Exception Handling

CPE 212 -- Lecture 05

Robustness

- The ability of a program to recover following an error

Types of Errors

- A user may accidentally or deliberately enter incorrect inputs
- Hardware devices such as disk drives and random access memory have size limits
- Hardware devices may fail or become inaccessible
- Software components may contain defects

Options for Handling Errors

- Assume errors will not occur
- Print a descriptive error message
- Return an unusual value to indicate an error has occurred
- Alter a status variable's value
- Use assertions to block further execution
- Add error handlers (not in CPE 212)
- Use exception handlers

Exception Handling in C++

- ***Exception***
  - An unusual event, detectable by software or hardware, that requires special processing
- ***Exception Handler***
  - A section of program code that is executed when a particular exception occurs

try-throw-catch - 1

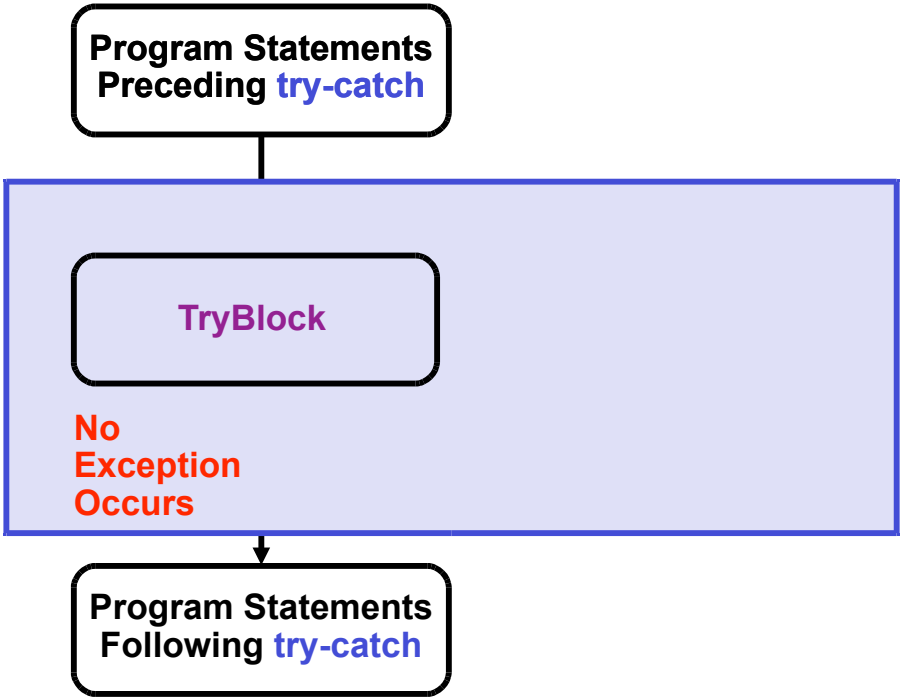
- **try** Clause
  - Contains statements that may cause an exception
- **throw**
  - Signals that an exception has occurred
  - May throw a value of any built-in or user-defined data type
- **catch** Clause
  - Contains statements that comprise an exception handler
  - Multiple catch clauses may be used to trap different errors -- processed in order listed
  - Ellipsis parameter used for catch-all exception handler

try-throw-catch - 2

```
try
{
    // Statements that may cause exception
}
catch ( datatype  optionalparametername )
{
    // Exception handler statements
}
```

try-throw-catch - 3

```
try
   TryBlock
catch  (datatype)
   CatchBlock
```



Assuming CatchBlock does not cause any further exception or transfer flow of control elsewhere

try-throw-catch - 4

```
#include <iostream>
using namespace std;

int main()
{
  // Code below works but not a very good
  // description of the exception that occurred
  try
  {
    throw  7;        // Raise/throw the exception
  }
  catch ( int  n )  // Handle a thrown integer
  {
    cout << "Error: " << n << " encountered\n";
  }

  return 0;
} // End main()
```

try-throw-catch - 5

```
#include <iostream>
using namespace std;

int main()
{
  // Note that code for a string value is skipped
  try
  {
    throw  7;        // Raise/throw the exception
  }
  catch ( string s )  // Handle a thrown string
  {
    cout << "Error: " << s << " encountered\n";
  }
  catch ( int  n )  // Handle a thrown integer
  {
    cout << "Error: " << n << " encountered\n";
  }

  return 0;
} // End main()
```

try-throw-catch - 6

```
#include <iostream>
using namespace std;

int main()
{
  // Code below does not perform as a one might expect since
  // literal "dog" is not a string data type so last catch block executed
  try
  {
    throw  "dog";      // Raise/throw the exception
  }
  catch ( string s )  // Handle a thrown string
  {
    cout << "Error: " << s << " encountered\n";
  }
  catch ( int  n )    // Handle a thrown integer
  {
    cout << "Error: " << n << " encountered\n";
  }
  catch ( ... )       // Handle any type of value
  {
    cout << "Unknown error encountered\n";
  }

  return 0;
} // End main()
```

try-throw-catch - 7

```
#include <iostream>
using namespace std;

enum Animal {DOG,CAT,BIRD};

class UnknownAnimalEncountered      // Empty error class
{
  // NO CODE HERE!!   Creates a named category of errors
};

void PrintAnimal(Animal someCreature);

int main()
{
  // Code below works but not a very good
  // description of the exception that occurred
  Animal  critter = DOG;

  while (critter <= BIRD)
  {
    PrintAnimal(critter);
    critter = Animal(critter + 1);  // Type cast integer back to enumerated type
  }
```

try-throw-catch - 8

```
  // main() - continued

  try
  {
    PrintAnimal(critter);
  }
  catch (UnknownAnimalEncountered someAnimal)
  {
    cout << "Error: unknown animal\n";
  }
  catch ( ... )
  {
    cout << "Unknown error encountered\n";
  }

  return 0;
} // End main()
```

try-throw-catch - 9

```
void PrintAnimal(Animal someCreature)
{
  switch (someCreature)
  {
    case DOG:     cout << "dog" << endl;
                  break;

    case CAT:     cout << "cat" << endl;
                  break;

    case BIRD:    cout << "bird" << endl;
                  break;

    default:      throw UnknownAnimalEncountered();
                   // Constructor creates object to throw
  }
} // End PrintAnimal()
```

try-throw-catch - 10

```
try
{
   if (num != 0)
        average = sum / num;
   else
        throw string("Divide by zero error");
}
catch (string errmsg)
{
   cout << errmsg << endl;
   return 1;
}

cout << "The average is " << average << endl;
return 0;
```

try-throw-catch - 11

```
#include <iostream>
#include <new>
using namespace std;

int main()
{
  double* ptr;
  try
  {
        ptr = new double[2000];                  // new may throw bad_alloc

        cout << "Array created" << endl;         // does not execute if
                                                 // exception thrown above
  }
  catch ( bad_alloc )   // a standard exception in  #include <new>
  {
        cout << "Error -- insufficient memory" << endl;
        return 1;
  }

  cout << "Rest of program here" << endl;
  return 0;
} // End main()
```