Новое в C++, не связанное с классами

Тема 1

Ввод-вывод в C++

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

```
#include <iostream>


int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Ввод-вывод в C++

```
int a, b, c;

float p;

//получение значений этих переменных с клавиатуры

std::cin >> a >> b >> c >> p;

//вывод на экран этих переменных в столбик

std::cout << a << std::endl
          << b << std::endl
          << c << std::endl
          << p << std::endl;
```

Ввод-вывод строк в виде массива символов

```
char str[100];
std::cin.getline(str, 100);
std::cout << str <<std::endl;
```

Форматирование вывода

```
std::cout.setf(ios::fixed);//формат вывода
//количество знаков после запятой
std::cout.precision(2);
for(int i=0;i<size;i++)
{
   std::cout.width(6); //ширина столбца
   std::cout << arr[i];
}
std::cout << std::endl;
```

Тип string

```
#include <string>
int main() {
    std::string s0 = "abcde"; //создаем переменную типа string
    std::string s1 = "fg"; //создаем еще одну переменную типа string
    std::string s = s0 + s1; // Конкатенация двух строк.
    std::cout << s.c_str() << std::endl; //Вывод строки на экран
    std::cout << s << std::endl; //Вывод строки на экран Visual studio 2017
    // Присваиваем и сравниваем 2 строки
    s1 = s0; //возможно прямое присваивание двух строк
    if(s1 == s0)
    //и сравнение двух строк как обычных переменных
        std::cout << "Strings are equal"<< std::endl;
    else
        std::cout << "Strings are not equal"<< std::endl;
    // Чтение введенной с клавиатуры строки
    std::getline(std::cin, s1);
    std::cout << s1 << std::endl;
    // Получение длины строки
    std::cout << s1.length() << std::endl;
    //Получение индекса первого вхождения подстроки в строку
    //второй параметр - позиция начала поиска - по умолчанию равен 0
    int a = s1.find("123", 0);
    std::cout << a << std::endl;
}
```

Операторы new и delete

```
char * MyArr;
std::cout << "Enter a number" << std::endl;
std::cin >> i;
//выделение памяти под массив //символьного типа
MyArr = new char[i];
…
//освобождение памяти из-под массив
delete [] MyArr;
```

Ссылка - это

второе имя переменной, псевдоним

```
int count = 1;       //объявление целой переменной count
int &cRef = count;   //создание cRef как псевдонима для count
++cRef;              //count = 2
int *p= &cRef;       //p указывает на count
```

Ни один оператор не выполняет действия над ссылкой

Инициализация ссылки

Ссылка должна быть инициализирована в момент объявления

```
int count = 1;
int &cRef = count;
int &cRef1;                 //ошибка
int &cRef2 = 2;             //ошибка
short &cRef3 = count;       //ошибка
```

Константные ссылки

```
const int &cRef = 1;


void fun (const int & ref);
int main()
{
    short m = 15;
    int n = 5;
    fun(10);
    fun(n);
    fun(m);
}
```

Передача параметров по ссылке

```
void swap(int &a, int &b)
{
   int temp = a;
   a = b;
   b = temp;
}
```

```
int main()
{
   int a = 5, b = 7;
   std::cout << "a= " << a << " " << "b= " << b
             << std::endl;
   swap(a, b);
    std::cout << "a= " << a << " " << "b= " << b
             << std::endl;
   return 0;
}
```

Возвращаемое значение

```
int & fun1(int a)
{
    int n = 0;
    n += a;
    return n; //плохо
}
```

```
int & fun1(int a)
{
    static int n = 0;
    n += a;
    return n; //хорошо
}
```

Перегрузка функций

```
int square (int x)
{
    return x*x;
}
```

```
double square (double y)
{
    return y*y;
}
```

```
int main()
{
    std::cout << "Integer square is " << square(7) << std::endl;
    std::cout << "Double square is " << square(7.1) << std::endl;
    return 0;
}
```

Параметры по умолчанию

```
int test (double a, int b);              //1
int test (int a, double b);              //2
int test (int a, int b, int c=3);        //3
int test (int a, int b = 4);             //4
int test (int a = 7);                    //5
```

```
int main()
{
   test(3.5, 6);                         //1
   test(6,3.5);                          //2
   test();                               //5
   test(3,4,5);                          //3
   test(2,3);                            //???
   test(5);                              //???
   return 0;
}
```

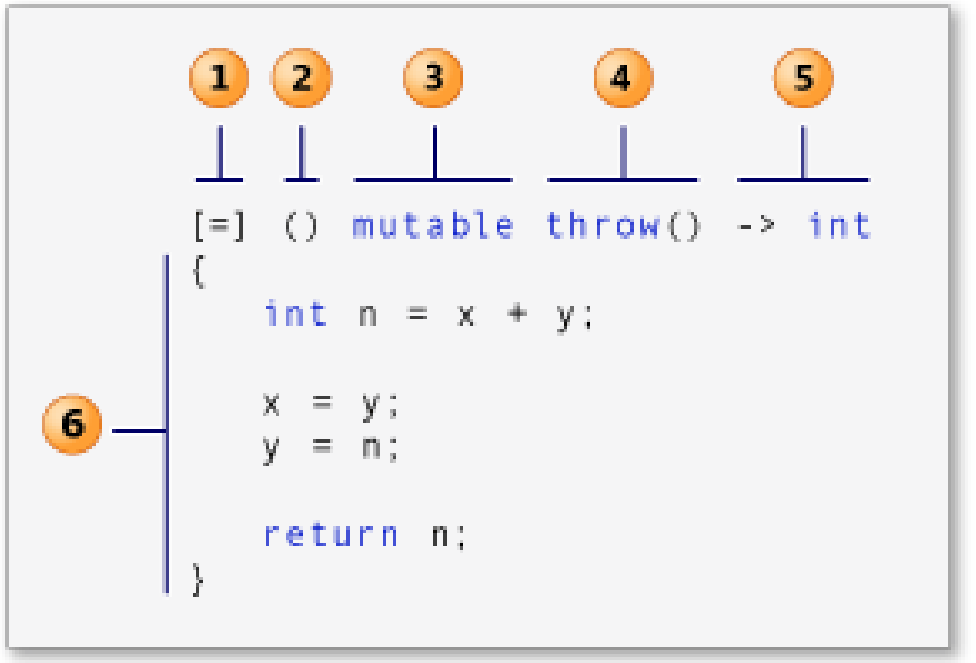
Лямбда - выражение

- Анонимный функтор
- Определяется прямо в месте его вызова или передачи в функцию в качестве аргумента
- Используются для инкапсуляции нескольких строк кода, передаваемых алгоритмам или асинхронным методам

```
#include <algorithm>
#include <cmath>
void abssort(float* x, unsigned n)
{
    std::sort(x, x + n,
    // Lambda expression begins
    [](float a, float b)     {
         return (std::abs(a) < std::abs(b)); }
    // end of lambda expression
    );
}
```

Структура лямбда-выражения

1. Предложение фиксации (в спецификации C++ это lambda-introducer.)
2. Список параметров (необязательно). (Также именуется как lambda declarator.)
3. Отключаемая спецификация (необязательно).
4. Спецификация исключений (необязательно).
5. Завершающий возвращаемый тип (необязательно).
6. Тело лямбда-выражения



Фиксация

- Лямбда-выражения могут использовать внешние переменные
- В предложении фиксации указывается способ доступа к внешним переменным:
  - `=` - по значению (используется по умолчанию)
  - `&` - по ссылке

Если тело лямбда-выражения осуществляет доступ к внешней переменной `total` по ссылке, а к внешней переменной `factor` по значению, следующие предложения фиксации эквивалентны:

```
[&total, factor]
[factor, &total]
[&, factor]
[factor, &]
[=, &total]
[&total, =]
```

Фиксация

- Если предложение фиксации включает **capture-default &**, ни один **identifier** в параметре capture этого предложения фиксации не может иметь форму **& identifier**.
- Если предложение фиксации включает **capture-default =**, ни один параметр **capture** этого предложения фиксации не может иметь форму **= identifier**.
- Идентификатор или **this** не могут использоваться более одного раза в предложении захвата
- Фиксировать статические переменные нельзя

```
struct S { void f(int i); };
void S::f(int i)
{
    [&, i]    {};    // OK
    [&, &i]   {};    // ERROR: i preceded by & when & is the default
    [=, this]{};     // ERROR: this when = is the default
    [i, i]    {};    // ERROR: i repeated
}
```

Список параметров

- Лямбда-выражения могут принимать входные параметры
- Похожи на список параметров функции

```
int y = [] (int first, int second) {
    return first + second;
};
```

- Не являются обязательными
- Если используются универсальные параметры, в качестве спецификатора типа можно задействовать ключевое слово **auto**. Это отдает компилятору команду создать оператор вызова функции в качестве шаблона. Каждый экземпляр ключевого слова **auto** в списке параметров эквивалентен параметру отдельного типа

```
auto y = [] (auto first, auto second) {
    return first + second;
};
```

- Может принимать другое лямбда-выражение

Отключаемая спецификация

Позволяет телу лямбда-выражения изменять переменные, захваченные по значению

```
int m = 0;
int n = 0;
[&, n] (int a) mutable { m = ++n + a; }(4);
std::cout << m << std::endl << n << std::endl;
```

Результат:

```
5

0
```

Использование статических переменных внутри тела лямбда-функций

```
void fillVector(vector<int>& v)
{
    static int nextValue = 1;
    generate(v.begin(), v.end(),
             [] { return nextValue++; });
}
```

Спецификация исключений

- Можно указать **throw-list** – список исключений, которые лямбда может сгенерировать
- В финальном варианте стандарта **throw-спецификации** объявлены устаревшими. Вместо этого оставили ключевое слово **noexcept**, которое говорит, что функция не должна генерировать исключение вообще.

```
//лямбда не может генерировать исключений
[] (int _n) throw() { … }

//лямбда генерирует std::bad_alloc
[=] (const std::string & _str) mutable throw(std::bad_alloc)   { … }
```

Тип возвращаемого значения

- По умолчанию лямбда-выражение имеет тип **`void`**
- Если тело лямбда-выражения имеет один оператор **`return`**, тип возвращаемого значения выводится автоматически
- Если тело лямбда-выражения содержит более одного оператора **`return`**, тип возвращаемого значения необходимо указать явно

```
vector<int> srcVec;
for (int val = 0; val < 10; val++){
    srcVec.push_back(val);
}
```

```
int result = count_if (srcVec.begin(),
                       srcVec.end(),
                       [] (int _n) {
                          return (_n % 2) == 0;
                       });
```

Тип возвращаемого значения

```
vector<int> srcVec;
for (int val = 0; val < 10; val++)
{
    srcVec.push_back(val);
}
```

```
vector<double> destVec;
transform(srcVec.begin(), srcVec.end(),
          back_inserter(destVec), [] (int _n) -> double
    {
        if (_n < 5)
            return _n + 1.0;
        else if (_n % 2 == 0)
            return _n / 2.0;
        else
            return _n * _n;
    });
```

Тело лямбда-выражения

Тело лямбда-выражения может осуществлять доступ к следующим типам переменных:

- Фиксированные переменные из внешней области видимости
- Параметры
- Локально объявленные переменные
- Данные-члены класса (при объявлении внутри класса и фиксации `this`)
- Любая переменная, которая имеет статическую длительность хранения (например, глобальная переменная)

Лямбда-выражения. Пример

```
// The number of elements in the vector.
const int elementCount = 9;

// Create a vector object with each element set to 1.
vector<int> v(elementCount, 1);

// These variables hold the previous two elements of the vector.
int x = 1; int y = 1;

// Sets each element in the vector to the sum of the previous two elements.
generate_n(v.begin() + 2,
         elementCount - 2,
         [=]() mutable throw() -> int { // lambda is the 3rd parameter
          // Generate current value.
          int n = x + y;
          // Update previous two values.
          x = y; y = n;
          return n;  });
```

1 1 2 3 5 8 13 21 34

Лямбда-выражения. Примеры

```
class MyMegaInitializer
{
public:
     MyMegaInitializer(int _base, int _power)
          : m_val(_base)
          , m_power(_power) {}
     void initializeVector(vector<int> & _vec)
     {
          for_each(_vec.begin(), _vec.end(), [this] (int & _val) mutable
          {
               _val = m_val;
               m_val *= m_power;
          });
     }
private:
     int m_val, m_power;
};
```

Вложение лямбда-выражений

```
int timestwoplusthree =
        [](int x) {
            return [](int y) {
                return y * 2;
            }(x) + 3;
        } (5);

std::cout << timestwoplusthree << std::endl;  //13
```

Лямбда-функции высшего порядка

```
auto addtwointegers = [](int x) -> function<int(int)> {
        return [=](int y) { return x + y; };
    };
```

```
auto higherorder = [](const function<int(int)>& f, int z) {
        return f(z) * 2;
    };
```

```
auto answer = higherorder(addtwointegers(7), 8);
```

```
std::cout << answer << std::endl;  //30
```

Конец