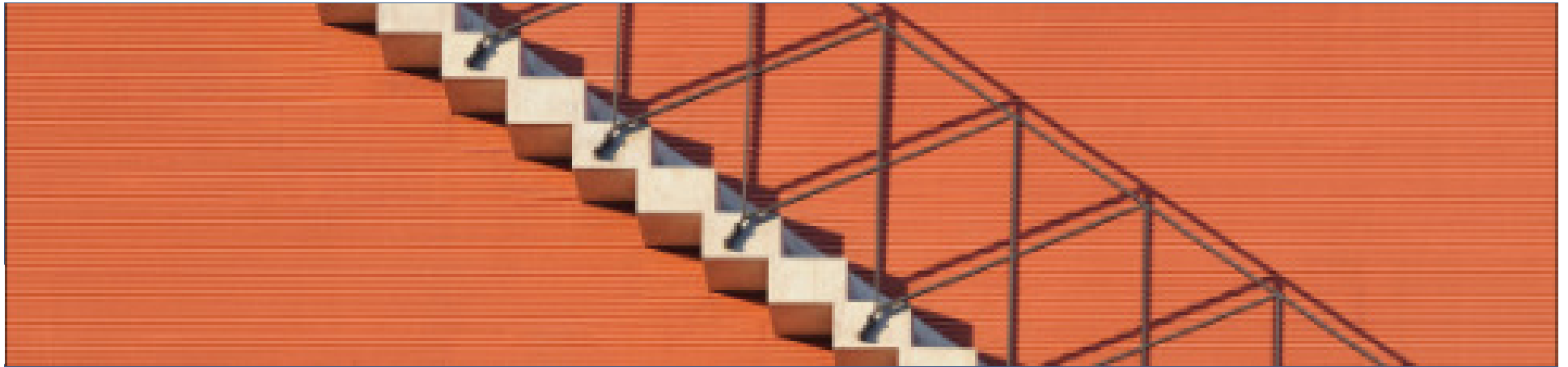




Module I

Basic elements in C++

Objectives

- In this topic, you will:
 - Become familiar with functions, special symbols, and identifiers in C++
 - Explore simple data types
 - Discover how a program evaluates arithmetic expressions
 - Learn about assignment statements

This is a C++ program



```
int main ()
```

```
{
```



```
}
```



A simple program in C++

We can declare at the point when we need them

```
#include <iostream>
using namespace std;
```

```
const double PI = 3.14159;
```



Constant in C++

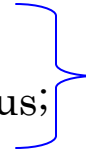
```
int main () {
    int radius;
```

```
    cout << "Enter a radius " ;
    cin >> radius;
```



Input in C++

```
    double area = PI * radius * radius;
    double circumference = 2 * PI * radius;
```



Note the declarations

```
    cout << "Area is " << area << endl;
    cout << "Circumference is " << circumference << endl;
}
```



Output in C++

The Basics of a C++ Program

- Function (or subprogram): collection of statements; when executed, accomplishes something
 - May be predefined or standard
- Syntax rules: rules that specify which statements (instructions) are legal or valid
- Semantic rules: determine the meaning of the instructions
- Programming language: a set of rules, symbols, and special words

Comments

- Comments are for the reader, not the compiler
- Two types:

- Single line: begin with //

- // This is a C++ program.

- // Welcome to C++ Programming.

- Multiple line: enclosed between /* and */

- /*

- You can include comments that can
occupy several lines.

- */

Special Symbols

- Token: the smallest individual unit of a program written in any language
- C++ tokens include special symbols, word symbols, and identifiers
- Special symbols in C++ include:

+	-	*	/
.	;	?	,
<=	!=	==	>=

Reserved Words (Keywords)

- Reserved word symbols (or keywords):
 - Cannot be redefined within program
 - Cannot be used for anything other than their intended use

Examples:

- int
- float
- double
- char
- const
- void
- return

Identifiers

- Identifier: the name of something that appears in a program
 - Consists of letters, digits, and the underscore character (`_`)
 - Must begin with a letter or underscore
- C++ is case sensitive
 - `NUMBER` is not the same as `number`
- Two predefined identifiers are `cout` and `cin`
- Unlike reserved words, predefined identifiers may be redefined, but it is not a good idea

Data Types

- Data type: set of values together with a set of operations
- C++ data types fall into three categories:
 - Simple data type
 - Structured data type
 - Pointers

Simple Data Types

- Three categories of simple data
 - Integral: integers (numbers without a decimal)
 - Can be further categorized:
 - `char, short, int, long, bool, unsigned char, unsigned short, unsigned int, unsigned long`
 - Floating-point: decimal numbers
 - Enumeration type: user-defined data type

Simple Data Types (cont'd.)

TABLE 2-2 Values and Memory Allocation for Three Simple Data Types

Data Type	Values	Storage (in bytes)
<code>int</code>	-2147483648 to 2147483647	4
<code>bool</code>	<code>true</code> and <code>false</code>	1
<code>char</code>	-128 to 127	1

- Different compilers may allow different ranges of values

int Data Type

- Examples:

–6728

0

78

+763

- Cannot use a comma within an integer
 - Commas are only used for separating items in a list

bool Data Type

- `bool` type
 - Two values: `true` and `false`
 - Manipulate logical (Boolean) expressions
- `true` and `false`
 - Logical values
- `bool`, `true`, and `false`
 - Reserved words

char Data Type

- The smallest integral data type
- Used for single characters: letters, digits, and special symbols
- Each character is enclosed in single quotes
 - 'A', 'a', '0', '*', '+', '\$', '&'
- A blank space is a character
 - Written ' ', with a space left between the single quotes

char Data Type (cont'd.)

- Different character data sets exist
- ASCII: American Standard Code for Information Interchange
 - Each of 128 values in ASCII code set represents a different character
 - Characters have a predefined ordering based on the ASCII numeric value
- Collating sequence: ordering of characters based on the character set code

Floating-Point Data Types

- C++ uses scientific notation to represent real numbers (floating-point notation)

TABLE 2-3 Examples of Decimal Numbers in Scientific and C++ Floating-Point Notations

Decimal Number	Scientific Notation	C++ Floating-Point Notation
75.924	$7.5924 * 10^1$	7.592400E1
0.18	$1.8 * 10^{-1}$	1.800000E-1
0.0000453	$4.53 * 10^{-5}$	4.530000E-5
-1.482	$-1.482 * 10^0$	-1.482000E0
7800.0	$7.8 * 10^3$	7.800000E3

Floating-Point Data Types (cont'd.)

- `float`: represents any real number
 - Range: $-3.4\text{E}+38$ to $3.4\text{E}+38$ (four bytes)
- `double`: represents any real number
 - Range: $-1.7\text{E}+308$ to $1.7\text{E}+308$ (eight bytes)
- Minimum and maximum values of data types are system dependent

Data Types and Variables

- To declare a variable, must specify the data type it will store
- **Syntax:** `dataType identifier;`
- Examples:

```
int counter;
```

```
double interestRate;
```

```
char grade;
```

Arithmetic Operators, Operator Precedence, and Expressions

- C++ arithmetic operators:
 - + addition
 - – subtraction
 - * multiplication
 - / division
 - % modulus (or remainder) operator
- +, –, *, and / can be used with integral and floating-point data types
- Use % only with integral data types

Expressions

- Integral expression: all operands are integers
 - Yields an integral result
 - Example: $2 + 3 * 5$
- Floating-point expression: all operands are floating-point
 - Yields a floating-point result
 - Example: $12.8 * 17.5 - 34.50$

Mixed Expressions

- Mixed expression:
 - Has operands of different data types
 - Contains integers and floating-point

- Examples of mixed expressions:

$2 + 3.5$

$6 / 4 + 3.9$

$5.4 * 2 - 13.6 + 18 / 2$

Type Conversion (Casting)

- Implicit type conversion: when value of one type is automatically changed to another type
- Cast operator: provides explicit type conversion
`static_cast<dataTypeName> (expression)`

Type Conversion (cont'd.)

EXAMPLE 2-9

Expression	Evaluates to
<code>static_cast<int>(7.9)</code>	7
<code>static_cast<int>(3.3)</code>	3
<code>static_cast<double>(25)</code>	25.0
<code>static_cast<double>(5 + 3)</code>	= <code>static_cast<double>(8)</code> = 8.0
<code>static_cast<double>(15) / 2</code>	= 15.0 / 2 (because <code>static_cast<double>(15)</code> = 15.0) = 15.0 / 2.0 = 7.5
<code>static_cast<double>(15 / 2)</code>	= <code>static_cast<double>(7)</code> (because <code>15 / 2</code> = 7) = 7.0
<code>static_cast<int>(7.8 +</code> <code>static_cast<double>(15) / 2)</code>	= <code>static_cast<int>(7.8 + 7.5)</code> = <code>static_cast<int>(15.3)</code> = 15
<code>static_cast<int>(7.8 +</code> <code>static_cast<double>(15 / 2))</code>	= <code>static_cast<int>(7.8 + 7.0)</code> = <code>static_cast<int>(14.8)</code> = 14

Variables, Assignment Statements, and Input Statements

- Data must be loaded into main memory before it can be manipulated
- Storing data in memory is a two-step process:
 - Instruct computer to allocate memory
 - Include statements to put data into memory

Allocating Memory with Constants and Variables

- Named constant: memory location whose content can't change during execution
- Syntax to declare a named constant:

```
const dataType identifier = value;
```

- In C++, `const` is a reserved word

EXAMPLE 2-11

Consider the following C++ statements:

```
const double CONVERSION = 2.54;  
const int NO_OF_STUDENTS = 20;  
const char BLANK = ' ';
```

Allocating Memory with Constants and Variables (cont'd.)

- Variable: memory location whose content may change during execution
- Syntax to declare a named constant:

```
dataType identifier, identifier, . . . ;
```

EXAMPLE 2-12

Consider the following statements:

```
double amountDue;  
int counter;  
char ch;  
int x, y;  
string name;
```

Putting Data into Variables

- Ways to place data into a variable:
 - Use C++'s assignment statement
 - Use input (read) statements

Assignment Statement

- The assignment statement takes the form:

```
variable = expression;
```

- Expression is evaluated and its value is assigned to the variable on the left side
- A variable is said to be initialized the first time a value is placed into it
- In C++, = is called the assignment operator

Assignment Statement (cont'd.)

EXAMPLE 2-13

Suppose you have the following variable declarations:

```
int num1, num2;  
double sale;  
char first;  
string str;
```

Now consider the following assignment statements:

```
num1 = 4;  
num2 = 4 * 5 - 11;  
sale = 0.02 * 1000;  
first = 'D';  
str = "It is a sunny day.";
```

Declaring & Initializing Variables

- Not all types of variables are initialized automatically
- Variables can be initialized when declared:

```
int first=13, second=10;
```

```
char ch=' ';
```

```
double x=12.6;
```

- All variables must be initialized before they are used
 - But not necessarily during declaration

Input (Read) Statement

- `cin` is used with `>>` to gather input

```
cin >> variable >> variable ...;
```

- This is called an input (read) statement
- The stream extraction operator is `>>`
- For example, if `miles` is a double variable

```
cin >> miles;
```

- Causes computer to get a value of type `double` and places it in the variable `miles`

Input (Read) Statement (cont'd.)

- Using more than one variable in `cin` allows more than one value to be read at a time
- Example: if `feet` and `inches` are variables of type `int`, this statement:

```
cin >> feet >> inches;
```

- Inputs two integers from the keyboard
- Places them in variables `feet` and `inches` respectively

Input (Read) Statement (cont'd.)

EXAMPLE 2-17

```
// This program illustrates how input statements work.

#include <iostream>

using namespace std;

int main()
{
    int feet;
    int inches;

    cout << "Enter two integers separated by one or more spaces: ";
    cin >> feet >> inches;
    cout << endl;

    cout << "Feet = " << feet << endl;
    cout << "Inches = " << inches << endl;

    return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

Enter two integers separated by one or more spaces: 23 7

Feet = 23
Inches = 7

Increment and Decrement Operators

- Increment operator: increase variable by 1
 - Pre-increment: `++variable`
 - Post-increment: `variable++`
- Decrement operator: decrease variable by 1
 - Pre-decrement: `--variable`
 - Post-decrement: `variable--`
- What is the difference between the following?

```
x = 5;  
y = ++x;
```

```
x = 5;  
y = x++;
```

Output

- The syntax of `cout` and `<<` is:

```
cout << expression or manipulator << expression or manipulator...;
```

- Called an output statement
- The stream insertion operator is `<<`
- Expression evaluated and its value is printed at the current cursor position on the screen

Output (cont'd.)

- A manipulator is used to format the output
 - Example: `endl` causes insertion point to move to beginning of next line

EXAMPLE 2-21

Consider the following statements. The output is shown to the right of each statement.

Statement	Output
1 <code>cout << 29 / 4 << endl;</code>	7
2 <code>cout << "Hello there." << endl;</code>	Hello there.
3 <code>cout << 12 << endl;</code>	12
4 <code>cout << "4 + 7" << endl;</code>	4 + 7
5 <code>cout << 4 + 7 << endl;</code>	11
6 <code>cout << 'A' << endl;</code>	A
7 <code>cout << "4 + 7 = " << 4 + 7 << endl;</code>	4 + 7 = 11
8 <code>cout << 2 + 3 * 5 << endl;</code>	17
9 <code>cout << "Hello \nthere." << endl;</code>	Hello there.

Output (cont'd.)

- The new line character is '\n'
 - May appear anywhere in the string

```
cout << "Hello there.";  
cout << "My name is James.";
```

Output:

```
Hello there.My name is James.
```

```
cout << "Hello there.\n";  
cout << "My name is James.";
```

Output :

```
Hello there.
```

```
My name is James.
```

Output (cont'd.)

TABLE 2-4 Commonly Used Escape Sequences

	Escape Sequence	Description
<code>\n</code>	Newline	Cursor moves to the beginning of the next line
<code>\t</code>	Tab	Cursor moves to the next tab stop
<code>\b</code>	Backspace	Cursor moves one space to the left
<code>\r</code>	Return	Cursor moves to the beginning of the current line (not the next line)
<code>\\</code>	Backslash	Backslash is printed
<code>\'</code>	Single quotation	Single quotation mark is printed
<code>\"</code>	Double quotation	Double quotation mark is printed

Preprocessor Directives

- C++ has a small number of operations
- Many functions and symbols needed to run a C++ program are provided as collection of libraries
- Every library has a name and is referred to by a header file
- Preprocessor directives are commands supplied to the preprocessor program
- All preprocessor commands begin with #
- No semicolon at the end of these commands

Preprocessor Directives (cont'd.)

- Syntax to include a header file:

```
#include <headerFileName>
```

- For example:

```
#include <iostream>
```

- Causes the preprocessor to include the header file `iostream` in the program

- Preprocessor commands are processed before the program goes through the compiler

namespace and Using cin and cout in a Program

- `cin` and `cout` are declared in the header file `iostream`, but within `std` namespace
- To use `cin` and `cout` in a program, use the following two statements:

```
#include <iostream>  
using namespace std;
```

Creating a C++ Program

- A C++ program is a collection of functions, one of which is the function `main`
- The first line of the function `main` is called the heading of the function:
 - `int main()`
- The statements enclosed between the curly braces (`{` and `}`) form the body of the function

Creating a C++ Program (cont'd.)

- A C++ program contains two types of statements:
 - Declaration statements: declare things, such as variables
 - Executable statements: perform calculations, manipulate data, create output, accept input, etc.

Use of Semicolons, Brackets, and Commas

- All C++ statements end with a semicolon
 - Also called a statement terminator
- { and } are not C++ statements
 - Can be regarded as delimiters
- Commas separate items in a list

Summary

```
//*****  
// Given the length and width of a rectangle, this C++ program  
// computes and outputs the perimeter and area of the rectangle.  
//*****
```

Comments

```
#include <iostream>  
  
using namespace std;  
  
int main()  
{  
    double length;  
    double width;  
    double area;  
    double perimeter;  
  
    cout << "Program to compute and output the perimeter and "  
         << "area of a rectangle." << endl;  
  
    length = 6.0;  
    width = 4.0;  
    perimeter = 2 * (length + width);  
}
```

Variable declarations. A statement such as `double length;` instructs the system to allocate memory space and name it `length`.

Assignment statement. This statement instructs the system to store 6.0 in the memory space `length`.

Summary (cont'd)

```
length = 6.0;
width = 4.0;
perimeter = 2 * (length + width);
area = length * width;
cout << "Length = " << length << endl;
cout << "Width = " << width << endl;
cout << "Perimeter = " << perimeter << endl;
cout << "Area = " << area << endl;
return 0;
}
```

Assignment statement. This statement instructs the system to store 6.0 in the memory space length.

Assignment statement. This statement instructs the system to evaluate the expression `length * width` and store the result in the memory space area.

Output statements. An output statement instructs the system to display results.

FIGURE 2-1 Various parts of a C++ program

More on Input/Output

Objectives

- In this topic, you will:
 - Learn what a stream is and examine input and output streams
 - Explore how to read data from the standard input device
 - Learn how to use predefined functions in a program
 - Explore how to use the input stream functions get, clear and ignore

Objectives (cont'd.)

- Become familiar with input failure
- Learn how to write data to the standard output device
- Discover how to use manipulators in a program to format output
- Learn how to perform input and output operations with the `string` data type
- Learn how to debug logic errors
- Become familiar with file input and output

I/O Streams and Standard I/O Devices

- I/O: sequence of bytes (stream of bytes) from source to destination
 - Bytes are usually characters, unless program requires other types of information
 - Stream: sequence of characters from source to destination
 - Input stream: sequence of characters from an input device to the computer
 - Output stream: sequence of characters from the computer to an output device

I/O Streams and Standard I/O Devices (cont'd.)

- Use `iostream` header file to receive data from keyboard and send output to the screen
 - Contains definitions of two data types:
 - **`istream`**: input stream
 - **`ostream`**: output stream
 - Has two variables:
 - **`cin`**: stands for common input
 - **`cout`**: stands for common output

I/O Streams and Standard I/O Devices (cont'd.)

- Variable declaration is similar to:
 - `istream cin;`
 - `ostream cout;`
- To use `cin` and `cout`, the preprocessor directive `#include <iostream>` must be used
- Input stream variables: type `istream`
- Output stream variables: type `ostream`

`cin` and the Extraction Operator

`>>`

- The syntax of an input statement using `cin` and the extraction operator `>>` is:

```
cin >> variable >> variable...;
```

- The extraction operator `>>` is binary
 - Left-side operand is an input stream variable
 - Example: `cin`
 - Right-side operand is a variable

`cin` and the `get` Function

- The `get` function
 - Inputs next character (including whitespace)
 - Stores in memory location indicated by its argument
- The syntax of `cin` and the `get` function:

```
cin.get(varChar);
```

- `varChar`
 - Is a `char` variable
 - Is the argument (or parameter) of the function

`cin` and the `ignore` Function

- `ignore` function
 - Discards a portion of the input
- The syntax to use the function `ignore` is:

```
cin.ignore(intExp, chExp);
```

- `intExp` is an integer expression
 - `chExp` is a `char` expression
- If `intExp` is a value `m`, the statement says to ignore the next `m` characters or all characters until the character specified by `chExp`

cin and the ignore Function (cont'd.)

EXAMPLE 3-5

Consider the declaration:

```
int a, b;
```

and the input:

```
25 67 89 43 72  
12 78 34
```

Now consider the following statements:

```
cin >> a;  
cin.ignore(100, '\n');  
cin >> b;
```

The first statement, `cin >> a;`, stores 25 in `a`. The second statement, `cin.ignore(100, '\n');`, discards all of the remaining numbers in the first line. The third statement, `cin >> b;`, stores 12 (from the next line) in `b`.

Output and Formatting Output

- Syntax of `cout` when used with `<<`

```
cout << expression or manipulator << expression or manipulator...;
```

- `expression` is evaluated
- `value` is printed
- `manipulator` is used to format the output
 - Example: `endl`

setprecision Manipulator

- Syntax:

```
setprecision(n)
```

- Outputs decimal numbers with up to *n* decimal places
- Must include the header file `iomanip`:
 - `#include <iomanip>`

`fixed` Manipulator

- `fixed` outputs floating-point numbers in a fixed decimal format
 - Example: `cout << fixed;`
 - Disable by using the stream member function `unsetf`
 - Example: `cout.unsetf(ios::fixed);`
- `scientific` manipulator: outputs floating-point numbers in scientific format

showpoint Manipulator

- `showpoint` forces output to show the decimal point and trailing zeros
- Examples:
 - `cout << showpoint;`
 - `cout << fixed << showpoint;`

setw

- Outputs the value of an expression in a specified number of columns
 - `cout << setw(5) << x << endl;`
- If number of columns exceeds the number of columns required by the expression
 - Output of the expression is right-justified
 - Unused columns to the left are filled with spaces
- Must include the header file `iomanip`

Additional Output Formatting Tools

- Additional formatting tools that give you more control over your output:
 - `setfill` manipulator
 - `left` and `right` manipulators

Types of Manipulators

- Two types of manipulators:
 - With parameters
 - Without parameters
- Parameterized: require `iomanip` header
 - `setprecision`, `setw`, and `setfill`
- Nonparameterized: require `iostream` header
 - `endl`, `fixed`, `showpoint`, `left`, and `flush`

Control Structures II (Selection)

Objectives

- In this topic, you will:
 - Learn about control structures
 - Examine relational and logical operators
 - Explore how to form and evaluate logical (Boolean) expressions
 - Discover how to use the selection control structures `if`, `if . . . else`, and `switch` in a program

Logical (Boolean) Operators and Logical Expressions

- Logical (Boolean) operators: enable you to combine logical expressions

TABLE 4-2 Logical (Boolean) Operators in C++

Operator	Description
!	not
&&	and
	or

The `bool` Data Type and Logical (Boolean) Expressions

- The data type `bool` has logical (Boolean) values `true` and `false`
- `bool`, `true`, and `false` are reserved words
- The identifier `true` has the value 1
- The identifier `false` has the value 0

Selection: `if` and `if...else`

- `if` and `if...else` statements can be used to create:
 - One-way selection
 - Two-way selection
 - Multiple selections

One-Way Selection

- One-way selection syntax:

```
if (expression)  
    statement
```

- Statement is executed if the value of the expression is `true`
- Statement is bypassed if the value is `false`; program goes to the next statement
- `Expression` is called a decision maker

Two-Way Selection

- Two-way selection syntax:

```
if (expression)
    statement1
else
    statement2
```

- If expression is true, statement1 is executed; otherwise, statement2 is executed
 - statement1 and statement2 are any C++ statements

Compound (Block of) Statements

- Compound statement (block of statements):

```
{  
    statement_1  
    statement_2  
    .  
    .  
    .  
    statement_n  
}
```

- A compound statement functions like a single statement

Compound (Block of) Statements (cont'd.)

```
if (age > 18)
{
    cout << "Eligible to vote." << endl;
    cout << "No longer a minor." << endl;
}
else
{
    cout << "Not eligible to vote." << endl;
    cout << "Still a minor." << endl;
}
```

Multiple Selections: Nested `if`

- Nesting: one control statement is located within another
- An `else` is associated with the most recent `if` that has not been paired with an `else`

Multiple Selections: Nested `if` (cont'd.)

EXAMPLE 4-16

Assume that `score` is a variable of type `int`. Based on the value of `score`, the following code outputs the grade:

```
if (score >= 90)
    cout << "The grade is A." << endl;
else if (score >= 80)
    cout << "The grade is B." << endl;
else if (score >= 70)
    cout << "The grade is C." << endl;
else if (score >= 60)
    cout << "The grade is D." << endl;
else
    cout << "The grade is F." << endl;
```

Short-Circuit Evaluation

- Short-circuit evaluation: evaluation of a logical expression stops as soon as the value of the expression is known
- Example:

```
(age >= 21) || ( x == 5)      //Line 1
```

```
(grade == 'A') && (x >= 7)    //Line 2
```

Confusion Between the Equality (==) and Assignment (=) Operators

- C++ allows you to use any expression that can be evaluated to either `true` or `false` as an expression in the `if` statement:

```
if (x = 5)
    cout << "The value is five." << endl;
```

- The appearance of `=` in place of `==` resembles a *silent killer*
 - It is not a syntax error
 - It is a logical error

Conditional Operator (?:)

- Conditional operator (?:)
 - Ternary operator: takes 3 arguments
- Syntax for the conditional operator:
`expression1 ? expression2 : expression3`
- If `expression1` is true, the result of the conditional expression is `expression2`
 - Otherwise, the result is `expression3`
- Example: `max = (a >= b) ? a : b;`

switch Structures

- switch structure: alternate to if-else
- switch (integral) expression is evaluated first
- Value of the expression determines which corresponding action is taken
- Expression is sometimes called the selector

```
switch (expression)
{
    case value1:
        statements1
        break;
    case value2:
        statements2
        break;
    .
    .
    .
    case valuen:
        statementsn
        break;
    default:
        statements
}
```

Control Structures II (Repetition)

Objectives

- In this topic, you will:
 - Learn about repetition (looping) control structures
 - Explore how to construct and use counter-controlled, sentinel-controlled, flag-controlled, and EOF-controlled repetition structures
 - Examine break and continue statements
 - Discover how to form and use nested control structures

while Looping (Repetition) Structure

- Syntax of the `while` statement:

```
while (expression)  
statement
```

- `statement` can be simple or compound
- `expression` acts as a decision maker and is usually a logical expression
- `statement` is called the body of the loop
- The parentheses are part of the syntax

while Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-1

Consider the following C++ program segment: (Assume that `i` is an `int` variable.)

```
i = 0;                                //Line 1
while (i <= 20)                        //Line 2
{
    cout << i << " ";                //Line 3
    i = i + 5;                        //Line 4
}

cout << endl;
```

Sample Run:

```
0 5 10 15 20
```

Case 1: Counter-Controlled while Loops

- When you know exactly how many times the statements need to be executed
 - Use a counter-controlled while loop

```
counter = 0;           //initialize the loop control variable

while (counter < N)    //test the loop control variable
{
    .
    .
    .
    counter++;        //update the loop control variable
    .
    .
    .
}
```

Case 2: Sentinel-Controlled while Loops

- Sentinel variable is tested in the condition
- Loop ends when sentinel is encountered

```
cin >> variable;           //initialize the loop control variable

while (variable != sentinel) //test the loop control variable
{
    .
    .
    .
    cin >> variable;       //update the loop control variable
    .
    .
    .
}
```

Case 3: Flag-Controlled `while` Loops

- Flag-controlled `while` loop: uses a `bool` variable to control the loop

```
found = false;           //initialize the loop control variable

while (!found)           //test the loop control variable
{
    .
    .
    .
    if (expression)
        found = true; //update the loop control variable
    .
    .
    .
}
```

More on Expressions in `while` Statements

- The expression in a `while` statement can be complex

– Example:

```
while ((noOfGuesses < 5) && (!isGuessed))  
{  
    . . .  
}
```

for Looping (Repetition) Structure

- `for` loop: called a counted or indexed `for` loop
- Syntax of the `for` statement:

```
for (initial statement; loop condition; update statement)  
    statement
```

- The `initial statement`, `loop condition`, and `update statement` are called `for` loop control statements

for Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-9

The following **for** loop prints the first 10 nonnegative integers:

```
for (i = 0; i < 10; i++)  
    cout << i << " ";  
cout << endl;
```

The initial statement, `i = 0;`, initializes the `int` variable `i` to 0. Next, the loop condition, `i < 10`, is evaluated. Because `0 < 10` is **true**, the print statement executes and outputs 0. The update statement, `i++`, then executes, which sets the value of `i` to 1. Once again, the loop condition is evaluated, which is still **true**, and so on. When `i` becomes 10, the loop condition evaluates to **false**, the **for** loop terminates, and the statement following the **for** loop executes.

for Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-10

1. The following **for** loop outputs Hello! and a star (on separate lines) five times:

```
for (i = 1; i <= 5; i++)  
{  
    cout << "Hello!" << endl;  
    cout << "*" << endl;  
}
```

2. Consider the following **for** loop:

```
for (i = 1; i <= 5; i++)  
    cout << "Hello!" << endl;  
    cout << "*" << endl;
```

This loop outputs Hello! five times and the star only once. Note that the **for** loop controls only the first output statement because the two output statements are not made into a compound statement. Therefore, the first output statement executes five times because the **for** loop body executes five times. After the **for** loop executes, the second output statement executes only once. The indentation, which is ignored by the compiler, is nevertheless misleading.

do...while Looping (Repetition) Structure

- Syntax of a `do . . . while` loop:

```
do  
    statement  
while (expression);
```

- The `statement` executes first, and then the `expression` is evaluated
 - As long as `expression` is true, loop continues
- To avoid an infinite loop, body must contain a statement that makes the `expression` false

do...while Looping (Repetition) Structure (cont'd.)

- The statement can be simple or compound
- Loop always iterates at least once

do...while Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-18

```
i = 0;

do
{
    cout << i << " ";
    i = i + 5;
}
while (i <= 20);
```

The output of this code is:

0 5 10 15 20

After 20 is output, the statement:

```
i = i + 5;
```

changes the value of `i` to 25 and so `i <= 20` becomes **false**, which halts the loop.

break and continue Statements

- `break` and `continue` alter the flow of control
- `break` statement is used for two purposes:
 - To exit early from a loop
 - Can eliminate the use of certain (flag) variables
 - To skip the remainder of a `switch` structure
- After `break` executes, the program continues with the first statement after the structure

break and continue Statements (cont'd.)

- `continue` is used in `while`, `for`, and `do...while` structures
- When executed in a loop
 - It skips remaining statements and proceeds with the next iteration of the loop

DO NOT USE `break` or `continue` in any UOW subjects when dealing with repetition control structure

**Let us now look
at workshop 1
question**

