

HW 1: Review of C Language, Chapters 1, 2 & 3

Due: Monday September 26, 2011 on or before 3:20 pm

1 Big-endian or little-endian (15 pts)

Write a C program that specifies whether the processor that it is running on is big-endian or little-endian. Test your program on `crow.unl.edu` and `cse.unl.edu`.

2 Self-reproducing code (30 pts)

In this problem, you will create a self-reproducing program or *quine*. Self-reproducing program, when execute, generates an output that is an exact copy of its own source code. For example, the following program is quine.

```
int main(){char *c="int main(){char *c=%c%s%c;printf(c,34,c,34);}";printf(c,34,c,34);};
```

To help you start, I am providing you with a simple program that simply outputs the one line to *stdout*. The program is as follows:

```
1:
2: #include <stdio.h>
3:
4: char name[] = "Ying Lu";
5: int main(){ printf ("%s\n", name);}
```

Modify this program to create a quine. Replace the value of the variable *name* with your name. You **must** keep *line 2* and *line 4*. You must also keep a blank line before and after *line 2*. You are allowed to replace *line 5* and add any variables or instructions (after *line 4*) deemed necessary. At the end, your program must generate an output that is exactly the same as your new source code. **Notice, if you add any comments, they must also be displayed in the output.** To validate your result, redirect (`>`) the output of your program into a file and use `diff -w` command to compare your output to the source program. **Do not submit your program unless *diff* reports no differences.**

3 Having fun with C language (25 pts)

Download *problem.zip* from *the course website*. Unzip this file in your CSE account. Inside the problem directory, there are three programs that can be compiled using `cc` command in CSE. To compile and run a program (e.g. *prob1.c*), you would issue the following commands:

- `cc prob1.c -o prob1` (to compile)
- `./prob1` (to execute)

Note that *prob2.c* needs one additional argument. It can be any integer (e.g. `./prob2 5000`). The detailed descriptions of these three programs can be described as follows:

1. *prob1.c* [10 pts] is a simple C program that calls two functions *func1* and *func2*. I want you to observe the output of this program on `crow.unl.edu` or `cse.unl.edu`. Notice that *result1* is initially set to 2500, and *result2* is set to -500. However, the code did not touch *result1* after initializing it, but its value changes from 2500 to -500. Your task is to explain why such situation occurs.
2. *prob2.c* [10 pts] is another simple program that contains a dynamic array. Traditionally, dynamic array is not possible as the size of an array must be known at compile time. In later generation compilers, dynamic array is possible. Specifically, this program sets the array size based on the user's input. Your task is to explain how is this possible, that is, in which part of memory (stack, heap, etc.) is this dynamic array located? Try using 20,000,000 as the argument, what happen to this program and explain why?
3. *prob3.c* [5 pts] is a short program to illustrate type casting. Your task is to explain how this process works by simply drawing the memory contents of after *myX* is initialized and then do the same for the memory contents of *myY* after casting.

4 Chapters 1, 2 & 3 problem assignment (30 pts)

1. What is multiprogramming? What is the main purpose of multiprogramming? [5 pts]
2. On early computers, every byte of data read or written was handled by the CPU (i.e., there was no direct memory access (DMA)). What implication does this have for multiprogramming? [10 pts]
3. What is swapping and what is its purpose? [5 pts]
4. List three general categories of information in a process control block. [5 pts]
5. What are the steps performed by an OS to create a new process? [5 pts]

5 Submission procedure

Create a zip file that has all your solutions and submit through hand-in. The step to create proper directory structure is as follows:

1. Create a directory called *lastname_hw1* (replace *lastname* with your lastname).
2. Create subdirectories: *prob1*, *prob2*, *prob3*, *prob4* under *lastname_hw1*.
3. Place your solutions in the proper directory. Provide README.txt file for *prob1*, *prob2*. Each README file should specify:
 - Specific instructions on how to test your solution.
 - The number of hours you spent on the problem.
 - The level of challenge from 0 to 10 (10 is most difficult).
 - The necessary prior knowledge you have to attack the problem.
4. Once all solutions are properly stored, zip the folder *lastname_hw1* and submit the zip file through handin.