Mechatronic Engineering

Object Oriented Programming and Software Engineering
Laboratory instruction 1
C++ introduction

AGH Krakow, 2021

Contents

1 First steps into C++ programming

1.1 Creating *.cpp files and compiling

In order to write the first program in C ++ we will use the UNIX console and the Midnight Commander text editor. To start Midnight Commander Editor, enter the following in the console:

```
mcedit -c filename.cpp
```

A * .cpp file will be created on the hdd (in current directory), which is the source file of the program written in C ++.

In order to compile the source code into an executable file, we will use the g ++ compiler.

```
g++ filename.cpp -o filename
```

To run the program, enter (being in the program directory):

```
./filename
```

1.2 Source code basics

To create a program in C++ it is necessary to use a library loading directive **#include** (similar as in C language).

#include <**name**> - is used to inform the preprocessor that the included library file is located in the standard folder with header files of the compiler. **#include ”name”** is used to inform the preprocessor that the included library file is located in the same folder as the file containing the #include directive.

In C++ the standard input and output library is called **iostream**.

The next important part of the C++ source file is the **main()** function. All the major instructions for the program should be stored in it. The format of main function declaration goes as follows:

Example:

```
#include <iostream>

int main() {
   std::cout << "Hello world! \n";
   std::cout <<"Again" << "Hello"
         << "world!" <<
"\n"; return 0;
}
```

The above example is a very simple, working program written in C ++. Its function is to display a string on the screen.

In the above example one could see some new instructions. In C, for the purpose of displaying text on the screen **printf()** was used. In C++ the function responsible for that is **std::cout**.

```
std::cout << "Hello World! \n";
```

In C, **scanf()** was used to input the necessary data into any variables. C++ uses **std::cin**.

```
std::cin >> name_of_the_variable;
```

Example:

```
#include <iostream>

int main(){
   int age;
         std::cout << "Enter your age: ";
         std::cin >> age;
         std::cout << "\n";
         std::cout << "You are " << age << " years old"
       << std::endl;
return 0;
}
```

2 Variables

The data in C++ is stored (in a similar way as in case of C language) in variables. Variables can be declared as in C, by providing the type and the variable name.

In C++ the variable types can be divided in following categories:

- fundamental types;
- derived types;
- built-in types;
- user defined types.

Fundamental types:

Types used for real numbers:
short int; int; long int; long long int;

Types used for alphanumerical characters:
char;

Types used for floating point numbers:
float; double; long double;

In C++ it is possible to declare and define variables on the go.

Example:

```
#include <iostream>

int main() {
std::cout << "Let’s calculate the area
of a square\n"
     << "To do so, we will need the length of the side of a
     square, \n"
     << "which, unfortunately, is not available for this
     software yet.\n"
     << "Type in the length of the square’s
   side: "; int length;
     std::cin >> length;
     std::cout << "\n";
     int p;
     p = length * length;
     std::cout << "The area of a square is
   equal to: " << p << "\n";
return 0;
}
```

3 Control instructions

3.1 Conditional instruction if.

As the name suggests, the if instruction is used to check if a provided condition is full field. Two forms of it exist:

```
if(expression) instruction1;
```

and

```
if(expression) instruction1;
else       instruction2;
```

In the first case, the value of the expression is checked. If the value is not zero, the instruction 1 is executed. If the value is zero or negative, then the instruction1 is omitted.

In the second case, the value of the expression is also checked. If the value is not zero, the instruction 1 is executed. If the value is zero or negative, program runs instruction2.

For the purpose of running more than one instruction, an instruction block is necessary. An instruction block is a set of instructions between { and } brackets.:

```
if(expression) {
     instruction1;
     instruction2;
}
```

It is possible to create a multi variant if instruction:

```
if(expression) instruction1;
else if(expression2)
instruction2;
else if(expression3)
instruction3;
```

Example:

```
#include <iostream>

int main(){
   int a=1, b=2,
   c=3;

   if (a) std::cout << "Variable a is not equal to 0 and its
 value"
      <<" is: " << a << "\n";
   if (a > b)std::cout << "it will not display\n";
   else std::cout << a << " is smaller than " << b
   << "\n";
   if (c < b) std::cout << "it will not display\n";
   else if (b = 2) {
        b++;
       std::cout<< "b = " << b << "\n";
   }
return 0;
}
```

3.2 While loop.

The way while loop works is very simple. The expression value is checked at the beginning of the loop. If its equal zero or less, nothing happens (instruction1 is not executed). If the value is greater than zero, the instructions in the loop's instruction block are executed. Then the value of the expression is checked again. The loop runs while the expression has a positive value. The expression is always checked at the beginning of the loop cycle.

```
while (expression) instruction1;
```

Example:

```
#include <iostream>
using namespace std;

int main () {
int age=0, years=0;
   cout << "How old are
   you? \n";
   cin >> age;
```

```
    cout << endl;
    while (age < 24) {
       age++;
       years++;
    }

    if (age < 25) {
       cout << "Theoretically in " << years;
       if (years == 1) cout << " years";
       else if (years > 1 && years < 5) cout << " years";
       else if (years >= 5 || years==0) cout << " years";
       cout << " you will graduate. \n";
    }
    else cout << "So old and still studying at the
    University…. \n";
return
0;
}
```

3.3 Do while loop.

Do while loop works very similar to while loop. The main difference is the time when the expression value is checked. In the do while loop, the expression is checked at the end of the loop cycle. Because of that, a do while loop always executes at least once.

```
do instruction1 while(expression);
```

Example:

```
#include <iostream>
using namespace std;

int main() {
    int i=1;
    do {
       cout << "Value of i is: " << i << endl;
       i++;
    } while(i<=4 && i>=2);
return 0;
}
```

3.4 For loop.

```
for (initialization; expression; incrementation)
instruction1;
```

Example:

```
for (i = 0; i < 10; i++)
{
cout << i;
}
```

Initialization – this instruction is executed before the first cycle of the loop. **Expression** – the value of expression is calculated before every run of the loop. If the value is true, the loop executes.
Incrementation – incrementation that is executed at the end of every loop cycle.

Number of instructions provided in the initialization and incrementation may be greater than one. They have to be separated with the use of a colon. The incrementation, expression and initialization instructions can be left blank, however the semicolons that divide those, have to be provided. If the expression is left blank it is treated as always being true.

Example:

```
#include <iostream>
using namespace std;

int main() {
   int i=0, hmany=0;
   cout << "Enter the number of subsystems: ";
   cin >> hmany;
   cout << endl;
   cout << "Begin to scan the subsystems" << endl;
   for (i = 1; i <= hmany; i++) {
      cout << "Subsystem " << i << " scanned" << endl;
   }
   cout << "Scan finished. No anomalies found.\n";
return 0;
}
```

3.5 Switch instruction.

The switch instruction is used for multi variant operations.

```
switch (expression)
{
    case con1:
    instr1;
    break; case
    con2:
    instr2;
    break;
    default:
    instr3; break;
}
```

When the expression value is obtained and it corresponds to the value of one of the provided cases, then the instructions in a specific case are executed until the brake instruction is found. If the value doesn't correspond to any case, then the default case is executed.

Example:

```
#include <iostream>
using namespace std;

int main() {
   int i = 3;
   switch(i) {
   case 1: cout << "Value is equal to 1 \n"; break;
   case 2: cout << "Value is equal to 2 \n"; break;
   case 3: cout << "Value is equal to 3 \n"; break;
   case 4: cout << "Value is equal to 4 \n"; break;
   default: cout << "Values is different than 1,2,3,4 \n";
   }
return 0;
}
```

4 Functions

Functions in C++ may be called a subprograms. Function is used to define a set of user defined instructions, that can return values set by users to the main program. To invoke a function it is necessary to input function type, name and arguments used in it.

Example:

```
#include <iostream>
using namespace std;

float rect_area (float side1, float side2);

int main() {
   float a, b, area;
   cout << "Enter the length of the first side: ";
   cin >> a;
   cout << endl << "Enter the length of the second side: ";
   cin >> b;
   cout << endl;
   cout << "Your rectangular has sides of lengths equal to:
   a = " << a;
   cout << ", b = " << b;
   cout << endl;
   area = rect_area(a,b);
   cout << "The area of your rectangular is: " << area <<
   endl;
return 0;
}

float rect_area (float side1, float side2) {
   float result;
   result = side1* side2;
   cout << "Calculating the area...\n\n";
return result;
}
```

Usually at the function end a return instruction is used. It is used to return a value from function to the place in the main code where it was executed. In the example program, the function calculating the area of a rectangle was linked to a variable field. It means, that the variable value will be equal to the variable returned by the function. The variable and function that are linked together must have the same data type. Void function is an exception to this rule. Void functions can’t return any data.

Stack
Stack is a kind of cache memory, that stores variables defined in the function.

Function arguments
Arguments sent to the function (actual arguments) are only copies of used variables or constants. They are saved on the stock. Any operations performed on the arguments doesn’t affect the original variables. There is a way of sending the variables directly to the function. This method is called sending arguments by references.
Example:

```
#include <iostream>
using namespace std;
void zer (int valu, int &ref);
void nzer (int valu, int ref);

int main () {
   int a = 44, b = 77;
   cout << "Before executing function nzer:\n";
   cout << "a = " << a << ", b = " << b << endl;
   nzer (a,b);
   cout << "After the execution of function nzer:\n";
   cout << "a = " << a << ", b = " << b << endl;
   cout << "Before executing function zer:\n";
   cout << "a = " << a << ", b = " << b << endl;
   zer (a,b);
   cout << "After the execution of function zer:\n";
   cout << "a = " << a << ", b = " << b << endl;
return 0;
}
void zer (int valu, int &ref)
{
   cout << "\tIn function zer before zeroing: \n";
   cout << "\tvalu = " << wart << ", ref = "
   << ref << endl;
   wart = 0;
   ref = 0;
   cout << "\tIn function zer after zeroing: \n";
   cout << "\tvalu = " << wart << ", ref = "
   << ref << endl;
} void nzer (int wart, int ref)
{
   cout << "\tIn function nzer before zeroing: \n";
   cout << "\tvalu = " << wart << ", ref = "
   << ref << endl;
   wart = 0;
   ref = 0;
   cout << "\tIn function nzer after zeroing: \n";
   cout << "\tvalu = " << wart << ", ref = "
   << ref << endl;
}
```

Tasks

In base of provided topics in the following instruction, please create a simple user interface for a vending machine:

Program requirements:

1. Software displays the list of provided products (at least five) with the amount of the stock. The user should be able to pick a product.
2. After the product has been picked, the user should be able to input quantity of the purchase. If the quantity exceeds the stock amount, the user is asked to input a proper value into the program.
3. After the transaction is finished, program returns to displaying the list of products with updated stock value.
4. Program must be equipped with a secret code unknown to user. After inputting the code, the program should stop running.
5. For the purpose of this program at least two functions must be used (additional to main() function).