Instructor

- Rana Forsati
- Email: forsati@msu.edu

- Office Hour
  - Wed 4pm-5pm, 3335 EB
  - By appointment
Teaching Assistant

- Iman Barjasteh
- Email: barjaste@msu.edu

Office Hour
- Tu & Th 4:50pm-5:50pm, 3353 EB
Course Structure

• 2 Lectures (50 minutes)
  Monday+Wednesday 3:00 PM – 3:50 PM

• 1 Lab (2 hours)
  Section 1: Tu 3:00pm-4:50pm (3353 Engineering Building)

  Section 2: Th 3:00pm-4:50pm (3353 Engineering Building)
Course Objectives

- Learn syntax and semantics of C
- Learn general programming concepts
- Learn problem solving
- Design, implement and test C programs

*Previous programming experience is not required*
Course Material

  - by K. N. King
  - ISBN: 978-0393979503

- Lecture slides on the course website:
  - [www.cse.msu.edu/~cse220](http://www.cse.msu.edu/~cse220)
  - posted on the day of the lecture
Course’s web site:

http://www.cse.msu.edu/~cse220
Course’s web site:

http://www.cse.msu.edu/~cse220

Piazza:

https://piazza.com/msu/fall2015/cse220/home

- Please do not post solutions on Piazza and only discussions
- Please be active on Piazza and discuss the material
- The announcement will be made here
- Please register in the Piazza.
Course Structure

- **13 lab exercises**
  - Attendance is mandatory and important
- **10 written homework assignments**
- **3 programming projects**
  - Working on “real-world” problems
- **1 midterm exam**
- **1 final exam**
Homework Assignments

- Due weekly
- Must be submitted by the due date
- Can use CSE labs, DECS labs, remote access
Rules about assignments

• Collaborative study is encouraged
  – Feel free to discuss ideas, design, etc
  – But not code!

• Do not share your code with other students

• Protect your files on public computers
# Grading

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>20%</td>
</tr>
<tr>
<td>Homework</td>
<td>15%</td>
</tr>
<tr>
<td>Project</td>
<td>30%</td>
</tr>
<tr>
<td>Midterm Exam</td>
<td>15%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>20%</td>
</tr>
<tr>
<td>Optional Extra Credit</td>
<td>10%</td>
</tr>
</tbody>
</table>
Grade Scale

4.0  90% of points available
3.5  85% of points available
3.0  80% of points available
2.5  75% of points available
2.0  70% of points available
1.5  65% of points available
1.0  60% of points available

(The instructor reserves the right to adjust the scale for course grades, if necessary)
Important Dates

• Midterm Exam: Wed, 10/14/2015
• Final Exam: As posted by registrar’s schedule.

• Drop Dates
  – Full refund: 8:00pm, September 28
  – No grade reported: 8:00pm, October 21
Excuses for ...

- Make-ups for exams and extend assignments’ due date
  - documented illness
  - personal emergency
  - Job interview (only applicable for this class)
    - Must notify before the fact and provide appropriate documentation
    - Same would apply if you are attending a conference based on recommendation from a professor in our department.

- Student’s explanation: must be acceptable
- MSU’s policy on Integrity of Scholarship and Grades
Announcements for class

- Most announcements will be made on Piazza or course webpage.

- I will be assuming that you are aware of all announcements made in class.
CSE Computing System

• Need CSE account
  – Students enrolled in CSE course get a CSE account
  – Activate during first lab

• CSE labs
  – Linux workstations: 2240, 3353 EB
  – Widows workstations: 3203, 2220, 3353 EB

• DECS labs

• Remote access

• CSE Facilities: http://www.cse.msu.edu/?Pg=79
Outline

- Computer System
- Programming Languages and C
- Programming Lifecycle
- Basic Linux Commands
Computer Systems
Hardware and Software

Figure 2.1 Computer Hardware and Software Infrastructure
Hardware

Input Device  \rightarrow  CPU  \rightarrow  Output Device

\begin{align*}
\text{ALU} & \quad \uparrow \quad \text{Main memory (RAM)} \\
\text{CU} & & \downarrow \\
\end{align*}

\begin{align*}
\text{Secondary storage} & \quad \rightarrow \\
\end{align*}
Software

Programs – instructions that tell the computer what to do

Categories

- **Application software** - enables you to solve specific problems or perform specific tasks.

- **System software** - handles tasks specific to technology management and coordinates the interaction of all technology devices.

- **Utility software** - provides additional functionality to your operating system software.
System Software

- **Operating System:**
  - UNIX / Linux
  - Windows
  - Mac OS
  - Palm OS
  - Android

- **Language Translators:**
  - C, C++, Basic, Java, etc

- **Device Drivers**
  - driver of hard disk
Program Execution

Compiler & Linker

High Level Program

Machine Language
Program Execution

High Level Program

Compiler & Linker

Machine Language

Operating System
Programming Languages and C!
A programming language is a formal constructed language designed to communicate instructions to a computer.
Programming Languages

A **programming language** is a formal constructed language designed to communicate instructions to a computer.

- Machine Language
- Assembly Language
- High level Languages
Machine Language

- Machine language is the “natural language” of a computer and as such is defined by its hardware design.

- The fundamental language of the computer’s processor (CPU), also called Low Level Language.

- All programs are converted into machine language before they can be executed.

- Consists of combination of 0’s and 1’s that represent high and low electrical voltage.

- Machine languages are machine dependent (i.e., a particular machine language can be used on only one type of computer).
Machine Level Programming

• Example: suppose we want the computer to add two numbers, and if the preliminary result is less than 10, then add 10 to the result

• The instructions that the CPU carries out might be:
  [INSTR1] Load into ALU the number from mem location 15
  [INSTR2] Load into ALU the number from mem location 7
  [INSTR3] Add the two numbers in the ALU
  [INSTR4] If result is bigger than 10 jump to [INSTR6]
  [INSTR5] Add 10 to the number in the ALU
  [INSTR6] Store the result from ALU into mem location 3

• The processors instruction set: all basic operations that can be carried out by a certain type of processor
The instructions and operands are represented in binary notation (sequences of 0s and 1s).
- Why binary? Because computer hardware relies on electric/electronic circuits that have/can switch between 2 states
  - *bit* (binary digit)
  - *Byte*: 8 bits

The program carried out by the CPU, on a hypothetical processor type, could be:

```
1010 1111
1011 0111
0111
...
```

This way had to be programmed the first computers!

The job of the first programmers was to code directly in machine language and to enter their programs using switches.

These languages are cumbersome for humans.
Assembly Language

- A low level language that is similar to machine language.
- Uses **symbolic names** to represent the machine operation code.

  **Example**: a hypothetical assembly language program sequence:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>1111</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>0111</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>LD2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>JGE</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assembly Language

- Translation of assembly language into machine language: in the beginning done manually, later done by a special computer program – the assembler

- Disadvantages: Low-level language:
  - programmer must learn the instruction set of the particular processor
  - Program must be rewritten in order to run on a different processor type – program is not portable
High-level Languages

- Computer (programming) languages that are easier to learn.
- To speed the programming process, high-level languages were developed.
- Uses English like statements.
- Examples are C, C++, Java, C#, F#, Python, etc.
Why C?

- **C** is *stable* (the language doesn’t change much).
- **C** is *quick running*.

- **Powerful language**
  - Core of many software/hardware systems
  - Windows, Mac, Linux are in C (at least kernel)

- **Low level language**
  - Suitable for system programming

- **Small language**
  - Easy to learn

- **Base** to other languages
  - C++, Java, C#, Python
Strengths of C

• Efficiency
  – Run quickly and with limited memory

• Power
  – Accomplish a lot with few lines of code

• Standard Library
  – input/output, string handling, etc

• Users
  – Long history & huge community
Weaknesses

• Error prone
  – Flexibility allows programmers to make mistakes without warnings
  – Hard to debug, hard to maintain

• Lacks new and useful features
  – Object-oriented design, exceptions, ...
Programming in C
Development Lifecycle

- Write/Edit a program
- Compile
- Execute & Test

(Image from skillcrush.com)
C Development Environment

Creating a Program

Editor

Preprocessor

Compiler

Linker

Phase 1:
Programmer creates program in the editor and stores it on disk.

Phase 2:
Preprocessor program processes the code.

Phase 3:
Compiler creates object code and stores it on disk.

Phase 4:
Linker links the object code with the libraries, creates an executable file and stores it on disk.

translates the C program into machine-language code.
Execution Environment

executes the program one instruction at a time
Compiling & Linking

- The source code is **readable** for human, but **not executable** for computer.
Tips

• Think and design before you start coding
• Experiment
• Start early
• Ask questions
Tips

The best way to be a programmer is to practice coding over and over!

How do you get to be a great musician?

It helps to know the theory, and to understand the mechanics of your instrument. It helps to have talent.

But ultimately, greatness comes from practicing; applying the theory over and over again, using feedback to get better every time.

image from [http://codekata.com/](http://codekata.com/)
Some Programmer Jargon

- **Some words that will be used a lot:**
  - **Source code:** The stuff you type into the computer. The program you are writing.
  - **Compile (build):** Taking source code and making a program that the computer can understand.
  - **Executable:** The compiled program that the computer can run.
  - **Language:** (Special sense) The core part of C central to writing C code.
  - **Library:** Added functions for C programming which are bolted on to do certain tasks.
  - **Header file:** Files ending in .h which are included at the start of source code.
CSE 220

Programming in C

Fall 2015

Rana Forsati
Announcement

• HW #1 is Posted on the course web page

• Due date of HW#1 is on Sunday (Sep. 11)
Compiling Source Code

```c
#include <stdio.h>
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

```
gcc HelloWorld.c -o HelloWorld.exe
```

"-o": specify the name of the output file
Compiling

• The source code is **readable** for human, but **not executable** for computer

  A compiler acts as a translator, transforming human-oriented programming languages into computer-oriented machine languages.

  Ignore **machine-dependent** details for programmer
Compiling Source Code

(Image modified from www.aboutdebian.com)
Compiling Source Code

• Compilation
  • Checks if you followed the rules of C
  • If no errors => creates an executable
  • If errors found => returns list of errors [compile error]

• Executable
  • The “program file” you run, e.g., “.exe” files in Windows OS
  • Generated only when all compilation errors are fixed

![Diagram of compiling source code]
GNU Compiler

- `gcc myProgram.c`
  - Generates an executable called `a.out` in the same directory

- `gcc myProgram.c -o myExecutable`
  - Generates executable `myExecutable`
  - `-o` means to specify a file name for gcc's output (i.e., the executable file)
Execution

- gcc -o myExecutable myProgram.c

- To execute myExecutable
  - ./myExecutable
IDE

Writing code, compiling, and linking can be done separately! OR, can be integrated in an Integrated Development Environment

- Editor
- Compiler
- Debugger

- MS Visual C++
- Xcode
- Eclipse for Java
Operating System

Editors: A number of editors are available like gedit, vi, emacs
[C program file names should end with the .c extension]

Compiler
[the command to compile and link a C program is called cc (or gcc)]

Execute
[the command to execute a compiled C program is ./ ]
Basic Linux Commands
File Management Commands

ls                List “normal” files.
ls -a             List all files.
cd dir            Change directory to dir.
pwd              Print working directory
rm file           Remove file.
rm -r dir         Remove dir and all subdirs.
mv file1 file2    Rename file1 to file2.
mv dir1 dir2      Rename dir1 to dir2.
cp file1 file2    Copy file1 to file2.
mkdir dir         Make a directory dir.
rmdir dir         Remove the directory dir

more file         Show contents of file.
head -n file      Show first n lines of file.
tail -n file      Show last n lines of file.
grep pattern file Search file for pattern.
cat file1 file2 > file Append file2 to file1 and save as file
**Linux Shell Commands (I)**

**pwd**: print working directory

```bash
$ pwd
/home/me
```

**ls**: list files and directories

```bash
$ ls
Desktop  Xrootenv.0  linuxcmd
GNUstep  bin       nedit.rpm
GUILG00.GE hitni123.jpg nsmap
```
Linux Shell Commands (II)

**cp**: copy files and directories

```
[me@linuxbox me]$ cp -u *.html destination
```

**mv**: move or rename files and directories

```
[me@linuxbox me]$ mv file... directory
```

**mkdir**: create directories

```
[me@linuxbox me]$ mkdir directory...
```