

C++ Files

6.1 Header Files

A C++ class is usually split over a *header file* and an *implementation file*. The header file lists the data and gives prototypes for the functions. Many people put the public members first. The implementation file gives the member function implementation. These are specified with the :: *scope resolution* operator. Your own header files should be loaded after the system ones, and the name should be in quotation marks.

Multiple inclusion of header files leads to multiple definitions of the same class, which the compiler cannot handle. So, header files should have `ifndef` as a preprocessor directive. The standard is to define a value with the same name as the file, but capitalized and with period replaced by underscore:

```
// file myHeader.h
#ifndef MYHEADER_H
#define MYHEADER_H
    ... code as before ...
#endif
```

6.2 Libraries

Mathematics. Mathematical functions are available in the library `cmath`. Note that angles are represented in radians.

More on output. Including `<iomanip>` allows one to format stream output using what are called *manipulators*. For example, `setw()` sets the minimum size of the output field, `setprecision()` sets the precision, and `fixed` ensures that a fixed number of decimal places are displayed. For example

```
double A = 4.999;
cout << setprecision(2) << showpoint;
cout << A;
```

produces 5.0 as output.

6.3 File Input

File input can be made through use of an input file stream. This is opened with the external name of the file. There are `cin`-style operators: that is, `stream >> A` reads

the variable A); note that the `stream` becomes null if the read fails. There are also functions taking the stream and a variable. The file should be closed after using. Here is code to copy a file to the standard output:

```
ifstream inFile;
inFile.open( "testing.txt" );
if( !inFile )
    cout << "Could not open file" << endl;
else {
    string oneLine;
    while( inFile.peek() != EOF ) {
        getline(inFile, oneLine);
        cout << oneLine << endl;
    }
    inFile.close();
}
```

6.4 Sample Program: DoubleList

A simple implementation of a List of double's. Note the use of a `main` method in the implementation file as a quick way to do some testing.

```
// DoubleList.h - wdg - 2009
// header for primitive List

#ifndef DOUBLELIST_H
#define DOUBLELIST_H

class DoubleList
{
public:
    static const double NO_SUCH_ELEMENT = -99.99;
    DoubleList();
    void add(double item);
    double getItem(int position) const;
    void loadFile(string fileName);

private:
    static const int MAX=100;
    double A[MAX];
    int count;
};
```

```

};

#endif

```

```

// DoubleList.cpp - wdg - 2009
// implementation code for a list of doubles

#include <fstream>
#include <iostream>
using namespace std;
#include "DoubleList.h"

DoubleList::DoubleList() : count(0)
{
}

void DoubleList::add(double item)
{
    if(count==MAX)
        return;
    A[count]=item;
    count++;
}

double DoubleList::getItem(int position) const
{
    if(position<0 || position>=count)
        return NO_SUCH_ELEMENT;
    else
        return A[position];
}

void DoubleList::loadFile(string fileName)
{
    ifstream inFile;
    inFile.open( fileName.c_str() );
    if( !inFile ) {
        cout << "Could not open file" << fileName << endl;
        return;
    }
}

```

```

    cout << "Loading ";
    double temp;
    while( (count<MAX) && (inFile >> temp) ) {
        cout << temp << " ";
        A[count]=temp;
        count++;
    }
    cout << endl;
    inFile.close();
}

// a very inadequate test program
int main( )
{
    DoubleList B; // calls constructor
    B.loadFile("numbers.txt"); // integers 1 up to 10
    B.add(1.414);
    cout << B.getItem(0) << " " << B.getItem(10) << endl;
    return 0;
}

```