CSE 332 Studio Session on C++ Statements

These studio exercises are intended to introduce basic features and techniques for using C++ statements (with a specific focus on statements related to exceptions), and to give you experience using them within the Visual Studio C++ environment.

In this studio you will again work in small groups of 2 or 3 people, selected at your discretion. As before, students who are more familiar with the material are encouraged to help those for whom it is less familiar. Asking questions of your professor and teaching assistants (as well as of each other) during the studio sessions is highly encouraged as well.

Please record your answers you work through the following exercises. After you have finished please send your answers to the required exercises, and to any of the enrichment exercises you completed, in an e-mail to the cse332@seas.wustl.edu course e-mail account, with the subject line "Statements Studio". The enrichment exercises are optional but are a good way to dig into the material a little deeper, especially if you breeze through the required ones.

PART I: REQUIRED EXERCISES

1. Find your team members in the studio area, sit down at/around and log in to one of the Windows machines, and write down the names of the team members who are present (if a team member arrives late please catch them up on the work, and add their name) as the answer to this exercise.

2. Open up Visual Studio, and create a new Visual C++ Win32 Console Application project for this studio. Throughout these exercises you will replace the code for previous exercises with new code – one way to do this without losing what you've already done (so you can refer back to it later it that's useful) is to comment out the old code. Change the signature of the main function in the source file that Visual C++ generated to match the one that was specified in the lecture slides on C++ program structure. Use the name of this project as the answer to this exercise.

3. In your main project source file, define a function with a statement that is always reached in the execution of that function, which throws an **`int`**. Call that function directly from main (without a surrounding try/catch block). As the answer to this exercise, please describe what happens when you run the program and why you think that happens.

4. Extend your solution to the previous exercise by putting a try block around the call to that function, with a catch block that catches an **`int`** (by value), prints out the value of the integer using cout, and returns the integer as the result of the main function. As the answer to this exercise, please show the output of your program when you run it.

5. Using the solution to the previous exercise, what happens if the function throws a **`char`** instead of an **`int`**? What happens if the function throws a variable of type **`long`**? What happens if the function throws a **`string`**? As the answer to this exercise, please describe what happens in each of these cases, and explain why what you saw is happening.

6. Extend your solution to the previous exercise so that each of the following types are caught, their values are printed out, and an appropriate non-zero integer is returned from the main function: **`char`**, **`int`**, **`long`**, **`string`**, **`char *`**, and **`int *`**. Run your program with the function throwing different variables of type **`char`**, **`int`**, **`long`**, **`string`**, **`char *`**, and **`int *`**, and as the answer to this exercise please print out the results of each of those runs.

Note: while it is fine to catch **`char`**, **`int`**, **`long`**, **`char *`**, and **`int *`** exceptions by value, you should declare the catch block for a **`string`** to catch that type by reference.

PART II: ENRICHMENT EXERCISES (optional, do the ones that interest you)

7. Add a default catch block (that can catch any type), and in that catch block have the program print out a message saying that an unrecognized type was caught and then return an appropriate (non-zero) integer value. As the answer to this exercise please explain what happens if you throw a type (e.g., **`long *`**) that is not listed among the specific types caught by the other catch blocks than the one you just added.

8. Using the code from the previous exercises, as the answer to this exercise please explain what happens if the order of the catch blocks is changed – do some catch blocks "intercept" types that should have been caught by other catch blocks and if so why is that happening?

9. Using the code from exercise 7, as the answer to this exercise please investigate and explain what happens if (1) the default catch block says **`throw;`** (with no exception type) in order to rethrow whatever exception it had caught, or (2) the function that originally threw the exception says simply **`throw;`** (with no exception type) instead of something like **`throw lp;`** (where **`lp`** is of type **`long *`**) – in each of these cases, does the code compile, and if it does is an exception caught, and why do you think that does or does not happen?