CSE 425 Studio: Control Abstraction II

These studio exercises are intended to extend your familiarity with ideas and techniques for program control flow involving exception handling, co-routines, and events, again in a multi-paradigm programming language (C++). In this studio you will again work in self-selected groups of ~3 people, and will report the results of your work on the studio exercises on the course message board. Students who are more familiar with the material are encouraged to partner with and help those who are less familiar with it, and asking questions of your classmates and professor during the studio sessions is highly encouraged as well. There are also links on the main course web page to helpful web sites for these exercises, which you are also encouraged to use as resources. Please record your answers as you work through the following exercises. After you have finished please post your answers to the required exercises (and optionally to the enrichment exercise) to the course e-mail account with "Control Abstraction Studio II" in the subject line. The enrichment exercise is a chance to dig deeper into the material, especially if you breeze through the required ones.

PART I: REQUIRED EXERCISES

1. Form a team of ~3 people and write down your names as the answer to this exercise.

2. Create a new project in Visual Studio 2013 (for example, **`ControlAbstractionStudioII`** or something similar that identifies which studio this is for). Modify the main function signature so that it matches the standard portable (e.g., between Windows and Linux) main function signature (and variable names) for a C++ program: **`int main (int argc, char * argv[])`**. Declare and define a function that always throws an **`int`** and call that function directly from main (without a surrounding try/catch block). Build and run your program and observe what happens. Then, surround the call to the function with a try/catch block, again build and run your program, and observe what happens. As the answer to this exercise, please describe what you observed in each case and why you think what you saw happens for each variation.

3. In your main function, declare a try/catch block that has a single catch statement that catches an **`int`** variable by value and prints out its value. Inside the try block declare an **`int`** variable but do not initialize it. Right after the declaration, throw the int variable. See if you can build and run your code (and if so what value is output). Then, initialize the variable, and see if you can build and run your code (and if so what value is output). Change the type of the variable you throw (but not the type of variable you catch) to **`char`** and see if you can build and run your code (and if so whether an exception was caught and if so what value is output). Change the type of the variable you throw to **`double`** and see if you can build and run your code (and if so whether an exception was caught and if so what value is output). As the answer to this question please indicate what happened in each of those cases, and for each of them please explain why you think that happened.

4. Declare and define (or use ones from a previous set of studio exercises) a base class (or struct) and another class (or struct) derived from it, both of which have a const virtual method named **`print`** that takes no parameters and prints out the name of the class to the standard output stream. In your main function declare a try/catch block that has catch statements that catch each of the class (or struct) types by reference and each call the caught exception's **`print`** method. Inside the try block instantiate and throw an instance of the derived class (or struct).

Try to build and run your program (1) with the more specific catch block ahead of the more general one, and then (2) with the more general catch block ahead of the more specific one. As the answer to this exercise please describe what you saw in each of those cases and what your observations say about how catch blocks should be ordered in C++ and to what extent the language enforces that.

5. Declare and define a default constructor, copy constructor, and assignment operator in both the base and derived types (classes or structs) from the previous exercise – the copy constructor and assignment operator should take a reference to a const object of the base type in the base type's version and of the derived type in the derived type's version. In each of those methods add a statement that prints out (to the standard output stream) which method it is and the address of the object on which it was called.

Repeat the previous exercise and observe (1) how many times each of those methods was called on each particular object, and (2) on how many different objects they were each called, and as the answer to this exercise please offer an explanation for why you saw what you saw.

6. Repeat the previous exercise but catch by value instead of by reference. Again build and run your program with the catch blocks being declared in the two different orders. As the answer to this exercise please indicate whether (and if so how) what you saw in this exercise differed from what you saw in the previous exercises, and explain why you saw (or did not see) any difference.

PART II: ENRICHMENT EXERCISE (Optional, feel free to skip or try variations that interest you)

7. Write an input handler function that uses the standard output stream to prompt the user to type in a string of text, reads a (C++ style) string of text from the standard input stream, and if the string is "quit" throws an exception but otherwise returns the string. From within a try/catch block in your main function call the input handler repeatedly and pass each string returned from the input handler to another function that keeps track of how many times the user has typed in each different string it sees, and prints out a message with the string and its current count. Build and run your program with different inputs and observe its behavior. As the answer to this exercise, show your program and its output.