Язык C++

Шаблоны классов и функций

max

```
int max(int a, int b) {
  return a > b ? a : b;
}

const CRational& max(const CRational& a, const CRational& b) {
  return a > b ? a : b;
}
```

Шаблоны

- Шаблоны определяют семейство функций, классов, типов и переменных
- Шаблон параметризуется одним или несколькими параметрами, которые могут являться:
  - Тип
  - Константные выражения (интегральных типов, энумов)
  - Указатели (на объект, функцию, член класса)
  - std::nullptr_t
  - Вещественные числа, литеральные типы (C++20)
- Реализуют статический полиморфизм (полиморфизм времени компиляции)
- Не требуют дополнительных расходов по сравнению с “прямыми” реализациями

Шаблоны функций

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

template<class T>
void printMe(const T& value) {
  std::cout << value;
}
```

Template argument deduction

- Перед инстанциацией все параметра шаблона должны быть известны
- Компилятор, может вывести эти параметры из аргументов, если это можно сделать однозначно

```
int main() {
  int a = 10;
  int b = 20;

  printMe<int>(max<int>(a, b));
  printMe(max(a, b));
  printMe<CRational>(max<double>(a, b));
}
```

Шаблоны функций

```
int main() {
    std::cout << max(100500, 1005001) << std::endl;
    /*
    const int& max(const int& a, const int& b) {
        return a > b ? a : b;
     }
    */

    std::cout << max(100.500, 100.501) << std::endl; // const double& max(const double&, const
double&) {.....}
    std::cout << max("def", "abc") << std::endl;     // const char[4]& max(const char[4]&,
const char[4]&) {..}

    //std::cout << max(10, 20.2) << std::endl;   // error: no matching function for call to
'max'

    std::cout << max<double>(10, 20.2) << std::endl;
    std::cout << max<std::string>("def", "abc") << std::endl;
}
```

Шаблоны функций

```
template<class InputIt, class UnaryFunc>
void for_each(InputIt first, InputIt last, UnaryFunc f) {
  for(;first != last; ++first) {
      f(*first);
  }
};

int main() {
 int values[] {1,2,3,4,5};
 for_each(values, values+5, printMe<int>);
}
```

Инстанциация

- Без инстанциации не происходит генерации конкретного шаблона
- При использовании шаблонной функции или класса, требуется полно определение, поэтому для использования в других единицах трансляции шаблоны требуется определять в заголовочных файлах
- Шаблон генерирует “настоящий” класс или функцию
- Явная и неявная инстанциация

Шаблон класса

```
template<typename T1, typename T2>
class CPair {
private:
  T1 first_;
  T2 second_;
};
```

Шаблон класса

```
template<typename T1, typename T2>
class CPair {
public:
  CPair() = default;

  CPair(const T1& first, const T2& second)
      : first_{first}
      , second_{second}
  {}
```

Шаблон класса

```
template<typename T1, typename T2>
class CPair {
public:
    CPair& operator=(const CPair& other) {
        if(this == &other)
            return *this;
        first_ = other.first_;
        second_ = other.second_;
        return *this;
    }
};
```

Шаблон класса

```
template<typename T1, typename T2>
std::ostream& operator<<(std::ostream& stream, const CPair<T1,T2>& pair) {
  stream << "CPair(" << pair.First() << "," << pair.Second() << ")";
  return stream;
};

int main() {
  CPair<int, float> p1(10, 11.2);
  CPair<std::string, int> p2 ("qwerty", 23);
  std::cout << p1 << std::endl;
  std::cout << p2 << std::endl;
}
```

Шаблон класса

```
int main() {
  CPair<int, float> p1(10, 11.2);
  CPair<std::string, int> p2 ("qwerty", 23);
  CPair<int, float> p3;

  p3 = p1;
}
```

Шаблон класса

```
int main() {
  CPair<int, float> p1(10, 11.2);
  CPair<std::string, int> p2 ("qwerty", 23);
  CPair<int, float> p3;
  CPair<float, float> p4;

  p3 = p1;
  p4 = p1 // Error

}
```

Шаблон класса

```
template<typename T1, typename T2>
class CPair {
public:
    template<class U1, class U2>
    CPair&operator=(const CPair<U1, U2>& other) {
        first_ = static_cast<T1>(other.first());
        second_ = static_cast<T2>(other.second());

        return *this;
    };
};
```

Class template argument deduction (CTAD)

```
int main() {
  CPair p1(10, 11.2);
  CPair p2(std::string{"qwerty"}, 23);
  CPair p3(20, 21.1);
  CPair p4(1.2, 2.3);

  p3 = p1;
  p4 = p1;
}
```

Явно не указываем параметры шаблона

Non-type template parameter

- Константные выражения (интегральных типов, энумов)
- Указатели (на объект, функцию, член класса)
- std::nullptr_t
- Вещественные числа, литеральные типы (C++20)

Non-type template parameter

```
template<typename T, size_t SIZE>
class CArray {
public:
   T& operator[](size_t index) { return arr[index]; }
   const T& operator[](size_t index) const { return arr[index]; }

   bool empty() const { return SIZE == 0; }
   size_t size() const { return SIZE;}
private:
   T arr[SIZE];
};
```

Аргументы по умолчанию

```
template <class T>
struct greater{
  bool operator()(const T& a, const T& b) const {
      return a > b;
  }
};

template <class T>
struct less {
  bool operator()(const T& a, const T& b) const {
      return a < b;
  }
};
```

NB! Функторы

Аргументы по умолчанию

```
template <class T, class cmp_t=less<T>>
void buble_sort(T* begin, T* end, cmp_t cmp = cmp_t()) {
  size_t count = end - begin;

  for(size_t i = 0; i < count; ++i) {
      for(size_t j = 0; j < count - i - 1; ++j) {
          if(!cmp(begin[j],begin[j+1]))
              std::swap(begin[j], begin[j+1]);
      }
  }
}
```

Non-type template parameter

```
template<unsigned N>
struct factorial {
  enum {value = N * factorial<N-1>::value };
};

template <>
struct factorial<0> {
  enum {value = 1};
};
```

Инстанциацию можно посмотреть на https://cppinsights.io/

Шаблоны

```
template <unsigned N>
struct fibonacci {
  enum {value = fibonacci<N-1>::value + fibonacci<N-2>::value};
};

template <>
struct fibonacci<0> {
  enum {value = 0};
};
template <>
struct fibonacci<1> {
  enum {value = 1};
};
```