Lab 12

Part 3:

* + 1. *Which exception handler is called?*

The following exception handler is called:

catch (...)

{

cerr << "exception\n";

return 2;

}

* + 1. *Based on this observation, what happens if the exception handler nearest to the try block does not contain an exception handler for the thrown object?*

If an exception is thrown, every catch block is checked for the exception.

* + 1. *What does the exception handler with the ellipsis (...) catch?*

All errors besides runtime error and someother\_error.

Part 4:

* + 1. *Which exception handler is called?*

The val\_not\_found exception is called.

* + 1. *Based on this observation, what happens if there is an exception handler nearest to the try block for the thrown object?*

The program will check that exception handler and attempt to catch the thrown object.

Changed code:

#include <iostream>

#include <vector>

#include <stdexcept>

// creates two new types... don't worry about this syntax, but understand that we can make objects of type someother\_error and of type val\_not\_found... that we can throw... and subsequently catch.

class someother\_error {};

class val\_not\_found {};

using namespace std;

int& find\_int(vector<int>&, int);

int main()

try

{

vector<int> vint {2, 4, 6, 8};

cout << "Enter value to find: ";

int val;

cin >> val;

try {

int& i = find\_int(vint, val);

} catch(val\_not\_found &e) {

cerr << "val\_not\_found" << endl;

return 7;

} catch (someother\_error &e) {

cerr << "exeption: someother\_error" << '\n';

}

return 0;

} catch (runtime\_error& e)

{

cerr << e.what() << '\n';

return 1;

} catch (...)

{

cerr << "exception\n";

return 2;

}

int& find\_int(vector<int>& vint, int val)

{

for (decltype(vint.size()) i = 0; i < vint.size(); ++i)

// decltype as used in this expression essencially says that I want the base type of i to be whatever type the value returned by vint.size() is.

if (val == vint.at(i))

return vint.at(i);

throw val\_not\_found{};

}