Семинар #6: Дополнительные возможности C++.

Часть 1: Ключевое слово `auto`

Ключевое слово `auto` используется для автоматического вывода типа.

```
#include <string>

int main() {
    auto a = 123;  // a будет иметь тип int
    auto b = 4.1;  // b будет иметь тип double

    auto s1 = "Hello";               // s1 будет иметь тип const char*
    auto s2 = std::string("Hello");  // s2 будет иметь тип std::string
}
```

Задачи:

- В примере ниже создан вектор строк и напечатано его содержимое. Тип итератора имеет очень длинное название (и название будет ещё больше если контейтер будет хранить не просто строки, а что-нибудь посложнее). Используйте `auto`, чтобы упростить код.

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

  int main() {
      std::vector<std::string> v {"Cat", "Dog", "Elephant"};
      for (std::vector<std::string>::iterator it = v.begin(); it != v.end(); ++it) {
          std::cout << *it << std::endl;
      }
  }
  ```

- Протестируйте, можно ли использовать `auto` вместо возвращаемого типа функции. Напишите функцию, которая принимает на вход вектор строк и возвращает строку, которая является результатом конкатенации всех строк. Вместо возвращаемого типа используйте `auto`.

- Протестируйте, можно ли создать функцию, которая будет принимать целое число и, в зависимости от этого числа, возвращать значения разных типов. (Если вместо возвращаемого типа используется `auto`).

- Протестируйте, можно ли создать функцию, у которой на месте типов аргументов будет стоять ключевое слово `auto`.

- Функция вычисления факториала, написанная ниже с использованием `auto` не работает.

  ```
  auto factorial(int n) {
      if (n > 0)
          return n * factorial(n - 1);
      return 1;
  }
  ```

  Почему? Исправьте эту функцию, не убирая `auto`.

Часть 2: Литералы `std::string`

Часть 3: Вывод шаблонных аргументов класса

Начиная со стандарта C++17 появилась возможность автоматического вывода шаблонных аргументов классов. Например, в примере ниже больше не нужно указывать шаблонный тип вектора. Компилятор догадается о нём сам.

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

int main() {
    std::vector v {4, 8, 15, 16};
    for (auto it = v.begin(); it != v.end(); ++it) {
        std::cout << *it << std::endl;
    }
}
```

Задачи:

- Создайте вектор, содержащий несколько элементов типа `double` и напечатайте его.
- Предположим, что мы создали вектор строк и попытались добавить к концу каждой строки букву `s` вот так:

  ```
  std::vector v {"Mouse", "Cat", "Dog"};
  for (auto it = v.begin(); it != v.end(); ++it) {
      (*it) += "s";
  }
  ```

  Данный код выдаст ошибку. Почему и как её исправить?
- Создайте контейнер `std::set`, содержащий строки `Mouse`, `Dog`, `Cat`. Напечатайте всё содержимое этого контейнера, оно должно напечататься в алфавитном порядке.
- Создайте вектор, содержащий пары(`std::pair`) целых чисел и напечатайте его. Используйте вывод шаблонных аргументов класса.

Часть 4: Range-based циклы

Циклы, основанные на диапазоне, предоставляют более простой способ обхода контейнера:

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

int main() {
    std::vector v {6, 1, 7, 4};
    for (int num : v) {
        std::cout << num << std::endl;
    }
}
```

Для изменения элементов контейнера при обходе нужно использовать ссылки:

```
for (int& num : v) {
    num += 1;
}
```

Задачи:

- Проверьте, можно ли использовать ключевое слово `auto` внутри таких циклов.
- Пусть у нас есть вектор строк:

  ```
  vector<string> v {"Cat", "Axolotl", "Bear", "Elephant"};
  ```

  - Напишите range-based цикл, который будет печатать все элементы вектора
  - Напишите range-based цикл, который будет добавлять в конец каждой строки символ `s`.
  - Напишите range-based цикл, который будет обращать каждую строку. Используйте стандартную функцию `reverse`.
- Проверьте, можно ли использовать range-based циклы если контейнер является:
  - `std::list`
  - `std::set`
  - `std::map`
  - `std::pair`
  - Обычным массивом
  - `std::string`
  - Строкой в стиле `C`
- Будет ли работать такой код:

  ```
  for (auto x : {4, 8, 15, 16}) {
      cout << x << " ";
  }
  ```

  Объяснить почему это работает/не работает.
- Для печати массива целых чисел была написана следующая функция:

  ```
  void print(int array[]) {
      for (int num : array) {
          std::cout << num << std::endl;
      }
  }
  ```

  Оказывается, что она не работает. В чём заключается ошибка?

Часть 5: Контейнер `std::initializer_list`

Часть 6: Structure binding (структурное связывание)

В стандарте C++17 был добавлен новый вид объявления и инициализации нескольких переменных. В коде ниже мы объявляем переменные a и b одной строкой с помощью структурного связывания.

```
#include <iostream>
#include <utility>

int main() {
    std::pair p {5, 1};
    auto [a, b] = p;

    std::cout << a << " " << b << std::endl;
}
```

Структурное связывание работает только в том случае, если размер контейнера справа известен на стадии компиляции. Например, пары, кортежи(std::tuple), статические массивы, std::array, простые структуры.

Задачи:

- Пусть у нас есть пара:

  ```
  std::pair p {std::string{"Moscow"}, 1147};
  ```

  - Создайте две переменные name и age и присвойте их соответствующим элементам пары.
  - Создайте две ссылки name и age и инициализируйте их соответствующими элементами пары. Убедитесь, что при изменении переменной name меняется и пара p.

- Метод insert контейнера std::set пытается вставить элемент в множество. Если же такой элемент в множестве уже существует, то он ничего с множеством не делает. Но этот метод возвращает пару из итератора на соответствующий элемент и переменной типа bool, которая устанавливается в true если новый элемент был добавлен и в false, если такой элемент уже существовал. Вот пример программы, которая пытается вставить элемент в множество и печатает соответствующее сообщение. В любом случае программа печатает все элементы, меньшие вставляемого.

  ```
  #include <iostream>
  #include <utility>
  #include <set>
  using std::cout;
  using std::endl;
  int main() {
      std::set<int> s {1, 2, 4, 5, 9};

      std::pair<std::set<int>::iterator, bool> result = s.insert(5);
      if (result.second == true) {
          cout << "Element added successfully" << endl;
      }
      else {
          cout << "Element already existed" << endl;
      }

      for (std::set<int>::iterator it = s.begin(); it != result.first; ++it) {
          cout << *it << " ";
      }
  }
  ```

  Упростите эту программу, используя ключевое слово auto и структурное связывание.

Структурное связывание можно использовать и в цикле.

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

int main() {
    std::vector<std::pair<std::string, int>> v {{"Moscow", 1147}, {"Berlin", 1237},
                                                {"Rome", -753},  {"Bogota ", 1538}};
    for (auto [city, year] : v) {
        std::cout << city << " " << year << std::endl;
    }
}
```

Задачи:

- В файле `books.cpp` лежит заготовка кода. В ней содержится инициализированный массив из структур. Сделайте следующее:
  - Напечатайте массив `books`, используя range-based цикл. Нужно напечатать все поля через запятую.
  - Напечатайте массив `books`, используя range-based цикл со структурным связыванием.
  - Увеличьте поле `price` всех книг на одну величину, используя range-based цикл.
  - Увеличьте поле `price` всех книг на одну величину, используя range-based цикл со структурным связыванием.
- Ниже есть пример программы – решение задачи с предыдущего семинара. Она считывает слова и печатает количества всех введённых до этого слов.

```
#include <iostream>
#include <map>
#include <utility>
#include <string>
using std::cout;
using std::endl;

int main() {
    std::map<std::string, int> word_count;

    while (true) {
        std::string word;
        std::cin >> word;
        std::pair<std::string, int> wc {word, 1};
        std::pair<std::map<std::string, int>::iterator, bool> p = word_count.insert(wc);
        if (p.second == false) {
            word_count[word] += 1;
        }

        cout << "Dictionary:" << endl;
        for (std::map<std::string, int>::iterator it = word_count.begin();
                                it != word_count.end(); ++it) {
            std::cout << (*it).first << ": " << (*it).second << endl;
        }
        cout << endl;
    }
}
```

Упростите код этой программы, используя `auto` и структурное связывание.