Семинар #8: Move-семантика и rvalue-ссылки.

Часть 1: `lvalue` и `rvalue`

Пусть есть функция `func`, принимающая объект некоторого типа по значению. В качестве типа объекта в этом примере возьмём `std::string`, но вообще это может быть любой тип.

```
void func(std::string s) {...}
```

Предположим, что мы передаём на вход этой функции некоторое выражение. Что лучше использовать при передаче в этом случае: копирование или перемещение? На самом деле это зависит от выражения, которое приходит на вход функции.

```
std::string s1 = get1();
std::string s2 = get2();

func(s1);      // В этом случае лучше использовать копирование, так как s1 нам ещё нужна
func(s1 + s2); // В этом случае лучше использовать перемещение, так как s1 + s2 нам не нужен
```

В общем случае нам бы хотелось, чтобы те выражения, у которых есть имя или известный адрес, передавались копированием. Так как они могут быть использованы после вызова функции. Такие выражения называются `lvalue`-выражениями. Остальные выражения, то есть те, которые мы хотим перемещать, называются `rvalue`-выражениями.

В коде ниже представлены примеры `lvalue` и `rvalue` выражений.

```
#include <string>
#include <math>

int main()
{
    int a = 123;
    int array[5] = {1, 2, 3, 4, 5};
    std::string s = "Cat";

    // Примеры lvalue выражений:
    a   array    s   array[i]

    // Примеры rvalue выражений:
    a+1    -a    2*array[i]    s+"!"    sqrt(5)
}
```

На самом деле, приведённое выше определение `lvalue` не полностью верно и не учитывает некоторые случаи. Более точное определение понятий `lvalue` и `rvalue` приведено в стандарте, оно слишком громоздко, чтобы описать его здесь.

При передаче в функцию по значению компилятор автоматически определяет является выражение `lvalue` или `rvalue` и, либо копирует его, либо перемещает. Но иногда бывают ситуации, когда нам хочется переметить `lvalue` выражение. В этом случае нужно просто использовать стандартную функцию `std::move`. Эта функция сама по себе ничего не передвигает, а просто превращает `lvalue` в `rvalue`.

```
func(s1);             // Произойдёт копирование
func(s1 + s2);        // Произойдёт перемещение временного объекта s1 + s2
func(std::move(s1));  // Произойдёт перемещение s1
// Тут s1 стал пустой строкой, так как мы его переместили
```

Часть 2: rvalue-ссылки

В предыдущей часте мы рассмотрели передачу в функцию по значению. Но чаще используется передача по ссылке. Для того, чтобы можно было различать категорию выражения при передаче по ссылке в язык были введены rvalue-ссылки. Такие ссылки очень похожи на обычные ссылки (которые теперь называются lvalue-ссылками). Основное отличие таких ссылок в том, что они инициализируются только rvalue-выражениями.

```
int a = 123;
// rvalue ссылку можно инициализировать так:
int&& r1 = 10;
int&& r2 = a + 1;
// Следующее не будет работать так как a это lvalue:
int&& r3 = a;
```

С помощью rvalue ссылок и перегрузки функций можно различить категорию выражения, приходящую на вход функции. В примере ниже вызовется соответствующий вариант перегрузки в зависимости от категории выражения.

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

void func(std::string& s)
{
    cout << "Pass by lvalue reference" << endl;
}
void func(std::string&& s)
{
    cout << "Pass by rvalue reference" << endl;
}

int main()
{
    std::string s1 = "Cat";
    std::string s2 = "Dog";

    func(s1);               // Передадим по lvalue ссылке
    func(s1 + s2);          // Передадим по rvalue ссылке
    func(s1.substr(0, 2));  // Передадим по rvalue ссылке
}
```

Часть 3: Конструктор перемещения и оператор присваивания перемещения

Для объекта можно написать конструктор копирования и оператор присваивания, которые должны производить глубокое копирование объекта. По аналогии с копированием, для объекта можно создать конструктор перемещения и оператор присваивания перемещением.