CSE 425 Studio: Target Machine Architecture

These studio exercises are intended to give you familiarity with ideas and techniques for managing and understanding target machine architecture details, through examples 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 in an e-mail sent to the course account. 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. Please record your answers as you work through the following exercises. After you have finished please e-mail your answers with "Target Machine Studio" in the subject line, to the cse425@seas.wustl.edu course account. The enrichment exercises offer 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. On the Start menu in Windows, open up Visual Studio 2013 and create a new Win32 Console Application project for these studio exercises. 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[])`**.

In your main function use the **`sizeof`** operator to determine and print out the specific sizes of the following data types: **`char`**, **`bool`**, **`short`**, **`int`**, **`wchar_t`**, **`char16_t`**, **`long`**, **`char32_t`**, **`float`**, **`double`**, **`long long`**, and **`long double`**. Also use the **`sizeof`** operator to determine and print out the specific sizes of unsigned and signed **`char`**, **`short`**, **`int`**, **`long`**, and **`long long`**. Build and run your program, and as the answer to this exercise comment on whether or not each of the data size rules described in today's lecture are obeyed (and also explain why or why not for each rule).

3. For each of the types in the previous exercise, use the **`sizeof`** operator to obtain and print out the size of a pointer to that type. Also declare a pointer to a function (e.g., to the **`main`** function) and use the **`sizeof`** operator to obtain and print out its size.

Build and run your program and as the answer to this exercise describe the relationship (or relationships) among the sizes of those different pointer types.

4. Declare arrays of a couple of different types of different sizes, and print out the addresses of the elements at different locations in each array. Also initialize a pointer to the start of each array and increment it until it points just past the last element of the array, printing out its value (i.e., the address where it points) at each step. Build and run your program and as the answer to this exercise describe (1) the direction in which pointers move (upward or downward) in the address space when moving through the array, (2) the arrangement of successive locations in the array relative to the address space (ascending or descending), and (3) the relationship between the sizes of the object and how far each pointer moves.

5. Declare a simple function that takes an **`unsigned int`** by value that prints out the value and address of the passed **`unsigned int`** parameter, and if the value of the **`unsigned int`** is greater than zero calls itself recursively with one less than the value it was passed. Call that function a few times in the **`main`** function with different non-zero values.

Build and run your program and as the answer to this exercise please describe (1) the relationship between the address of the passed **`unsigned int`** parameter on subsequent recursive calls, (2) whether or not you can infer the sizes of the stack frames from the addresses that were printed out, and (3) if so what the sizes were (or if the same what the size was).

6. Declare a basic struct with two (public) member variables of different types of different sizes. Declare another struct derived from the previous one through public inheritance, that has its own (public) member variable of yet another type. In your main function declare an object of each of the struct types, and print out the addresses of those objects and of each of their member variables.

Build and run your program and as the answer to this exercise please describe the layouts of the objects in terms of the order in which their member variables are arranged, the sizes of the objects relative to the sums of the sizes of their member variables, and any other interesting details you may notice.

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

7. Allocate a large number of variables on the heap (using the **`new`** operator), and print out the address of each one. Is there a pattern to the addresses throughout the allocations, and if you allocate even more variables does the pattern change? What if you allocate different types of objects vs. all the same type? What happens if you allocate some variables, de-allocate some (using the **`delete`** operator), then allocate some more? Give the answers to those questions as the answer to this exercise.

8 and beyond. Repeat any of the previous exercises on the shell.cec.wustl.edu Linux server using g++. As the answer to this question please indicate whether or not any of the details changed on that machine, and if so which details and how they changed.