C++ Basics

Revisiting
Lecture One

1.Variables

variable	a location in memory, referenced by name, where a data value that can be changed is stored.

Identifier declarations:

- C++ requires that all identifiers be declared before they are used.
- declaration specifies identifier name and type.
- multiple identifiers of the same type may be declared together.

Basic declaration syntax is:

type identifier1, identifier2, . . . identifierN;

Examples:

```
int    Weight,
       Height,
       Length;
double ClassAverage, GPA;
char   MiddleInitial;
string Major;
```

1.1Variable Declaration

type variable-name;

Meaning: variable <variable-name> will be a variable of type <type>

Where type can be:

- int //integer
- double //real number
- Char //character
- String

Example:

```
int a, b, c;
double x;
int sum;
char my-character;
```

1.2 C++ program

```
//include headers; these are modules that include functions that you may
    use in your
 //program; we will almost always need to include the header that
// defines cin and cout; the header is called iostream.h
#include <iostream.h>

int main() {

//variable declaration
//read values input from user
//computation  and print output to user
return 0;
}
```

1.3 Hello World Program

When learning a new language, the first program people usually write is one that salutes the world :)

Here is the Hello world program in C++.

```
#include <iostream.h>
int main() {
  cout << "Hello world!";

   return 0;
}
```

1.4 Input Statements

```
cin >> variable-name;
```

Meaning: read the value of the variable called <variable-name> from the user

Example:

```
cin >> a;
cin >> b >> c;
cin >> x;
cin >> my-character;
```

1.5 Output statements

`cout << variable-name;`

Meaning: print the value of variable <variable-name> to the user

`cout << "any message ";`

Meaning: print the message within quotes to the user

`cout << endl;`

Meaning: print a new line

Example:

```
cout << a;
cout << b << c;
cout << "This is my character: " << my-
character << " he he he"
    << endl;
```

1.6 Comments

It is good practice to include descriptive comments in program source code.

Comments may explain the purpose of a declared identifier, or of a statement or group of statements that perform some calculation, or input or output.

There are two different syntaxes for comments in C++:

```
int quizScore;              // score on a quiz
int numQuizzes = 0;         // number of quizzes given
int totalPoints;            // sum of all quiz scores
double quizAverage;         // average of all quiz scores

/* Read in quiz scores until the user enters
   one that's negative.                          */
cin >> quizScore;
while (quizScore >= 0) {
   totalPoints = totalPoints + quizScore;
   numQuizzes  = numQuizzes + 1;
   cin >> quizScore;
}
// Calculate average quiz score:
quizAverage = double(totalPoints) / numQuizzes;
```

1.7 Arithmetic Operators in C++

C++ uses the following symbols to represent the basic arithmetic operations:

Symbol	Meaning	Example	Value
+	addition	`43 + 8`	`51`
-	subtraction	`43.0 - 8.0`	`35.0`
*	multiplication	`43 * 8`	`344`
/	division	`43.0 / 8.0`	`5.375`
%	remainder	`43 % 8`	`3`
-	unary minus	`-43`	`-43`

```
int x = 0,
    y = 7;
x++;            // same effect as x = x + 1;
y--;            // same effect as y = y - 1;
```

1.8 Compound Statement

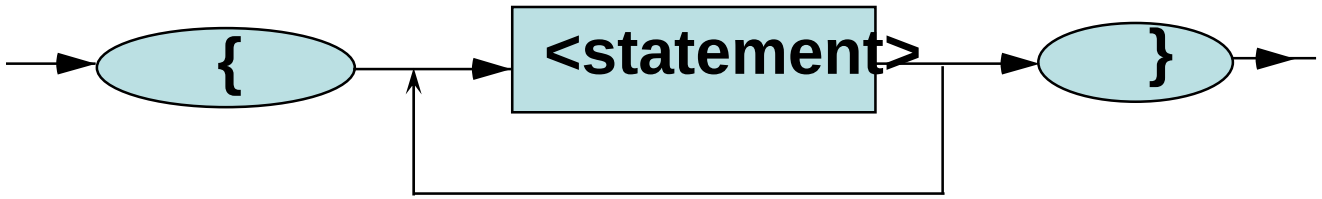
It is often necessary to group statements together (like a paragraph in a term paper).
In C++ this is accomplished by surrounding a collection of statements with "curly braces":

```
int main() {            // begin compound stmt
  int x = 42;
  cout << "x = " << x;
  return 0;
}                       // end compound stmt
```

Curly braces MUST come in matched pairs. The rule is that a closing brace `'}'` matches the closest preceding unmatched opening brace `'{'`.

Compound statements are terminated by the closing brace, NOT by a semicolon.

Syntax:



Notes

- All statements ends with semicolon
- **Lower vs. upper case matters!!**
  - Void is different than void
  - Main is different that main

2. Flow Of Control-Selection

2.1 Flow Control

- program statement may be executed sequentially, selectively or iteratively.

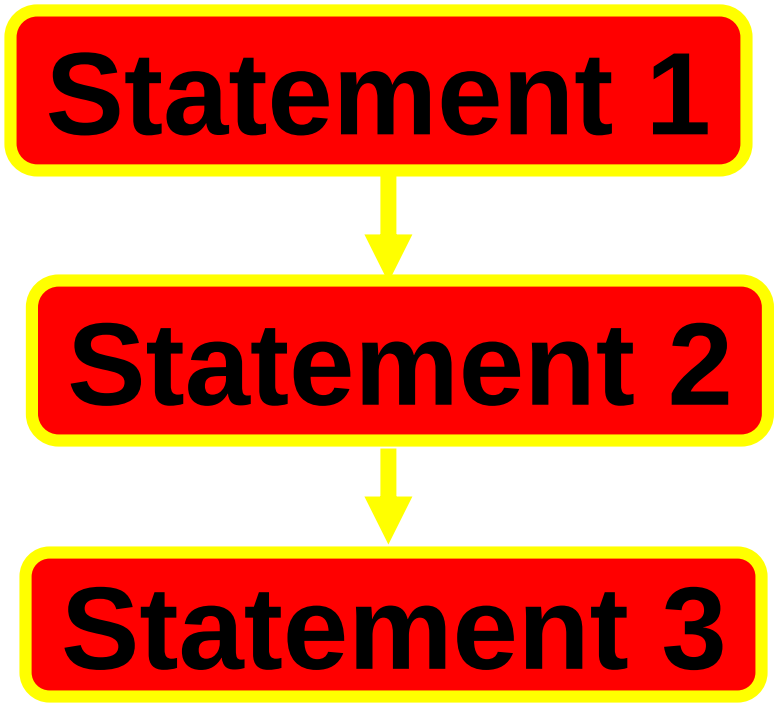
- Every program language provides constructs to support sequence, selection or iteration.

2.2 Sequence

- Sequence construct mean statement are executed sequentially.

- Every program begins with the first statement of main(). Each statement in turn executed sequentially when the final statement of main() is executed the program is done.

2.3 The Sequence Construct



2.6 Boolean conditions

..are built using

- Comparison operators

  == equal

  != not equal

  < less than

  > greater than

  <= less than or equal

  >= greater than or equal

- Boolean operators

  && and

  || or

  ! not

2.7 Syntax Of If Statement

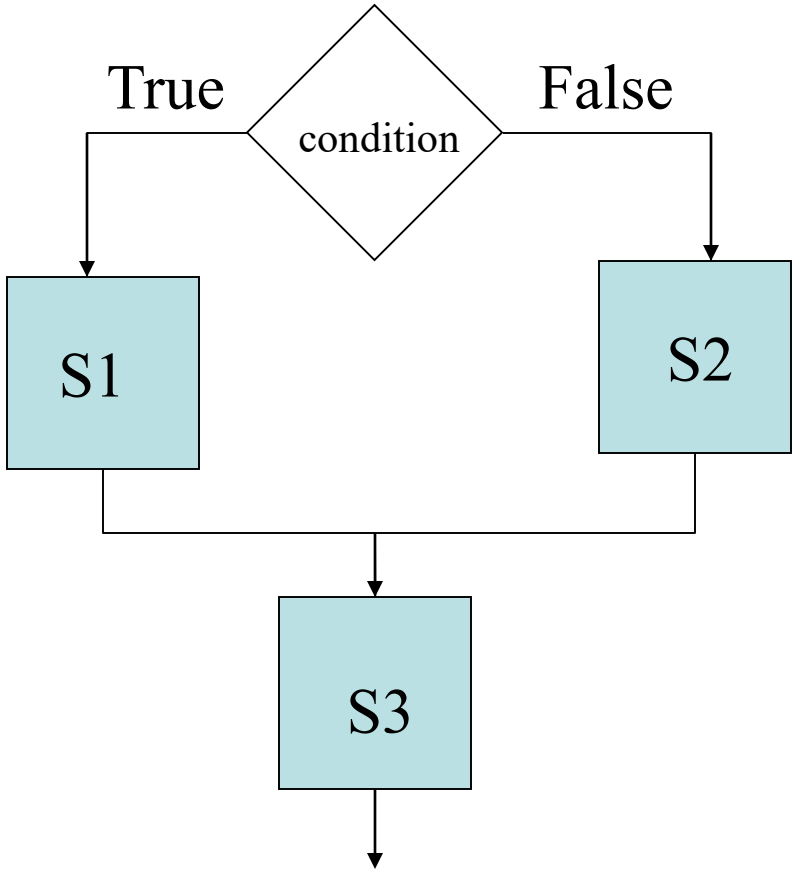
- if (condition)

  statement 1;

The statement may consist of single or compound. If the condition evaluates non zero value that is true then the statement 1 is executed otherwise if the condition evaluates zero i.e., false then the statement 1 is ignored.

If statements

```
if (condition) {
    S1;
}
else {
    S2;
}
S3;
```



2.8 Example Of If Statement

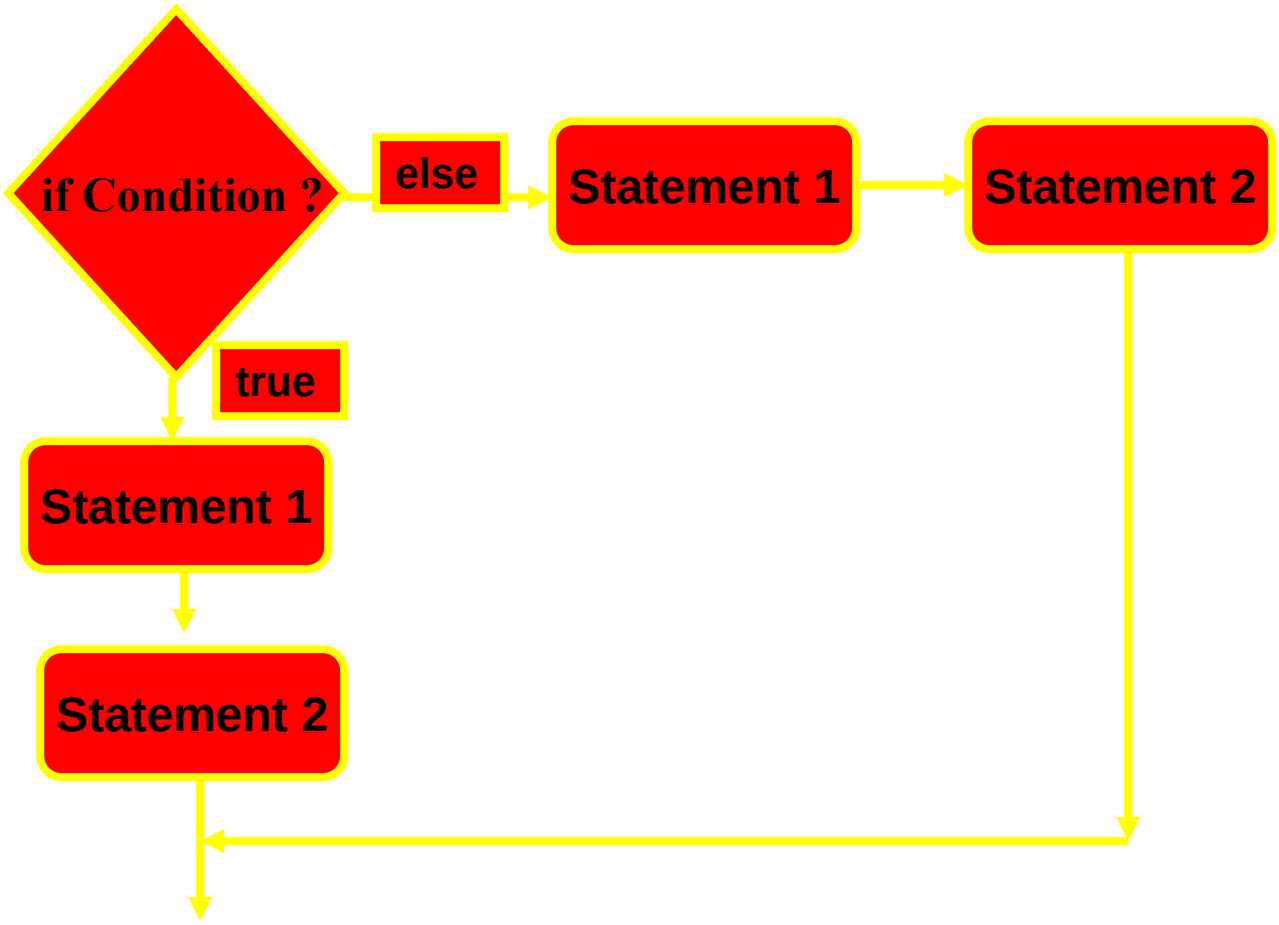
Example 1:

```
if (age>18)
cout<<“The person is eligible for vote”
```

Example 2:

```
if(ch==‘ ‘)
spaces++;
```

Flow chart of if/else statement



2.9 If - Else Format

```
if (condition)
{
        Statement 1
        Statement 2
}
else
{
    Statement 1
    Statement 2
}
```

Example of if-else

```
If (basic>8000)
{
        total_da=(40*basic)/100
        gross=total_da + basic
}
else
{
        total_da=(40*basic)/100
        gross=total_da + basic
}
```

3.0 The Switch STATEMENT

- C++provides **multiple- branch selection statement** known as **switch**

  This selection statement successively tests the value of an expression against the list of integer or character constants. When a match is found, the statements associated with that construct are executed.

3.1 The Switch STATEMENT

- The syntax is,

```
switch(expression)
{
case constant 1 :statement sequence 1;
break;
case constant  2 : statement sequence 2;
break;
case constant n-1 :statement sequence n-1;
break;
default:     statement sequence n;
break; }
```

3.2 Example of switch

```
  include<iostream.h>
void main()
{
int dow;
cout<<“Enter the number of week’s day”;
cin>>dow;
switch(dow)
{
case 1 : cout<<“\n Sunday”;
break;
```

Example Of Switch

```
case 2 : cout<<“\n Monday”;
break;
case 3 : cout<<“\n Tuesday”;
break;
case 4 : cout<<“\n Wednesday”;
break;
case 5 : cout<<“\n Thursday”;
break;
```

Example of switch

```
case 6 : cout<<“\n Friday”;
break;
case 7 : cout<<“\n Saturday”;
break;
default :cout<<“Wrong number of day”
break;
}
}
```

4. Flow Of Control-LOOPS

4.1 Why Is Repetition Needed?

- Repetition allows you to efficiently use variables
- Can input, add, and average multiple numbers using a limited number of variables
- For example, to add five numbers:
  - Declare a variable for each number, input the numbers and add the variables together
  - Create a loop that reads a number into a variable and adds it to a variable that contains the sum of the numbers

4.2 The while Loop

- The general form of the `while` statement is:

```
while (expression)
    statement
```

  `while` is a reserved word
- 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

Counter-Controlled `while` Loops

- If you know exactly how many pieces of data need to be read, the `while` loop becomes a counter-controlled loop

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

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

4.3 The for Loop

- The general form of the `for` statement is:

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

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

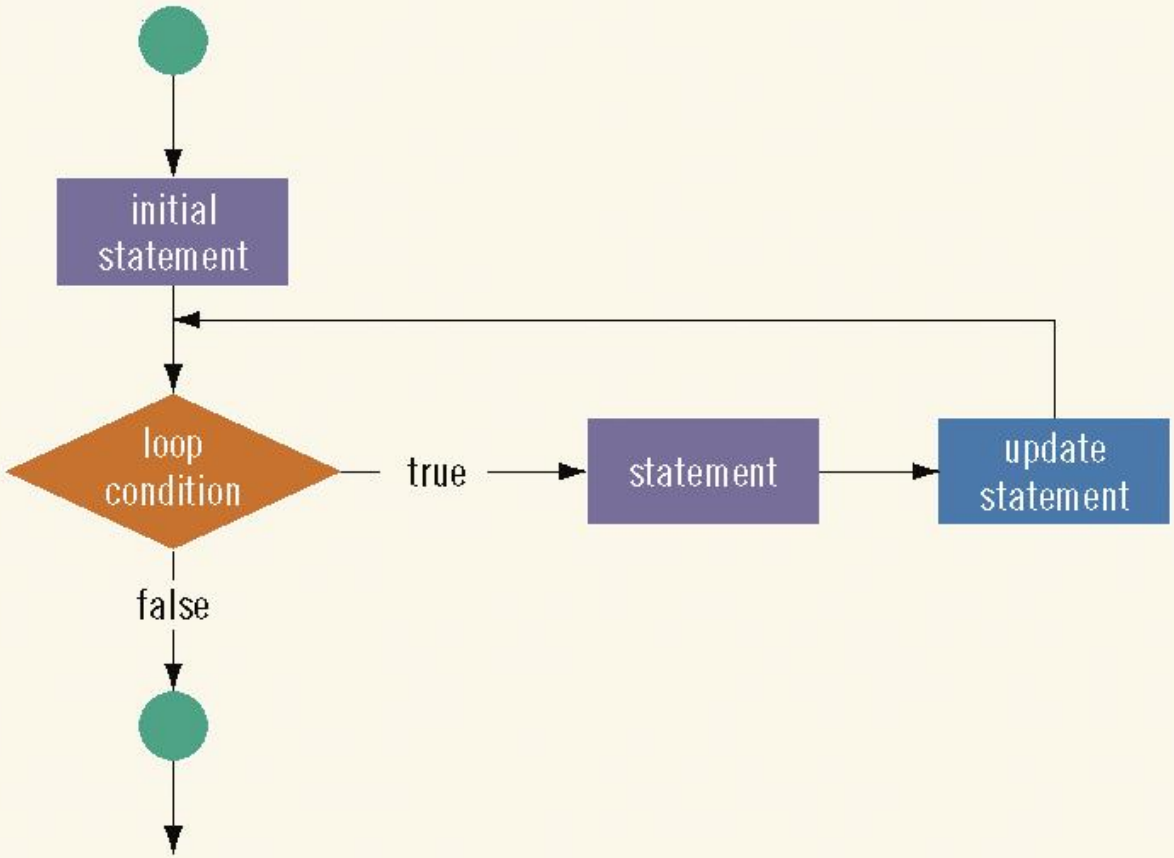


FIGURE 5-2 `for` loop

The for loop executes as follows:

1. The initial statement executes.
2. The loop condition is evaluated. If the loop condition evaluates to true
   i. Execute the for loop statement.
   ii. Execute the update statement (the third expression in the parentheses).
3. Repeat Step 2 until the loop condition evaluates to false.

The initial statement usually initializes a variable (called the for **loop control**, or for **indexed**, **variable**).

In C++, for is a reserved word.

EXAMPLE 5-7

The following `for` loop prints the first 10 non-negative integers:

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

EXAMPLE 5-8

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;
}
```

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.

EXAMPLE 5-9

The following `for` loop executes five empty statements:

```
for (i = 0; i < 5; i++);        //Line 1
    cout << "*" << endl;        //Line 2
```

4.4 The `do`…`while` Loop

- The general form of a `do`...`while` statement is:

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

- The statement executes first, and then the expression is evaluated
- If the expression evaluates to `true`, the statement executes again
- As long as the expression in a `do`...`while` statement is `true`, the statement executes

The `do`…`while` Loop (continued)

- To avoid an infinite loop, the loop body must contain a statement that makes the expression `false`
- The statement can be simple or compound
- If compound, it must be in braces
- `do`...`while` loop has an exit condition and always iterates at least once (unlike `for` and `while`)

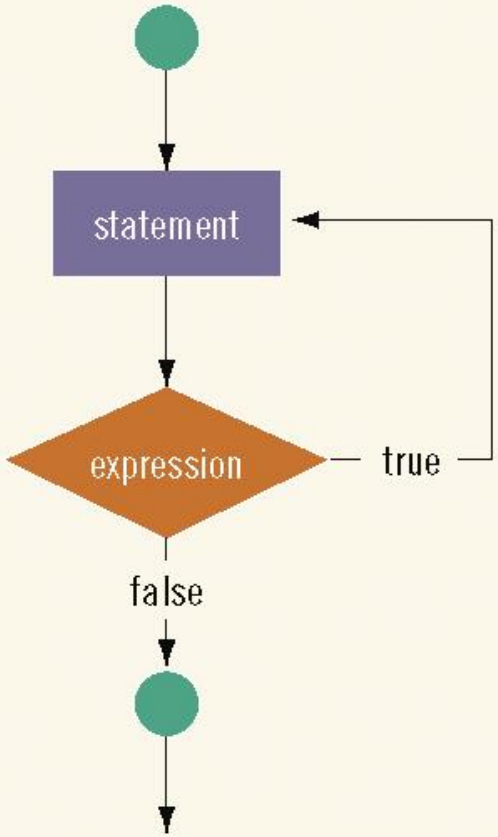


FIGURE 5-3 `do...while` loop

EXAMPLE 5-15

```
i = 0;

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

The output of this code is:

```
0 5 10 15 20
```

EXAMPLE 5-16

a.
```
i = 11;
while (i <= 10)
{
    cout << i << " ";
    i = i + 5;
}
cout << endl;
```

b.
```
i = 11;
do
{
    cout << i << " ";
    i = i + 5;
}
while (i <= 10);

cout << endl;
```

5.0 C++ keywords

– Cannot be used as identifiers or variable names.

C++ Keywords				
Keywords common to the C and C++ programming languages				
`auto`	`break`	`case`	`char`	`const`
`continue`	`default`	`do`	`double`	`else`
`enum`	`extern`	`float`	`for`	`goto`
`if`	`int`	`long`	`register`	`return`
`short`	`signed`	`sizeof`	`static`	`struct`
`switch`	`typedef`	`union`	`unsigned`	`void`
`volatile`	`while`			
C++ only keywords				
`asm`	`bool`	`catch`	`class`	`const_cast`
`delete`	`dynamic_cast`	`explicit`	`false`	`friend`
`inline`	`mutable`	`namespace`	`new`	`operator`
`private`	`protected`	`public`	`reinterpret_cast`	
`static_cast`	`template`	`this`	`throw`	`true`
`try`	`typeid`	`typename`	`using`	`virtual`
`wchar_t`				

Assignment One

- Write a program that asks the user
  - `Do you want to use this program? (y/n)`
- If the user says ‘y’ then the program terminates
- If the user says ‘n’ then the program asks
  - `Are you really sure you do not want to use this program? (y/n)`
  - If the user says ‘n’ it terminates, otherwise it prints again the message
  - `Are you really really sure you do not want to use this program? (y/n)`
  - And so on, every time adding one more “really”.