Mechatronic Engineering

Object Oriented Programing and Software Engineering
Laboratory instruction 17
C++ introduction

AGH Kraków, 2022

Materials created for educational purposes.
Dedicated for students attending Software Engineering course.
Author would apreaciate any feedback regarding errors of any kind found in the instruction script.
Please report those to the following email address: danielt@agh.edu.pl

Contents

1 Recursion **4**

2 Operators **5**
2.1 Arithmetic operatosr 5
2.2 Assignment operators 6
2.3 Relational operators 6
2.4 Logical operators 7
2.5 Bitwise operators 8
2.6 Other operators 9

3 Operators overloading **9**

4 Namespaces **13**

1 Recursion

Recursion in simplest way occures when a function calls itself directly or indirectly. It is a usefool tool that simplifies some complicated algorythms.

```
void function();
...
int main(){
   ...
   function();
   ...
}

void function(){
   ...
   function();
   ...
}
```

With the use of recursion a programmer obtains much shorter and cleaner code. Moreover recursion is required in some promblems concerning advanced algorythms and complicated data structures. Unfortunately there are some drawbacks to the use of recursion. It uses lot of stack space and needs more processing time. Recurence can also be more difficult to handle when it comes to debugging proces of ones program.

Example:

```
#include <iostream>

int fact(int); //function that calculates factorial of a given
    number

int main(){
    int n, result;

    std::cout << "Enter a non-negative number: ";
    std::cin >> n;

    result = fact(n);
    std::cout << "Factorial of " << n << " = " << result;
    return 0;
}
```

```
15
16 int fact(int n){
17     if (n > 1) {
18         return n * fact(n - 1);
19     } else {
20         return 1;
21     }
22 }
```

2 Operators

Operators a very common symbols in C++ and are used to perform operations on variables annd data structures. There are six types of operators in C++.

2.1 Arithmetic operatosr

This kind of operator is used to perform arithmetic operatios o data.

```
1 a+b-c*d
```

List of atiyhmetic operators

Operator	Operation
+	Addition
−	Substraction
∗	Multiplication
/	Division
%	Modulo

Example:

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5    int a=3, b=2, c=7;
6    cout << "a+b= " << (a+b) << endl;
7    cout << "a*c= " << (a*c) << endl;
8    cout << "c-b= " << (c-b) << endl;
9 }
```

2.2 Assignment operators

In C++, assignment operators are used to assign values to variables
List of assignment operators

Operator	Implementation	Equivalet
$=$	$a = b$	$a = b$
$+=$	$a+ = b$	$a = a + b$
$-+$	$a- = b$	$a = a - b$
$*=$	$a* = b$	$a = a * b$
$/=$	$a/ = b$	$a = a/b$
$\%=$	$a\% = b$	$a = a\%b$

Example:

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

int main(){
   int a=3, b=2, c=7;
   cout << "a= " << a << endl;
   cout << "b= " << b << endl;
   cout << "c= " << c << endl;
   for(int i=0; i<5;++i){
      a+=i
      cout << "a= " << a << endl;
   }
   a*=c;
   cout << "a= " << a << endl;
   a-=b;
   cout << "a= " << a << endl;
}
```

2.3 Relational operators

A relational operator is used to check the relationship between operands. The outcome of the checking proces is, if the relation is true, returned 1, if the relation is false, it returns 0

List of relational operators

Operator	Meaing
==	Is equal to
! =	Is not equal to
>	Greater than
<	Less then
>=	Greater than or equal to
<=	Less than or equal to

Example:

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

int main(){
   int a=3, b=2, c=7;
   cout << "a= " << a << endl;
   cout << "b= " << b << endl;
   cout << "c= " << c << endl;
   while(c>a){
      a+=b
      cout << "a= " << a << endl;
      if(a==c)cout << "c= " << c << endl;
   }
   if(a==c)cout << " after while loop c= " << c << endl;
}
```

2.4 Logical operators

Logical operator is used to check if an expression is true or false. If the expression is true, it returns 1 and if the expression is false, it returns 0.

List of logical operators

Operator	Implementation	Equivalet
&&	expressionA&&*b*expressionB	Logical AND, True only if all the operands are true
||	expressionA||expressionB	Logical OR, True if at least one operand is true
!	!expression	Logical NOT, True only if operand is false

Example:

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

int main(){
   int a=3, b=2, c=7;
   cout << "a= " << a << endl;
   cout << "b= " << b << endl;
   cout << "c= " << c << endl;
   do{
      a+=b
      cout << "a= " << a << endl;
      if(a==c)cout << "c= " << c << endl;
   }while(c!=a)
   if(a==c)&&(c*b>=12)cout << " after while loop c= " << c << endl;

}
```

2.5 Bitwise operators

In C++, bitwise operators are used to perform operations on individual bits. They can only be used with char and int data types.

List of Bitwise operators

Operator	Description
&	Binary AND
|	Binary OR
∧	Binary XOR
∼	Binary one's complement
<<	Binary shift left
>>	Binary shift right

The bitwise operators are only mentioned and won't be discussed in detail during this course.

2.6 Other operators

There are other kinds of operators in C++, that won't be discussed during this course. They can be easily found ib books, publications and on-line, depending on the needs of the application.

3 Operators overloading

You can redefine or overload most of the built-in operators available in C++. Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

```
1 type operator overloaded_operator (cnonst type);
```

Most overloaded operators may be defined as ordinary non-member functions or as class member functions.
The declaration of an overloading can look as follows:

Class definition:

```
1 Matrix& operator -(const Matrix& a);
```

Non-member definition:

```
1 int operator +(const int& a, const int& b);
```

The example bellow presents a program that provides operations on matrixes, and the whole process is optimised with the use of operator overloading.

Example:

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include<unistd.h>
using namespace std;


class Matrix {
public:
  unsigned int row;
  unsigned int column;
  int **matrix;

  Matrix() {}

  Matrix(unsigned int r, unsigned int c): row(r), column(c){
    //srand(time(NULL));
   matrix = new int*[r];
   for(unsigned int i=0; i<r; i++) {
      matrix[i] = new int[c];
      for(unsigned int j=0; j<c; j++) {
         this->matrix[i][j] = (rand()%10)+1;
      }
   }
  }

  Matrix& operator +(const Matrix& a){

   if(a.row == this->row && a.column == this->column) {
      for(unsigned int i=0; i<a.row; i++) {
         for(unsigned int j=0; j<a.column; j++) {
            this->matrix[i][j] += a.matrix[i][j];
         }
      }
      return *this;
   }
   else {
      printf("Niezgodne wymiary Matrixy\n");
      return *this;
   }
}
```

```
42
43      Matrix operator -(const Matrix& a){
44
45        if(a.row == this->row && a.column == this->column) {
46           for(unsigned int i=0; i<a.row; i++) {
47              for(unsigned int j=0; j<a.column; j++) {
48                 this->matrix[i][j] -= a.matrix[i][j];
49              }
50           }
51           return *this;
52        }
53        else {
54           printf("Niezgodne wymiary macierzy\n");
55           return *this;
56        }
57
58      }
59
60    friend std::ostream& operator<<(std::ostream& out,const Matrix&
        m);
61  };
62
63  Matrix operator +(const Matrix& a, const Matrix& b){
64
65     if(a.row == b.row && a.column == b.column) {
66        Matrix c(a.row,a.column);
67        for(unsigned int i=0; i<a.row; i++) {
68           for(unsigned int j=0; j<a.column; j++) {
69              c.matrix[i][j] = (a.matrix[i][j] + b.matrix[i][j]);
70           }
71        }
72        return c;
73     }
74     else {
75        printf("Niezgodne wymiary macierzy\n");
76        return a;
77     }
78
79  }
80
81  Matrix operator -(const Matrix& a, const Matrix& b){
82
83     if(a.row == b.row && a.column == b.column) {
```

```
84        Matrix c(a.row,a.column);
85        for(unsigned int i=0; i<a.row; i++) {
86           for(unsigned int j=0; j<a.column; j++) {
87              c.matrix[i][j] = (a.matrix[i][j] - b.matrix[i][j])%3;
88           }
89        }
90        return c;
91     }
92     else {
93        printf("Niezgodne wymiary macierzy\n");
94        return a;
95     }
96
97  }
98
99  Matrix operator *(const Matrix& a, const int& b) {
100   Matrix c(a.row,a.column);
101   for(unsigned int i=0; i<a.row; i++) {
102     for(unsigned int j=0; j<a.column; j++) {
103       c.matrix[i][j] = a.matrix[i][j] * b;
104     }
105   }
106   return c;
107 }
108
109 Matrix operator *(const int& b, const Matrix& a) {
110   Matrix c(a.row,a.column);
111   for(unsigned int i=0; i<a.row; i++) {
112     for(unsigned int j=0; j<a.column; j++) {
113       c.matrix[i][j] = a.matrix[i][j] * b;
114     }
115   }
116   return c;
117 }
118
119 std::ostream& operator<<(std::ostream& out,const Matrix& m){
120
121     out <<"{";
122     for(unsigned int i=0; i<m.row-1; i++) {
123       out << "{";
124       for(unsigned int j=0; j<m.column-1; j++) {
125         out << m.matrix[i][j] << ",";
126       }
```

```
127       out << m.matrix[i][m.column-1] << "},"<<endl;
128     }
129     out <<"{";
130     for(unsigned int i=0; i<m.column-1; i++) {
131       out  << m.matrix[m.row-1][i] << ",";
132     }
133     out << m.matrix[m.row-1][m.column-1] << "}";
134     out << "}";
135 
136     return out;
137 
138   }
139 
140 int main() {
141   srand(time(NULL));
142   Matrix a=Matrix(2,3);
143   Matrix b=Matrix(2,3);
144   int c = 4;
145 
146   cout << "Matrix A:" << endl;
147   cout << a << endl<< endl;
148 
149   cout << "Matrix B:" << endl;
150   cout << b << endl<< endl;
151 
152   cout << "Result of A + B:" << endl;
153   cout << a+b << endl<< endl;
154 
155   cout << "Result of A - B:" << endl;
156   cout << a-b << endl<< endl;
157   cout << "Result of C * A:" << endl;
158   cout << c*a << endl<< endl;
159   cout << "Result of B * C:" << endl;
160   cout << b*c << endl<< endl;
161 }
```

4 Namespaces

Namespaces define a scope with a specific name. They are used to group related names and avoid conflicts between similar names in the code.

```
1 namespace nname_of_namespace{
```

```
    ...
    elements
    ...
}
```

Namespaces are open-ended, which means they can be extended freely using multiple declarations.

The name can be used outside the space that declares it, you should first specify which space is to be used by giving its name:

```
name_of_namespae::function_of_namespace();
```

Names that are not declared inside any namespaces belong to the global namespace. To simplify access to namespace elements and to eliminate the need to write the name of the space that is being applied, you can apply the *using* directive

Example:

```
#include <iostream>

namespace conversion{

    double div=0.39370;

    template <typename T> T cmToInch(T a){
        return(a*div);
    }

    template <typename T> T inchToCm(T a){
        return(a/div);
    }
}

void disp(std::string a){
    std::cout<<"Disp function operational! "<< a <<std::endl;
}

namespace display{
    using namespace std;
    void disp(string a);
}

int main(){
```

```
26    int a=20, b=0 ;
27    float f = 6.5, c=0, d=0;
28    b=conversion::cmToInch(a);
29    std::cout<<"Converting a to inch: "<< b <<std::endl;
30    {
31       using namespace conversion;
32       c=cmToInch(a);
33       std::cout<<"Converting a againt to inch: "<< c <<std::endl;
34    }
35    d=conversion::inchToCm(f);
36    std::cout<<"Converting f to cm: "<< d <<std::endl;
37    display::disp("Let's display all conversions!");
38    std::cout<<std::endl<<"a = "<<a<<std::endl<<"b =
          "<<b<<std::endl<<"c = "<<c<<std::endl<<"d = "<<d<<std::endl;
39
40    disp("non namespace function");
41    display::disp("namespace disp function");
42
43    using namespace display;
44    display::disp("now a cover of the function"); //if the name of
          the namespace is not added generates an error due to the
          function overloading principles
45
46    return 0;
47 }
48
49 void display::disp(string a){
50    cout << a << endl;
51 }
```