SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220504<A|D>

Replace <A|D> with this section's letter

Exceptions and Assertions

—

CS 2124: Object Oriented Programming
Darryl Reeves, Ph.D.

Agenda

- Exceptions
- The `try-block`
- The `throw` expression
- Assertions
- Defining new exceptions
- Bounds checking

What do we do when something unexpected happens in our program?

Exceptions

Dealing with an exceptional situation

Bank account balance: $1,050

Request to withdrawal: $1,000,000

Possible outcomes:

- dispense full amount requested — *bad idea!*
- alert law enforcement — *potentially unnecessary escalation*
- inform funds not available — *most neutral response*

Exceptions allow detection of exceptional circumstances and determine how to respond

Handling an exceptional situation

```
int divide_int(int num, int denom) {
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);

    cout << "Integer division: " << num1 << " / ";
    cout << num2 << endl;
    cout << "Result: " << divide_int(num1, num2) << endl;
}
```

```
% g++ --std=c++11 divide_exc.cpp -o divide_exc.o
% ./divide_exc.o
Integer division: 8.9 / 3.3
Result: 2
```

TurningPoint

SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220504<A|D>

Replace <A|D> with this section's letter

Which exceptional situation can arise based on the definition of `divide_int()`?

```
int divide_int(int num, int denom) {
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);

    cout << "Integer division: " << num1 << " / ";
    cout << num2 << endl;
    cout << "Result: " << divide_int(num1, num2) << endl;
}
```

```
% g++ --std=c++11 divide_exc.cpp -o divide_exc.o
% ./divide_exc.o
Integer division: 8.9 / 3.3
Result: 2
```

Handling an exceptional situation

```
int divide_int(int num, int denom) {
    return num / denom;
}
```

exception generated here

```
int main() {
    double num1(8.9), num2(3.3);

    cout << "Integer division: " << num1 << " / ";
    cout << num2 << endl;
    cout << "Result: " << divide_int(num1, num2) << endl;

    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;

}
```

```
% g++ --std=c++11 divide_exc2.cpp -o divide_exc2.o
% ./divide_exc2.o
Integer division: 8.9 / 3.3
Result: 2
Integer division: 8.9 / 0
floating point exception  ./divide_exc2.o
```

we have a problem

statement execution interrupted

The try-block

The `try-block`

begins exception handling statement

```
try {

   // execute (try) instructions that may throw exception

} catch (exception_type& exception_name) {

   // execute instructions when exception_type thrown


}
```

statement specifying exception to handle

name used for referring to exception in body of `catch` clause

type of exception

The `try-block` (multiple exceptions)

```
try {
  // execute (try) instructions that may throw exception
} catch (exception_type1& exception_name1) {
  // execute instructions when exception_type1 thrown
} catch (exception_type2& exception_name2) {
  // execute instructions when exception_type2 thrown
}
```

Handling an exceptional situation

```
int divide_int(int num, int denom) {
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

Handling an exceptional situation

```
int divide_int(int num, int denom) {
    try {
        return num / denom;
    } catch (std::exception& ex) {
        cerr << "Attempt to divide by 0" << endl;
        return 0;
    }
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

```
% g++ --std=c++11 divide_exc3.cpp -o divide_exc3.o
% ./divide_exc3.o
Integer division: 8.9 / 3.3
Result: 2
Integer division: 8.9 / 0
floating point exception  ./divide_exc2.o
```

exception not handled

The `throw` expression

Signaling a problem has occurred

```
int divide_int(int num, int denom) {
    try {
        return num / denom;
    } catch (std::exception& ex) {
        cerr << "Attempt to divide by 0" << endl;
        return 0;
    }
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

explicitly generate an exception that can be handled

Signaling a problem has occurred

```
int divide_int(int num, int denom) {
    try {
        if (denom == 0) throw exception();
        return num / denom;
    } catch (std::exception& ex) {
        cerr << "Attempt to divide by 0" << endl;
        return 0;
    }
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

explicitly generate an exception that can be handled

```
% g++ --std=c++11 divide_exc3.cpp -o divide_exc3.o
% ./divide_exc4.o
Integer division: 8.9 / 3.3
Result: 2
Integer division: 8.9 / 0
Result: Attempt to divide by 0
0
```

Signaling a problem has occurred

```
int divide_int(int num, int denom) {
    try {
        if (denom == 0) throw exception();
        return num / denom;
    } catch (std::exception& ex) {
        cerr << "Attempt to divide by 0" << endl;
        return 0;
    }
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

need to return an int

dividing by 0 is undefined (not 0)

```
% g++ --std=c++11 divide_exc3.cpp -o divide_exc3.o
% ./divide_exc4.o
Integer division: 8.9 / 3.3
Result: 2
Integer division: 8.9 / 0
Result: Attempt to divide by 0
0
```

output not clear

Signaling a problem has occurred

```
int divide_int(int num, int denom) {
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

Signaling a problem has occurred

```
int divide_int(int num, int denom) {
    if (denom == 0) throw exception();
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: " << divide_int(num1, num3) << endl;
    } catch (exception& ex) {
        cerr << "Attempt to divide by 0" << endl;
    }
}
```

Signaling a problem has occurred

```
int divide_int(int num, int denom) {
    if (denom == 0) throw exception();
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: " << divide_int(num1, num3) << endl;
    } catch (exception& ex) {
        cerr << "Attempt to divide by 0" << endl;
    }
}
```

```
% g++ --std=c++11 divide_exc5.cpp -o divide_exc5.o
% ./divide_exc5.o
Integer division: 8.9 / 3.3
Result: 2
Integer division: 8.9 / 0
Result: Attempt to divide by 0
```

Assertions

The `assert` statement

```
int divide_int(int num, int denom) {
    if (denom == 0) throw exception();
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Enter a divisor: ";
    cin >> num3;

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

The `assert` statement

when `assert` condition `false`

1) message written to `stderr`
2) `abort` called (ends program)

```
int divide_int(int num, int denom) {
    assert (denom != 0);
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Enter a divisor: ";
    cin >> num3;

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

```
% g++ --std=c++11 divide_exc6.cpp -o divide_exc6.o
% ./divide_exc6.o
Integer division: 8.9 / 3.3
Result: 2
Enter a divisor:   0

Integer division: 8.9 / 0
Assertion failed: (denom != 0), function divide_int, file divide_exc6.cpp, line 6.
Result: zsh: abort      ./divide_exc6.o
```

The `assert` statement

```
#define NDEBUG
#include <cassert>
```

turns off all asserts

```
int divide_int(int num, int denom) {
    assert (denom != 0);
    return num / denom;
}

int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Enter a divisor: ";
    cin >> num3;

    cout << "Integer division: " << num1 << " / ";
    cout << num3 << endl;
    cout << "Result: " << divide_int(num1, num3) << endl;
}
```

```
% g++ --std=c++11 divide_exc6.cpp -o divide_exc6.o
% ./divide_exc6.o
Integer division: 8.9 / 3.3
Result: 2
Enter a divisor: 0

Integer division: 8.9 / 0
zsh: floating point exception  ./divide_exc6.o
```

Defining new exceptions

Custom exceptions

```
int divide_int(int num, int denom) {
    if (denom == 0) throw exception();
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;

    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (exception& ex) {
        cerr << "Attempt to divide by 0";
        cerr << endl;
    }
}
```

catches any exception that may occur in the try clause

Custom exceptions

inherits from exception

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {

    }
};
```

no exception can be thrown by function

```
int divide_int(int num, int denom) {
    if (denom == 0) throw exception();
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (exception& ex) {
        cerr << "Attempt to divide by 0";
        cerr << endl;
    }
}
```

Custom exceptions

inherits from exception

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {

    }
};
```

full function signature

```
int divide_int(int num, int denom) {
    if (denom == 0) throw exception();
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (exception& ex) {
        cerr << "Attempt to divide by 0";
        cerr << endl;
    }
}
```

Custom exceptions

inherits from exception

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        provides explanatory information
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw exception();
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (exception& ex) {
        cerr << "Attempt to divide by 0";
        cerr << endl;
    }
}
```

Custom exceptions

inherits from exception

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw exception();
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (exception& ex) {
        cerr << "Attempt to divide by 0";
        cerr << endl;
    }
}
```

Custom exceptions

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw ZeroDivException();
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (exception& ex) {
        cerr << "Attempt to divide by 0";
        cerr << endl;
    }
}
```

Custom exceptions

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw ZeroDivException();
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (ZeroDivException& ex) {
        cerr << "Attempt to divide by 0";
        cerr << endl;
    }
}
```

Custom exceptions

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw ZeroDivException();
    return num / denom;
}
```

```
% g++ --std=c++11 divide_exc7.cpp -o divide_exc7.o
% ./divide_exc7.o
Integer division: 8.9 / 3.3
Result: 2
Integer division: 8.9 / 0
Result: attempt to divide by 0
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (ZeroDivException& ex) {
        cerr << ex.what();
        cerr << endl;
    }
}
```

Custom exceptions

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw ZeroDivException();
    return num / denom;
}
```

can be marked as const to prevent modification of the exception object

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (ZeroDivException& ex) {
        cerr << ex.what();
        cerr << endl;
    }
}
```

Custom exceptions

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw ZeroDivException();
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (const ZeroDivException& ex) {
        cerr << ex.what();
        cerr << endl;
    }
}
```

Custom exceptions

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw ZeroDivException();
    return num / denom;
}
```

other types can follow

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (const ZeroDivException& ex) {
        cerr << ex.what();
        cerr << endl;
    }
}
```

Custom exceptions

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw "attempt to divide by 0";
    return num / denom;
}
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (~~const ZeroDivException&~~ ex) {
        ~~cerr << ex.what();~~
        ~~cerr << endl;~~
    }
}
```

Custom exceptions

```
struct ZeroDivException : public exception {

    const char* what() const noexcept {
        return "attempt to divide by 0";
    }
};
```

```
int divide_int(int num, int denom) {
    if (denom == 0) throw "attempt to divide by 0";
    return num / denom;
}
```

```
% g++ --std=c++11 divide_exc8.cpp -o divide_exc8.o
% ./divide_exc8.o
Integer division: 8.9 / 3.3
Result: 2
Integer division: 8.9 / 0
Result: attempt to divide by 0
```

```
int main() {
    double num1(8.9), num2(3.3);
    ...
    double num3(0);

    cout << "Integer division: ";
    cout << num1 << " / ";
    cout << num3 << endl;
    try {
        cout << "Result: ";
        cout << divide_int(num1, num3);
        cout << endl;
    } catch (string ex) {
        cerr << ex << endl;
    }
}
```

Bounds checking

Boundary violations

```
int main() {
    vector<int> vec;

    vec[17] = 42;
}
```

dynamic array pointer is initialized to nullptr

operation requires dereferencing nullptr

```
% g++ --std=c++11 bounds_check.cpp -o bounds_check.o
% ./bounds_check.o
segmentation fault  ./bounds_check.o
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    vec[17] = 42;
    cout << vec[17] << endl;

}
```

dynamic array pointer not nullptr

```
% g++ --std=c++11 bounds_check.cpp -o bounds_check.o
% ./bounds_check.o
42
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    vec.at(17) = 42;

}
```

alternative to vec[17] = 42;

throws std::out_of_range exception

```
% g++ --std=c++11 bounds_check.cpp -o bounds_check.o
% ./bounds_check.o
libc++abi: terminating with uncaught exception of type std::out_of_range: vector
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    vec.at(17) = 42;
}
```

prevent this exception from terminating the program

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    ___ {
        vec.at(17) = 42;
    } ___ {

    }

}
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    _1_ {
        vec.at(17) = 42;
    } ___ {

    }

}
```

Which keyword replaces blank #1 for defining a clause that will attempt to execute the code that may throw an exception?

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    _1_ {
        vec.at(17) = 42;
    } ___ {

    }

}
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } ___ {

    }

}
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } _2_ {

    }

}
```

Which keyword replaces blank #2 for defining a clause to execute code when an exception is thrown?

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } _2_ {

    }

}
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } catch () {

    }

}
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } catch (___ ex) {

    }

}
```

Boundary violations

```
int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } catch (_3_ ex) {

    }

}
```

Boundary violations

```
using namespace std;

int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } catch (_3_ ex) {

    }

}
```

Which type replaces blank #3 to catch an `out_of_range` exception?

```
using namespace std;

int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } catch (_3_ ex) {

    }

}
```

Boundary violations

```
using namespace std;

int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } catch (const out_of_range& ex) {
        cout << "Caught an out_of_range exception: " << ex.what() << endl;
    }

}
```

```
% g++ --std=c++11 bounds_check.cpp -o bounds_check.o
% ./bounds_check.o
Caught an out_of_range exception: vector
```

ex.what()

Boundary violations

```
using namespace std;

int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } catch (const exception& ex) {
        cout << "Caught an exception: " << ex.what() << endl;
    }

}
```

```
% g++ --std=c++11 bounds_check.cpp -o bounds_check.o
% ./bounds_check.o
Caught an exception: vector
```

ex.what()

Boundary violations

```
using namespace std;

int main() {
    vector<int> vec;

    vec.push_back(28);
    try {
        vec.at(17) = 42;
    } catch (exception ex) {
        cout << "Caught an exception: " << ex.what() << endl;
    }

}
```

need reference

```
% g++ --std=c++11 bounds_check.cpp -o bounds_check.o
% ./bounds_check.o
Caught an exception: std::exception
```

ex.what()