

#### EECS402 Lecture 18

Andrew M. Morgan

Savitch, Ch. 18
Exceptions
Exception Handling



#### **Current Ways to Handle Errors**

로로C8 402

- Return a value that signifies an error
  - This value must be checked back in the calling function
  - Every time the function is called, you must check for an error being returned
  - Not only you but other programmers using your function must remember to do the appropriate checks
- Use assert() Not expecting u to use this
  - Assertions are generally used when a condition simply "will never happen"
  - You generally write your program in a way that won't even handle these situations because you expect them never to happen
  - Instead of making a (usually bad) assumption, use assert()
  - When the condition of an assertion is false, the program aborts immediately, telling you what condition was false, and a line number

EECS 402

Andrew M Morgan

M



#### **Exceptions**

- C++ allows you to use a mechanism called exceptions
- Exceptions allow a more attractive method to handling errors
- Exceptions should be used to handle "exceptional circumstances" - and nothing else
- While all errors could be handled with exceptions, you should not stop using other techniques
- Using exceptions allows you to do all your error checking in appropriate places, improving the readability of your proram
- Exceptions allow you to handle an error in a different area of your program from where the error occurred
- There are three keywords associated with C++ exceptions
  - try, throw, and catch

EECS 703

Andrew M Morgan





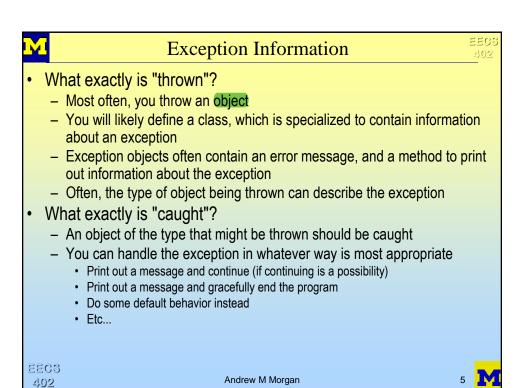
#### **Exception Keywords**

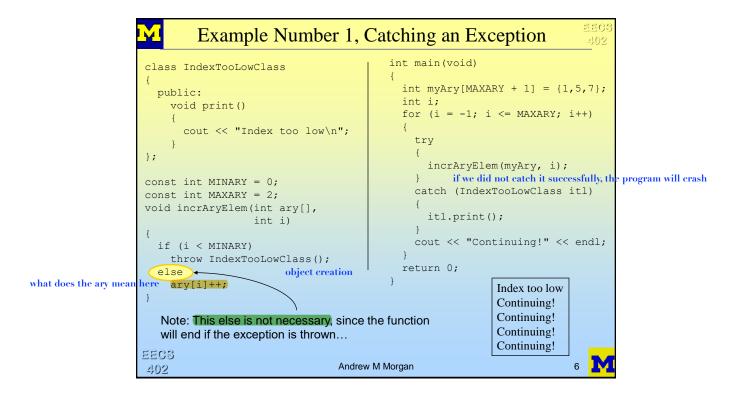
- When a function comes across an exceptional situation, it should throw an exception
  - An exception that was thrown should then be handled in a specified way, so that the program can recover, or exit gracefully
- When calling a function that throws an exception, the function may complete successfully, or it may fail
  - Therefore, when you call the function, you are going to try it and see what happens.
  - We say the function call is placed in a **try** block
- After a try block, an exception might have been thrown
  - In order to correctly handle that exception, we must make an attempt to catch the exception that was thrown
  - A catch block should come after a try block to handle the exception

EECS 402

Andrew M Morgan







# M

#### **Catching Multiple Types Of Exceptions**

**三三**の2

- Some of the power of exceptions results from the fact that any number of user-defined exception types can be caught and handled appropriately
- There are two ways to handle multiple types
- If the exception types are handled differently, multiple catch blocks are included after the single try block
- If all exception types are handled the same way, there is no need to handle each individual type
  - Instead, "catch (...)" is used to indicate any type of exception thrown should be caught and handled within that catch block

402 402

Andrew M Morgan

M

```
Handling Multiple Types, Example 1
                                       int main(void)
class IndexTooLowClass
                                         int myAry[MAXARY + 1] = \{1, 5, 7\};
  public:
    void print()
    {cout << "Index too low\n";}
                                         for (i = -1; i \le MAXARY + 1; i++)
class IndexTooHighClass
                                           try
  public:
                                             incrAryElem(myAry, i);
    void print()
    {cout << "Index too high\n";}
                                           catch (IndexTooLowClass itl)
                                Index too low
void incrAryElem(int ary[],
                                Continuina!
                                             itl.print();
                 int i)
                                Continuing!
                                Continuing!
                                           catch (IndexTooHighClass ith)
 if (i < MINARY)
                                Continuing!
                               Index too high
   throw IndexTooLowClass();
                                             ith.print();
 if (i > MAXARY)
   throw IndexTooHighClass();
                                           cout << "Continuing!" << endl;</pre>
  ary[i]++;
                                         return 0;
EECS
                                Andrew M Morgan
402
```

```
Handling Events With The Catch-All
class IndexTooLowClass
                                        int main(void)
 public:
                                          int myAry[MAXARY + 1] = \{1,5,7\};
    void print()
                                          int i;
    {cout << "Index too low\n";}
                                          for (i = -1; i \le MAXARY + 1; i++)
class IndexTooHighClass
                                            try
  public:
                                             incrAryElem(myAry, i);
   void print()
    {cout << "Index too high\n";}
};
                                            catch (...)
void incrAryElem(int ary[],
                                              cout << "Caught exception" <<</pre>
                int i)
                                                      endl;
  if (i < MINARY)
   throw IndexTooLowClass();
                                            cout << "Continuing!" << endl;</pre>
  if (i > MAXARY)
                                                           Caught exception
   throw IndexTooHighClass();
                                         return 0;
                                                           Continuing!
                                                           Continuing!
 ary[i]++;
                                                           Continuing!
                                                           Continuing!
                                                           Caught exception
                                                           Continuing!
EECS
                                 Andrew M Morgan
703
```

# M

#### Combining Exceptions Into One Class

로로C8 402

- Since user-defined exception types are classes, they may have data and/or function members
- Allowing exception types to have descriptive attributes means that the exceptions shown previously can be combined into a single class
- Common exception class attributes include:
  - An identifier for the specific type of exception an object represents
  - A string containing the name of the function which threw the exception
  - Other information describing a specific exception object thrown

402

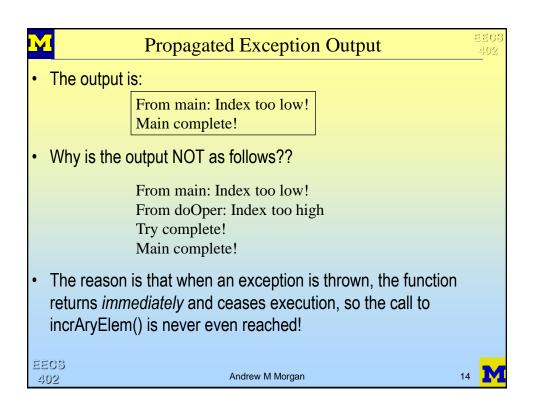
Andrew M Morgan

M

```
Using A Single Index Exception Class, Example
const int TOO LOW TYPE = 1;
                                            void decrAryElem(int ary[], int ind)
const int TOO HIGH TYPE = 2;
class IndexExcepClass
                                              if (ind < MINARY)
                                               throw IndexExcepClass(TOO LOW TYPE);
                                              if (ind > MAXARY)
    IndexExcepClass(int inType)
                                                throw IndexExcepClass (TOO_HIGH_TYPE);
                                             ary[ind]--;
       excepType = inType;
    void print()
                                            From main:
                                              for (i = -1; i \le MAXARY + 1; i++)
      if (excepType == TOO LOW TYPE)
       cout << "Index too low!";</pre>
                                                  decrAryElem(myAry, i);
      else if (excepType == TOO HIGH TYPE)
                                                catch (IndexExcepClass iex)
        cout << "Index too high!";
                                                 iex.print();
                            Index too low!
                                                 cout << endl;
                            Continuing!
  private:
   IndexExcepClass()
                                               cout << "Continuing!" << endl;</pre>
                            Continuing!
    { ; }
                            Continuing!
                            Continuing!
    int excepType;
};
                            Index too high!
                            Continuing!
EECS
                                    Andrew M Morgan
 703
```

### More About Exceptions Exceptions will propagate upwards until a handler is found When a function throws an exceptions It returns immediately to the calling program and looks for an appropriate catch block If no catch block is found, then THAT function returns immediately, looking for an appropriate catch block If no catch block is found ... ... and so on until: a) An appropriate catch block is found · b) main() is reached and no catch block is found - This results in an unhandled exception error and the program aborts immediately This would allow you to do all your error checking in main(), with all your code in one big try block If that is appropriate for your program EECS Andrew M Morgan 402

```
Example 4, Exception Propagation
  void doOper(int ary[])
                                       int main(void)
                                         int myAry[MAXARY + 1] = \{1, 5, 7\};
    try
                                         int i;
      //recall decrAryElem may
      //throw an IndexExcepClass
                                         try
      decrAryElem(ary, -1);
                                           doOper(myAry);
                                           cout << "Try complete!" << endl;</pre>
      //recall incrAryElem may
      //throw IndexTooLowClass
      //or IndexTooHighClass
                                         catch (IndexExcepClass iex)
      incrAryElem(ary, 7);
                                           cout << "From main: ";</pre>
    catch (IndexTooHighClass ith)
                                           iex.print();
      cout << "From doOper: ";</pre>
                                        cout << "Main complete!" << endl;</pre>
      ith.print();
                                         return 0;
                  What is the output of the above program??
EECS
                                Andrew M Morgan
703
```



```
How Exceptions Help success = myData.loadDataFromFile(inFname);
                                         cout << "Unable to load data from file" << endl;</pre>

    Exceptions can improve

 readability and
                                          success = myData.convertDataFormat();
 understandability of your
                                          if (!success)
 program
                                           cout << "Error converting data" << endl;</pre>
                                           cout << " --Data: " << myData << endl;
  -removes the clutter of error
    checking in the flow your code
                                            success = mergeWithDatabase(myData, database);
                                            if (!success)

    Consider the code shown

                                             cout << "Error merging data into database" << endl;</pre>
 here, which does not utilize
 exceptions
                                              success = database.doStatisticalAnalysis(stats);
                                              if (!success)
  -Compare to code to do the same
                                               cout << "Statistical analysis failed!" << endl;</pre>
    thing, but using exceptions, on
                                             else
    the next slide
                                               success = stats.outputToFile(outFname);
                                               if (!success)
                                                 cout << "Could not write output!" << endl;</pre>
EECS
 703
```

## How Exceptions Help, p2 The code using exceptions isn't cluttered up with error handling The actual flow of useful statements is now easy to see "stats" can now be a return value, rather than a pass-by-reference parameter since we no longer need to return success try myData.loadDataFromFile(inFname); myData.convertDataFormat(); mergeWithDatabase(myData, database); stats = database.doStatisticalAnalysis(); stats.outputToFile(outFname); catch (SevereException ge) ge.print(); EECS Andrew M Morgan 402