

# Introduction to C++

Ali Haider

[syedalihaider.ciit@gmail.com](mailto:syedalihaider.ciit@gmail.com)

Department of Computer Science IUB

# We will discuss

- C++ basic program Structure
- Variable Declaration
- Conditional Structures
- Iterative Structures
- Arrays
- Functions



# A 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;
}
```

After you write a C++ program you compile it; that is, you run a program called **compiler** that checks whether the program follows the C++ syntax

- if it finds errors, it lists them
- If there are no errors, it translates the C++ program into a program in machine language which you can execute

# Things to Remember

- what follows after **//** on the same line is considered **comment**
- indentation is for the convenience of the reader; compiler ignores all spaces and new line ; the delimiter for the compiler is the semicolon
- all statements ended by semicolon
- **Lower vs. upper case matters!!**
  - Void is different than void
  - Main is different that main

# 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;
}
```

# Variable Declaration

**Datatype variable-name;**

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

Where type can be:

- int //integer
- double //real number
- char //character

Example:

```
int a, b, c;
```

```
double x;
```

```
int sum;
```

```
char my-character;
```

# Take Input in C++

**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;
```

# Output in C++

**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;
```

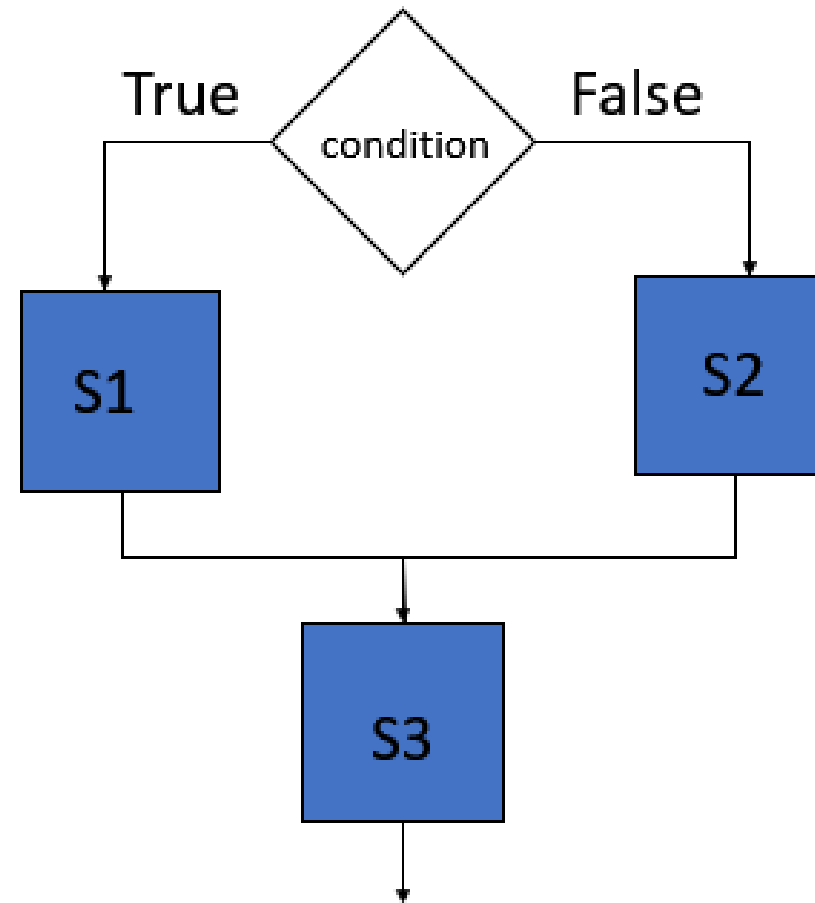


# Conditional Structures

- Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program,
- along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

# If statements

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



# Boolean Conditions

..are built using

- Comparison operators

<code>==</code>	equal
<code>!=</code>	not equal
<code>&lt;</code>	less than
<code>&gt;</code>	greater than
<code>&lt;=</code>	less than or equal
<code>&gt;=</code>	greater than or equal

- Boolean operators

<code>&amp;&amp;</code>	and
<code>  </code>	or
<code>!</code>	not

# Examples for Boolean Conditions

Assume we declared the following variables:

```
int a = 2, b=5, c=10;
```

Here are some examples of Boolean conditions we can use:

- `if (a == b) ...`
- `if (a != b) ...`
- `if (a <= b+c) ...`
- `if (a <= b) && (b <= c) ...`
- `if !( (a < b) && (b<c) ) ...`

# If else Example

```
#include <iostream.h>
void main() {
    int a,b,c;
    cin >> a >> b >> c;

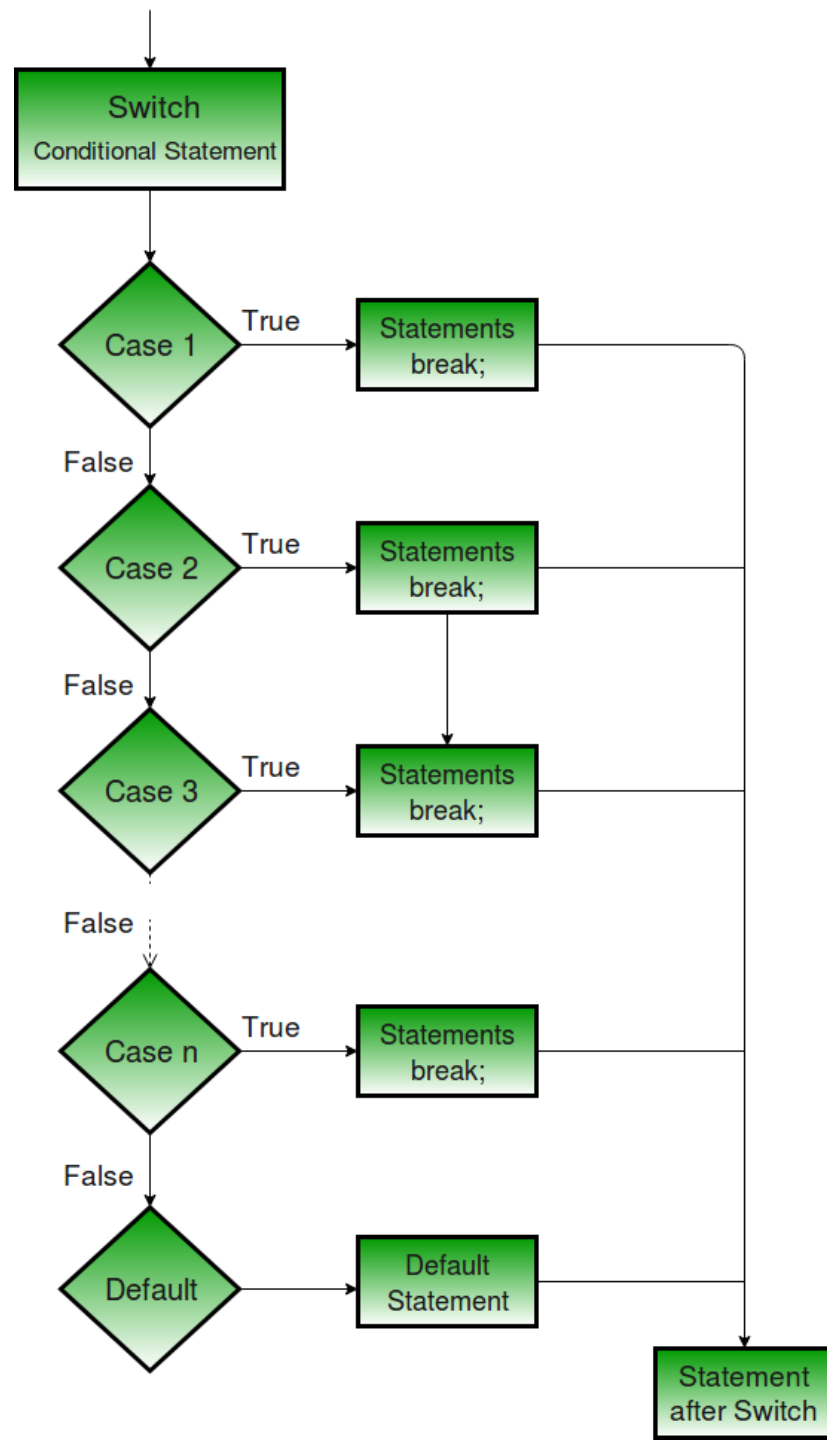
    if (a <=b) {
        cout << "min is " << a << endl;
    }
    else {
        cout << " min is " << b << endl;
    }
    cout << "happy now?" << endl;
}
```

# Switch Statement in C/C++

- Switch case statements are a substitute for long if statements that compare a variable to several integral values
- The switch statement is a multiway branch statement.
- It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- Switch is a control statement that allows a value to change control of execution.

# Syntax of Switch Statment

- switch (n)
- {
- case 1: // code to be executed if n = 1;
- break;
- case 2: // code to be executed if n = 2;
- break;
- default: // code to be executed if n doesn't match any cases
- }





# Example

```
// Following is a simple C++ program
// to demonstrate syntax of switch.
include <iostream>
using namespace std;

int main() {
int x = 2;
    switch (x)
    {
        case 1:
            cout << "Choice is 1";
            break;
        case 2:
            cout << "Choice is 2";
            break;
        case 3:
            cout << "Choice is 3";
            break;
        default:
            cout << "Choice other than 1, 2 and 3";
            break;
    }
return 0;
}
```

# The Conditional Operator

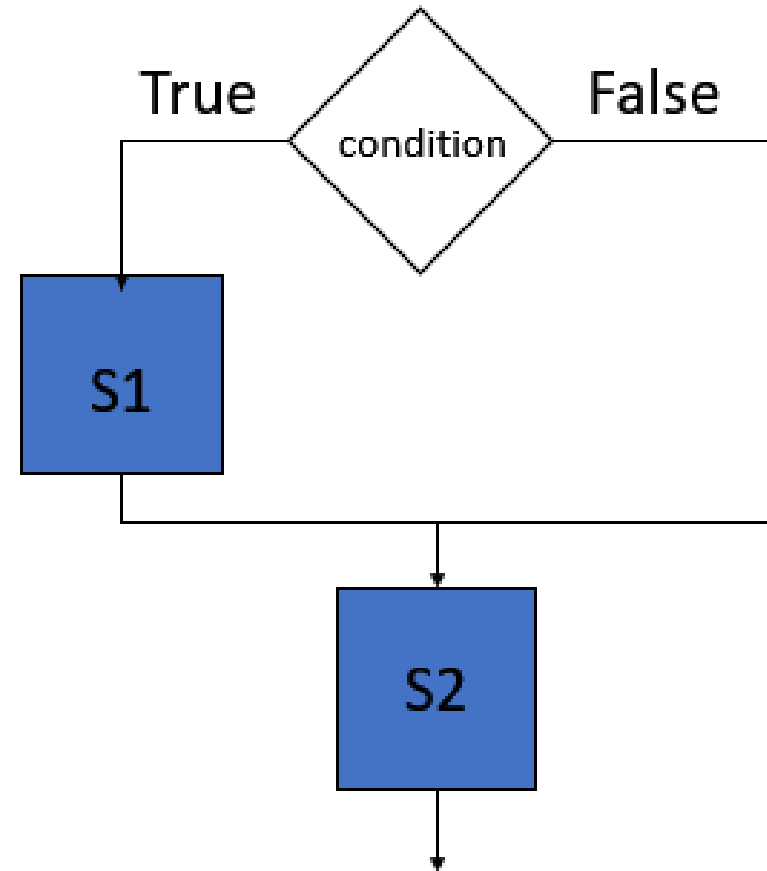
- conditional operator “? :” can be used to replace if...else statements. It has the following general form –
- `Exp1 ? Exp2 : Exp3;`
- `Exp1`, `Exp2`, and `Exp3` are expressions. Notice the use and placement of the colon.
- The value of a ‘?’ expression is determined like this: `Exp1` is evaluated.
- If it is true, then `Exp2` is evaluated and becomes the value of the entire ‘?’ expression.
- If `Exp1` is false, then `Exp3` is evaluated and its value becomes the value of the expression.

# Iterative Structures

- There may be a situation, when you need to execute a block of code several number of times.
- In general, statements are executed sequentially:
- The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –

# While statements

```
while (condition) {  
    S1;  
}  
S2;
```



# While Loop Example

```
//read 100 numbers from the user and output their sum
#include <iostream.h>
void main() {
    int i, sum, x;
    sum=0;
    i=1;
    while (i <= 100) {
        cin >> x;
        sum = sum + x;
        i = i+1;
    }
    cout << "sum is " << sum << endl;
}
```

# For Loop

- A **for loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- The syntax of a for loop in C++ is –
- `for ( init; condition; increment ) {`
- `statement(s);`
- `}`

# For Loop Example

```
//read 100 numbers from the user and output their sum
#include <iostream.h>
void main() {
int i, sum, x;
sum=0;
For(i=1;i <= 100;i++) {
    cin >> x;
    sum = sum + x;
}
cout << "sum is " << sum << endl;
}
```

# Question

- What will be the infinite loop?





# Array in C++

- C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.

# Declaring Arrays

- To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows –
- **type arrayName [ arraySize ];**
- This is called a **single-dimension** array.
- The arraySize must be an integer constant greater than zero and type can be any valid C++ data type.
- For example, to **declare a 10-element array** called balance of type double, use this statement –
- **double balance[10];**

# Initializing Arrays

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

- You can initialize C++ array elements either one by one or using a single statement as follows –
- **`double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};`**
- The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ].
- **If you omit the size of the array**, an array just big enough to hold the initialization is created. Therefore, if you write –
- **`double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};`**

# Accessing Array Elements

- An **element is accessed by indexing the array name.**
- This is done by placing the index of the element within square brackets after the name of the array.
- For example –
- `double salary = balance[9];`
- The above statement will take 10<sup>th</sup> element from the array and assign the value to salary variable.

# Two-Dimensional Arrays

- The simplest form of the multidimensional array is the two-dimensional array.
- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- To declare a two-dimensional integer array of size x,y, you would write something as follows –
- **Data-type arrayName [ x ][ y ];**
- Where type can be any valid C++ data type and arrayName will be a valid C++ identifier.
- A two-dimensional array can be think as a table, which will have x number of rows and y number of columns.

# Example

	Column 0	Column 1	Column 2	Column 3
Row 0	a[ 0 ][ 0 ]	a[ 0 ][ 1 ]	a[ 0 ][ 2 ]	a[ 0 ][ 3 ]
Row 1	a[ 1 ][ 0 ]	a[ 1 ][ 1 ]	a[ 1 ][ 2 ]	a[ 1 ][ 3 ]
Row 2	a[ 2 ][ 0 ]	a[ 2 ][ 1 ]	a[ 2 ][ 2 ]	a[ 2 ][ 3 ]

- A 2-dimensional array a, which contains three rows and four columns can be shown as below –
- `int a[3][4] = {`
- `{0, 1, 2, 3}, /* initializers for row indexed by 0 */`
- `{4, 5, 6, 7}, /* initializers for row indexed by 1 */`
- `{8, 9, 10, 11} /* initializers for row indexed by 2 */`
- `};`
- or
- `int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};`

# Accessing Two-Dimensional Array Elements

- An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array.
- For example – **int val = a[2][3];**

# Example

- `#include <iostream>`
- `using namespace std;`
- `int main () { // an array with 5 rows and 2 columns.`
- `int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};`
- `// output each array element's value`
- `for ( int i = 0; i < 5; i++ )`
- `for ( int j = 0; j < 2; j++ ) {`
- `cout << "a[" << i << "][" << j << "]: ";`
- `cout << a[i][j]<< endl;`
- `}`
- `return 0;`
- `}`



# Function

- A function is a group of statements that together perform a task. Every C++ program has at least one function, which is `main()`, and all the most trivial programs can define additional functions.
- You can divide up your code into separate functions.
- How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.
- A function declaration tells the compiler about a function's name, return type, and parameters.
- A function definition provides the actual body of the function.
- The C++ standard library provides numerous built-in functions that your program can call.
- For example, function `getch()` to get character
- A function is known with various names like a method or a sub-routine or a procedure etc.

# Defining a Function

- The general form of a C++ function definition is as follows –
- **return\_type** function\_name( **parameter list** ) {
- body of the function
- }
- **Return Type** – A function may return a value. The return\_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword void.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** – The function body contains a collection of statements that define what the function does.

# Function Overloading

- ***Function overloading*** (also *method overloading*) is a programming concept that allows programmers to define two or more functions with the same name and in the same scope.

Each function has a unique signature (or header), which is derived from:

- function/procedure name
- number of arguments
- arguments' type
- arguments' order
- arguments' name
- return type

# Example

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

void print(int i) {
    cout << " Here is int " << i << endl;
}

void print(double f) {
    cout << " Here is float " << f << endl;
}

void print(char const *c) {
    cout << " Here is char* " << c << endl;
}

int main() {
    print(10);
    print(10.10);
    print("ten");
    return 0;
}
```