Name: _____ **Date: _____**

CS270 Homework Assignment 4 (HW4)

Due Thursday, May 1 at 9:20am, bring to class, no late submissions.
Homework and programming assignments are to be done individually.

.

Goals
To understand C pointers and structs, and functions and the activation record:

- Basic pointer manipulation
- C pointers and functions
- C pointers and arrays
- C pointers and strings
- C pointers and structs
- Static memory allocation
- Dynamic memory allocation
- C pointers and swapping
- C pointers and efficiency
- C pointers to pointers
- Activation records
- Managing activation records, stack pointer, frame pointer, and function calls

Instructions
You may need to write C programs for this assignment, but these do not need to be handed in. A hard copy of your written solution is to be turned in before class on the due date. Try to figure out the output of the programs before running any code, ***and only then*** compare your answer after running the program. This will maximize your learning, and ensure greater chance of success if similar questions appear on the final exam. Each question is worth 10 points.

Question 1 (10 points): Basic C Pointers

```
int i;
float x;
int *pInteger = &i;
float *pFloat = &x;

i = 5678;
*pInteger = 1234;
x = 0.5678f;
*pFloat = 0.1243f;

printf("i = %d, %d, %d\n", i, *(&i), *pInteger);
printf("x = %f, %f, %f\n", x, *(&x), *pFloat);
```

---

a) What is the output of the code shown above?

b) Is there any difference between the address of a variable, and the value of a pointer to that variable?

c) What should the difference in the values of `pInteger` and `pFloat` be on a 32-bit system?

_____ bytes

Question 2 (10 points): C Pointers and Functions

```
void function(int i, int *j, float x, float *y)
{
        i = 5544;
        *j *= 100;
        x = 0.1234f;
        *y /= 10.0;

        printf("%d, %d, %f, %f\n", i, *j, x, *y);
}

int i = 2233;
int j = 1122;
float x = 5.678f;
float y = 2.468f;

printf("%d, %d, %f, %f\n", i, j, x, y);
function(i, &j, x, &y);
printf("%d, %d, %f, %f\n", i, j, x, y);
```

---

a) What is the output of the code shown above?

b) Which parameters can be changed by the function? Which cannot?

c) The function appears to modify the parameters `i` and `x`, but these values never make it out of the function. Why not?

Question 3 (10 points): C Pointers and Arrays

```
int iArray[4] = {11, 22, 33, 44};
int *pInteger = &iArray[0];
printf("%d %d %d %d\n", iArray[0], iArray[1], iArray[2], iArray[3]);

iArray[0] *= 2;
*(pInteger+1) *= 3;
pInteger[2] *= 4;
*(iArray+3) *= 5;

printf("%d %d %d %d\n", iArray[0], iArray[1], iArray[2], iArray[3]);
```

---

a) What is the output of the code shown above?

b) Are the following identical: `pInteger[1]`, `*(pInteger+1)`, `iArray[1]` and `*(iArray+1)`? Why?

Question 4 (10 points): C Pointers and Strings

```
char *str = "hello";
char str1[6] = {'t','h','e','r','e','\0'};

for (unsigned int i=0; i<strlen(str); ++i)
{
        printf("str[%d] = %c(%c)\n", i, str[i], *(str+i));
}

printf("str = %s\n", str);

for (unsigned int j=0; j<strlen(str1); ++j)
{
        printf("str1[%d] = %c(%c)\n", j, str1[j], *(str1+j));
}

printf("str1 = %s\n", str1);
```

---

a) What is the output of the code shown above?

b) Is there a string data type in C? If not, what is used instead?

c) What additional limitation does a string have that a character array does not?

Question 5 (10 points): C Pointers and Structs

```
typedef struct
{
        int i;
        float f;
} simple;

simple s;
simple *p=&s;

s.i = 1234;
s.f = 0.112233f;
printf("s.i = %d, s.f = %f\n", s.i, s.f);

p->i += 2345;
p->f *= 2.0f;
printf("s.i = %d, s.f = %f\n", s.i, s.f);
```

---

a) What is the output of the code shown above?

b) How does the **.** operator differ from the **->** operator with respect to structure access?

c) How many bytes does the *struct* defined above require on a 32-bit system?

Question 6 (10 points): C Pointers and Static Allocation

```
int i = 11;
int j = 12;

float x = 0.123f;
float y = 0.234f;

printf ("Values: %d, %d, %f, %f\n", i, j, x, y);
printf ("Addresses: %p, %p, %p, %p\n", &i, &j, &x, &y);
```

---

a) What is the output of the code shown above? You must run the code to find out.

b) Are local variables pushed onto the stack in forward order (i,j,x,y) or reverse order (y,x,j,i)?
[**Hint**: pushing data onto the stack decreases the stack pointer.]

Question 7 (10 points): C Pointers and Dynamic Allocation

```
int array1[4];
int array2[4];

int *array3 = (int *)malloc(sizeof(int) * 4);
int *array4 = (int *)malloc(sizeof(int) * 4);

printf("Addresses: %p, %p, %p, %p\n", array1,array2,array3,array4);
```

---

a) What is the output of the code shown above? You must run the code to find out.

b) Why are the addresses of array1/array2 so different from array3/array4. Which memory pool is used for each allocation?

Question 8 (10 points): C Pointers and Data Swapping

```
void swap0(int x, int y)
{
        int temp = x;
        x = y;
        y = temp;
        printf("x = %d, y = %d\n", x, y);
}

void swap1(int *x, int *y)
{
        int temp = *x;
        *x = *y;
        *y = temp;
        printf("x = %d, y = %d\n", *x, *y);
}

int i = 1234;
int j = 5678;

printf("i = %d, j = %d\n", i, j);
swap0(i, j);
printf("i = %d, j = %d\n", i, j);
swap1(&i, &j);
printf("i = %d, j = %d\n", i, j);
```

---

a) What is the output of the code shown above?

b) Why does `swap0` fail to swap the values, even though it seems to have worked locally? Why does `swap1` work?

Question 9 (10 points): C Pointers and Efficiency

```
typedef struct
{
        int iArray[32];
        float fArray[32];
} large;

void f1(large l)
{
        printf("sizeof(l) = %d\n", (int)sizeof(l));
}

void f2(large *l)
{
        printf("sizeof(l) = %d\n", (int)sizeof(l));
}

large s;
for (int i=0; i<32; ++i)
{
        s.iArray[i] = i;
        s.fArray[i] = (float) i;
}
f1(s);
f2(&s);
```

---

a) What is the output of the code shown above?

b) How many bytes are required on the stack for parameter storage for `f1()`? `f2()`? Which is more efficient and why?

Question 10 (10 points): C Pointers to Pointers

```
int i = 12345;
int *p = &i;
int **pp = &p;

i = 2345;
printf("i = %d\n", i);

*p = 3467;
printf("i = %d\n", i);

**pp = 4567;
printf("i = %d\n", i);

printf("&i = %p, p = %p, pp = %p, *pp = %p\n", &i, p, pp, *pp);
```

---

a) What is the output of the code shown above? You must run the code to find out.

b) Why do **&i**, **p**, and ***pp** all point at the same address?

c) What address is pointed at by the "pointer to a pointer" **pp**?