Topic: Exception handling

- An alternative to local error handling
- An alternative to program shutdown on error
- Permits multi-level communication of error information
- Keywords: *try, throw, catch*
- Example: division by 0

Why exception handling?

- With C++ exceptions, error information can travel directly, across function boundaries, to where it is handled
C++ try, throw, catch

- **try**: marks program block where exception might occur
- **throw**: on error, passes control directly to **catch**, throws a data item
- **catch**: marks block where a certain kind of exception is handled; catches an item of a particular type
- Program may define an exception class for objects that are to be thrown and caught

Throwing and catching exceptions

```cpp
class math_errors
{
public:
    char message[80];
    math_errors(char *msg) { strcpy(message,msg); };
};

void main()
{
    cout << "Enter two integers: ";
    int a,b;
    cin >> a >> b;
    try { cout << a << " / " << b << " = " << quotient(a,b); }
    catch (math_errors error)
    { cout << error.message << " not permitted.\n"; }
}

float quotient(int a,int b)
{
    if (b == 0) throw math_errors("Division by 0");
    return (float)a / b;
}
```

Sample I/O:
```
Enter two integers: 2 0
a / b = Division by 0 not permitted
```
Exceptions: guidelines

- Throw object, catch by class
- Uncaught exceptions terminate the program
- Exceptions may be re-thrown
- Relinquish all resources through destructors; *throw* causes destructor call
- If you can resolve a problem in current scope, do so rather than throw exception