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 from 7 question, so pace yourself accordingly.

Questions:

1. Multiple Choice: 8 pts
2. Tracing C Code: 12 pts
3. Algorithmic Basics: 12 pts
4. 4: Dynamic Allocation: 10 pts
5. Coding 1: 11 pts
6. Coding 2: 11 pts
7. Recursion: 11 pts

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. Conceptually, what does the unary `*` operator mean in C?
   (a) Follow the arrow. ←
   (b) Give an arrow pointing at something.
   (c) Allocate memory for something.
   (d) Concatentate two strings.
   (e) None of these

2. Which tool helps you find memory leaks?
   (a) gdb
   (b) gcc
   (c) svn
   (d) valgrind ←
   (e) None of these

3. A recursive function is...
   (a) ...guaranteed to run in at most $O(N^3)$ time.
   (b) ...one that calls itself. ←
   (c) ...one that uses the `ncurses` library.
   (d) ...incompatible with `gdb`.
   (e) None of these

4. When white box testing...
   (a) ...if you have statement coverage, then you are guaranteed to have path coverage too.
   (b) ...you cannot have statement coverage and path coverage at the same time.
   (c) ...if you have decision coverage, then you are guaranteed to need at most one more test case to have path coverage.
   (d) ...you cannot have statement coverage and decision coverage at the same time.
   (e) None of these ←

Question 2 Tracing C Code [12 pts]

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

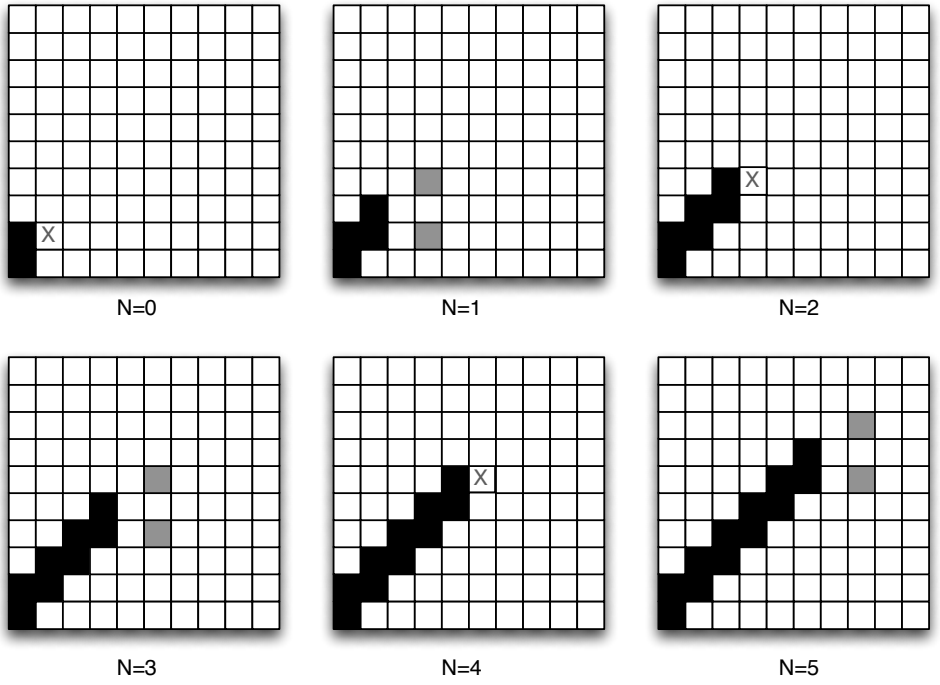
```
#include <stdio.h>
#include <stdlib.h>

int g(int * p) {
  *p = p[1] + 20;
  p++;
  int ans = *p;
  p[0] = 456;
  return ans;
}

int f(int * q, int ** p) {
  *q = 99;
  int n = g(*p);
  printf("**p = %d\n", **p);
  *p = q;
  return n;
}

int main (void) {
  int a = 4;
  int b[] = {6, 9, 88};
  int * p = &b[1];
  int c = f(&a, &p);
  printf("c = %d\n", c);
  if (p == &a) {
    printf("*p aliases a, both = %d\n", *p);
  }
  else {
    printf("*p is %d\n", *p);
    printf("a is %d\n", a);
  }
  for (int i = 0; i < 3; i++) {
    printf("b[%d] = %d\n", i, b[i]);
  }
  return EXIT_SUCCESS;
}
```

Answer:

```
**p = 108
c = 88
*p aliases a, both = 99
b[0] = 6
b[1] = 108
b[2] = 456
```

Question 3 Algorithmic Basics [12 pts]

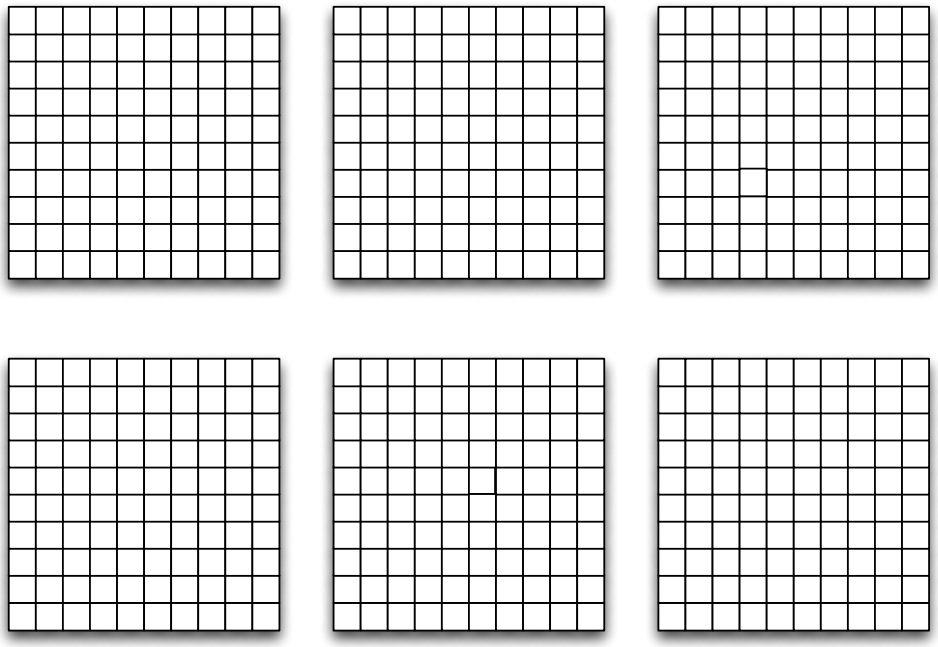
The diagrams shown below are the result of executing an algorithm (parameterized by N) which draws black squares, grey squares, and grey Xes on a 10x10 grid of white boxes.



Determine the algorithm used to create these diagrams, and write the algorithm below in clear-step-by-step English for any N. Note that the next page has blank grids and space for scratch work.

Answer:
Note: coordinates start from (0,0) in the bottom left)
Count from 0 to N (inclusive)
For each number "i" that you count,
Draw a black box at (i,i)
Draw a black box at (i,i+1)
After you finish counting,
If N is even
Draw a box with a grey X at (N+1, N+1)
Otherwise
Draw a grey box at (N+2, N+1)
Draw a grey box at (N+2, N)

This page is for scratch work. It will not be graded.



This page is for scratch work. It will not be graded.

Question 4 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 `copyArray` which takes an array of **pointers to** `info structs` (`array`), and the number of items in that array (`n`) and makes a **DEEP** copy of the array. This function should return the copied array. Hint: you may wish to use `strdup` on this question.

```
struct _info {
  int nStrs;     //number of strings in strs
  char ** strs;  //array of strings
};
typedef struct _info info;

info ** copyArray (info ** array, int n) {
```

```
  info ** ans = malloc(n * sizeof(*ans));
  for (int i = 0; i < n; i++) {
    ans[i] = malloc(sizeof(*ans[i]));
    ans[i]->nStrs = array[i]->nStrs;
    ans[i]->strs = malloc(array[i]->nStrs * sizeof(*ans[i]->strs));
    for (int j = 0; j < ans[i]->nStrs; j++) {
      ans[i]->strs[j] = strdup(array[i]->strs[j]);
    }
  }
  return ans;
```

```
}
```

Then write a `freeArray` function which frees all memory associated with an array of info struct pointers. Note that `freeArray(p,n)` should free all memory allocated by `copyArray` if `p` is the return value of `copyArray` and `n` is the same value as was passed to `copyArray`.

```
void freeArray(info ** p, int n) {
```

```
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < p[i]->nStrs; j++) {
      free(p[i]->strs[j]);
    }
    free(p[i]->strs);
    free(p[i]);
  }
  free(p);
```

```
}
```

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, and prints out the longest line in the file (and **ONLY** the longest line). That is, your program should have one single line of output, which is the longest line in the file. If the input file is empty (does not contain any lines) then your program should print no output at all.

You may find `getline` and `strlen` useful on this problem (though you may use any standard library functions you wish).

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 (so fopen succeeds) (3) malloc and realloc always succeed (4) fclose succeeds.

Answer on the next page

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

int main(int argc, char ** argv) {
  FILE * f = fopen(argv[1],"r");
  char * longestLine = NULL;
  size_t longestLength = 0;
  char * currentLine = NULL;
  size_t lineSize = 0;
  while (getline(&currentLine, &lineSize, f) >= 0) {
    size_t currentLength = strlen(currentLine);
    if (currentLength > longestLength) {
      free(longestLine);
      longestLine = strdup(currentLine);
      longestLength = currentLength;
    }
  }
  fclose(f);
  if (longestLine != NULL) {
    printf("%s", longestLine);
    free(longestLine);
  }
  free(currentLine);
  return EXIT_SUCCESS;
}
```

Question 7 Recursion [11 pts]

Write the recursive function `strXpos` which makes a copy of a string, transposing (flipping) every pair of characters. If the string has odd length, then the last character is *not* transposed with the `'\0'` character (the output string is always the same length as the input string). For example, given an input (`src`) of `"abcde"` the string produced (written into `dest`) would be `"badce"`. As another example, if the input were `1234`, then the function would produce the string `2143`. You may assume that `dest` points to a writeable area with space for each character in `src` when you function is first called.

You may not use any library functions (`strlen`, `strdup`, `malloc`, etc). You **MUST** use only recursion for this problem. You **may not** use iteration (no `for`, `while`, or `do-while` loops).

```
void strXpos ( char * dest, const char * src ) {
```

```
  if (src[0] == '\0') {
    dest[0] = '\0';
  }
  else if (src[1] == '\0') {
    dest[0] = src[0];
    dest[1] = src[1];
  }
  else {
    dest[0] = src[1];
    dest[1] = src[0];
    strXpos(dest + 2, src + 2);
  }
```

```
}
```