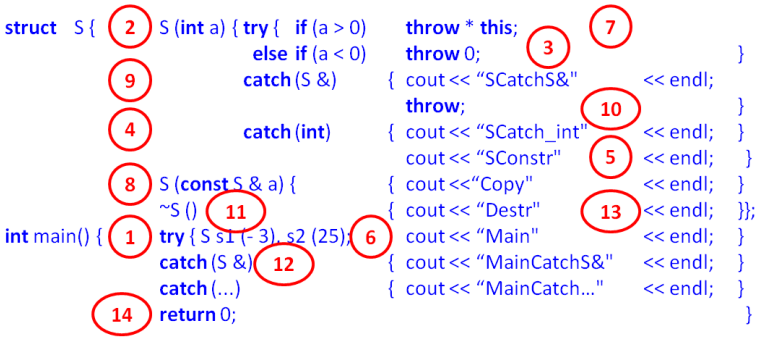
Решение проверочных задач на исключения

Данные технические задачи предназначены для освоения синтаксиса и семантики механизма исключений в C++, понимания тонкостей его взаимодействия с другими механизмами (конструкторы/деструкторы, рекурсия и т.п.). Приведённые в задачах программы не являются образцами программирования с помощью исключений, то есть прагматика механизма исключений в них отсутствует.

Задача 1.

- Что будет выдано в выходной поток при работе программы?



SCatch_int SConstr Copy SCatchS& Destr MainCatchS& Destr

Последовательность действий:

1. Вход в функцию *main()*, создание объекта *s1* (автоматическое выделение памяти на системном стеке как в Си) и его инициализация конструктором класса *S* с одним целым параметром, равным *-3*. Конструктор конструирует значение объекта, а не сам объект (по-другому говорят, что объект инициализируется). Сам объект создаётся средствами низкого уровня (соответствующих разным видам памяти – автоматической, статической, динамической), как при использовании языка Си.
2. Вход в конструктор с одним целым параметром класса *S*.
3. Проверка знака параметра *a* и выброс объекта-исключения в виде целого нуля.
4. Перехват исключения целого типа, выдача строки *"SCatch_int"*.
5. Выход из обработчика, переход на первый оператор после последней ловушки действующего ***try***-блока, выдача в поток строки *"SConstr"*.
6. Создание объекта *s2* (автоматическое выделение памяти на стеке) и попытка его инициализации конструктором с одним целым параметром класса *S* с переданным значением, равным *25*.
7. Вход в конструктор класса *S* с одним целым параметром, проверка знака параметра *a* и выполнение операции ***throw *this***.
8. Сначала происходит создание копии объекта ***s2*** [****this*** в данном контексте означает ***s2***] типа *S* и ее инициализация конструктором копирования, напечатается *"Copy"*. Созданная копия есть объект-исключение типа *S*, который летит вниз по коду к подходящей ловушке. То, что копия берётся с неинициализированного объекта, не является ошибкой – сам объект создан низкоуровневыми механизмами как в Си, и в выделенной памяти какой-то «мусор» уже есть – обычно он и копируется в такой ситуации. Поведение программы в данном случае (когда объект не проинициализирован и идёт обращение к его значению) *определяется реализацией*, это не является *неопределённым поведением* (не *undefined behavior*).
9. Вход в ловушку ссылочного типа *S&*, вторая копия не делается, обработчик выдаёт в поток строку *"SCatchS&"*.
10. Повторный выброс (проброс) перехваченного объекта-исключения типа *S*. Выход из обработчика, исключение покидает конструктор объекта *s2* класса *S* (значение объекта *s2* не сконструировано!), поиск динамической ловушки (возврат в функцию *main()*, выход из ***try***-блока функции *main()*). При выходе происходит свёртка стека: сначала автоматический объект *s2* уничтожается, но деструктор перед его уничтожением не вызывается, так как его значение не было сконструировано, затем должен быть уничтожен (выброшен из стека) объект *s1*.

11. Перед уничтожением автоматического объекта *s1* проработает деструктор *~S()*, выдав в выходной поток строку *“Destr”* (деструкторы не разрушают сам объект, они разрушают его значение, объект разрушается низкоуровневыми механизмами, такими же как в Си).
12. Перехват исключения типа *S* ловушкой ссылочного типа *S&*, в поток будет выдана строка *“MainCatchS&”*.
13. Завершение работы обработчика с уничтожением объекта-исключения, для которого вызывается деструктор *~S()*, который выдаёт в поток строку *“Destr”* и передаёт управление первому оператору после последнего перехватчика действующего ***try***-блока, то есть на оператор ***return*** *0;*
14. Возврат из функции *main()*.

Ответ. Будет выдано: SCatch_int ↵ SConstr ↵ Copy ↵ SCatchS& ↵ Destr ↵ MainCatchS& ↵ Destr

Задача 2.

- Что будет выдано в выходной поток при работе программы?

```
class Ex { int code; public: Ex (int i)            : code (i);                {}
                             Ex (const Ex & ex) : code (ex.code)           {}
                             int  Get ()  const    { return code;             } };
struct Ex90 : Ex {           Ex90 () : Ex (90)                              {} };
void f ()                    { throw Ex90 ();      cout << "dog"     << endl; }
void t ()                    { try { f (); }
                               catch (Ex90 & x)  { cout << "cat"     << endl;
                                                   throw Ex (x.Get () + 1);
                                                   cout << "sheep"   << endl; }
                             catch (Ex &)      { cout << "horse"   << endl; }
                                                 cout << "cow"     << endl; }
int main()  { try { t (); }
              catch (Ex & x)  { cout << "elephant" << x.Get () << endl;  }
              catch (...)     { cout << "wolf"                << endl  }
              return 0;                                                  }
```

cat elephant 91

Последовательность действий:

1. Вход в функцию *main()*, вызов функции *t()* из функции *main()*.
2. Вызов функции *f()* из функции *t()*.
3. Вычисление операнда операции ***throw***:
4. Явный вызов конструктора означает создание временного объекта класса с инициализацией конструктором умолчания производного класса ***Ex90()***, который предварительно вызовет конструктор базового класса с целым параметром *90*.
5. Запуск конструктора базового класса ***Ex*** с одним параметром, равным *90*. Установка значения закрытого поля *code* временного объекта (*=90*).
6. Работа конструктора производного класса.
7. Возбуждение исключительной ситуации и создание объекта-исключения, который будет копией созданного в выражении ***throw*** временного объекта, это исключение инициализируется автоматически сгенерированным конструктором копирования, который предварительно вызывает конструктор копирования базового класса ***Ex***.
   Далее объект-исключение летит вниз по коду в поисках подходящей ловушки, а временный объект, созданный в выражении операнда ***throw***, исчезает. Перед его исчезновением проработает автоматически сгенерированный деструктор *~Ex90()*, который в конце своей работы вызовет автоматически сгенерированный базовый деструктор *Ex()* (оптимизирующий компилятор сразу создаст объект-исключение без промежуточного временного объекта, но обязательно проверит, что конструктор копирования доступен, иначе будет зафиксирована ошибка компиляции).
   Безуспешный поиск статической ловушки – исключение покидает функцию *f()*, и вылетает в точке вызова *f()* в функции *t()*, далее выполняется поиск динамической ловушки.
8. Вход в ловушку производного типа *Ex90&*, выдача в выходной поток строки *“cat”*.
9. Вычисление операнда операции ***throw***, обращение к методу *Get()* объекта-исключения, возвращающего целое значение *90*.
   Создание временного объекта с инициализацией конструктором с одним целым параметром базового класса *Ex* со значением, равным *91*.
10. Установка конструктором с одним целым параметром базового класса *Ex* значения закрытого поля *code* объекта-исключения (*=91*).

11. Создание объекта-исключения, инициализируемого конструктором копирования (фактически, это копия созданного в операции ***throw*** временного объекта). Для созданного нового исключения выбирается первый по тексту подходящий обработчик, временный объект из операции ***throw*** уничтожается, но перед его уничтожением работает деструктор *~Ex()*. Оптимизирующий компилятор сразу создаёт объект-исключение без промежуточного временного объекта, но обязательно проверяет доступность конструктора копирования, в противном случае фиксирует ошибку компиляции.
    Новое исключение покидает обработчик, а старое исчезает, но перед его исчезновением работает деструктор *~Ex90()*, который в конце своей работы вызывает автоматически сгенерированный базовый деструктор *Ex()*.
12. Выход из функции *t()*, поиск динамической ловушки в вызвавшей функции *main()* и вход в ловушку типа *Ex&*. Выдача в выходной поток строки *"elephant"*.
13. Обращение к методу *Get()* объекта-исключения, возвращающего целое значение *91*, а также выдача этого значения в выходной поток. Управление покидает обработчик, объект-исключение уничтожается, и перед его уничтожением работает автоматически сгенерированный конструктор *~Ex()*.
14. Переход на первый оператор после последнего обработчика исключений функции *main()*, завершение работы функции *main()*.

Ответ. В выходной поток будет выдано: cat ↵ elephant ↵ 91 ↵

Задача 3.

- Что будет выдано в выходной поток при работе программы?

```
class A { public:  A () {                          cout << 1 << endl; } }   3   6
class B:  public   A { public:  B (int n) {   2
                          try {  if (n == 0) throw * this;   4
                                 if (n > 11) throw 11;     }
                          catch (int)          {  cout << 2 << endl; }
                          catch (B &)          {  cout << 3 << endl; throw; }   8
                                                  cout << 4 << endl;
                                                                          }
                          5   B (B &)   7   {  cout << 5 << endl; }
                              ~B ()         {  cout << 6 << endl; }   10
};
int main () {      1   try { B b (0);   B c (3);   }
                          catch (...)          {  cout << 7 << endl; }   9
                                        11     cout << 8 << endl;
}
```

1 1 5 3 7 6 8

Последовательность действий:

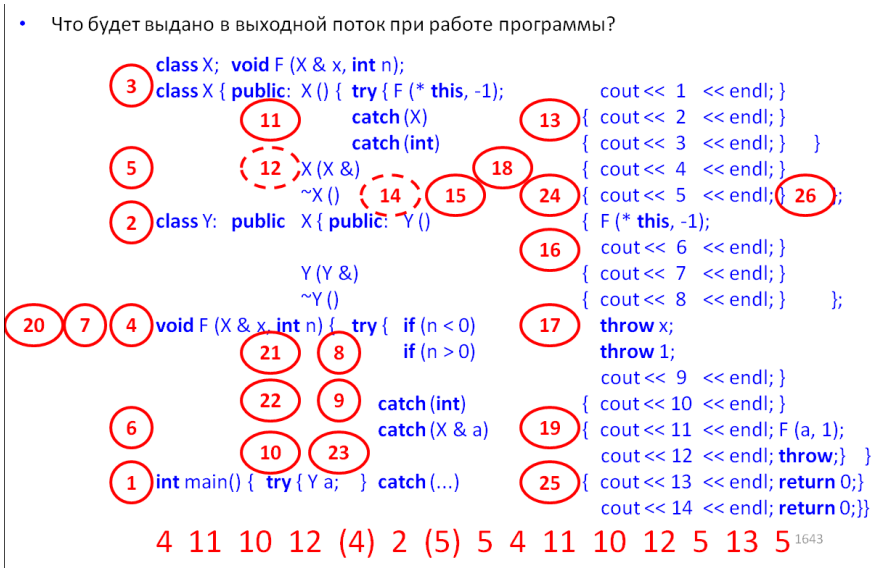
1. Создание объекта *b* производного класса *B* в функции *main()*.
2. Запуск конструктора преобразования производного класса B (с одним целым параметром, равным *0*).
3. Обращение к конструктору умолчания базового класса *A* и печать числа *1*, корректное его завершение. Возврат в тело конструктора производного класса *B*.
4. Проверка в конструкторе преобразованиям класса *B* значения параметра и возбуждение исключительной ситуации типа *B* (через значение ***this** доступна базовая часть объекта *b,* она и копируется в объект-исключение): создание объекта-исключения,
5. Инициализация объекта-исключения конструктором копирования класса *B*, который предварительно вызывает конструктор умолчания класса *A*.
6. Обращение к конструктору умолчания базового класса *A* и печать числа *1*, корректное завершение работы базового конструктора.
7. Возврат в тело конструктора копирования класса *B* и выдача в выходной поток числа *5*.

Для созданного исключения ищется первый по тексту подходящий перехватчик, которым оказывается обработчик исключений типа *B&*.

8. Обработчик выдаёт в выходной поток число *3*. Проброс исходного объекта типа *B* операцией ***throw*** без операнда. Исключение покидает конструктор и вылетает в функцию *main()* в точке описания объекта *b* (объект *c* создан не будет).
   Далее исключение летит вниз по коду и пересекает закрывающую фигурную скобку ***try***-блока – в этот момент происходит свёртка стека, объект *b* уничтожается, и перед его уничтожением работает деструктор базового класса *A* (он сгенерирован автоматически и ничего в поток вывода не выдаёт).
   Деструктор производного класса *B* для объекта *b* не вызывается, так как значение объекта *b* не было полностью сконструировано (исключение покинуло конструктор *B*).
9. Перехват исключительной ситуации типа *B* в объемлющем блоке (динамически) ловушкой с многоточием (ловит все исключения) и выдача в выходной поток числа *7*.
10. Завершение работы обработчика и вызов для исчезающего объекта-исключения деструкторов класса *B* (с выдачей в выходной поток *6*) и класса *A* (без выдач в поток).
11. Выдача функцией *main()* в выходной поток числа *8*.

Ответ. В поток будет выдано: 1 ↵ 1 ↵ 5 ↵ 3 ↵ 7 ↵ 6 ↵ 8 ↵

Задача 4.



Последовательность действий:

1. Вход в функцию *main()* и в ***try***-блок, создание объекта *a* и вызов для его инициализации конструктора умолчания производного класса *Y*.
2. Вход в конструктор производного класса *Y*, вызов конструктора базового класса *X*.
3. Вход в конструктор базового класса *X*, вызов функции *F(*this,-1)*. Указатель ***this*** (типа *X**) может использоваться, несмотря на то, что конструктор ещё не отработал. Память под объект уже отведена системой на стеке, объект имеет вполне конкретный адрес, который может быть доступен с помощью указателя ***this***. Значение, размещаемое по этому адресу до инициализации, в большинстве реализаций будет случайным «мусором», ранее записанным в данную область памяти другими программами.
4. Вход в функцию *F()*, проверка знака второго параметра *n* и выброс объекта-исключения типа *X*, являющегося копией параметра *x* (то есть копией базовой части объекта *a*).
5. Создание объекта-исключения типа *X* и его инициализация – работа конструктора копирования в классе *X*, выдача в поток числа *4*.
6. Вход в ловушку ***try***-блока функции *F()* для ссылочного типа *X&*, выдача обработчиком в выходной поток числа *11*, рекурсивный вызов функции *F(a,1)*.
7. Вход в функцию *F()*, проверка знака второго параметра *n*.
8. Выброс объекта-исключения целого типа (*=1*).
9. Вход в ловушку функции *F()* для целого типа, выдача в поток числа *10*, переход на первый оператор функции *F()*, стоящий после последнего обработчика текущего блока, то есть на конец функции *F()*.
10. Возврат из вызова функции *F(a,1)*, выдача в поток числа *12*, проброс операцией ***throw*** без операнда исходного объекта-исключения. Исключение покидает функцию *F()*.

11. После выхода из функции *F()*, исключение вылетает в конструкторе умолчания базового класса *X* в точке вызова *F(*this,-1)*, далее летит вниз и перехватывается ловушкой для типа базового класса *X*.
12. Создание копии объекта-исключения конструктором копирования класса *X*. Такое копирование может не выполняться при использовании оптимизирующего компилятора, так как сама копия в ловушке не именована и не может быть использована в обработчике. Однако даже оптимизирующим компилятором проверяется доступность конструктора копирования класса *X*. Если переместить этот конструктор в закрытую часть интерфейса класса *X*, компилятор зафиксирует ошибку доступа.
13. Выдача в блоке обработчика числа *2* в выходной поток.
14. Выход из обработчика типа базового класса *X*. Уничтожение копии объекта-исключения типа *X*, созданного ничего не оптимизирующим компилятором на шаге *12*, с выдачей в поток числа *5*.
15. Уничтожение самого объекта-исключения типа базового класса *X*, созданного на шаге *5*, с выдачей в поток числа *5*, деструктором класса *X*, завершение работы конструктора базового класса *X*. Базовая часть значения объекта *a* из функции *main()* сконструирована.
16. Начало выполнения тела конструктора производного класса *Y*, вызов функции *F(*this,-1)* (фактический параметр ****this*** имеет тип *Y*, формальный параметр имеет тип *X&*, такая передача возможна, так как базовый класс однозначен и доступен).
17. Вход в функцию *F()*, проверка знака второго параметра *n* и выброс объекта-исключения, являющегося копией параметра *x* (то есть копией базовой части объекта *a*).
18. Создание объекта-исключения типа *X* и его инициализация – работа конструктора копирования в классе *X*, выдача в поток числа *4*.
19. Вход в ловушку ***try***-блока функции *F()* для ссылочного типа *X&*, выдача в поток числа *11*, рекурсивный вызов функции *F(a,1)*.
20. Вход в функцию *F()*, проверка знака второго параметра *n*.
21. Выброс объекта-исключения целого типа (*=1*).
22. Вход в перехватчик функции *F()* для целого типа, выдача в поток числа *10*, переход на первый оператор функции *F()*, стоящий после последнего перехватчика текущего блока, то есть на конец функции *F()*.
23. Возврат из вызова функции *F(a,1)*, выдача в поток числа *12*, проброс операцией ***throw*** без операнда исходного объекта-исключения типа *X*. Исключение покидает конструктор *Y()*, значение объекта *a* не сконструировано полностью – создана только его базовая часть. Исключение вылетает в функции *main()* в точке описания объекта *a* и летит вниз по коду к первой подходящей ловушке.
24. Когда исключение пролетает над закрывающей фигурной скобки ***try***-блока функции *main()*, происходит свёртка стека – уничтожается объект *a*, и перед его уничтожением работает базовый деструктор *~X()*, выдавая в выходной поток число *5*, деструктор производного класса *~Y()* не вызывается, так как значение объекта *a* не было полностью сконструировано.
25. Вход в единственную ловушку всех исключений в функции *main()*. Выдача обработчиком в поток числа *13* и выполнение оператора ***return*** функции *main()*, то есть выход из обработчика и завершение программы.
26. При завершении обработчика обработанное исключение исчезает, и перед этим работает деструктор базового класса *~X()*, выдающий в выходной поток число *5*.

Ответ. В поток будет выдано: 4 ↵ 11 ↵ 10 ↵ 12 ↵ (4 ↵) 2 ↵ (5 ↵) 5 ↵ 4 ↵ 11 ↵ 10 ↵ 12 ↵ 5 ↵ 13 ↵ 5 ↵
(В скобках отмечены те печати, котрые будут опущены оптимизирующим компилятором.)