CS 590: Algorithm

Week 1: Introduction

In Suk Jang
Department of Computer Science
Stevens Institute of Technology

Introduction

- About me
- Course Information
- Course Description
- Important Points
- Technology Requirements
- Course Requirements
- Schedule
- Grading Policy
- Overview of Algorithm
- Overview of C++

About Me

Prof. In Suk Jang

- Contact: ijang@stevens.edu
  - Please include the section in the subject line, e.g., CS590-A: xxxxx
- Teaching Courses: Algorithm, Machine Learning
- Research Interests: Black Holes, Machine Learning

Course Information

Course Title: CS 590 Algorithms
Course Period: A - Tuesday 6:30 PM – 9:00 PM
Office Hours:

- Wednesday 10– 11 AM
- Link: https://stevens.zoom.us/j/5516841287

Resources:

- Textbook: Introduction to Algorithms, 3rd Edition, MIT press – CLRS
- Lecture Notes, C++ documents are available in Canvas.

Course Description

- This is a course on more complex data structures, and **algorithm design and analysis**, using one or more modern imperative language(s), **as chosen by the instructor**.

- Topics include: advanced and/or balanced search trees; further asymptotic complexity analysis; standard algorithm design techniques; graph algorithms; complex sort algorithms; and other "classic" algorithms that serve as examples of design techniques.

Teaching Assistants

- Jain, Nirav (CS590A):
  - Email: njain13@stevens.edu
  - Office Hours: TBA
- Parikh, Dhruv:
  - Email: dparikh8@stevens.edu
  - Office Hours: TBA

Course Description

After successful completion of this course, students will be able to:

- **Complexity** – Explain the meaning of big-O, Theta, and Omega notations. Calculate
- the asymptotic running time of standard algorithms, and use it to compare efficiency.
- **Master Theorem** – Use the Master Theorem to prove asymptotic assumptions
- **Sorting** - Compare and analyze basic and advanced sorting algorithms.
- **Trees** - Implement advanced search trees such as Binary Search Trees, and Red-Black Trees.
- **Graphs** - Implement standard algorithms using graphs and weighted graphs in C++ (e.g., DFS, BFS, MST, topological sort).
- **Shortest Paths –** Implement standard algorithms to solve the shortest path finding problem. (Dijkstra, Bellman-Ford, Floyd-Warshall)
- **Algorithmic Design -** Apply standard algorithm design techniques such as the greedy technique, dynamic programming, hashing, and space/time trade-offs.

Important Points

- Communication
- Office hours
- Questions

TECHNOLOGY REQUIREMENTS

Baseline technical skills necessary for online courses

- Basic computer and web-browsing skills
- Navigating Canvas

Required Equipment

- Computer: current Mac (OS X) or PC (Windows 7+) with high-speed internet connection

Required Software

- Microsoft Word

Integrated Developing Environment (IDE)

- Dev C++
- Visual Studio
- Eclipse
- Check this clip if you did not install yet: http://www.youtube.com/watch?v=Y8So6Hh-ZSs
- Virtual Box: https://www.virtualbox.org/wiki/Downloads (See Canvas for instruction)
- Online Compiler: https://www.onlinegdb.com/

GRADING PROCEDURE

Assignments	60%
Midterm (3/7)	15%
Final Exam (5/2)	25%

- Grades will be based on:

Any complaint regarding a grade must be presented **no later than 7 weekdays** following the publication of grades of respective assignments. Penalties for specific mistakes that are applied to exams, assignments and quizzes are equal for all students in the course. If you contact me to negotiate these penalties, I will not respond.

- Late Policy

Late assignment (even by 2 seconds) will be given a -25% decrease penalty per day, for the first 2 days after the deadline. So, if you send an assignment 1 second late, you will receive 75% of your grade for the assignment. If you send it, 24 hours, and 1 second late, you will receive 50% of your grade for the assignment etc. After 48hrs from the deadline there will be a -90% decrease penalty, so you will receive 10% of your grade.

COURSE REQUIREMENTS

- **Homework:** The programming assignments will be done individually. No collaboration is allowed between students. No code from online resources is allowed to be used besides the code that I will share with you. Any sign of collaboration will result in a 0 and being reported to the Honor Board. Programming assignments might be tested for similarity using the MOSS, or similar software, and any sign of collaboration will be reported to the HONOR board. Students who are caught collaborating for a second time, they will receive a failing grade (F) in the course.
  - Only C++? You can do the first assignment with other language if you are new to C++.
    - But will be penalized for not using C++ from Assignment 2. (10% for each assignment)
  - Most of assignments are implementations of algorithms we discuss in class.
    - But you will implement from the stretch but will translate pseudocode to C++.
  - There will be some mathematical deriving questions.
  - All assignment results must be written and submit together with codes.
  - Each assignment will be available on Friday 12 AM.

- **Exams:** The midterm exam will be given in week 7. The final exam will be given on the last day of the semester. An announcement will be posted with more details on week 13. The exam will be a paper exam in the classroom.

Introduction to Algorithm

- Algorithm
- Data Structure
- Efficiency

Introduction to Algorithm

What are Algorithms?

- Any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values as output.
- Sequences of computational steps that transform the input into the output
- Tools for solving a well-specified computational problem (input/output relationship)
- instance of a problem consists of the input needed to compute a solution to the problem.
- correct algorithm solves the given computational problem.

Example Sorting Problem (We will look into this detail in Week 3):

- Input: $\langle 31, 41, 59, 26, 41, 58 \rangle$
- Output: $\langle 26, 31, 41, 41, 58, 59 \rangle$

Data Structure

Data Structure

- a way to store and organize data in order to facilitate access and modification.
- no single data structure optimal for all purposes.
- usually optimized for a specific problem setting.
- important to know the strength and limitations of several of them.

Efficiency

Computing time and memory are bounded resources.

Efficiency:
• Different algorithms that solve the same problem often differ in their efficiency.
• More significant than differences due to hardware (CPU, memory, disks, ...) and software (OS, programming language, compiler, ...).

Algorithms and other technologies

Consider algorithms like computer hardware, as a technology

In this course, we care most about *asymptotic performance*
– How does the algorithm behave as the problem size gets very large?
• Running time
• Memory/storage requirements
• Bandwidth/power requirements/logic gates/etc.

Introduction to C++

- Introduction
- Evolution
- General Structure
- A Simple C++ Program
- About Learning

Introduction to C++

- The C++ programming language is a very powerful general – purpose programming language that *supports procedural programming as well as object* – oriented programming. It incorporates all the ingredients required for building software for large and complex problems.

- The C++ language is treated as *super set of C language* because the developer of C++ language have retained all features of C, enhanced some of the existing features, and incorporated new features to support for object – oriented programming.

- The importance of C++ can well be judged from the following statement:
- "Object – Oriented Technology is regarded as the ultimate paradigm for the modeling of information, be that information data or logic. The C++ has by now shown to fulfill this goal."

Evolution of C++ Language

- The C++ programming language was developed by **Bjarne Stroustrup** at *AT&T BELL LABORATORIES*, NEW JERSEY, USA, in the early 1980's.

- He found that as the problem size and complexity grows, it becomes extremely difficult to manage it using most of procedural languages, even with C language.

GENERAL STRUCTURE OF A C++ PROGRAM

Declarations: data types, function signatures, classes –

- Allows the compiler to check for type safety, correct syntax
- Usually kept in "header" (.h) files
- Included as needed by other files (to keep compiler happy)

```
class Simple {
public:
   Simple (int i);
   void print_i ();
private:
   int i_;
};
```

```
typedef unsigned int UINT32;
int usage (char * program_name);

struct Point2D {
  double x_;
  double y_;
};
```

Definitions: static variable initialization, function implementation

- The part that turns into an executable program
- Usually kept in "source" (.cpp) files

```
void Simple::print_i ()
{
 cout<< "i_is "<<i_<<endl;
}
```

Directives: tell compiler (or pre-compiler) to do something

GENERAL STRUCTURE OF A C++ PROGRAM

Section 1: Comments

- It contains the description about the program.

Section 2: Preprocessor Directives

- The frequently used **preprocessor** directives are include and define. These directives tell the preprocessor how to prepare the program for compilation. The include directive tells which header files are to be included in the program and the define directive is usually used to associate an identifier with a literal that is to be used at many places in the program.

Section 3: Global Declarations

- These declarations usually include the declaration of the data items which are to be shared between many functions in the program. It also include the decorations of functions.

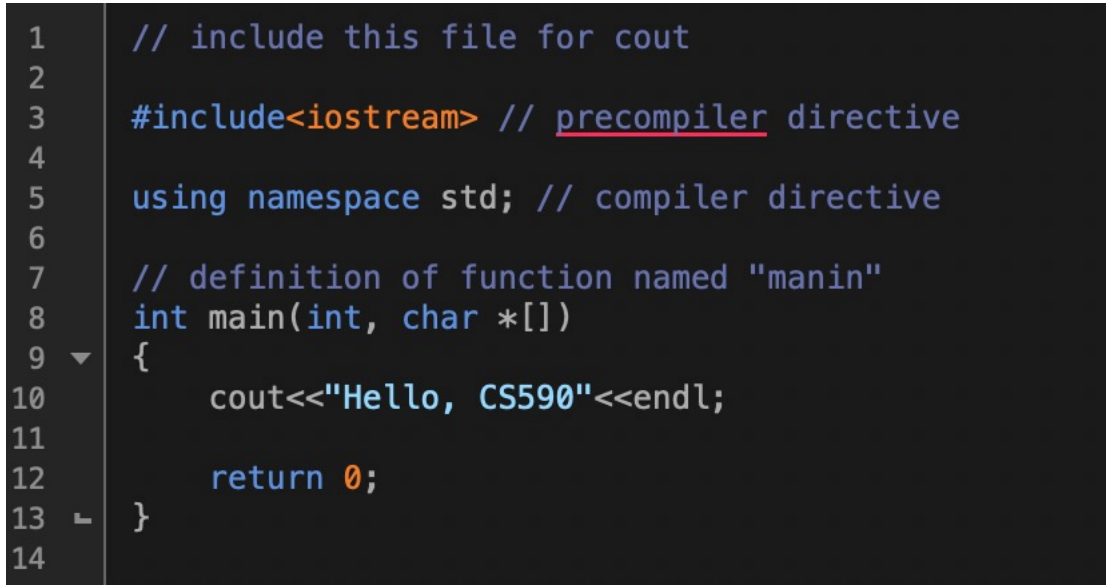
Section 4: Main function

- The execution of the program always begins with the execution of the main function. The main function can call any number of other functions, and those called function can further call other functions.

Section 5: Other functions as required

```
1   // include this file for cout
2
3   #include<iostream> // precompiler directive
4
5   using namespace std; // compiler directive
6
7   // definition of function named "manin"
8   int main(int, char *[])
9   {
10      cout<<"Hello, CS590"<<endl;
11
12      return 0;
13  }
14
```

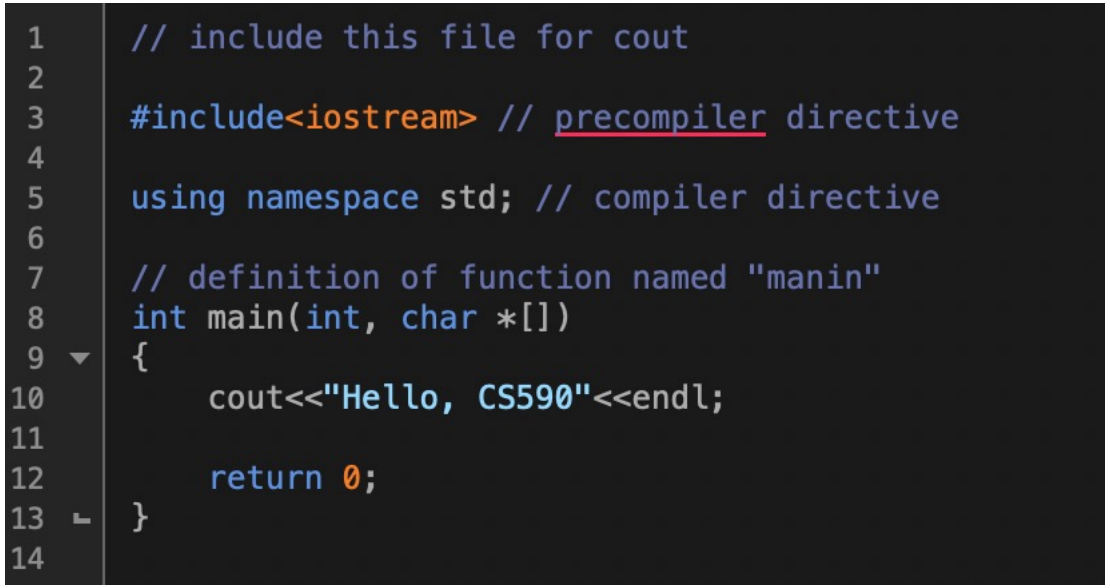
A Simple C++ Program



```
1   // include this file for cout
2
3   #include<iostream> // precompiler directive
4
5   using namespace std; // compiler directive
6
7   // definition of function named "manin"
8   int main(int, char *[])
9   {
10      cout<<"Hello, CS590"<<endl;
11
12      return 0;
13  }
14
```

What is #include<iostream> ?

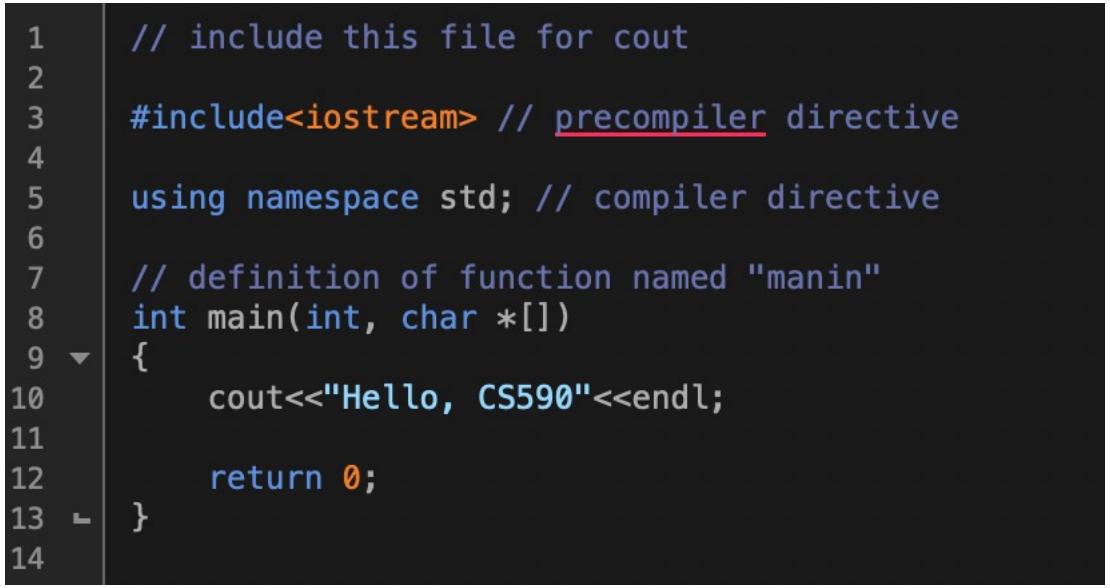
- #include tells the precompiler to include a file
- Usually, we include header files
  - Contain *declarations* of structs, classes, functions
- Sometimes we include template *definitions*
  - Varies from compiler to compiler
  - Advanced topic, out of scope for this course, but feel free to look it up!
- <iostream> is the C++ label for a standard header file for input and output streams



```
1   // include this file for cout
2
3   #include<iostream> // precompiler directive
4
5   using namespace std; // compiler directive
6
7   // definition of function named "manin"
8   int main(int, char *[])
9   {
10      cout<<"Hello, CS590"<<endl;
11
12      return 0;
13  }
14
```

What is using namespace std; ?

- The **using** directive tells the compiler to include code from libraries that have separate *namespaces*
  - Similar idea to "packages" in other languages
- C++ provides a namespace for its standard library
  - Called the "standard name space" (written as std)
  - cout, cin, and cerr standard iostreams, and much more
- Namespaces reduce collisions between symbols
  - Rely on the **::**scoping operator to match symbols to them
  - If another library with namespace mylib defined **cout** we could say **std::cout vs. mylib::cout**
- Can also apply **using** more selectively:
  - *E.g.,*just **usingstd::cout**



```
1   // include this file for cout
2
3   #include<iostream> // precompiler directive
4
5   using namespace std; // compiler directive
6
7   // definition of function named "manin"
8   int main(int, char *[])
9   {
10      cout<<"Hello, CS590"<<endl;
11
12      return 0;
13  }
14
```

What is int main (int, char*[]) { ... } ?

- *Defines* the main function of any C++ program
- Who calls main?
  - The runtime environment, specifically a function often called something like **crt0** or **crtexe**
- What about the stuff in parentheses?
  - A list of types of the input arguments to function main
  - With the function name, makes up its *signature*
  - Since this version of main ignores any inputs, we leave off names of the input variables, and only give their types
- What about the stuff in braces?
  – It's the *body* of function main, its *definition*

```
1   // include this file for cout
2
3   #include<iostream> // precompiler directive
4
5   using namespace std; // compiler directive
6
7   // definition of function named "manin"
8   int main(int, char *[])
9   {
10      cout<<"Hello, CS590"<<endl;
11
12      return 0;
13  }
14
```

What's cout << "Hello, CS590" << endl; ?

- Uses the standard output iostream, named **cout**
  - For standard input, use **cin**
  - For standard error, use **cerr**
- << is an operator for *inserting* into the stream
  - A member **operator** of the ostream class
  - Returns a *reference* to stream on which its called
  - Can be applied repeatedly to references left-to-right
- **"hello, cs590"** is a C-style string
  - A14-position character *array* terminated by **'\0'**
- **endl** is an iostream manipulator
  - Ends the line, by inserting end-of-line character(s)
  - Also *flushes* the stream

```cpp
// include this file for cout

#include<iostream> // precompiler directive

using namespace std; // compiler directive

// definition of function named "manin"
int main(int, char *[])
{
    cout<<"Hello, CS590"<<endl;

    return 0;
}

```

What about return 0;?

- The main function should return an integer
  - By convention it should return 0 for success
  - And a non-zero value to indicate failure
- The program should not exit any other way
  - Letting an exception propagate uncaught
  - Dividing by zero
  - Dereferencing a null pointer
  - Accessing memory not owned by the program
    - Indexing an array “out of range” can do this
    - Dereferencing a “stray” pointer can do this

About Learning

A key goal is to expand & refine your mental models

- For C++ mainly, but also for algorithms in general
- Notice and try out new ideas, share them, discuss them
- Challenge your understanding in as many ways as you can

If you don't remember every detail at first, that's ok

- We'll revisit concepts and techniques from different angles
- Try to refresh your memory early and often
- Apply what you learn, early and often, towards mastery

We'll work together to build understanding in stages

- First as a consumer of an approach (can you use it?)
- Then understanding it thoroughly (*when* can you use it?)
- Then as a contributor to the approach (can you *expand* it?)