EL2310 Scientific Programming
LAB3: C++ lab session

Patric Jensfelt

Rev 1: Autumn 2007
Rev 2: Autumn 2009

Chapter 1

Introduction

1.1 Reporting errors

As any document, this document is likely to include errors and typos. Please report these to `patric@kth.se`.

1.2 Acknowledgements

Valuable feedback and error reports on this document have been provided by:

- No one so far...

1.3 Before getting started

Have a look at the first part of the Matlab lab session if you are not familiar with Unix/Linux computers. If you want to run the lab on your own Windows laptop then Cygwin might be an alternative. Another way would be to using VMWare and create a virtual Linux machine on your windows machine. For information on Cygwin see the lecture notes.

Chapter 2

Lab instructions

2.1 Class definition

One of the core concepts in C++ is the `class`. In C we looked at using a `struct` to create compound data types of simple data types. The class takes this a step further and allows you to have not only data but also functions that operates on the data all collected in one unit, the class.

Task

Define a class Matrix such that the you can run the following program:

```
Matrix m1(3,3);
m1 = 0; // set all elements to the same scalar value
for (int i = 0; i < 2; i++) m1(i,i) = 1+i*2;
cout << ‘‘m1=’’ << m1 << endl;

Matrix m2(3,3);
m2 = m1; m2(0,2) = 5;
m2(1,0) = 42;
cout << ‘‘m2=’’m2 << endl;

Matrix m3(m1+m2);
cout << ‘‘m3=’’ << m3 << endl;
cout << ‘‘m3-m2=’’ << m3-m2 << endl;
cout << ‘‘m3*m2=’’ << m3*m2 << endl;
```

Help

You might want to take a look at operators like the following when solving the task

- ostream& operator<<(ostream &os, const Matrix &m);
- Matrix operator+(const Matrix &m1, const Matrix &m2);

2.2 Reference

The standard way of passing arguments to functions is “by value”, i.e. you pass the value of the function argument into the function. For small data types like the basic datatypes this is not in general a big problem from a performance point of view. For large data types, like images, this should be avoided if possible. There are two ways in C++ to avoid this in C and C++, one is to use pointers (which you can do in C as well), the other is to use a reference.

Consider the three functions

```
void fcn1(double x) {
x = 2;
}

void fcn2(double *x) {
x = 42;
}

void fcn3(double &x) {
x = 4711;
}
```

Task

Make sure you knwo the difference between the three? Make sure you know how to call the three functions. What will happen to the variable you call it with? Can you do `fcn1(4)`? What about `fcn2(4)` and `fcn3(4)`.

2.3 `const`

Using references is often motivated by avoiding to copy large pieces of data. However, passing a variable by reference allows the function to change the value of the variable which might not always be desirable.

Task

- Make sure you know the difference between the following constructions (you can combine them as well)

  1. `const double& fcn1(double x);`
  2. `double fcn2(double x) const;`
  3. `double fcn3(const double& x);`

- Give examples of when they would be used.
- What is the condition for a function to be called from inside `fcn2` from above?

2.4 Inheritance

Inheritance is a powerful method that allows you to express dependencies of type "is a" between classes. The syntax for inheritence is
`class B : public A { ...`
if you want `B` to inherit from `A`.

Task

- Implement a class hierachy with the base class Animal and at least two levels of subclasses such as Mammal and Human.
- What information/functionality would you put at what level
- If you have a function that accepts an Animal as input is ok to send in a Mammal? Why? What if it the function wanted a Mammal, could you send in an Animal?
- What if you implement the same function in all classes, which one will be used?
- Investigate what happens if you change the public in the inheritance to `protected` and `private`.

2.5 virtual

Function overloading allows you to re-define the meaning of a function. However, if you use normal overloading you need to have a reference or poiner to the exact right subclass to get the right implementation. In other words, continuing the example from above, if you have a function that takes an Animal as input there is no way that you can use any of the overloaded functions that might be in the subclasses. The solution to this is to use the `virtual` keyword.

Task

Implement a base class `A` and let this class have two function printInfo1 and printInfo2 defined like this

```
void printInfo1();
virtual void printInfo2();
```

Add some data member to this base class and implement the print functions so that they print the information about the object.

Now implement a subclass `B` of `A`. Add some new data in B and overload the two print functions such that they print the content of A and B (Can you re-use the print functions in A somehow for printing the A part?).

What happens and why with the following program?

```
void print(A &a) {
  a.printInfo1();
  a.printInfo2();
}
int main() {
  A a;
  B b;
  a.printInfo1();
  b.printInfo1();
  a.printInfo2();
  b.printInfo2();
  print();
}
```

2.6 STL

The Standard Template Library (STL) contains some handy classes that are often of use. These classes are so called template classes that allow you to specify what data type they should operate on. The idea with templates is that you can define how a class or function should work on some general data type. One example of this is a list. Instead of having to implement a list class for every data type you might want to put in the list you can define a list template class so that you define when you use it what type of data to put in it.

These template classes can be found in the `std` namespace. To use a class such as the list class you need to specify what type the list should contain, such as `std::list<std::string>` for a list of strings.

Three of the most common types are `std::list`, `std::list` and `std::map`.

2.6.1 `std::list`

A list provides the means to store elements in a list and provides means to step back and forth between elements, check how many elements (size()). A list is a good choice if you do not know before hand how many elements you will want to store and it is expensive to create new objects. When a new element is added to a list space only needs to be allocated for the new element and it is inserted into the structure by redefining some pointers internally. A list is a bad choice if you want to be able to access an element at a particular position in the list fast as you need to step element by element through the list. To move between the element you use so called *iterators*. The code below illustrates how you can use a list.

```
#include <list>

std::list<double> dlist;
dlist.push_back(4.12);
dlist.push_back(1);
dlist.push_front(3.14);
dlist.push_back(2.78);

std::cout << "List contains:" << std::endl;
for (std::list<double>::iterator i = dlist.begin(); i != dlist.end();
i++)
  std::cout << *i << std::endl;
std::cout << std::endl;

dlist.pop_front();
dlist.pop_back();
std::cout << "List contains:''  << std::endl;
for (std::list<double>::iterator i = dlist.begin(); i != dlist.end();
i++)
  std::cout << *i << std::endl;
std::cout << std::endl;

std::cout << "The first element is " << dlist.front() << std::endl;
std::cout << "The last element is " << dlist.back() << std::endl;
```

2.6.2 `std::vector`

A vector provides an interface for access similar to that of an array. That is if `v` is a vector you can access the first element with `v[0]`. You can access the elements using using the iterators as well. You can only push new objects at the end and only pop them from the back. A vector allows you to preallocate the size (resize()) of the vector so that you do not need to resize it every time you add an element.

2.6.3 `std::map`

A `map` allows you to start pairs of objects where the first one is the index/key which can be used in find operations. An example of using a map is given below.

```
#include <map>

std::map<int,double> dmap;
dmap.insert(std::make_pair(2,2.78));
dmap.insert(std::make_pair(1,3.14));
dmap.insert(std::make_pair(42,4711));
std::cout << "Elements in map:" << std::endl;
int n = 0;
for (std::map<int,double>::iterator i = dmap.begin(); i != dmap.end();
i++)
std::cout << "Element " << ++n << " " << i->first << ":" << i->second
<< std::endl;
std::cout << std::endl;
std::map<int,double>::iterator f = dmap.find(2);
if (f != dmap.end()) std::cout << "found " << f->second << std::endl;
else std::cout << "Did not find any element" << std::endl;
```

Task

- Play with vectors and lists in different situations to see how they behave when you need to allocate many elements dynamically or when you need to access a certain one. Which is more effecient in what situation?
- Use a map to define a lookup table from month number to month name.
- If you want to create a list of Vehicles where Vehicle is the base class for Car, Motorcycles, etc. How would you define the list such that your list can have any type of Vehicle in it?