Exceptions

Rupesh Nasre.

OOAIA
January 2020

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.

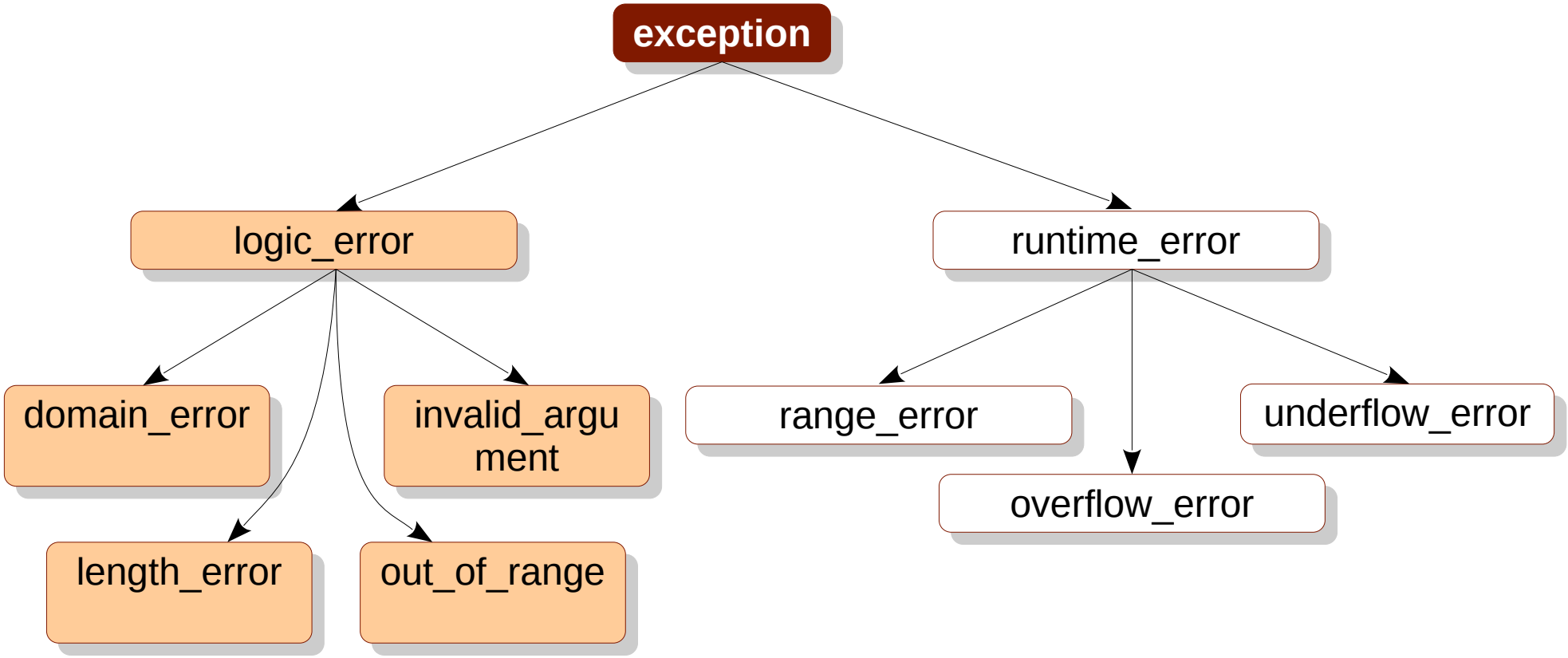
What are Exceptions?

- 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, ...
- We separate core logic from error handling.

Why not C error handling?

```
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.
}
```

C++ Exceptions



Multiple Template Arguments

```
template<class T1, class T2>
class Group {
public:
        Group() { std::cout << "class instantiated.\n"; }
        void add(std::pair<T1, T2> e);
        bool present(std::pair<T1, T2> e);
private:
        std::vector<std::pair<T1, T2> > elements;
};

template<class T1, class T2>
void Group<T1, T2>::add(std::pair<T1, T2> e) {
        elements.push_back(e);
}
template<class T1, class T2>
bool Group<T1, T2>::present(std::pair<T1, T2> e) {
        return (find(elements.begin(), elements.end(), e) != elements.end());
}
```