Fundamental C++

Cross-platform C++ programming for beginners

Chapter 2 – draft version

2. How does a computer work?

"Home computers are being called upon to perform many new functions, including the consumption of homework formerly eaten by the dog." –
Doug Larson

In this tutorial we will deal with programming, which means writing instructions to be executed by a computer – taken in the largest sense of the word. An example of programming is to write instructions that make it possible to operate a VCR by pressing a button on the remote control. Another example – that doesn't incorporate interaction with a human – is to display the time on a digital clock. Such applications are called embedded programming, since these programs will be entirely contained on a circuit board, possibly with chips. Embedded programming is often done in C++, due to its efficiency of program execution in comparison with other programming languages.

We will, however, be concentrating on writing programs for a Personal Computer (PC). This term is officially only applicable to IBM computers, but is used in daily life for all computers with a similar purpose – officially called IBM PC compatible –, though it does not apply to Apple Macintosh computers, which have a very similar purpose as other PC's. In this tutorial, we classify a PC as a computer that is fit for use by a single person simultaneously for general applications, ranging from text editing or e-mailing to making a video clip or playing a computer game. Nowadays, PC's come in a variety of shapes and sizes; think, for example, of laptop computers. However, there are also some types of computers that should not be mistaken for a PC, such as game consoles. These computers differ from PC's by their limited application possibilities and different way of communicating with humans – a joystick instead of a keyboard and mouse. The core of these machines is, however, remarkably similar to that of a PC.

Two basic concepts that are important when dealing with computers are *hardware* and *software*. Hardware means all tangible parts of a computer. Software means all intangible parts that are required to let a computer do what it is supposed to do. In a PC this is not just the programs stored on the hard disk, but also the embedded programs in a hard disk, DVD-burner or audio chip. In line with this definition, data that is stored on a computer, like a text document, a picture or a piece of music, is not considered software.

Exercise 2.1 Think of three more examples of embedded programming. For each example, state whether it deals with hardware or software.

2.2. Components of a PC

"A PC is as fast as its slowest component."

The largest part of all PC's is built up of the same type of components. See Figure 2.1 for an illustrative overview. First off, there is the motherboard (1), which is a big circuit board to which almost all other components are connected. The motherboard enables the communication between all these components. The heart of a PC is formed by its Central Processing Unit (CPU), often called processor (2). This is a chip that carries out most of the calculations in a PC. The data for the calculations and the results thereof are stored in

working memory, which is placed on the motherboard in the form of memory modules (3). Moreover, instructions that tell the PC what to do are also stored in working memory in the form of programs. Everything that is stored in working memory is lost once the computer is switched of or restarted.

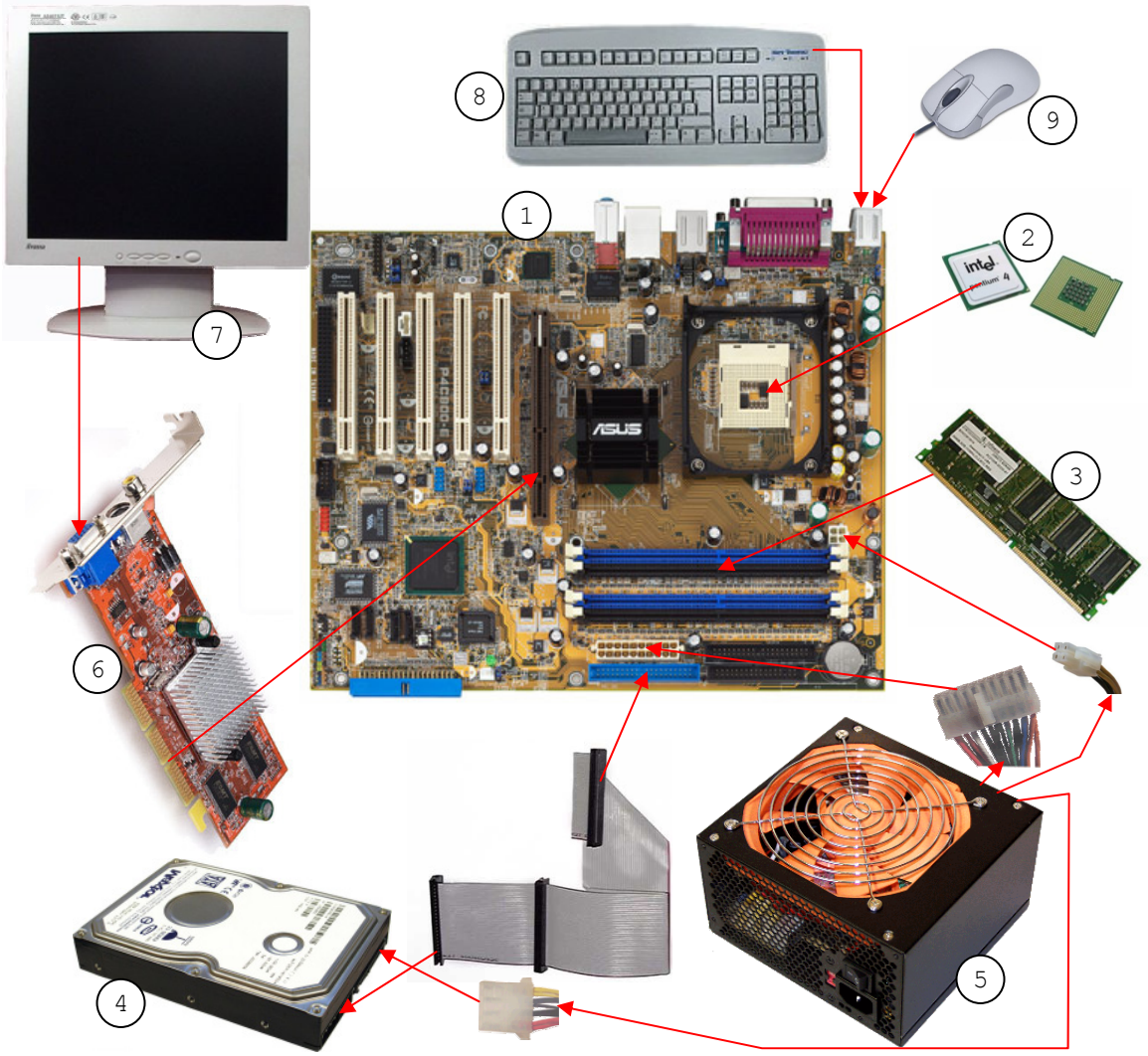

Figure 2.1: Overview of the components of a PC

Since the size of the working memory is too small to store all programs and data, most PC's have a hard disk (4), on which one can store much more. Also, the data on a hard disk remains when the computer is switched of or restarted. A hard disk works quite similar to a cassette tape, only the data is not written on a long tape, but on rotating platters. Since a hard disk contains moving parts, the speed with which it can read and write data will always be limited. Also it requires power for its motors, which in a PC is supplied by the power supply (5). The latter also connects to the motherboard to power all other components.

The last internal component that we consider is the graphics card or video card (6). It is placed onto the motherboard and has a connection for a monitor (7). On the monitor, the user of a computer can see the state of the computer and the output of the calculations it

has performed. Therefore, it is known as an output device. Conversely, the keyboard (8) and mouse (9) are known as input devices. The keyboard can be used to enter text or commands and with the mouse a cursor on the screen can be moved and commands can be given with a tap on one of the mouse-buttons, which is called clicking after the sound it makes.

A program on a PC is not able to address any of the components directly. Rather it gives instructions to the *Operating System* (OS), which the OS translates into instructions suitable for the specific component. An OS is totally software – excluding such things as the installation CD or manual – and is normally stored on the hard drive. Well known examples of operating systems are Microsoft Windows, Mac OS X, Linux and BSD. Not every program is able to run on all these operating systems. On the contrary, most applications out there only run on a couple of versions of a single OS.

In this tutorial, we will consider only *cross-platform* C++ programming, that is, programs written in C++ that can run on most, if not all, of the major operating systems. This implies that we have to restrict ourselves to so-called command line programs, which lack a graphical user interface. All the user of a program sees, when he runs it, will be some text – or other characters, which makes some crude drawing possible – and he will only be able to instruct the program by the use of the keyboard.

Exercise 2.2 Imagine that you are typing a letter on a PC. Explain as precise as you can what the described components, including the OS, do in order to facilitate you writing that letter.

2.3. Binary counting

"There are 10 types of people; those that understand binary numbers and those that do not."

A computer is able to count by the use of transistors; tiny electronic switches which can take one of two states: on (1) or off (0). Therefore, a computer counts with the binary system instead of the decimal system that we humans use. As its name conveys, the decimal system uses ten different figures – 0 through 9 –, whereas the binary system uses only two – 0 and 1. A figure in a binary number is called a *bit*; a collection of 8 bits is called a *byte*. Transformations between both systems can be made by use of the powers of 2, that is, 1, 2, 4, 8, 16, etc.

Given a binary number, we can convert it to a decimal number by summing a power of 2 for each 1 in the binary number. In particular, we must take the power equal to the position that the corresponding 1 takes, starting with 0 from the right. Hence, 1010 equals $2^1 + 2^3 = 2 + 8 = 10$. Starting from the right, the first bit is set to 0, so we can skip it. Were it set to 1, we would have to add $2^0 = 1$ to our decimal number. The second bit is set to 1, so we add 2. The third bit is set to 0 and the fourth to 1, so we add 8. Hence, the total is 10.

The other way round is a bit more difficult. We have to extract all powers of 2 from a decimal number and set the bits corresponding to those powers to 1. For example, the number 13 contains the powers 2^3, 2^2 and 2^0 ($8 + 4 + 1 = 13$). Hence, we have to set bits

0, 2 and 3 – again, starting with 0 from the right – to 1 and all other bits to 0, which gives the binary number 1101.

We have already mentioned that a collection of 8 bits is called a byte. A byte is the smallest amount of memory space that a computer can allocate. Hence, every elementary data object – such as a number, a letter or a colour – will be stored in a certain number of bytes. The leading bits will have the value 0. For example, if we store the number 5 in a byte, its values would be 00000101.

We have only considered the representation of non-negative integer numbers so far, but it is also possible to store negative and so-called *floating point* (real) values. Negative numbers can be stored by assigning one bit to represent the sign of the number; normally the first bit is taken. Such numbers are called *signed*. If the bit is equal to 0, the number is non-negative. Else, the number is negative and the value of the rest of the bits is added to the smallest (negative) number that can be represented. So, if we have a signed byte where the leftmost bit represents the sign, the number 5 would be 00000101, -5 would be 11111011, -1 would be 11111111 and -128 would be 10000000.

Floating point numbers can be used to store both positive and negative decimal numbers – and 0, of course. The numbers consist of one bit for the sign, several bits for the *exponent* and the remaining bits for the *mantissa*. The exponent is used to be able to store larger or smaller numbers by multiplying the value in the mantissa by $2^{c - exponent}$, where c is a constant determined by the number of bits in the exponent. It follows that floating point numbers use a scientific notation with a fixed accuracy determined by the size of the data element. For exactly this reason, floating point values are by nature imprecise and should not be used in for example accounting software, where rounding errors would be devastating. For most other purposes, the normal floating point data types provide ample precision.

Exercise 2.3 Convert a bunch of non-negative integer numbers from decimal to binary and vice versa.

Exercise 2.4 If a binary number is made up of *n* figures, how many figures would the corresponding decimal number have at least and at most?

a) How many different numbers can be stored in a binary number that has at most 16 bits? How about 32 bits? How many different numbers can be stored in a byte?

b) Why do you think 32-bit computers can use at most 4 GB (Giga bytes) of memory?

Exercise 2.5 Since binary numbers can get quite lengthy, programmers often work with the hexadecimal system, which uses the figures 0 through 9 and A through F, where A is 10 and F is 15 in decimal numbers.

a) Convert the numbers of Exercise 2.3 to hexadecimal numbers.

b) How many figures would the hexadecimal equivalent of a binary number consisting of *n* figures have?

Exercise 2.6 What does the binary value 10100111 represent?

2.4. Memory

"I hear and I forget. I see and I remember. I do and I understand."

The working memory in a PC is partitioned into the *stack* and the *heap*. The stack is a small part of memory, in which instructions to be performed by the processor are stored. Data can also be stored in the stack, but due to its limited size larger amounts of data need to be stored in the heap. To be able to store data in the heap, one first has to allocate the required memory space, before the data can be written. When the data is no longer needed, the allocated memory space should be freed. The heap part of working memory is huge in comparison to the stack part. However, there are programs that still require more memory space than is available in the working memory of the PC. Therefore, most operating systems are able to use part of the hard disk as an extension of working memory.

When the OS is moving data from the working memory to the hard disk to make more room in working memory or when that data is being placed back into working memory, we speak of *swapping*. Since the speed at which a hard disk can read or write data is much slower than that of the working memory, swapping will result in a large performance decrease. Therefore, it is important to free reserved memory in the heap when it is no longer needed, to prevent swapping.

Data can be stored both in working memory and on a hard drive. All data in working memory is lost when the PC is restarted or switched off, while the data on the hard disk remains even when the hard disk is taken out of the computer. Another difference is the way the data storage is structured. In working memory, data is stored and retrieved by programs – the actual physical memory is managed by the OS –, but on a hard disk data can only be stored in a *file*. A file is a collection of data that can be identified and referenced by its unique name and extension, which is used to indicate the type of data stored in the file. Examples of files are: text documents, such as this tutorial, pictures, web-pages, programs and archives.

Because the number of files on a hard disk can grow quite large, files are usually organised in a tree-like structure consisting of *directories* – sometimes called folders. The concatenation of all directories in the path from the root of the tree to a file separated by a character – normally / (slash) or \ (backslash) – is called the *path* of that file. A path may also include an identification of the *drive* on which the file is located, for example, 'C:\users\jhurkens\docs\tutorial.doc'. The restrictions on the name of a file or directory and the manner in which a drive is identified are all dependant on the OS. Most operating systems support a file name of 8 letters or numbers followed by an extension of 3 letters or numbers. Since the separating character in a path differs among operating systems, we cannot take advantage of directory structures in a cross-platform program.

Exercise 2.7 Try to find the exact path to the location of this file on your PC.

2.5. Programs

"program - A set of instructions, given to the computer, describing the sequence of steps the computer performs in order to accomplish a specific task. The task must be specific, such as balancing your check book or editing your text. A general task, such as working for world peace, is something we can all do, but not something we can currently write programs to do." – from the Unix User's Manual

We have explained earlier what software is and that an OS is a piece of software. We have also talked about programs – also pieces of software – without a more detailed explanation of what a program constitutes, as it is a quite general term. However, since this term is used quite often in this tutorial, we shall go deeper into its exact meaning in this section. Note that the term program has more than one definition in general, but we will restrict ourselves to just one of them for clarity.

A program is an executable file that contains a set of instructions that can be carried out by the OS. The file should have the extension '.exe', which is short for executable. When the program is started – also called *running the program* –, the OS performs the instructions that are contained in it and addresses the needed components of the PC to take action. This may include waiting for the user to give some input with the keyboard or mouse. The things that a program can do are limited only by the hardware of the PC, the time available for the program to run, logic and the creativity of the programmer(s) who made it.

A program can be started by the OS when the user instructs it to do so. The way to instruct the OS to start a program differs for different operating systems. Well known examples are typing the name of the program file at the command prompt or double clicking on the name of the file in operating systems with a graphical interface. A running program can be stopped prematurely by instructing the OS. In a command prompt this can usually be done by pressing Ctrl+C on the keyboard. If a program is not stopped manually, it may end when it has executed all its instructions or it may contain such instructions that it keeps running until the computer is turned off.

In this tutorial you will learn how to build programs with the help of the programming language C++ and other programs that can translate code written in that language to an actual program file. The exercises and examples that we give will mostly target programs for mathematical purposes, as this is a very useful way to explain the features of the C++ language. Only when one has learned to use a lot of the features of C++ in a proficient way, one is capable of truly using the great power of this wonderful language. Moreover, if you master C++, it is easy to learn almost any other programming language.

Chapter summary

- Though C++ can be used to program for almost any type of computer, we focus on personal computers in this tutorial. For all types of computers we can distinct hardware (tangible) and software (intangible) components.

- Most calculations on a PC are performed by the processor. Data for the calculations is stored in working memory or on the hard drive for long term storage. A PC's main output device is the monitor and its input devices are the keyboard and mouse. The operating system is the piece of software that enables programs to use the hardware components. This tutorial restricts itself to cross-platform C++ programming, meaning a program that we make can be run on most operating systems.
- Humans count with the decimal system, computers with the binary system. Numbers can be converted between both systems with the use of the powers of 2. A single figure in the binary system is called a bit. A collection of 8 bits is called a byte and is the smallest amount of memory space that a computer can allocated. Negative and floating point numbers are stored in a special way. Floating point numbers are imprecise by design.
- Working memory is partitioned into the stack and the heap. The stack is small, the heap big. Memory on the heap needs to be allocated in order to be used and freed when no longer needed, to prevent the OS from swapping, which decreases performance. On a hard disk, data can only be stored in a file, which can be identified by its unique name and extension. Files are organised in a tree-like structure consisting of directories. The concatenation of directory names separated by a separation character is called the path of a file.
- A program is a file that contains a set of instructions for the operating system. The execution of these instructions is called running the program. How a program can be started or stopped manually differs per OS.