Ten-Years Pedagogical Experiment at Moscow University of Geodesy and Cartography: C++ Programming Course Tailored for Surveying Students

Vladimir ZABLOTSKII, Russia

Key words: teaching C++ programming, pointer, reference, geodetic true azimuth

SUMMARY

A C++ programming course tailored for cartographers and surveyors has been developed and implemented in the educational process at the Moscow State University of Geodesy and Cartography. Pedagogical experiments on the development of the new course have been carried out since 2009. The C++ programming course contains a large number of cartographic and geodetic tasks aimed at illustrating various constructions of the programming language. Resource books and guidelines for students have been developed in the form of separate lectures containing more than seventy model C++ programs. The program being developed is designed to study passing of parameters to the function by value and with the help of a pointer and a reference. The program computes the reverse true azimuth of heading using several functions. The program clearly illustrates such important constructs of C++ language as pointers and references, introducing the rules of their declaration and use. The readability of function code using pointers and references is compared.

SUMMARY

Разработан и внедрен в Московском государственном университете геодезии и картографии курс программирования C++ адаптированный для картографов и геодезистов. Педагогические эксперименты по разработке нового курса проводились начиная с 2009 г. Курс программирования C++ содержит большое количество картографических и геодезических задач, направленных на иллюстрацию различных конструкций языка программирования. Подготовлены методические пособия для студентов в виде отдельных лекций по курсу, содержащих более семидесяти типовых C++ программ. Рассматриваемая программа предназначена для изучения передачи параметров в функцию по значению и с помощью указателя и ссылки. В программе вычисляется обратный истинный азимут направления с помощью нескольких функций. Программа наглядно иллюстрирует такие важные конструкции языка C++ как, указатели и ссылки; знакомит с правилами их объявления и использования в программах. Сравнивается читаемость кода функций с использованием указателей и ссылок.

Ten-Years Pedagogical Experiment at Moscow University of Geodesy and Cartography: C++ Programming Course Tailored for Surveying Students

Vladimir ZABLOTSKII, Russia

1. INTRODUCTION

As a result of the tremendous development of personal computers and mobile devices, programming is becoming an extremely popular knowledge area. Currently, there are prerequisites for integrating computer science into other sciences, forming new areas of computer science and programming. The introduction of computer science and programming into any science also serves as an incentive for the development of this discipline. Such an effective interaction can be found on the example of statistics, which over the past decades has received a range of extremely powerful instruments for conducting practical research.

In order to implement effective interaction between programming and other scientific disciplines, it is necessary to train specialists ever since they were at university who would be able to organize such interaction in practice. If, for example, students of the University of Geodesy and Cartography are trained to use computer science and programming in the field of their future activities, then after graduation, specialists will be able to solve problems in computer science and programming, geodesy and cartography effectively. Computer science and programming are in such a stage of development when it is possible to combine them with various scientific disciplines, at least for educational purposes.

The Moscow State University of Geodesy and Cartography (Russia) conducts an educational experiment, the essence of which is to teach cartographers and surveyors C++ programming on the basis of solving training tasks in the field of cartography and geodesy. This experiment has been carried out for more than 10 years and has been received extensive educational and methodical material, including computer programs developed to illustrate the solutions to various issues of cartography and geodesy, methods and techniques of presentation of educational material to students, the sequence of issues studied. Artwork and illustrations have also been developed to perform lectures in large auditoriums equipped with computer projectors and workshops in laboratories with electronic interactive boards. A set of tasks that serve as take-home assignment is developed. All training materials cover the full C++ programming course.

Supplying the educational process with literature is an important characteristic of the academic discipline. Nowadays the adequate supply is observed. There are many useful general (but not special) programming textbooks. In regard to the C++ language, for example, they are a classic, multi-page textbook written by Steven Prata, or the textbook written by Bjarne Stroustrup, tutorials of Jesse Liberty and many others. The little list of classical C++ programming textbooks is given in reference. In fairness it must be said that programming textbooks for students of different specialties have not yet been introduced. This is a very pressing problem. There are examples of successful solutions to similar problems in the other

scientific disciplines. For example, there is a physics textbook for chemical specialties or a textbook on higher mathematics for economists. Today the draft textbook C++ programming for cartographers and surveyors was written by author and materials are testing on practical work with the students (Zablotskii, 2019).

2. METHODS OF RESEARCH

A program that transfers the parameter to a function by value, using a pointer and a reference is considered as an example of a computer program which illustrates the course of C++ programming for cartographers and surveyors. We will scrutinize this important issue using the example of the following geodetic task. On the topographic map, the straight line A-B is set and the true azimuth of this line at point A is known (44°06′00″). For simplicity, we will consider that point A lies on the axial meridian of the six-degree Gaussian zone. Then for point A, the direction to true north and grid north coincide. It is required to calculate the reverse true azimuth of this line at point B if the convergence of meridians for points A and B is known. For points A and B the convergence of meridians is $\gamma = +01°24'00''$. The numerical values for this task are shown in Figure 1.

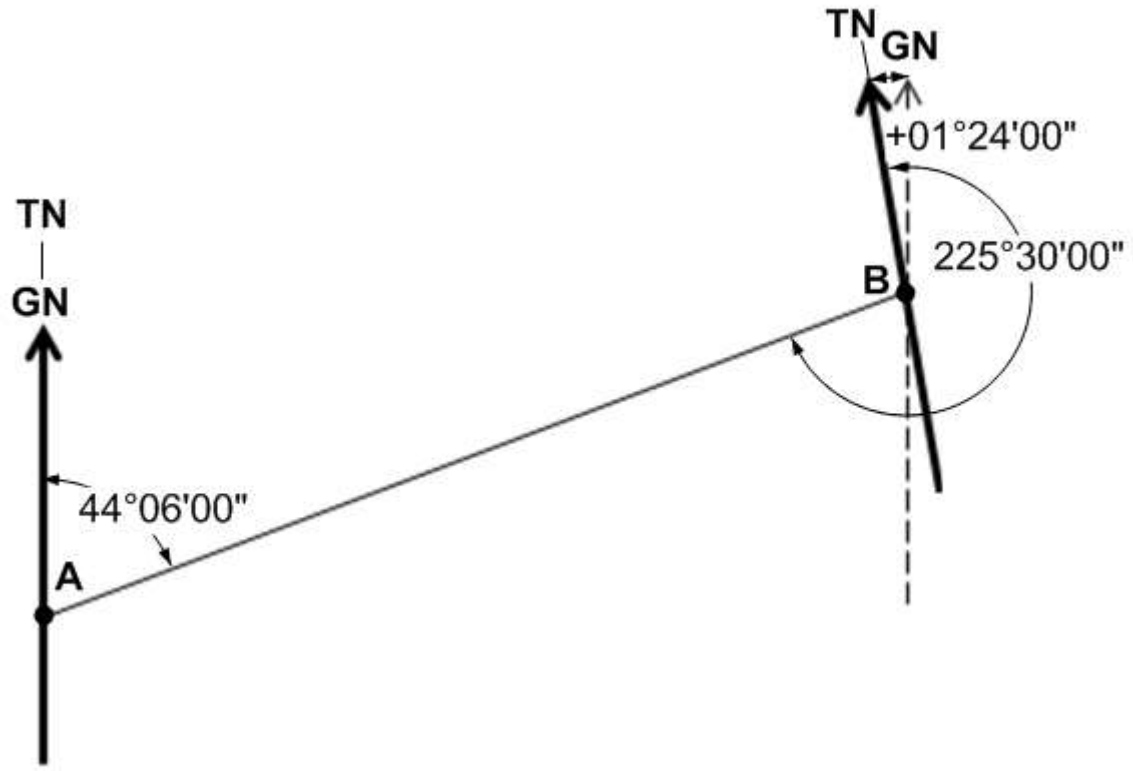

Fig. 1. The diagram illustrates a geodesic problem. Line A-B has a known direct true azimuth at point A. The bold arrows show the direction to true north; the dashed arrow for point B shows the direction to the grid north.

In the presented task, the true azimuth is considered as the angle between the direction of the true (geographical) meridian and the direction of the given line A-B. The solution to the problem is based on the relation between the true direct and reverse azimuth. It is known, if the direct true azimuth of the line is in the range of angles from 0 to 180°, the relation between the reverse and direct azimuth is determined by the formula:

$$A_{rev} = A_{dir} + 180° + \gamma$$

where A_{rev}, A_{dir} - reverse and direct true azimuths, γ - convergence of meridians.

To compare the features of passing the parameter to function in different ways, a computer program was developed. There the reverse azimuth is calculated using three functions: *NoChangingAzimuthByValue*, *ChangingAzimuthByPointer*, *ChangingAzimuthByReference*. In the first function, the parameters for calculating the reverse azimuth are passed by value, the second and third function - by the address, through a pointer and a reference. This program is intended at the making students of cartography and geodesy familiar with the features of using pointers and references to pass parameters to functions.
Together with the program, students are given brief background information on pointers and references. The text presented below is an example of such an information guideline for students. The analysis of the program code is also a teaching material intended for students.

3. AN EXAMPLE OF TRAINING PROGRAM

Let us take a closer look to the code of the program. The main function (line 09) creates two variables, one is *trueAzimuth* - true azimuth and two is *convergenceOfMeridian* literally - convergence of meridians. The *trueAzimuth* variable is initialized by 44.1(°), and the *convergenceOfMeridian* is initialized by value of 1.4°. The program defines the function *NoChangingAzimuthByValue*, the prototype of which is written on line 04. The *ChangeAzimuthByPointer* prototype is in line 05. Two parameters *trueAzimuth* and *convergenceOfMeridian* are passed to these functions. The *NoChangingAzimuthByValue* function computes the reverse azimuth value and displays the value on the screen. Then the *ChangingAzimuthByPointer* function is called, which also computes and displays the reverse azimuth. After each call of these functions, the control flow return to the main function, where the *trueAzimuth* variable is displayed on the screen. As a result, it is possible to compare the values of the azimuth before and after the calling of these functions. In the final part of the program, the *ChangingAzimuthByReference* is called which also computes and displays the reverse azimuth on the screen.

```
01:  #include <iostream>
02:  using namespace std;
03:
```

```
04:  void NoChangingAzimuthByValue(double, double );
05:  void ChangingAzimuthByPointer(double *, double);
06:  void ChangingAzimuthByReference(double &, double);
07:
08:  int main(void)
09:  {
10:   double trueAzimuth = 44.1;              //  44°06′00″
11:   double convergenceOfMeridian = +1.4;    // +01°24′00″
12:   double &azimuth = trueAzimuth;
13:
14:   cout <<"Initial azimuth value: "<< trueAzimuth << endl;
15:   cout <<"Reference value: "<< azimuth <<"Reference address: "
15:        << &azimuth << endl;
16:
17:        //passing the first parameter by value
18:   NoChangingAzimuthByValue(trueAzimuth,convergenceOfMeridian);
19:   cout <<"Azimuth after calling NoChangingAzimuthByValue:"
19:        << trueAzimuth << endl;
20:
21:   cout <<"TrueAzimuth variable address:"<< &trueAzimuth << endl;
22:
23:        //passing the first parameter by address through pointer
24:   ChangingAzimuthByPointer(&trueAzimuth, convergenceOfMeridian);
25:   cout <<"Azimuth after calling ChangingAzimuthByPointer: "
25:        << trueAzimuth << endl;
26:
27:        //passing the first parameter by address through reference
28:   ChangingAzimuthByReference(azimuth, convergenceOfMeridian);
29:   cout <<"Azimuth after calling ChangingAzimuthByReference: "
29:        << trueAzimuth << endl;
30:
31:   return 0;
32:  }
33:  void NoChangingAzimuthByValue(double trueAzimuth, double
33:                                                  convergenceOfMeridian)
34:  {
35:   trueAzimuth += 180 + convergenceOfMeridian;
36:   cout <<"Reverse true azimuth in NoChangingAzimuthByValue:"
36:        << trueAzimuth << endl;
37:  }
38:  void ChangingAzimuthByPointer(double *trueAzimuth, double
38:                                                  convergenceOfMeridian)
39:  {
40:   *trueAzimuth += 180 + convergenceOfMeridian;
41:   cout <<"Reverse true azimuth in ChangingAzimuthByPointer: "
41:        << *trueAzimuth << endl;
42:  }
43:  void ChangingAzimuthByReference(double &azimuth, double
43:                                                  convergenceOfMeridian)
44:  {
45:   azimuth -= 180 - convergenceOfMeridian;
46:   cout <<"Reverse true azimuth in ChangingAzimuthByReference: "
46:        << azimuth << endl;
47:  }
```

After compiling and running the program, the following output will appear on the screen:

```
Initial azimuth value:                                  44.1
Reference value: 44.1                                   Reference
                                                        address: 0x12ff4c

Reverse true azimuth in NoChangingAzimuthByValue:       225.5
Azimuth after calling NoChangingAzimuthByValue:         44.1
TrueAzimuth variable address:                           0x12ff4c
Reverse true azimuth in ChangingAzimuthByPointer:       225.5
Azimuth after calling ChangingAzimuthByPointer:         225.5
Reverse true azimuth in ChangingAzimuthByReference:     44.1
Azimuth after calling ChangingAzimuthByReference:       44.1
```

4. DISCUSSIONS

According to the data that the program displays on the screen, the parameter value in the *NoChangingAzimuthByValue* function has been changed (from 44.1 to 225.5 degrees). However, once the function is complete, the value of the *trueAzimuth* variable in the main function remains the same. To understand why changing the setting in *NoChangingAzimuthByValue* didn't affect the main-based *trueAzimuth* variable, let us investigate how the parameter is passed to function.

The reason is that when the parameter is passed to the function, the compiler copies the values of that parameter to a temporary memory area called a stack. The feature that uses this parameter functions with a copy of the source value. When the function exits, the stack contents are reset and all changes made by the function in the copy of the parameter values disappear (Fig. 2).

Suppose the *trueAzimuth* variable occupies a memory location with address 0x12ff4c and *convergenceOfMeridian* is stored in a memory location 0x12ff54. Since these two variables are passed to the *NoChangingAzimuthByValue* function; the compiler copies their values to the stack. By the function *NoChangingAzimuthByValue* copies of the value of each variable will be used, meaning that this function can only access the contents of the stack, which consists of copies of 44.1 and 1.4, and modify them to make them suitable. However, since the *NoChangingAzimuthByValue* function is in no way associated with 0x12ff4c and 0x12ff54 memory locations, *NoChangingAzimuthByValue* cannot change the values of the *trueAzimuth* and *convergenceOfMeridian* variables that are defined in the main function.

Thus, the experiment shows that copying the value of a parameter from memory to the stack and to the function does not change the value of the parameter defined in the external function.

How to make a function change the value of a variable specified in another function? This can be done, but it requires the function to know the memory address of the required variable. To write the function with the address as a parameter, the operator "ampersand" (&) is used. The following function calling illustrates how using the take-address operator, the address of the variable *trueAzimuth* could be passed to the *ChangingAzimuthByPointer* function :

```
ChangingAzimuthByPointer(&trueAzimuth, convergenceOfMeridian);
```

When determining *ChangingAzimuthByPointer* function, you need to tell the compiler that the program will transmit the parameter using the address. To do this, a pointer variable is created in the function header using the asterisk operator, as shown below:

```
ChangingAzimuthByPointer(double *trueAzimuth,double convergenceOfMeridian)
```

In addition, inside the function, you need to show that the feature works with the option address. To do this, the pointer has to be "dereferenced", which could also be done using the asterisk operator:

```
*trueAzimuth = *trueAzimuth - convergenceOfMeridian;
```

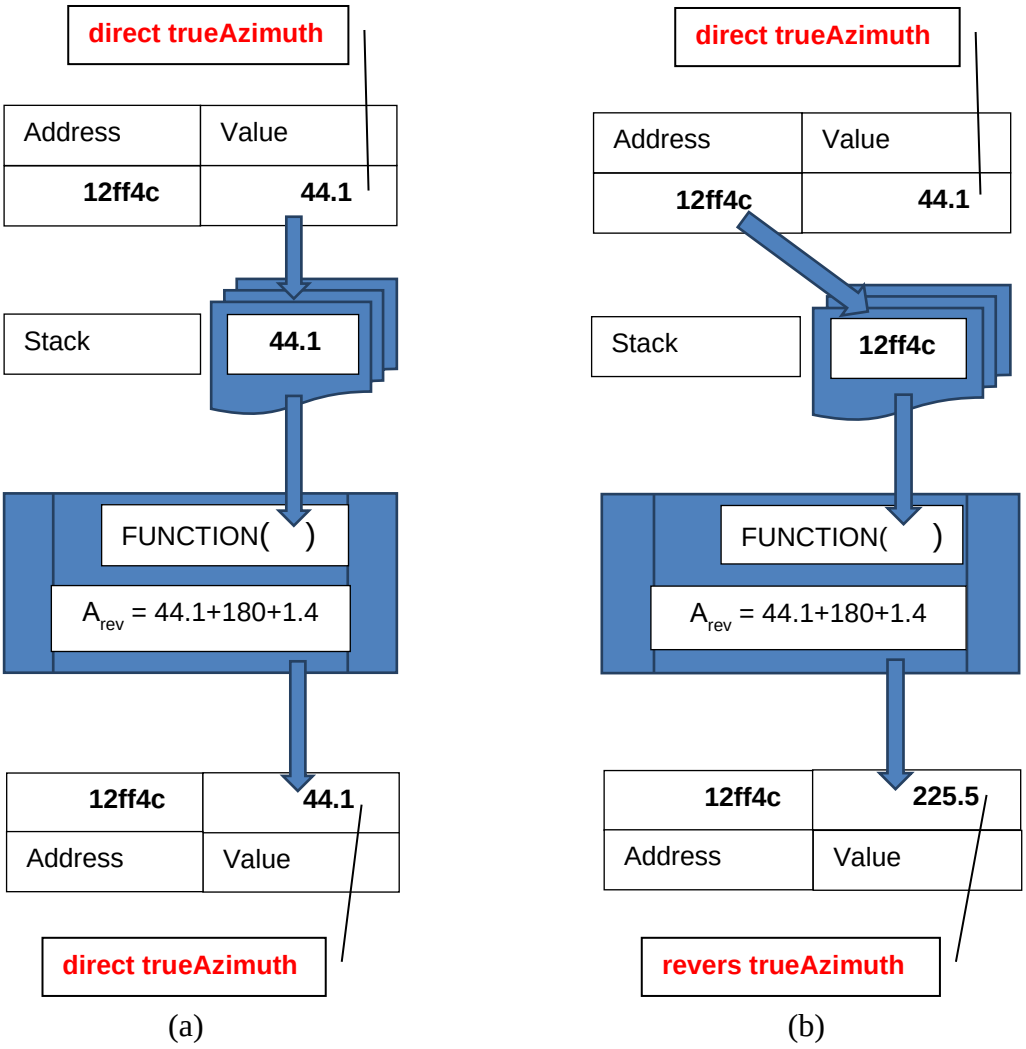

Fig. 2. The scheme for passing a parameter to a function in two ways: by value (a) and by address (b). The stack and function are grayed out. The arrows show the sequence of actions when passing a parameter to a function and the results

The address of the *trueAzimuth* parameter is passed to the *ChangingAzimuthByPointer* function (line 24). The function uses this pointer to the memory location where the parameter value is stored. The value that *ChangingAzimuthByPointer* assigns to the parameter remains after the function is completed. To understand why the change that the feature performed over the variable remained after it was completed, it has to be kept in mind that the *ChangingAzimuthByPointer* function has access to the variable memory cell. Using the pointer (memory addresses) from the function, it is possible to access the memory at the parameter address and thereby change the parameter value.
During the experiment with the program, it was shown that copying the parameter address from memory to the stack and passing it to the function leads to a change in the parameter value. In the *ChangingAzimuthByPointer* function, 180° and the value of *convergenceOfMeridian* are added to the value of the *trueAzimuth* variable. In this case, the address *trueAzimuth* is passed to the function and the variable is dereferenced there, using the dereference operator (*) to get the value of the variable at its address. As a result, the *trueAzimuth* variable will be 225.5°.
As regards the function *ChangingAzimuthByReference*, the *convergenceOfMeridian* value and 180° are subtracted out of the *trueAzimuth* variable, wherein the used reference *azimuth* variable *trueAzimuth*, which is established in line 12. The reason for computing the reverse azimuth in this function using a slightly different formula is that the previously executed *ChangingAzimuthByPointer* function irreversibly changed the direct azimuth of the A-B line to the reverse azimuth. Therefore the angle of the newly obtained direction is in the range from 180° to 360°. The reverse true azimuth in this case is computed by a different formula:

$$A_{rev} = A_{dir} - 180° + \gamma$$

It is also necessary to take into account the fact that in this case, the convergence of the meridians becomes "western" that is, and the value of the convergence of the meridians must be taken with a minus sign.
Also comparing lines 45 and 40, it is observed that the code:

```
azimuth -= 180 - convergenceOfMeridian;
```

is read easier than the code:

```
*trueAzimuth += 180 + convergenceOfMeridian;
```

Using a reference simplifies the code and makes it easy to read. Thus, the values of the parameters passed to the *ChangingAzimuthByPointer* and *ChangingAzimuthByReference* functions using a pointer and a reference were changed. However, the asterisk (*) operator does not appear in the reference and the function code becomes easier to understand. This is the main advantage of using references over pointers.

5. CONCLUSIONS

The program considered illustrates the features of passing of parameters to function by value and with the help of a pointer and a reference. The program can be used in the C++ programming course for cartographers and surveyors as well as for related specialties of earth sciences. The program introduces students to programming language constructions such as pointers, references, the rules of their declaration, and the use of the program on the example of a geodesic task of computing direct and reverse true azimuth of line.
Teachers of computer science and programming in high school should be prepared for the requests of students to study specific programming. For example, to teaching programming aimed at solving geodetic and cartographic problems. Such training would allow students to develop programming skills on the basis of solving specified tasks, namely, problems of geodesy and cartography.
The process of teaching the C++ programming language implemented at the Moscow University of Geodesy and Cartography, and the features of its adaptation to the requirements of geodetic science are briefly described. The special role of computer and information solutions in the academic process at the university is emphasized. On the example of a specific C++ training course, methods for solving problems that a lecturer and a teacher encounter while conducting the educational process in a higher engineering school are considered.

6. REFERENCES

Deitel H., Deitel P. (2006) C++ How to program, 5th Edition, Published by Prentice Hall, New Jersey, USA

Liberty J., Cashman M. (2002) SAMS Teach Yourself C++ in 10 Minutes, 2nd Edition, Published by SAMS, Indianapolis, USA.

Prata S (2011) C++ Primer Plus, 6th Edition, Published by Addison–Wesley, Reading, USA.

Stroustrup B. (1997). The C++ Programming Language. Special Edition, Published by Addison-Wesley, Reading, USA.

Zablotskii V.R. (2019) A New Approach to Teaching C++ Programming for Cartography Students by Means Training Programs with Emphasizing Cartography and Geodesy, Proc. Int. Cartogr. Assoc., 2, 155, https://doi.org/10.5194/ica-proc-2-155-2019.

BIOGRAPHICAL NOTES

Vladimir R. Zablotskii (b. 1953) graduated from the Lomonosov Moscow State University (soil–science faculty in 1976 and physics faculty in 1985). He has the degree of Candidate Science (Biology). He is now the Associate Professor on department of applied cosmonautics and photogrammetry at Moscow University of Geodesy and Cartography and Associate Professor on department of the Physics at Bauman Moscow State Technical University. He is author of more than 50 publications in the field of physics of soils, remote sensing and GIS, application of computer technologies in education and C++ programming for cartographers and geodesist.

CONTACTS

Vladimir Zablotskii
Moscow State University of Geodesy and Cartography,
Gorokhovskiy pereulok 4,
Moscow, 105064
RUSSIA
Tel. +7 9163801461
Email: zablotskii@miigaik.ru