Exceptions

Rupesh Nasre.

OOAIA
January 2025

Why not C error handling?

```
int *p = (int *) malloc (20);
if (p == NULL) error("Not enough memory", __FILE__, __LINE__);

FILE *fp = fopen("data.txt", "r");
if (fp == NULL) error("Error reading data file", __FILE__, __LINE__);

assert(root != NULL);
fscanf(fp, "%s", p);

do {
        process(p, root);
        fscanf(fp, "%s", p);
} while (!feof(fp));

if (root->count < n) error("Some issue with logic", __FILE__, __LINE__);

free(p);
```

- Hinders code understanding.
- Error handling may create issues such as leaks.

What are Exceptions?

- Run-time anomalies
- The program can recover from these.
- When your software is running on client machines, it should not crash.
  - should at least make a *graceful exit*.
  - Think of cloud containers, apache web server, gmail, ola app, ...
- Advantages
  - allow separating core logic from error handling
  - separate error situations between callers and callees

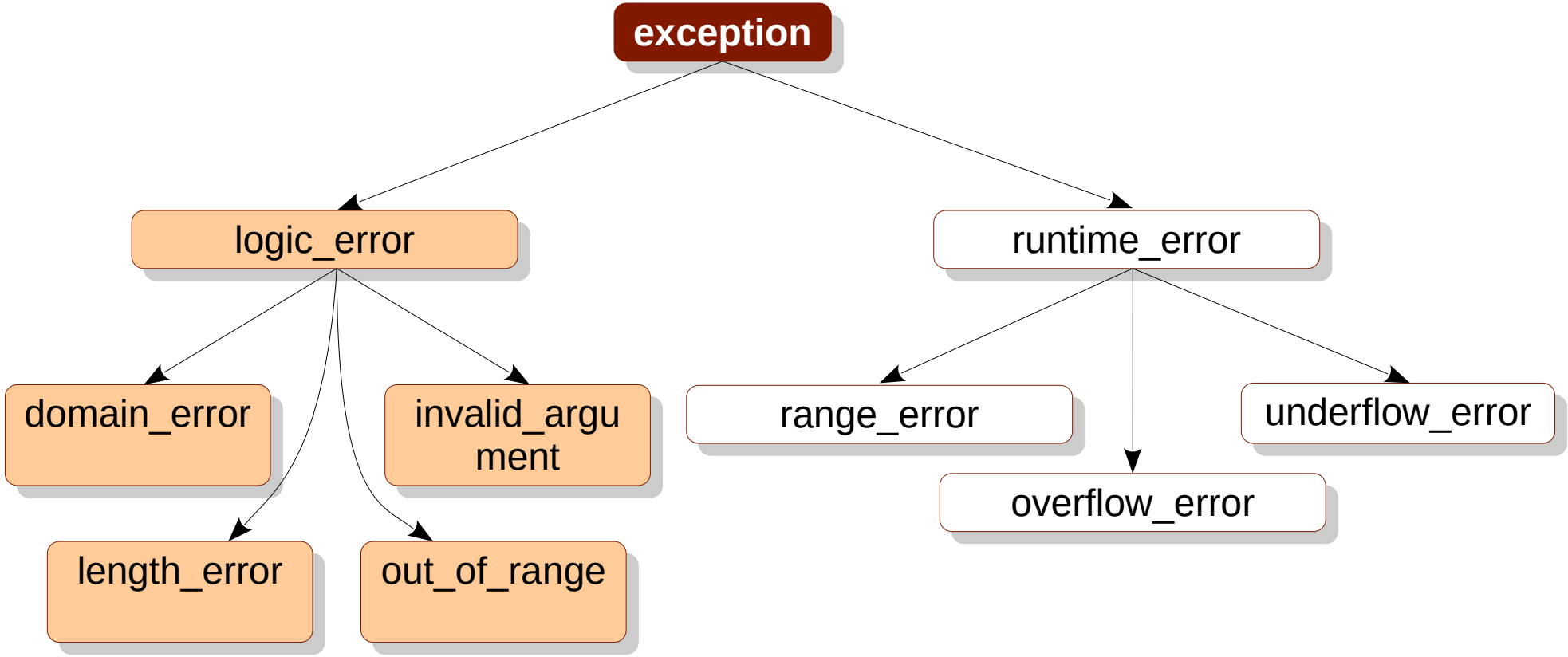
Using exceptions

```
try {
        int *p = (int *) malloc (20);

        FILE *fp = fopen(“data.txt”, “r”);

        fscanf(fp, “%s”, p);

        do {
                process(p, root);
                fscanf(fp, “%s”, p);
        } while (!feof(fp));

        free(p);

} catch (...) {
        // error handling.
}
```

Block of code where exception may occur.

What to do when exception occurs.

C++ Exceptions



try-catch

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

int main() {
    string s;

    try {
      //s.insert(0,"Hello");
      s.insert(1, "Hello");
    } catch (const std::exception& e) {
      cout << "The exception is caught." << endl;
    }

    return 0;
}
```

Regular processing in try

Exception handling in catch

try-catch within a Function

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

string s;
void fun() {
   s.insert(1,"Hello");
}
int main() {
    try {
      fun();
    } catch (const std::exception& e) {
      cout << "The exception is caught." << endl;
    }

    return 0;
}
```

Exception in a callee can be caught in a caller.

Stack unwinding, till the exception is caught or the program terminates.

Null Pointer Dereference

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

int main() {
	try {
		int *p = nullptr;
		*p = 100;
	} catch (const std::exception& e) {
		cout << "The exception is caught." << endl;
	}
	cout << "The program ends now.\n";
	return 0;
}
```

What is the output?

```
$ a.out
Segmentation fault (core dumped)
```

C++ catches only those exceptions that are **explicitly thrown**.
Catching exceptions such as null-pointer dereference,
divide by zero error, etc. need a managed runtime.

Throwing Exceptions

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

string s;
void fun() {
        try {
                s.insert(1,"Hello");
        } catch(std::exception &e) {
                cout << "Caught in fun.\n";
                 throw e;
        }
}
int main() {
      try {
            fun();
      } catch (const std::exception& e) {
            cout << "Caught in main: " << e.what() << endl;
      }
      cout << "The program ends now.\n";
      return 0;
}
```

Catching by reference allows no more copying code to be executed. This avoids further exceptions to be thrown.

Even if you use only "**throw**;" it would rethrow the caught exception.

What is the output?

```
string s;
void fun() {
        try {
            try {
                s.insert(1,"Hello");
            } catch (std::out_of_range) {
                cout << "caught in fun: out of range\n";
                throw;
            }
        } catch (std::exception &e) {
                cout << "Caught in fun.\n";
        }
        cout << "continue with fun\n";
}
int main() {
    try {
        fun();
    } catch (std::exception& e) {
        cout << "The exception is caught." << endl;
        exit(1);
    }
    cout << "The program ends now.\n";
}
```

Your Exceptions

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

class myexception {
public:
      myexception(const string &errmsg) {
            this->errmsg = errmsg;
      }
      virtual string what() {
            return errmsg;
      }
      //virtual ~myexception() throw() { }
private:
       string errmsg;
};
```

```
string s;
void fun() {
        try {
           s.insert(1,"Hello");
        } catch(std::exception &e) {
           cout << "Caught in fun: "
                   << e.what() << endl;
           throw myexception("CS2810 exception.");
        }
}
int main() {
       try {
           fun();
       } catch (myexception &e) {
           cout << "The exception is caught in main: "
                   << e.what() << endl;
       }
       cout << "The program ends now.\n";
       return 0;
}
```

Exception Chain

```
// some code.
int main() {
	try {
		fun();
	} catch (myexception &e) {
		cout << "My exception is caught: " << e.what() << endl;
	} catch (const std::exception &e) {
		cout << "Standard exception is caught: " << e.what() << endl;
	} catch (...) {
		cout << "An unknown exception was caught." << endl;
	}
	cout << "The program ends now.\n";
	return 0;
}
```

- A base-class reference can catch a derived class exception.
- The most specialized class should be caught first, with base being later.

You can throw any type!

```
int main() {
        try {
                int x = 2, y = 3;
                throw x+y;
        } catch (int z) {
                cout << z << endl;
        }
}
```

You can throw any type!

```
bool search(array<int, 10> a, int key) {
        for (int ii = 0; ii < a.size(); ++ii)
                if (a[ii] == key)
                        throw true;
        return false;
}

int main() {
        array<int, 10> a;
        populate(a);
        int key = 3;

        try {
                bool found = search(a, key);
                cout << "key " << key << " is not present in the array.\n";
        } catch (...) {
                cout << "key " << key << " is present in the array.\n";
        }
}
```

Exceptions for Control-Flow

```
void strUpper(char *s) {
        try {
                if (*s == '\0') throw s;
                strUpper(s + 1);
        } catch(...) {
                *s = toupper(*s);
                throw;
        }
}
int main() {
        char s[ ] = "abcdefgh";
        try {
                strUpper(s);
        } catch (...) {
                cout << s << endl;
        }
        return 0;
}
```

- Try linked-list reversal with try-catch.
- This is only for your understanding. Please do not try such methods in production code (and risk your job ... and may be mine too).