CS-1201 Object Oriented Programming

Exception Handling

Arbish Akram

Department of Computer Science
Government College University

Exceptions and Why Exception Handling

Exceptions are errors that occur at runtime, caused by various exceptional circumstances, such as:

- Running out of memory
- Failing to open a file
- Initializing an object to an impossible value
- Using an out-of-bounds index in an array

Why Exception Handling?

- **Separation of Error Handling Code from Normal Code:**
  - Exception handling allows errors to be dealt with separately from the main program logic.
  - This improves readability, maintainability, and makes debugging easier.

Exception Handling

- **try:** Represents a block of code that can throw an exception.
- **catch:** Represents a block of code that is executed when a particular exception is thrown.
- **throw:** Used to throw an exception. Also used to list the exceptions that a function throws, but doesn't handle itself.

Example I

```
1  int main()
2  {
3      int x = -1;
4      cout << "Before try \n";
5      try {
6          cout << "Inside try \n";
7          if (x < 0)
8          {
9              throw x;
10             cout << "After throw (Never executed) \n";
11         }
12     }
13     catch (int x) {
14         cout << "Exception Caught \n";
15     }
16     cout << "After catch (Will be executed) \n";
17     return 0;
18 }
```

Example II

```
1  int main()
2  {
3      int x = -1; float y = -1;
4      cout << "Before try \n";
5      try {
6          cout << "Inside try \n";
7          if (x < 0)
8          {
9              throw y;
10             cout << "After throw (Never executed) \n";
11         }
12     }
13     catch (int x) {
14         cout << "Exception Caught integer\n";
15     }
16     catch (float x) {
17         cout << "Exception Caught float \n";
18     }
19     cout << "After catch (Will be executed) \n";
20     return 0;
21 }
```

Example III

```
1  int getUserNumber() {
2      int userEntry;
3      cout << "Enter a positive value ";
4      const string MSG = "Invalid entry";
5      cin >> userEntry;
6      if (userEntry < 0) {
7          throw(MSG);
8      }
9      return userEntry;
10 }
11 int main()
12 {
13     int returnedValue;
14     try {
15         returnedValue = getUserNumber();
16         cout << "Data entry value is " << returnedValue << endl;
17     }
18     catch (const string message) {
19         cout << "There was an error!" << endl << message << endl;
20     }
21     cout << "End of program" << endl;
22     return 0;
23 }
```

Example IV

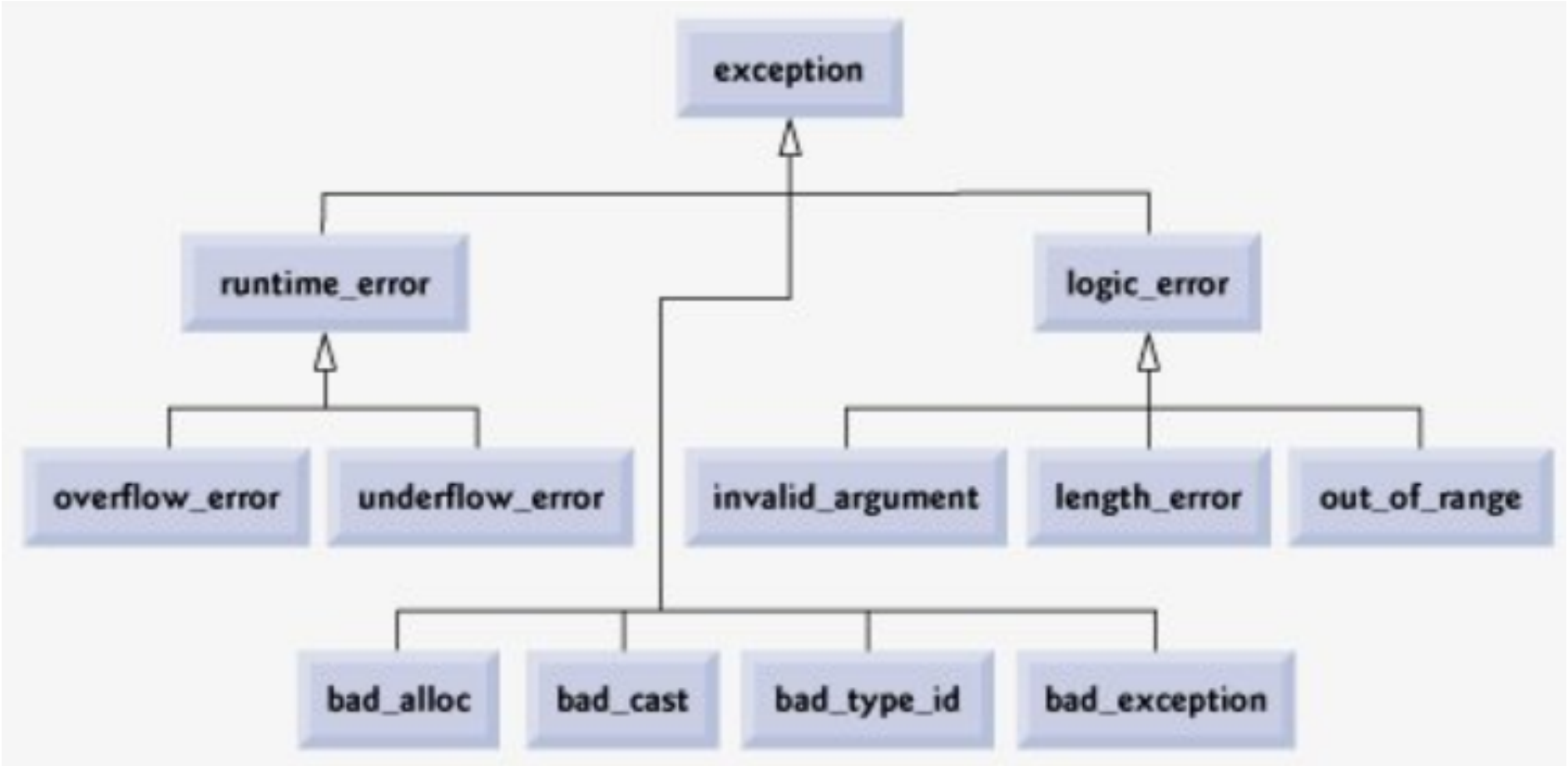
```
1  int main() {
2      double numerator, denominator, divide;
3      cout << "Enter numerator: ";
4      cin >> numerator;
5      cout << "Enter denominator: ";
6      cin >> denominator;
7      try {
8          // throw an exception if denominator is 0
9          if (denominator == 0)
10             throw 0;
11         // not executed if denominator is 0
12         divide = numerator / denominator;
13         cout << numerator << " / " << denominator << " = " << divide << endl;
14     }
15     catch (int num_exception) {
16         cout << "Error: Cannot divide by " << num_exception << endl;
17     }
18     return 0;
19 }
```

Standard Exception

The C++ Standard Library provides a set of classes for reporting errors. These errors fall into two broad categories:

- **Logic Errors:** Errors caused by problems in the internal logic of the program. They are generally preventable.
- **Runtime Errors:** Errors that are generally not preventable or predictable. These are generated by circumstances outside the control of the program, such as peripheral hardware faults.

Programs that use the exception handling classes must include the following header file: `#include <stdexcept>`


exception
runtime_error
logic_error
overflow_error
underflow_error
invalid_argument
length_error
out_of_range
bad_alloc
bad_cast
bad_type_id
bad_exception