

CS240 Final Solution, summer 2024

P1(a) 15 pts

`char *f(int *)` denotes a function that takes a pointer to `int` as argument and returns a pointer to `char` (i.e., an address where its content is `char`).
5 pts

Function pointer, `char (*h)(int *)`, points to a function that takes a pointer to `int` as argument and returns a value of type `char`.
5 pts

Change to, `char *(*h)(int *)`. By prepending `*` to `(*h)` we denote that the function pointer returns a pointer to `char`.
5 pts

P1(b) 15 pts

$8 + 8 + 8 = 24$ bytes
3 pts

// 8 bytes for `char m` since gcc allocates in blocks of 8 bytes due to size of other fields. `char *n` is 8 bytes since lab machines are 64-bit.
// $4 + 8 + 8$ is also acceptable if `char` is assumed to be allocated 4 bytes.

`r` takes up 8 bytes since it is a pointer, i.e., its content is an address.
2 pts

`r = malloc(sizeof(superdat_t));` // No penalty for not checking NULL return.
2 pts

`r->n = malloc(sizeof("hello"));`
2 pts

`strcpy(r->n, "hello");`
2 pts

`r->m = '?';`
1 pt

`r->k = 12.345;`
1 pt

`strcpy()` modifies the string pointed to by the first argument (hence no `const`) but the second argument is only accessed for read. Hence the `const` qualifier.
2 pts

P1(c) 15 pts

Values output: 5, 5, 5
5 pts

Since the three fields share the same 4 byte memory location, the last assignment statement, `u.z = 5`, overwrites previous writes to `u.x` and `u.y`.
5 pts

In a situation where a data structure such as `u` can possess three different features (i.e., its value can be of three different types) but at any one time only one feature is relevant (needs to be remembered), using union over struct saves memory.
5 pts

// Last sentence of P1(c) has a typo: `_float y_` should be `_int y_`.

P2(a) 15 pts

z = &x;
5 pts

```
*(char *) (z+4) = 'A'; // Skip 4 bytes over the first field int a.  
*(char *) (z+5) = 'B';  
*(char *) (z+6) = 'C';  
*(char *) (z+7) = 'D';  
8 pts
```

No.

2 pts

// Using char *z instead of void *z will generate a warning.

P2(b) 15 pts

```
int main(int argc, char **argv) {  
FILE *fp;  
int i, sum = 0;  
2 pts
```

```
    if(argc < 3) { // If comparison is argc < 2 don't deduct points since the question  
                  // states "at least two command-line arguments" which may be taken  
                  // literally by some.  
        printf("Invalid use.\n");  
        exit(1);  
    }
```

3 pts

```
    if((fp = fopen(argv[1], "w")) == NULL) {  
        fprintf(stderr, "File write failed\n");  
        exit(1);  
    }
```

3 pts

// Deduct 1 point if NULL return check is not performed.

```
    for(i=2; i<argc; i++)  
        sum += atoi(argv[i]);
```

5 pts

```
    fprintf(fp, "%d", sum);  
2 pts
```

}

P3 25 pts

```
int subchar(char *s, ...) {  
va_list arglist;  
int n, x, i;  
2 pts
```

```
    n = strlen(s); // strlen() does not count '\0'.  
2 pts
```

```
    va_start(arglist, s);  
2 pts
```

```
    for(i=0; i<n; i++) {  
5 pts
```

```
        x = va_arg(arglist, int); // Must be int, not char. Deduct 2 pts if char.  
5 pts
```

```
        if(x > 127) {  
            printf("Invalid argument(s).\n");
```

```

        exit(1);
    }
    2 pts

    *(s+i) = (char) x; // Type conversion to char is not needed.
    5 pts
}

va_end(arglist);
2 pts
}

```

Bonus 10 pts

In Windows, text files terminate lines with two characters '\r' (carriage return) and '\n'. When a text file is opened using fopen() without binary mode "b" for reading (i.e., only "r") then Windows may suppress '\r' so that it is not read by fgetc().
5 pts

To prevent Windows from interfering and forcing to truthfully read all characters including '\r' (and '\n') the binary mode is used for read ("rb").
5 pts
 // The above applies not only to Windows but, in practice, that's the most
 // relevant case which was discussed in class. Talking about POSIX etc. is
 // outside the scope, not expected nor needed.