Object-Oriented Programming

Resource Management

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Automatic Variables

```
class Data {
public:
    Data();
    ~Data();
    void process();
};

Data::Data() {
    calls.push_back(__FUNCTION__);
}

Data::~Data() {
    calls.push_back(__FUNCTION__);
}

void Data::process() {}

class DataWrapper {
public:
    DataWrapper(Data& data);
    ~DataWrapper();
private:
    Data& data;
};

DataWrapper::DataWrapper(Data& data)
: data(data) {

    calls.push_back(__FUNCTION__);
}

DataWrapper::~DataWrapper() {
    data.process();

    calls.push_back(__FUNCTION__);
}
```

```
{
    Data data;
    DataWrapper wrapper(data);

    // ...
}

assert(calls[0] == "Data");
assert(calls[1] == "DataWrapper");
assert(calls[2] == "~DataWrapper");
assert(calls[3] == "~Data");
```

Automatic Variables

```
{
    Data data;
    DataWrapper wrapper(data);

    // ...
}

assert(calls[0] == "Data");
assert(calls[1] == "DataWrapper");
assert(calls[2] == "~DataWrapper");
assert(calls[3] == "~Data");
```

- The constructor is called when the declaration is reached

- The destructor is called right before the block exits

- Order of destructor calls is inverse of constructor order

Software View of Resources

- Memory
- File handles (e.g., file descriptors)
- Network connections
- Locks & semaphores
- Database connections

Resource must be...

- Properly *allocated*

- Properly *initialized* before use

- Properly *validated* before use

- Properly *deallocated* after use

Resource Problems

- Improper initialization
- Use when invalid
- Resource leak (never deallocated)
- Double free (deallocated more than once)

File Descriptor Resource Leak

```cpp
// write to the output file as many lines as we can
int counter = 0;
while (true) {

    // Open a file using low-level open()
    int fd = open(argv[1], O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fd == -1) {
        std::cerr << "Failed to open file." << std::endl;
        return 1;
    }

    // Write to the file
    const std::string_view text = "Writing this to a file.\n"sv;
    write(fd, text.data(), text.size());

    // Status
    std::cerr << ++counter << '\n';
}
```

- File descriptor is *allocated* each time through the loop
- But file descriptor is not *deallocated* each time through the loop
- Will hit a limit that depends on your O.S. environment
  - To view the limit: `ulimit -n`
  - Note: Some discrepancy on the default limit in Ubuntu (and in srcml/codespaces)
  - Even without a resource leak, you can open too many files simultaneously

Effect of Improperly Used Resources

- Exceed resource limits
- Program crash
- Invalid output
- Security vulnerabilities

C++ Pointers

- Pointers to memory

- Limits on the amount of memory available to a program/process

- Good pointer usage is not easy to do

C++ Pointer Problems

```
{
    // improper initialization
    int* f;

    // use when invalid
    *f = 4;

    f = new int(3);

    // memory leak
}
```

```
{
    int* f = new int[3];
    int* g = f;

    // wrong delete (need delete [])
    delete g;

    // double free
    delete f;
}
```

Avoiding C++ Pointer Problems

- Behavioral issue

- Requires complex tools to detect, with lots of *false positives*

- Must have a scenario for each usage pattern

- Good design can prevent the problem from occurring