



C++ Overview & Basics

(CS 1002)

Dr. Muhammad Aleem,

Department of Computer Science,
National University of Computer & Emerging Sciences,
Islamabad Campus



History

- **C** evolved from two languages (**BCPL** and **B**)
- 1980: “**C with Classes**”
- 1985: **C++ 1.0**
- 1995: **Draft standard**
- Developed by **Bjarne Stroustrup** at **Bell Labs**
- **Based** on **C**, added **Object-Oriented Programming concepts** (OOP) in C
- **Similar** program **performance** (**compared to C**)



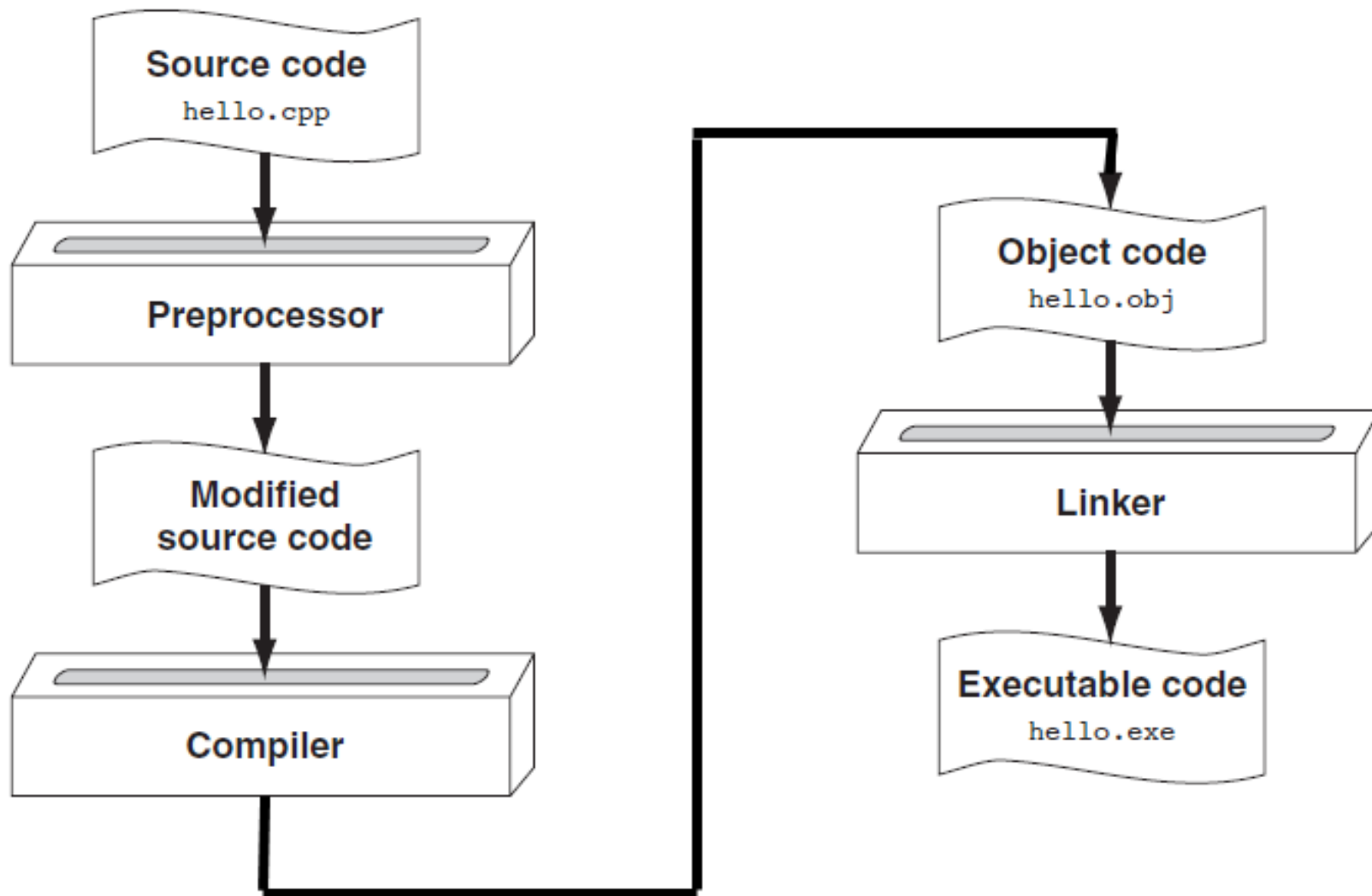
C vs. C++

- **Advantages:**

1. Faster development time (code reuse)
2. Creating / using new data types is easier
3. Easier memory management
4. Stricter syntax & type checking => **less bugs**
5. Easier to implement Data hiding
6. Object Oriented concepts



C++ Program Compilation





From a High-level Program to an Executable File

- a) Create file containing the program with a *text editor (e.g., pico, gedit, etc.)
- b) Run preprocessor to convert source file directives to source code program statements (#include lines).
- c) Run compiler to convert source program statements into machine instructions (g++).

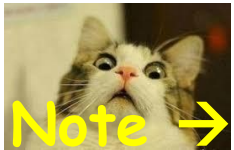


From a High-level Program to an Executable File

d) Run **linker** to **add/connect hardware-specific library code** to **machine instructions**, producing an **executable file**. (**g++**)

Steps: b) through d) are often performed by a single command or button click (such as **g++**).

Compiling a C++ program:
`g++ -o first.exe hello.cpp`



Errors occurring at **any step** will **prevent execution** of the **step** that follows.



What Is a Program Made Of?

Common elements in programming languages:

- **Key/reserved words** (predefined meaning)
- **Programmer-defined identifiers** (rules apply)
- **Operators** (e.g., + for “add, * for multiply)
- **Punctuation** (symbols that organize, e.g., comma (,), semicolon(;), parentheses, etc.)
- **Syntax** (rules of “grammar”)



First C++ Program

```
#include<iostream>

using namespace std;

int main()
{
    cout << "Hello World \n";

    return 0;
}
```

Preprocessor Directive

Standard Namespace

main function

Print message on screen

end main function



Preprocessor Directives

#include<iostream>

is a *preprocessor directive*

- The **preprocessor** runs before the actual **compiler** and **prepares** your **program** for **compilation**.
- Lines starting with # are **directives** to **preprocessor** to perform certain **tasks**, e.g., “**include**” command instructs the **preprocessor** to **add** the *iostream* library in this program



main() function

- Every C++ program start executing **from main ()**
- A **function** is a construct that **contains/encapsulates** statements in a **block**.
- Block starts from “{” and ends with “}” brace
- Every statement in the block must end with a semicolon (;)
- Examples...



Example Program 1

```
#include <iostream>
using namespace std;
int main()
{
    int num1 = 5, num2, sum;
    num2 = 12;

    sum = num1 + num2;
    cout << "The sum is " << sum;
    return 0;
}
```



Example Program 2

```
#include <iostream>
using namespace std;
int main()
{
    int num1 = 5, num2, sum;

    cout << "Enter second number: ";
    cin >> num2;

    sum = num1 + num2;
    cout << "The sum is " << sum;
    return 0;
}
```



Key Words

- Also known as **reserved words**
- Have a **special meaning** in C++
- **Can not be used for another purpose**
- **Written** using **lowercase letters**
- Examples in program (**shown in green**):

```
using namespace std;  
int main()
```



Some C++ Reserve Words

auto	break	int	long
case	char	register	return
const	continue	short	signed
default	do	sizeof	static
double	else	struct	switch
enum	extern	typedef	union
float	for	unsigned	void
goto	if	volatile	while



Programmer-Defined Identifiers

- Names made up by the programmer
- Not part of the C++ language
- Used to represent various things, such as **variables** (*memory locations*)
- Example in program (**shown in green**):
`int num1;`



Variables

- A **variable** is a **name for a cell** in computer memory (RAM) where a **value can be stored**.
- The **memory cell (variable)** holds a data value
- A **variable** must be **defined before it can be used**
- Example variable definition (declaration):

```
double num1;
```




Variables

- **Variables** are **identifiers** which represent some unknown, or variable-value.
- A **variable** is named storage (some memory address's contents)

$x = a + b;$

Speed_Limit = 90;



Declaring Variables

TYPE <Variable Name> ;

Examples:

int marks;

double Pi;

int suM;

char grade;

- **NOTE:** Variable names are case sensitive in C++ ??



Declaring Variables

- C++ is case sensitive

— Example:

area

Area

AREA

ArEa

are all seen as different variables



Variable Names

Valid Names:

- Start with a letter
- Contains letters
- Contains digits
- Contains underscores
- Do not start names with underscores: `_age`
- Don't use C++ *Reserve Words*



Variable Names

- Choose meaningful names
 - Don't use abbreviations and acronyms: `mtbf`, `TLA`, `myw`, `nbv`
- Don't use overly long names
 - **Ok:**
 - `partial_sum`
 - `element_count`
 - `staple_partition`
 - **Too long (valid but not good practice):**
 - `remaining_free_slots_in_the_symbol_table`



Which are valid variable names?

AREA

2D

Last Chance

x_yt3

Num-2

Grade***

area_under_the_curve

_Marks

#values

areaofCirCLe

%done

return

Ifstatement

Declaring Variables...

- When we declare a variable, what happens ?
 - Memory allocation
 - How much memory (*data type*)
 - Memory associated with a name (variable name)
 - The allocated space has a unique address

```
int Marks;
```



Marks

%\$^%\$%\$*^%

FE07



Variables Initialization

- Variables may be given initial values, or **initialized**, when declared. Examples:

int length = 7 ; → length
7

float diameter = 5.9 ; → diameter
5.9

char initial = 'A' ; → initial
'A'



Operators

- Used to **perform operations** on **data**
- **Many types** of **operators**:
 - Arithmetic: **+**, **-**, *****, **/**
 - Assignment: **=**
 - Input: (stream extraction) **>>**
 - Output: (stream insertion) **<<**
- Examples in program (**shown in blue**):

```
num2 = 12;  
sum = num1 + num2;  
cin >> num2;    cout << sum;
```



Punctuations

- Characters that **mark** the **end of a statement**, **separate items** in a **list**, and **separate elements** of a **statement**.

- Example in program (**shown in blue**):

```
int main ( )  
{  
    double num1 , num2=3 , num3 ;  
    num1=5 ;  
    cout << sqrt (num2) ;  
}
```



Lines vs. Statements

In a source (.cpp) file,

A line is all of the characters entered **before** a carriage return (ENTER key).

Blank lines improve the **readability** of a program.

Here are four sample lines. Line 3 is blank:

```
double num1 = 5, num2, sum;  
num2 = 12;  
  
sum = num1 + num2;
```



Lines vs. Statements

- A **statement** is an **instruction** to the **computer** to **perform an action**.
- A **statement** may **contain keywords**, **operators**, **programmer-defined identifiers**, and **punctuation**.
- A statement may fit on one line, or it may occupy multiple lines.

Here is a single statement that uses two lines:

```
double num1 = 5,  
       num2, sum;
```



Comments

- Two types of comments

1. Single line comment: *// my program*

2. Multi-line (paragraph) comment:

```
/* my  
Program */
```

- The compiler ignores all the comment related text



Input / Output Example

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

int main ( )
{
    string name; //Name of student
    cout<< "Enter you name";
    cin>>name;

    /* Now print hello , and students name */
    cout<< "Hello " << name;
    return 0;
}
```



Omitting std:: prefix

- **using directive** brings namespaces or its sub-items into current scope

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

```
int main()  
{  
    cout<<"Hello World!"<<endl;  
    cout<<"Bye!";  
    return 0;  
}
```



Namespaces

- **Namespace pollution**
 - Occurs **when building large systems** from **pieces**
 - ***Identical globally-visible names clash***
 - **How many programs** have a **“print”** function?
 - Very **difficult to fix**

using namespace std;
`cout<<“Hello World”;`

std is a **standard C++ namespace**,

You can define your own namespaces too (we see this in future)



rvalue and lvalue

Assignment Rule: On the **left side** of an **assignment** there must be a ***lvalue*** or a **variable** (address of **memory location**)

```
int i, j;
```

```
i = 7;
```

```
7 = i;
```

```
j * 4 = 7;
```



rvalue and lvalue

- Are the two occurrences of “a” in this expression the same?

a = a + 1;

- One on the **left** : **location** of the **variable** (whose name is a, or address);
- One on the **right**: **value** of the **variable** (whose name is a);

- Two attributes of variables **lvalue** and **rvalue**
 - The **lvalue** of a variable is its address
 - The **rvalue** of a variable is its value



Using iostream

- **Standard iostream objects:**
 - cout** - object providing a connection to the **monitor**
 - cin** - object providing a connection to the **keyboard**



The Insertion Operator (<<)

- To **send output** to the **screen** we use the **insertion operator** (i.e., <<) on the **object cout**

```
cout << age;
```

- Different **type of objects** can be **printed**:

```
cout << 7;    // Outputs 7
```

```
cout << 3.6;  // Outputs 3.6
```

```
cout << "String"; // Outputs String
```

```
cout << '\n';  // Outputs a newline
```



The Extraction Operator (>>)

- To **get input** from the **keyboard** we use the **extraction operator** and the object **cin**

cin >> Variable;

- Multiple uses** of the **insertion** and **extraction operator** can be **chained** together:

cout << E1 << E2 << E3 << ... ;

cin >> V1 >> V2 >> V3 >> ...;

- Example:

cout << "Total sales are \$" << sales << '\n';

cin >> Sales1 >> Sales2 >> Sales3;



The >> Operator

- Values must be separated by whitespace (*space*, *tab*, *end-of-line* [ENTER], *end-of-file*).
- Multiple values need not all be of the same type



The >> Operator

- When **ENTER** key **pressed**, **keyboard input** goes to the **input buffer** (where it is stored as characters)

 1 2 3 T O M B R O W N 7 2 . 5 e o l
1 2 3 4 **5** 6 7 8 9 0 1 2 3 4 5 6 7 8 9 10 ← **position**

>> **extracts** characters from the **input buffer** and **converts** them into the **data type** of the **variable**

```
int count;  
cout << "How may chairs in the room? ";  
cin  >> count;
```

- String “123” converted to **whole number (int) 123** and **stored into variable count**. Next, >> starts at pos 5 (space)



String input (Variables)

```
// Read first and second name

#include<iostream>

#include<string>

int main() {

    string first;

    string second;

    cout << "Enter your first and second names:";

    cin >> first >> second;

    cout << "Hello " << first << " " << second;

    return 0;

}
```




Any Questions