Little endian and big endian

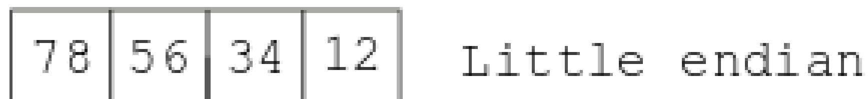
```
unsigned char uc[4];
int x = 10;
memcpy(uc, &x, 4);
printf("%X %X %X %X\n", uc[0], uc[1], uc[2], uc[3]);
```

The computer prints *A 0 0 0*, altough 10 is *0x 00 00 00 0A*

The reason is that *x* is stored in little endian format: the least signficant byte (0A) has the lowest address. You may also read "the smallest address", "the leftest byte", "the first byte", "the youngest byte".

In big endian format the least significant byte has the highest address and the printed result is *0 0 0 A*.

int x = 0x12345678



Little endian

Big endian

Addresses

The Intel processors and their compatibilities use little endian format. Data transformed by TCP/IP connection, however, are in big endian format. Working with Windows or Linux on a PC we may forget about endianness. The endianness is important when the data is transferred from one device to another: if the different endianesses are applied, we have to convert data.

Signed integers (1)

To store the sign (- or +) we have agreed that we use the lowest (first, leftest) bit: if it is 1, the integer is negative and if 0, then positive. Let us have $37_{10} = 25_{16} = 0010\ 0101_2$. Then we may suppose that -37 = $1\ 010\ 0101_2$. However, it does not work:

0 010 0101 // 37

+

1 010 0101 // -37

----

1 100 1010 // but should be 0000 0000

The negative integers are presented using the two's complement scheme:

1. The digits are inverted: 1 to 0 and 0 to 1, we get 11011010
2. 00000001 is added, we get 11011011

Now:

0010 0101 // 37

+

1101 1011 // -37 as presented in memory (0xDB)

----

1 0000 0000 // overflow, bit 1 is dropped, the result is 0

Signed integers (2)

```
unsigned char uc[4];
int x = -37;
memcpy(uc, &x, 4);
printf("%X %X %X %X\n", uc[0], uc[1], uc[2], uc[3]);
```

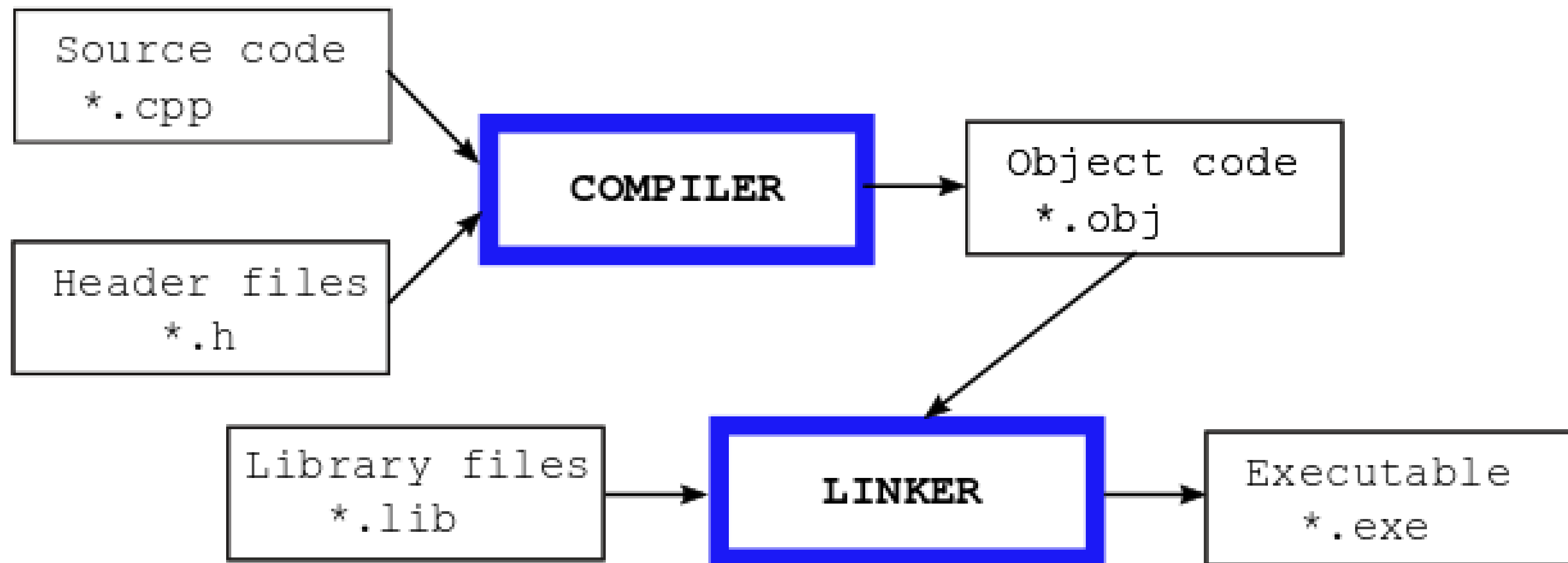
The computer prints *DB FF FF FF.*

```
0000 0000   0000 0000   0000 0000   0010 0101 // +37 on four bytes
1111 1111   1111 1111   1111 1111   1101 1010 // invert
1111 1111   1111 1111   1111 1111   1101 1011 // +1
1101 1011   1111 1111   1111 1111   1111 1111 // in little endian
  D    B      F    F      F    F      F    F  // in hexadecimal
```

More about two's complement scheme read:

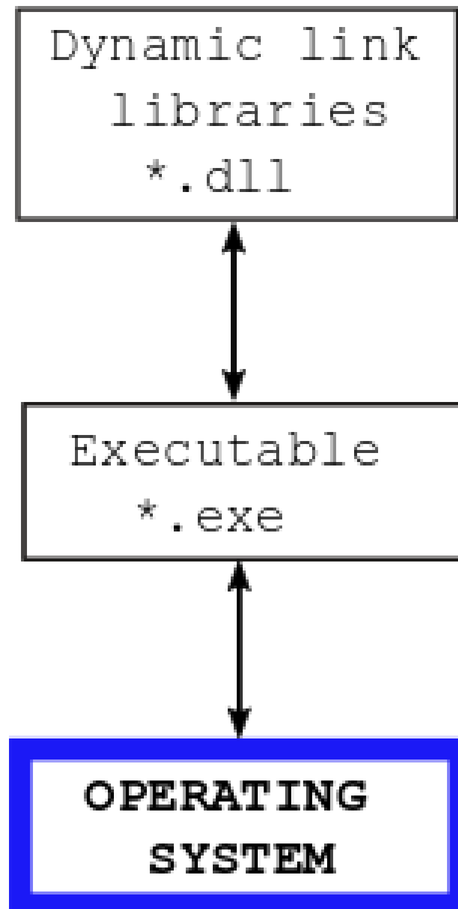
https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html

C / C++: compiled language (1)



The compiler takes the source code files (*.cpp and *.h) and converts them into machine language (*.obj - object module file). The linker combines the *.obj files with previously created (including standard) object modules packed into library files (*.lib) and produces the executable file.

C / C++: compiled language (2)



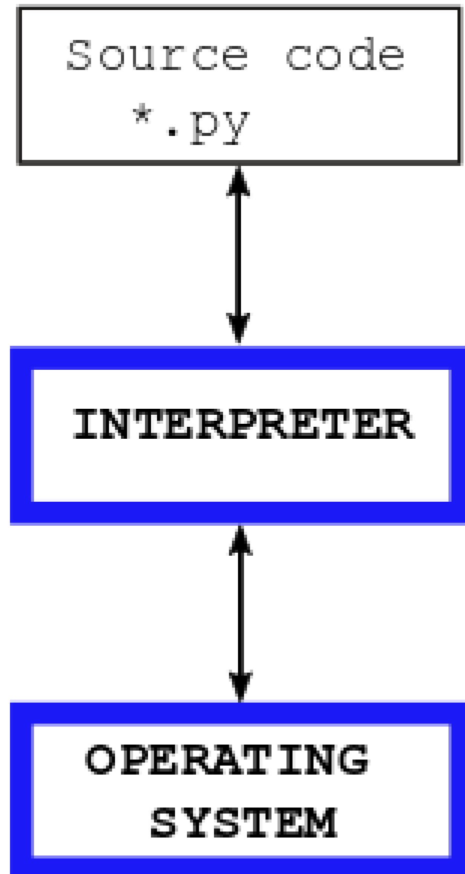
In large software products only the kernel part is linked into executable, the other parts are linked into dynamic link libraries (*.dll). When the program is running, the needed dynamic link libraries are temporarily connected to the executable.
If, for example, the developer has decided to divide his/her software product into executable and three DLL-s, he/she needs 4 Visual Studio projects. When starting with the new project, he/she must tell to the wizard, what is expected to get: *.exe or *.dll.

C / C++: compiled language (3)

As C/C++ is standardized, the C/C++ source code is mostly platform-independent, for example the source code developed on Windows is applicable also in Linux. Minor problems, however, may arise because not all the development environments follow the standard completely. Also, the operating systems have their own non-standard libararies for C/C++ (for example for file handling, input, output, etc.) and if those libraries are used, the porting from one platform to another is difficult.

Of course, the executable and DLL-s compiled and linked on one platform cannot run on some other platform. Each platform needs its own software development tools and produces runable software only for that platform.

Python: interpreted language

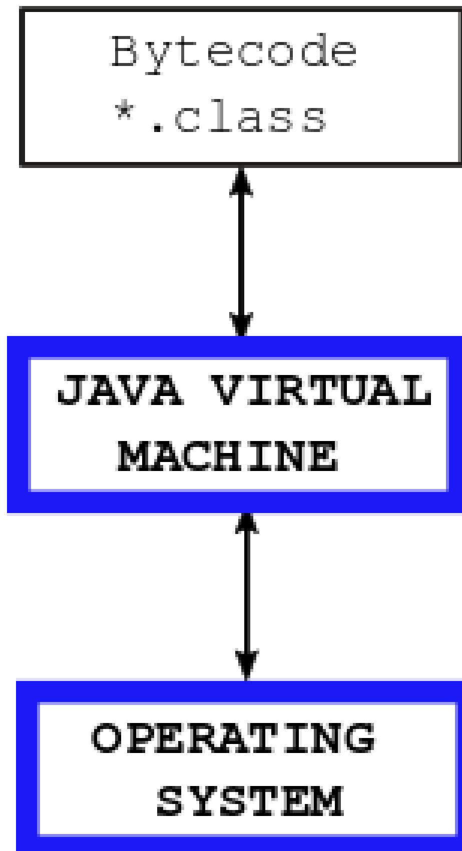


There is no compiling, no linking, no executables. There are only source code files.

Interpreted languages use a special program called as interpreter that converts the source code into machine language and executes the resulting machine instructions. The conversion takes place every time when an interpreted language program is running.

To develop Python software you need a text editor and the Python Interpreter. The source code is platform-independent, but of course each platform needs its own Interpreter.

Java



Source code files written in Java (*.java) are compiled but not into the machine language but into bytecode (*.class) – an intermediate and platform-independent language. There is no linking, but the bytecode files belonging to the same project must be in one folder or packed into archive file (*.jar).

The folder containing bytecode files or the archive is processed by a special program called as the Java Virtual Machine (JVM). The JVM (actually an interpreter) converts the bytecode into machine instructions and controls the run.

Each platform needs its own compiler and its own JVM. But the source code and bytecode are universal for any platforms. For example, bytecode created by Java / Windows development environment can run with JVM / Linux.

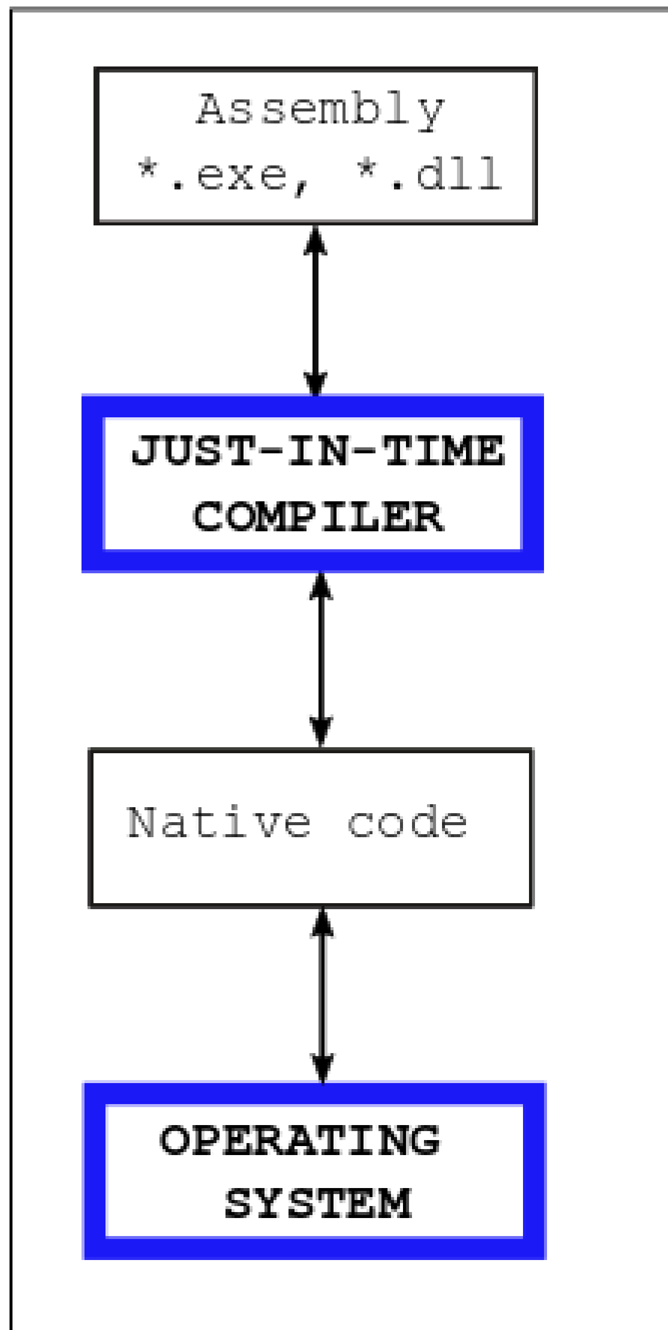
.NET family (1)

Mivrosoft .NET suppports about 40 languages but many of them are seldom used, abandoned or deprecated. The most populars are C# (*.cs) and Visual Basic (*.vb). Among the others there is also a C++ version called C++/CLI which is mostly used for transforming older C++ software into .NET environment.

The .NET compiler compiles all the source code files specified in the current project and links them into assembly (*.exe or *.dll). However, the assembly (although *.exe) is not an executable and cannot run without the support from Common Language Runtime (CLR).

Actually, the assembly is the code translated into Common Intermediate Language (similar to bytecode in Java). The source language may be any of languages from the .NET family, for example some of the project source code files may be in C# and the others in Visual Basic.

.NET family (2)



To run, the assembly needs Common Language Runtime, which includes Just-In-Time Compiler (similar to Java JVM) which translates the code from Common Intermediate Language into machine code (in .NET terminology native code) and controls the program when it is running.

Common Language Runtime is the standard component of Windows. The compiler and other development tools, however, are implemented in Visual Studio.

.NET technology is created by Microsoft for Windows. There is a .NET compatible freeware called Mono for developing and running .NET software on some other platforms (Linux, Android).

Version control (1)

If several programmers are simultaneously working on a single project, the version control tools are very necessary.

Each programmer as a member of the project team is responsible for a part of program (mostly called as module). His/her task is to write the code, debug and test it, introduce modifications, etc. But in most cases he/she cannot work independently: for debugging and testing he/she needs also modules developed by the teammates.

Suppose that programmer John must modify his module. It may take several days. If all the source code files of this project are on a server and accessible to all the members of team, John's tinkering with his code disturbes the others: they can continue to work only when John has finished. Consequently, John must have the complete project (i.e. all the source code files) in his own computer. In that case he may without any hurry modify his code and test the new version: his teammates are not affected. When John is sure that his new version is OK, he stores his source code on server and informs the others.

So, we need a database containing only those source code files that are considered to be correct. This database is called the repository. Modifying of source code directly on repository is not allowed. The members of team download the source code files into their personal computers getting thus their personal working copies. The modifying, debugging and testing is performed on the working copies. Source code files that are ready for sharing with the others are uploaded (commited) into the repository.

Version control (2)

Now suppose that after some weeks it turns out that the modifications introduced by John were not needed and the previous state of project must be restored. It is possible only if the repository contains not only the latest version but also all the older versions.

When John commits his modified source code, he must also add an explanatory note in which he decribes what he has done, which of his source code files are modified, which are new and which are removed. The project development history is very important for the managers as well as the programmers themselves.

To create a repository and work with it the version control tools are used. The most popular of them seem to be:

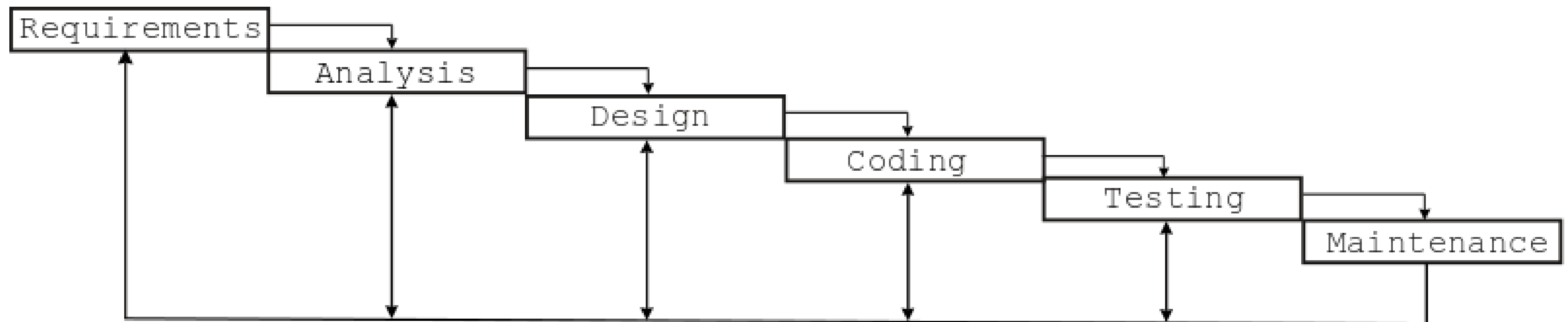
- Git (https://git-scm.com/)
- Subversion or SVN (https://tortoisesvn.net/)
- Perforce (https://www.perforce.com/)

A more detailed introduction to version control systems is on link:
https://homes.cs.washington.edu/~mernst/advice/version-control.html

Software development life cycle (1)

Software is developed through a series of steps. The sequence of steps is referred as the software development life cycle (SDLC).

There are several software development life cycle models. The simplest of them is the waterfall model:



The software development cycle itself is endless. We may from any stage go back to an earlier stage.

The phases are not absolutely separated from each other. For example, rather often the coding and testing are performed concurrently: if a modul seems to be ready, it will tested immediately. Design is partly performed by coding tools (specifying the function prototypes, etc.).

Software development life cycle (2)

You should bear in mind the order of phases from waterfall model when you are writing your program on the examination:

1. Requirements on the examination are presented by the examiner.
2. Analysis means that you must very attentively read the specification. If you have questions or uncertanties, ask the examiner. Your software product must strictly follow the specification. If not, your mark may be downgraded.
3. Design means that at first you must think about the general framework of your software product. Draw some flowcharts, decide how to divide the software into functions, write the protoypes of functions. Never start to write code without well thought-out plan.
4. When writing the code do not try to be too clever. Cumbersome C/C++ statements are elegant and short but there may be hidden rocks under the surface. Use parentheses and temporary variables to make your code readable and understandable. Remember that your functions must keep running even if the input data was not correct. Software that may crash or hang is worthless.
5. When testing turn special attention to cases in which the data to process is abnormal: has illegal values, is missing, etc.
6. Maintenance here means the evaluation by examiner.