ComSSA teaches UCP: test 2 revision

October 10, 2015

Questions in UCP tests and exams follow a predictable format, and each of the sections below represent a kind of question that you may be asked in an assessment, you may have seen in a practical, or may otherwise improve your understanding.

To better prepare you for your assessments, the questions that follow are often slightly more difficult than you may expect to encounter, but they will help hone your skills and ensure that you pay extra attention when answering the real questions.

Dave likes to write tricky questions with subtle pitfalls, many of which test your understanding of *entirely valid code* that will compile but either run incorrectly, run surprisingly, or be otherwise less than ideal. Watch out for these warning signs:

- When Dave mentions a "pointer to an array", he nearly always actually means a "pointer to the first element of an array", but pointers to entire arrays are also a real concept, even if they are used rarely in practice.

The following are some other hints that may be useful for today:

- When a function is referred to in a manner such as `function(3)`, you can learn more about the function in section 3 of the Unix manual by typing `man 3 function`.
- The exact data type of an expression is what one would write to declare a variable for it, without the name of the variable itself; for example, the type of a pointer to a function with no formal parameters that returns an `int` is `int (*)(void)`.
- The expressions `foo[i]` and `*(foo + i)` are equivalent, and either form can be used to access both fixed and `malloc(3)`'d arrays equally well.
- Whenever the name of an array is used in any expression, the array is temporarily converted to a pointer to its first element in a process that is colloquially known as "array decay", except for the expressions `sizeof(array)` and `&array`.
- The only two places where `[]` makes sense are in the first dimension of an array declaration, where the compiler fills in the size based on the initialiser, and in the first dimension of a function parameter, where it is syntactic sugar for `*`.

Questions marked * are probably beyond the scope of the assessment.

1 Declarations, expressions, and data types

Explain the meanings and the data types of the variables and/or expressions below:

a) `argc; argv; argv[0]; argv[4];`

b) `int *foo = malloc(5 * sizeof int); foo; &foo; *foo;`

c) `int *foo = malloc(5 * sizeof int); foo[3]; &foo[3];`

d) `int *foo = malloc(5 * sizeof int); foo + 3; *(foo + 3);`

e) `int foo[5]; foo; &foo; *foo;`

f) `int foo[5]; foo[3]; &foo[3];`

g) `int foo[5]; foo + 3; *(foo + 3);`

h) `int foo[3][2] = { { 4, 5 }, { 6, 7 }, { 8, 9 } };`

i) `int foo[5][4]; foo; &foo; *foo;`

j) `int foo[5][4]; foo[3]; &foo[3];`

k) `int foo[5][4]; foo[3][1]; &foo[3][1];`

l) `int foo[5][4]; foo + 3; *(foo + 3);`

m) `int foo[5][4]; *(foo + 3) + 1; *(*(foo + 3) + 1);`

n) `int *foo[5][4], **bar[5][4];`

o) `int (*foo)[5][4], (*bar(void))[5][4];`

* p) `int *(*foo[5][4])(int bar, int);`

q) `int foo(float bar[], size_t length);`

r) `int foo(float bar[][COLS], size_t rows);`

s) `int foo(float bar[ROWS][COLS]);`

t) `char *foo = "Hello, world!";`

u) `char foo[] = "Hello, world!";`

* v) `struct Point { int x, y; };`

* w) `struct Point { int x, y; } place;`

* x) `struct { int x, y; } place = { 13, 420 };`

y) `typedef struct { int x, y; } Point;`

z) `typedef struct Point { int x, y; } Point;`

α) `typedef struct Foo { char *bar; } Foo;`

β) `typedef struct Foo { char bar[9]; } Foo;`

γ) `foo.a; foo.b->c.d; foo.e[7]->f[6].g;`

δ) `foo->a; foo->b.c->d; foo->e + 13;`

2 Evaluating the outputs of `printf(3)` calls

For each of the questions below, write what would be sent to `stdout`:

a) `printf("~%d~", 13);`

b) `printf("~%+d~", 13);`

c) `printf("~%5d~", 13);`

d) `printf("~%+5d~", 13);`

e) `printf("~%-5d~", 13);`

f) `printf("~%+-5d~", 13);`

g) `printf("~%05d~", 13);`

h) `printf("~%+05d~", 13);`

i) `printf("~%3.9f~", 13.0);`

j) `printf("~%+3.9f~", 13.0);`

k) `printf("~%9.3f~", 13.0);`

l) `printf("~%+9.3f~", 13.0);`

m) `printf("~%-9.3f~", 13.0);`

n) `printf("~%+-9.3f~", 13.0);`

o) `printf("~%09.3f~", 13.0);`

p) `printf("~%+09.3f~", 13.0);`

q) `printf("%s,%6s!", "Hello", "world");`

r) `printf("~%d~", '7' - '0');`

s) `printf("~%c~", 7 + '0');`

3 Discussion

a) What values can `fopen(3)` return and why?

b) What values can `fgetc(3)` return and why?

c) What values can `scanf(3)` return and why?

d) What values can `fgets(3)` return and why?

e) Why is it incorrect to read a file by looping until `feof(3)` returns true?

f) How can one distinguish reaching the end of a file from an I/O error?

g) How should a C program handle an I/O error?

4 `struct` declaration and allocation

For each of the following descriptions:

i) Write a suitable type declaration (or set of type declarations) for representing the data.

ii) Show how to dynamically allocate the necessary memory, and initialise any obvious fields.

Assume that all strings have an upper limit of 168 characters.

(Note: based on your declarations, it should be possible to use and manipulate all the information through a single pointer variable.)

a) A pharmaceutical drug, with a generic name, a trade name, a year of discovery, a year of approval, a molecular mass in g/mol, and a half-life in hours.

b) A digital audio file, with an artist name, album name, track name, sampling rate in Hz, and an array of audio samples, where each sample is an integer. When allocating and initialising the structure, assume that the integer variable `samples` exists and contains the number of samples in the digital audio file.

c) A web address, or URL, with a host name, a port number, a path, an array of fields, and a fragment name. Each field consists of a name and a value, both of which are strings, as well as the path. When allocating and initialising the structure, assume that the integer variable `fields` exists and contains the number of fields.

d) A text file, consisting of a file name, as well as a double ended, doubly linked list of lines of text. When allocating and initialising the structure, assume that the text file contains no lines. Writing any code to manipulate the linked list is not necessary.

5 Pointer diagrams

Say you have the following declarations:

```
int x = 13;
int y[] = { 1, 4, 9 };
int z[][2] = { 2, 4, 6, 8 };
int *p = NULL;
int *q[2] = { &z[0][2], &y[1] };
int **s;
int **t[2];
```

For each of the following code snippets, draw a diagram showing all pointer relationships created, and state the resulting value of `x`. Assume that the declarations "reset" between questions — in other words, the questions are independent of one another.

a)

```
s = &p;
p = &x;
t[0] = &q[1];
t[1] = &q[0];
s[0][0] = **t[0] + **t[1];
```

b)

```
*t = &p;
t[z[1][0] - 5] = q;
p = (int *) malloc(2 * sizeof(int));
*(p + 0) = y[sizeof(z) / sizeof(z[0])];
*(p + 1) = q[0] - *z;
q[0] = &x;
t[1][0][0] = t[0][0][1] * ***t;
```