**Перечислите все проблемы, которые вы видите в данном коде:**

class Foo {

public:

Foo(int j){

i = new int[j];

}

~Foo(){

delete i;

}

private:

int\* i;

};

class Bar: Foo {

public:

Bar(int j){

i = new char[j];

}

~Bar(){

delete i;

}

private:

char\* i;

};

void main(){

Foo\* f = new Foo(100);

Foo\* b = new Bar(200);

\*f = \*b;

delete f;

delete b;

}

1. **Для начала попытаемся скомпилировать данный код чтобы отловить ошибки на момент компиляции.**

'Foo': no appropriate default constructor available

return type of 'main' should be 'int' instead of 'void'

'type cast': conversion from 'Bar \*' to 'Foo \*' exists, but is inaccessible

В первой ошибке говорится о том, что у Foo нет дефолтного конструктора. Так как Foo является базовым классом, а Bar производным, то при создании объекта класса Bar сначала вызывается конструктор по умолчанию класса Foo, но так как он не указан явно, компилятор попытается вызвать конструктор Foo без аргументов, которого нет в данном классе. Для решения этой проблемы нужно явно через список инициализации указать какой конструктор класса Foo использовать:

Bar(int j) : Foo(j) { …

Во втором пункте это фактически предупреждение, а не ошибка. Без int и return не выведется код ошибки, main является точкой входа в программу, а 0 укажет ОС на корректное завершение программы. Возможно некоторые компиляторы воспринимают void main как ошибку, но на тех что тестировал я (msvc, clang, intel icx), это просто предупреждение.

В третьей ошибке проблема в том, что по умолчанию класс наследуется приватно и он как бы недоступен, надо написать: class Bar: public Foo { …

A text on a white background

AI-generated content may be incorrect.

После исправления этих проблем программа будет компилироваться, но не завершится и выведется ошибка с памятью.

Следующим шагом будет исправление неправильно написанных delete в обоих классах. Так как i – это указатель на стеке, который содержит адрес выделенного на куче массива, то вызовется деструктор лишь для одного объекта, а не для всего массива. Надо написать delete[] i; в обоих классах.

Далее видим проблему с данной конструкцией: \*f = \*b, класс Foo не имеет конструктора копирования и в данном случае происходит просто копирование указателя i\_ и теперь f и b будут указывать на один и тот же массив и при вызове delete[] будет попытка 2 раза очистить одну область памяти. Реализуем оператор копирования для класса Foo:

foo& operator = (const foo& other){

if (this != &other){

delete[] i\_;

size\_ = other.size\_;

i\_ = new int[size\_];

std::copy\_n(other.i\_, size\_, i\_);

}

return \*this;

}

Если не происходит указание на одну область памяти, то вызываем деструктор, переопределяем размер массива, выделяем новую область памяти и копируем содержимое. Тут расширение resharper для VS 2022 предложило использовать copy\_n вместо copy. Я почитал про это и понял, что так выгоднее, у нас заранее известен размер массива и не нужно будет иметь итератор на конец массива.

Сделал деструктор класса Foo виртуальным так как ресурсы, выделенные в Bar, не будут освобождены без virtual и произойдет утечка памяти.

Так же реализовал правило “пяти” про которое читал у Скотта Майерса.

В часть \*f = \*b происходит срез объекта и копируется только базовая часть foo, а поля класса bar теряются. Для этого можно еще добавить виртуальное копирование чтобы не срезать уникальные поля объекта:

virtual foo\* copy() const {return new foo(\*this)};

foo\* copy() const override {return new bar(\*this)};

Остальные небольшие улучшения уже приведены в коде и добавлен файл extra.cpp в котором реализуется логика этой программы, но с умными указателями и тд.