#include <iostream>

#include <vector>

#include <string>

using namespace std;

int main()

{

vector<string> names;

while (more\_data())

{

string temp = get\_more\_data();

names.push\_back(temp);

}

// Maybe a little inefficient because of the insertions.

// If we know in advance how many elements we will have,

// we could do:

names.resize (num\_elements);

for (int i = 0; i < num\_elements; i++)

{

names.at(i) = get\_more\_data();

}

// We could also use names[i] instead of names.at(i),

// given that we know for sure that there won't be a

// subscript overflow in that for loop, therefore, we

// don't need checked access to the elements of the array

// Sort the elements -- assuming that we provided a function to

// do so, say,

// template <class T>

// void sort (vector<T> &);

sort(names);

// Now print the values

for (int i = 0; i < names.size(); i++)

{

cout << names[i] << endl;

}

// We could also do it using an iterator

vector<string>::iterator i;

for (i = names.begin(); i != names.end(); ++i)

{

cout << \*i << endl;

}

return 0;

}

#include <iostream>

#include <list>

#include <string>

using namespace std;

class Student

{

public:

// ... various functions to perform the required operations

private:

string name, ID;

int mark;

};

int main()

{

list<Student> students;

// Read from data base

while (more\_students())

{

Student temp;

temp.read();

students.push\_back (temp);

}

// Now print the students that failed (mark < 60%) - of

// course, the particular Student object should provide a

// member-function (say, passed()) that will determine that

list<Student>::iterator i;

for (i = students.begin(); i != students.end(); ++i)

{

if (! i->passed()) // iterators also provide operator ->

{

cout << "The student " << \*i << " failed." << endl;

// provided that class Student provides the overloaded

// stream insertion operator <<

}

}

// Now remove the failed students (of course, this could have

// been done in the previous loop)

i = students.begin()

while (i != students.end())

{

if (! i->passed())

{

i = students.erase (i);

}

else

{

++i;

}

}

// ...

return 0;

}