Mechatronic Engineering

Object Oriented Programing and Software Engineering
Laboratory instruction 12
C++ introduction

AGH Kraków, 2020

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

Spis treści

1 Templates

Templates allow the creation of generic code partly independent of the type of data used. Using templates one can generate both functions and classes.

1.1 Function templates

Syntax for a function template is presented bellow:

```
template <typename type_var> type_var function_name(type_var
    arg_name) {
   //function body
}

//first line can be broken after '>'

template <typename type_var>
type_var function_name(type_var arg_name) {
   //function body
}
```

The template keyword is necessary to let the compiler know that a template is being created here. Another necessary keyword is typename (can be replaced with the class keyword) type_var followed by specifies the name to be used where the data type appears. Instead of arg_name, the function argument name is inserted.

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

template <typename T> T print(T a){
   cout << a << endl;
}

template <typename T> T square(T a){
   print(a*a);
}

int main(){
   int first = 4;
   float second = 2.8;
   cout << "\t\t\tProgram calculates square of 4 (int) and 2.8
```

```
        (float) numbers with same function\n";
16      square(first);
17      square(second);
18
19   return 0;
20 }
```

The above program will return some warnings because the functions aren't returning anything. It can be corrected by setting a void type for the functions:

```
1  #include <iostream>
2  using namespace std;
3
4  template <typename T> void print(T a){
5     cout << a << endl;
6  }
7
8  template <typename T> void square(T a){
9     print(a*a);
10 }
11
12 int main(){
13    int first = 4;
14    float second = 2.8;
15    cout << "\t\t\tProgram calculates square of 4 (int) and 2.8
          (float) numbers with same function\n";
16       square(first);
17       square(second);
18
19    return 0;
20 }
```

1.2 Class templates

Syntax for a class template is presented bellow:

```
1  template <class type_var> class class_name{
2     //class body
3  } ;
4
5  //first line can be broken after '>'
6
```

```
template <class type_var>
class class_name{
   //class body
} ;
```

To define a template class object, use the following syntax:

```
class_name <type> object_name;
```

In this case, type referes to the specific type the class will operate on. Below is an example of how class templates can be used:

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


template <class T>
class numberOperations{
   T var;
public:
   numberOperations() { }
   void print(){
      cout << var<< endl;
   }
   void square(){
      var = var*var;
   }
   void set();
};

int main(){
   numberOperations<int> intVar;
   numberOperations<double> doubleVar;

   cout << "\t\t\tProgram calculates square of int and float
       numbers\n";
   cout<<"Enter integer number:\n";
   intVar.set();
   intVar.print();
   intVar.square();
   intVar.print();
   cout<<"Enter floating point number:\n";
   doubleVar.set();
   doubleVar.print();
```

```
32    doubleVar.square();
33    doubleVar.print();
34
35    return 0;
36 }
37
38 template <class T>
39 void numberOperations<T>::set(){
40    cin>>var;
41 }
```

2 Vectors

So far, several ways to store data have been presented. Until now mainly arrays have been used to store data groups, in this section containers otherwise known as vectors will be presented.
Vectors are actually specific dynamic arrays that have the ability to resize (adding and remmoving fields). To use vectors one will need to use the vector library. The following are the syntaxes of vector declarations:

```
1 std::vector <data_type> vector_name;
```

Po stworzeniu pustego wektora dostępne jest kilka użytecznych metod pozwalających na zarządzanie kontenerem:
push_back() - dodaje element na końcu,
size() - zwraca rozmiar wektora,
capacity() - zwraca rozmiar wektora w zarezerwowanej pamięci.

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main(){
7      vector < int > container;
8
9      for( int i = 0; i < 10; i++ ) {
10        if (i==0) {cout << "Size: " << container.size() << "
              Capacity: " << container.capacity() << endl;}
11         container.push_back( i );
```

```
12          cout << "Size: " << container.size() << " Capacity: " <<
                container.capacity() << endl;
13      }
14      return 0;
15  }
```

The push_back method allows you to add elements only at the end of the vector, from the memory perspective it is the fastest way. It is possible to add data to any place, but this involves some operations. The data in the containers are stored in a continuous block, adding an element in the middle of the vector will require moving some of the data in memory to free up space for that added one.

```
1   #include <iostream>
2   #include <vector>
3
4   using namespace std;
5
6   int main(){
7       vector < int > container( 10, 0 ); // this creates ten elements
            filled with zeros
8       container.insert( container.begin(), 1 );
9       container.insert( container.end(), 3, 6 );
10      container.insert( container.begin() + 3, 2 );
11
12      for( int i = 0; i < container.size(); ++i )
13          cout << container[ i ] << ' ';
14        cout <<endl;
15     return 0;
16  }
```

As one can see in the above example, adding elements is handled by the **insert()** method. You can add several elements at once with it. This method does not specify the place of insertion based on the index (as was the case with arrays), but with an iterator. The vector iterator behaves similarly to a pointer. It is possible to create variables for iterators using the following syntax:

```
1   std::vector <data_type>::iterator iterator_name = variable_name;
```

```
1   #include <iostream>
2   #include <vector>
3
```

```
using namespace std;

int main(){
    vector < int > container( 10, 0 );
    container.insert( container.begin(), 1 );
    container.insert( container.end(), 3, 6 );
    container.insert( container.begin() + 3, 2 );

    for( int i = 0; i < container.size(); ++i )
        cout << container[ i ] << ' ';
      cout <<endl;

   vector<int>::iterator it;
   for( it=container.begin(); it!=container.end(); ++it ){
     cout<< *it << ' ';
    }
    cout<<endl;

   return 0;
}
```

To remove vector elements, the following methods are used:
pop_back() - deletes the last element,
clear() - clears the vector of all data,
erase() - removes the range of any elements based on an iterator or a pair of them (a single iterator indicates a specific element, pair range).

```
#include <iostream>
#include <vector>

using namespace std;

int main(){
    vector < int > container( 10, 0 );
    container.insert( container.begin(), 1 );
    container.insert( container.end(), 3, 6 );
    container.insert( container.begin() + 3, 2 );

    for( int i = 0; i < container.size(); ++i )
        cout << container[ i ] << ' ';
      cout <<endl;

```

```
16    container.erase( container.begin() );
17     container.erase( container.begin() + 2, container.begin() + 4 );
18     container.erase( container.begin() + 4, container.end() );
19
20    vector<int>::iterator it;
21    for( it=container.begin(); it!=container.end(); ++it ){
22      cout<< *it << ' ';
23     }
24     cout<<endl;
25
26    return 0;
27  }
```

Task

Based on the informations provided in this manual, please improve the simple RPG caracter creation program.

Program requirements:
1. Create a battle simulation between the hero and monsters.
2. Battle mechanics should be created as a template class
3. Add 'experience' field into hero and monster class (for numerical values storage). After a fight if hero wins its ammount of experience should grow accordingly to the value of experience stored in particuar monster that was slain. If Hero looses it should be decreased by that value.
4. Add 'level' to the hero class. It should grow or fall according to the experience values gathered by the hero.