Обработка ошибок и исключения

Тема 8

Типы ошибок

- Ошибки программирования (**Software Errors**)
- Ошибки, связанные с нехваткой или недоступностью ресурсов (**Run-Time Errors**)

Традиционная обработка ошибок

- завершить программу
- возвратить значение, трактуемое как "ошибка"
- возвратить нормальное значение и оставить программу в неопределенном состоянии
- вызвать функцию, заданную для реакции на такую ошибку

Завершение программы

```
#include <cassert>


int Stack::pop()
{
  assert( head != 0 );
  Node* temp = head;
  head = head->next;
  int val = temp->val;
  delete temp;
  return temp->val;
}
```

```
stack_assert: simple_stack.cpp:61:
int Stack::pop (): Assertion 'head!= 0' failed.
Abort (core dumped)
```

Возвратить значение «ошибка»

```
int Stack::pop()
{
  if( head == 0 )
    return -1;
  Node* temp = head;
  head = head->next;
  int val = temp->val;
  delete temp;
  return temp->val;
}
```

Возвратить значение «ошибка»

```
enum StackState { OK, UNDERFLOW, OVERFLOW };
```

```
int Stack::pop (StackState *err)
{
  if( head == 0 ) {
     err = UNDERFLOW;
     return -1;
  }
  Node* temp = head;
  head = head->next;
  int val = temp->val;
  delete temp;
  return val;
}
```

```
StackState Stack::pop (int *val)
{
  if( head == 0 )
     return UNDERFLOW;
  Node* temp = head;
  head = head->next;
  * val = temp->val;
  delete temp;
  return OK;
}
```

Оставить программу в ненормальном состоянии

```
enum StackState { OK, UNDERFLOW, OVERFLOW }state;
```

```
int Stack::pop ()
{
  if( head == 0 ){
     state = UNDERFLOW;
     return -1;
  }
  Node* temp = head;
  head = head->next;
  int val = temp->val;
  delete temp;
  err = OK;
  return val;
}
```

```
int main()
{
  Stack stack;
  int c = stack.pop();
  if (state != OK)  {
    // обработка ошибки
  }
  else  {
    // OK
  }
}
```

Вызвать функцию обработки ошибки

У функции обработки ошибок есть только три первые альтернативы, как обрабатывать ошибку

Проблема обработки ошибок

Разработчик кода

- Способен обнаружить динамические ошибки
- Не знает, какой должна быть реакция на них

Пользователь кода

- Знает, как написать реакцию на ошибки
- Не способен обнаружить ошибки

Решение

Особые ситуации или исключения

Исключения

- Генерация сообщения об ошибке (`throw`)
- Перехват этих сообщений (`catch`)

- В программе может одновременно существовать только одно исключение

Синтаксис

```
try
{
    …
    throw Exception;    //«выбрасываем» исключения
}
catch(Exception)        //«перехватываем» исключения
{
    …
}
```

Класс Vector с исключениями

```
class Vector {
  int* p;
  int sz;
public:
  enum { max = 32000 };
  class Range
  {
  public:
     int index;
     Range(int i) : index(i) { }
    };
  class Size
  {
   public:
     int index;
     Size(int i) : index(i) { }
  };
  Vector(int sz = 1000);
  int& operator[](int i);
  ~Vector();
  };
```

особая ситуация индекса

особая ситуация "неверный размер"

Исключения в конструкторах

Если в конструкторе произошло исключение, то объект не будет создан

Исключение в конструкторе

```
Vector::Vector(int sz)
{
  if (sz < 0 || max < sz) throw Size(sz);
  if(!sz)
  {
      p = 0;
      Vector::sz = sz;
      return;
  }
  p = new int [Vector::sz = sz];
  for(int i = 0; i < sz; i++)
      p[i] = i;
}
```

Исключения и инициализация полей

```
Employee::Employee(Date d)
try
  : hire_date(d)
{
  // тело конструктора
}
catch (Date::Bad_Date) {
    // обработка неправильной даты
}
```

Исключения в деструкторах

Деструктор объекта вызывается в двух случаях –

- при нормальных условиях
- когда объект удаляется механизмом обработки исключений.

Если при уже сгенерированном исключении в деструкторе также генерируется исключение, то при передаче управления вызывающей функции C++ сразу же запускает функцию `terminate`.

Исключения в деструкторах

Исключение не должно покинуть деструктор

```
Vector::~Vector()
{
    try
    {
        …
    }
    catch(...)
    {
    }
}
```

Исключения в деструкторах

Если в деструкторе выбрасывается исключение, то нужна предварительная проверка на уже существующее исключение

```
Vector::~Vector()
{
        if(!uncaught_exception())
                throw Exception ex;
}
```

Исключение в методе класса

```
int& Vector::operator [](int i)
{
    try
    {
        if(i < 0|| sz <= i) throw Range(i);
        return p[i];
    }
    catch(...)
    {
        std::cout << "Aha" << std::endl;
    }
}
```

Создание вектора

```
Vector* CreateV(int s)
{
    Vector *vnew;
    try
    {
        vnew  = new Vector(s);
    }
    catch(Vector::Size s)
    {
        std::cerr << "Wrong vector size " << s.index << std::endl;
        vnew = CreateV(100);
    }
    catch(...)
    {
        std::cerr << "OPS" << std::endl;
    }
    return vnew;
}
```

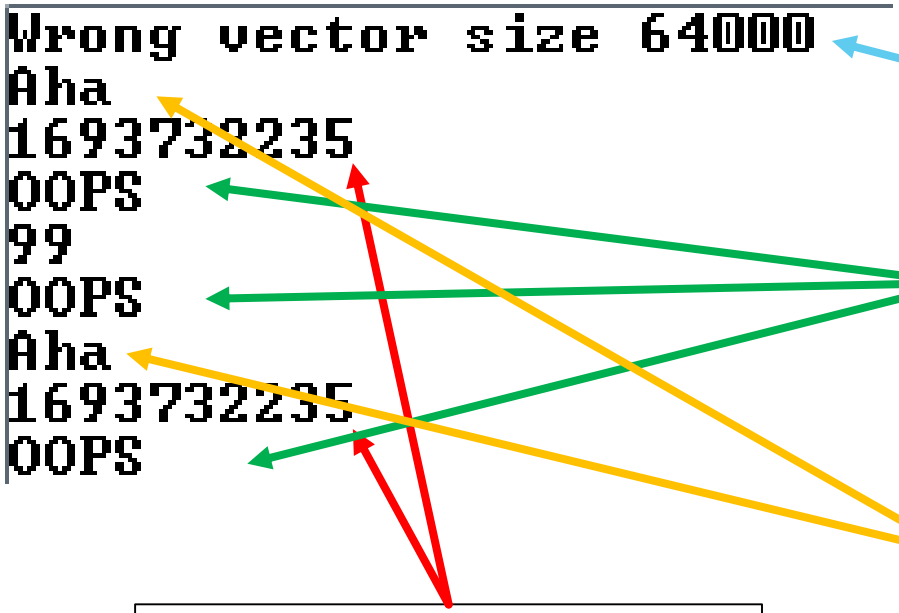
Печать вектора

```
void print(Vector& v, int i)
{
    try
    {
        std::cout << v[i] << std::endl;
        Vector *vnew = new Vector(64000);
    }
    catch(Vector::Range r)
    {
        std::cerr << "Wrong index: " << r.index << std::endl;
        print(v, 0);
    }
    catch(...)
    {
        std::cerr << "OOPS" << std::endl;
    }
}
```

Функция main

```
int main()
{
    Vector * vec[3];
    vec[0] = CreateV(0);
    vec[1] = CreateV(64000);
    vec[2] = CreateV(1000);
    print(*vec[0],6);
    print(*vec[1], 99);
    print(*vec[2],1000);
    delete vec[0];
    return 0;
}
```

Работа программы



Исключение при создании вектора

Исключение при создании вектора в функции print

Исключение при обращении по неправильному индексу

Исключение при обращении по неправильному индексу обрабатывается в перегруженном операторе [], поэтому в потоке оказывается мусор

Группировка исключений

```
class Exception {};
```

```
class VectorError: public Exception {};
```

```
class Overflow:  public VectorError{};
class WrongSize: public VectorError{};
```

```
class StackError: public Exception {};
```

```
class Underflow: public StackError{};
```

Повторная генерация исключений

```
void f()
{
  try {
    // ...
    throw Underflow();
  }
  catch (Exception& re) {
    if ( canHandleItCompletely(re) ) {
       // обработка исключения в функции
       return;
    }
    else {
       doWhatYouCanDo(re);
       throw;
    }
  }
}
```

Повторная генерация исключений

```
void g()
{
  try {
    f();
  }
  catch (StackError& re) {
     // обработка stack error
  }
  catch (VectorError& re) {
     // обработка vector error
  }
}
```

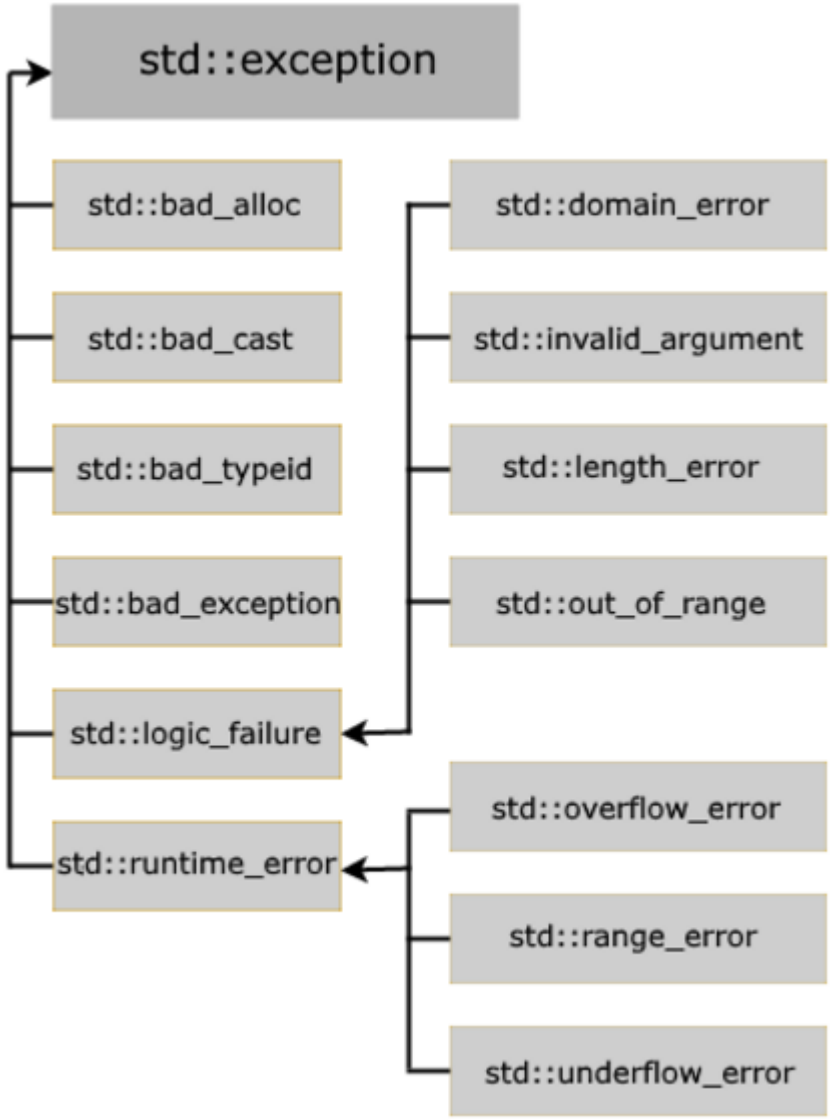
Перехват исключений

```
try {
}
catch (Overflow& re) {
}
catch (VectorError& re) {
}
catch (Exception& re) {
}
catch (...) {
}
```

Перехват исключений

```
try {
}
catch (...) {
}
catch (Exception& re) {
}
catch (VectorError& re) {
}
catch (Overflow& re) {
}
```

Встроенные типы исключений



Файлы:

- `exception`
- `stdexcept`

Конец