EL2310 – Scientific Programming

Lecture 15: Templates, STL, File I/O and Strings in C++

Hakan Karaoguz (hkarao@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 15: Templates, STL, File I/O and Strings in C++

- Reminders
- Wrap up
- Templates
- File I/O
- string in C++

The help session

- C++ help session: 10 October

Lecture 15: Templates, STL, File I/O and Strings in C++

Reminders
Wrap up
Templates
File I/O
string in C++

Inheritance

- Inheritance is a way to show a relation like “is a”
- Ex: a Car is a Vehicle
- A Car inherits many of its properties from being a vehicle
- These same properties could be inherited by a Truck or a Bus
- Syntax:
  `class Car : public Vehicle`
  specifies that Car inherits from Vehicle

Overloading in sub-classes

- We can overload a method in a sub-class

```
class Vehicle {
  void drive();
}
class Car:  public Vehicle {
  void drive();
}
```

- `Vehicle *v1 = new Vehicle();`
- `Vehicle *v2 = new Car();`
- `Car *c = new Car();`
- `v1->drive();` and `v2->drive();` run drive() from the Vehicle
- `c->drive();` runs drive() from the Car

`virtual` functions

- What if we want the object know what it “really” is and run the correct `drive()` method?
- Declare the method with the keyword `virtual`

```
class Vehicle {
  virtual void drive();
}
class Car:  public Vehicle {
  virtual void drive();
}
```

- `Vehicle *v1 = new Vehicle();`
- `Vehicle *v2 = new Car();`
- `v1->drive();` runs drive() from the Vehicle
- `v2->drive();` runs drive() from the Car

Polymorphism with `virtual` functions

- What `virtual` function to run is determined at run-time
- Depends on the “real” type of objects

Interfacing: Abstract class

- In C++, abstract classes provides interfaces
- Not to be confused with data abstraction
- To make a class abstract : declare at least one of its functions as pure ”virtual” function.
- A pure virtual function is specified by placing ”= 0”
- ```
  class Car
  {
  public:
    virtual double getNrWheels() = 0; // pure
  virtual function
  private:
    double NrWheels
  };
  ```
  ```

Abstract class

- Abstract classes cannot be instantiated
- Purpose : A base class which could be inherited in other classes
- Inherited classes have to overload each of the virtual functions in the base class

Lecture 15: Templates, STL, File I/O and Strings in C++

Reminders
Wrap up
Templates
File I/O
string in C++

Template Function

- Templates offers a way to write code compatible with any data types
- Use it when you want a generic function that can work on many different data types
- Example of a template function:

```
template <typename T>
T getMax (T a, T b)
{
  if(a<b) {return b;}
   return a;
}
```

- `getMax<int>(4,5)` returns 5
- `getMax<double>(4.2, 4.1)` returns 4.2
- Compiler generates a version for each data type you use it with

Template Class

- Use it when you want a generic class that can work on many different data types
- Example of a template class:

```
template <class T>
class mypair {
 T a,b;
 public:
  mypair(T first, T second){a=first; b=second;}
  T getmax ();
}
```

Standard Template Library: STL

- The Standard Template Library (STL) provides classes for:
  - Collections: lists, vectors, sets, maps
- Defined as templates: can store data of any type!
- Examples:
  - `std::list<T>`
    Ex: `std::list<std::string> names;`
  - `std::vector<T>`
    Ex: `std::vector<double> values;`
  - `std::set<T>`
    Ex: `std::set<std::string> nameOfPerson;`
  - `std::map<T1,T2>`
    Ex: `std::map<int, std::string> nameOfMonth;`
    Ex: `std::map<std::string, int> monthNumberByName;`

Standard Template Library: STL

- Different collections are optimized for different use, e.g.:
  - `std::list<T>`
    Cannot access elements with `x[i]`, need to use so called *iterators* to step through the list, can add/remove elements at low cost
  - `std::vector<T>`
    Can access elements with `x[i]`, but resizing is more costly
  - `std::set<T>`
    Does not allow for redundant elements
  - `std::map<T1,T2>`
    Provides a mapping from one object to another
- More in C++ Library Reference, e.g.
  `http://www.cplusplus.com/reference/stl/`

Often used: vector (from C++ reference)

```
// erasing from vector
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  unsigned int i;
  vector<unsigned int> myvector;

  // set some values (from 1 to 10)
  for (i=1; i<=10; i++) myvector.push_back(i);
}
```

Often used: vector (from C++ reference)

```
// erase the 6th element
myvector.erase(myvector.begin()+5);

// erase the first 3 elements:
myvector.erase(myvector.begin(),
myvector.begin()+3);

cout << "myvector contains:";
for (i=0; i<myvector.size(); i++) {
 cout << " " << myvector[i];
 cout << endl;
}
return 0;
```

Often used: iterators (from C++ reference)

```
using namespace std;
vector<int> v;
vector<int>::iterator vIt;
v.push_back(2);
v.push_back(3);
for(vIt = vIt.begin(); vIt != vIt.end(); vIt++) {
    cout<<*vIt;
}
```

STL Algorithm Library (from C++ reference)

```
#include <algorithm>
int myints[] = { 10, 20, 30 ,40 };
int * p; // pointer to array element:
p = std::find (myints,myints+4,30);
++p;
std::cout << "The elem. following 30 is ";
std::cout << *p << '\n';
```

STL Algorithm Library (from C++ reference)

```
#include <algorithm>
#include <vector>
int myints[] = {32,71,12,45,26,80,53,33};
std::vector<int> myvector (myints, myints+8);
// 32 71 12 45 26 80 53 33
// using default comparison (operator <):
std::sort (myvector.begin(), myvector.begin()+4);
//(12 32 45 71)26 80 53 33
```

STL Algorithm Library (from C++ reference)

```
#include <algorithm>
#include <vector>
int myints[] = {32,71,12,45,26,80,53,33};
std::vector<int> myvector (myints, myints+8);
// 32 71 12 45 26 80 53 33
// using default comparison (operator <):
bool myfunction (int i,int j) { return (i>j); }
std::sort(myvector.begin()+4,
          myvector.end(), myfunction);
// 32 71 12 45 (80 53 33 26)
```

From double ** pointers to double vectors for Matrix Representations

- `vector` allows us to define multi-dimensional data structures

```
std::vector< std::vector<double> > matrix;
// A 2D matrix using double vectors
```

Lecture 15: Templates, STL, File I/O and Strings in C++

Reminders
Wrap up
Templates
File I/O
string in C++

file streams (fstream)

- We use the headers `iostream` and `fstream` for performing file operations in C++
- Compared to C, C++ offers object based approach through classes `ofstream` and `ifstream`
- File modes, `ios::out` for write, `ios::in` for read, `ios::app` for append
- Ex for writing data to a file:

```
ofstream file("file.txt",ios::out);
if(!file) return -1;
string name;
int age;
cout<<"Enter name and age:";
cin>>name>>age;
file<<name<<" "<<age<<" "<<endl;
```

- Read from file Ex:

```
ifstream file("file.txt",ios::in);
if(!file) return -1;
string name;
int age;
while(file>>name>>age)
  cout<<name<<" "<<age<<endl;
```

Lecture 15: Templates, STL, File I/O and Strings in C++

Reminders
Wrap up
Templates
File I/O
string in C++

Class string

- `string` class provides an easy interface for string-manipulation such as copying, searching, etc.
- The header `<string>` should be included for using strings
- Ex initializations:

```
string text("Hi");
string str; // Empty string
string str = "john";
string charstr('a'); ERROR!!
string numstr = 22; ERROR!!
```

- There is no direct conversion from a single `char` or `int` to string.

string operations

- Assignment: `string str1 = str2;`
- Concatenate: `string str1 += str2`
- Comparison: `if(str1 == str2){}`
- Getting substring: `str1.substr(7,5) //Get the substring composed of 5 characters which starts from the 7th index of str1`
- Swapping: `str1.swap(str2)`
- Finding a substring: `str1.find("is") // Returns the position of the first character if found, -1 otherwise`
- Accessing to the character array `char*` of string: `char* ptr = str.data()`

stringstream

- It is a class that works on string processing (string version of filestream)
- It simplifies creating strings that are composed of various data types. In order to use it, `sstream` header should be included
- Ex:

```
string str = "Hi, I was born in 1990."
int year = 1990;
for(int i = 1; i < 2016-1990; i++) {
   stringstream sstream(str);
  sstream<<" In year "<<year+i<<" I was
"<<i<<" years old.";
  cout<<sstream.str()<<endl;
}
```

Task 1

- Open cpp.sh from your web browser.
- Assume that you need to read a bunch of files from a directory. You have the root path which is given as */home/data/*.
- You should append the filename given as *person*, the person id from 1000 to 2000 and the file ending *.db* to the root path in order to access to a file. Ex: path = /home/data/person1234.db
- Use strings and any other tools to create all the paths iteratively. Output all the paths to the command terminal.
- Optionally write all paths to a text file called *path.txt* if you are running on your local machine.