Introductory Lecture Notes on Computer Programming

These notes are for students taking a computer programming course in the University of Gaziantep. Here, the basic parts of a computer and ideas about problem solving with computers are introduced. A definition of flowcharts and algorithms are given at the end.

1. Course Resources

Web:

- Course web page → `www.gantep.edu.tr/~bingul/ep241`
- University of Gaziantep C++ resource page → `cpp.gantep.edu.tr`
  (you are responsible for everything under the tutorial link)
- C++ Resources Network → `www.cplusplus.com`
- C++ Reference → `www.cppreference.com`
- Türkçe ‘C Programlama Dili’ne Giriş’: `www.gantep.edu.tr/~bingul/c/`

Books:

- Programming with C++, *John R. Hubbard*, Schaum Outline Series (2000)
- Practical C++ Programming, *Steve Qualline*, O’Reilly Media (2003)

2. Computers and Programming

A *computer* is an automatic device that performs calculations, making decisions, and has capacity for storing and processing vast amounts of information. A computer can be divided into two main parts: *hardware* and *software*.

Hardware (=Donanım)
Hardware is the electronic and mechanical parts of the computer.

Examples of common hardware is given below:

Storage Units	*These are used in both input and output of data:* HDD (“Hard Disk Drive”) – high capacity. RAM (“Random Access Memory”) – low capacity, expensive, but very fast. Others: Flash memory (memory cards, USB flash drives), CD, DVD.
Input Units	*Used for input of data:* Keyboard, Mouse, Touch screen/pad, RAM, HDD, Flash memory.
Output Units	*Used for output of data:* Monitor, Printer, Speaker, RAM, HDD, Flash Memory
Process Unit	CPU: Central Processing Unit. *This coordinates the operation of computer system and performs arithmetic logic operations.*

Figure 1 shows the general block diagram of the hardware parts of a computer. Data is input to the CPU, results of computations are output.

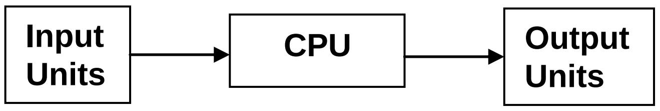


Figure 1: *General block diagram for the hardware parts of a digital computer*

Figure 2 shows a more specific block diagram where a program is input from an HDD(1) and executed in RAM(2). Data is input from a keyboard(3) which is again stored in RAM(4). The CPU operates on the program and data in RAM(5) and outputs results to the HDD(6) as well as the monitor(7). This is all controlled by the CPU requiring only basic data flow instructions from the programmer.

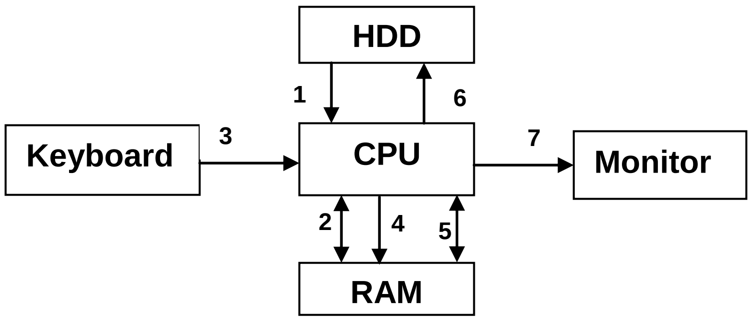


Figure 2: *Specific example of hardware involved in a computation.*

Software (=Yazılım)

Software consists of programs loaded from storage units. The programs execute on the computer hardware forming, for example, the *Operating System*, *Compilers*, *device drivers* and *Application Programs*:

Operating System (OS)	The OS is a program written to interface between the computer and it’s user. All other software runs under the OS. *Examples are: Windows 7, Linux,* Mac OS X*.*
Compilers	Many programming languages require a compiler to translate the statements of program written in a high level language into a low level language (machine code). *Examples are: Fortran, C, C++, Java, Pascal, Basic.*
Application Programs	These are (usually compiled) programs written to perform a specific task. Examples are: Miscrosoft Word, *AutoCAD, Mozilla Firefox.*
Device drivers	These programs are written to interface devices with the OS. For example a mouse, keyboard, HDD, or USB device.

Firmware (=)

A third kind of “computer ware” is *firmware*. This is usually a small permanent program that controls hardware at a low level such as a computer BIOS or HDD.

3. Creating and Running a Program

Editing, Compiling, and Running
To create and execute a program you need to invoke three environments; the first is the *editor* environment where you will create the program source code, the second is the *compilation* environment where your source program will be converted into a machine code, the third is the *execution* environment where your program will be run. In the lectures, we will use the *Dev-C++* compiler under Windows XP/Vista/7.

Steps of Program Development
A program consists of a set of instructions written by the programmer. Normally a high level language (such as Basic, C, C++, Pascal, Fortran) is used to create a plain-text source code, for example stored in the file `hello.cpp`. The file extension `.cpp` indicates that this is a C++ program source (use could also use `.cxx` or `.c++`). The following is an example for a simple C++ program source code; is is stored in the file `hello.cpp`.

```
// A simple C++ program
#include <iostream>
using namespace std;

int main(){
  cout << "Hello World!";
  return 0;
}
```

A compiler is then used to translate this source code into machine code, the compiled code is called the *object code*. The object code may require an additional stage where it is linked with other object code that readies the program for execution. The machine code created by the *linker* is called the *executable code* or *executable program*. Instructions in the program are finally executed when the executable program is executed (run). During the stages of compilation, linking, and running, error messages may occur that require the programmer to make corrections to the program source (debugging). The cycle of modifying the source code, compiling, linking, and running continues until the program is complete and free of errors. The Steps of executable program generation for the source code `hello.cpp` complition steps are shown in Figure 3.

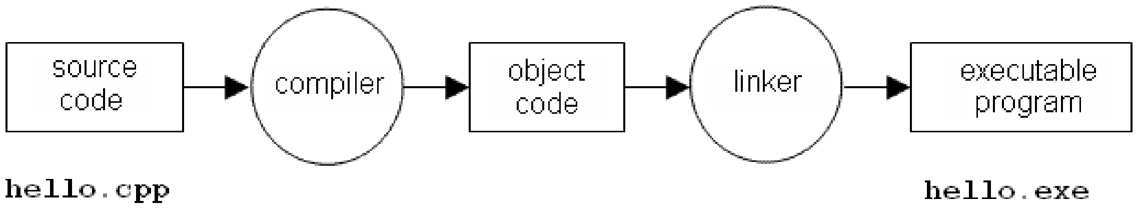


Figure 3: *Compiling a source code into and executable machine program.*

4. Problem Solving with Computers

A knowledge of using and programming computers is essential for scientists and engineers. The strength of the computer lies in its ability to manipulate and store data. The speed at which computers can manipulate data, and the amount of data they can store, has increased dramatically over the years doubling about every 18 months!
See: `http://en.wikipedia.org/wiki/Moores_law`

Problem solving with computers involves several steps:

1. Clearly define the problem.
2. Analyze the problem and formulate a method to solve it (see also “validation”).
3. Describe the solution in the form of an algorithm.
4. Draw a flowchart of the algorithm.
5. Write the computer program.
6. Compile and run the program (debugging).
7. Test the program (debugging) (see also “verification”).
8. Interpretation of results.

Verification and Validation
If the program has an important application, for example to calculate student grades or guide a rocket, then it is important to test the program to make sure it does what the programmer intends it to do and that it is actually a valid solution to the problem. The tests are commonly divided as follows:

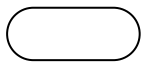
Verification	verify that program does what you intended it to do; steps 7(8) above attempt to do this.
Validation	does the program actual solve the original problem i.e. is it valid? This goes back to steps 1 and 2 - if you get these steps wrong then your program is not a valid solution.

5. Algorithms

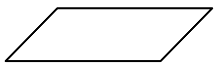
The algorithm gives a step-by-step description of the solution. This may be written in a non-formal language and structure. An example is given in the lecture.

6. Flow Charts

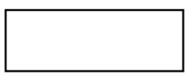
A flow chart gives the logical flow of the solution in a diagrammatic form, and provides a plan from which the computer program can be written. The logical flow of an algorithm can be seen by tracing through the flowchart. Some standard symbols used in the formation of flow charts are given below.



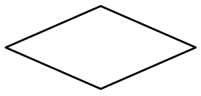
An oval is used to indicate the beginning



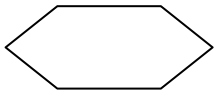
A parallelogram indicates the input or output of information.



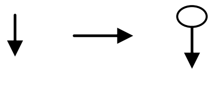
A rectangle indicates a computation, with the result of the computation assigned to a variable.



A diamond indicates a point where a decision is made.



A hexagon indicates the beginning of a repetition structure.



An arrow indicates the direction of flow of the algorithm.
Circles with arrows connect the flowchart between pages.