Язык программирования Си++

Иванов А.П., Князева О.С.

Семинар 8. Пространства имен. Обработка исключений. Оператор преобразования типа и explicit-конструктор.

1. Пространства имен Си++

При создании программ коллективами из нескольких программистов возникает своеобразная проблема: разные программисты могут завести одно и то же имя (функции, класса, глобальной переменной, метки и т.п.) для своих собственных целей. В таком случае возникает *конфликт имен*: линковщик не может создать единую программу, так как одно и то же имя используется по-разному в разных частях программы.

Для преодоления конфликта имен в языке программирования Си++ введен механизм группирования имен в логически связанные группы имен или *пространства имен*.

Пространство имен объявляется с помощью ключевого слова `namespace` следующим образом:

```
// ns_head.h

namespace my_names {

float my_fun( double A[], int dim );

class Ratio {
public:
  int a,b;
  ............
};

class other_class;

............

} // my_names namespace end
```

При этом в пространство имен достаточно включить только заголовок класса или функции, тело класса и функции можно объявить позже, обычным порядком.

Пространство имен является областью видимости, то есть, без явного указания пространства имен объекты, определенные в этом пространстве имен просто не видны, соответствующие имена считаются неопределенными. Однако в пределах пространства имен все имена из него можно использовать без ограничений и указания дополнительных префиксов.

Напротив, чтобы воспользоваться объектом (функцией, классом и т.п.) за пределами данного пространства имен нужно явно указать префикс пространства имен:

```
#include <math.h>
#include "ns_head.h"

void main()
{
```

```
  ............
  double A[10];
  float y = my_names::my_fun(A,10);
  my_names::other_class obj;
  ............
}
```

Здесь если пространства имен не использовать, то для функции `sin()` возникнет конфликт имен с функцией `sin()` из системной математической библиотеки.

Если имена данного пространства имен в каком-то модуле используются часто, то можно объявить его используемым «по умолчанию» в данном модуле (функции, блоке программного кода и т.п.):

```
#include <math.h>
#include "ns_head.h"

using namespace my_names;

void main()
{
  ............
  double A[10];
  float y = my_fun(A,10);
  other_class obj;
  ............
}
```

Пространства имен – открыты, то есть, их можно пополнять новыми именами в любой момент после их объявления (в том числе – в разных заголовочных файлах):

```
namespace my_names {
  float my_fun( double A[], int dim );
  class other_class;
}

void main()
{
  // здесь можно использовать только my_fun() и other_class.
  ............
}

namespace my_names {
  double summa( double A[], int dim );
  class Ratio;
}

void f()
{
  // здесь можно использовать и my_fun() и summa(), other_class и Ratio.
  ............
}
```

Все имена из системной библиотеки языка Си (такие, как `printf()`, `sin()`, `strlen()`) помещены в стандартное пространство имен `std`, подключенное директивной `using` по умолчанию в заголовочных файлах системных библиотек.

Поэтому, если возникает конфликт имен с системной функцией, его можно разрешить явно, полностью указывая префикс пространства имен:

```
#include <math.h>

namespace my_names {
  float sin( float x );
}

void main()
{
  ............
  float  y = my_names::sin(3.14159/8); // вызывается собственная функция
  double z = std::sin(3.14159/8); // вызывается системная функция
  ............
}
```

Напомним, что потоки ввода-вывода языка Си++ тоже находятся в системном пространстве имен `std`, только, в отличие от функций системных библиотек языка Си, стандартное пространство имен для потоков Си++ не подключено автоматически, поэтому мы всегда употребляли директиву подключения этого пространства явно:

```
#include <iostream>
using namespace std;
```

2. Обработка исключений

При работе программы часто возникают ситуации, которые нарушают нормальное ее течение: то пользователь ввел что-то неожиданное, то программист передал в функцию неправильный параметр, то файла, который нужно открыть, нет на месте.

Особенно сложные ситуации возникают, когда ошибка возникает в каком-либо «глубоком» вызове некоторой функции, а с тем, что делать с этой ошибкой можно определиться только на несколько уровней выше по дереву вызовов функций.

Обычная практика в таких случаях заключается в том, чтобы определить целочисленные коды возврата функций, каждый из которых обозначает ту или иную ошибку. Функции, столкнувшиеся с ошибкой, возвращают эти коды наверх по иерархии вызовов до тех пор, когда на эти ошибки может быть организована какая-то разумная реакция.

Но и тут остаются проблемы:

- конструкторы и деструкторы классов не могут возвращать никаких значений;
- целочисленного кода возврата зачастую мало для того, чтобы точно дать диагностировать проблему: скажем, если не открылся какой-то файл, то помимо фиксации факта «какой-то файл не открылся» хорошо бы явно сохранять информацию о его имени;
- такой код обработки ошибочных состояний может занимать до половины всего кода программы, программа делается чересчур громоздкой (т.к. у каждого вызова приходится проверять его код возврата).

В языке программирования Си++ предусмотрен отдельный механизм для обработки ошибочных, или, точнее, исключительных (exceptions) состояний программы более гибким и удобным образом.

Рассмотрим фрагмент программы, который преобразует целое число в символ с обработкой ошибочных ситуаций, когда переданное число больше максимально возможного кода символа:

```
// Класс, в котором будут сохраняться детальная информация по
// возникшему ошибочному состоянию
struct Range_error {
  int m_i;                    // поле, сохраняющее преобразуемое число
  Range_error( int i ) { m_i = i; } // инициализирующий конструктор
};

struct Other_error {
  ..... // класс для обработки какой-то другой ошибки
};

char to_char( int i )
{
  char c = (char)i;
  if ( i != (int)c ) // проверка на точность преобразования
      throw Range_error(i);  // возбуждение исключительного состояния
                             // нормальное исполнение функции прерывается
                             // управление передается иерархии вызвавших
                             // функций
  return c; // нормальный возврат, если ошибки не было
}

void f( int i )
{
  try { // оператор «тестовой зоны»
    .....
    char c = to_char(i); // проблемный вызов
    .....
  }
  catch( Range_error x ) { // поймали исключительное состояние
    // печатаем диагностику:
    cerr << "Try to convert to char illegal number: " << x.m_i << endl;
    cerr << flush;
  }
  catch( Other_error ) {
    // обработка какого-то другого исключительного состояния, обратить
    // внимание: имя переменной после типа исключительного состояния
    // не указано, т.е. подробной диагностики не будет!
    .....
    throw; // данный оператор после частичной обработки возбуждает
           // то же самое исключительное состояние, которое было поймано –
           // для окончательной обработки на более высоком уровне
           // иерархии вызовов
  }
  catch(...) {
    // обработка любого непойманного ранее исключительного состояния
    cerr << "Unknown exception" << endl << flush;
    exit(1); // завершение программы
  }
  ..... // продолжение нормального исполнения функции f().
}
```

В этом примере вызывающая функция ограждает блок операторов, который может вызвать исключительное состояние, блоком оператора `try`. Любое исключительное состояние, которое возникнет в пределах операторов и вызовов функций этого блока может быть перехвачено обработчиком `catch`, аргументом которого указывается тип

перехватываемого исключительного состояния, и, возможно, имя переменной, в которой будут сохранены дополнительные параметры исключительного состояния для подробной его диагностики.

Принципиально важен порядок обработчиков `catch`, так как сработает первый же подошедший по типу обработчик, прочие – не сработают. То есть, здесь аналогия с блоками `else if()` условного оператора.

В конце (раньше – бессмысленно!) можно указать необязательный блок, который поймает вообще любое возникшее исключение `catch(…)`.

С помощью механизма наследования исключительные состояния можно объединять в древовидные иерархии:

```
class math_err {
public:
  double arg1, arg2;
  char op;

  math_err( double a1, char o, double a2 ) : arg1(a1), arg2(a2), op(o) {}

};

class overflow: public math_err
{
public:
  overflow( double a1, char o, double a2 ) : math_err(a1,o,a2) {}
};

class div_by_0: public math_err
{
public:
  div_by_0( double a1, double a2 ) : math_err(a1,'/',a2) {}
};

double divide( double a, double b)
{
  if ( fabs(b) < 1.0E-10 ) throw div_by_0(a,b);
  double c = a / b;
  if ( fabs(c) > 1.0E+10 ) throw overflow(a,'/',b);
  return c;
}

void main()
{
  ............
  double x, y, z=0;

restart:
  cin >> x >> y;
  try {
    z = divide(x,y);
  }
  catch (math_err q) {
    cerr << "illegal operation: " << x << ' / ' << y << ", repeate!";
    cerr << endl << flush;
    goto restart;
  }
  ............
}
```

– так обрабатывать можно либо некоторую группу исключительных состояний целиком, либо, по желанию – отдельные исключительные состояния можно обработать отдельно, указывая блоки их обработки до общего блока группы:

```
void main()
{
  ............
  double x, y, z=0;

restart:
  cin >> x >> y;
  try {
    z = divide(x,y);
  }
  catch (div_by_0) {
    cerr << "divide by zero, result will be large!" << endl << flush;
    z = 1.0E+10;
  }
  catch (math_err q) {
    cerr << "illegal operation: " << x << ' / ' << y << ", repeate!";
    cerr << endl << flush;
    goto restart;
  }
  ............
}
```

3. Оператор преобразования типа и explicit-конструктор.

Оператор преобразования типа в классе тоже может быть перегружен:

```
class my_complex
{
protected:
  double re,im;
public:
  my_complex(double x=0.0,double y=0.0): re(x), im(y) {}
  my_complex& operator=(const my_complex& src)
      { re = src.re; im = src.im; return *this; }
  ............
};

class Ratio {
protected:
  int a, b;
public:
  Ratio( int aa = 0, int bb = 1 ) : a(aa), b(bb) {}
  ............
  operator double() const { return double(a)/double(b); }
  operator my_complex() const { return my_complex(double(*this),0); }
  ............
};

............

void main()
{
  ............
  Ratio z(3,2);
  double q = double(z) * 2; // Пример 1
```

```
  double t = 3.5 * z;        // Пример 2
  my_complex cmp; cmp = z;   // Пример 3
  ............
}
```

В первом примере оператор преобразования рационального числа к вещественному в выражении приведет к вызову перегруженного метода в классе `Ratio`. Во втором примере преобразование к вещественному числу будет подобрано компилятором неявно, но результат будет точно такой же, как и в первом примере: вызов перегруженного преобразования типа в классе `Ratio()`.

В третьем примере показывается, что оператор преобразования типа может быть перегружен не только для базовых типов данных, но и для любых других классов.

Аналогичные преобразования к данному типу могут быть реализованы и при помощи конструкторов:

```
class Ratio {
protected:
  int a, b;
public:
  Ratio( int aa = 0, int bb = 1 ) : a(aa), b(bb) {}
  Ratio( double x ) {
    b = 1000;
    a = int(x) * b;
  }
  ............
};
```

– но это только в тех случаях, когда к коду класса-цели у нас есть доступ, то есть, мы его можем модифицировать.

Вместе с тем, наличие конструкторов, создающих объект из объекта другого типа иногда может приводить к довольно неочевидным ошибкам, связанным с неявным преобразованием типов:

```
class String {
protected:
  char* buf;
  int len;
public:
  String( int l ) : buf(0), len(0) {
    if ( l > 0 ) {
        buf = new char[l];
        if ( buf ) len = l;
        memset(buf,'S',len-1);
        buf[len-1] = '\0';
    }
  }
  const char* cstr() { return buf; }
  ............
};

void print_str( String s )
{
  std::cout << s.cstr() << std::endl;
}

............
```

```
void main()
{
  ............
  String s1(1024);  // Нормально, выделяется буфер указанной длины.
  String s3('a');   // Потенциальная ошибка: автор хотел создать
                    // строку из одного символа, а получит строку,
                    // состоящую из int('a') символов.
  ............
  print_str(s1);    // OK: нормально распечатается строка
  print_str(3);     // Потенциальная ошибка: создастся строка
                    // из трех символов, она и распечатается
}
```

Для устранения возможности подобных ошибок в языке Си++ вводится возможность запретить неявное преобразование типа для избранных конструкторов:

```
class String {
protected:
  char* buf;
  int len;
public:
  explicit String( int l ) : buf(0), len(0) {
    if ( l > 0 ) {
        buf = new char[l];
        if ( buf ) len = l;
        memset(buf,'S',len-1);
        buf[len-1] = '\0';
    }
  }
  const char* cstr() { return buf; }
  ............
};
............
void main()
{
  ............
  String s1(1024);  // Нормально, выделяется буфер указанной длины.
  String s3('a');   // Увы, это явный вызов конструктора, здесь конверсия
                    // из 'a' в соответствующий целый код произойдет ДО
                    // вызова конструктора, поэтому explicit не поможет.
  ............
  print_str(s1);    // OK: нормально распечатается строка
  print_str(3);     // Error: теперь здесь ошибка компиляции:
                    // неявное создание строки из числа запрещено exlicit.
}
```

Решением будет использование ключевого слова `explicit` перед заголовком конструктора. Это ключевое слова объявляет конструктор «явным» и запрещает неявное преобразование к данному типу с использованием указанного конструктора.

Типовое задание: модифицировать программу задания предыдущего семинара по потокам ввода-вывода, так, чтобы она применяла обработку исключительных состояний для реакции на любые ошибочные файловые операции, а также операции разбора формата файлов.

1. Вариант

Создайте класс целых чисел, определите в нем функции арифметических операций. В классе должны генерироваться исключения при делении на 0, выходе за пределы максимального значения целого числа, при потере точности при делении.

Потеря точности происходит, когда меньшее число делится на большее, при этом получается значение, равное целочисленному нулю, при попытке потом умножить любое, сколь угодно большое число на этот результат получится опять ноль, что почти всегда является ошибкой.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

2. Вариант

Создайте класс рациональных чисел. В классе должны генерироваться исключения при знаменателе равном 0, при потере точности при приведении типов к целому числу.

Потеря точности происходит, когда меньшее число делится на большее, при этом получается значение, равное целочисленному нулю, при попытке потом умножить любое, сколь угодно большое число на этот результат получится опять ноль, что почти всегда является ошибкой.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

3. Вариант

Создайте класс динамический массив. В классе должно генерироваться исключение при попытке получить значение по индексу, выходящему за пределы массива.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

4. Вариант

Создайте функцию поиска по значению в STL-контейнерах типа `vector`, которая должна генерировать исключение, если искомое значение не найдено.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

5. Вариант

Создайте функцию, вычисляющую факториал от целого числа, которая должна генерировать исключение, если решение вышло за пределы максимального значения целого числа.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

6. Вариант

Создайте программу калькулятор, вычисление должно продолжаться до тех пор, пока не введено слово `quit`, реализовать прерывание счета с помощью исключения. Исключение также должно генерироваться при ошибочном вводе и при делении на 0.

7. Вариант

Создайте класс времени (часы, минуты, секунды), конструктор должен генерировать исключение при неправильно заданном времени.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

8. Вариант

Создайте функцию поиска корня уравнения вида `f(x)=0` методом деления отрезка пополам. Функция должна генерировать исключение, если корня нет.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

9. Вариант

Создайте класс даты (день, месяц, год), конструктор должен генерировать исключение при неправильно заданной дате.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

10. Вариант

Создайте класс динамический массив, определите функцию сложения массивов. Функция должна генерировать исключения, если массивы имеют разную длину.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

11. Вариант

Создайте класс прямых на плоскости, определите функцию нахождения точки пересечения двух прямых. Функция должна генерировать исключение, если прямые параллельны или совпадают.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

12. Вариант

Создайте класс матриц размера 2x2, определите функцию обращения матриц. Функция должна генерировать исключение, если определитель матрицы равен 0.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

13. Вариант

Создайте функцию, считывающую массив целых чисел из файла и распечатывающею его на экран. Функция должна генерировать исключение, если файл не найден и при неверном задании массива в файле (текст вместо чисел, `double` вместо `int` и т.д.).

Нужно написать программу, в которую пользователь вводит имя файла и которая затем демонстрирует обработку исключений.

14. Вариант

Создайте функцию, считывающую матрицу целых чисел из файла и распечатывающею ее на экран. Матрица произвольных размеров, размеры матрицы должны задаваться в виде 2 чисел в начале файла. Функция должна генерировать исключения, если файл не найден, если неверно заданы размеры матрицы (нулевые или вместо `int` в начале файла идет текст) и при неверном задании матрицы в файле (например, размеры матрицы не совпадают с количеством данных в файле).

Нужно написать программу, в которую пользователь вводит имя файла и которая затем демонстрирует обработку исключений.

15. Вариант

Создайте функцию, подсчитывающую наибольший делитель 2 чисел, функция должна генерировать исключение, если одно из чисел отрицательно или равно 0.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

16. Вариант

Создать функцию, которая считывает из файла числовые данные, записанные в 2 колонки, и переписывает их в два других файла. Функция должна генерировать исключение, если файл не найден, пуст или если данные записаны неверно, т.е. их нечетное число, или на месте данных в файле встречается текст.

Нужно написать программу, в которую пользователь вводит имя исходного файла (имена выходных файлов получаются дописыванием суффикса в конец имени исходного файла) и которая затем демонстрирует обработку исключений.

17. Вариант

Создать функцию решения системы 3-х линейных уравнений с тремя неизвестными. Функция должна генерировать разные исключения, если система недоопределена или плохо обусловлена (определитель системы меньше 10^{-3}).

Нужно написать программу, которая считывает данные из файла и демонстрирует обработку исключений.

18. Вариант

Создать функцию, считывающую из файла массив вещественных чисел `(x)` и распечатывающую на экран массив `log(x)`. Функция должна генерировать исключение, если файл не найден или пуст, а так же если в файле встретится значение `x <= 0`.

Нужно написать программу, которая считывает данные из файла и демонстрирует обработку исключений.

19. Вариант

Создать функцию, считывающую из файла массив вещественных чисел `(x)` и распечатывающую на экран массив `asin(x)`. Функция должна генерировать исключение, если файл не найден или пуст, а так же если в файле встретится значение `|x| > 1`.

Нужно написать программу, которая считывает данные из файла и демонстрирует обработку исключений.

20. Вариант

Функция принимает строку вида `3 + 5 – 3 / 6 * 10 =`. и возвращает результат счета (без учета приоритета операций) в виде текстовой строки. Функция должна генерировать исключение при неверном вводе, при делении на 0, при потере точности.

Потеря точности происходит, когда меньшее число делится на большее, при этом получается значение, равное целочисленному нулю, при попытке потом умножить любое, сколь угодно большое число на этот результат получится опять ноль, что почти всегда является ошибкой.

Нужно написать программу, которая считывает данные из файла и демонстрирует обработку исключений.

21. Вариант

Создайте класс окружностей на плоскости и STL-контейнер таких окружностей. Создайте функцию добавления новой окружности в контейнер. Функция должна генерировать исключение, если окружность пересекается с любой другой или если окружность задана не верно (например, радиус `R <= 0`).

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

22. Вариант

Создать свой класс `myset` (массив уникальных ключей). Определите функцию поиска и добавления ключа в массив `myset`, функция поиска должна генерировать исключение, если ключ не найден, функция добавления должна генерировать исключение, если такой ключ уже встречается.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

23. Вариант

Создайте класс студенческих групп (количество студентов, номер, фамилия старосты), переопределите операции «+=», «−=», меняющие количество студентов в группах. В классе должны генерироваться исключения при отрицательном количестве студентов или неверном задании номера группы.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

24. Вариант

Создайте класс квадратных трехчленов $ax^2+bx+c=0$, определите в нем функцию поиска корней уравнения, функция должна генерировать исключения при $D<0$, при $D=0$, при $a=0$ и т.д.

Нужно написать программу, которая получает ввод данных от пользователя и демонстрирует обработку исключений.

25. Вариант

Создайте функцию, которая перемножает две произвольные матрицы и генерирует исключение, если размерности матриц не соответствуют правилам матричного произведения.

Нужно написать программу, которая считывает исходные матрицы из двух файлов, а результат записывает в третий файл.