# Dynamic Array Example

In the last lab we used a fixed array with a large max value to hold a variable sized list. Work an example of making this a formal class.

#include <iostream>

using namespace std;

void main() {

int data[100]; // Fixed size ... max size

int dataSize = 0; // Current size

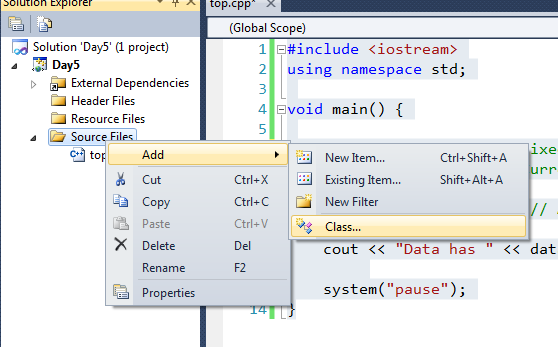
data[dataSize++] = 43; // Adding an item to the end

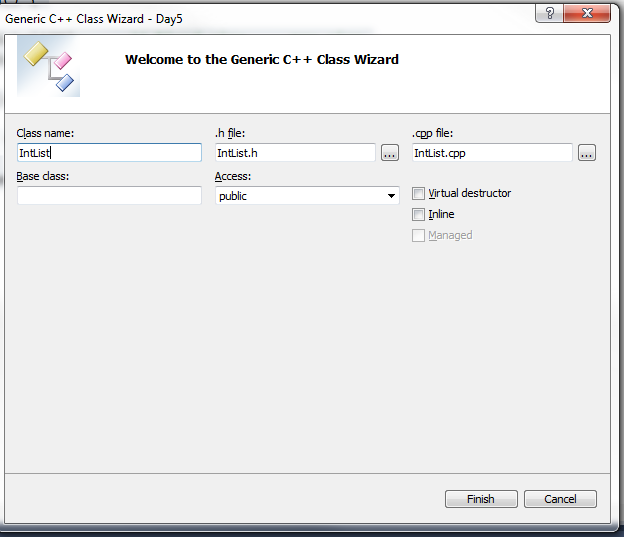
cout << "Data has " << dataSize << " elements.";

system("pause");

}

The IDE will stub out headers/source for you. In Express it looks like this.





You get a “.h” and a “.cpp” with the constructor and destructor already stubbed out.

Start with the interface. What kinds of things do you want to be able to do to this object? Add those to the header file.

#pragma once // Microsoft specific ... better than #ifndef stuff

class IntList

{

public:

IntList(void);

~IntList(void);

int getSize();

void addElementToEnd(int value);

int getElementAt(int index);

void setElementAt(int index, int value);

void insertElementAt(int index, int value);

void removeElementAt(int index);

};

For now just a fixed size array. Add data and constructor.

class IntList

{

int data[1000]; // For now a fixed size

int size; // How many elements are in the list

public:

IntList::IntList(void) {

size = 0;

}

int IntList::getSize() {

return size;

}

#include <iostream>

using namespace std;

#include "IntList.h"

void main() {

IntList data;

cout << "Data has " << data.getSize() << " elements.";

system("pause");

}

Notice that the compiler doesn’t complain because you haven’t defined the other methods. It won’t care until you actually try and use them.

Some thoughts are to design the entire class and write the code completely then test. I prefer a more “extreme” (a type of programming methodology) technique by adding/testing/adding/testing incrementally. If you are working in an embedded environment this may not be practical.

void IntList::addElementToEnd(int value) {

if(size==99) return;

data[size++] = value;

}

IntList data;

data.addElementToEnd(123);

int IntList::getElementAt(int index) {

if(index<0 || index>99) return 0;

return data[index];

// Would be better to use an unsigned to avoid the <0 check

// Would be better to use a constant instead of 99

}

IntList data;

data.addElementToEnd(123);

cout << "Data has " << data.getSize() << " elements." << endl;

cout << data.getElementAt(0) << endl;

void IntList::setElementAt(int index, int value) {

if(index<0 || index>99) return;

data[index] = value;

}

Now for the fun ones: inserts and deletes.

For an insert the new element will be at that index. Make a note of this in the comments.

void IntList::insertElementAt(int index, int value) {

if(size==99) return;

if(index<0 || index>=size) return;

// A picture helps here. Check boundary cases.

for(int x=size;x>index;x=x-1) {

data[x] = data[x-1];

}

data[index] = value; // could call setElementAt

++size;

}

void IntList::removeElementAt(int index) {

if(size==99) return;

if(index<0 || index>=size) return;

// A picture helps here. Check boundary cases.

for(int x=index;x<size;++x) {

data[x] = data[x+1];

}

--size;

}

IntList data;

data.addElementToEnd(100);

data.addElementToEnd(200);

data.addElementToEnd(300);

data.insertElementAt(2,150);

cout << "Data has " << data.getSize() << " elements." << endl;

for(int x=0;x<data.getSize();++x) {

cout << data.getElementAt(x) << endl;

}

# Templates

Want to make a list of “doubles” … of “float” … of ANYTHING else. The logic is the same for all, just different type.

The logic is GENERIC … works with a GENERIC type.

I can use a word processor to search/replace the type for me. Mark the type:

class List

{

TOPHER data[1000]; // For now a fixed size

int size; // How many elements are in the list

public:

List();

~List();

int getSize();

void addElementToEnd(TOPHER value);

int getElementAt(int index);

void setElementAt(int index, TOPHER value);

void insertElementAt(int index, TOPHER value);

void removeElementAt(int index);

};

void List::addElementToEnd(TOPHER value) {

if(size==99) return;

data[size++] = value;

}

Search TOPHER with “double” or “float” or “long” or whatever.

Compiler can do this for you!

template <class TOPHER>

class List

{

TOPHER data[1000]; // For now a fixed size

int size; // How many elements are in the list

public:

The compiler needs all the code in the header file.

List<int> data;

List<double> more;

List<std::string> sdat;

Compiler does the substitution for you and makes code for a class that handles “int”.

Does it work for “string”? Does it work for “Point”?

Try this simple implementation of Point:

class Point {

int x;

int y;

public:

Point(int nx,int ny) {

x = nx;

y = ny;

}

};

1>c:\cpp\_7\_2012\day5\day5\intlist.h(11): error C2512: 'Point' : no appropriate default constructor available

* The data[1000] uses the no-arg constructor. Must have one.
* data[size++] = value; Assignment operator
* Copy constructor (passing to function)

Can have regular template functions (that aren’t classes).

Change getElement and setElement to use []