C++ Exceptions
C++ Exceptions

- An exception is a mechanism for error handling.
- When an error occurs during execution of a function, the function can throw an exception.
  - transfers the control to the exception handler.
- The function can then handle the error
  - catching the exception and
  - executing appropriate code.
C++ Exceptions: try-catch blocks

- **try block**
  - contain the statement(s) that
    - might cause an exception.
    - under exception inspection;

- **one or more catch blocks**
  - each indicates a type of exception you want to handle
  - the format is similar to a regular function that always has at least one parameter.
  - the type of the argument passed by the throw expression is checked against the type of this parameter
  - only in the case they match, the exception is caught.

```cpp
try
{  //The throw expression throws (raises) an exception.
    statement(s);
}

catch (type arg) { ... }
```

- The operand of throw is syntactically similar to the operand of a return statement.
// exceptions_trycatchandthrowstatements.cpp
// compile with: /EHsc
#include <iostream>
using namespace std;
int main()
{
    char *buf;
    try {
        buf = new char[512];
        if( buf == 0 ) throw "Memory allocation failure!";
    }
    catch( char * str )
    {
        cout << "Exception raised: " << str << '\n';
    }
}
// exceptions
#include <iostream>
using namespace std;
int main ()
{
   try
   {
      throw 20;
   }
   catch (int e)
   {
      cout << "An exception occurred. Exception Nr. " << e << endl;
   }
   return 0;
}
try
{ statement(s); // The throw expression throws (raises) an exception. }
catch (int param)
{ cout << "int exception"; }
catch (char param)
{ cout << "char exception"; }
catch (...) // this clause handles any type of exception
{ cout << "default exception"; }

Notes:
- Including C exceptions and system- or application-generated exceptions such as memory protection, divide by zero, and floating-point violations.
- (...) clause must be the last handler for its try block.
Catching Exceptions

- When a statement in a try block causes an exception, the control immediately passes to the statements in the corresponding catch block.
- Compiler determines which catch block to use by iterating through the catch clauses in the order they appear and select the first one matched.
- After an exception has been handled the program execution resumes after the try-catch block, not after the throw statement!
- If an exception has no applicable catch block, abnormal program termination usually occurs.
  - Destructors of local objects are called and resources to the block are released.
C++ standard exception class – “exception”

- A base class specifically designed to declare objects to be thrown as exceptions.
- It is defined in the `<exception>` header file under the namespace std.
- It has
  - default and copy constructors,
  - operators and
  - destructors,
  - a virtual member function called `what`
    - that returns a null-terminated character sequence (char *) - a message describing the exception
    - allow users to write their own exception handling.
Throwing exceptions

- When detect an error within a function, one can throw an exception
  
  throw exceptionClass (stringArgument)

- `exceptionClass` – the type of exception to throw.
- `stringArgument` – an argument to the `exceptionClass` constructor; usually providing a more detailed description of what may have caused the exception.
Example “throw”

```c
void myMethod (int x) throw (BadArgException, MyException)
    //limit the exception type it might directly or indirectly throw
{
    if (x==MAX)
        throw BadArgException("BadArgException: reason");
    // some other code that you want to do here
    ...
    throw MyException("MyException: reason");
} // end myMethod
```

- Including a throw statement in the function spec. ensures that the function can throw only those exceptions.
- An attempt to throw any other exception will result in a runtime error.
Example exception handling in the ADT List

- Exceptions
  - An *out-of-bounds* list index
    - Including attempts to delete or retrieve from an empty list
  - An attempt to insert into a *full* list
Define the exception “ListIndexOutOfRangeException”

```cpp
#include <stdexcept>
#include <string>
using namespace std;
class ListIndexOutOfRangeException : public out_of_range
{
    //more general exception class from the std lib

    public:
        ListIndexOutOfRangeException(const string & message="")
            : out_of_range(message.c_str())
        {}
}; // end ListIndexOutOfRangeException
```
Define the exception “ListException”

```cpp
#include <stdexcept>
#include <string>
using namespace std;
class ListException : public exception
{
public:
    ListException(const string &message="") : out_of_range(message.c_str())
    {}
}; // end ListException
```
Array-based ADT List: List.h

// **************************************************
// Header file List.h for the ADT list.
// Array-based implementation.
// **************************************************
#include “ListException.h”
#include “ListIndexOutOfRangeException.h”
const int MAX_LIST = maximum-size-of-list;
typedef desired-type-of-list-item listItemType;

class List
{
public:
    List (); // default constructor
    // destructor is supplied by compiler
Array-based ADT List: List.h (cont’d)

// list operations:
- bool isEmpty() const;
  // Exception: None
- int getLength() const;
  // Exception: None
Array-based ADT List: List.h (cont’d)

- void insert (int index, ListItemType newItem)
  throw (ListIndexOutOfRangeException, ListException);

  // before: insert (in index:integer, in newItem:ListItemType, out
  // success: boolean)

  // Exception: Throws ListIndexOutOfRangeException if
  // index < 1 or index > getLength()+1.
  // Exception: Throws ListException if newItem cannot be placed
  // in the list because the array is full
Array-based ADT List: List.h (cont’d)

- **remove** (int index)
  
  throw (ListIndexOutOfRangeException);

  // Exception: Throws ListIndexOutOfRangeException if
  // index < 1 or index > getLength().

- **void retrieve**(int index, listItemType & DataItem) const;
  
  throw (ListIndexOutOfRangeException);

  // Exception: Throws ListIndexOutOfRangeException if
  // index < 1 or index > getLength().
Array-based ADT List: List.h (cont’d)

private:

    listItemType items [MAX_LIST];  // array of list items

    int size;  // number of items in list

    int translate (int index) const;

};  // End of header file.
void List::insert(int index, listItemTypeNewItem)  
{
   if (size >= MAX_LIST) 
      throw ListException("ListException: List full on insert");
   if (index >=1 && index <= size +1) 
   {
      for (int pos = size; pos >= index; --pos) 
         items[translate(pos+1)] = items[translate(pos)] ;
      items[translate(index)] = newItem;
      ++Size;
   } else throw 
      ListIndexOutOfRangeException("ListIndexOutOfRangeExcep
tion: Bad index on insert");
} // end insert
The end. 😊