Computer Science & Engineering 155E
Computer Science I: Systems Engineering Focus

Lecture 01 - Course Introduction

Christopher M. Bourke
`cbourke@cse.unl.edu`

Overview of Computers and Programming

- Computer Hardware
- Computer Software
- Software Development (Problem Solving)
- Pseudocode
- Flowchart

Introduction to Computers

- Computers receive, store, process, and output information.
- Computer can deal with numbers, text, images, graphics, and sound, to name a few.
- Computers are useless without programming.
- Programming Languages allow us to write programs and thus communicate with computers.
- It takes our code and converts it into a format so the computer can understand it.

Different Types of Computers I

- Embedded Systems - iPod, cell phones, etc.
- Personal Computer - used by everyday people, least powerful of the three and typically used by just one person at a time.
- Mainframes - used for real-time systems, ATMs, and such, very powerful and reliable computers.
- Supercomputers - used by research laboratories for computationally intensive applications such as weather forecasting, the largest capacity and fastest mainframes.

Different Types of Computers II

Figure: PrairieFire is a 98-node Beowulf cluster, originally constructed in 2002. It originally scored an HPL benchmark number of 442.50 GFlops, which was then the 107th most powerful supercomputer in the world, according to the TOP500 supercomputer list. See `http://rcf.unl.edu/prairiefire/index.php`

Different Types of Computers III

Figure: 2007: Firefly built; 1,151 nodes (280 AMD Quad core, 871 dual core Opteron), 43rd fastest super computer at the time.

Hardware vs. Software

- **Hardware** is the equipment used to perform the necessary computations. Eg. CPU, monitor, keyboard, mouse, printer, etc.
- **Software** is the programs that enable us to solve problems with a computers by providing it with a list of instructions to follow. Examples: Word, FireFox, games, etc.
- **Firmware** - small programs stored in non-volatile memory

Computer Hardware

- Main Memory
  - **RAM** - Random Access Memory - Memory that can be accessed in any order.
  - **ROM** - Read Only Memory - Memory that cannot be written to
- Secondary Memory - Hard disks, floppy disks, zip disks, CDs, & DVDs.
- Central Processing Unit (CPU) - Coordinating all computer operations and perform arithmetic and logical operations.
- Input/Output Devices - Monitor, printer, keyboard, & mouse.
- Computer Networks (not hardware, but configuration of the hardware) - WAN, LAN, MAN, Wireless-LAN.

Memory I

Memory is an essential component in any computer.

- Memory Cell - the storage location.
- Address - the location of the memory cell relative to other memory cells
- Content - what is stored in the memory cell
  - All programs run in memory
  - Every memory cell has content, whether we know it or not. So always initialize variables

Memory II

Address	Contents
0	-27.2
1	42
2	0.005
3	-26
4	H
⋮	⋮
998	x
999	75.62

Figure: Portion of Memory Cells

Memory III

- bit - deriving from **b**inary dig**it** is either a 0 or 1.
- byte - a memory cell is actually a grouping of smaller units called bytes. A byte is made up of 8 bits.
- Example: A single character, `H` requires a byte to store
- kilobyte: 1024 bytes (*not* 1000 bytes)
- In general, memory sizes are powers of two, $2^{10} = 1024$ is the power of two closest to $1,000$
- megabyte: $2^{20} = 1,048,576$ bytes (or 1024 kilobytes)
- gigabyte: $2^{30} = 1,073,741,824$ bytes (or 1024 megabytes)
- kilo-, mega-, giga- may refer to bytes in base-10 in some contexts (network data) or when discussing *bits*

Memory

Exercise

Problem

A certain portable MP3 player is advertised as a having a "30GB" hard drive. In reality, it has $30,000,000,000$ bytes. How many actual (base-2) gigabytes does it have?

Divide by the number of bytes in a gigabyte:

$$\frac{30,000,000,000 \text{ bytes}}{1024 \times 1024 \times 1024 \text{ bytes-per-gigabyte}} = 27.93\text{GB}$$

Computer Software I

Operating System - controls the interaction between machine and user.

- Communicate with computer user.
- Manage memory.
- Collect input/Display output.
- Read/Write data.

Computer Software II

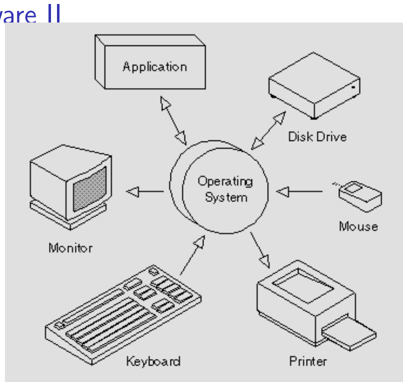

Figure: OS/device interaction

Computer Software III

- Application Software - developed to assist a computer use in accomplishing specific tasks (i.e. Word, Excel, & Explorer).

Computer Languages

- Machine Language - A collection of binary numbers
  - Machine language is not standardized, and will vary between families of processors, such as Intel (x86) and Macintosh.
- Assembly Language - mnemonic codes rather than binary.
  - Low-level language - A language that is close to the hardware.
  - Same structure and set of commands as the hardware, but uses names instead of numbers.
- High-level Languages - combine algebraic expressions and symbols from English
  - High-level language (HLL) - Closer to human language, easier to read, write, and maintain.
  - Must be translated to Machine language
  - Independent from the hardware
  - (Fortran, Cobol, Lisp, **C**, Prolog, Pascal, C#, & Java).

Computer Languages

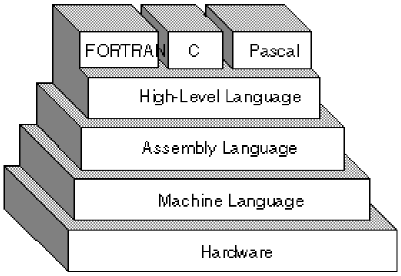

Figure: Language Hierarchy

Examples of Different Levels of Computer Languages

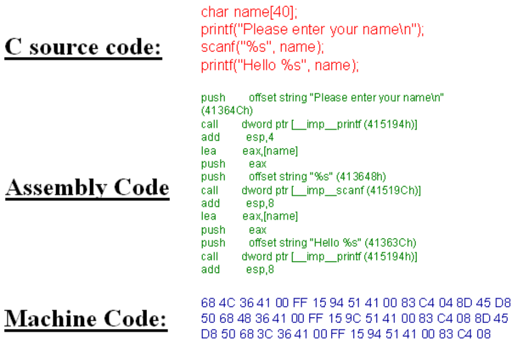

Figure: Examples of Languages

Compiling Code

- **Compiling** is the process of taking your source code and turning it into executable code.
- **Source file** - A file containing the program code.
  - A compiler turns the source file into an object file.
- **Object file** - a file containing machine language instructions.
  - A **Linker** turns the object file into an executable.
- **Integrated Development Environment (IDE)** - a program that combines simple word processing with a compiler, linker, and loader.

Compiler Overview

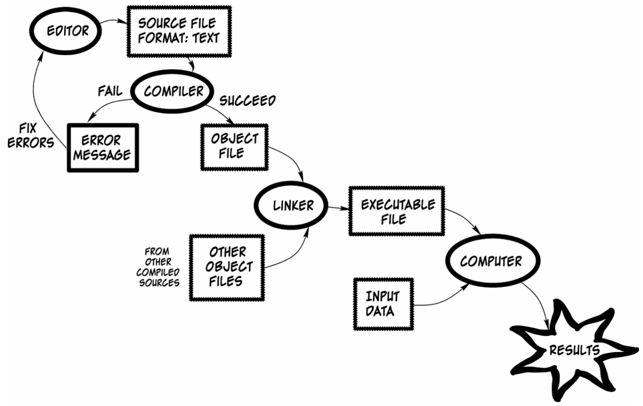

Figure: Compiling Process

Software Development Method

- Specify the problem requirements.
- Analyze the problem.
- Design the algorithm to solve the problem.
- Implement the algorithm.
- Test and verify the completed program.
- Maintain and update the program.

Steps Defined

- Problem - specifying the problem requirements forces you better understand the problem.
- Analysis - analyzing the problem involves identifying the problems inputs, outputs, and addition requirements.
- Design - designing the algorithm to solve the problem requires you to develop a list of steps that solve the problem and verify the steps.
- Implementation - implementing is writing the algorithm as a program.
- Testing - testing accuracy of the program.
- Maintenance - maintaining involves finding previously undetected errors and update the program to code.

Failure is part of the process.

Example: Converting Miles to Kilometers

- Problem - you summer job wants you to convert a list of miles to kilometers
- Analysis
  - Input: Number of miles,
  - Output: Number of kilometers,
  - Relevant info: 1 mile = 1.609 kilometers
  - Design:
    1. Get distance in miles
    2. Convert to kilometers
    3. Display kilometers

Example Program

Miles to Kilometer Conversion

```
#include <stdio.h>
int main(void)
{
    double miles, kilometers;
    printf("How many miles do you have?");

    scanf("%lf",&miles);

    kilometers = miles * 1.609;
    printf("You have %f kilometers\n",kilometers);

    return 0;
}
```

Testing

We need to test the previous program to make sure it works. To test we run our program and enter different values and make sure the output is correct.

Pseudocode

- **Pseudocode** - A combination of English phrases and C constructs to describe algorithm steps.
- **Flowchart** - A diagram that shows the step-by-step execution of a control structure.
  - Less commonly used than pseudocode, but gives you a visual feel for the flow of the program.
- **Algorithm** - A list of steps for solving a problem.

Pseudocode

- Pseudocode is simply an outline of a program.
  - Cannot be compiled nor executed,
  - There are no formatting or syntax rules.
- The benefit of pseudocode is that it enables the programmer to concentrate on the algorithms without worrying about the syntactic details of a particular programming language. In fact, You can write pseudocode without even knowing what programming language you will use for the final implementation.
- Program M2KM:
  1. Input Miles
  2. kilometers = 1.609 $\times$ miles
  3. Output Miles

Example of Pseudocode

Problem - How do I compute my grade for this class?

- **Specify the problem** - get the grades for the class and compute the final grade.
- **Analyze the problem** - we need to input the grades, output the grade, and percentage for each part of the class.
- **Design** -
  1. Get the grades homeworks, quizzes, exams and lab.
  2. Grade = homework * .25 + quizzes * .10 + exam1 * .2 + exam2 * .2 + lab * .25
  3. Output the Grade
- Implement - We can implement after we learn how to program.

Flowchart

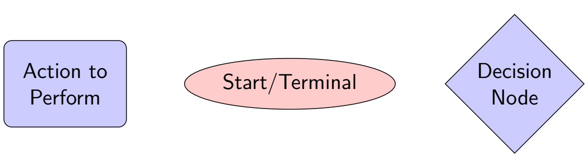

Figure: Flowchart objects

Example of Flowchart

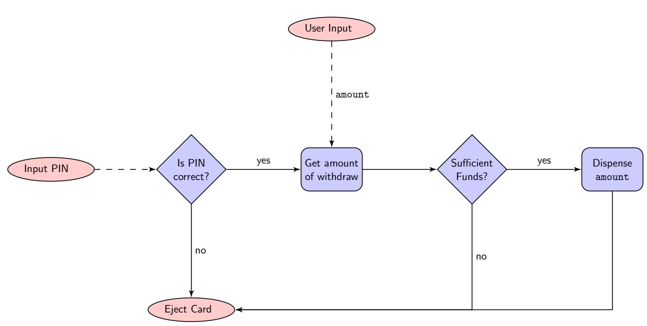

Figure: Flowchart Example: ATM

Seven Structures

- **Control Structure:** A method for controlling the order in which instructions execute.
- In C, there are 7 control structures
  1. Sequential
  2. If Then
  3. If Then Else
  4. Switch
  5. For Do
  6. While Do
  7. Do While

Sequential

- Use a sequential structure whenever program statements follow one after the another with no decisions and no repetitions.
- Processing flow is always downward from top to bottom in sequential structures.

Figure: Sequence

If Then

- Use the If-Then structure when there is a single process to do or not do.
- Processing flow is down either the left side or the right side.
- A *conditional* is checked.
- If it is *true* then the action is performed.
- If the conditional is *false* then the action is *not* performed and the flow continues.

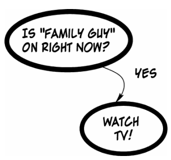

Figure: If Then

If Then Else

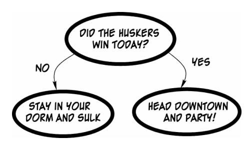

Figure: If Then Else

- Use If Then Else when one of two processes must be chosen.
- Processing flow is down either the left side or the right side.
- Similar to the If-Else, a conditional is checked, but *some* action is performed in either event.

Switch

- Whenever there are multiple potential options depending on a single values, use the switch statement.
- Example: Multiple If-Then-Else statements.
- If a conditional can match *several* possibilities, then a different action must be chosen for each possibility.

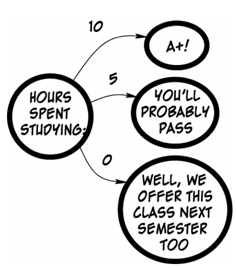

Figure: Switch

For Do

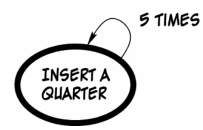

Figure: For-Do

- Use For Do when you need to repeat an action multiple times, and you know how many times you will repeat it.
- Also known as a *For-Loop*
- A specific action or actions are executed for as many times as are specified.
- Must know the number of times to be executed up front (though may still be variable, n).

While Do

- Use While-Do when the number of loops is unknown and process might not be executed at all (indeterminate pre-test).
- A conditional is checked before each execution to see if the loop should continue or end.
- Each time the loop executes, (hopefully) progress is made toward its *terminating condition*

Figure: While Do

Do While

- Do-While: similar to While-Do, *but* the action is executed at least once unconditionally.
- Conditional can be seen as being checked at the *end* of the loop.
- Difference is subtle but important.

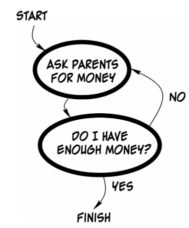

Figure: Do While

Which Control Structure to Use

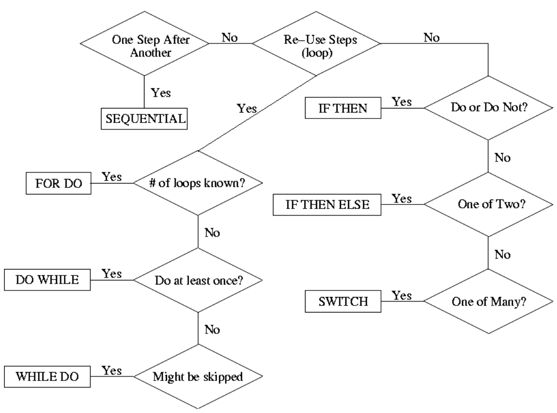

Figure: Flowchart to decide which type of control

Questions

Questions?

Example Program

Your First Program

Your first program:

1. Edit code using your favorite text editor (pico, emacs, gvim (tip: use gvim Easy), etc.)
2. Save code to a file, `helloWorld.c`
3. CSE command line: use `gcc` to compile into an executable file, `a.out`
4. Run program by calling `a.out`
5. More details in lab

Example Program

Hello World

```c
#include<stdlib.h>
#include<stdio.h>

int main(int argc, char *argv[])
{
  printf("Hello World!\n");
  return 0;
}
```