

ANSI/ISO Standard
Standard Template Library
Templates
Namespaces
Strings
Vectors
Virtual Functions
Exception Handling

ABSOLUTE C++



Stream I/O
Inheritance
LAMBDA
Encapsulation
Patterns

WALTER SAVITCH



Chapter 2

Flow of Control



Learning Objectives

- Boolean Expressions
 - Building, Evaluating & Precedence Rules
- Branching Mechanisms
 - if-else
 - switch
 - Nesting if-else
- Loops
 - While, do-while, for
 - Nesting loops



Boolean Expressions

- Logical Operators
 - Logical AND (&&)
 - Logical OR (||)

Display 2.1, page 46

Display 2.1 Comparison Operators

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
=	Equal to	==	<code>x + 7 == 2*y</code>	$x + 7 = 2y$
≠	Not equal to	!=	<code>ans != 'n'</code>	$ans \neq 'n'$
<	Less than	<	<code>count < m + 3</code>	$count < m + 3$
≤	Less than or equal to	<=	<code>time <= limit</code>	$time \leq limit$
>	Greater than	>	<code>time > limit</code>	$time > limit$
≥	Greater than or equal to	>=	<code>age >= 21</code>	$age \geq 21$



Evaluating Boolean Expressions

- Data type bool
 - Returns true or false
 - true, false are predefined library consts
- Truth tables
 - Display 2.2 next slide



Evaluating Boolean Expressions (cont.)

Display 2.2,
page 48

Display 2.2 Truth Tables

AND

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1 && Exp_2</i>
true	true	true
true	false	false
false	true	false
false	false	false

OR

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1 Exp_2</i>
true	true	true
true	false	true
false	true	true
false	false	false

NOT

<i>Exp</i>	<i>!(Exp)</i>
true	false
false	true



Precedence of Operators

Display 2.3,
pages 49-50

Display 2.3 Precedence of Operators (part 1 of 2)

::	Scope resolution operator
.	Dot operator
->	Member selection
[]	Array indexing
()	Function call
++	Postfix increment operator (placed after the variable)
--	Postfix decrement operator (placed after the variable)
++	Prefix increment operator (placed before the variable)
--	Prefix decrement operator (placed before the variable)
!	Not
-	Unary minus
+	Unary plus
*	Dereference
&	Address of
new	Create (allocate memory)
delete	Destroy (deallocate)
delete[]	Destroy array (deallocate)
sizeof	Size of object
()	Type cast
*	Multiply
/	Divide
%	Remainder (modulo)
+	Addition
-	Subtraction
<<	Insertion operator (console output)
>>	Extraction operator (console input)

Highest precedence
(done first)

Lower precedence
(done later)



Precedence of Operators (cont.)

Display 2.3,
pages 49-50

Display 2.3 Precedence of Operators (part 2 of 2)

All operators in part 2 are of lower precedence than those in part 1.

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal
!=	Not equal
&&	And
	Or
=	Assignment
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign
? :	Conditional operator
throw	Throw an exception
,	Comma operator

*Lowest precedence
(done last)*



Precedence Examples

- Arithmetic before logical
 - $x + 1 > 2 \ || \ x + 1 < -3$ means:
 $(x + 1) > 2 \ || \ (x + 1) < -3$
- Short-circuit evaluation
 - $(x \geq 0) \ \&\& \ (y > 1)$
 - Be careful with increment operators!
 - $(x > 1) \ \&\& \ (y++)$
- Integers as boolean values
 - All non-zero values \rightarrow true
 - Zero value \rightarrow false



Branching Mechanisms

- if-else statements
 - Choice of two alternate statements based on condition expression
 - Example:
if (hrs > 40)
 grossPay = rate*40 + 1.5*rate*(hrs-40);
else
 grossPay = rate*hrs;



if-else Statement Syntax

- Formal syntax:
if (<boolean_expression>)
 <yes_statement>
else
 <no_statement>
- Note each alternative is only ONE statement!
- To have multiple statements execute in either branch → use compound statement



Compound/Block Statement

- Only 'get' one statement per branch
- Must use compound statement { } for multiples
 - Also called a 'block' stmt
- Each block should have block statement
 - Even if just one statement
 - Enhances readability



Compound Statement in Action

- Note indenting in this example:

```
if (myScore > yourScore)
{
    cout << "I win!\n";
    wager = wager + 100;
}
else
{
    cout << "I wish these were golf scores.\n";
    wager = 0;
}
```



Common Pitfalls

- Operator '=' vs. operator '=='
- One means 'assignment' (=)
- One means 'equality' (==)
 - VERY different in C++!
 - Example:
if (x = 12) ← Note operator used!
 Do_Something
else
 Do_Something_Else



The Optional else

- else clause is optional
 - If, in the false branch (else), you want 'nothing' to happen, leave it out
 - Example:
if (sales >= minimum)
 salary = salary + bonus;
 cout << "Salary = %" << salary;
 - Note: nothing to do for false condition, so there is no else clause!
 - Execution continues with cout statement



Nested Statements

- if-else statements contain smaller statements
 - Compound or simple statements (we've seen)
 - Can also contain any statement at all, including another if-else stmt!
 - Example:

```
if (speed > 55)
    if (speed > 80)
        cout << "You're really speeding!";
    else
        cout << "You're speeding.";
```
 - Note proper indenting!



Multiway if-else

- Not new, just different indenting
- Avoids 'excessive' indenting
- Syntax:

Display
page 60

MULTIWAY if-else STATEMENT

SYNTAX

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
    .
    .
    .
else if (Boolean_Expression_n)
    Statement_n
else
    Statement_For_All_Other_Possibilities
```



Multiway if-else Example

Display page 60

EXAMPLE

```
if ((temperature < -10) && (day == SUNDAY))
    cout << "Stay home.";
else if (temperature < -10) //and day != SUNDAY
    cout << "Stay home, but call work.";
else if (temperature <= 0) //and temperature >= -10
    cout << "Dress warm.";
else //temperature > 0
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first true Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is *true*, then the *Statement_For_All_Other_Possibilities* is executed.



The switch Statement

- A new stmt for controlling multiple branches
- Uses controlling expression which returns bool data type (true or false)
- Syntax:
 - Display page 62 next slide



switch Statement Syntax

Display, page 62

switch STATEMENT

SYNTAX

```
switch (Controlling_Expression)
{
    case Constant_1:
        Statement_Sequence_1
        break;
    case Constant_2:
        Statement_Sequence_2
        break;
        .
        .
        .
    case Constant_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement_Sequence
}
```

You need not place a break statement in each case. If you omit a break, that case continues until a break (or the end of the switch statement) is reached.



The switch Statement in Action

Display,
page 62

EXAMPLE

```
int vehicleClass;  
double toll;  
cout << "Enter vehicle class: ";  
cin >> vehicleClass;  
  
switch (vehicleClass)  
{  
    case 1:  
        cout << "Passenger car.";  
        toll = 0.50;  
        break;  
    case 2:  
        cout << "Bus.";  
        toll = 1.50;  
        break;  
    case 3:  
        cout << "Truck.";  
        toll = 2.00;  
        break;  
    default:  
        cout << "Unknown vehicle class!";  
}
```

*If you forget this break,
then passenger cars will
pay \$1.50.*



The switch: multiple case labels

- Execution 'falls thru' until break
 - switch provides a 'point of entry'
 - Example:

```
case 'A':  
case 'a':  
    cout << "Excellent: you got an 'A'!\n";  
    break;  
case 'B':  
case 'b':  
    cout << "Good: you got a 'B'!\n";  
    break;
```
 - Note multiple labels provide same 'entry'



switch Pitfalls/Tip

- Forgetting the break;
 - No compiler error
 - Execution simply 'falls thru' other cases until break;
- Biggest use: MENUS
 - Provides clearer 'big-picture' view
 - Shows menu structure effectively
 - Each branch is one menu choice



switch Menu Example

- Switch stmt 'perfect' for menus:
switch (response)
{
 case '1':
 // Execute menu option 1
 break;
 case '2':
 // Execute menu option 2
 break;
 case '3':
 // Execute menu option 3
 break;
 default:
 cout << "Please enter valid response.";
}



Conditional Operator

- Also called 'ternary operator'
 - Allows embedded conditional in expression
 - Essentially 'shorthand if-else' operator
 - Example:
if (n1 > n2)
 max = n1;
else
 max = n2;
 - Can be written:
max = (n1 > n2) ? N1 : n2;
 - '?' and ':' form this 'ternary' operator



Loops

- 3 Types of loops in C++:
 - while
 - Most flexible
 - No 'restrictions'
 - do-while
 - Least flexible
 - Always executes loop body at least once
 - for
 - Natural 'counting' loop



while Loops Syntax

Display page 66

SYNTAX FOR `while` AND `do-while` STATEMENTS

A `while` STATEMENT WITH A SINGLE STATEMENT BODY

```
while (Boolean_Expression)  
    Statement
```

A `while` STATEMENT WITH A MULTISTatement BODY

```
while (Boolean_Expression)  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
}
```



while Loop Example

- Consider:

```
count = 0;           // Initialization
while (count < 3)   // Loop Condition
{
    cout << "Hi ";  // Loop Body
    count++;        // Update expression
}
```

- Loop body executes how many times?



do-while Loop Syntax

Display page 67

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

```
do  
    Statement  
while (Boolean_Expression);
```

A do-while STATEMENT WITH A MULTISTatement BODY

```
do  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
} while (Boolean_Expression);
```

*Do not forget the
final semicolon.*



do-while Loop Example

- ```
count = 0; // Initialization
do
{
 cout << "Hi "; // Loop Body
 count++; // Update expression
} while (count < 3); // Loop Condition
```
- Loop body executes how many times?
- do-while loops always execute body at least once!



## while vs. do-while

- Very similar, but...
- One important difference:
  - Issue is 'WHEN' boolean expression is checked
  - while: checks BEFORE body is executed
  - do-while: checked AFTER body is executed
- After this difference, they're essentially identical!
- while is more common, due to it's ultimate 'flexibility'



## Comma Operator

- Evaluate list of expressions, returning value of the last expression
- Most often used in a for-loop
- Example:  
`first = (first = 2, second = first + 1);`
  - first gets assigned the value 3
  - second gets assigned the value 3
- No guarantee what order expressions will be evaluated.



## for Loop Syntax

```
for (Init_Action; Bool_Exp; Update_Action)
 Body_Statement
```

- Like if-else, Body\_Statement can be a block statement
  - Much more typical



## for Loop Example

- ```
for (count=0;count<3;count++)  
{  
    cout << "Hi "; // Loop Body  
}
```
- How many times does loop body execute?
- Initialization, loop condition and update all 'built into' the for-loop structure!
- A natural 'counting' loop



Loop Issues

- Loop's condition expression can be ANY boolean expression
- Examples:
 - ```
while (count<3 && done!=0)
{
 // Do something
}
```
  - ```
for (index=0;index<10 && entry!=-99)
{
    // Do something
}
```



Loop Pitfalls: Misplaced ;

- Watch the misplaced ; (semicolon)
- Example:

```
while (response != 0) ;←  
{  
    cout << "Enter val: ";  
    cin >> response;  
}
```
- Notice the ‘;’ after the while condition!
- Result here: INFINITE LOOP!



Loop Pitfalls: Infinite Loops

- Loop condition must evaluate to false at some iteration through loop
 - If not → infinite loop.
 - Example:

```
while (1)
{
    cout << "Hello ";
}
```
 - A perfectly legal C++ loop → always infinite!
- Infinite loops can be desirable
 - e.g.: 'Embedded Systems'



The break and continue Statements

- Flow of Control
 - Recall how loops provide 'graceful' and clear flow of control in and out
 - In RARE instances, can alter natural flow
- break;
 - Forces loop to exit immediately.
- continue;
 - Skips rest of loop body
- These statements violate natural flow
 - Only used when absolutely necessary!



Nested Loops

- Recall: ANY valid C++ statements can be inside body of loop
- This includes additional loop statements!
 - Called 'nested loops'
- Requires careful indenting:

```
for (outer=0; outer<5; outer++)  
    for (inner=7; inner>2; inner--)  
        cout << outer << inner;
```
- Notice no { } since each body is one statement
- Good style dictates we use { } anyway



Summary 1

- Boolean expressions
 - Similar to arithmetic → results in true or false
- C++ branching statements
 - if-else, switch
 - switch statement great for menus
- C++ loop statements
 - while
 - do-while
 - for



Summary 2

- do-while loops
 - Always execute their loop body at least once
- for-loop
 - A natural 'counting' loop
- Loops can be exited early
 - break statement
 - continue statement
 - Usage restricted for style purposes