

Chapter 1

Preface and history

On the choice
of the correct language -

I speak Spanish to God,
Italian to women,
French to men,
And German to my horse.

Charles V ^a

This textbook is designed to teach the beginners without any previous programming experience. The university or college students will receive the knowledges and skills how to solve regular practical tasks in console or in graphical window - using C/C++ programming language.

1.1 Programming paradigms

Programming language. An artificial language containing instructions how to communicate to computer, particularly computer system, is called the *programming language*. Optimal design of programming language has been under question about 60 years due to psychological human aspects. Programming language is devoted for programmers, who need to create abstract algorithms and/or electronic equipment controlling programs. Therefore the programming language has to be user-friendly, algorithmic, constructed using items of popular human language, for example English.

Most programming languages describe sequence of commands in an *imperative* style, although some languages, these that support functional programming, use alternative forms of description in *non-imperative* style.

Usage of existing programming languages as well as constructing of new programming languages depends on many sophisticated factors belonging to the current era:

^aCharles V (1500 - 1558) - ruler of the Holy Roman Empire from 1519.

- i) facilities of hardware (amount of memory, number of processors and registers, quantity and quality of servers, etc);
- ii) practical tasks to be solved (arithmetical, physical or data-mining);
- iii) education of programmers (mathematical or informatics);
- iv) dissatisfaction of programmers (in language expressions, style and hardware possibility);
- v) dominant programming styles and substyles in the market (OOP or procedural);
- vi) outlook for the future times (to be improved... something).

Which programming language is the best? The question belongs to the field of incorrect questions. A universal language should be *eclectic*. It means, that many user-friendly expressions and paradigms should be included in renewed version or in a new dialect. Big number of pragmatic and social factors determine the language usage. Significant factors, which are described below, could prelimit the usage of one or several languages as well as language dialects:

- a) language prevalence in the world and particularly in the local place;
- b) attitude of boss;
- c) possibility to use modern tools;
- d) previous *run-in*, etc.

The purpose of programming languages could be described in several positions. Firstly, programming language is the powerful tool for task and management programming: possibility to describe the algorithms and to design the high-level projects. Secondly, programming language is some sort of "communication language" between programmers because it presents possibility of conceptual representation of the relationship (paradigms).

Fig. 1.1 represents schematic behaviour of programming as a human cultural phenomena. First of all, urbanistic society must be ready to solve some complicated mathematical/arithmetical tasks, and/or public demand for information management must occur. Secondly, all types of practical tasks must be expressed in abstracted form such a mind (logical sequence of ideas). Thirdly, abstracted form of task (structured algorithm) must be presented graphically as a chart. Programming languages occur in this stage of development in order to make an algorithmisation process easier and errorless.

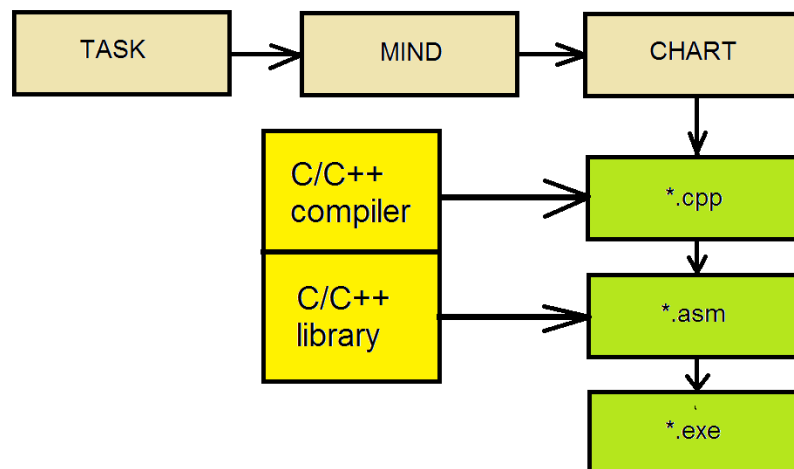


Fig. 1.1. Programming as a cultural phenomena.

C/C++ compiler converts source code (*.cpp), containing some sophisticated expression in human readable form, into an object containing Assembly code (*.asm). C/C++ linker converts *.asm file into executable *.exe by adding the libraries and standard templates. Several significant paradigms of programming - procedural, structured, object-oriented, aspect-oriented, generic - will be observed below.

Procedural (imperative) programming. Program consist of a set of instructions. Each operator prescribes to processor the work routine. Paradigm of *procedural (imperative) programming* was established for *Assembler*, and also early *Fortran* versions. It was quite useful for short programs. Generally, all programs of such type are of *single-entry/single-exit* type.

Structured programming. Program consists of a blocks (functions, subroutines, libraries) and block-files, containing structurized data. Paradigm of *structured programming* was established for *Pascal*, *C*, *Fortran*, *Basic*. It was useful for a quite long programs (for example, until fifty thousand lines of code). It is easy to read, understand, test and modify the programming code. Each action could be changed by set of two to three action. Simple procedure scheme could be inserted. For structural programming see Berry publication [29].

Object-oriented programming, OOP. Program consist of a real objects, constructed from abstract classes using templates. Classes contain the data of different types associated to the class-functions or methods. Encapsulation, inheritance, polimorfism allow to manipulate the objects associated by different degree of dependence. The essence of OOP is to create a real-world image in the structure of the program code. *Object-oriented programming, OOP* is a programming paradigm, when the program is modelled as a collection of objects associated with each other.

Paradigm of OOP was constructed in 1966 by Dhal ^b ir Nygaard ^c. Paradigm of OOP was established for *C++*, *C#*, *Java*, *Visual Basic*. It was very useful for a very long programs (for example, until million lines of code). For OOP see Coplien publication [9].

Aspect-oriented programming, AOP. *Aspect-oriented programming, AOP* is a programming paradigm which aims to increase program modularity by allowing the separation of *cross-cutting* concerns. AOP allows us the ability to apply aspects that alter the behaviour to classes or objects - independent of any inheritance hierarchy. We can then apply these aspects either during runtime or compile time. Popularity of AOP increases due to activity of the *Microsoft .NET* development community.

Generic programming, GP. *Generic programming, GP* is a modern style of programming when all algorithm or part is presented in non-specified items or terms. Such terms must be specified with initiation for specific types provided as parameters before its usage. Such software entities are known as generics in *Eiffel*, *Java*, *C#*, etc., and also in templates of *C++* and *D*. For generic programming see Czarnecki publication [13].

^bOle-Johan Dahl (1931 - 2002) - Norwegian computer scientist, Professor of Oslo university. He is known by pioneering concepts of class, inheritance, dynamic object, devoted for *SIMULA* programming language.

^cKristen Nygaard (1926 - 2002) - Norwegian computer scientist, Professor of Aarhus and Oslo universities. He is known as programming language pioneer (initiator of object-oriented programming paradigm) and politician.

1.2 Family of C programming languages

A full list of programming languages and dialects, similar or related to *C*, includes up to hundred positions. The main positions are occupied by *C*, *C++*, *C#*.

C(1973) was developed by Ritchie^d as a general-purpose system programming language. From 1980 to 2010 *C* was titled as a programming language, one of the mostly spread in the world. *C* compilers have been created almost for all types of computers. The first widely available book on the *C* programming language is known as *K&R C* (Kernighan and Ritchie) [38] [39].

C was standartized by the American National Standards Institute (ANSI) as *ANSI C* 1986). Most part of *C* code for current working computers is of *ANSI C* type. The usage of *ANSI C* guarantees the code portability (possibility to compile correctly on any hardware platform containing *C* software)

C++ means an implementation of *C* by *Object-Oriented programming*, *OOP* paradigms. According to that, compatibility of *C* and *C++* persists. *C++* started as a preprocessor for *C* and is currently nearly a superset of *C*. *C++*, created firstly as *C with classes* (1979), assumes the standartized implementation as *C++ std* (1995) due to Stroustrup^e [46] [47] activity. *C++* (named according to the incrementing operator) is becoming very popular in the world due to the well-balanced user-friendly command system.

C# (2000) is a OOP general-purpose programming language developed by *Microsoft* within its *.NET* initiative. *C#* is one of the programming languages designed for the *Common Language Infrastructure*, *CLI*. Several joined and new type paradigms were used: generic, object-oriented (class-based), and component-oriented programming.

Many later languages were created using *C* manner, including: *D*, *Go*, *Java*, *JavaScript*, *Perl*, *PHP*, *Python*, and *Unix's C Shell*. Similar syntax (main part) as well as expressional forms could be found.

C and *C++* are useful programming languages especially for large projects. There are many opinions in order to estimate the functionality or quantity of user-friendly expression in both languages.

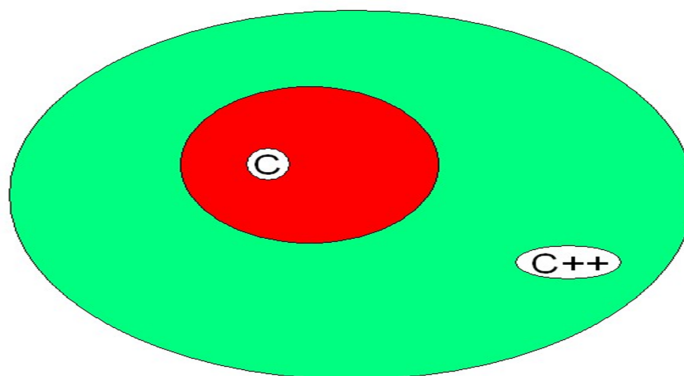


Fig. 1.2. *C++* as a superset of *C*.

^dDennis MacAlistair Ritchie (1941 - 2011) - an American computer scientist. He created the *C* programming language at *Bell Labs* between 1969 and 1973 and, with long-time colleague Ken Thompson, the *UNIX* operating system.

^eBjarne Stroustrup (born 1950) - a Danish computer scientist, mostly known for the creation and the development of the widely used *C++* programming language. Professor at Texas A&M University.

Main *C++* priorities to *C* could be formulated as follows.

1. *C++* shows about 10 % higher code productivity, in comparison to *C* - due to optimized code of libraries and usage of statements according to OOP paradigms.
2. It is easier to modify very large programs of *C++*, neither *C* - due to user-friendly expressions.
3. It is easy to use the specific libraries in *C++* due to the class existing, which is absent in *C*.
4. Class templates (present in *C++*, but absent in *C*) automatically modifies programming code.
5. *C++* programs could be very large in comparison to *C* - due to possibility to separate class method declaration from method interface.
6. Overloading of functions and operators allow to use the same names and symbols for different functions - methods.
7. Existing of different namespaces in *C++* allows to use the names without any danger of merging and allows to avoid the overlapping names, as could occur in *C*.
8. Using the references in arguments of functions or methods and returned values in *C++* is more preferable and user-friendly than pointers only in *C*.

1.3 Programming environment

Generally it is possible to write programs using text editor (for example *Notepad*) and then to use special compiler to create an executable program. But nowadays one can use integrated development environments (IDEs) to facilitate the program creating process.

An IDE is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, built automation tools and a debugger.

Usually there are several different IDEs for certain programming language. With *C++* we can use different IDEs: *C++Builder*, *Code::Blocks*, *Dev-C++*, *Microsoft Visual Studio*. Some of them are commercial, others are not. There are differences in functionality, additional features, but all of them help to create working program. *WxDev-C++* IDE was used to write code and create programs for this book.

Dev-C++ is a free IDE, developed by Laplace ^f, distributed under the *General Public License*, *GPL* for programming in *C* and *C++*. *MinGW*, a free compiler, bundled with it. It runs exclusively on *Microsoft Windows*. The *Dev-C++* project was not active from February 22, 2005 to June 2011. In 2006 forum post lead developer Colin Laplace stated, that he was busy with real-life issues and did not have time to continue development of *Dev-C++*. However, in June 30, 2011 an non-official version 4.9.9.3 of *Dev-C++* was released by an independent programmer, featuring the more recent GCC 4.5.2 compiler, Windows' SDK resources (Win32 and D3D), numerous bugfixes, and improved stability. In August 27, after five years of officially being in a beta stage, version 5.0 was released by the developer from Orwell's blog. *wxDev-C++* is also a free IDE that is based on the *Dev-C++*.

This version has additional features not found in the original *Dev-C++*. One of these is a visual RAD designer that works like *C++ Builder* to create *wxWidgets* applications. *wxDev-C++* supports both *Microsoft* and GNU *MinGW* compilers.

^fColin Laplace, programmer.

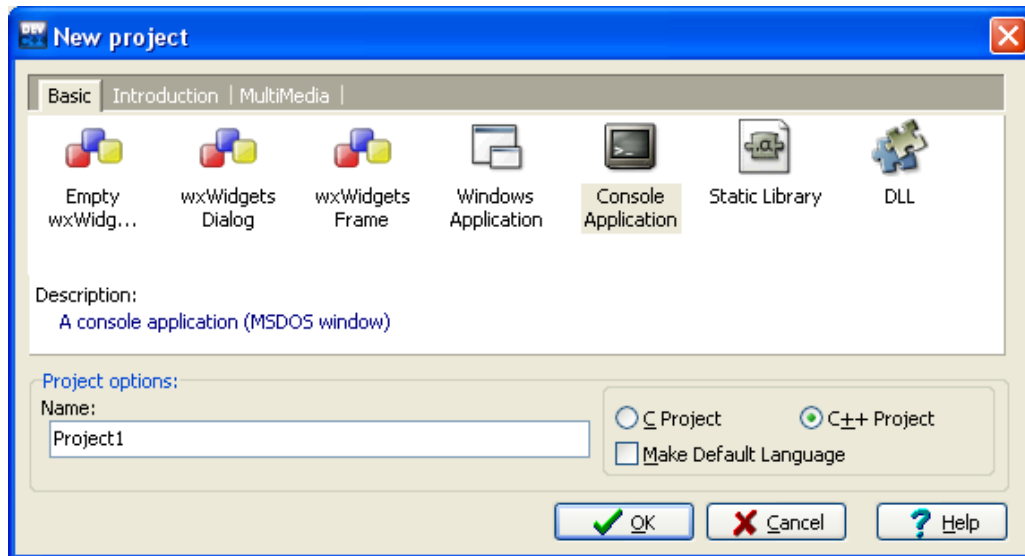
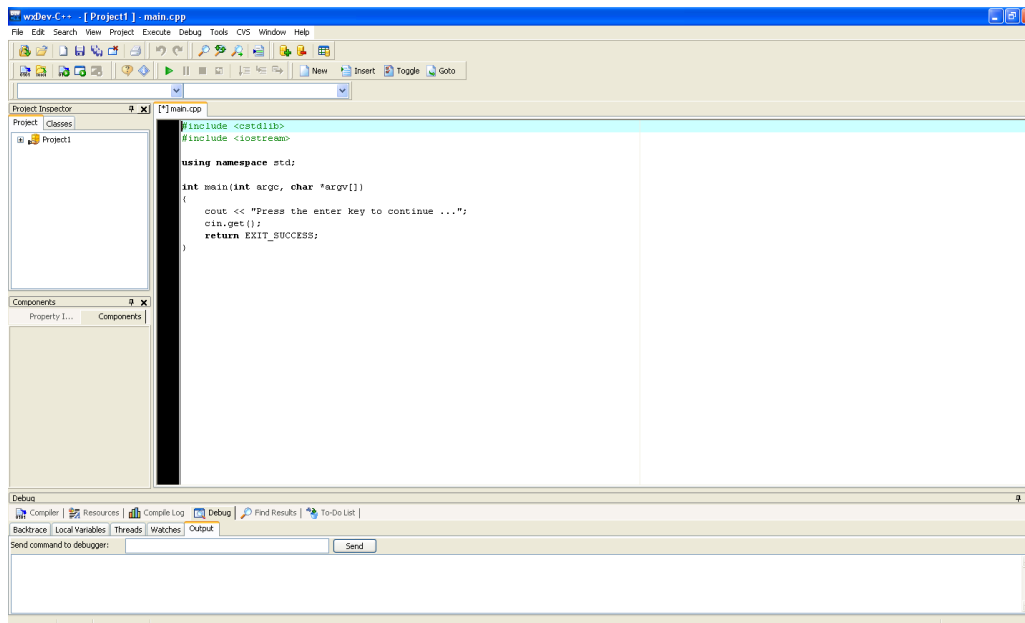


Fig. 1.3. New project menu.

Fig. 1.4. *wxDev-C++* window.

The latest version is 7.4.2, released in June 2012.

Modern IDE is a complex software system which has different functions. The scope of this book is *C++* programming. Necessary IDE functions will be also overviewed.

Program creation process is started by creating a new project in *wxDev-C++* (*File - New - Project*). Project of *Console Application* type will be used while studying *C++* basics (see Fig. 1.3). Then we will proceed with *Windows Application* and *wxWidgets* projects. This project consists of several different files and folders (project file, program code file, makefile, objects and output folders etc.). The number of files depends on project type and complexity. Project file of *.dev extension stores the main information about project. Program file of *.cpp extension stores code as the program text. Executable file of *.exe extension is present in the *Output* folder or in its sub-directory.

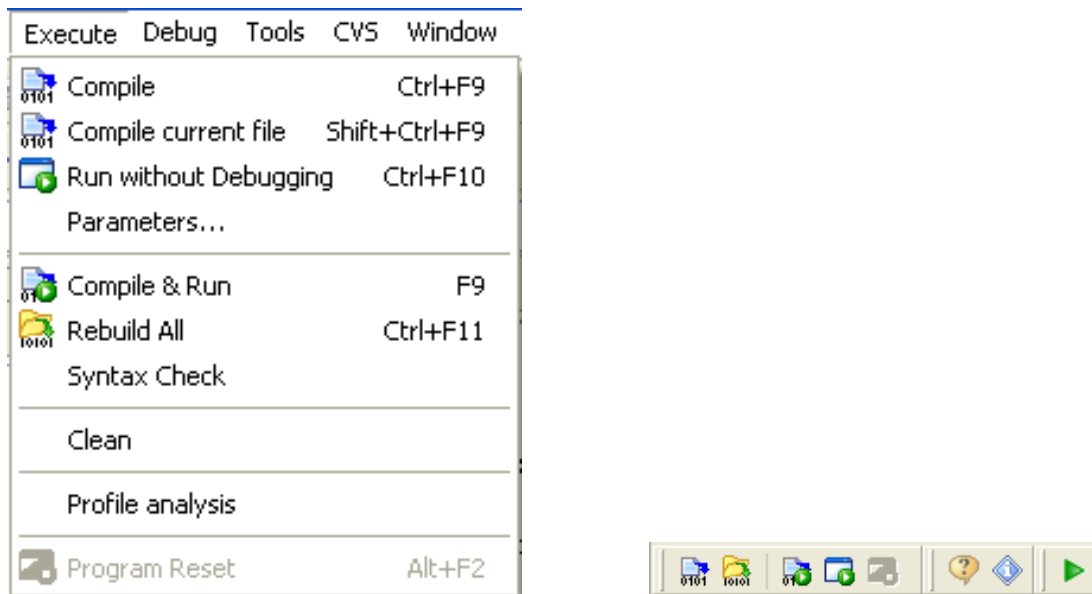


Fig. 1.5. Execute menu and corresponding command buttons.

In order to start programming one must choose a folder and name the project file and press *Save* button. The main program screen appears (see Fig. 1.4). On the top part of the window there is a main command bar, lower we can see command buttons. There is a Project inspector on the left part of the window and we see coding field (`main.cpp`) in the centre. We can see sample program text in the coding field. This program can be compiled and run immediately. Usually writing programs in C++ one must compile the program before running it. There are several ways to compile and run the program in *wxDev-C++*. The first way is to use the main menu command *Execute*, or we can use corresponding command buttons (see Fig. 1.5).

Another way is to use hot keys (*F9 Compile&Run*, *Ctrl+F9 Compile*, *Ctrl+F11 Rebuild All*, *F8 Run*, etc.). Compilation regime *Rebuild All* is more preferable to the others.

Figs. 1.6÷1.9 represent the progress of compilation.

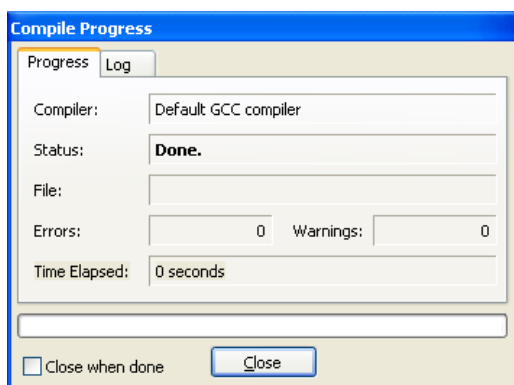


Fig. 1.6. *Compile Progress* window.



Fig. 1.7. Command button for executing - green triangle.

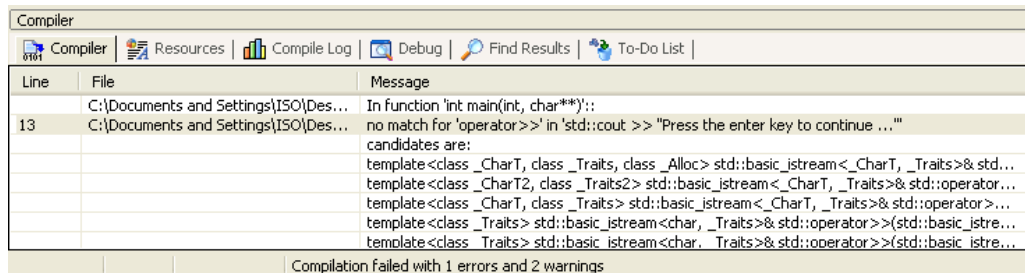


Fig. 1.8. Compiler message window.

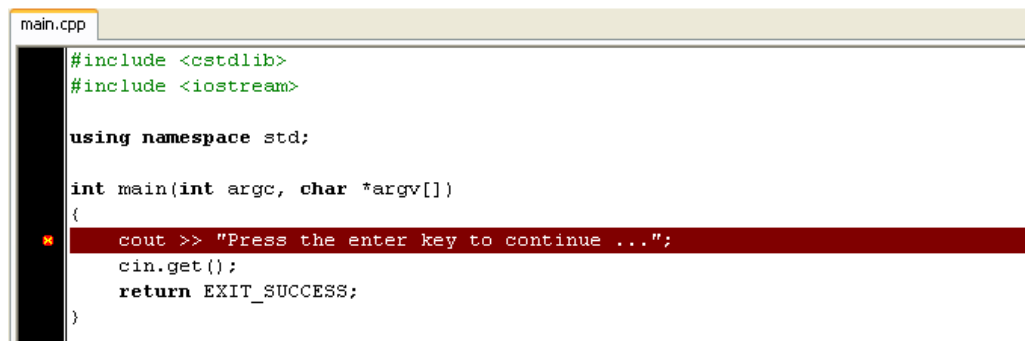


Fig. 1.9. A faulty program line.

The main program file must be saved before compiling the program for the first time (*Save File* dialog appears). Then *Compile Progress* window appears. *Done* status will appear if the program is compiled successfully (see Fig. 1.6). *Close* button should be pressed and then the program can be run.

The compiler message will appear in the lower part of the window if the compilation process fails (see Fig. 1.8). The faulty program line will be highlighted if there is a problem in program text (see Fig. 1.9). One must debug the program, fix all the errors and then recompile it. The program can be executed pressing *F8* or a command button (green triangle, Fig. 1.7.).

A message about program debugging appears before running the program as presented in Fig. 1.10. The default answer is *Yes*. This will enable the program debugging mode and recompile the project. It is possible to choose not to show the message again by putting the tick in lower left corner. The sample program will show as a console window - see Fig. 1.11.

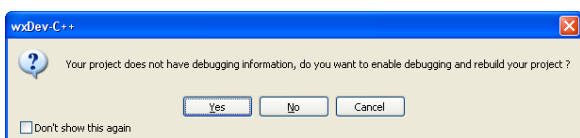


Fig. 1.10. The message about debugging.

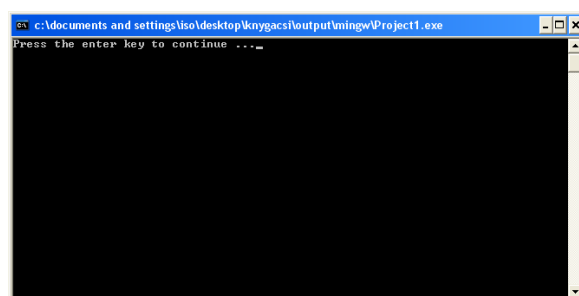


Fig. 1.11. The sample program window.