CSE 220: Systems Programming

Introduction to C

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 *(irrelevant, of course)*
Office Hours:	Monday 10:00-11:00
	Thursday 14:00-16:00
	or by appointment

The syllabus is available on the course web page, at https://www.cse.buffalo.edu/~eblanton/course/cse220/.

So are these — and all other — slides!

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

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.

This may be one of the hardest classes you take at UB.

Next Lecture

Starting next lecture, we will alternate materials.

We will have:

- Lecture A: Variables, Strings, and Loops (primary lecture)
- Lecture B: Setting up and using the course environment

On Friday, we will have:

- Lecture A: Getting around in Unix (probably)
- Lecture B: Conditionals and Control Flow (primary lecture)

You must watch all primary lectures.
You will have to watch some additional lectures.
I highly recommend you attend all of your lectures.

Today's Assignments

Immediately:

- Read the Syllabus.
- Watch the intro videos linked on Piazza

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 and Course Format quiz.

Lab

We have labs this week!

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 we will use is C99.[1]

The compiler we will use is `gcc`.

[1] K&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 programs[2] 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.

[2] Indeed, 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:[3]

```
$ gcc trivial.c
```

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

You can run the binary as follows:

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

[3] K&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/cse220-2020-2f/materials/syllabus.pdf.

Optional Readings

[2] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Second Edition. Introduction, Chapter 1. Prentice Hall, 1988.

License

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