CS 211 Notes

Introduction to Programming

Jaeden Bardati

September 12, 2021

1 Course Overview

September 10, 2021

1.1 Objectives

The course is intended to teach how to develop a computer program to solve a problem. C++ is a tools that will be used to develop these skills and logical thinking. These skills will be transferable to other languages.

2 Computer Organization

## 2.1 Hardware

September 11, 2021

2.1.1 Components

Modern computers are built using the **Von Neumann machine**. There are three aspects:

- **Architecture**: **I/O** (User interaction) + **Memory** (Storage) + **CPU** (*CU*: Control Unit, *ALU*: Arithmetic and Logic Unit). These are all connected by a shared bus.

- **Stored Programs**: All programs and data are stored in memory (binary).

- **Sequential Execution**: Also called the **fetch-decode-execute** cycle. Instructions are **fetched** from memory, **decoded** by the CU and then **executed** by the ALU. If there is a result, it is stored back in memory.

2.2 Memory

September 11, 2021

The memory is organized in a **hierarchy**. At the bottom of the hierarchy is the Hard Drive (in TB). At the top is the CPU. Since the hard drive is slow, when some data from the hard drive is needed, it is first loaded into **RAM (Random Access Memory)** (in GB). The RAM is still too slow for the RAM, so the data is stored in **cache** (in KB or MB). Yet still, this is not fast enough for the CPU, so **registers** (in Bytes) in the CPU itself are used to store variables.
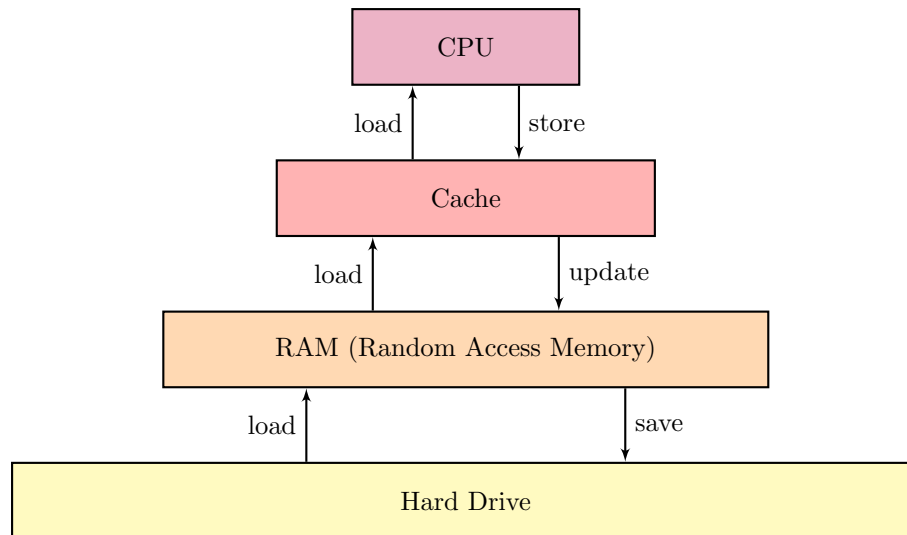
As you **go up** the hierarchy, the **speed increases**, but the **size decreases** and the **cost increases**.

2.2.1 RAM

Random access memory is organized in an array of Bytes ("words").

Words in RAM are addressed with a byte themselves (e.g. 01101101 is an address). These are typically written in hexadecimal (e.g. 6D).

Words in RAM can be data or machine code instructions. Instructions contain a binary code for each operation (for example, addition). Instructions codes are dependent on the CPU.

2.3 A Flow Chart: Program to Binary
September 12, 2021

This is a flow chart of what is done by the computer when compiling a C++ file. In blue is the Python equivalent.
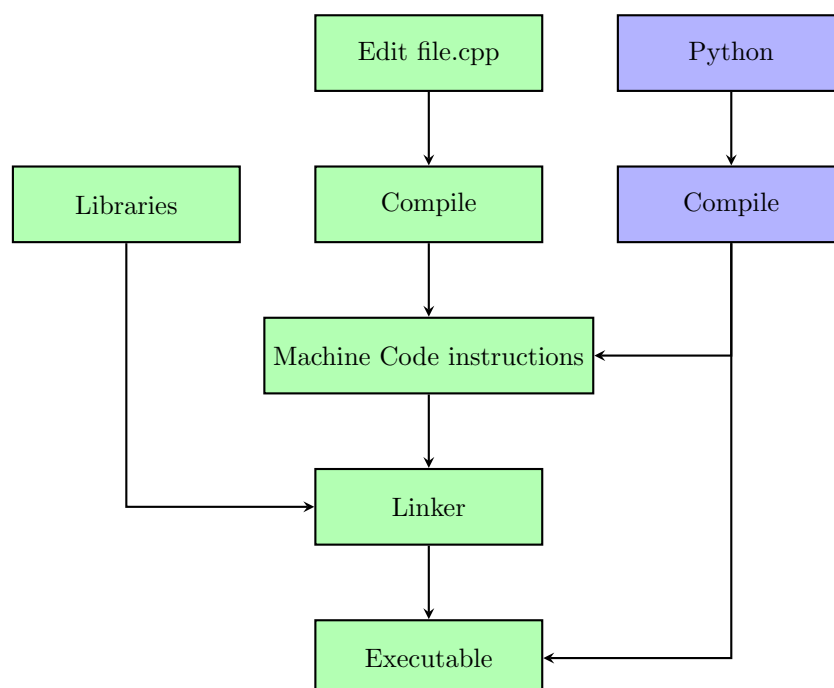
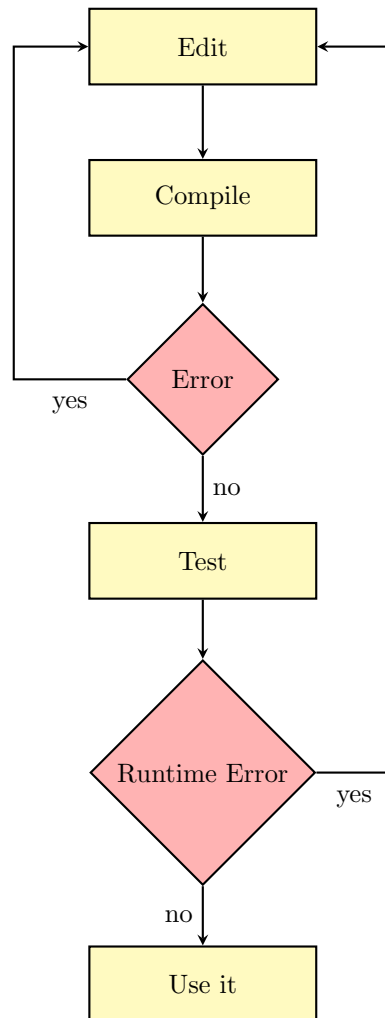Note that the bottom of the flow chart is the same for all programming languages, because in all languages, CPU-specific machine code is needed to execute code.

The process of catching errors is as according to the following flow chart.

```
                    ┌──────────────┐
          ┌────────▶│     Edit     │◀────────┐
          │         └──────────────┘         │
          │                │                 │
          │                ▼                 │
          │         ┌──────────────┐         │
          │         │   Compile    │         │
          │         └──────────────┘         │
          │                │                 │
          │                ▼                 │
          │             ◇ Error ◇            │
          └─── yes ──────          ──        │
                           │ no              │
                           ▼                 │
                    ┌──────────────┐         │
                    │     Test     │         │
                    └──────────────┘         │
                           │                 │
                           ▼                 │
                   ◇ Runtime Error ◇         │
                          ───── yes ─────────┘
                           │ no
                           ▼
                    ┌──────────────┐
                    │    Use it    │
                    └──────────────┘
```

In this context, **errors** are caught by the compiler. This is opposed to **runtime errors**, which are not caught by the compiler. These can be something like division by zero or infinite loops.