Семинар #7: Move-семантика и умные указатели. Домашнее задание.

Задача 1. lvalue или rvalue

Пусть есть следующий код:

```
#include <cstdlib>

int funca()
{
    return 10;
}

int& funcb(int& x)
{
    x = x + 1;
    return x;
}

int&& funcc(int& x)
{
    x = x + 1;
    return static_cast<int&&>(x);
}

int main()
{
    int a = 100;
    int& l = a;
    int&& r = 200;

    // Мы находимся тут
}
```

Какие из следующих выражений являются lvalue, а какие - rvalue:

1. `100`
2. `a`
3. `a + 1`
4. `l`
5. `r`
6. `l + r`
7. `std::abs(a)`
8. `funca()`
9. `funcb(a)`
10. `funcb(l)`
11. `funcb(r)`
12. `funcc(a)`
13. `funcc(l)`
14. `funcc(r)`
15. `static_cast<int>(a)`
16. `static_cast<int&>(a)`
17. `static_cast<int&&>(a)`
18. `static_cast<int&&>(r)`
19. `rand() > 5 ? a : 10`
20. `rand() > 5 ? a : r`
21. `+a`

Решение этой задачи – текстовый файл с ответами.

Задача 2. Стек строк

Напишите класс `StringStack`, объекты которого будут хранить некоторое количество строк типа `std::string`. Нужно написать следующие методы класса `StringStack`:

- `push`: В этот метод передаётся одна строка и эта строка добавляется в стек. Причём, если строка передаёся по lvalue, то она должна скопироваться, а если по rvalue, то переместиться.
- `print`: Этот метод должен печатать на экран все элементы стека в скобках, через запятую.
- `pop`: Этот метод должен удалять из стека последний элемент и возвращать его. В этом случае не должно происходить копирования строк, а только перемещения.

Напишите этот класс БЕЗ использования rvalue ссылок и универсальных ссылок. Используйте стандартные классы `std::string` и `std::vector` внутри класса это сильно упростит задачу.

```
...
StringStack ss;
std::string a {"Cat"};

ss.push(a);                     // должен скопировать строку a внутрь объекта ss
ss.push(std::string{"Mouse"}); // должен переместить временную строку внутрь объекта ss
ss.print();                     // печатает  (Cat, Mouse)
cout << a << endl;              // печатает  Cat

ss.push(std::move(a));          // должен переместить строку a внутрь объекта ss
ss.print();                     // печатает  (Cat, Mouse, Cat)
cout << a << endl;              // скорей всего напечатает  пустую строку
                                // ( зависит от длины строки и компилятора )

cout << a.pop() << endl;        // печатает  Cat
ss.print();                     // печатает  (Cat, Mouse)
cout << a.pop() << endl;        // печатает  Mouse
cout << a.pop() << endl;        // печатает  Cat
ss.print();                     // печатает  ()
...
```

Задача 3. Разделитель по категориям

Напишите класс `CategorySeparator`, который будет хранить в себе некоторое количество строк. Причём он должен запоминать, какие строки добавлялись как lvalue-выражения, а какие строки, как rvalue-выражения. Нужно написать следующие методы класса `CategorySeparator`:

- `push`: В этот метод передаётся одна строка (lvalue или rvalue) и эта строка добавляется в разделитель.
- `printLvalues`: Этот метод должен печатать на экран строки, которые были добавлены в разделитель как lvalue-выражения.
- `printRvalues`: Этот метод должен печатать на экран строки, которые были добавлены в разделитель как rvalue-выражения.

Используйте стандартные классы `std::string` и `std::vector` внутри класса это сильно упростит задачу.

```
...
CategorySeparator cs;
std::string a {"Cat"};
std::string b {"Dog"};

cs.push(a);                      // должен скопировать строку a внутрь объекта cs
cs.push(std::string{"Mouse"});   // должен переместить временную строку внутрь объекта cs
cs.push(a + b);                  // должен переместить временну строку a + b внутрь объекта cs
cs.push(b);                      // должен скопировать строку b внутрь объекта cs
cs.push(std::move(b));           // должен переместить строку b внутрь объекта cs

std::cout << a << std::endl;     // печатает "Cat"
std::cout << b << std::endl;     // скорей всего напечатает  пустую строку
                                 // ( зависит от длины строки и компилятора )

cs.printLvalues();               // печатает  "(Cat, Dog)"
cs.printRvalues();               // печатает  "(Mouse, CatDog, Dog)"
...
```

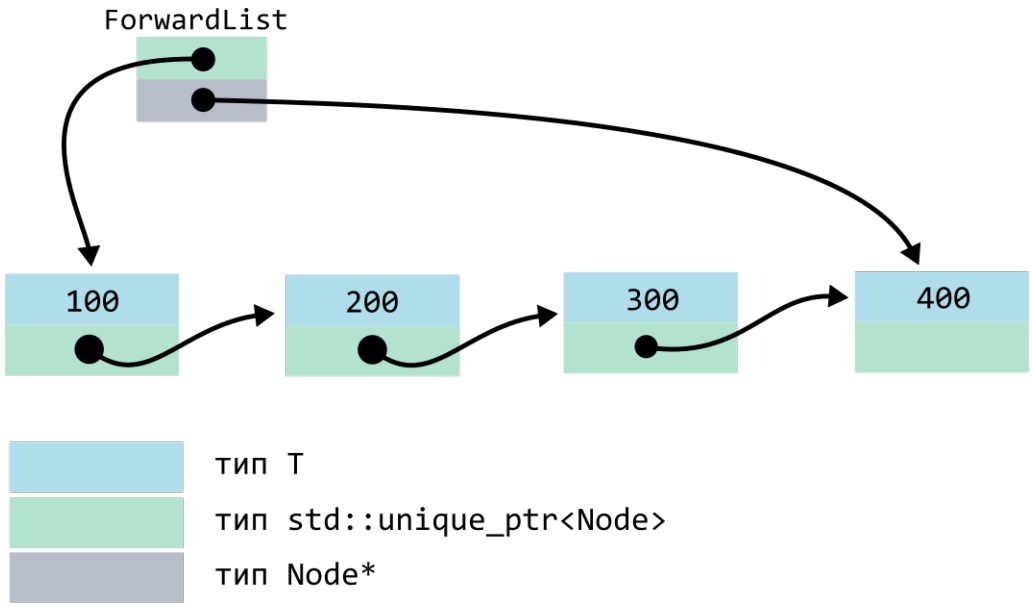
Задача 4. Односвязный список, используя `std::unique_ptr`

Создайте шаблонный класс `ForwardList<T>`, при этом, поле в узле списка, которое будет указывать на следующий узел должно иметь тип `std::unique_ptr`. Поля такого класса должны выглядеть так:

```
template <typename T>
class ForwardList
{
    struct Node
    {
        T value;
        std::unique_ptr<Node> next;
    };

    std::unique_ptr<Node> mpHead;
    Node* mpTail;
        ...
};
```

Схематическое строение объекта такого класса:



Вам нужно написать следующие методы данного класса:

- Конструктор по умолчанию.
- `void print()`
- `void push_front(T elem)`
- `void push_back(T elem)`
- `std::unique_ptr<T> pop_front()`
- `std::unique_ptr<T> pop_back()`
- `void clear()`
- `template <typename F> void foreach(F f)` – применяет функцию `f` к каждому элементу связного списка. Функция f должна принимать объект типа `T` по обычной ссылке.
- `void swap(ForwardList& fl)` - меняет местами содержимое данного связного списка и списка `fl`.
- `ForwardList copy()` - возвращает полную копию данного связного списка.