of these use Boolean expressions to as the fundamental building block of program logic.

Relational Operators

How do two numbers relate to each other

==	equality
!=	non-equality
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

```
1 cout << ( 0 < 1 ) << endl;
2 cout << ( 1 <= 0 ) << endl;
3 cout << ( -1.2345 >= -1.2345 ) << endl;
4 cout << ( -1.2345 > -1.2345 ) << endl;
```

```
1
0
1
0
```

RUN EDIT relational_ops.cxx

Logical Operators

Combining True and False

&&	AND
||	OR

Logical AND

	T	F
T	T	F
F	F	F

Logical OR

	T	F
T	T	T
F	T	F

Logical Operators

Combining True and False

&&	AND
||	OR

Logical AND

	T	F
T	T	F
F	F	F

Logical OR

	T	F
T	T	T
F	T	F

Inverting or Negating a Boolean Value

!	NOT

Logical NEGATION

T	F
F	T

Logicial Operator Examples

```
1 cout << ( 0 < 1 ) << endl;
2 cout << ( 1 <= 0 ) << endl;
3 cout << (( 0 < 1 ) || ( 1 <= 0 )) << endl ;
4 cout << endl;
5 cout << ( -1.2345 >= -1.2345 ) << endl;
6 cout << ( -1.2345 > -1.2345 ) << endl;
7 cout << (( -1.2345 >= -1.2345 ) &&
8         ( -1.2345 > -1.2345 )) << endl;
```

```
1
0
1

1
0
0
```

RUN EDIT logical_ops.cxx

```
1 bool ecks, why;
2 cout << "Enter␣Bools␣for␣x␣and␣y:␣" << flush;
3 cin >> ecks >> why;
4
5 bool andResult = ecks && why;
6 bool orResult = ecks || why;
7 bool notOrNotResult = !( ecks || !why );
8
9 cout << "For␣x=" << ecks <<
10        "␣and␣y=" << why << endl;
11 cout << "(x␣&&␣y)␣is␣" << andResult << endl;
12 cout << "(x␣||␣y)␣is␣" << orResult << endl;
13 cout << "!(x␣||␣!y)␣is␣" <<
14        notOrNotResult << endl;
```

```
<<Interactive Program>>
```

RUN EDIT logical_ops_interactive.cxx

The (Growing) Precedence Table

Precedence	Operator(s)	Associativity	Notes
First	( )	innermost	
:	Unary: `++ -- .`	⇒	Postfix++
:	Unary: `++ -- + - cast()`	⇐	++Prefix
:	Binary: `* / %`	⇒	
:	Binary: `+ -`	⇒	
:	Relational: `< <= > >=`	⇒	
:	Relational: `== !=`	⇒	
:	Logical: `&&`	⇒	
:	Logical: `||`	⇒	
Last	Assignment: `= += -= *= /= %=`	⇐	

Lecture Question 2

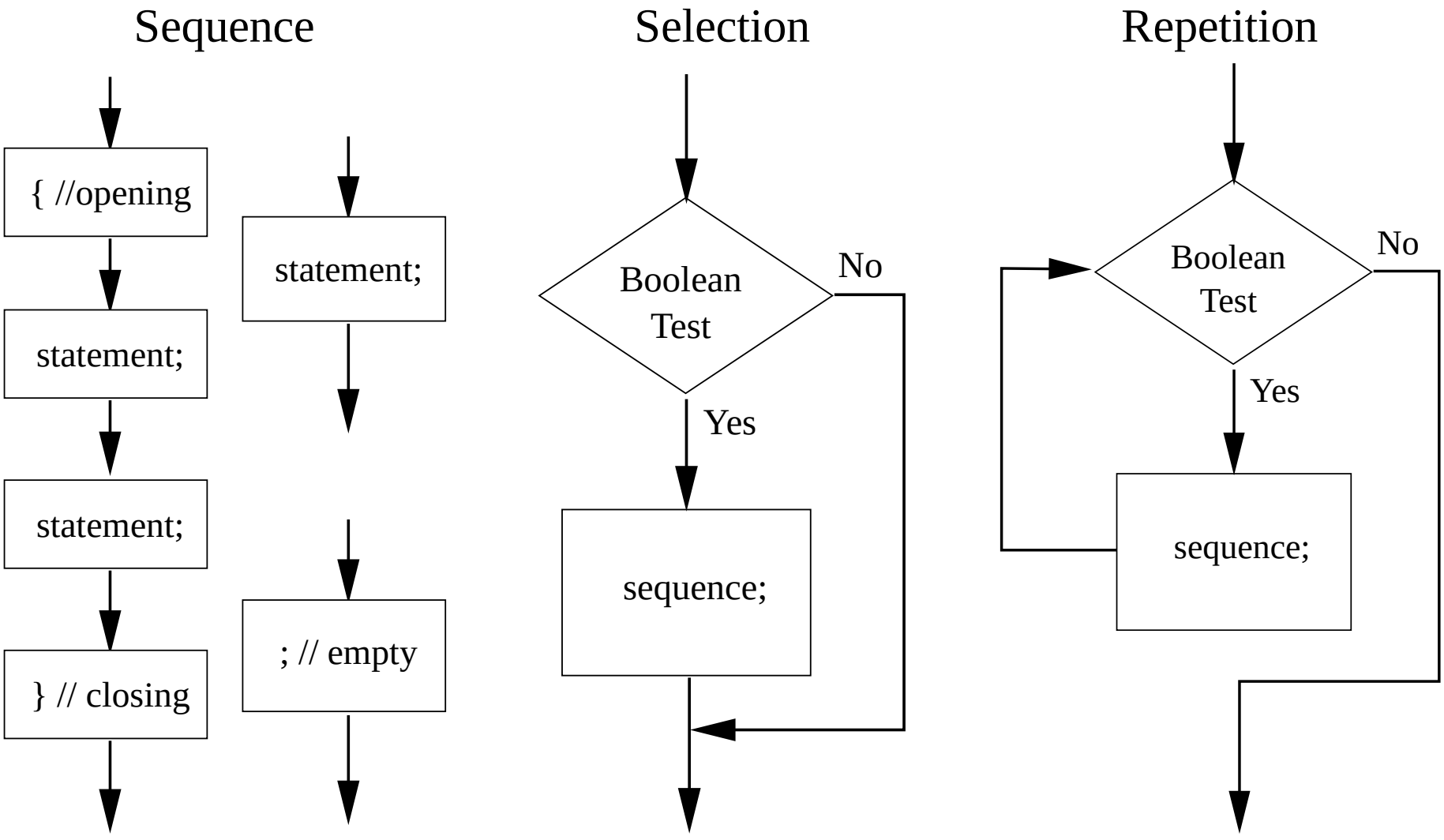
C++ Control Structures

Sequence Is a simple sequence of commands to be executed.
Most of what we've seen so far are simple program sequences.

Selection Changes the path taken in the code based on a *Boolean test*.

Repetition Loops code through the same sequence of statements until a *Boolean test* fails.

Flowchart Notation



Selection Statements

if-then Statements

```
1
2 if( expression ) {
3     // if expression is true,
4     // then execute
5     // these statements
6
7     statements;
8 }
9
10 // then IS NOT a keyword!
```

Example

```
1 int eight(8);
2
3 if( eight > 7 ) {
4     cout << "8 is greater than 7." ;
5     cout << endl;
6 }
7 cout << "past if-then statement" << endl;
```

if-then-else Statements

```
1
2 if( expression ) {
3     // if expression is true,
4     // then execute
5     // these statements
6
7     statements;
8
9 } else {
10    // otherwise, excecute
11    // this block of statements
12
13    statements;
14 }
15
16 // else IS a keyword!
```

Example

```
1 int neg_eight(-8);
2
3 if( neg_eight > 7 ) {
4     cout << "-8 is greater than 7." ;
5     cout << endl;
6 } else {
7     cout << "-8 is NOT greater than 7." ;
8     cout << endl;
9 }
10 cout << "past if-then-else statement" << endl;
```

Practice!

Lecture Question 5

```
1  int x(9), y(7), z(2), k(0);
2  double m(1.1), j(0);
3
4  if (x > y) {
5      if (y > z && y > k) {
6          m--;
7      } else {
8          k++;
9      }
10 } else {
11     j++;
12 }
13
14 cout << "m=" << m << endl;
15 cout << "k=" << k << endl;
16 cout << "j=" << j << endl;
```

Practice!

Lecture Question 5

```
1  int x(9), y(7), z(2), k(0);
2  double m(1.1), j(0);
3
4  if (x > y) {
5      if (y > z && y > k) {
6          m--;
7      } else {
8          k++;
9      }
10 } else {
11     j++;
12 }
13
14 cout << "m=" << m << endl;
15 cout << "k=" << k << endl;
16 cout << "j=" << j << endl;
```

```
m=0.1
k=0
j=0
```

RUN EDIT ifthenelse_practice.cxx

Terse if Statements

If there is *only* one statement in the "then" (or "else") clause, the {} may be omitted.

```
1 int x( 3 ), y(10);
2 if( x > 0 ) y = ++x;
```

```
1 int x( 3 ), y(10);
2 if( x > 0 ) y = ++x; else x = y--;
```

Upside	Less characters to type, "prettier" code.
Downside	Not suitable for complex logic. Easy to screw up during debugging and maintenence.

Recommendation: If you omit the curly braces, keep all the logic on one line!

Terse if Statements

A Common and Unfortunate Terse `if-else` Pattern

```
11 if( x > 0 )
12     y = ++x + 1;
13 else if ( y < x )
14     x = y--;
15 else if ( y > 0 )
16     y %= x*2 + 1;
```

- When will the last statement be executed?
- How does this pattern fail during debugging or maintenence?

Terse if Statements

When will the last statement be executed?

```
11 if( x > 0 )
12     y = ++x + 1;
13 else if ( y < x )
14     x = y--;
15 else if ( y > 0 )
16     y %= x*2 + 1;
```

Critical Question: with which `if` does the last `else` belong?

Terse if Statements

When will the last statement be executed?

```
11 if( x > 0 )
12     y = ++x + 1;
13 else if ( y < x )
14     x = y--;
15 else if ( y > 0 )
16     y %= x*2 + 1;
```

Critical Question: with which `if` does the last `else` belong?

When $x \leq 0$ and $y \geq x$, and $y > 0$.

```
 7 if( x > 0 ) {
 8     y = ++x + 1;
 9 } else {
10     if ( y < x ) {
11         x = y--;
12     } else {
13         if ( y > 0 ) {
14             y %= ++x*2 + 1;
15         }
16     }
17 }
```

An `else` always "belongs" to the `if` **nearest** (and behind) it.

Terse if Statements

When will x be changed?

```
7  if( x > 0 )
8      y = ++x + 1;
9  else if ( y < x )
10     x = y--;
11 else if ( y > 0 )
12     y %= x*2 + 1;
13     x += 100;
```

Terse if Statements

When will x be changed?

```
7  if( x > 0 )
8      y = ++x + 1;
9  else if ( y < x )
10     x = y--;
11 else if ( y > 0 )
12     y %= x*2 + 1;
13     x += 100;
```

Always!

Probably not what the programmer intended.

Indentation is for us humans, compilers **ignore it**.

Terse if Statements

What happens when debug statements are added?

```
7  if( x > 0 )
8      cout << "inside x > 0" << endl;
9      y = ++x + 1;
10 else if ( y < x )
11     cout << "inside y < x" << endl;
12     x = y--;
13 else if ( y > 0 )
14     cout << "inside y > 0" << endl;
15     y %= x*2 + 1;
```

Terse if Statements

What happens when debug statements are added?

```
7  if( x > 0 )
8      cout << "inside x > 0" << endl;
9      y = ++x + 1;
10 else if ( y < x )
11     cout << "inside y < x" << endl;
12     x = y--;
13 else if ( y > 0 )
14     cout << "inside y > 0" << endl;
15     y %= x*2 + 1;
```

It won't compile!

If you omit {}, only 1 statement may be used after the `if(...)` or `else`.

Terse if Statements

Can we use this `if-else` pattern?

```
11 if( x > 0 )
12     y = ++x + 1;
13 else if ( y < x )
14     x = y--;
15 else if ( y > 0 )
16     y %= x*2 + 1;
```

Terse if Statements

Can we use this `if-else` pattern?

```
11 if( x > 0 )
12     y = ++x + 1;
13 else if ( y < x )
14     x = y--;
15 else if ( y > 0 )
16     y %= x*2 + 1;
```

YES! Just write it “correctly” the first time.

```
11 if( x > 0 ) {
12     y = ++x + 1;
13 } else if ( y < x ) {
14     x = y--;
15 } else if ( y > 0 ) {
16     y %= x*2 + 1;
17 }
```

The right-hand-side is just as easily read, and is resilient against quick changes and debugging statements.

finis