ECE 551 PRACTICE Midterm Exam

This is a full length practice midterm exam. If you want to take it at exam pace, give yourself 75 minutes to take the entire test. Just like the real exam, each question has a point value. There are 75 points in the exam, so that you can pace yourself to average 1 point per minute (some parts will be faster, some slower).

Questions:

1. Multiple Choice: 8 points
2. Tracing C code: 12 points
3. Tracing C++ code: 12 points
4. Dynamic Allocation: 10 points
5. Coding 1: 11 points
6. Coding 2: 11 points
7. Recursion: 11 points

This is the solution set to the practice exam. The solutions appear in blue.

Question 1: Multiple Choice [8 pts]

For each of the following questions, circle the **best** answer:

1. Which tool helps you find memory leaks? valgrind

   (a) gdb
   (b) gcc
   (c) svn
   (d) valgrind
   (e) None of these

2. Returning the address of a local variable... results in a dangling pointer

   (a) ...is always ok.
   (b) ...is ok as long as you `free` it later.
   (c) ...results in a dangling pointer.
   (d) ...immediately segfaults.
   (e) None of these

3. Which of the following is a templated class? vector

   (a) vector
   (b) string
   (c) int
   (d) const int *
   (e) None of these

4. Doing `std::cout << something` is most closely related to what C function? printf

   (a) fgets
   (b) printf
   (c) malloc
   (d) free
   (e) realloc

Question 2: Tracing C Code [12 pts]

What is the output when the following C code is executed?

```
#include <stdio.h>
#include <stdlib.h>

void f(int x, int * p) {
  printf("x = %d, *p = %d\n", x, *p);
  *p = x + 42;
  x--;
  p = NULL;
}

int main (void) {
  int a = 1;
  int b = 2;
  int * p = & a;
  int **q = & p;
  printf("**q = %d\n", **q);
  f(*p, *q);
  printf("a = %d\n", a);
  if ( p == NULL ) {
    printf("p is NULL\n");
  }
  else {
    printf("*p is %d\n", *p);
  }
  *q = &b;
  printf("*p = %d\n", *p);

  return EXIT_SUCCESS;
}
```

Answer:

```
**q = 1
x = 1, *p = 1
a = 43
*p is 43
*p = 2
```

Question 3: Tracing C++ Code [12 pts]

What is the output when the following C++ code is executed?

```
#include <iostream>
#include <cstdlib>
class C {
private:
  int x;
public:
  C(int _x): x(_x) {
    std::cout << "C(" << x<< ")\n";
  }
  ~C() {
    std::cout << "~C(" << x<< ")\n";
  }
  int & getX() {
    return x;
  }
  C operator+(const C& rhs) const {
    std::cout << "x = " << x << " rhs.x = " << rhs.x << "\n";
    C ans(x + rhs.x);
    return ans;
  }
};
int main(void) {
  C a(3);
  C b(4);
  C c = a + b;
  int & x = a.getX();
  std::cout << "x is " << x << "\n";
  x = 42;
  C d(c + a);
  std::cout << "ret\n";
  return EXIT_SUCCESS;
}
```

Answer:

```
C(3)
C(4)
x = 3 rhs.x = 4
C(7)
x is 3
x = 7 rhs.x = 42
C(49)
ret
~C(49)
~C(7)
~C(4)
~C(42)
```

Question 4: Dynamic Allocation [10 pts]

Consider the following C code (which has errors in it):

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct _my_struct {
5    int nNums;
6    int * nums;
7  };
8  typedef struct _my_struct my_struct;
9
10 void f(my_struct * ptr) {
11   for (int i = 0; i <= ptr->nNums; i++) {
12     printf("%d\n", ptr->nums[i]);
13   }
14   free(ptr);
15 }
16 int main(void) {
17   my_struct * s = malloc(sizeof(s));
18   s->nNums = 5;
19   s->nums = malloc(5 * sizeof(*s->nums));
20   for (int i = 0; i < s->nNums; i++) {
21     s->nums[i] = i + 4;
22   }
23   f(s);
24   free(s);
25   return EXIT_SUCCESS;
26 }
```

`valgrind` reports the following information about this program (which you may find useful in helping you find the problems):

```
Invalid write of size 8
   at 0x400612: main (q4.c:19)
 Address 0x51f1048 is 0 bytes after a block of size 8 alloc'd
   at 0x4C2B6CD: malloc (in ...)
   by 0x4005F2: main (q4.c:17)

Invalid read of size 8
   at 0x400623: main (q4.c:21)
 Address 0x51f1048 is 0 bytes after a block of size 8 alloc'd
   at 0x4C2B6CD: malloc (in ...)
   by 0x4005F2: main (q4.c:17)

Invalid read of size 8
   at 0x40059D: f (q4.c:12)
   by 0x400656: main (q4.c:23)
 Address 0x51f1048 is 0 bytes after a block of size 8 alloc'd
   at 0x4C2B6CD: malloc (in ...)
   by 0x4005F2: main (q4.c:17)

Invalid read of size 4
   at 0x4005AE: f (q4.c:12)
   by 0x400656: main (q4.c:23)
 Address 0x51f10a4 is 0 bytes after a block of size 20 alloc'd
   at 0x4C2B6CD: malloc (in ...)
   by 0x40060A: main (q4.c:19)

Invalid free() / delete / delete[] / realloc()
   at 0x4C2A82E: free (in ...)
   by 0x400662: main (q4.c:24)
 Address 0x51f1040 is 0 bytes inside a block of size 8 free'd
   at 0x4C2A82E: free (in ...)
   by 0x4005DE: f (q4.c:14)
   by 0x400656: main (q4.c:23)

20 bytes in 1 blocks are definitely lost in loss record 1 of 1
   at 0x4C2B6CD: malloc (in ...)
   by 0x40060A: main (q4.c:19)
```

Identify and fix the four errors in the code. For each error (1) state the line number that the error is on, (2) state the problem with this line (reason it is an error), and (3) state the correction that fixes this error:

1. Error 1
   (a) Line Number: 17
   (b) Problem: Wrong size allocated
   (c) Fix: Change sizeof(s) to sizeof(*s)
2. Error 2
   (a) Line Number: 11
   (b) Problem: Causes ptr− >nums[i] to go out of bounds
   (c) Fix: Change <= to <
3. Error 3
   (a) Line Number: 14
   (b) Problem: Double free with line 24
   (c) Fix: Delete this line
4. Error 4
   (a) Line Number: 24
   (b) Problem: s− >nums needs to be freed before s is freed (memory leak)
   (c) Fix: Insert free(s− >nums)

Question 5: Coding 1 [11 pts]

Write the function `reverseArray` which takes an array of ints (`array`), and the number of items in that array (`n`) and reverses the contents of the array. This function should modify the original array.

```
void reverseArray (int * array, int n) {

  for (int i = 0; i < n/2; i++) {
    int temp = array[i];
    array[i] = array[n-i-1];
    array[n-i-1] = temp;
  }

}
```

Then write a `main` function which (1) makes an array containing 1,2,3,4, and 5. (2) calls `reverseArray` to reverse this array. (3) prints out the elements of the reversed array (one number per line, starting from index 0). (4) returns `EXIT_SUCCESS`

```
int main(void) {

    int testArray[] = {1,2,3,4,5};
    reverseArray(testArray,5);
    for (int i = 0; i < 5; i++) {
      printf("%d\n", testArray[i]);
    }
    return EXIT_SUCCESS;

}
```

Question 6: Coding 2 [11 pts]

Write a program which takes one command line argument (a file name), reads all of the lines of text in the file named by the command line argument, sorts those lines, and then prints them out. You may assume that you have two functions (not pictured) already written which work correctly (you do **NOT** need to write these):

```
void strSort(char ** strs, int n);
char * readStr (FILE * f);
```

The first of these takes an array of strings, and a count of how many strings there are and sorts the strings (re-arranging the pointers in the array). The second of these reads a line from `f`, allocating an appropriate amount of space (with malloc), and returns it. This function returns `NULL` on end of file (You may use `getline` instead if you prefer).

For this problem, we will relax the "no assumptions" restriction by allowing you to assume that (1) a command line argument is provided (2) the file requested exists and is readable (3) malloc and realloc always succeed (4) fclose succceds.

Answer on the next page

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char ** argv) {
  FILE * f = fopen(argv[1],"r");
  char ** strs = NULL;
  int nStrs = 0;
  char * line;
  while ((line = readStr(f)) != NULL) {
    strs = realloc(strs, (nStrs+1) * sizeof(*strs));
    strs[nStrs] = line;
    nStrs++;
  }
  fclose(f);
  strSort(strs, nStrs);
  for (int i = 0; i < nStrs; i++) {
    printf("%s", strs[i]);
    free(strs[i]);
  }
  free(strs);
  return EXIT_SUCCESS;
}
```

Question 7: Recursion [11 pts]

Write a program which takes one command line argument, and determines if it is a palindrome or not. You may write in C or C++, but you must use only recursion for this problem (you may not use `while`, `for`, or `do` in your solution). For this problem, we will relax the "no assumptions" restriction and you may assume that a command line argument is always given. Your program should print either `yes` (if the argument is a palindrom) or `no` (if it is not) and then exit successfully. You may write any functions you want. You may use any standard library functions you want.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int isPalindrome(const char * p, int n) {
  if (n <= 1) {
    return 1;
  }
  if (p[0] != p[n-1]) {
    return 0;
  }
  return isPalindrome (p + 1, n - 2);
}

int main(int argc, char ** argv) {
  if (isPalindrome(argv[1], strlen(argv[1]))) {
    printf("yes\n");
  }
  else {
    printf("no\n");
  }
  return EXIT_SUCCESS;
}
```