Name: _____ SID: _____

CS270 Homework 5

Due Tuesday, April 30 Beginning of the class

Goals

To understand the following aspects of C pointers:

- Basic pointer manipulation
- Functions
- Arrays
- Strings
- Structs
- Static memory allocation
- Dynamic memory allocation
- C pointers and swapping
- C pointers and efficiency
- C pointers to pointers

Instructions

A copy of your written solution is to be turned in 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. You may write C programs for this assignment, but these do not need to be handed in. Each question is worth 10 points.

Late Policy

Late assignments will be accepted up to 24 hours past the due date, with a penalty of 25%.

The Assignment

Question 1 (10 points): Basic C Pointers

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

i = 5555;
*pInteger = 7777;

*pFloat = 3.166;
x = 1.3456f;

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? Yes _____ No _____

c) What would you expect the difference in the addresses of pInteger and pFloat to be?
_____ bytes

Question 2 (10 points): C Pointers and Functions

```
void function(int i, int *p, float x, float *y)
{
        i = 3322;
        *p *= 50;
        x = 2.1234;
        *y /= 21.0;

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

int i = 12;     int j = 4;
float x = 3.468f;    float y = 6.678f
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 enable variables outside the function to be changed by the function? Which do not?

c) The function appears to modify the parameters i and x? Are these values actually modified? Yes _____ No _____

Question 3 (10 points): C Pointers and Arrays

```
int iArray[4] = {1, 3, 5, 7};
int *pInteger = &iArray[0];
printf("%d %d %d %d\n", iArray[0], iArray[1], iArray[2], iArray[3]);

iArray[2] *= 2;
*(pInteger+1) *= 4;
pInteger[1] *= 6;
*(iArray+2) *= 8;

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)?

Yes ____ No _____

Question 4 (10 points): C Pointers and Strings

```
char *str = "hello";
char str1[6] = {'w','o','r','l','d','\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?

Question 5 (10 points): C Pointers and Structs

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

Foo foo;
Foo *p=&foo;

foo.i = 6879;
foo.f = 4.1f;
printf("foo.i = %d, foo.f = %f\n", foo.i, foo.f);

p->i += 16;
p->f *= 9.0f;
printf("foo.i = %d, foo.f = %f\n", foo.i, foo.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 the lab linux systems in the department? _____ (Lab room number _____)

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

```
int array1[4];
int array2[8];

int *array3 = (int *)malloc(sizeof(int) * 10);
int *array4 = (int *)malloc(sizeof(int) * 8);

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.

c) Which memory region is used for each allocation?

array1/array2 _____, array3/array4_____

Question 7 (10 points): C Pointers and Efficiency

```
typedef struct     {
        char cArray[32];
        float fArray[32];
} large;

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

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

large s;
for (int i=0; i<32; ++i)      {
        s.cArray[i] = i+30;
        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()?

_____

Question 8 (15 points): C Pointers to Pointers

```
int i = 321;
int *p = &i;
int **pp = &p;

i = 345;
printf("i = %d\n", i);

*p = 42;
printf("i = %d\n", i);

**pp = 7;
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 is pointed at by the “pointer to a pointer” **pp**?

Question 9 (15 points): Run-time stack and heap visualization

```
typedef struct listnode Node;

struct listnode {
    int value;
    Node* next;
};

void main() {
    Node* head = (Node*)malloc(sizeof(Node));
    head->value = 42;
    head->next = NULL;
    Node* temp = head;
    head = (Node*)malloc(sizeof(Node));
    head->value = 7;
    head->next = temp;

    head->next->next = (Node*)malloc(sizeof(Node));
    head->next->next->value = 27;
    head->next->next->next = NULL;

}
```

Draw Here

Draw what the run-time stack and heap look like after the above code executes. Show all of the parameters, indicate where return addresses are stored, and for any pointers including the current stack and frame pointer use arrows to show where they are pointing.