4.14 Приведения типов при наследовании: static_cast, dynamic_cast и reinterpret_cast. Особенности использования каждого из этих операторов. Пример ситуации, когда все три этих оператора ведут себя по-разному. Способы приведения “вверх”, “вниз”, и “вбок” по иерархии наследования. Особенности использования данных операторов при приватном наследовании, множественном наследовании.

- **static_cast**: принимает решение на этапе компиляции и может, в частности:
  - Вызвать конструктор или определённый пользователем оператор преобразования типа — в частности, помеченный как explicit
  - Преобразовать тип указателя или ссылки в случае наследования и ряде других
  - Использовать стандартное преобразование типа. При наследовании возможно одностороннее преобразование Derived → Base. Derived неявно преобразуется в const Derived &; далее, благодаря полиморфизму, в const Base &; после чего будет вызван конструктор копирования Base. При этом произойдёт срезка: потеряются поля наследника. Обратное преобразование невозможно, если явно не написан конструктор Derived(const Base)

```
Base b;
Derived d;

static_cast<Base>(d);
//ok: Вызывается конструктор копирования Base, Derived& неявно
преобразуется к const Base&. (Конструктор сделает срезку)

static_cast<Derived>(b);
```

```
//error: Здесь хотел бы быть вызов конструктора копирования
Derived, но ссылка на родителя не преобразуется в ссылку на
потомка неявно (Base& !-> Derived&) => подходящей перегрузки
конструктора нет. Вероятно, это бы работало, если бы мы определили
свой конструктор Derived(const Base&)

static_cast<Base&>(d);
//ok: Ссылка на потомка преобразуется в ссылку на родителя даже
неявно

static_cast<Derived&>(b);
//ok: ссылку на родителя можно преобразовать в ссылку на потомка,
но явно
```

static_cast позволяет кастовать и вниз, но будет UB
static_cast проверяет легальность каста. То есть если мы пытаемся кастовать вверх к классу, от которого наследовались приватно, нам не дадут этого сделать :(

Не сработает

Так тоже

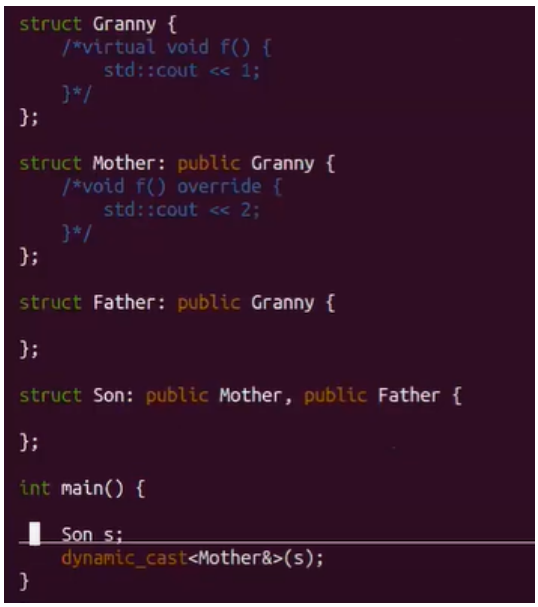
- **reinterpret_cast**: более топорно меняет тип выражения. Не выполняет никаких дополнительных операций в рантайме. Разрешаются любые преобразования указателей, не понижающие константность. Благодаря этому, в отличие от static_cast, можно преобразовывать указатель на наследника к родителю при private наследовании

- **dynamic_cast** dymamic_cast это такое преобразование типов, которое в runtime проверяет, действительно ли тип того, на что мы сейчас указываем, совместим с типом того, к чему мы хоим скастовать. Если да, то выполняется преобразование типов, иначе - RE (можно навесить исключение).

  dynamic_cast можно делать как к ссылке, так и к указателю

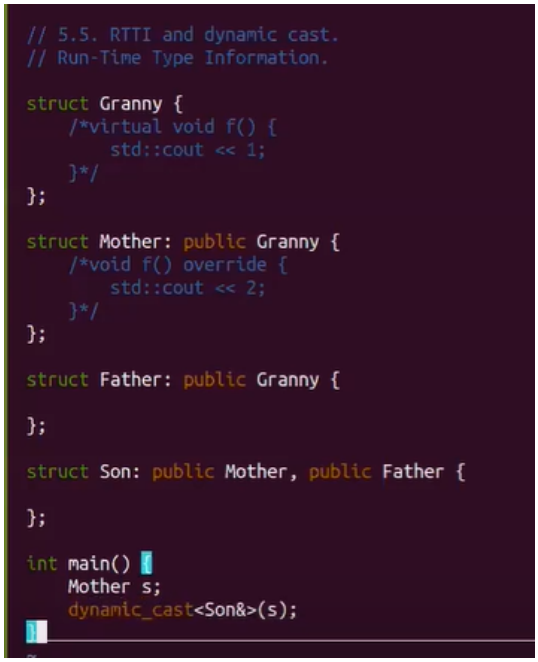
  dymamic_cast может:

  ✓ Делать каст вверх и не для полиморфных типов



Сработает

× **Не может** делать каст вниз **не для полиморфных** типов, потому что он не может проверить в runtime, возможен ли каст, ибо типы не полиморфные



Не сработает

× Если тип полиморфный и под указателем лежит родительский тип, то каст вниз сработает, но будет неуспешным (если каст прошел неуспешно, то dynamic_cast<...*>(...) выдаст nullptr, а dynamic_cast<...>(...) - ошибку std::bad_cast)

Скомпилируется, но не скастуется

Так произошло потому, что изначально под указателем лежал не сын, а мама. Но если бы мы изначально указатель на маму проинициализировали сыном, все бы сработало

✓ (То же самое, но по указателю на маму лежит сын)

```
struct Granny {
    virtual void f() {
        std::cout << 1;
    }
};

struct Mother: public Granny {
    /*void f() override {
        std::cout << 2;
    }*/
};

struct Father: public Granny {

};

struct Son: public Mother, public Father {

};

int main() {
    Mother* pm = new Son();

    std::cout << dynamic_cast<Son*>(pm);

    delete pm;
}
```

Сработает

✓ И вот так dynamic_cast тоже со всем справится

```
struct Granny {
    virtual void f() {
        std::cout << 1;
    }
};

struct Mother: public Granny {
    /*void f() override {
        std::cout << 2;
    }*/
};

struct Father: public Granny {

};

struct Son: public Mother, public Father {

};

int main() {
    Mother* pm = new Son();

    std::cout << dynamic_cast<Father*>(pm);

    delete pm;
}
```

Сработает

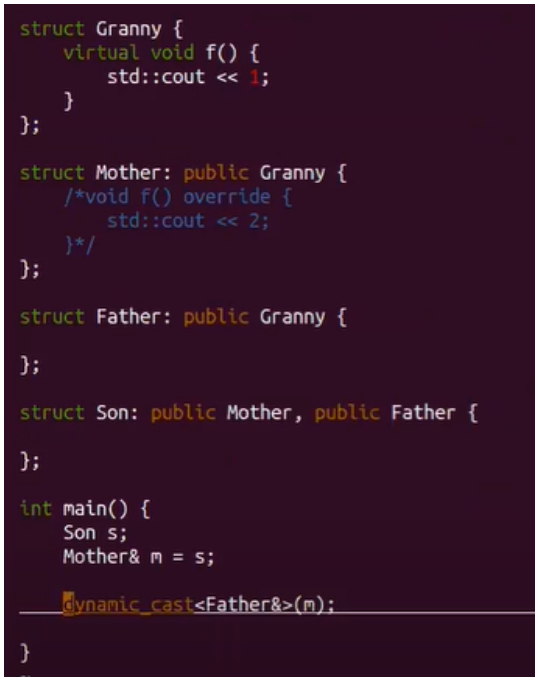
Что в этом случае происходит? dynamic_cast смотрит, возможно ли теоретически прикастовать маму к папе. Да! Это возможно, так как они лежат в одном графе, а значит, под указателем может лежать сын, и чисто теоретически мы могли бы прикастоваться. Но проверить, кто там лежит, мы можем лишь в runtime.

* Если бы под указателем на маму лежала мама, то в runtime мы бы это засекли и dynamic_cast вернул бы nullptr
* Если под указателем на маму лежит сын, то каст корректен, и все нормально кастуется, куда надо

Пример ситуации, когдавсе три этих оператора ведут себя по-разному
Собственно, последний пример работы dynamic_cast нам подходит

* **dynamic_cast** корректно отработает и прикастует маму к папе
* **static_cast** не сработает, так как static_cast работает только вверх-вниз, но не вбок (CE)
* **reinterpret_cast** выдаст UB (так как мы пытаемся кастовать несовместимые типы)

Тут мы можем кастовать не только к указателю, но и к ссылке. dynamic_cast по-прежнему молодец, reinterpret_cast все еще выдает UB, а static_cast не работает



Сработает