CS221 C and Systems Programming – Midterm Review

Which of the following is the correct way to declare a pointer to an integer in C?

a) int *ptr;
b) int ptr;
c) int &ptr;
d) int ptr[];

How do you correctly declare a string in C using a pointer?

e) char str = "Hello, World!";
f) char str[] = 'Hello, World!';
g) char *str = "Hello, World!";
h) char *str = 'Hello, World!';

What is the purpose of the malloc function in C?

i) malloc allocates memory on the heap.
j) malloc allocates memory on the stack.
k) malloc allocates memory in the data segment.
l) malloc allocates memory in the code segment.

Which function is used to deallocate memory that was previously allocated with malloc in C?

m) delete(ptr);
n) remove(ptr);
o) free(ptr);
p) clear(ptr);

What does a pointer store in C programming?

q) A pointer stores the value of a variable.
r) A pointer stores the memory address of a variable.
s) A pointer stores the size of a variable.
t) A pointer stores the type of a variable.

How do you correctly allocate memory for an integer using malloc and cast it in C?

u) int *ptr = malloc(sizeof(int));
v) int ptr = (int *)malloc(sizeof(int));
w) int ptr = malloc(sizeof(int));
x) int *ptr = (int *)malloc(sizeof(int));

What is a string in C programming?

a) A string in C is a single character.
b) A string in C is an array of characters terminated by a null character.
c) A string in C is a pointer to an integer.
d) A string in C is a structure containing characters.

Which expression would you use to determine the size of an integer in bytes in C?

e) sizeof(int *)
f) sizeof(int)
g) sizeof(char)
h) sizeof(float)

How do you declare an array of 10 integers in C?

i) int arr[10];
j) int arr(10);
k) int arr{10};
l) int arr<10>;

What is the purpose of a cast in C programming?

m) A cast is used to convert a variable from one type to another.
n) A cast is used to declare a variable.
o) A cast is used to allocate memory.
p) A cast is used to free memory.

How do you correctly assign the address of an integer variable 'var' to a pointer 'ptr' in C?

q) int *ptr = &var;
r) int ptr = &var;
s) int *ptr = var;
t) int ptr = var;

What is the purpose of the dereference operator in C?

u) The dereference operator (*) is used to access the value at the address stored in a pointer.
v) The dereference operator (&) is used to access the value at the address stored in a pointer.
w) The dereference operator (->) is used to access the value at the address stored in a pointer.
x) The dereference operator (.) is used to access the value at the address stored in a pointer.

What is a pointer to a pointer in C programming?

a) A pointer to a pointer is a variable that stores the value of another pointer.
b) A pointer to a pointer is a variable that stores the address of another pointer.
c) A pointer to a pointer is a variable that stores the size of another pointer.
d) A pointer to a pointer is a variable that stores the type of another pointer.

How do you correctly allocate memory for an array of 10 integers using malloc and cast it in C?

e) int *ptr = malloc(10 * sizeof(int));
f) int *ptr = (int *)malloc(10 * sizeof(int));
g) int ptr = (int *)malloc(10 * sizeof(int));
h) int ptr = malloc(10 * sizeof(int));

What is the purpose of the address-of operator in C?

i) The address-of operator (*) is used to obtain the memory address of a variable.
j) The address-of operator (->) is used to obtain the memory address of a variable.
k) The address-of operator (&) is used to obtain the memory address of a variable.
l) The address-of operator (.) is used to obtain the memory address of a variable.

What is a segmentation fault in C programming?

m) A segmentation fault occurs when a program runs out of memory.
n) A segmentation fault occurs when a program enters an infinite loop.
o) A segmentation fault occurs when a program tries to access a memory location that it is not allowed to access.
p) A segmentation fault occurs when a program encounters a syntax error.

Short Answer Questions

1. Explain the difference between stack memory and heap memory in C.

Stack memory stores local variables, input variables, and return value for each method currently running.

Heap stores memory allocated via malloc().

2. What is a segmentation fault and what are common causes of it in C programming?

A segfault is an attempt to access memory that the program does not have permission to access. Could be from trying to write to read-only memory ("data memory") or to read from "code memory area".

Code Analysis Questions

1. Given the following C code snippet, identify any errors and correct them:

```
#include <stdio.h>
int main() {
    int *ptr = (int*) malloc(sizeof(int));
```

```
    *ptr = 10;

    printf("%d\n", *ptr);
    free(ptr);
    return 0;
}
```

2. What will be the output of the following code snippet, and why?

```
#include <stdio.h>
int main() {
    int x = 5;
    int y = 10;
    int *p1 = &x;
    int *p2 = &y;
    *p1 = *p2;
    printf("%d %d\n", x, y);
    return 0;
}
```

Output is : 10 10. When we set *p1 to *p2, we set the value of x to the value of y, since p1 points at x and p2 points at y.

1. Implement a method to swap two numbers using pointers.

```
void swap(int* a, int* b) { int tmp = *a; *a = *b; *b = tmp; }
```

2. Write a **recursive** function to calculate the length of a string, without using strlen().

```
int myStrLen(char* ptr) {
  if (ptr[0] == 0) { return 0; }
  return 1 + myStrLen(ptr+1);
}
```

3. Write a function to reverse a string without using the standard library functions.

```
void reverse(char* str) {
  int len = 0;
  while (str[len] != 0) { len++; } // get length of string
  for (int n = 0; n < len/2; n++) {  // swap the characters
     char tmp = str[n]; str[n] = str[len-n-1]; str[len-n-1] = tmp;
  }
}
```

4. Write a program that reads the first command line argument, allocates memory for a string long enough to hold that argument, and copies the command line argument to the string, replacing any numerical digits with '#'.

```
#include <string.h>
int main(char argc, char** argv) {
   const int LEN = strlen(argv[1]); // argv[0] probably ok too, given the instructions
   char* myString = (char*) malloc(LEN);
   strncpy(myString, argv[1], LEN);
   for(int n = 0; n < LEN; n++) {
      if (myString[n] >= '0' && myString[n] <= '9') {
         myString[n] = '#';
      }
    }
   free(myString);
   return 0;
}
```

5. Write a Makefile to compile your previous program

```
myProgram: myProgram.c
   gcc -o myProgram myProgram.c
```

6. Write a C program to read a text file and count and print the number of capital letters in it.

```
#include <stdio.h>
int main() {
   FILE* theFile = fopen("TheTextFile");
   char theString[256]; // should be big enough
   int numCaps = 0;
   while(fscanf(theFile, "%256s", theString)) { // while we read a string successfully…
      for (int n = 0; theString[n] != 0; n++) {
         if ('A' <= theString[n] && theString[n] <= 'Z') { numCaps++; }
      }
   }
   fclose(theFile);
```

```
  printf("%d\n", numCaps);
  return 0;
}
```

7. Debug the given C code and explain the fix.

```
unsigned short* array =
     (unsigned short*) malloc(200 * sizeof(unsigned short));
for (int v = 0; v < 200; v++) {
     array[v] = (short) v;
}
free(array); // cannot change the value of the pointer given
to free()
```

8. Implement a function that counts the number of vowels in a given string.

```
int vowels(char* str) {
  int retval = 0;
  while (*str != 0) {
     if (*str == 'a' || *str == 'A' || *str == 'e' || *str == 'E' || *str == 'i' || *str == 'I' ||
        *str == 'o' || *str == 'O' || *str == 'u' || *str == 'U') { retval++; }
     }
     str++;
  }
  return retval;
}
```

9. Write a method to concatenate two strings without using strcat(). How would you make a "safe" version?

```
void myCat(char* dest, char* catMe) {
  // find end of 'dest'
  while (*dest != 0) { dest++; }
  while (*catMe += 0) { *dest++ = *catMe++; }
}
```

To make this safe, I would pass in the size of 'dest' and would make sure I don't write past the end of the allocated memory for 'dest'.

10. Write a function that checks if a given string is a palindrome, using pointers.

```
int isPal(char* inp) {
  // find end of 'inp'
  char* end = inp;
  while ( *end != 0 && *(end+1) != 0) { end++; }
  // check whether 'inp' is a palindrome
  while (inp != end) {
    if (*inp != *end) { return 0; }
    inp++; end--;
  }
  return 1;
}
```