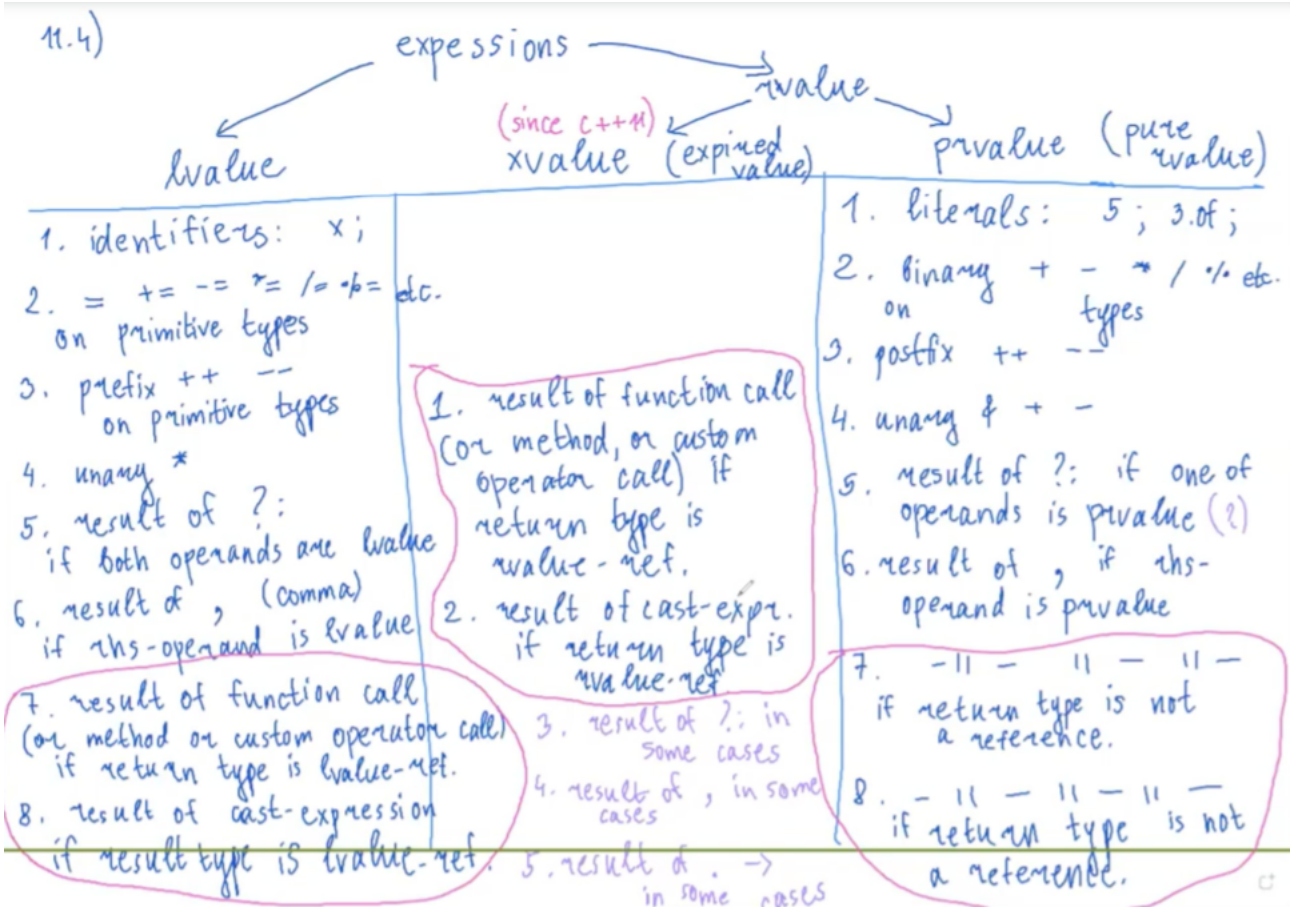
4.20 Формальное определение lvalue и rvalue. Объяснение идеи, стоящей за этим определением. Объяснение, как по выражению понять его вид value. Примеры, когда rvalue-выражение допускает присваивание и когда lvalue-выражение не допускает присваивания. Ссылочные квалификаторы (ref-qualifiers), синтаксис и пример использования.

Эти понятия применимы не к объектам и не к типам объектов, а к expressions.
Любой объект может быть как lvalue так и rvalue в зависимости от контекста, хотя опять-таки нельзя говорить про rvalue и lvalue относительно объектов. Интуитивно rvalue - выражение, которе представляет из себя создание нового объекта (то, что не может стоять справа от знака равенства). А lvalue - выражение, представляющее из себя обращение к уже существующему объекту (то, что может стоять слева от знака равенства). К данным определениям есть контр-примеры, это не всеобъемлющее определение.



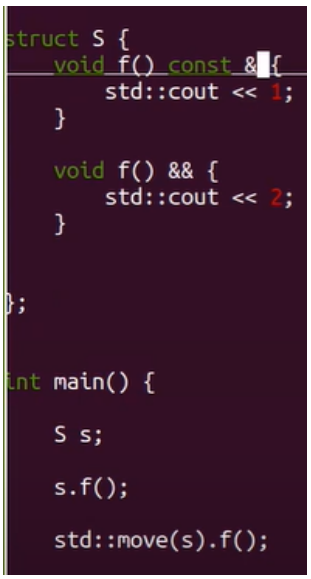
Примеры, когда rvalue-выражение допускает присваивание и когда lvalue-выражение не допускает присваивания.

```
int main() {
    int x = 0;
    int& rx = 2; // тут lvalue нельзя инициализировать rvalue
    int&& rrx = 1; // тут можно присвоить rvalue rvalue
}
```

Ссылочные квалификаторы (ref-qualifiers), синтаксис и пример использования

Аналогично константности, мы можем захотеть перегружать функции в зависимости от того, какое выражение является левым операндом функции - lvalue или rvalue. Если не ставим & то считается, что функция как для lvalue, так и для rvalue.

```
struct S{
    void f() &{...}
    void f() &&{...}
}
```



Сначала попадем в 1 версию, потом - во 2

Зачем эту нужно. Пусть у нас есть какой-нибудь метод, который возвращает какие-то данные. Если мы понимаем,что нам передали объект, у которого мы можем все забрать (наш объект временный и скоро будет уничтожен), тогда можем отправить его в соответствующую функцию