1a Computers, Problem Solving

Objectives

- To introduce the architecture of a computer.
- To introduce the notion of a programming language.
- To explore computational problem-solving.
- Read Chapter 1 of the textbook.
Computer Hardware

- Central Processing Unit (CPU)
- Main memory (MM)
  - Aka Random Access Memory (RAM)
  - Temporary storage of data/programs
- Input/output devices

- Secondary storage
  - Store information permanently

![Diagram of computer hardware components](image-url)
CPU (Central Processing Unit)

- **CU (Control Unit):**
  - Fetches and decodes instructions
  - Controls flow of information in and out of MM
  - Controls operation of internal CPU components
- **PC (program counter):** points to next instruction to be executed

CPU (Central Processing Unit) (continued)

- **IR (instruction register):** holds instruction currently being executed
- **ALU (arithmetic logic unit):** carries out all arithmetic and logical operations
Main Memory

- Directly connected to the CPU
- All programs must be loaded into main memory before they can be executed
- All data must be brought into main memory before it can be manipulated
- When computer power is turned off, everything in main memory is lost

FIGURE 1-2  Main memory with 100 storage cells
Software

- **Software**: Programs that do specific tasks
- System programs take control of the computer, such as an operating system
- Application programs perform a specific task
  - Word processors
  - Spreadsheets
  - Games

Programs

- CPU executes programs
  - Sequences of instructions
- CPU understands a small set of instructions
  - `MOVE 1002 R1`
    - Move the value in memory location 1002 to register R1
  - `MULT R1 R2`
    - Multiply the contents of R1 and R2
- And these assembly language instructions are coded in binary machine code
  - 00110110  01101101
Programs cont.

- We want to be able to write “higher-level” instructions like
  - area = height * length;
- And get a **compiler** to turn it into machine code.
- See example in lecture.

A C++ Program

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "My first C++ program." << endl;
    cout << "The sum of 2 and 3 = " << 5 << endl;
    cout << "7 + 8 = " << 7 + 8 << endl;
    return 0;
}
```

**Sample Run:**
My first C++ program.
The sum of 2 and 3 = 5
7 + 8 = 15
Processing a Program

To execute a program written in a high-level language such as C++
- Use an editor to create a source program in C++
- Use the compiler to
  - Check that the program obeys the rules
  - Translate into machine language (object program)

Processing a Program (continued)

- **Linker:**
  Combines object program with other programs to create executable code

- **Loader:**
  - Loads executable program into main memory
  - The last step is to execute the program
Use Smultron

Type into Terminal window:

make myprog

../myprog

FIGURE 1-3 Processing a C++ program

Simple View of Running a Program

This model allows a program to work with data previously stored in memory.

(We haven’t seen such a program yet.)
Problem Solving

- Programming is a process of problem solving
- Problem solving techniques
  - Analyze the problem
  - Outline the problem requirements
  - Design steps (algorithm) to solve the problem
- Algorithm:
  - Step-by-step problem-solving process
  - Solution achieved in finite amount of time
Problem Solving Process

- **Step 1 - Analyze the problem**
  - Outline the problem and its requirements
  - Design steps (algorithm) to solve the problem
- **Step 2 - Implement the algorithm**
  - Implement the algorithm in code
  - Verify that the algorithm works
- **Step 3 - Maintenance**
  - Use and modify the program if the problem domain changes

[Diagram: Problem analysis→coding→execution cycle]

Figure 1-4: Problem analysis→coding→execution cycle
Analyze the Problem

- Thoroughly understand the problem
- Understand problem requirements
  - Does program require user interaction?
  - Does program manipulate data?
  - What is the output?
- If the problem is complex, divide it into subproblems
  - Analyze each subproblem as above

Design an Algorithm

- If problem was broken into subproblems
  - Design algorithms for each subproblem
- Check the correctness of algorithm
  - Can test using sample data
  - Some mathematical analysis might be required
Write the Code

- Once the algorithm is designed and correctness verified
  - Write the equivalent code in high-level language
  - Enter the program using text editor

Compile and Execute

- We’ll be compiling our programs using “make” which takes care of details.
  - Compiler may find errors in your use of the rules (syntax) of the language
- The final step is to execute the program
  - Logic errors may show up then.