C++ Programming

Input/Output and Files

M1 Math
Michail Lampis
michail.lampis@dauphine.fr

Basic C++ I/O

- We have already seen how we can read/write basic things

```
#include <iostream>
using namespace std;
...
int x;
cout << “Give me a number” << endl;
cin >> x;
cout << “You gave me “ << x << endl;
```

Streams

- In the previous slide we were using the cout and cin objects
- These objects belong in the general category of **streams**, which is the class C++ uses for I/O
- Basic stream objects
  - cout : ostream
  - cin : istream
  - cerr : ostream

Using streams

- Include the header file <iostream> (this is where all the definitions are contained)
- Main operators are << and >>
- << “sends” something to an ostream **and returns the stream again**

  cout << x1 << x2 << x3 << …
- >> “gets” something from an istream

  cin >> x1 >> x2 >> x3 >> ...

Reminder: << and >>

- Recall that << and >> have a standard meaning in C/C++: binary shift

  (23 << 2) == 92

  (23 >> 2) == 5

- However, when applied to stream objects, these operators have a different meaning
  - This is called **operator overloading.** We will see more of it later...

Strings in C/C++

- Recall that for C strings of characters are stored as arrays of type char

  char *myword = “Hello”;

  //myword[1] == 'e'

- Normally, C-type strings are terminated with a special '\0' character. Thus, their size can be figured out by looking for this character
- **THIS IS A PROBLEM!**
  - No.1 cause of security bugs in C programs: user is allowed to supply a string, and we trust them to give reasonable length...

Strings in C++

- In C++ it is strongly preferred to use the special type (class) string to store strings.

  string x = “Hello”;

- Advantage: memory management more robust
- Also: many high-level operations easy

  string y = “ World”;

  cout << (x+y) << endl; //OK

  cout << (x<y) << endl; //Alphabetic ordering

- Array-like access still works: x[1] == 'e';

Strings and cin

- Normally cin tries to “parse” the input into the appropriate type

```
int x;
string y;
cin >> x;
cin >> y;
```

- An input like “5 15” would set x=5 and y=”15”
- What if the user does not enter a number for x?

Strings and cin

- cin also tries to parse the input when reading a string

  ```
  string x;

  cin >> x;
  ```

- If input is “Hello world” x will be set to “Hello” → strings are broken at whitespaces

Strings and cin

- A solution to this is the getline( ) function, which reads a whole line up to the enter key

```
string x;

getline(cin, x); //Sets x to be equal to a full line
```

Reading/Writing in files

- Files (stored on the OS filesystem) can also be used as streams
- Include the header <fstream>
- Use one of the following classes:
  - ofstream: output file stream (for writing only)
  - ifstream: input file stream (for reading)
  - fstream: file stream (for both)

File streams

- Example:

```
ofstream myfile;
myfile.open("testfile.txt");
myfile << "Hello file!" << endl;
myfile.close();
```

- You must first open a (text) file
- Then write on it as with cout
- When done, you must close it

Opening a file

- The first step to start our I/O stream on a file is to open it, with the open method
- Syntax: myfile.open(NAME, MODE)
- MODE specifies options on how the file is opened
  - ios::in → for input
  - ios::out → for output
  - ios::ate → start (reading) at the end
  - ios::app → append (start writing at the end)
  - ios::binary → not a text file

Opening a file

- More than one options can be used with the | (bitwise OR) operator

  ofstream myfile;

  myfile.open(“test.txt”, ios::app | ios::out)
- The file can be opened at declaration (useful)

  ofstream myfile(“test.txt”, ios::app); //equivalent
- Check for errors!

  if(myfile.is_open( )) { … OK … }

Reading a text file

- An ifstream can be read in the same way as cin (with the >> operator)
- You can also use the .eof() function to check if you have reached the end of the file
- Typical pattern

  while(!myfile.eof()) { … read more … }

Exercise

- Write a program that reads a text file made up of integers (one in each line)
- The size of the file is not known beforehand
- Your program must find and print the largest integer in the file

Solution

```
ifstream myfile;
myfile.open("testfile.txt");
int max;
myfile >> max;
while(!myfile.eof()){
        int x;
        myfile >> x;
        if(x>max) max = x;
}
myfile.close();
cout << "The max was " << max << endl;
```

Binary files

- For files that do not contain text we cannot use the << and >> operators
- We can however use .read and .write
- Syntax:

  myfile.read(position,size);
- Def: position is a char *. It is a generic pointer to the place in memory where data will be stored. **You need to allocate this!**
- Size is the number of bytes to read from the file