UMass Boston CS 240 Test 2 Practice Questions

Oct 27, 2019

Name: _____ Student Number: _____

By signing, I certify that I have neither given nor received unauthorized assistance on this test.

Signature _____

Instructions

1. **Turn off all digital devices.**
2. Textbook and lecture notes are allowed.

Data Type Specification

- char: 1 byte
- int: 4 bytes
- short: 2 bytes
- long: 8 bytes
- long long: 8 bytes
- float: 4 bytes
- double: 8 bytes
- signed int: is int
- unsigned int: or just unsigned

1. What is the output?

```
void question(void)
{
  char str[] = "Coaxial flutter";
  char *p1, *p2;
  short *p3 = (short *) str;
  p1 = &str[7 % 8];
  p2 = str + 3;

  printf("1) [%c]\n", *(p3 + 4));
  printf("2) [%s]\n", &p2[5]);
  printf("3) [%c]\n", *p1++);
  printf("4) [%c]\n", (*p1)++);
  p2[6] += 6;
  p2[8] -= 6;
  printf("5) [%s]\n", (char *)(p3 + 4));
}
```

Answer:

```
1) [f]
2) [flutter]
3) [ ]
4) [f]
5) [grunter]
```

2. If the code allocates just enough memory for the data, say so. If it allocates more than needed or less than needed, fix it.

   (a) We want to make a copy of a string. Does the following code work?

```
char msg[] = "message";
char *msg2;

msg2 = (char*) malloc(sizeof(char) * strlen(msg));
strcpy(msg2, msg);
```

   (b) We want to concatenate two strings into a third string. Does the following code work?

```
char msg1[] = "message ";
char msg2[] = "in a bottle";
char *msg3;

msg3 = (char *) malloc(strlen(msg1) + strlen(msg2) + 1);
strcpy(msg3, msg1);
strcat(msg3, msg2);
```

   (c) We want to concatenate two `int` arrays into a third array. Does the following code work?

```
int a1[5] = {0, 1, 2, 3, 4};
int a2[5] = {5, 6, 7, 8, 9};
int *a3, i;

a3 = (int *) malloc(10);
for (i = 0; i < 5; i++)
  a3[i] = a1[i], a3[i + 5] = a2[i];
```

   (d) What is the output of the `printf` statement?

```
int a[10][10], *b, i, j, k;

for (i = k = 0; i < 10; i++)
  for (j = 0; j < 10; j++)
    a[i][j] = k++;

b = (int *) a;

printf("%d %d\n", b[5], b[95]);
```

   (a) No. Should be:

```
msg2 = (char*) malloc(sizeof(char) * strlen(msg) + 1);
```

   (b) Yes.

   (c) No. Should be:

```
a3 = (int *) malloc(10*sizeof(int));
```

   (d) 5 95

3. (a) The main program has the following prototype:

```
int main(int argc, char *argv[]);
```

You run the executable with the following command:

```
> ./cntSort -m 8 -n 1024 -s 2019
```

i. What is the value of `argc`?
ii. What is the type of `argv`?
iii. What is the type of `argv[3]`?
iv. What is the type of `argv[4]`?
v. What is the output of the following printf statement?

```
printf("%s\n", argv[0])
```

(b) What is the most number of stack frames in memory for this program ?

```
void f1(){};void f2(){}; void f3(){f1();}
void f4(){f3();}void f5(){}; void  f6(){};

int main(int argc, char *argv[])
{
      f1(); f2(); f3();
      f6(); f4(); f5();
return 0;
}
```

(c) Define pointers to reference to this array for all the larger and small dimensions of it:

```
int b[3][3];
```

Answer:

(a) i. 7
ii. char **
iii. char *
iv. char *
v. ./cntSort

(b) 4.
They are main, f4, f3, f1.

(c)
```
int *p = b[0];
int (*pp)[3] = b;
int (*ppp)[3][3] = &b;
```

4. When we want to generate a uniformly distributed integer between 0 and `n - 1`, including 0 and `n - 1`, we can do the following:

```
rndInt = (unsigned) floor(drand48() * n);
```

The function drand48() returns nonnegative, double-precision, floating-point values, uniformly distributed over the interval [0.0,1.0).

(a) Write code to generate a uniformly distributed integer between 2 and 29, including 2 and 29.

(b) Write code to generate a uniformly distributed integer between -17 and 7, including -17 and 7.

Answer:

(a)
```
rndInt_a = (unsigned) floor(drand48() * 28) + 2
```

(b)
```
rndInt_b = (unsigned) floor(drand48() * 25) - 17
```

5. A Caesar cipher shifts the letters by a fixed number of places. For example, let the shift be 3. Then a becomes d, b becomes e, c becomes f, and so on, and z becomes c. Note that last 3 letters rotate to the front.
   (a) Write a function that receives a `char` as a parameter, and uses a switch statement to shift the `char`. If the `char` is a lowercase letter, it is shifted. Otherwise, leave it unchanged. You don't need to write the whole switch. It is sufficient to write the cases for a, b, c, d, and w, x, y, z, and default. The return value of the function is either the shifted lowercase letter or an unchanged `char`.
   (b) Write a loop that uses the function you just wrote to shift every lowercase letter in a string.

```
char msg[] = "my secret message";

/* write a loop to shift/encrypt msg[]*/
```

   (c) What is the output from the following `printf` statement?

```
char ch1 = 'x';
char ch2 = '=';

char func(char ch) {
  return (ch >= 'a' && ch <= 'z') ?
         ((ch - 'a' + 3) % 26 + 'a') :
         ch;
}

printf("%c%c\n", func(ch1), func(ch2));
```

   (d) Write a function that unshift lowercase letters in a message that have been shifted earlier. For example, d becomes a, e becomes b, f becomes c, and so on.

Answer:

(a)

```
char shift(char c) {
  switch (c) {
  case 'a': return 'd';
  case 'b': return 'e';
  case 'c': return 'f';
  ...
  case 'x': return 'a';
  case 'y': return 'b';
  case 'z': return 'c';
  default: return c;
  }
}
```

(b)

```
    char *ch;
    ch = msg;
    while (*ch != '\0') {
      *ch = shift(*ch);
```

```
        ch++;
    }
```

(c) a=

(d)

```
char func1(char ch)
{
    return (ch >= 'a' && ch <= 'z') ?
     ((ch - 'a' + 23) % 26 + 'a') : ch;
}

or

char func2(char ch)
{
    return  ((ch - 'a' + 23) % 26 + 'a');
}
```