Programming in C++

Introduction

Start Programming

Layout of a Simple C++ Program

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

int main( )
{
    Variable_Declarations

    Statement_1
    Statement_2
      ...
    Statement_Last

    return 0;
}
```

C++ is a case-sensitive language
Semicolons (;) denote the end of statements

Preprocessor Directives

```
/* Converts distances from miles to kilometers */
#include <iostream>                    /* cin and cout definitions */
#define KMS_PER_MILE 1.609             /* conversion constant */
using namespace std;    /* allows using cin and cout functions without std:: prefix   */
int main()
{
    double miles,      //distance in miles
           kms;        //equivalent distance in kilometers

    //Get the distance in miles
    cout<<"Enter the distance in miles> ";
    cin>>miles;

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    cout<<"That equals " << kms << " kilometers.\n";

    return 0;
}
```

Preprocessor Directives

- Preprocessor directives are commands that give instructions to the C++ preprocessor.
- Preprocessor is a system program that modifies a C++ program prior to its compilation.
- Preprocessor directives begins with a #
  - Example. #include or #define

#include

- `#include` is used to include other source files into your code.
- The `#include` directive gives a program access to a library.
- **Libraries** are useful functions and symbols that are predefined by the C++ language (standard libraries).
  - `# include<iostream>`
  - `# include<cmath>`

  insert their definitions to your program before compilation.

#define

- The `#define` directive instructs the preprocessor to replace each occurrence of a text by a particular constant value before compilation.
  - Example:

```
#define KMS_PER_MILES 1.60
#define PI 3.14159
```

Comments

```
/* Converts distances from miles to kilometers */

#include <iostream>                /* cin and couit definitions */
#define KMS_PER_MILE 1.609         /* conversion constant */
using namespace std;
int main()
{
    double miles,    //distance in miles
           kms;      //equivalent distance in kilometers

    //Get the distance in miles
    cout<<"Enter the distance in miles> ";
    cin>>miles;

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    cout<<"That equals " << kms << " kilometers.\n";

    return 0;
}
```

Comments

- Comments provide supplementary information making it easier for us to understand the program, but are ignored by the C++ compiler.
- Two forms of comments:
  - /* */ - anything between them with be considered a comment, even if they span multiple lines. [multi-line comment]
  - // - anything after this and before the end of the line is considered a comment. [single line comment]
- Comments are used to create **Program Documentation**
  - Information that help others read and understand the program.
- The start of the program should consist of a comment that includes programmer's name, date of the current version, and a brief description of what the program does.
- **Always Comment your Code!**

The “main” Function

```
/* Converts distances from miles to kilometers */
#include <iostrem>                    /* cin and couit definitions */
#define KMS_PER_MILE 1.609            /* conversion constant */
using namespace std;
int main()
{
    double miles,    //distance in miles
           kms;      //equivalent distance in kilometers

    //Get the distance in miles
    cout<<"Enter the distance in miles> ";
    cin>>miles;

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    cout<<"That equals " << kms << " kilometers.\n";

    return 0;
}
```

The “main” Function

- The heading `int main()` marks the beginning of the `main` function where program execution begins.
- Every C+ program has a `main` function.
- Braces ({,}) mark the beginning and end of the body of function main.
- A function body has two parts:
  - **declarations** - tell the compiler what memory cells are needed in the function
  - **executable statements** - (derived from the algorithm) are translated into machine language and later executed by the compiler.

Variables and Data Types

```
/* Converts distances from miles to kilometers */

#include <iostrem>                    /* cin and couit definitions */
#define KMS_PER_MILE 1.609            /* conversion constant */
using namespace std;
int main()
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    cout<<"Enter the distance in miles> ";
    cin>>miles;

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    cout<<"That equals " << kms << " kilometers.\n";

    return 0;
}
```

Variables Declarations

- **Variable** – The memory cell used for storing a program's data and its computational results
  - Variable's value can change.
  - Example: `miles`, `kms`
- **Variable declarations** –Statements that communicates to the compiler the names of variables in the program and the kind of information they can store.
  - Example: `double miles`
    - Tells the compiler to create space for a variable of type `double` in memory with the name `miles`.
  - C++ requires you to declare every variable used in the program.

Data Types

- **Data Types**: a set of values and a set of operations that can be performed on those values
  - **int**: Stores integer values – whole numbers
    - 65, -12345
  - **double**: Stores real numbers – numbers that use a decimal point.
    - 3.14159 or 1.23e5 (which equals 123000.0)
  - **char**: An individual character value.
    - Each char value is enclosed in single quotes. E.g. ‘A’, ‘*’.
    - Can be a letter, a digit, or a special symbol
  - Arithmetic operations (+, -, *, /) and compare can be performed in case of **int** and **double**. Compare can be performed in **char** data.

Executable Statements

```
/* Converts distances from miles to kilometers */
#include <iostrem>                    /* cin and couit definitions */
#define KMS_PER_MILE 1.609            /* conversion constant */
using namespace std;
int main()
{
    double miles,    //distance in miles
           kms;      //equivalent distance in kilometers

    //Get the distance in miles
    cout<<"Enter the distance in miles> ";
    cin>>miles;

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    cout<<"That equals " << kms << " kilometers.\n";

    return 0;
}
```

Executable Statements

- Executable Statements: C++ statements used to write or code the algorithm. C++ compiler translates the executable statements to machine code.
  - Input/Output Operations and Functions
  - Assignment Statements
  - return Statement

Input/Output Operations

- **Input operation** - data transfer from the outside world into computer memory
- **Output operation** - program results can be displayed to the program user
- **Input/output functions** - special program units that do all input/output operations
  - `cout` << used for output
  - `cin` >> used for input function

Output – The command “cout”

```
cout << ”..............”
```

where “<<” is the output operator.

i.e.

```
cout << ”My name is George”;
```

will display on the screen:

Another Example

```
cout << ”Hello” << ”My name is George”;
```

Output:

HelloMy name is George

Note:

If you want to leave a space between the words Hello and My you must add it either after Hello or before My, i.e.

```
cout << ”Hello ” << ”My name is George”;
cout << ”Hello” << ” My name is George”;
```

Is this different?

```
cout << ”Hello”;
cout << ”My name is George”;
```

Output:

```
HelloMy name is George
```

Note:
It does not change the line unless we ask it to.

Changing the line

We can change the line using the special word “endl”.

```
cout << ”Hello” << endl;
cout << ”My name is George”;
```

Output:

```
Hello
My name is George
```

Same output:

```
cout << ”Hello” << endl << ”My name is George”;
```

Structure of cout

$$\text{cout} << p_1 << p_2 << \ldots << p_n ;$$

where $p_1, p_2, \ldots p_n$ are parameters.

What is the output of the following statements?

```
cout << "I am" << "a student";
cout << " of:" << endl << endl << "C";
cout << "++" << endl << "!+-%45  @";
```

```
I ama student of:

C++
!+-%45  @
```

Special characters

- '\n' newline
- '\b' backspace
- '\0' null
- '\"' double quote
- '\t' tab
- '\r' return
- '\'' single quote
- '\\' backslash

How can we display the following output?

```
He said: "Hello"
```

Error. Why?

```
cout << "He said: "Hello"";
```

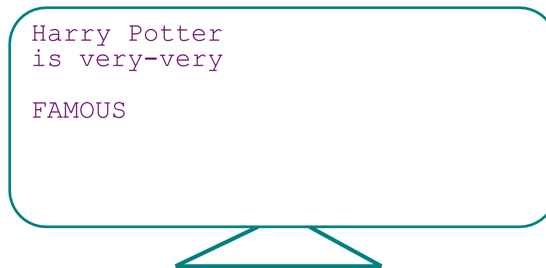
Correct:

```
cout << "He said: \"Hello\"";
```

Special Characters

What is the output of the following statement?

```
cout << "Harry Potter\nis very-very\n\nFAMOUS";
```



Numbers

What is the output of the following statements?

Statement	Output
`cout << "5";`	5
`cout << "5 + 6";`	5 + 6
`cout << 5;`	5
`cout << 5 + 6;`	11

```
cout << 5 << " + " << 6 << " = " << 5 + 6;
```

5 + 6 = 11

Input and Output

Library: iostream

Input – The command "cin"

cin >> var_1 >> var_2 >> … >> var_n

where ">>" is the input operator.

i.e.

```
cin >> x >> y;
```

will read a value for variable x,
and a value for variable y.

Input Example

```
int x,y;
cin >> x >> y;
cout << x << " + " << y << " = " << x + y;
```

Input values in RED. Program holds.

```
5 12
5 + 12 = 17
```

A Better Program

```
int x,y;
cout << "Enter two Numbers: ";
cin >> x >> y;
cout << x << " + " << y << " = " << x + y;
```

```
Enter two numbers: 5 12
5 + 12 = 17
```

When we want input from the user we should display some output explaining what the input should be.

Assignment Statements

- **Assignment statement** - Stores a value or a computational result in a variable

```
kms = KMS_PER_MILE * miles;
```

- The assignment statement above assigns a value to the variable `kms`. The value assigned is the result of the multiplication of the constant `KMS_PER_MILE` by the variable `miles`.

More on Assignments

- In C++ the symbol **=** is the assignment operator
  - Read it as ”becomes”, ”gets”, or ”takes the value of” rather than ”equals” because it is not equivalent to the equal sign of mathematics. In C++, **= =** tests equality.
- In C++ you can write assignment statements of the form:

  ```
  sum = sum + item;
  ```

  where the variable sum appears on both sides of the assignment operator.

  This is obviously not an algebraic equation, but it illustrates a common programming practice. This statement instructs the computer to add the current value of `sum` to the value of `item`; the result is then stored back into `sum`.

return Statement

```
return (0);
```

- Transfers control from your program to the operating system.
- `return (0)` returns a 0 to the Operating System and indicates that the program executed without error.
- It does not mean the program did what it was supposed to do. It only means there were no syntax errors. There still may have been logical errors.
- Once you start writing your own functions, you'll use the `return` statement to return information to the caller of the function.

Reserved Words

```
/* Converts distances from miles to kilometers */

#include <iostream>                    /* cin and couit definitions */
#define KMS_PER_MILE 1.609             /* conversion constant */
using namespace std;
int main()
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    cout<<"Enter the distance in miles> ";
    cin>>miles;

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    cout<<"That equals " << kms << " kilometers.\n";

    return 0;
}
```

Reserved words

- A word that has special meaning to C++ and can not be used for other purposes.
- These are words that C++ reserves for its own uses (declaring variables, control flow, etc.)
  - For example, you couldn't have a variable named `return`
- Always lower case
- Other examples: `double, int, if , else`, ...

Identifiers

```
/* Converts distances from miles to kilometers */

#include <iostrem>                      /* cin and couit definitions */
#define KMS_PER_MILE 1.609              /* conversion constant */
using namespace std;
int main()
{
    double miles,       //distance in miles
           kms;         //equivalent distance in kilometers

    //Get the distance in miles
    cout<<"Enter the distance in miles> ";
    cin>>miles;

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    cout<<"That equals " << kms << " kilometers.\n";

    return 0;
}
```

User Defined Identifiers

- We choose our own identifiers to name memory cells that will hold data and program results and to name operations that we define.
- **Rules for Naming Identifiers:**
  - An identifier must consist only of letters, digits, and underscores.
  - An identifier cannot begin with a digit.
  - A C++ reserved word cannot be used as an identifier.

- Valid identifiers: `letter1, inches, KM_PER_MILE`
- Invalid identifiers: `1letter, Happy*trout, return`

Few Guidelines for Naming Identifiers

- Some compliers will only see the first 31 characters of the identifier name, so avoid longer identifiers
- Uppercase and lowercase are different
  - `LETTER` != `Letter` != `letter`
  - Avoid names that only differ by case; they can lead to problems to find bugs
- Choose meaningful identifiers that are easy to understand. Example: `distance = rate * time` means a lot more than `d=r*t`
- All uppercase is usually used for constant macros (`#define`)
  - `KMS_PER_MILE` is a defined constant
  - As a variable, we would probably name it `KmsPerMile` or `Kms_Per_Mile`

Punctuation and Special Symbols

```
/* Converts distances from miles to kilometers */
#include <iostream>                 /* cin and couit definitions */
#define KMS_PER_MILE 1.609          /* conversion constant */
using namespace std;
int main()
{
     double miles,      //distance in miles
            kms;        //equivalent distance in kilometers
     //Get the distance in miles
     cout<<"Enter the distance in miles> ";
     cin>>miles;
     //Convert the distance to kilometers
     kms = KMS_PER_MILE * miles;
     //Display the distance in kilometers
     cout<<"That equals " << kms << " kilometers.\n";
     return 0;
}
```

Punctuation and Special Symbols

- **Semicolons (;)** – Mark the end of a statement
- **Curly Braces ({,}) –** Mark the beginning and end of the main function
- **Mathematical Symbols (*,=) –** Are used to assign and compute values

Arithmetic Expressions

- To solve most programming problems, you will need to write arithmetic expressions that manipulate type `int` and `double` data.
- The next slide shows all arithmetic operators. Each operator manipulates **two operands**, which may be constants, variables, or other arithmetic expressions.
- Example
  - 5 + 2
  - sum + (incr* 2)
  - (b/c) + (a + 0.5)

C++ Operators

Arithmetic Operator	Meaning	Examples
+ (*int*, *double*)	Addition	5 + 2 is 7 5.0 + 2.0 is 7.0
- (*int*, *double*)	Subtraction	5 - 2 is 3 5.0 - 2.0 is 3.0
* (*int*, *double*)	Multiplication	5 * 2 is 10 5.0 * 2.0 is 10.0
/ (*int*, *double*)	Division	5 / 2 is 2 5.0 / 2.0 is 2.5
% (*int*)	Remainder	5 % 2 is 1

Operator / & %

- **Division**: When applied to two positive integers, the division operator (/) computes the integral part of the result by dividing its first operand by its second.
  - For example 7.0 / 2.0 is 3.5 but the but 7 / 2 is only 3
  - The reason for this is that C makes the answer be of the same type as the operands.
- **Remainder**: The remainder operator (%) returns the integer remainder of the result of dividing its first operand by its second.
  - Examples: 7 % 2 = 1, 6 % 3 = 0
  - The value of m%n must always be less than the divisor n.
  - / is undefined when the divisor (second operator) is 0.

Data Type of an Expression

- The data type of each variable must be specified in its declaration, but how does C++ determine the data type of an expression?
  - Example: What is the type of expression `x+y` when both `x` and `y` are of type `int`?
- The data type of an expression depends on the type(s) of its operands.
  - If both are of type `int`, then the expression is of type `int`.
  - If either one or both is of type `double`, then the expression is of type `double`.

Mixed-Type Assignment Statement

- The expression being evaluated and the variable to which it is assigned have different data types.
  - Example what is the type of the assignment `y = 5/2` when `y` is of type `double`?
- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable to the left side of assignment operator.
- **Warning**: assignment of a type `double` expression to a type `int` variable causes the fractional part of the expression to be lost.
  - What is the type of the assignment `y = 5.0 / 2.0` when `y` is of type `int`?

Type Conversion Through Casts

- C++ allows the programmer to convert the type of an expression.
- This is done by placing the desired type in parentheses before the expression.
- This operation called a **type cast**.
  - `(double)5 / (double)2` is the `double` value 2.5, and not 2 as seen earlier.
  - `(int)3.0 / (int)2.0` is the `int` value 1
- When casting from `double` to `int`, the decimal portion is just truncated – *not* rounded.

Example

```
/*   Computes a test average  */
#include <iostream>
using namespace std;
int main()
{
        int    total_score, num_students;
        double average;
        cout<<"Enter sum of students' scores> ";
       cin>> total_score;
       cout<<"Enter number of students> ";
       cin>>num_students;
        average = (double) total_score / (double) num_students;
        cout<<"Average score is "  << average;
        return 0;
}
```

Expressions with Multiple Operators

- Operators can be split into two types: **unary** and **binary.**
- **Unary operators** take only one operand
  - **-** (negates the value it is applied to)
- **Binary operators** take two operands.
  - **+,-,*,/**
- A single expression could have multiple operators
  - -5 + 4 * 3 - 2

Rules for Evaluating Expressions

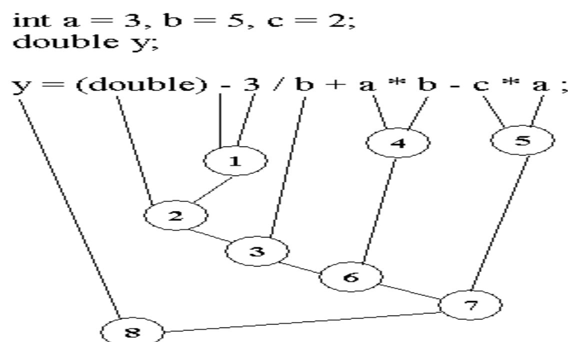
- **Rule (a): Parentheses rule** - All expressions in parentheses must be evaluated separately.
  - Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.
- **Rule (b): Operator precedence rule** – Multiple operators in the same expression are evaluated in the following order:
  - First: unary –
  - Second: *, /, %
  - Third: binary +,-
- **Rule (c): Associativity rule**
  - Unary operators in the same subexpression and at the same precedence level are evaluated right to left
  - Binary operators in the same subexpression and at the same precedence level are evaluated left to right.

Precedence and Associativity Rules

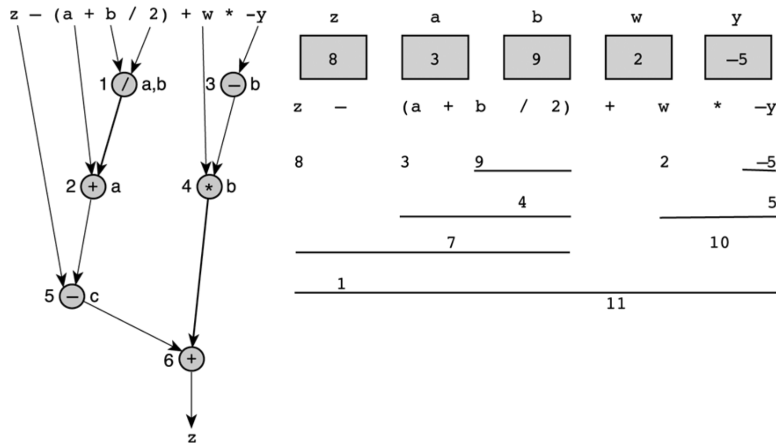
In C, mathematical expressions are evaluated according to the following precedence and associativity rules:

		Operators	Order of Evaluation of operands with same precedence (Associativity)
Higher Priority	1	(*expression*) and function calls	Left to right
	2	unary +, unary – Type cast: (type)	Right to left
	3	*, /, %	Left to right
	4	binary +, binary -	Left to right
Low Priority	5	=	Right to left

Example:



Evaluation Tree and Evaluation for
z - (a + b / 2) + w * -y
with type int variables only



Writing Mathematical Formulas in C++

- You may encounter two problems in writing a mathematical formula in C++.
- First, multiplication often can be implied in a formula by writing two letters to be multiplied next to each other. In C, you must state the * operator
  - For example, 2a should be written as 2 * a.
- Second, when dealing with division we often have:

$$\frac{a+b}{c+d}$$

  - This should be coded as (a + b) / (c + d).

Library Functions

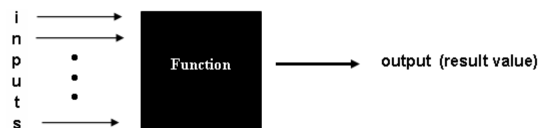
- So far, we have learnt how to use operators, +, -, *, / and % to form simple arithmetic expressions.
- However, we are not yet able to write many other mathematical expressions we are used to.
- For example, we cannot yet represent any of the following expression in C++:

$$\sqrt{x}$$

- C++ does not have operators for "square root" etc.
- Instead, C++ provides program units called functions to carry out these and other mathematical operations.

Library Functions …

- A function can be thought of as a black box that takes one or more input arguments and produces a single output value.



- For example, the following shows how to use the sqrt function that is available in the standard math library:

```
y = sqrt (x);
```

- If x is 16, the function computes the square root of 16. The result, 4, is then assigned to the variable y.
- The expression part of the assignment statement is called function call.
- Another example is: z = 5.7 + sqrt (w);
  If w = 9, z is assigned 5.7 + 3, which is 8.7.

Some Mathematical Library Functions

Function	Header File	Purpose	Arguments	Result
sin(x),cos(x), tan(x)	<cmath>	Returns the sine, cosine, or tangent of angle x.	double (in radians)	double
pow(x, y)	<cmath>	Returns x^y	double, double	double
sqrt(x)	<cmath>	$\sqrt{x}$	double (must be >= 0)	double

Example

- We can use C functions ***pow*** and ***sqrt*** to compute the roots of a quadratic equation in x of the form:

$$ax^2 + bx + c = 0$$

- If the discriminant ($b^2 – 4ac$) is greater than zero, the two roots are defined as:

$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad root_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- In C, these two roots are computed as:

```
/* compute two roots, root_1 and root_2, for disc > 0.0 */
disc = pow(b, 2) - 4 * a * c;
root_1 = (-b + sqrt(disc)) / (2 * a);
root_2 = (-b - sqrt(disc)) / (2 * a);
```

Example: Find the roots of the quadratic equation $ax^2+bx+c = 0$ (where a, b and c are coefficients).

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double a, b, c, x1, x2, discriminant;
    cout << "Enter coefficients a, b and c: ";
    cin >> a >> b >> c;
    discriminant = b*b - 4*a*c;

     x1 = (-b + sqrt(discriminant)) / (2*a);
     x2 = (-b - sqrt(discriminant)) / (2*a);
     cout << "Roots are:" << endl;
     cout << "x1 = " << x1 << endl;
     cout << "x2 = " << x2 << endl;

   return 0;
}
```

Exercise:

The **area** of a triangle with sides **A**, **B**, and **C** is calculated as

$$Area = \sqrt{S(S-A)(S-B)(S-C)}$$

Where **S=P/2** and **P** is the triangle **perimeter** computed as

$P=A+B+C$

Write a code to read the coordinates of three points that form the triangle vertices **P1(x1,y1), P2(x2,y2),** and **P3(x3,y3)** and computes and prints on the monitor the **area** of the triangle. Note the distance between two points, **P1** and **P2** for example, is computed as

$$Distance = \sqrt{(x2-x1)^2 + (y2-y1)^2}$$

Let all variables be of type double.

<u>**A sample run of the code can be as**</u>

Enter the coordinates of point 1: 2 5

Enter the coordinates of point 2: 2 8

Enter the coordinates of point 3: 6 8

The area of the triangle is 6.0