BENEFITS OF C ++ AS A FIRST LANGUAGE FOR TEACHING PROGRAMMING

Isoqova Adiba,
Termez State University,
Student of the Faculty of Information Technology.
E-mail: isabida010120@mail.ru
Salimova Nafisa,
Termez State University,
Student of the Faculty of Information Technology.
E-mail: nafisasalimova8@gmail.ru

Annotation:

This article is about why it is still worth choosing C ++ as the first programming language for teaching students, and a little about the problems of teaching in universities. Some of the judgment regarding the learning process is based on personal teaching experience as well as communication with teachers and students.

Keywords:

Programming languages, java, C ++, C #, Python.

Introduction

To begin with, I would like to note that for students of non-core specialties, the issue of choosing the first programming language is not considered. The basics of algorithmization (if necessary) can be mastered either in Python or in C ++ (Java, C #, Pascal). In this case, the simpler the language, the better it is: people need to form at least some culture of algorithmic thinking and understanding of basic constructions. For students of some specialties (legal, economic, humanitarian), the study of PL is not required at all.

Despite the obviousness of the above, many first-year economics students are taught Pascal programming in hands-on computer science classes. Students who still do not really know how to work with MS Word. The benefits of such activities are very, very doubtful. Likewise, math students can be taught to program in C ++ / C # / Java for a couple of years ... but why? It is much more useful for the subsequent application of your knowledge to study programs like Mathcad, Simulink, Surfer, etc.

Considering the above, we will consider the process of choosing the first language exclusively for students of specialized specialties (for example, "Software Engineering") and mixed specialties with a bias towards IT (for example, "Applied Mathematics and Computer Science"). Firstly, the curriculum of such specialties involves a sufficient number of lectures and practices (since the first language is considered, only the first course is taken into account): for two related disciplines (computer science and programming) about 230 hours, depending on the specialty. Secondly, the presence of interest and a certain mindset among students. Such students most often have already tried programming, and perhaps even wrote a website / toy. These two reasons, taken together, provide a good basis for starting learning and lower the entry threshold for language learning. In addition, graduates of these specialties will have to work in the software development industry in the future. Therefore, the choice of the first language is especially important for them.

Why C ++?

In the first year, the basis for further education is laid and the student's approach to further knowledge acquisition is formed. The programming language plays an important role here.

There are four reasons for choosing C ++ as your first programming language:

1. A statically typed compiled language.
2. A combination of high-level and low-level tools.
3. Implementation of OOP.
4. STL.

Let's consider these reasons in more detail.

Compiler. This is where C ++ is at its best. A lot of compilers, console commands, stages of building a program ... Yes, the first program must be written in a simple text editor without syntax highlighting and autocomplete, find how and how it can be launched. This approach forms in a person some understanding of how everything works:

• Program code is just text and won't work on its own.

• A compiler is a separate program that needs to be told what and how to do with the source code so that it becomes an executable file. A text editor is also a separate program for writing source code.

• There are build options and more than one compiler.

• The source code written by the programmer can be preprocessed and modified (for example, by the preprocessor).

The future specialist realizes that the code itself does not run (in the future, he may be interested in how, for example, the Python interpreter or JIT compilation works). A person will ask himself questions: “Why?”, “What is the difference?”, “How?”. There will be no illusion that everything works by pressing two magic buttons or in an interactive command line. The student will know that the build process of the program is customizable and that the source code can be processed by third-party programs. In the future, when using an IDE, a person will understand that this is just a convenient complex of programs that performs most of the routine operations and in case of insufficient flexibility, it can be abandoned or expanded.

Static typing.

Using a statically typed language as an example, it is easier to understand what a data type is, why it is needed, and what it depends on. You can see what is the declaration, definition and initialization. The use of the C ++ language makes this explicit, which contributes to further understanding of how these mechanisms work in other languages. In addition, you can use real examples to understand how unsigned integers differ from signed integers, how double and single precision numbers differ, how a character differs from a string, etc.

High-level and low-level tools.

Using tools such as pointers and dynamic memory allocation allows you to understand (or further helps to understand) what a stack, heap, call stack, stack unwinding, etc. are. In addition, the understanding of the concept of addresses and address arithmetic is reinforced in practice. Examples demonstrate that memory must be allocated, released, because it is not infinite, that there are memory leaks. In the future, when learning languages with GC, it will be easier to understand what it is.

Separately, it is worth noting a simple mechanism for passing values by reference, value, pointer and object transfer. What are mutable and non-modifiable parameters. In the future, these concepts can be used in the study of other languages. The student will understand, for example, that an object in the N language is passed by reference, and if its value is changed in a member function, then it will change everywhere.

OOP implementation.

This is a relatively clean implementation of OOP without any syntactic sugar (relative to some other languages). Clearly delineated levels of access to class members, the possibility of multiple inheritance and dynamic polymorphism make it possible to quickly learn the basic concepts of OOP (abstraction, inheritance, encapsulation and polymorphism). Pointers and dynamic memory allocation provide a clear understanding of important mechanisms such as upcasting and downcasting. In the future, based on this knowledge, you can easily understand all the syntactic sugar in other languages. The need to control resources (including the "rule of three" or already the "rule of five", taking into account C ++ 11), capturing them in the constructor and freeing them in the destructor also contribute to a deeper understanding of OOP.

It is worth noting such an important point as non-forced OOP. That is, this approach to programming is used when it is convenient, and it can be mixed, for example, with functional programming. This contributes to the formation of an understanding that the means of implementation are selected based on the task. STL. The very concept of C ++ templates, code generation and applying a wide range of algorithms to different containers has a positive effect on the learning process. Everything is on the surface here and it is clear why you can create a vector of integers and a vector of custom objects based on the same container class. Why can you apply some operation to a sequence of objects, or how to sort objects for which there is no built-in comparison operation. You can understand how elements are accessed and learn about the categories of iterators. In addition, the understanding of generic programming is reinforced.

A little about learning problems

The learning process is perhaps one of the most significant obstacles in order to realize everything that is written about in the previous paragraph. Probably, this question does not apply to top IT universities, but if we take ordinary educational institutions, there is a shortage of qualified personnel and weak student motivation. For most of the practical disciplines, specialists who are directly involved in software development are rarely involved. For example, a person who has not used STL in real projects is unlikely to be able to explain how to do it, and most importantly why. Just like the teacher, who sincerely believes that programming in Delphi with forms is already the most real OOP, given that all the code (without a hint of its own classes, abstraction and encapsulation) is written in the button click handler with a very clear name "Button1 ”Does not contribute to the learning process. There are also problems on the part of students, who did not really understand where and why they entered. Many students lack motivation for further learning and understanding, as well as for self-education. Despite the fact that such students and teachers complement each other perfectly, at the end of the training it will not work out a specialist applying for a junior vacancy.

Conclusion

At the moment, in the world of software development, a situation has developed that the price is the knowledge of certain technologies and the experience of their application, and not understanding. Modern society needs many programmers who can perform strictly defined functions. Developing development technologies also contribute in part to this. It is possible that most of the developers who know how to simply use a certain set of tools will never face the "law of leaky abstractions." However, people seeking leadership roles in optimization and architecture need a deeper understanding of how things work. One of the factors leading to such an understanding can be refused and the correct choice of the first programming language. Based on this, the C ++ language, which is statically typed, compiled, supports low-level memory management and is not overloaded with syntactic sugar, OOP implementation can be recommended as the first programming language.

Literature

1. Programming in Python / E. Mark Lutz - M., Symbol-Plus, 2018- 15 p.
2. Java. Programming Methods / E. Blinov I.N. - M., Four quarters, 2019 - 34 p.
3. Programming language C ++. Special edition / Ed. Bjorn Stroustrup - M., Binom 2019 - 45 p.
4. C ++ for Beginners / E. Martynov N. - M., Kudits-rss, 2017 -23 p.