

Introduction to CSE 220 and C

CSE 220: Systems Programming

Ethan Blanton

Department of Computer Science and Engineering

University at Buffalo



Welcome to CSE 220

My name is Ethan Blanton.

Contacting me:

Email: `eblanton@buffalo.edu`

Office: Davis 334

Office Hours: Monday 10:00-11:00 (Davis 334)
Thursday 13:00-15:00 ([Zoom](#))
or by appointment

The syllabus is on the course web page¹ and UBlearns.

So are these — and all other — slides!

¹<https://www.cse.buffalo.edu/~eblanton/course/cse220/>

Classroom Guidelines

UB has specific guidelines for classroom safety this semester.

Please follow them, and [don't make me nag](#).



Systems Programming

This course is concerned with **systems programming**.

You will learn:

- More about the structure and properties of **computer systems**
- How **architecture** affects **programs**
- How to effectively write **efficient** and **correct** programs
- The **C programming language** and **POSIX API**

Our goal is to **improve your intuitions** about how computers work.

Programming in Context

Programming doesn't occur in a vacuum.

Computer systems **have greatly influenced** our:

- Programming languages
- Development tools
- Preferred algorithms

This course will help you **understand that context**.

Course Difficulty

This course is **NOT** a “weed-out” course!

It is, however, **quite difficult**.

It is difficult because:

- The material is precise and unforgiving.
- 2xx courses provide **less guidance** than 1xx courses.
- Success requires careful time management.

You will get out of this course **what you put into it**.

Expectation

For this course, I expect that you:

- Will be **respectful** to me, TAs, classmates
- Attend **every lecture**
- Attend **every lab**
- **Adhere strictly** to the academic integrity policy
- Will seek assistance **early** if necessary
- Meet prereqs; among other things:
 - Have some experience programming
 - Understand linked lists and object references

Most of all, **behave as adults** and strive to **maximize** your and your classmates' **learning experience** in this course.

Attendance

Lecture attendance **is mandatory**.

- I will not repeat lectures.
- Recitations will not repeat lectures.
- You are **expected to catch up on your own** if you skip.

Lab attendance **is mandatory**.

- Lab attendance **will be tracked**.
- You will **practice** what you learn in class.
- You will be **frequently tested** on your understanding.

Ways to Fail this Class

- Miss lectures or labs
- Start assignments at the last minute
- Not visiting office hours
- Not asking questions on Piazza
- Don't commit to git
- Don't submit to Autograder early
- Cheat

Readings

Most lectures will have both **required** and **optional** readings.

Readings will appear at the end of the slides.

You must read the required readings **even if I do not mention them in class**.

You may wish to read the optional readings:

- to expand your understanding of related topics
- to help you understand the required material

Assistance

Our primary forum for assistance will be [Piazza](#).

You should have been added to our course Piazza.

Please consult existing postings before asking a question!

- However, if you're not sure your question is answered, ask!
- If you wish to include code, solutions, *etc.*, send a private message to the instructors!

I will post important course announcements and materials to Piazza.

Programming Projects

A significant portion of your course grade will be projects.

- These are **individual projects**.
- Projects will be written in C.

Projects must run on the **course VM image**.

You should download and configure this image in a VM.
You will receive a VMware license for this purpose.²

²Apple Silicon users, we're working on it!

Project Assistance

Your TAs will be your primary source of help for projects.

To get the most out of your TAs, **do**:

- try the obvious things first,
- create minimal examples to show problems, and
- **consult the documentation.**

To avoid wasting TA time and failing to get help, **don't**:

- ask for help before you've tried to understand the problem
- **start at the last minute.**

Programming Tools

I will talk a lot about **tools** this semester.

You will be expected to use a few tools for this course:

- The C compiler
- `make`
- The `gdb` symbolic debugger
- A programmer's editor
- Others ...

We will help you learn these tools.

Editors

I don't care what editor you use, but **it must be a programmer's editor**, and be capable of:

- Syntax highlighting
- Automatic indentation
- Brace/parenthesis/*etc.* matching
- Extensibility

Neither I nor the TAs will help you if you are not using an appropriate environment when you seek help!

My personal recommendation is **Emacs**.

Emacs is installed and configured on the VM image.

Today's Assignments

Immediately:

- Read the [Syllabus](#).
- Watch the academic integrity video on Panopto.

By **Beginning of lab this week**:

- Create a [GitHub account](#) if you don't already have one.
- Download and install the course VM.
- Watch the lab video, read the handout, and take the quiz.

By **next Friday**:

- Complete the AI quiz, Lab 01 quiz, Lab 01.

Lab

We have labs this week!
Go to your lab.

Grading

Passing this course requires **two extra conditions**:

- Completion of the AI quiz with perfect score
- Completion of Lab 01 with perfect score

Failing **either of these points** means an F in 220.

Your course grade will be calculated according to the Syllabus.

Why C?

There are dozens of programming languages. Why C?

C is “high level” — but not very.

- C provides functions, structured programming, complex data types, and many other powerful abstractions
- ...yet it also exposes many architectural details

Most **operating system kernels** are written in C.

Many **runtimes** and **virtual machines** are written in C.

C influences **many other languages**.

Effective C

Effective C programming requires that you **master the machine**.

You must be aware of its **architecture** and **details of operation**.

We will be using C in Linux on x86-64.

The **dialect** of C that we will use is **C99**.³

The **compiler** that we will use is gcc.

³K&R describes ANSI C (C89), but we will discuss the differences when important.

CSE 220 and C

That said, CSE 220 is not (only) about learning C.

CSE 220 teaches you systems concepts, and you will implement them in C.

We will not cover all details of C syntax.

We will cover key ideas and particularly important syntax.

You should consult:

- The C Programming Language (K&R)
- Unix man pages
- Given code

On Precision

In this course I will attempt to be precise, but must **simplify some things**.

Usually this is because the details:

- are unnecessarily confusing, or
- require knowledge you are not expected to have.

If something here conflicts with the standard or the compiler, **the standard or compiler wins**.

I will try to mark **imprecise statements** with a pilcrow: ¶.

The Processor and Memory

The C language exposes a particular machine model.

Data is **stored in memory** at **accessible addresses**. ¶

The **CPU manipulates data** stored in memory.

Program code is executed as a series of instructions:

- Also **stored in memory**
- Though possibly **not accessible**

A Dedicated Computer

Most modern, multi-tasking OSes (including Unix) provide a particular model.

That model is that **each process has its own dedicated machine**.

Each process **appears to have**:

- A dedicated CPU
- Private, dedicated memory
- Private input and output facilities

That isn't **strictly true**, but it is **approximated by the OS**.

The OS provides mechanisms to **share resources** in this model.

Programs as Instructions

C programs⁴ are translated into **machine instructions**.

The computer **executes these instructions in order**. ¶

Instructions are things like:

- Add two numbers together
- Compare a number to zero
- Store a number to a location in memory

As we will see, **it's all numbers**.

⁴Indeed, all programs!

main()

Every C program starts with the function `main()`. ¶

```
int main() {  
    return 0;  
}
```

Every C function:

- takes zero or more parameters
- returns a single value

All arguments are **pass-by-value**, which means they are copies of whatever is passed to them.

C program statements **end with a semicolon (;)**.

Program Arguments

The `main()` function is given two parameters:

Return value Parameter list

```
int main(int argc, char *argv[])
```

Program Arguments

The `main()` function is given two parameters:

```
int main(int argc, char *argv[])  
        First parameter
```

The first is an **integer** containing the number of arguments passed to the program **on the command line**.

Program Arguments

The `main()` function is given two parameters:

```
int main(int argc, char *argv[])  
                Second parameter
```

The first is an **integer** containing the number of arguments passed to the program **on the command line**.

The second is the program arguments as an **array of strings**. (We will discuss strings and arrays more later.)

Program Arguments

The `main()` function is given two parameters:

```
int main(int argc, char *argv[])  
          Type Name
```

The first is an **integer** containing the number of arguments passed to the program **on the command line**.

The second is the program arguments as an **array of strings**. (We will discuss strings and arrays more later.)

Each parameter has a **type** and a **name**.

Aside on Slide Syntax

```
$ gcc program.c -o program
```

The \$ indicates the terminal prompt.

- Please do not type this — **you will get an error!**
- You should type everything that follows the \$

This is a good time to brush up on your Unix basics:

- Quick tutorial:

<https://www.digitalocean.com/community/tutorials/an-introduction-to-linux-basics>

- Comprehensive tutorial

<https://ryanstutorials.net/linuxtutorial/>

Compiling the Example

Assume that this code is in `trivial.c`:

```
int main() {  
    return 0;  
}
```

We can **compile** it into an **executable** as follows:⁵

```
$ gcc trivial.c
```

This will produce the file `a.out`, which is a **native binary**.

You can run the binary as follows:

```
$ ./a.out
```

⁵K&R uses `cc`, which will also work.

Developing Hello World

“Hello World” is a classic **first program** when learning a language.

We will develop a Hello World together.

Summary

- C is a **high level language** used in **systems programming**.
- **Architectural details** are important in C.
- The C/POSIX model is:
 - A **dedicated machine** for each program
 - Sequential execution of program instructions
 - Data is stored in accessible, **addressed memory**
- We explored some trivial C programs.

Remember your required readings!

Next Time ...

- More about types
- Variable declaration and usage
- C Strings
- Looping

References I

Required Readings

- [1] *Course Syllabus.*
<https://www.cse.buffalo.edu/~eblanton/course/cse220-2021-0s/materials/syllabus.pdf>.
- [2] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Second Edition. Introduction, Chapter 1. Prentice Hall, 1988.

License

Copyright 2019, 2020, 2021 Ethan Blanton, All Rights Reserved.
Copyright 2019 Karthik Dantu, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see <https://www.cse.buffalo.edu/~eblanton/>.