CS 240 (Park) **Final** **July 31 (Wed.), 2024**

Remarks: Keep the answers compact, yet precise and to-the-point. Long-winded answers that do not address the key points are of limited value. Binary answers that give little indication of understanding are no good either. Time is not meant to be plentiful. Make sure not to get bogged down on a single problem.

PROBLEM 1 (45 pts)

(a) Explain the difference between function prototype, char *f(int *), and function pointer, char (*h)(int *). How can the declaration of h be modified so that the assignment statement, h = f, is meaningful? Explain your thought process.

(b) Suppose we are given, typedef struct superdat {char m; char *n; double k;} superdat_t, and r declared as, superdat_t *r. How many bytes does superdat_t take up in the main memory of our lab machines? How many bytes does r take up? Suppose we want to store ASCII character '?' in field m, string "hello" in field n, and real number 12.345 in field k. Write code snippet that makes calls to malloc() and uses assignment statements to achieve this goal. You may use the string library function strcpy(char *, const char *) that copies the string pointed to by the second argument into the first argument. Why does the second argument have the const qualifier whereas the first argument does not?

(c) Suppose variable u is declared as, union abc {int x; int y; unsigned int z;}. Suppose the following three consecutive assignment statements are made followed by a call to printf(): u.x = -5; u.y = -7; u.z = 5; printf("%d %d %u", u.x, u.y, u.z). What values are output, and why? Describe a scenario where union abc {int x; int y; unsigned int z;} may be preferred over struct abc {int x; float y; unsigned int z;}.

PROBLEM 2 (30 pts)

(a) Given, struct legacydat {int a; float b; int c;}, and variable x declared as, struct legacydat x, suppose the second field, float b, is not being used and, therefore, is to be repurposed to store four ASCII characters instead of a real number of type float. Using variable, void *z, write code snippet that stores characters 'A', 'B', 'C', 'D' into the middle four bytes of x that were originally intended to store a value of type float. Will this code generate a warning when compiled with gcc?

(b) Code an app myadder as, int main(int argc, char **argv), that takes a filename (a string) followed by integers as command-line arguments, adds up the integers, and outputs the sum to a file of the specified filename. For example, running myadder in a shell

```
% myadder output.dat 10 20 30
```

will save 60 into file output.dat. The file should be created if it doesn't already exist. Since command-line arguments are passed as strings, utilize the library function, int atoi(const char *), to convert a string representing an integer into a value of type int. The app needs to be executed with at least two command-line arguments, the first specifying a filename, the second argument an integer. Verify that this condition is met. If not, output an error message and terminate the app. No need to worry about header files or checking for other error conditions.

PROBLEM 3 (25 pts)

Code a variadic function, int subchar(char *, ...), that takes a string as first argument. You may assume that string length is at least 1, not counting the EOS character. Denoting string length as n, subchar() assumes that there are n additional arguments of type char containing ASCII characters that follow the first argument. subchar() modifies the content of the string by replacing each character of the string with its corresponding argument of type char that has been passed. For example, assuming variable, char *s, points to string "123", calling subchar(s, 'a', 'b', 'c') changes the content of s to "abc". subchar() returns string length n. subchar() should perform error checking by verifying that the additional arguments passed are ASCII characters. If not, subchar() outputs an error message to stdout and terminates. Ignore header files.

BONUS PROBLEM (10 pts)

In our lab assignments where we called fopen() to open a binary file to read, we did not have to use mode "rb"; instead using "r" sufficed. In what practical situation is using the binary flag "b" for reading bytes meaningful?