EL2310 – Scientific Programming

Lecture 16: C++1y and Conclusion

Hakan Karaoguz (hkarao@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 16: C++1y and Conclusion

- Reminders
- Wrap up
- C++11
- Conclusion of the Lectures

The help session and deadline

- C++ second help session: Thu 13 October, 08:00-10:00
- C++ assignment deadline: Wed 19 October, 23:55

Tips for the C++ project

- Initializing a float Mat object:
```
Mat mat = Mat::zeros(rows,cols,CV_32F);
```
- Reading an image from file:
```
Mat image = imread("filename");
```
- Accesing an element of Mat:
```
T element = mat.at<T>(i,j) ;
```
- Assigning a value to an element of Mat (T depends on Mat type such as float, integer, char,etc):
```
T element;
mat.at<T>(i,j) = element;
```

Lecture 16: C++1y and Conclusion

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;`

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;
```

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
```

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
```

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;
```

Lecture 16: C++1y and Conclusion

C++11

- A new revision from 2011 of C++, supported by g++
- Many improvements to C++
- activate using -std=c++11 (default?)
- example: `g++ -std=c++11 main.cpp -o main`

C++11

- Variable type inference:
- `auto a = 42;`
- `auto b = 42.01;`
- `auto c = new MyObject();`
- `auto d = myfunction(a,b,c);`

C++11

- Lambda functions: Ex: `auto func = [] ()  cout << "Hello KTH"; ;`

```
int main() {
    vector<int> x;
    for(int i=1;i<10;i++) { x.push_back(i);}
    auto pos = std::find_if(std::begin(x),
          std::end(x), [](int n) {return n%2==0;} );
    cout << *pos; // is 2
};
```

C++11

easy looping

```
using namespace std;
int main() {
    vector<int> x;
    for(int i=1;i<10;i++) { x.push_back(i);}
    for(auto v:x) {cout << v <<" ";};
    // 1 2 3 4 5 6 7 8 9
};
```

C++14

- Minor improvements over C++11 and bug fixes
- Extension of ”auto” data type to all functions (not just lambda)
- Digit separators: ’ character, Ex: `auto fpnum = 0.113'343`
- Template for variables as well
- Ex: `template<typename T> constexpr T pi = T(3.14159265358979323846264643L);`

Other tools for Scientific Programming:

- Python + Numpy - Simple, interpreted (no compilation)
- Java - general purpose, portable, no pointers!
- Mathematica - powerful computation
- Maple - extensive analytics
- R - majorly used in statistics
- Note: Matlab now has OO features!

EL2310

- covered basics of programming,
- started with MATLAB, continued with C and finished with C++.

MATLAB - What you should have learned:

- Be comfortable working with MATLAB
- Preparing scripts and functions using basic elements of programming (loops, branching, ...)
- Taking advantage of built-in functions (load data, plot data), especially the visualization capabilities.
- Translating a scientific problem into MATLAB code.

- Interpret MATLAB code when you see it.
- Know when (and how) to use MATLAB when required.

MATLAB - What you should have learned:

- Be comfortable working with MATLAB
- Preparing scripts and functions using basic elements of programming (loops, branching, ...)
- Taking advantage of built-in functions (load data, plot data), especially the visualization capabilities.
- Translating a scientific problem into MATLAB code.
- Interpret MATLAB code when you see it.
- Know when (and how) to use MATLAB when required.

MATLAB - What you should have learned:

- Be comfortable working with MATLAB
- Preparing scripts and functions using basic elements of programming (loops, branching, ...)
- Taking advantage of built-in functions (load data, plot data), especially the visualization capabilities.
- Translating a scientific problem into MATLAB code.

- Interpret MATLAB code when you see it.
- Know when (and how) to use MATLAB when required.

MATLAB - What you should have learned:

- Be comfortable working with MATLAB
- Preparing scripts and functions using basic elements of programming (loops, branching, ...)
- Taking advantage of built-in functions (load data, plot data), especially the visualization capabilities.
- Translating a scientific problem into MATLAB code.

- Interpret MATLAB code when you see it.
- Know when (and how) to use MATLAB when required.

C - What you should have learned:

- Working with C: how to write, compile, link, execute.
- Declaring and initializing variables, basic data types, pointers(!), memory allocation(!)...
- Preparing programs using basic elements of programming (loops, branching, ...)
- Taking advantage of built-in libraries (e.g. for printing data)
- Interpret C code when you see it.
- Know when (and how) to use C for the upcoming scientific problems.

C - What you should have learned:

- Working with C: how to write, compile, link, execute.
- Declaring and initializing variables, basic data types, pointers(!), memory allocation(!)...
- Preparing programs using basic elements of programming (loops, branching, ...)
- Taking advantage of built-in libraries (e.g. for printing data)
- Interpret C code when you see it.
- Know when (and how) to use C for the upcoming scientific problems.

C - What you should have learned:

- Working with C: how to write, compile, link, execute.
- Declaring and initializing variables, basic data types, pointers(!), memory allocation(!)...
- Preparing programs using basic elements of programming (loops, branching, ...)
- Taking advantage of built-in libraries (e.g. for printing data)
- Interpret C code when you see it.
- Know when (and how) to use C for the upcoming scientific problems.

C - What you should have learned:

- Working with C: how to write, compile, link, execute.
- Declaring and initializing variables, basic data types, pointers(!), memory allocation(!)...
- Preparing programs using basic elements of programming (loops, branching, ...)
- Taking advantage of built-in libraries (e.g. for printing data)
- Interpret C code when you see it.
- Know when (and how) to use C for the upcoming scientific problems.

C++ - What you should have learned:

- Everything in C can be used in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Interpret C++ code when you see it.
- Know when (and how) to use C++ for the upcoming scientific problems.

C++ - What you should have learned:

- Everything in C can be used in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Interpret C++ code when you see it.
- Know when (and how) to use C++ for the upcoming scientific problems.

C++ - What you should have learned:

- Everything in C can be used in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Interpret C++ code when you see it.
- Know when (and how) to use C++ for the upcoming scientific problems.

C++ - What you should have learned:

- Everything in C can be used in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Interpret C++ code when you see it.
- Know when (and how) to use C++ for the upcoming scientific problems.

C++ - What you should have learned:

- Everything in C can be used in C++ and what C++ has to offer more (or in a different way) ...
- especially, the Object Oriented Programming Paradigm(!): Encapsulation, Polymorphism, Inheritance.
- Declaring classes and instantiating objects, accessing members, ...
- Understanding of 'conceptual programming', i.e. hiding of functions, declaring of static, const, virtual ...
- Interpret C++ code when you see it.
- Know when (and how) to use C++ for the upcoming scientific problems.

In general:

- have a good understanding of basic concepts in programming.
- get to know MATLAB so that you can use it in other courses.
- solve problems and implement algorithms in C and C++.
- be able to read and understand existing code written in C or C++.
- know the importance of writing reusable code that can shared among the community.

In general:

- have a good understanding of basic concepts in programming.
- get to know MATLAB so that you can use it in other courses.
- solve problems and implement algorithms in C and C++.
- be able to read and understand existing code written in C or C++.
- know the importance of writing reusable code that can shared among the community.

In general:

- have a good understanding of basic concepts in programming.
- get to know MATLAB so that you can use it in other courses.
- solve problems and implement algorithms in C and C++.
- be able to read and understand existing code written in C or C++.
- know the importance of writing reusable code that can shared among the community.

In general:

- have a good understanding of basic concepts in programming.
- get to know MATLAB so that you can use it in other courses.
- solve problems and implement algorithms in C and C++.
- be able to read and understand existing code written in C or C++.
- know the importance of writing reusable code that can shared among the community.

In general:

- have a good understanding of basic concepts in programming.
- get to know MATLAB so that you can use it in other courses.
- solve problems and implement algorithms in C and C++.
- be able to read and understand existing code written in C or C++.
- know the importance of writing reusable code that can shared among the community.

Summary

We have learnt a tool but we have not covered everything especially Computer Science:

- Algorithms: *Sorting*, *Mapping*, ...
- Data structures: *Trees*, *Graphs*, ...
- Complexity
- Discrete Math
- ...

How to continue?

- The aim of this course was to get you started
- Hundreds of References and Books - to learn more and have a quick lookup for more specific things you need.
- Some more concentrated programming courses at KTH:
  - **DD2387** Programsystemkonstruktion med C++ 6,0 hp
  - **DD2456** Avancerade objektorienterade system 7,5 hp (assumes OOP knowledge)

- Experience(!) - your own project.
- Demoscene

Still to do:

- Your Evaluation
  - C++-project still to go
  - The course is only pass or fail
- Our Evaluation
  - Will be available through BILDA after the C++ project
  - For collecting feedback and opinions about the course.

Still to do:

- Your Evaluation
  - C++-project still to go
  - The course is only pass or fail
- Our Evaluation
  - Will be available through BILDA after the C++ project
  - For collecting feedback and opinions about the course.

Getting involved

- RPL(CVAP) `http://www.nada.kth.se/cvap/` does research in,
  - Computer Vision
  - Robotics
  - **Machine Learning** and **AI**
- If you are interested,
  - Research interaction
  - **2D5348** Individual course in Computer Science
  - Thesis work

  Programming is a **tool** in our work but **NOT** a focus or motivation