

16 Exception Handling
 throwing an exception
 the program/system signals that
 something unusual has happened
 handling an exception
 deal w/ unusual event in code

16.1 Basics

`res = a / b;`

can cause a divide-by-zero error

method to throw exception

```

try
{
    if (b == 0)
        throw a;
    res = a / b;
}
catch (int e)
{
    cout << "Trying to divide "
         << e << " by zero.\n";
}

```

explanation

' try block
code to attempt that may have errors in it

throw statement
integer argument for example
can throw any datatype
invokes exception handler

catch block

param that matches throw type
handles the exception

catch block is not a function
cannot call catch()

only invoked via throw
catch block is skipped if throw
is not called

Difference from if-else statement

value can be passed to catch

no value can be passed to else

example used program parameter

but can be any value

as soon as thrown seen, rest of

code is stopped

- in if stmt, whole block executes

Basic guidelines

try is followed by catch

catch applies only to previous try

after catch, code continues on

w/ stmts after catch

Exception Classes

instead of throwing int, throw object
carries more information
useful when throwing/catching
multiple exceptions

Multiple Exceptions

can be many throws per try
followed by 1+ catches
each catch for specific errors
each catch for only one type/class

Example:

```
class NegativeNumber  
{  
    private:  
        string msg;  
    public:  
        NegativeNumber() {}  
        NegativeNumber(string s) : msg(s) {}  
        string getMsg() { return msg; }  
};  
class DivideByZero {};
```

```
int main()  
{  
    int a, b;  
    double res;
```

```
    try  
    {
```

```
        cout << "Enter amount of candy: ";
```

```

cin >> a;
if (a < 0)
    throw NegativeNumber("candy");
cout << "Enter number of kids: ";
cin >> b;
if (b < 0)
    throw NegativeNumber("kids");
if (b == 0)
    throw DivideByZero();

```

```

cout << "Each kid gets " << a/b
    << " pieces of candy.\n";
}

```

```

catch (NegativeNumber e)
{

```

```

{

```

```

    cout << "Cannot have a negative "
        << "number of "
        << e.getMsg() << endl;
}

```

```

}

```

```

catch (DivideByZero) //name not req.
{

```

```

{

```

```

    cout << "More candy for you!\n";
}

```

```

}

```

```

return 0;
}

```

```

}

```

Special catch case

```

catch (...)
{

```

```

{

```

```

{
    // default catch
}

```

must come last in catch list

Throwing Exception in Function

```

returnType funcName (ParamList) //exception
    throw (Exception); //specification

int main ()
{
    try
    {
        funcName (Args);
    }
    catch (Exception)
    {
    }
}

returnType funcName (ParamList)
    throw (Exception)
{
    // other stmts
    throw Exception;
    // other stmts
    return // type,
}

```

Exception Specification

states what exception function
can throw

can throw

throw list:

```
throw (Exception1, Exception2, ...);
```

has all exceptions that can be caught outside function body

```
throw (); // no external catches
```

w/o exception specification

all exceptions can be thrown

if exception thrown that isn't in list

can be caught w/in body

of function

if not caught, program exits

16.2 Techniques for Exception Handling

When to Throw

when cannot be handled another way

when more info than local vars needs

to be passed to exception handler

handling depends on how & where the

function is used (eg location of

the function call)

Automatic Throws by System

throws done by C++ libraries

Out of Memory - bad-alloc

// Note: new standard. Not all

// compilers support

```
try
```

```
{
```

```
    Node * = new Nodej
```

```
}
```

```
catch (bad-alloc)
```

```
catch (bad_alloc)
```

```
{
```

```
    cout << "Out of memory\n";
```

```
}
```

Any else — exception

```
try
```

```
{
```

```
    // library
```

```
}
```

```
catch (exception &e)
```

```
{
```

```
    cout << "Error: " << e.what() << endl;
```

```
}
```