Input/Output

I/O is not defined as part of either the C or C++ languages.

In C functions *printf()* and *scanf()* etc are provided by the *stdio* library.
These are also available to C++ programmers.

In C++ additional capabilities are provided by the *iostream* library,
which is part of the ANSI Standard Library.

Input/Output

- Programs using the *iostream* library must have

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

  The *iostream* library

  – Provides the predefined stream objects

    `cout` which is tied to the standard output (screen)
    `cin` which is tied to the standard input (keyboard)
    `cerr` which is tied to the standard error (screen)

  – Overloads the operators

    << and >>

    to provide output and input respectively

  – Provides member functions to provide additional functionality.

- The classes *ifstream*, *ofstream* and *fstream* allow

    a file to be used

  for *input*, *output* or *input/output* respectively.

<< Output Operator

`cout << value`

will display ***`value`*** on the screen, where ***`value`*** may be a *constant*, *variable* or *expression* of any of the standard types.

If ***`value`*** is of type:

`char*` (pointer to an array of chars)

Successive characters displayed until a NULL character is encountered.

`short, int, long, unsigned`

Number is displayed as a decimal integer using as many characters as needed.

`float, double`

Displays the number with a default precision of 6 significant figures.
Embedded decimal point used normally. Exponential format is used if this would require fewer characters

`pointer` (other than a pointer to an array of chars)

Address displayed in hexadecimal.

`endl`

Outputs a *newline* character.

`iostream` manipulators

Output formatting capability is provided by the *iomanip* library. Header used is

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

```
cout << setprecision(4);
```

Print 4 significant digits for floating point numbers (stays in force until changed).

```
cout << width(10);
```

The next (only) item output will occupy at least 10 characters.

```
cout << setf(ios::left);
cout << setf(ios::right);
```

Items will be output left (or right) justified in the field.

Further information on iostream manipulators is given at
http://www.cplusplus.com/ref/iostream/ios_base/

>> Input operator

`cin >> variable`

will read a value of the appropriate type and assign that value to *variable*.

If ***`variable`*** is of type:

`char*` (pointer to an array of chars)

white space characters will be skipped, and successive characters will be read into successive memory locations until *white space* is encountered.

`short, int, long, unsigned`

white space will be skipped, and successive decimal digits will be read until a non-digit character is encountered. These digits will be converted to a number and assigned to ***`variable`***.

`float, double`

white space will be skipped, and decimal digits, possibly including a decimal point and possibly followed by an exponent will be read until a character which could not be part of a floating point number is encountered. These characters will be converted to a number and assigned to ***`variable`***.

`white space –` any number of the characters *space*, *tab* or *newline* in any order.

Other Input Operations

- The overloaded operator >> will input the next *word* of a character string. Often this is inconvenient.
- There are functions of the `istream` (and its descendant `ifstream`) class providing different options:

```
istream& getline(char* buff, int len, char delim = '\n');
```

Extracts characters from input until

`delim` character is encountered, or

`len` characters have been read:

```
istream& get(char & ch);
```

Puts the next character (even if it is *white space*) into the reference variable `ch`.

Other Input Considerations

- The code sequence

```
int age; char name[50];
cout << “What is your age? ”;
cin >> age;
cout << “What is your name? “;
cin.getline(name, 50);
```

will put an empty string into name, this is because the `cin >>` left the newline character in the input buffer.

The correct sequence is:

```
int age; char name[50];
cout << “What is your age? ”;
cin >> age;
cin.ignore(50, ‘\n’);
cout << “What is your name? “;
cin.getline(name, 50);
```

The above is a common error which leads to strange results

Checking for Incorrect Input

Consider the code sequence

```
cout << “What is your age? ”;
cin >> age;
```

What happens if the user inputs the string `xyz` ? The system can not convert this to an integer.

To guard against this we need to do something like:

```
cout << “What is your age? ”;
cin >> age;
while(cin.fail()){
   cin.clear()
   cout << “You must input a number”;
   cin.ignore(50, ‘\n’);
   cin >> age;
}
```

File Input/Output

- Programs using any file I/O must:

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

- To open a file for **output**

```
ofstream myOut("filename");
if(!myOut)
   cout << "File filename could not be opened";
```

`myOut` can now be used in the same way as `cout`, except that output will go to `filename`, rather that to the screen.

- To open a file for **input**

```
ifstream myIn("filename");
if(!myIn)
   cout << "File filename could not be opened";
```

`myIn` can now be used in the same way as `cin`, except that input will come from `filename`, rather that from the keyboard.