283

Chapter 10

Exceptions

Error Handling in C++

284

Some Error Handling Styles

There are several different ways a function may react when encountering an unexpected/erroneous situation:

- **Return an impossible value:**

```
Set   SetCreate(/* args */);   // ok: return NULL on error
int   atoi(const char *);   // What to return for "bad-str" ?
```

- **Return a special type:**

```
Result sqrt(double arg, double *res); // No composition ☹
```

- **The user reports an interest:**

```
double sqrt(double, Result *r); // r may be  NULL
```

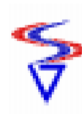
- **Provide a global mechanism:**

  E.g, `errno` in C   (should we have a single one? Many of them?)

- **Let an object hold the status:**

  E.g, `cin` in C++

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only

285

Some Error Handling Styles (cont.)

One common drawback with all

Error handling is local

The last two items need explanations

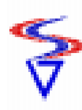
- The global mechanism loses information if not inquired immediately
- The status within an object can be retrieved just as long as this object is “alive”
  - It works for `cin` because it is a global object

286

The Challenge in Handling Errors

- **In every descent software there are cases where the function that encountered an unexpected situation has no clue about an appropriate reaction, nor does that function that called it.**
- **This is the case for practically all library functions.**

The consequence is that an *error-value* should propagate through a series of function-calls, where each call should be wrapped with error-handling code (e.g, `RoadMap() => GraphCreate() => SetCreate() => malloc()`)



287

The Challenge in Handling Errors (cont.)

- There are circumstances in C++ where none of the aforementioned error reporting styles can work:
  - In constructors (no return value and result argument is futile)
  - In overloaded operators
    - » How to report an error at

```
Rational r = (a + b)*(c + d/e);
```

- Note that error handling includes resource release before leaving a scope
  - No memory deallocation in **C**
  - No closing files in **Java**
  - **C++** provides practically anything via destructors

288

Error Handling is a Two-Stage Operation

- The implementation (supplier code) knows how to detect an error.
- The application (user code) knows how to properly handle an error.

Application

```
vector<double> Solve()
{ Polynomial<int> p;
  cin >> p;
  return p.solve();
}
```

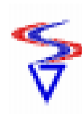
Implementation

```
int string2number(const string& s)
{ if (s == "")        // . . .
  int result(0), i(0);
  if (is_sign(s[i])) {++i;/*..*/}
  if (s[i] == '.')   // . . .
  if (!isdigit(s[i])// . . .
  // . . .
  return result;
}
```

`operator>>()` for `p` calls `string2number()` indirectly.

The functions between them are better be **ignorant** to possible errors

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

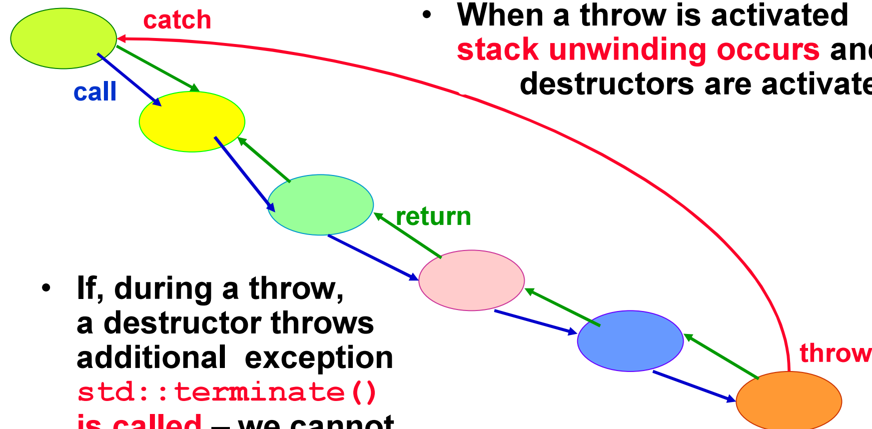


For personal use only

289

The Throw-Catch Game

- Error handling is a path parallel to the return path



- When a throw is activated stack unwinding occurs and destructors are activated

- If, during a throw, a destructor throws additional exception `std::terminate()` is called – we cannot handle two exceptions

290

The Throw-Catch Game (first try)

- The implementation (supplier code) knows how to detect an error and throws appropriate exceptions.
- The application (user code) knows how to properly handle an error by catching those it can.

Application

```
int main()
{ try {
    vector<int>
        v = Solve();
    cout << v << endl;
  }
  catch(const string& s)
  { if (s == "empty")..
    if (s == "dot")   ..
    if (s == "bad_ch").
  }
}
```

Implementation

```
int string2number(const string& s)
{ if (/**/) throw string("empty");
  int result(0), i(0);
  if (is_sign(s[i])) {++i;/*..*/}
  if (/**/) throw string("dot");
  if (/**/) throw string("bad_ch");
  // . . .
  return result;
}
```

- Middle functions intervention is eliminated
- Handling different errors is still cohesive

291

The Throw-Catch Game (second try)

Application

```
using namespace
            string_err;
int main()
try {
  vector<int>
       v = Solve();
  cout << v << endl;
  return 0;
}
 catch(const empty& e)  {
 // Do something
}catch(const dot& d)    {
 // Do something else
}catch(const bad_ch& b){
 // Do other stuff
}
```

Preparation

```
namespace string_err {
  class empty {};
  class dot    {};
  class bad_ch {};
}
```

Implementation

```
int string2number(const string& s)
{ using namespace string_err;
  if (/**/) throw empty();
  int result(0), i(0);
  if (is_sign(s[i])) {++i;/*..*/}
  if (/**/) throw dot();
  if (/**/) throw bad_ch();
  // . . .
  return result;
}
```

What did we gain?

292

The Throw-Catch Game (second try cont.)

Application

Once catches are not cohesive they can be moved around

```
using namespace
          string_err;
int main()
try {
  vector<int>
       v = Solve();
  cout << v << endl;
  return 0;
}
 catch(const dot& d)  {
 // Do something else
}catch(const bad_ch& b){
 // Do other stuff
}
```

```
vector<double> Solve()
{ Polynomial<int> p;
  try {
    cin >> p;
  } catch(const empty& e) {
    return vector<double>();
  }
  return p.solve();
}
```

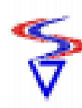
Preparation

```
namespace string_err {
  class empty {};
  class dot    {};
```

Implementation

```
...ring& s)
...rr;
.../*..*/}
  if (/**/) throw dot();
  if (/**/) throw bad_ch();
  // . . .
  return result;
}
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only

293

The Throw-Catch Game (third try)

Application

A base class catch will be activated for any derived one

```
int main()
try {
   vector<int>
        v = Solve();
   cout << v << endl;
   return 0;
}
 catch(const dot& d) {
 // Do something else
}catch(const bad_ch& b
 // Do other stuff
}catch(const str_err&s){
 // A general response
}catch(...){
 // Ultimate generality
}
```

```
vect
{ Po
  try {
    cin >> p;
  } catch(const empty& e) {
    return vector<double>();
  }
  return p.solve();
}
```

Preparation

```
class str_err {};
class empty : public str_err {};
class dot   : public str_err {};
class bad_ch: public str_err {};
// namespaces are still useful
```

mentation

```
ring& s)

/*..*/
  if (/**/) throw dot();
  if (/**/) throw bad_ch();
  // . . .
  return result;
}
```

294

The Throw-Catch Game (final version)

Application

A good practice is to publicly derive every exception from → std::exception

```
int main()
try {
   vector<int>
        v = Solve();
   cout << v << endl;
   return 0;
}
 catch(const dot& d) {
 // Do something else
}catch(const bad_ch& b
 // Do other stuff
}catch(const str_err&s
 // A general response
}catch(exception& e){
 // Ultimate generality
}
```

```
vect
{ Po
  try {
    cin >> p;
  } catch(const empty& e) {
    cerr << e.what();
    return vector<double>();
  }
  return p.solve();
}
```

Preparation

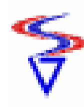
```
class str_err: public
               std::exception {};
class empty : public str_err {};
class dot   : public str_err {};
class bad_ch: public str_err {};
```

mentation

```
ring& s)

/*..*/
  if (/**/) throw bad_ch();
  // . . .
  return result;
}
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only

Error Handling at Constructors

- A constructor may throw an exception during initialization

```
String::String(char const *t)
try: len(len_eval(t)), s(strcpy(new char[len+1],t) {}
catch(NullPtr& e) {
  /*. . .*/ throw;
}
catch(std::bad_alloc& e) {
  cerr << e.what;  throw;
}
```

- A destructor should not throw an exception although it can

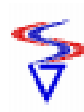
Exception Specification

- A function may provide information about exceptions that it may throw (directly or indirectly)

```
int f() throw(int,bad_alloc); // May throw any
                          // of them or derived thereof
double g() throw();   // Will throw none
void   h();           // May throw anything
```

- These specifications are checked at runtime
- If a specification is broken, the function `unexpected()` is called
- By default, `unexpected()` calls `terminate()`
- During such a process `bad_exception` may be thrown
- A destructor should not throw while an exception is active
  - if it does, `std::terminate()` is called

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only