

INPUT/OUTPUT - INTRODUCTION

- The line

```
#include <iostream.h>
```

enables I/O to be performed

Two variables are declared by C++

- **cin** default input stream
- **cout** default output stream

- A stream is a sequence of bytes (characters) that can be read from or written to

- **cin** is a stream on the keyboard input
- **cout** is a stream on the screen output

INPUT

- The extractor >> (called operator get-from) is used to read from the input stream

```
int x;
```

```
cin >> x;
```

causes an integer value to be read from the input and stored in x

- **Note:**

- When looking for the input value in the stream, the >> operator skips any leading **whitespace** characters and stops reading at the first character that is inappropriate for the data type (whitespace or otherwise).
- If an inappropriate character is read, the cin stream enters a **fail state** and the rest of the statements in our program that read from cin are ignored.

INPUT EXAMPLES

i, j and k are int variables, ch is a char variable, and f is a float variable:

Statement	Data	Contents After Input
cin >> i;	32	
cin >> i >> ch >> f;	25 A 16.9	
cin >> i >> ch >> f;	25A 16.9	
cin >> i >> ch >> f;	25 A16.9	
cin >> i >> ch;	25 A 32.6	
cin >> i >> ch >> f;	25 A	
i = j = k = 1;	2 3.5 A	
cin >> i >> j >> k;		
i = j = k = 1;	2 A 0	
cin >> i >> j >> k;		

Programming 1

Lecture 6 -- 3

MORE ON INPUT

- ❑ The >> operator skips any leading **whitespace** characters while looking for the next data value in the input stream
- ❑ You can use the **get** function to input the very next character in the input stream **without** skipping any whitespace characters:

```
char someChar;  
cin.get(someChar);
```
- ❑ The **ignore** function is used to skip characters in the input stream:

```
cin.ignore(200, '\n');
```
- ❑ The first parameter is an **int** expression; the second, a **char** value. This skips the next 200 characters or until a newline character is read, whichever comes first

Programming 1

Lecture 6 -- 4

OUTPUT

- The inserter << (called operator out-to) is used to add to the output stream

```
cout << "hello";
```

causes the characters `hello` to be written to the **output stream** and onto the terminal screen

```
int x = 100;  
cout << x;
```

causes the **textual representation** of the value of `x` to be written to the output stream

EXAMPLE PROGRAM OF INPUT/OUTPUT

```
#include <iostream.h>  
int main ()  
{  
    int value;  
    // ask user for some input  
    cout << "type a number: ";  
    cin >> value;  
    // print out the value  
    cout << "your input was: ";  
    cout << value << endl;  
    return 0;  
}
```

USING MANIPULATORS

- ❑ You can format **explicitly** by inserting whitespaces e.g.

```
cout << 5 << ' ' << 4 << " " << 3 << '\t' << 2;
```

which will cause

5 4 3 2

to be printed to the screen, alternatively you may wish to use **manipulators**

- ❑ The first manipulator we will look at is **setw()**

This manipulator sets the width of the field to be printed to the screen

- ❑ Example:

```
cout << 5 << setw(4) << 6 << 7;
```

will cause

5 67

to be printed to the screen

USING MANIPULATORS

- ❑ The next manipulator is **setprecision()**. This manipulator sets the precision that **floating point** types are printed at

setprecision works in the following way,

```
cout << setprecision(4) << 1.12345;
```

will cause

1.123

to be printed to the screen

```
cout << setprecision(1) << 1.12345;
```

will cause

1

to be printed to the screen

USING MANIPULATORS

- ❑ The command **setprecision** can also be used to specify the number of digits after the decimal point. To do this you must include the line

```
cout << setiosflags(ios::fixed);
```

before your cout statements, e.g.

```
cout << setiosflags(ios::fixed);  
cout << setprecision(2);  
cout << 12.1234 << endl;
```

will cause

12.12

to be printed to the screen

USING MANIPULATORS

- ❑ To put **setprecision** back to its original specification you can include the line

```
cout << setiosflags(ios::scientific);
```

- ❑ You can alternate between the different modes as often as you desire
- ❑ In order to use the previously described manipulators you will need to **include** in your C++ source code, the header file
#include <iomanip.h>

FILE INPUT/OUTPUT

- We can define other objects of class **istream** and **ostream**

```
istream inputStrm;  
ostream outputStrm;
```

Therefore we can have

```
irt x;  
inputStrm >> x;           // read in input  
outputStrm << x;         // print out output
```

- In a similar way C++ provides **streams** which can manipulate **files**

FILE STREAMS

- C++ provides **2 file streams**

```
ifstream input file stream  
ofstream output file stream
```

Must **#include <fstream.h>** to use them

- Example:

```
#include <fstream.h>  
  
irt number;  
ifstream in("in.dat");  
ofstream out("out.dat");  
  
in >> number;  
out << number;
```

INPUT FILE STREAMS (IFSTREAM)

- ❑ Allows data to be read from a file
- ❑ An input file stream can be defined as follows:

```
ifstream stream_var(filename);
```

Example:

```
ifstream inFile("test.dat");
```

- If stream opened **OK**, inFile evaluates to **positive** and the stream becomes attached to the file test.data
 - If stream open **failed** (e.g. file does not exist) inFile evaluates to **zero**
- ❑ **Important:** Effects of reading data from file which has failed to open is undefined

INPUT FILE STREAMS (IFSTREAM)

- ❑ When file opened **OK**, data can be read using normal **extractor** functions

```
int n;  
char c;  
inFile >> n;  
inFile.get(c);  
inFile.ignore(100, 'A');  
inFile.close();
```

- ❑ **Note:** When a file stream goes out of scope it will automatically close the file it is attached to

FILE INPUT FAILURE/END

- ❑ To check if the file has been **opened or not**, you can use:

```
if (inFile) // testing if the file opened OK
{ ... }
```

- ❑ To test for **end of file**, you can use:

```
while (!inFile.eof())
{ ... }
```

For instance:

```
int number;
inFile >> number;
while (!inFile.eof())
{
    cout << number;
    inFile >> number;
}
```

OUTPUT FILE STREAM (OFSTREAM)

- ❑ Allows data to be written to a file

An output file stream can be defined as follows:

```
ofstream stream_var(filename);
```

Example:

```
ofstream outFile("temp.data");
```

- If stream opened **OK**, outFile evaluates to *positive* and the stream becomes attached to the file temp.data
- If stream open **failed** (e.g. no disk space) outFile evaluates to **zero**

- ❑ **Note:**

- If the file already exists its contents will be deleted
- If the file does not exist, a file with the same name is created
- Data can be appended to a file by using constructor with two arguments
ofstream outFile("temp.data", ios::app);

EXAMPLE ON HOW TO WRITE TO A FILE

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>

int main ()
{
    float first, second, sum;           // Declaring variables
    ofstream outFile("out.dat");       // Opening file for output

    cout << "Enter two numbers" << endl;
    cin >> first >> second;           // Reading in the two numbers
    sum = first + second;
    outFile << setiosflags(ios::fixed); // Formatting the output
    outFile << setprecision(2);
    outFile << sum << endl;          // Writing into the file
    return 0;
}
```