We should accomplish the following steps in order to create a program in C++

1. Drink a cup of coffee :-) (we should be awake and aware of what we are doing).
2. Create a text file and put all the code of your program into this file. The file can have an extension (but it is not necessary) **.cpp**
3. You should run the program called compiler. This program parses our text file and try to check all syntax errors. After that it translates the code of our program into binary code consisted of zeros and ones the processor of our computer can understand. As an output of compiler we obtain a file with the same name as text file and **.obj** extension. Before the compiler goes in action the forward guard of it comes in field. It is a preprocessor program. It executes all preprocessor directives and macros (Visual Studio display them in blue). Some directives include the content of additional files into the code of our program. Another ones create some macros that preprocessor should unwind. Only after the preprocessor ended its work, the compiler starts to process the output of preprocessed source code of our program.
4. After the compilation process, the program called linker goes in action. It takes our **.obj** file and links to it the code of all library functions we use in our program plus additional information such as starting and initialization code of the program. If program is divided into few modules you can compile them separately (there will be few .obj files) and linker will link them together. This way **.exe** file is being created.
5. Now you can double-click it in Windows explorer to execute it.
6. If you use Integrated Development Environment Rapid Application Development tool as Visual Studio you can only use the main menu option *Build->Build Solution* to compile and link the program. After that you can only use the main menu option *Debug->Start Without Debugging* to execute the program.
7. Let’s create our first program. In Visual Studio click *File->New ->Project*, in the left pane choose Win32, in the right pane choose Win32 console Application, in the text field *Name* type the name of the program (for example *first*) and click OK and in the next window *Finish*. Then you will see the window with source code of your program.

```
//================================================================
#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{
      return 0;
}
//================================================================
```

C++ is a free style language. It means you can put in source code any number of white characters you want. White characters are spaces, enters (new line, caret return) and tabs. You can put any number of instructions on a single line and use any number of lines for a single instruction. The end of the instruction is determined by the semicolon. You should remember to put a semicolon after any instruction. There are some languages as Assembler or early versions of BASIC such that you should put any single instruction in new single line. In C++ you shouldn't.

C++ is a case sensitive language. It means that it distinguishes between uppercase and lowercase letters. That's why if we should type **main** we can't do it **Main** or **MAIN**.

Indentation is very helpful but is not obligatory as in Python language.

Comments

Comments are the pieces of the program source code that the compiler ignores. Comments give the convenient way to document the code. VC displays comments in green.

In C++ there are two types of comments.

So called one line comment starts from the double slashes // and ends at the end of line.

The so called multi-line comment starts from a sequence of two characters /* slash and star and stretches till the sequence of star and slash */.

We want to program in standard C++ without Microsoft extensions. You can learn about such extensions by your own, but it lies outside the scope of our interest. So, let's change this code a little bit to look like this one:

The simplest program, our first program.

```
//==================================================================
#include "stdafx.h"
#include <iostream>
using namespace std;

void main()
{     cout<<"Alice has a cat";
}
//==================================================================
```

The first two lines of code are preprocessor directives. Both of them include the code of some file into the source code of our program. It means that the hole contents of those two files will be included in our code. First of them is additional header file we should include in a case when we want create an application in a particular tool (in our example Microsoft Visual studio). In other tools (for example Borland C++ Builder we should include another/different header files). Second included file is IOSTREAM that is part of the standard C++ library that operates on streams. We need it in a case when we want to send a text to the standard output (to the monitor screen).

Header files can be included in four different ways.

1. As in classical C. That time there were no namespaces, that's why you shouldn't put directive USING NAMESPACE after this line:
   #include<stdio.h>
2. New way to include pure C libraries, the same as above:
   #include <cstdio.h>
3. The new way without extension. You should put directive that enables us to use the namespace right after it:
   #include <iostream>
   using namespace std;
4. If you want to include not the standard library file, but your own header file, you can do it without triangle brackets. Preprocessor will look for this file in the same directory as the file that is being processed:

# include "myown.h"

In a case when we are going to create a really big project we may have problems with the clashes of variables and function names: two or more programmers in our teem will use the same name for their variables or functions. To cope with this problem the namespases are used. It is quite new mechanism. All the variables created in the same namespace are visible inside this namespace. But outside namespace they are not visible. If you want to define a variable inside the namespace you should put into your source code an appropriate directive:

namespace name_of_namespace

If you want to use all the variables and functions created in a particular namespace you should put the directive

using namespace name_of_namespace

In the next line there is a definition of main function. We will talk about functions later. For now you should only know that the function is a peace of code you can create only once and use it many times in your program. Every function has its name, the type of the value it returns (void means that function returns no value at all) and the list of function formal arguments in brackets. Our function has no arguments.

main is a special function. It is the so called entry point of our program. When program starts, the first instruction of the program that is being executed is the first instruction of the **main** function.

The only instruction is `cout<<"Alice has a cat";` This instruction sends the text "Alice has a cat" to the standard output.

<u>Second program-creation of variables.</u>

Variables are the cells (or regions) of memory that programmer can reserve for the program data and give them names in order to have an easy access to them.

In C++, when you define the variable, you reserve the memory for it. You should give the name of the variable, type of the data that will reside in that variable and (optionally) you can initialize this variable - assign an initial value to the variable.

When you try to assign a value to the variable, compiler checks whether this value can be assigned to that variable and it allows for this operation only if the type of the variable and the type of assigned value are the same. It is very convenient for the programmer because it guards against the mistakes (you can't put the text "Alice has a cat" inside the integer variable). C++ is type-checking language.

You can't use white characters, polish characters, special characters (that can be typed using SHIFT) in the names of variables.

You can create variables inside the functions and outside them.

If you create a variable outside the function definition you create a global variable.

If you create a variable inside the function definition you create a local or automatic variable.

Every program consists of 4 different pieces in memory (in one piece but logically these pieces are different): data segment in which all global variables reside, program code segment where all instructions reside (const values are located in the code segment). The third piece of program is the stack. It is a special region of memory where all automatic variables are created. The management of the stack is very quick, but the amount of memory reserved for the stack is constant and can't be increased during the program execution. That's the explanation why there are attacks of buffer/stack overflow exist. The fourth memory segment called heap can be used for dynamically created objects.

In C++ all global variables are initialized using zero values, but automatic variables (created on the stack are not. It is said that the trash (or accidental values) resides in such variables.

When you use the variable (use it in calculations or print on the screen) the compiler doesn't check whether you assign a value to the variable or not. And this is a source of many programming bugs. Be careful! Initialize all your automatic variables!

```
//=================================================================
#include "stdafx.h"
#include <iostream>
using namespace std;
int a;
void main()
{int b;
cout<<a<<endl<<b<<endl;
}
//=================================================================
```

In this example variable *a* automatically obtained a value 0, but variable *b* has a trash inside!

endl means end of line

<u>Data types.</u>

In C++ we can use these data types for variable creation:

int, short int, long int to hold an integer values. **Unsigned** or **signed**. By default **signed**.

float, double, long double to hold a real value,

char to hold a single character,

string to hold text values; to work with this data type you need to use #include<string> directive,

bool for logical values. Zero is treated as logical false. Every other value is treated as a logical true.

When you assign a value to the **char** variable you should put the character into single apostrophes.

When you assign a value to the **string** variable you should put the text in quotation marks

```
//=================================================================
#include "stdafx.h"
#include <iostream>
using namespace std;
void main()
{
char var1;// Definition without initialization!
int var2;
var2=123;
char var3 = 'A', var4 = 'Z';// Simultaneous definition & initialization:
long int var5 = 100L,
var6 = 150l;
short int var7= 200;
float var8 = 3.14159;
double var9 = 6e-4;// Exponential notation:
```

```
int var10=010;//octal
int var11=0x10;//hexadecimal
cout<<var1<<endl<<var2<<endl<<var3<<endl<<var4
<<endl<<var5<<endl<<var6<<endl<<var7<<endl<<var8
<<endl<<var9<<endl<<var10<<endl<<var11<<endl;
}
//================================================================
```

Sizeof operator

In C++ the type of a variable decides about the size of the memory reserved for this variable. But this size is not the same for every operating system and hardware. That's why there is an operator **sizeof()** that returns the number of bytes that particular variable occupies in memory.

```
//================================================================
#include "stdafx.h"
#include <iostream>
using namespace std;
void main()
{
char var1;// Definition without initialization!
int var2;
var2=123;
char var3 = 'A', var4 = 'Z';// Simultaneous definition & initialization:
long int var5 = 100L,
var6 = 150l;
short int var7= 200;
float var8 = 3.14159;
double var9 = 6e-4;// Exponential notation:
int var10=010;//octal
int var11=0x10;//hexadecimal
cout<<sizeof(var1)<<endl<<sizeof(var2)<<endl<<sizeof(var3)<<endl<<sizeof(va
r4)
<<endl<<sizeof(var5)<<endl<<sizeof(var6)<<endl<<sizeof(var7)<<endl<<sizeof(
var8)
<<endl<<sizeof(var9)<<endl<<sizeof(var10)<<endl<<sizeof(var11)<<endl;
}
//================================================================
```

Operations on strings

```
//================================================================
#include "stdafx.h"
#include <string>
#include <iostream>
using namespace std;
void main()
{
string s1 = "Alice has a cat,"; // Initialized
string s2, s3; // Empty strings
string s4("and TV set"); // Also initialized
s2 = "dog"; // Assigning to a string
s3 = s1 + " " + s2; // Combining strings
s3 += " radio "; // Appending to a string
cout << s3 + s4 + " !" << endl;
}
//================================================================
```

Obtaining values from the user (cin and >> operator)

```
//=================================================================
#include "stdafx.h"
#include <iostream>
using namespace std;
void main()
{
      cout<<"Input an integer value";
    int i;
    cin>>i;
    cout<<"Your value is: "<<i<<endl;
      cout<<"Input a real value";
      float f;
      cin>>f;
    cout<<"Your value is: "<<f<<endl;
      cout<<"Input a single character";
      char c;
      cin>>c;
    cout<<"Your character is: "<<c<<endl;
}
```