CS111

Introduction to Computing Science

Hello World

- We saw a very simple program in C++:

```
#include <iostream>
  using namespace std;
  int main(){
  cout << "Hello, World!" << endl;
  return 0;
}
```

- In this lesson we'll try to modify the same program so that it does something more interesting.

Hello World

We also learnt how any C++ program is structured.

• Begins with a #include statement which specifies the library from which you would like to use some functions.

• Then comes the void main() and within curly {} braces, you write your program statements

Hello World

Note:

- The opening and closing curly brace is mandatory.
- Also each line of your program statement should terminate with a semi-colon **;**

But when we run our nice little program, it prints to console and then closes.

Today: A program with input and output.

Input/Output

C++ Input and Output (I/O)

Usually I/O is the most important part of any program.

- Input: Data provided by the user.
- Output: Data computed by the program.

To do anything useful your program needs to be
able to accept input data and report back your results.

In C++, the standard library (iostream) provides routines for input and output.

Input/Output

In our course all the input functions described read from **standard input** and all the output functions described write to **standard output**.

- **Standard input** usually means input using the keyboard.
- **Standard output** usually means output onto the monitor.

Input/Output

Example:

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

When this statement is executed, it sends the stream of characters Hello World to the standard output.

Standard output is usually the monitor.

In other words, anything after the << is written to the screen.

Input/Output

Suppose we want to write more than Hello World

```
#include <iostream>

using namespace std;
int main()
{
cout << "Hello World. This is the second program!";
  return 0;
}
```

The output of the program is:

```
Hello World. This is the second program!
```

Input/Output

Suppose we want to write

```
Hello World.
This is the second program!
```

Input/Output

We might try

```
#include <iostream>

using namespace std;
int main()
{
cout << "Hello World.";
cout << "This is the second program!";
  return 0;
}
```

The output of the program is again:

Hello World.This is the second program!

Input/Output

There are two ways to print

Hello World.
This is the second program!

We saw one solution last week. Use endl

```
#include <iostream>

using namespace std;
int main()
{
cout << "Hello World." << endl;
cout << "This is the second program!";
  return 0;
}
```

Input/Output

There are two ways to print

```
Hello World.
This is the second program!
```

The other is to use **escape sequences.**

Escape sequences are used for special characters in strings.

Input/Output

Common escape sequences

Escape sequence	Name	Example Use	Example Output
\'	single quote	cout << "Bob\'s uncle";	Bob' uncle
\"	double quote	cout << "Some \"quotes\" here";	Some "quotes" here
\n	newline	cout << "1st line.\n2nd line.";	1st line. 2nd line.
\t	tab	cout << "This is a \t tab.";	This is a tab.
\?	question mark	cout << "What\?";	What?
\\	backslash	cout << "Use a \\ here";	Use a \ here;

Input/Output

How do you print the following:

```
To print quotes use \".
What are the 'correct' escape sequences?
    OK?
```

Input/Output

How do you print the following:

```
To print quotes use \".
What are the 'correct' escape sequences?
    OK?
```

Use

```
cout <<"To print quotes use \\\".\n";
cout <<"What are the \'correct\' escape sequences\?\n";
cout <<"\tOK\?";
```

Input/Output

Suppose you want to ask someone to enter their age.

Please enter your age?

You can print the question to screen.

```
cout << "Please enter your age\?";
```

However, how do you read the answer?

Input/Output

To read input use cin.

- However, input needs to be stored. It needs to be written to a **variable**.
- First a variable needs to be **declared**:

```
int age;
```

- This means that age is a variable, and its value is an **integer** (a whole number).
- Then read the input and write it to age as follows.

```
cin >> age;
```

Input/Output

This gives

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

int main()
{
  int age;
  cout << "Please enter your age\?";
  cin  >> age;
  cout << "Hello.\nYou are " << age << " years
  old";
    return 0;
}
```

This program asks for the age and prints a welcome message.

Input/Output

Second look at

```
cin  >> age;
```

- The input cin comes from the keyboard. Use `#include <iostream>`
- The operator `>>` reads input from `cin`.
- The input is then written to variable age.

Input/Output

- Another type for input are strings.
- Suppose we want to read a **name**.
- Strings are a string of characters. A variable for a string is defined as follows:

```
string name;
```

- This tells the compiler:

`name` is a variable for a string

Input/Output

- We use this as follows:

```
#include <iostream>
using namespace std;
int main()
{
  string name;
  cout << "Please enter your name\? ";
  cin  >> name;
  cout << "Hello " << name << ".\n";
  return 0;
}
```

The First programs

Lessons learned (beyond standard input and output)

- Programming is an iterative process of trial and error.
- Many mistakes are made. Often inadvertently.
- Debugging is part of the process.
- An IDE and compiler help, but you have to do the work.

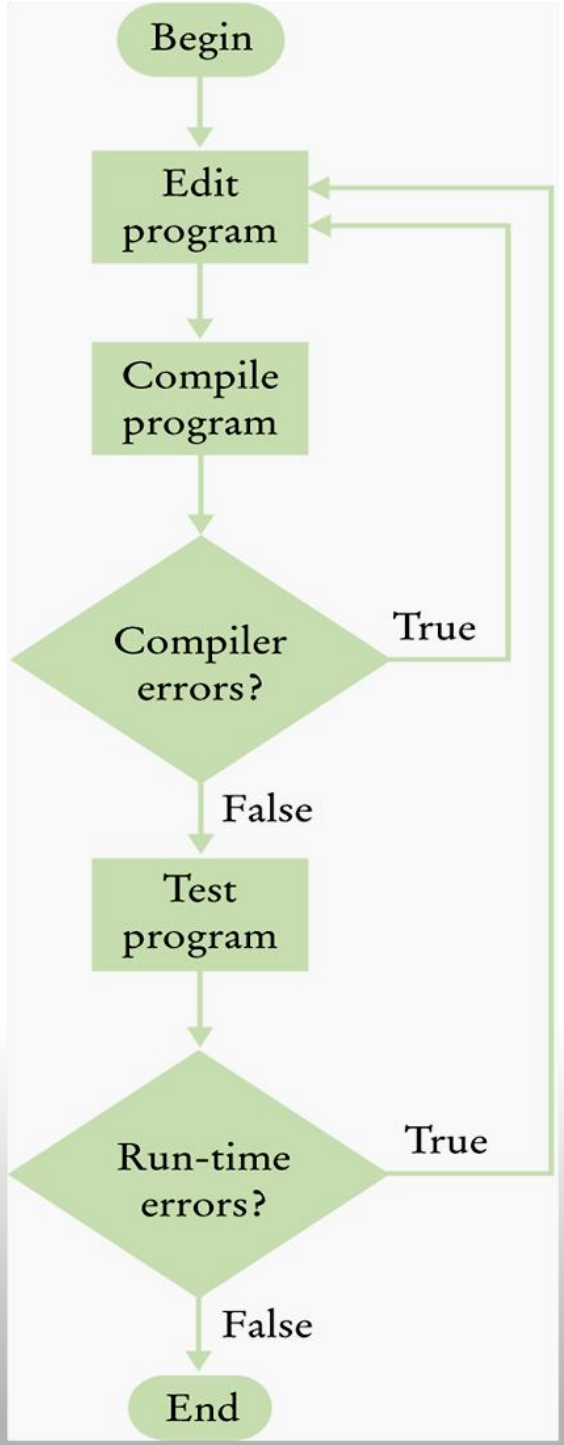
The First programs

Programming is a process

- Editing
- Compiling
- Debugging
- Testing
- Analysing

But this is only part of the bigger picture.

Next: Thinking about Algorithms



Errors and Warnings

A compiler can give you **Errors** and **Warnings**.

- **Error** means that the compiler could not finish its job.
- **Error** means that there is no executable code.
- **Error** means that you have to find and fix the problem.
- This is an **error at compile time.**

- **Warning** means that the compiler could finish its job.
- **Warning** mean there is an executable, but it may hang or worse.
- **Warning** means that you should look again, see if it is what you want.
- This could be an **error at runtime**.

Errors and Warnings

A compiler can give you **Errors** and **Warnings**.

A compiler will only find some errors.

There is a difference between **syntax** and **semantics**.

Compilers are good at finding **syntax errors**.

Compilers are bad at finding **semantic errors**.

Syntax and Semantics

- Syntax : is the structure of the language
  : the form or structure of the expressions, statements, and program units

Eg. Using: cout <<“ Hello World”;

Syntax and Semantics

- Semantics : the meaning of the expressions, statements, and program units

Eg. When Using: cout <<“ Hello World”;

Semantics is: what does “hello world” mean?

Syntax and Semantic

How do we prevent errors and mistakes that the compiler does not find?

By

- Analysing the problem.
- Analysing the solution.
- Testing.
- Thinking.
- Planning.