Семинар #4: Шаблоны. Класс `std::vector`. Домашнее задание.

Задача 1. Сумма чётных

Напишите функцию `int sumEven(const std::vector<int>& v)`, которая будет принимать вектор по константной ссылке и возвращать сумму всех чётных чисел этого вектора.

аргумент	возвращаемое значение
`{4, 8, 15, 16, 23, 42}`	`70`

Задача 2. Последние цифры

Напишите функциии которые должны будут принимать на вход вектор и заменять все его элементы на последние цифры этих элементов. Все функции должны делать одно и то же, но должны будут возвращать результат разными способами.

- Функция `std::vector<int> lastDigits1(const std::vector<int>& v)` должна возвращать вектор.
- Функция `void lastDigits2(std::vector<int>& v)` должна принимать вектор по ссылке и изменять его.
- Функция `void lastDigits3(std::vector<int>* pv)` должна принимать указатель, хранящий адрес некоторого вектора. Функция должна изменять вектор, на который указывает указатель.
- Функция `void lastDigits4(std::span<int> sp)` должна принимать объект типа `std::span` и менять вектор, на который указывает этот span.

Задача 3. Простые делители

Напишите функцию `std::vector<std::pair<int, int>> factorization(int n)`, которая будет находить все простые делители числа `n` и их количества.

аргумент	возвращаемое значение
60	{{2, 2}, {3, 1}, {5, 1}}
626215995	{{3, 3}, {5, 1}, {17, 1}, {29, 1}, {97, 2}}
107	{{107, 1}}
1	{{1, 1}}

Задача 4. Времена из строки

- Напишите простой класс для работы со временем:

```
class Time {
private:
    int mHours, mMinutes, mSeconds;
public:
    Time(int hours, int minutes, int seconds);
    Time(std::string_view s); // строка в формате "hh:mm:ss"
    Time operator+(Time b) const;
    int hours() const; int minutes() const; int seconds() const;
    friend std::operator<<(std::ostream& out, Time t);
};
```

- Напишите функцию `std::vector<Time> getTimesFromString(std::string_view s)`, которая будет принимать строку в формате `"hh:mm:ss ... hh:mm:ss"`, где за место букв должны стоять некоторые числа. Например, строка может иметь вид `"12:20:05 05:45:30 22:10:45"`. Функция должна возвращать вектор времен, соответствующий временам в строке.

- Напишите функцию `Time sumTimes(const std::vector<Time>& v)`, которая будет суммировать все времена и возвращать эту сумму.

Задача 5. Максимум

Напишите шаблонную функцию, которая будет принимать на вход вектор и возвращать максимальный элемент вектора. Протестируйте данную функцию на векторах, содержащих объекты следующих типов: `int`, `float`, `std::string` и `std::pair<int, int>`.

Задача 6. Разбиение на пары

Напишите шаблонную функцию, которая будет принимать на вход контейнер (`vector`, `array`, `string`, `span` или `string_view`) и возвращать вектор пар нечётных и чётных элементов. Если в контейнере было нечётное количество элементов, то второй элемент последней пары должен быть инициализирован с помощью value-инициализации.

аргумент	возвращаемое значение
`std::vector{10, 20, 30, 40, 50}`	`{{10, 20}, {30, 40}, {50, 0}}`
`std::array<std::string, 4>{"cat", "dog", "mouse", "lion"}`	`{{"cat", "dog"}, {"mouse", "lion"}}`
`"Hello"`	`{{'H', 'e'}, {'l', 'l'}, {'o', '\0'}}`

Протестировать функцию можно в файле `code/test_pairing.cpp`.

Задача 7. Менеджер

Напишите шаблонный класс `Manager`, который будет разделять процессы выделения/освобождения памяти и создания/уничтожения объекта. У этого класса должны быть следующие методы:

- Конструктор по умолчанию.
- `allocate()` - будет выделять необходимое количество памяти под объект типа `T` в куче (используйте `std::malloc`).
- `construct(const T& t)` - будет создавать объект типа `T`, используя конструктор копирования, в выделенной памяти. Используйте оператор `placement new`.
- `destruct()` - будет уничтожать объект в выделенной памяти.
- `dealocate()` - будет освобождать выделенную памяти.
- `get` - будет возвращать ссылку на объект.

Пример использования данного класса:

```
Manager<std::string> a;
a.allocate();

a.construct("Cats and dogs");
a.get() += " and elephant";
cout << a.get() << endl;
a.destruct();

a.construct("Sapere Aude");
cout << a.get() << endl;
a.destruct();

a.deallocate();
```

Задача 8. Указатель или ссылка?

Напишите шаблонный класс `Ref<T>` который будет совмещать свойства указателя и ссылки. Как и указатель этот объект можно будет копировать и ложить в контейнеры. Но инициалироваться данный объект должен как ссылка и все операторы, применяемые к этому объекту, должны применяться к тому объекту, на который он ссылается. Правда, к сожалению, перегрузить оператор точка (пока?) нельзя, поэтому вместо этого будем использовать оператор `->`.

- Конструктор от объекта типа `T`.
- Конструктор копирования.
- Оператор присваивания. Присваивание должно производится к объекту, на который ссылается `Ref`.
- Оператор `+=`.
- Оператор `+`. Должен возвращать новый объект типа `T`.
- Перегруженный оператор `->`
- Функция `get`. Должна возвращать ссылку на объект, на который `Ref` ссылается.
- Дружественный оператор `operator«(std::ostream&, Ref<T>)`.

Код для тестирования:

```
void toUpper(Ref<std::string> r)
{
    for (size_t i = 0; i < r->size(); ++i)
        r.get()[i] = toupper(r.get()[i]);
}
int main()
{
    int a = 10;
    Ref<int> ra = a;
    ra += 10;
    cout << a << " " << ra << endl;

    std::string s = "Cat";
    Ref<std::string> rs = s;
    rs = "Mouse";
    rs += "Elephant";
    cout << rs << endl;
    cout << s  << endl;

    toUpper(s);
    cout << s << endl;

    std::vector<std::string> animals {"Cat", "Dog", "Elephant", "Worm"};
    std::vector<Ref<std::string>> refs {animals.begin(), animals.end()};

    for (int i = 0; i < refs.size(); ++i)
        refs[i] += "s";

    for (int i = 0; i < animals.size(); ++i)
        cout << animals[i] << " ";
    cout << endl;
}
```

Этот код должен напечатать:

```
20 20
MouseElephant
MouseElephant
MOUSEELEPHANT
Cats Dogs Elephants Worms
```