Programowanie obiektowe

Zajęcia 7. Szablony, typy generyczne, lambda

Zadanie 1. Szablony i lambda w C++

1. Utwórz funkcję `maximum`, która będzie zwracać większy z dwóch przekazywanych obiektów.

```
template <typename T>
T maximum(T const& a, T const& b){
  return a > b ? a : b;
}

```

2. Przetestuj kod. Zastanów się dlaczego niektóre wywołania powodują błąd kompilacji.

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

int main(){
  cout<<maximum(6, 7)<<endl; //dedukcja typu
  cout<<maximum(6.3, 7.3)<<endl;
  cout<<maximum(6.5, 7)<<endl;
  cout<<maximum<double>(4, 5)<<endl; //bez dedukcji typu
  cout<<maximum<double>(4.1, 5.7)<<endl;
  cout<<maximum<int>(4.6, 3)<<endl;
  cout<<maximum<double>(4.6, 3)<<endl;
  cout<<maximum<bool>(4.6, 3)<<endl;
  cout<<maximum<string>(4.6, 3)<<endl;
```

```
14   cout<<maximum<string>("aaa", "bb")<<endl;
15 }
```

3. Stwórz klasę zawierającą jedną zmienną typu int:

```
1  class X{
2  private:
3    int x;
4  public:
5      X(){}
6    X(int _x){
7      this->x = _x;
8    }
9    int getX() const{
10     return x;
11   }
12   friend ostream& operator << (ostream& os, X const & x){
13     os << x.x;
14     return os;
15   }
16 };
```

4. Przetestuj funkcję maximum o typie `X`

```
1 cout<<maximum(X(4), X(3));
```

5. Dopisz przeciążenie operatora >, aby kod się skompilował.

6. Spraw, aby klasa X mogła mieć zmienną dowolnego typu. W tym celu stwórz szablon klasy X;

```
1  template <typename T>
2  class X{
3  private:
4    T x;
5  public:
6      X(){}
7    X(T _x){
8      this->x = _x;
9    }
10   T getX() const{
11     return x;
12   }
13   friend ostream& operator << (ostream& os, X const & x){
14     os << x.x;
15     return os;
16   }
17 };
18
```

```
19
20
21 template <typename T>
22 bool operator > (X<T> const & a, X<T> const & b){
23   return a.getX() > b.getX();
24 }
```

7. Wywołaj funkcję `maximum` z argumentami będącymi obiektami klasy X.

```
1 cout<<maximum(X<int>(4), X<int>(3))<<endl;
```

8. Szablony można specjalizować dla wybranych typów. Spraw, aby klasa X korzystająca z typu string dla operatora > brała pod uwagę długość łańcucha znaków, zamiast (domyślnie) kolejności leksykograficznej. W tym celu dopisz kolejne przeciążenie operatora >.

```
1 template <>
2 bool operator > (X<string> const & a, X<string> const & b){
3   return a.getX().size() > b.getX().size();
4 }
5
6
7 cout<<maximum(X<string>("aaa"), X<string>("bb"))<<endl;
```

9. Szablony mogą mieć domyślne wartości, oraz mogą się odwoływać do poprzednich parametrów szablonu.

```
1 template <typename T, typename C = X<T> >
2 class Y{
3   T zmienna;
4   C klasaX;
5 };
6
7 Y<int> y;
8 Y<int, double> y2;
```

10. Aby odwołać się do nazwy typu uzależnionego od parametru szablonu, należy kwalifikować tę nazwę słowem kluczowym `typename`.

```
1 template<typename T>
2 class A {
3   typename T::id i;
4   public:
5   void f() { i.g(); }
6 };
7
8 class B {
9 public:
```

```
  class id {
  public:
    void g() {}
  };
};

int main() {

...

  A<B> ab;
  ab.f();
}
```

11. Napisz funkcję maximum korzystając z funkcji lambda.

```
auto maxLambda = [](auto a, auto b) { return a > b ? a : b;
    };
bool b = maxLambda(3, 3.14);
auto c = maxLambda(3, 3.14);
```

Zadanie 2. Typy generyczne w Java

1. Zapoznaj się z podstawą tworzenia typów generycznych w Java.

```
package main;

import java.util.*;
import java.lang.*;
import java.io.*;

public class GenericsType<T> {

  private T t;

  public T get(){
    return this.t;
  }

  public void set(T t1){
    this.t=t1;
  }

  public static void main(String args[]){
    GenericsType<String> type = new GenericsType<>();
    type.set("PUT"); //valid
    System.out.println(type.get());
    GenericsType type1 = new GenericsType(); //raw type
    type1.set("UAM"); //valid
```

```
25      System.out.println(type1.get());
26      type1.set(10); //valid and autoboxing support
27      System.out.println(type1.get());
28    }
29  }
```

2. Stwórz publiczny interfejs (w nowym pliku), który będzie generyczny i będzie umożliwiał sprawdzanie czy dana zmienna jest parzysta.

```
1  package main;
2  public interface Checker<T> {
3      boolean check(T object);
4  }
```

do poprzedniego pliku w funkcji main dopisz:

```
1  Checker<Integer> isOddAnonymous = new Checker<Integer>() {
2    @Override
3    public boolean check(Integer object) {
4      return object % 2 != 0;
5    }
6  };
7
8  System.out.println(isOddAnonymous.check(123));
9  System.out.println(isOddAnonymous.check(124));
```

3. Wykonaj to samo zadanie korzystając z lambdy. W tym celu w funkcji main dopisz:

```
1  Checker<Integer> isOddLambda = object -> object % 2 != 0;
2  System.out.println(isOddLambda.check(123));
3  System.out.println(isOddLambda.check(124));
```