CS 1713
Introduction to Computer Programming II
Final Solutions

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td></td>
</tr>
<tr>
<td>Question 2</td>
<td></td>
</tr>
<tr>
<td>Question 3</td>
<td></td>
</tr>
<tr>
<td>Question 4</td>
<td></td>
</tr>
<tr>
<td>Question 5</td>
<td></td>
</tr>
<tr>
<td>Question 6</td>
<td></td>
</tr>
<tr>
<td>Question 7</td>
<td></td>
</tr>
<tr>
<td>Question 8</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
</tr>
</tbody>
</table>

NAME: ______________________________________

**Instructions**
1. Do All of the 8 problems
3. You have 120 minutes for the exam
4. Show all your work
5. Do not separate exam papers

<table>
<thead>
<tr>
<th>Easy</th>
<th>Difficulty Level</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
1. (20 pts) What is the output of the following program? Show what is printed when mystery() functions are called. Show all your work.

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
    int info;
    struct node *next;
};
typedef struct node node;

int mystery (node *head) {
    int temp = 0;
    while (head != NULL) {
        temp = temp + head -> info;
        head = head -> next;
    }
    return(temp);
}

int main() { 
    node *ptr, *ptr2, *ptr3;

    ptr = (node*)malloc(sizeof(node));
    ptr -> info = 4;
    ptr2 = (node*)malloc(sizeof(node));
    ptr -> next = ptr2;
    ptr2 -> next = NULL;
    ptr -> next -> info = 1;

    printf("Output1: %d\n", mystery(ptr2));
    printf("Output2: %d\n", mystery(ptr));

    ptr3 = (node*)malloc(sizeof(node));
    ptr3 -> next = ptr;
    ptr3 -> info = 2;
    printf("Output3: %d\n", mystery(ptr3));
    ptr2 -> info = 8;
    printf("Output4: %d\n", mystery(ptr3));
    ptr -> info = 16;
    printf("Output4: %d\n", mystery(ptr));
}
```
Solution:

Output1: 1
Output2: 5
Output3: 7
Output4: 14
Output4: 24
2. (20 pts) Write a function to find a superpeak element in an array. A superpeak element is an element that is greater than the sum of its neighbors. Given an input array where $num[i] \neq num[i+1]$, find a superpeak element and return its index. The array may contain multiple superpeaks, in that case return the index to any one of the superpeaks. The array may not contain any superpeaks, in that case return -1. You may assume that $num[-1] = 0$ and $num[n] = 0$. So, $num[0]$ and $num[n-1]$ could be superpeaks as well. For example, in array [1, 2, 4, 1], 4 is a superpeak element and your function should return its index 2. The array [1 2, 3, 2] does not have any superpeaks and you should return -1.

In below function prototype, $n$ is the size of the array

Solution

```c
int superpeak(int num[], int n) {
    int i;
    int index = -1;

    if (num[0]>num[1])
        return (0);

    if (num[n-1]>num[n-2])
        return(n-1);

    for (i=1; i<n; i++)
        if (num[i]>num[i-1]+num[i+1])
        {
            index = i;
            return(i);
        }
    return(-1);
}
```
3. (20 pts) Consider the following declaration of array of pointers

```
int *data[10];
```

Write a program fragment (for, if, assignment statements) to find out how many different memory locations they point to. For example, if data[3] and data[5] point to same location and all the other point to different locations your output should be

```
data array points to 9 different memory locations
```

Hint: Count it incrementally. If a new index points to a different location, increase the count. If a new index points to a previously counted location, move on to the next index.

**Solution:**

```
count = 0;
for (i=0; i<10; i++)
{
    found = 1;
    for (j=i+1; j<10; j++)
        if (data[i]==data[j])
            found = 0;
    if (found == 1)
        count++;
}
printf("data array points to %d different locations\n",count);
```
4. (20 pts) What is the output of the following program. Show all your work.

```
#include <stdio.h>

struct pyramid
{
    float length;
    float width;
    float height;
};

typedef struct pyramid pyramid;

int main()
{
    pyramid p1={3,4,5};
    pyramid p2={1,2,3};
    pyramid p3={4,6,7};
    pyramid *pptr1=&p2;
    pyramid *pptr2=&p1;

    printf("1 %f %f %f\n",p3.width, p3.length, p3.height);
    printf("2 %f %f %f\n",pptr2->width, pptr2->length,pptr2->height);

    pptr1 = &p3;
    printf("3 %f %f %f\n",(*pptr1).width, (*pptr1).length,(*pptr1).height);

    pptr2->width = 5;
    pptr1 = pptr2;

    printf("4 %f %f %f\n",pptr1->width, pptr1->length,pptr1->height);

    p1 = p2;
    printf("5 %f %f %f\n",p1.width, p1.length, p1.height);
}
```

Solution:

```
1 6.000000 4.000000 7.000000
2 4.000000 3.000000 5.000000
3 6.000000 4.000000 7.000000
4 5.000000 3.000000 5.000000
5 2.000000 1.000000 3.000000
```
5. (20 pts) Write a function `scramble` which scrambles the characters of a string. You can do this by repeatedly selecting two random characters in the string and swapping them. Function prototype is given below.

Solution:

```c
void scramble(char *str)
{
    int i,x,y,len;
    char temp;

    len=strlen(str);

    for (i=0; i<len; i++)
    {
        x = rand() % len;
        y = rand() % len;
        temp = *(str+x);
        *(str+x) = *(str+y);
        *(str+y) = temp;
    }
}
```
6. (20 pts) Write a function `alldistinct()` to check if all the elements of a linked list are different from each other. Return 1 if all the elements are distinct and 0 if some element appears multiple times. Node declaration of the linked list is given below.

```c
struct node
{
    int info;
    struct node *next;
};
typedef struct node node;
```

The function should return 1 for the following list since all the elements are distinct.

```
head → 5 → 6 → 12 → Null
```

**Solution:**

```c
int alldistinct(node *head)
{
    node *ptr = head;
    node *ptr2 = head;

    while (ptr != NULL)
    {
        ptr2 = ptr->next;
        while (ptr2 != NULL)
        {
            if (ptr->info == ptr2->info)
                return (0);
            ptr2 = ptr2->next;
        }
        ptr = ptr->next;
    }
    return(1);
}
```
7. (20 pts) What is the output of the following program? Show all your work.

```c
#include <stdio.h>

int mystery(int a)
{
    if (a == 1)
        return a;
    else
        return (1+mystery(a/2));
}

int main()
{
    int x;

    x = 3;
    printf("Out1 = %d\n",mystery(x));

    x = 11;
    printf("Out1 = %d\n",mystery(x));

    x = 24;
    printf("Out1 = %d\n",mystery(x));

    x = 7;
    printf("Out1 = %d\n",mystery(x));

    x = 9;
    printf("Out1 = %d\n",mystery(x));

}
```

Solution:

Out1 = 2
Out1 = 4
Out1 = 5
Out1 = 3
Out1 = 4
8. (20 pts) Write a complete program to read a file of rectangles represented as length and width fields and find and print the rectangle with the largest area.

Sample file named rectangles.txt given below. Each line has the length and width of a rectangle.

8.5 9.2
2.3 10.7
3.4 7.8
14.5 17.9

Output for above file should be

Maximum area is 259.549988 for rectangle with length 14.500000 and width 17.900000

Solution:

```c
#include <stdio.h>

int main()
{
    FILE *ifile;
    float length, width;
    float maxarea=0, maxlen, maxwidth;

    ifile = fopen("rectangles.txt","r");
    while (feof(ifile) <= 0)
    {
        fscanf(ifile,"%f %f", &length, &width);
        if (length * width > maxarea)
        {
            maxarea = length * width;
            maxlen = length;
            maxwidth = width;
        }
    }

    printf("Maximum area is %.6f for rectangle with length %.6f and width %.6f",
        maxarea, maxlen, maxwidth);

    fclose(ifile);
    return(0);
}
```