Department of Networks

First Year

Problem Solving and Programming 1

C++ Comments

- Comments can be used to explain C++ code, and to make it more readable. It can also be used to direct the execution when testing your code.
- Single-line comments start with two forward slashes (//).
- Any text between // and the end of the line is ignored by C++ (will not be executed).
- This example uses a single-line comment before a line of code:

Example

```
// This is a comment

cout<<“Hello World!”;
```

- This example uses a single-line comment at the end of a line of code:

Example

```
cout<<"Hello World!";  // This is a comment
```

C++ Multi-line Comments

- Multi-line comments start with /* and ends with */.
- Any text between /* and */ will be ignored by C++.
- This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/* The code below will print the words Hello
World to the screen, and it is amazing */

cout<<"Hello World!";
```

C++ Variables

- Variables are containers for storing data values.
- In fact, variables are memory locations used for holding values
- In C++, there are different types of variables (defined with different keywords), for example:

– **`int`** - stores integers (whole numbers), without decimals, such as 123 or -123

– **`long`** –stores integers (whole numbers), without decimals, but double the size of **`int`**

– **`float`** - stores floating point numbers, with decimals, such as 19.99 or -19.99

- **`double`** **–**stores floating point numbers, with decimal but double the size of **`float`**
- **`char`** - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- **`string`** **–** stores a series of characters surrounded by double quotations
- **`bool`** **-** stores values with two states: true or false

C++ Data Types

- As explained before, a variable in C++ must be a specified data type:
- A data type specifies the size and type of variable values.
- It is important to use the correct data type for the corresponding variable; to avoid errors, to save time and memory
- But it will also make your code more maintainable and readable.

Data Types and Their Sizes in Memory

Type	Width	Typical Range
char	1 byte	-127 to 127 or 0 to 255
int	4 bytes	-2147483648 to 2147483647
long	8 bytes	-9223372036854775808 to 9223372036854775807
float	4 bytes	1.17549e-38 to 3.40282e+38
double	8 bytes	2.22507e-308 to 1.79769e+308
string	1 byte per letter	-
bool	1 byte	true or false

Declaring (Creating) Variables

- To create a variable, you must specify the type and assign it a value:

Syntax

```
type variableName = value;
```

- Where **`type`** is a C++ type (such as **`int`** or **`float`**), and **`variableName`** is the name of the variable (such as x or name).
- The equal sign is used to assign values to the variable.

- To create a variable that should store a character, look at the following example:

Example

- Create a variable called `group` of type string and assign it the value ‘A’:

```
char group = ‘A’;
cout<<group;
```

- To create a variable that should store a number, look at the following example:

Example

- Create a variable called `myNum` of type `int` and assign it the value 15:

```
int myNum = 15;
cout<<myNum;
```

- You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;
myNum = 15;
cout<< myNum;
```

- Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

- Change the value of `myN um` to 20:

```
int myNum = 15;
myNum = 20; // myNum is now 20
cout<<myNum;
```

Rules for Identifier naming in C++

- It must begin with a letter (uppercase "A-Z" or lowercase "a-z") or an underscore (_) but cannot start with a digit.
- After the first character, subsequent characters can be letters, digits (0-9), or underscores.
- C++ is case-sensitive (`myVar` and `myvar` are different).
- It cannot be a keyword (reserved word in C++), for example, `int`, `bool`, `return`, ..etc.
- It must be unique within their namespace.
- Use meaningful names that reflect the purpose of the variable (e.g, `totalCount`, `Area`, `Volume`).
- There is generally no strict limit on the length, but avoid long names as they make code harder to read and understand.

Constants

- You can add the **`const`** keyword if you don't want others (or yourself) to overwrite existing values (this will declare the variable as "constant", which means unchangeable and read-only):

Example

```
const int myNum = 15;

myNum = 20; // error
```

- The **`const`** keyword is useful when you want a variable to always store the same value, so that others (or yourself) won't mess up your code.
- An example that is often referred to as a constant, is **`PI`** (3.14159...).
- Note: You cannot declare a constant variable without assigning the value. If you do, an error will occur:
- A **`const`** field requires a value to be provided.

Displaying Variables

- The **`cout`** instruction is often used to display variable values to the console window.
- To combine both text and a variable, use the << character:

Example

```
int age = 20;

cout<<"My age = "<<age;
```

- For numeric values, the + character works as a mathematical operator (notice that we use **`int`** (integer) variables here):

Example

```
int x = 5;

int y = 6;

cout<< x + y;     // Print the value of x + y
```

- From the example above, you can expect:
- **`x`** stores the value 5
- **`y`** stores the value 6
- Then we use the **`cout`** instruction to display the value of **`x`** + y, which is 11

Declare Many Variables

- To declare more than one variable of the same type, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;

cout<< x + y + z;
```

Numbers

- Number types are divided into two groups:

  - **Integer types** stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are **`int`** and **`long`**. Which type you should use, depends on the numeric value.
  - **Floating point types** represents numbers with a fractional part, containing one or more decimals. Valid types are **`float`** and **`double`**.

Int

- The **`int`** data type can store whole numbers from -2147483648 to 2147483647.
- In general, the **`int`** data type is the most popular data type when we create variables with a numeric value.

Example

```
int myNum = 100000;

cout<< myNum;
```

Long

- The **`long`** data type can store whole numbers from -9223372036854775808 to 9223372036854775807.

- This is used when **`int`** is not large enough to store the value.

Example

```
long myNum = 15000000000;
cout<< myNum;
```

Floating Point Types

- You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515.

Float

- The float data type can store fractional numbers from 3.4e−038 to 3.4e+038.

Example

```
float myNum = 5.75;
cout<<myNum;
```

Double

- The **`double`** data type can store fractional numbers from 1.7e−308 to 1.7e+308.

Example

```
double myNum = 19.99;
```

```
cout<<myNum;
```

Booleans

- A boolean data type is declared with the **`bool`** keyword and can only take the values true or false:
- Boolean values are mostly used for conditional testing, which you will learn more about later.

Example

```
bool isCPPFun = true;

bool isFishTasty = false;

cout<<isCPPFun;    // Outputs True

cout<<isFishTasty);   // Outputs False
```

Characters

- The char data type is used to store a single character. The character must be surrounded by single quotes, like ‘$' or ‘&':

Example

```
    char mySymbol = ‘&';

    cout<< mySymbol;
```

strings

- The **`string`** data type is used to store series of characters. The string value must be surrounded by double quotes, like “Ali” or “Kyle”

  ```
  string name= “ALI”;
  cout<< name;
  ```

- Strings can contain letters, numbers and special characters, such as (“Programming”, “Study Year 2024-2025”, “myemail@yahoo.com”)

C++ User Input

Getting User Input

- We have already learned that `cout` is used to output (print) values. Now we will use `cin` to get user input.
- In the following example, the user can input, which is stored in the variable `userName`. Then we print the value of `userName`:

Example

```
cout<<‟Enter your age: ";

cin>>age;

cout<<‟Your age is " <<age;
```

- To enable the user to input a value, use `cin` in combination with the insertion operator (>>).
- The variable containing the input data follows the operator.
- The following example shows how to accept user input and store it in the `num` variable:

```
int num;

cin >> num;
```

- As with `cout`, extractions on `cin` can be chained to request more than one input in a single statement:

```
cin >> a >> b;
```

Dealing with Different types of Input

- Users can input different data types using `cin`.

Example

```
int age;

string name;

cout<<"Enter your name ";

cin>>name;

cout<<"Enter your age ";

cin>>age;

cout<<"Your name is " <<name<<" and your
age is "<<age;
```

Creating a Simple Calculator

```
int x, y;
int sum;
cout << “Enter a number: ";
cin >> x;
cout << “Enter another number: ";
cin >> y;
sum = x + y;
cout << "Sum is: " << sum;
```

C++ Operators

- Operators are used to perform operations on variables and values.
- In the example below, we use the + operator to add together two values:

Example

```
int x = 100 + 50;
```

- Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

Example

```
int sum1 = 100 + 50;        // 150 (100 + 50)
int sum2 = sum1 + 250;      // 400 (150 + 250)
int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

Arithmetic Operators

- C++ supports these arithmetic operators.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

- The addition operator adds its operands together.
- **Example**

```
int x = 40 + 60;

cout << x;
```

- Dividing by 0 will crash your program.
- The modulus operator (%) returns the remainder after an integer division .