

CS481 - Object-Oriented Programming

Exception Handling

A variety of philosophical approaches

- ❖ Exceptions can't happen!?
- Don't worry about impossible (?) conditions

1

Exception Handling

A variety of philosophical approaches

- ❖ Exceptions can't happen!?
- ❖ Laissez-faire
- Propagate error conditions so they cause someone else's code to break

2

Exception Handling

A variety of philosophical approaches

- ❖ Exceptions can't happen!?
- ❖ Laissez-faire
- ❖ Exceptions are normal?
- Build exception code in everywhere
- Pass and check status every time
- ◆ Return value or global variable?

3

Exception Handling

A variety of philosophical approaches

- ❖ Exceptions can't happen!?
- ❖ Laissez-faire
- ❖ Exceptions are normal?
- ❖ Exceptions are exceptional?
- Optimize normal; cope with errors!


4

Exception Handling

A variety of philosophical approaches

- ❖ Exceptions can't happen!?
- ❖ Laissez-faire
- ❖ Exceptions are normal?
- ❖ Exceptions are exceptional?

5

Bailing Out 

- ❖ If exceptions are exceptional
- Don't occur in "normal" operation
- No recovery at point of exception
- ❖ Stop what we're doing
- Clean up (destructors) & give up
- ❖ Ask for higher-level help

6

CS481 - Object-Oriented Programming

C++ Exceptions

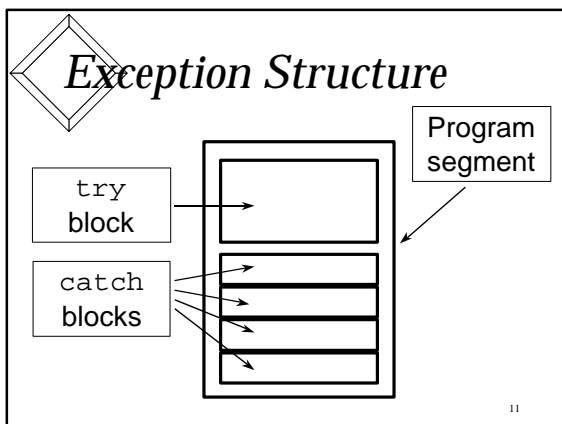
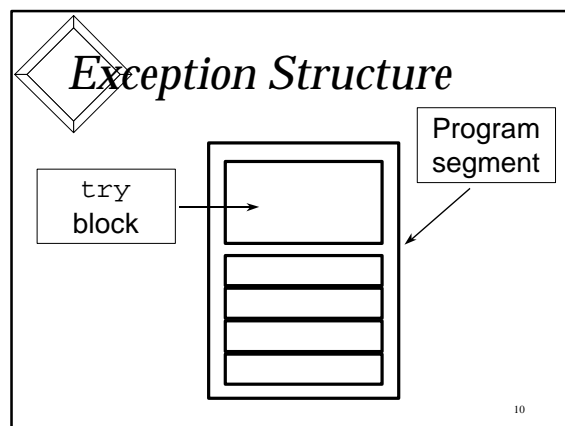
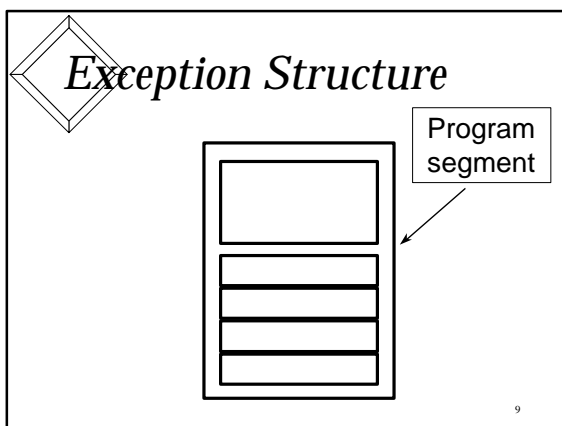
- ❖ At point of exception
 - Create object representing exception
 - ◆ Often of an exception class
- ❖ Throw exception
 - Leave current context (local destructors)
 - Passing exception object to ... ???

7

Exception Handler

- ❖ Defined area of responsibility
 - Section of code
 - ◆ Including any called routines
 - Unless lower-level handler intervenes
- ❖ Catches thrown exceptions
 - By type of exception object

8



Exception Example

```

try {
    cout << "Ready ..." << endl;
    throw "Test"; ...
}
catch (char* e) {
    cout<<"Caught { "<<e<< " }"<<endl;
}...
    
```

Ready ...
 Caught {Test}

12

CS481 - Object-Oriented Programming

Catch Options

Catch blocks are tried in order until a type match (including derived) is found.

```
try { /* code */ }
catch (char* e)
{cout<<"Caught {"<<e<< "}"<<endl;}
catch (int i)
{cout<<"Caught int "<< i <<endl;}
//Default: any type, last catch!!
catch (...)
{cout<<"Caught ??? " <<endl;}
13
```

Catch Options

Catch blocks are tried in order until a type match (including derived) is found.

```
try { /* code */ }
catch (char* e)
{cout<<"Caught {"<<e<< "}"<<endl;}
catch (int i)
{cout<<"Caught int "<< i <<endl;}
//Default: any type, last catch!!
catch (...)
{cout<<"Caught ??? " <<endl;}
14
```

Catch Options

Catch blocks are tried in order until a type match (including derived) is found.

```
try { /* code */ }
catch (char* e)
{cout<<"Caught {"<<e<< "}"<<endl;}
catch (int i)
{cout<<"Caught int "<< i <<endl;}
//Default: any type, last catch!!
catch (...)
{cout<<"Caught ??? " <<endl;}
15
```

Catch Options

Catch blocks are tried in order until a type match (including derived) is found.

```
try { /* code */ }
catch (char* e)
{cout<<"Caught {"<<e<< "}"<<endl;}
catch (int i)
{cout<<"Caught int "<< i <<endl;}
//Default: (any type, last catch!!)
catch (...)
{cout<<"Caught ??? " <<endl;}
16
```

Uncaught Exceptions

- ❖ If no matching catch block
- ❖ Or, if exception re-thrown
 - throw with no argument (in catch block)
- ❖ Exception passed to next level
 - Enclosing try block
 - If none, terminate() aborts program

Destructors

- ❖ Exception exits from "middle"
 - Some block objects have been created
 - Some have not (yet) been created
- ❖ Destructors are called
 - All created objects (constructor completed)
 - What about pointers to heap objects?

18

CS481 - Object-Oriented Programming

Destructor Exceptions

- ❖ **throw from destructor?**
 - If another exception in progress, then `terminate()` is called (bad idea!)
- ❖ **Allow no exceptions to escape**
 - Use `try/catch` block in destructor
 - Handle all exceptions locally

19

Constructors??

- ❖ **Constructor exceptions**
 - Constructor not completed
 - So destructor not called
- ❖ **What to do?**
 - Avoid if possible
 - Otherwise, do needed cleanup

20

Exception Specs

Controversial feature?

- ❖ **Part of function declaration**

```
- void f() throw (domain_error);
- void g() throw(); // none
- void h(); // any
```
- ❖ **Specifies possible exceptions**
 - Others, if any, are "unexpected"

21

Unexpected Exceptions

- ❖ **Not propagated from function**
 - Handled in function (no re-throw)
 - Or, guaranteed (?) not to occur at all
- ❖ **But, what if ... ?**
 - Function `unexpected()` is invoked
 - Default is to call `terminate()`

22

Termination

`terminate()`

- ❖ **Handler cannot be found**
- ❖ **Execution stack corrupted**
- ❖ **Destructor throws exception**
 - During object cleanup (1st exception)
 - Can't handle two at once - give up
 - Difficult to debug

23

Program Hooks

- ❖ **Function `set_terminate()`**
 - Caller provides new routine
 - Must still exit the program!! (no return)
- ❖ **Function `set_unexpected()`**
 - Re-throw, or throw another exception
 - May (need not) exit program

24

CS481 - Object-Oriented Programming

A Distinction

Caution

- ❖ Mechanism
 - Provides a way to do something
 - ◆ E.g., exceptions in C++ language
- ❖ Policy
 - Specifies how things should be done
 - ◆ E.g., exceptions in standard library

25

Standard Exceptions

- ❖ Defined by draft standard
 - But standard keeps changing!?
- ❖ Class hierarchy
 - Base class: `exception`
 - ◆ Used to be `xmsg`
 - Use existing or derive a class

26

Standard Exceptions

```

exception (xmsg)
  logic_error (xlogic)
    domain_error  invalid_argument
    length_error  out_of_range
    bad_cast      bad_typeid
  runtime_error (xruntime)
    range_error  overflow_error
    bad_alloc
ios::failure
    
```

27

Standard Exceptions

What about `raise()`?

```

exception (xmsg)
  logic_error (xlogic)
    domain_error  invalid_argument
    length_error  out_of_range
    bad_cast      bad_typeid
  runtime_error (xruntime)
    range_error  overflow_error
    bad_alloc
ios::failure
    
```

28

Exception Uses

- ❖ Meant for error conditions
 - Where local recovery not possible
 - Relatively infrequent events
 - ◆ Overhead?
- ❖ Synchronous events only
- ❖ Not to replace `switch/case`

29

