

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 (30 pts)

(a) Explain the key difference with respect to memory layout between `x` declared as, `struct abc x`, where

```
struct abc { int a; unsigned int b; };
```

and `y` declared as, `union qrs y`, where

```
union qrs { int a; unsigned int b; };
```

When would it make sense to choose union over struct? When would it not be appropriate? What is the value of `y.a` after the two consecutive assignment statements `y.a = 100; y.b = 200;` Explain your reasoning.

(b) What is the reason that the library function, `int fgetc(FILE *)`, that reads a byte from a file pointed to by its argument returns a value of type `int` instead of `char`? For the code snippet

```
float r; double s; s = -21.123456789; r = s; printf("%.9f", r);
```

what is likely to happen when `printf()` is called, and why?

PROBLEM 2 (30 pts)

(a) For the example function, `int mysum(int, ...)`, discussed in class which returns the sum of its variable number of arguments, the first argument of type `int` played a special role in that it specified how many additional arguments followed. How did function `mysum()` determine what the type of the additional arguments (e.g., `int`, `float`) to be added are? How does the variadic function `scanf()` whose first argument is not an integer know how many additional arguments to expect? What is likely to happen when `printf("%f", u, v)` is executed where `u` and `v` are variables of type `float`? What is likely to happen when `printf("%f %f", u)` is executed? Explain your reasoning based on how `printf()` is coded.

(b) Suppose a C programmer codes a function, `int dosomething(const char *)`, that takes a string as argument and returns a value of type `int`. What is the role of the qualifier `const` in the argument? Does it provide any guarantees to the programmer whose code calls `dosomething()`? Explain your reasoning. Provide an example code snippet in your explanation.

PROBLEM 3 (40 pts)

(a) Write `main()` that takes multiple strings where each string represents an integer as command-line arguments, converts them to integers using `atoi()`, multiplies the numbers, and prints the result to `stdout`. For example, executing `main()` compiled as `a.out` and run from a shell, `% a.out 10 10 3`, results in 300 being output. The code should be complete but for inclusion of header files.

(b) Suppose a text file containing only ASCII characters is comprised of lines where each line ends with newline character `'\n'`. You may assume that a line is not more than 100 characters long inclusive the newline character. Code `main()` that accepts a filename followed by an integer as command-line arguments where the integer specifies the number of lines in the file. `main()` opens the file and reads its content byte-by-byte using `fgetc()`. While doing so `main()` stores the lines in memory as strings in a 1-D array of strings, `char **stringarray`, by calling `malloc()` to allocate just enough memory to store the strings. You may use `strcpy(char *, char *)` that copies the string pointed to by the second argument into the first argument. For brevity, you may omit checks such as verifying that `fopen()` succeeded or that `malloc()` did not return `NULL`.

BONUS PROBLEM (10 pts)

How much space does, `struct abc q`, where `struct abc { unsigned int x0 : 1; unsigned int x1 : 4; unsigned int x2 : 2; }` occupy in main memory? What range of values may be assigned to `q.x0`, `q.x1`, `q.x2`?