CS 210 **October 13, 2016**
Day 13 In Class

Review programs

Sample Programs

1. The following sequence uses a function called MinMax to find the minimum and maximum of an array. Write the function.

```
{int a[] = {1, 5, 2, 6, 8, 23, 45};
 int min, max;
 int SIZE = 7;
 MinMax(a, &min, &max, SIZE);
 printf("The min is %d, the max is %d", min, max);
}
```

```
void MinMax(int a[], int *min, int *max, int n)
      {int i;
       *min = a[0];
       *max = a[0];
       for(i=1;i<SIZE;i++)
         {if(*min > a[i])
            *min = a[i];
          if(*max < a[i])
            *max = a[i];
         }
      }
```

2. Write a method which will accept an `int` array called *a* and will return the largest difference between any two adjacent elements. For example, the following sequence will print 4 since the largest difference is 7 – 3 = 4.

```
int[] a = {1, 1, 3, 7, 8, 9};
printf("%d\n", FindMaxDiff(a, 6));
```

```
int FindMaxDiff(int a[], int SIZE)
  {int i, diff;
   diff = a[1] - a[0];
   for(i=1;i<SIZE-1;i++)
      {if(diff < a[i+1] - a[i])
         diff = a[i+1] - a[i];
      }
   return diff;
  }
```

3. Show how to create a 1-dimensional array of doubles and fill it with the number x where x is the product of PI and the array index. The array should have 3,482 elements.

```
double PI = 3.141592653589793
int d[] = new double[3482];
int i;
for(i=0;i<3482;i++)
  d[i] = i*Math.PI;
```

4. Write a method which receives a 1-dimensional array named *a[]* and returns the *index* of the first nonzero element in the array. If there are no nonzero elements in the array your method should return -1.

```
int MyMethod(int a[] int SIZE)
  {int i;
   i = 0;
   while(i < SIZE && a[i] != 0)
     {i++;
     }
   if(i == SIZE)
     return -1;
   return i;
  }
```

5. Create an array of 100 random ints which range in value from 0 to 1000. Calculate and print the minimum and maximum values stored in the array.

```
#define SIZE 100
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i, min, max;
    int data[SIZE];
    srand(23);
    for(i=0;i < SIZE;i++)
       data[i] = rand() % 1001;
    min = data[0];
    max = data[0];
    for(i=1;i < SIZE;i++)
    {
        if(data[i] < min)
           min = data[i];
        if(data[i] > max)
           max = data[i];
    }
    printf("Min is %d, max is %d\n", min, max);
}
Min is 15, max is 984
Press any key to continue . . .
```

6. Create two arrays of doubles called x and y which have the following data: x = {0, 2.2, 3.4, -9.5, 10.3, 7.5, 2.1}, y = {0, -2.3, -8.5, -2.1, 4.4, 6.5, 0.9}. Treat values in the x and y vectors which have the same index as representing a point on a two dimensional grid. Use the distance formula to find the two points which are the farthest apart. $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

```
#include<stdio.h>
#include<math.h>
double GetDistance(double x[],double y[], int i,int j);
int main()
{
    double x[] = {0, 2.2, 3.4, -9.5, 10.3, 7.5, 2.1};
    double y[] = {0, -2.3, -8.5, -2.1, 4.4, 6.5, 0.9};
    double dist, distMax;
    int i, j, iMax, jMax;
    iMax = 0;jMax = 1;
    distMax = GetDistance(x, y, iMax, jMax);
    for(i=0;i<6;i++)
    {
        for(j=i+1;j<7;j++)
        {
             dist = GetDistance(x, y, i, j);
             if(dist > distMax)
             {
                 iMax = i;jMax = j;
                 distMax = dist;
             }
        }
    }
    printf("Max distance is %f\n", distMax);
    printf("Points are %5.2f, %5.2f and %5.2f, %5.2f\n", x[iMax], y[iMax], x[jMax],
y[jMax]);
}
//
double GetDistance(double x[],double y[], int i,int j)
{
   return sqrt(pow((x[i]-x[j]), 2) + pow((y[i]-y[j]), 2));
}
```

```
Max distance is 20.839626
Points are -9.50, -2.10 and 10.30,  4.40
Press any key to continue . . .
```

7. The Sieve of Eratosthenes is an ancient simple method of finding prime numbers. To use the sieve to find all of the prime number between 1 and say 1000, we make an array of all integers from 1 to 1000. Begin with the number 2 and delete all multiples of 2. Next take 3 and delete all of the higher multiples of 3. Continue eliminating multiples until you get to the square root of 1000. The numbers left in the array will be the primes from 1 to 1000. Write a program to do the Sieve of Eratosthenes and print the resulting prime numbers.

```c
#include<stdio.h>
#include<math.h>
int main()
{
    int i, j, sieve[1001];
    int root;
    for(i=1;i<1001;i++)
       sieve[i] = i;
    root = (int)sqrt(1000);
    for(i=2;i<=root;i++)
    {
        j = 2*i;
        while(j <= 1000)
            {sieve[j] = 0;
             j += i;
            }
    }
    for(i=1;i<1001;i++)
    {
        if(sieve[i] != 0)
           printf("%d, ", sieve[i]);
    }

}
```

```
1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101,
103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311,
313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431,
433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557,
563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661,
673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937,
941, 947, 953, 967, 971, 977, 983, 991, 997, Press any key to continue . . .
```

8. Fill in the memory map below for the following program:

```
#include<stdio.h>
int Confused(int y,  int *z);
int main()
  {int x = 9, y = 14, z = 2;
   printf("%d %d %d\n", x, y, z);
   z = Confused(x, &y);
   printf("%d %d %d\n", x, y, z);
  }
//
int Confused(int y,  int *z)
  {int a = 13, b;
   printf("%d %d %d\n", a, y, *z);
   b = 8;
   *z = 12;
   return a + b + *z;
  }
```

Confused	Main	Data
	x	**9**
z	**y**	~~**14**~~ **12**
	z	~~**2**~~ **33**
y		**9**
a		**13**
b		**8**

Printed Results

9 14 2

13 9 14

9 12 33

9. The main program below create and array named *a* with 1000 random ints and a second array *aRev* of the same size but empty. Write a function called Reverse which accepts the two arrays as parameters and returns with *aRev* having all of the values of *a* but in reverse order.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void Reverse(int a[], int aRev[]);
int main()
{
    int i, a[50], aRev[50];
    srand((unsigned) time(NULL));
    for(i=0;i<50;i++)
       a[i] = rand();
    Reverse(a, aRev);
    for(i=0;i<50;i++)
       printf("%d, %d\n", a[i], aRev[i]);
    return 0;
}
```

```
void Reverse(int a[],int aRev[])
{
   int i;

   for(i=0;i<50;i++)
      aRev[i] = a[49 - i];
}
```

10. A file named `Number.txt` contains an unknown number of ints with one int per line. Write a program which opens the file and prints all of the ints which are greater than zero to a second file named `Positive.txt`. Your program should print the number of positive numbers to the console.

```
#include<stdio.h>
int main()
{
      int err;
     FILE *inp;  //Declare pointers to in
      FILE *outp; //  and out files
      int dataIn, status;
     int cnt = 0;
     //Open the Number file for input
      err = fopen_s(&inp, "Number.txt", "r");
     //Open Positive for output
     err = fopen_s(&outp, "Positive.txt", "w");

     status = fscanf_s(inp, "%d", &dataIn);
      //read Number and print Positive if > 0
     while(status == 1)
      {
           if(dataIn > 0)
           {fprintf(outp," %d\n", dataIn);
            cnt++;
           }
         status = fscanf_s(inp, "%d", &dataIn);
      }
     fclose(inp);
     fclose(outp);
     printf("Positive count = %d\n", cnt);
}
```