C++ Basics

- Structure of a C++ program
- Variables and Assignments
- Input and Output.
- Operators and Expressions
- Control Structures

Structure of a C++ program

```
#include all required directives
Using namespace std;
[classes and/or structures]
[function prototypes]
[declaration of global variables]

int main()
{

    [declaration of local variables]
    program_statements;
    [system("pause");]
    return 0;

}
```

The order of declaration of these items is not important

The include directive

- causes the content of another file to be inserted into the program.
- Include directive begins with sharp sign (#) and is not an executable code line.
- The #include<iostream> is a directive or library file that contains definitions of the routines that handle inputs from the keyboard eg. cin and outputs to the screen, eg. cout

using namespace std

- #include directive must always be followed by the namespace
- allows us to group a set of global classes, objects and/or functions under a name
- Without using namespace std; when you write for example cout <<; you'd have to put std::cout <<;

int main()

- It indicates the main function
- The { and } are used to mark the beginning and the end of the main function respectively.
- Program execution will always start with int main() regardless of where it appears in a program.

return 0;

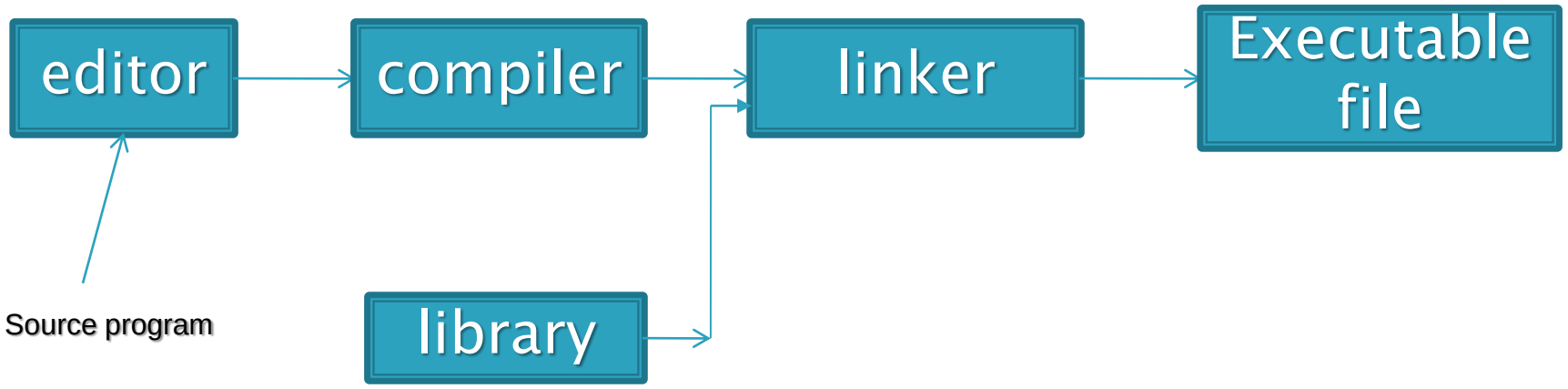
- It tells the computer that the program ends here
- The last executable statement of main function.
- It is possible to begin a function with void main().
- In this case return 0 would not be necessary since a void function does not return a value.

Structure of a C++ program

- Program statements: instructions that make up the main function. They include variables, decision making statements, looping, input and output, etc.
- Classes, function prototypes, declaration of global variables: These are optional and used only when they are needed.
- Function definitions: This is optional and is only needed when function prototypes have been included in the program.

Executable file

- The process of creating an executable file is shown below.



C++ header files

- A header file contains the description or definition of the library function

Header name	Description
iostream	Contains functions for the standard input and output functions.
fstream	Contains the function prototypes for functions that perform file input and output
cstdlib	Contains function prototypes for conversions of numbers to text, text to numeric, memory allocation and random numbers

C++ Comments

- They help to explain most of the program statements such that programmers who were not involved in writing a particular program can read and understand.
- // is used for single line comments
- /* */ is used for comments that span multiple lines
- Examples
- **// C++ is case sensitive**
- **/* this type is required for long comments that spread over a number a number of lines */**

Variables

- The name given to a variable is called an **identifier**.
- Variable names must be *meaningful* to make our programs easy to understand.
- An identifier must start with either an English alphabet (a-z or A-Z) or the underscore symbol (_), and the rest of the characters should be letters, digits, or the underscore symbol.
- Variable names can be of any reasonable length
- They should not include special characters such as @, #, $

Variables

- Variable names can be in lower, upper and/or a combination of lower and uppercase characters.
- Note that C++ is case sensitive as such Pay, PAY and paY are different variable names.
- Variable names must not be keywords or reserved words such as int, for, while, if, do, new

Variables – question

- Indicate whether the following are valid C++ identifiers. Those that are invalid indicate why they are so.

1. d.o.b
2. Interest-rate
3. Else
4. B
5. 7y
6. you

Variable declaration

- All variables used in C++ must be declared before they are first used.
- Declaring a variable tells the compiler the type of data that will be stored in the variable during program execution.
- Good programmers declare variables at the start of the main function or the user defined function in which they are used.

Syntax for declaring a variable

```
dataType  identifier[,identifier];
```

Identifier is a variable name.

dataType is one the valid C++ data types.

Predefined Data Types

- Some commonly used simple data types in C++ are as follows

Data type	description
[long] int	used for integer variables
float	used for real variables
[long] double	used for real variables
char	used for single character variables
String*	used for multiple character variables
bool	used for boolean variables

*some compilers do not recognize the string as a data type

The int Type

- A variable of type int can store only integers such as -100, 0, 1593.
- On some machines, variable of type int uses 2 bytes of memory to store its value.
- Identifiers of the type int can store values in the range of -32768 to 32767 inclusive.
- Eg int y, age;

The long int Type

- Variables that are to store integers outside the range -32768 to 32767 inclusive must be declared to be long int
- On some computers, long int uses 4 bytes of memory.
- long int can store integers as large as 2147483647 (2^{31}), and as small as -2147483647 (2^{-31})

The double Type

- Variables that are to store numbers with decimal (real) must be declared to be double.
- On some machines, double variables uses 8 bytes of memory to store a real number between 10^{-308} to 10^{308}, giving 15 digits of precision.
- double total_amount

The char Type

- Variables that are to store a SINGLE character must be declared as char.
- On some machines, char variables use 1 byte of memory. char is the “smallest” data type.
- To assign a character to a char variable, it must be enclosed in a single quotes
- e.g .char *initial=‘J’;* char str[9]="COP 2931";

Initialising variables during declaration

- Syntax

```
dataType variable1=value1;
Or
dataType variable1(value);
```

- Eg
- int age=5;
- double amount=5.7;

- Is there a need to initialise variables?
- Yes!

- When a variable is declare, the compiler automatically initialises the variable to some constant.

- Therefore, if the programmer uses a variable that should have been initialised and was not done, wrong results may be produced.

Defined constants

- # directive is used for defined constants.

- However, constant variables are defined using **const**
- Syntax

```
#define definedConstant value
```

- Egs.
- #define PI 3.14;
- #define integer int;

Scope of variables

- The scope of a variable is the block in which the variable exists and that is the only place the variable can be used.
- In C++, we have **local** and **global** variables
- Local variables are variables declared within the body of a function, that is either within the main function or within a user defined function.
- Global variables are those variables that can be used by either the main program or other functions. They are declared **before the int main()** in a program.

The First C++ Program

A program to print out or display “Hello, world!” to the screen.

```
#include <iostream>
using namespace std;
int main()
{
    cout << “Hello, world! ”<<endl;
system(“pause”);
    return 0;
}
```

A Typical C++ Program

- All C++ program begins with the following lines:

```
#include <iostream>
using namespace std;
int main()
{
```

These lines tells the C++ compiler or the Computer that the program starts here

The End of the Program

- All programs may end with the following three lines:

```
system("pause");
return 0;
}
```

- These two lines simply mean "The program ends here."

Assignment Statements

- An assignment statement is used to change the value of a variable.
- An assignment statement usually consists of a receiving variable on the left-hand side of the assignment (=) sign and an expression on the right-hand side.
- The statement ends with a semicolon.

Examples of assignment statements

```
NetPay = GrossPay – TotalDeductions;

Volume = Length*Breadth*Height;

FirstName = “Arthur”;

OwnsCar = true;

InterestRate = 0.25;

Name = FirstName+LastName;
```

Arithmetic Operators

- The following are arithmetic operators in C++:
  - + for addition,
  - - for subtraction,
  - * for multiplication,
  - / for division, and
  - % for the modulus operation, that returns the remainder of an integer division: e.g. 10 % 3= 1

Precedence Rules of Arithmetic Operators

- Expressions in brackets are evaluated first (starting with inner brackets if any)
- * / % are of the (same) highest order
- + - are the next in order of precedence
- = lowest precedence
- Arithmetic operators of the same order are evaluated from left to right in any given expression

Comparison Operators

- The simplest local expression consists of two expressions that are compared with one of the following comparison operators.

==	equal to	x ==3
!=	not equal	x != 3
<	less than	x <3
<=	less than or equal to	x <=3
>	greater than	x >3
>=	greater than or equal to	x >=3

The Logical Operator &&

- We combine two comparisons tests using the "**and**" operator, denoted by **&&** .

Syntax for a logical expression using &&

(Comparison_1) && (Comparison_2)

Example

```
if ( (score > 0) && (score < 10) )
    cout << “score is between 0 and 10
\n”;
else
    cout << “score is not between 0 and
10 \n”;
```

The Logical Operator &&

- When two comparisons are connected using an "and" operator &&, we have:
- The entire expression is true, if both of the comparisons are true.
- Otherwise, the entire expression is false.

The Logical Operator &&

- When two comparisons are connected using an "or" operator ||, we have:
- The entire expression is true, if at least one of the comparisons is true.
- Otherwise, the entire expression is false.

Short hand assignment statement

- All these three statements are equivalent

1. A+=1;
2. A++;
3. A=A+1;

Short hand assignment statement – question

- The statement A=++B is equivalent to A=B++. True/false?

INPUT AND OUTPUT CONCEPTS

Input and output concepts

- Input statement
- Output statement
- Using manipulators
- Creating manipulators

Input statement

- The standard input device is usually the keyboard.
- Handling the standard input in C++ is done by applying the extraction operator **(>>)** on the **cin** stream.
- The operator must be followed by the variable that will store the data that is going to be extracted from the stream.

Syntax of an input statement

```
cin>>variable1[>>variable2….variablen];
```

- Where variable1, variable2, etc are identifiers

Examples of input statements

- cin>>principal>>rate>>time;

- cin>>principal
  >>rate
  >>time;

cin>>principal;
cin>>rate;
cin>>time;

Output statement

- The standard output of a program is the screen, and the C++ stream object defined to access it is cout.

  cout is used in conjunction with the ***insertion operator***, which is written as <<

- Whenever we want to use constant strings of characters we must enclose them between double quotes (") so that they can be clearly distinguished from variable names.

Syntax of an output statement

```
cout<<variable1<<variable2,……<<
variablen
```

Where variable1, variable2, etc are either

identifiers or string of characters

Examples of an output statement

- cout<<variable1<<variable2;

- cout<<variable1
- <<variable2
- <<variablen;

- cout<<variable1;
- cout<<variable2;
- cout<<variablen;

Output layout controlling characters

Character	Used for
\a	alert
\b	backspace
\f	Form feed
\r	Carriage return
\n	New line
\t	tab
\v	Vertical tab
\’	Single quotes
\”	Double quotes
\\	backslash
\0	Null character

Manipulators

- Manipulators are functions specifically designed to be used in conjunction with the insertion (<<) and extraction (>>) operators on stream objects

- are operators used in C++ for formatting output.

- One of the commonest manipulators is the **endl**.

Commonly used output manipulators

manipulator	uses
setw(n)	Used to specify the minimum number of digits to be displayed for numbers
setprecision(n)	Used to set precision to n where n is an integer.
fixed	When included in a program, it allows fixed point notation to be used for floating point numbers.
scientific	Formats floating point numbers to be displayed in scientific notation.
showpos	Causes the sign (+ or –) to be displayed in front of the number

Examples of output manipulator effects

Example	output
cout.width(10); cout<<145	145
cout<<setw(10)<<setfill('0')<<145	0000000145
cout<<setprecision(3)<<145.05	145
cout.precision(2);cout<<145.09	1.4e+002

Creating your own manipulators

- C++ allows programmers to customize input/output system by creating your own manipulators.
- User defined manipulators are defined the same way as user defined functions.
- The main difference is that the data type of the result to be returned and the argument is ostream.

Syntax of an output manipulator

```
ostream &manipulatorName(ostream &stream){

//your manipulator code here
return stream;
}
```

- Example

```
ostream &myformat(ostream &stream){
stream.width(15);
stream.precision(2);
return stream;
}
```

Syntax of an input manipulator

```
istream &manipulatorName(istream &stream){

//your manipulator code here

return stream;

}
```

CONTROL STRUCTURES