Name: Section: Rollno:

ESC101: Fundamentals of Computing(Lab Exam 1)

11th/12th September, 2014

Total Number of Pages: 17 Total Points 140

Instructions

1. Read these instructions carefully.
2. DO NOT use the `math.h` library for any of the questions.
3. Range of `int` is -2147483648 to 2147483647.
4. Placeholder for `long int` is `%ld` and for `long long int` is `%lld`
5. Apart from the given test cases, you should check the correctness of your program on your own test cases as well.

Question	Points	Score
1	10	
2	10	
3	20	
4	30	
5	10	
6	10	
7	20	
8	30	
Total:	140	

Name: Section: Rollno:

Question 1. (10 points) In a set of data, **mode** is the value that appears most often in the set.

Write a program that finds the mode in a sorted interger array. The elements of the array are in non-decreasing order.

In case there are **more than one** mode for the set, the last such value is to be printed.

INPUT:

- First line contains the size of the array, **N**. ($N \leq 10^4$)
- Next line contains N integers in non-decreasing order. Each element will be in the range $[-10^5 \ldots 10^5]$.

OUTPUT:
Output will be a single line containing the mode. In case there are **more than one** mode for the array, the last such value is to be printed.

EXAMPLES:

Example 1	INPUT	OUTPUT
	5 1 3 3 3 7	3
Example 2	INPUT	OUTPUT
	10 -1 2 2 3 4 5 5 6 7 8	5
Example 3	INPUT	OUTPUT
	3 3 3 4	3
Example 4	INPUT	OUTPUT
	1 1	1

Solution:

```
#include <stdio.h>
#define MAX_SZ 10000
int main()
{
    int size, i;
    int array[MAX_SZ];

    scanf("%d", &size);
    for(i=0; i<size; i++)
    {
        scanf("%d", &array[i]);
    }
    int number = array[0];
    int mode = number;
    int count = 1;
    int countMode = 1;

    for(i=1; i<size; i++)
    {
        if (array[i] == number)
```

```
21          { // count occurrences of the current number
22              count++;
23          }
24          else
25          { // now this is a different number
26              if (count >= countMode)
27              {
28                  countMode = count; // mode is the biggest
                        ocurrences
29                  mode = number;
30              }
31              count = 1; // reset count for the new number
32              number = array[i];
33          }
34      }
35      // check for the last number
36      if (count >= countMode)
37      {
38          countMode = count; // mode is the biggest ocurrences
39          mode = number;
40      }
41
42      printf("%d\n", mode);
43      return 0;
44  }
```

Name: Section: Rollno:

Question 2. (10 points) In this problem, you have to read a positive integer from the user and check if it is a **palindrome** or not. A number is a palindrome if the reverse of the number is equal to the number itself.

INPUT:

- One line of input: the positive integer.
  NOTE: integer will **NOT** have leading zeroes (i.e. input of type 0123 is not allowed).

OUTPUT:
Output will be **YES** if the number is a palindrome, **NO** if it is not a palindrome.

EXAMPLES:

Example 1	**Example 2**	**Example 3**	**Example 4**
INPUT	INPUT	INPUT	INPUT
34	131	1234321	1111111111
OUTPUT	OUTPUT	OUTPUT	OUTPUT
NO	YES	YES	YES

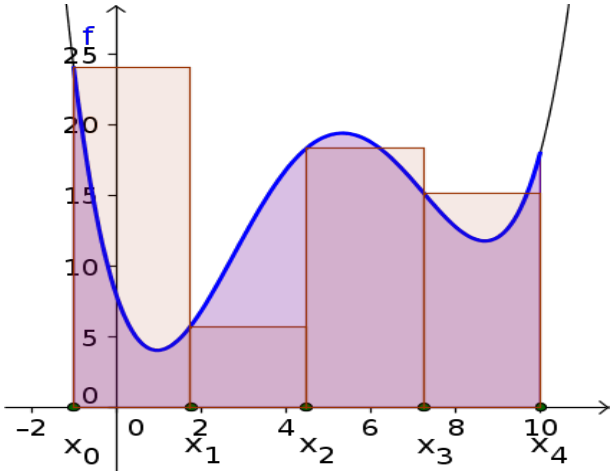
Solution:

```
#include<stdio.h>
int isPalindrome(int x) {
    int y,z;
    z = x;
    y = 0;
    if(z < 0)
        return 0;
    while(z != 0){
        y = y*10+z%10;
        z = z/10;
    }

    if(x < 0)
        return y==-x;
    return y==x;

}
int main(){
        int x;
        scanf("%d",&x);
        printf("%s\n",isPalindrome(x) ? "YES" : "NO");
    return 0;
}
```

Name: Section: Rollno:

Question 3. (20 points) Given a function $f(x)$ where $f(x) \geq 0$ over an interval $a \leq x \leq b$, we can compute the area of the region that is under the graph of $f(x)$ and above the interval $[a, b]$ on the x-axis as follows:

We divide the interval $[a, b]$ into n subintervals of length Δx (where Δx must be $(b - a)/n$). We label the endpoints of the subintervals by $x_0, x_1, \ldots, x_n$, so that the leftmost point is $a = x_0$ and the rightmost point is $b = x_n$. The picture shows the case with four subintervals.



To estimate the area under the graph of f, we just need to add up the areas of all the rectangles. Using summation notation, the sum of the areas of all n rectangles for $i = 0, 1, \ldots, n - 1$ is

$$\sum_{i=0}^{n-1} f(x_i) \times \Delta x$$

In this problem, you have to write a program to compute the area under the function $f(x)$ for a given interval $[a, b]$. The function is defined as:

$$f(x) = \begin{cases} 1 & x \leq -1 \\ x^2 & -1 < x < 1 \\ x^3 & x \geq 1 \end{cases}$$

INPUT: There will be three inputs: start of interval a, end of interval b and number of subintervals n. The type of a and b is **double**, while n will be an integer. Also, you can assume $b \geq a$.

OUTPUT: The output will be a **double** value, the desired area of the function. The output should be printed upto 4 places of decimal.

EXAMPLES:

Example 1	**Example 2**	**Example 3**
INPUT	INPUT	INPUT
5.0 5.0 5	-5.0 -4.0 10	-1.0 1.0 4
OUTPUT	OUTPUT	OUTPUT
0.0000	1.0000	0.7500

NOTE: For floating point comparison, use a small value (**const double epsilon = 0.000001**) as error margin.

Solution:

```c
#include<stdio.h>
const double epsilon = 0.000001;

double f_x(double x) {
    if(x<=-1)
        return 1;
    else if(x>-1 && x<=1)
        return x*x;
    else
        return x*x*x;
}

double mod(double x){
    if(x<0)
        return -1*x;
    else
        return x;
}

int main(){
    int n; double a, b, i;
    double del_x; //width of strip
    double area = 0;
    scanf("%lf %lf %d", &a, &b, &n);

    if (n==0) {
        printf("Error\n");
        return -1;
    }

    del_x = (b-a)/n;
    for(i=a; b-i > epsilon; i += del_x) {
        area += mod(f_x(i)*del_x);
    }
    printf("%.4lf\n",area);

    return 0;
}
```

Name: Section: Rollno:

Question 4. (30 points) In some application, we need to manipulate very large integers (few hundred digits long). To handle very large numbers, we can use **array of int**, where each element represents **one digit** of the number.

Write a program to add very large numbers stored in arrays. Each number can be up to 500 digits long. Store the result in an array (note that result can be 501 digit long!).

The program should print the result of addition of two as output.

INPUT:

- First two lines contain the size of the numbers.
- Next two line of the input will contain the numbers themselves (first number followed by the second number).

OUTPUT:
Output will be a single line containing the result of addition of two numbers.

EXAMPLES:

Example 1	**Example 2**	**Example 3**	**Example 4**
INPUT	INPUT	INPUT	INPUT
2 2 34 78	3 2 134 78	3 4 100 1100	16 16 1111111111111111 9222222222222222
OUTPUT	OUTPUT	OUTPUT	OUTPUT
112	212	1200	10333333333333333

Solution:

```
#include <stdio.h>

#define MAX_SZ 500

int main()
{
        int a[MAX_SZ];
        int b[MAX_SZ];
        int c[MAX_SZ+1];
        int a_size, b_size, i,j,k, carry;
        char ch;
        scanf("%d", &a_size);
        getchar(); // to eat new line
        scanf("%d", &b_size);
        getchar(); // to eat new line

    // numbers are really stored as digits in the array
    // Example: 1234 is stored as [1, 2, 3, 4]
    //           561 is stored as [5, 6, 1]
    // note that a[a_size-1] is aligned for addition with b[b_size
        -1]
```

```
21
22          for(i=0;i<a_size;i++)
23          {
24                  scanf("%c", &ch);
25                  a[i]=ch-'0';
26          }
27          getchar(); // to eat new line
28          for(i=0;i<b_size;i++){
29                  scanf("%c", &ch);
30                  b[i]=ch-'0';
31          }
32          getchar(); // to eat new line
33
34          i=a_size-1, j=b_size-1;
35          k=0,carry=0;
36
37      // digit a[i] is aligned for addition with b[j]
38      // The sum is getting stored in reverse!!, so [1, 2, 3] + [5,
            6]
39      // will give us  c = [9, 7, 1]
40
41          while(i>=0 && j>=0)
42          {
43                  c[k]=(a[i]+b[j]+carry)%10;
44                  carry=(a[i]+b[j]+carry)/10;
45                  i--;j--;k++;
46          }
47          if(a_size>b_size)
48          {
49                  while(i>=0)
50                  {
51                          c[k]=(a[i]+carry)%10;
52                          carry=(a[i]+carry)/10;
53                          k++;
54                          i--;
55                  }
56          }
57          else if(a_size<b_size)
58          {
59                  while(j>=0)
60                  {
61                          c[k]=(b[j]+carry)%10;
62                          carry=(b[j]+carry)/10;
63                          k++;
64                          j--;
65                  }
66          }
67          else
68          {
69                  if(carry>0)
70                  {
71                          c[k]=carry;
72                          k++;
73                  }
74          }
75
```

```
76      // k will point to next (empty) digit in c, so fix it.
77          k--;
78
79      // now print the digits of c in reverse order
80      // -> this is really the correct order for sum.
81          while(k>=0)
82          {
83                  printf("%d", c[k]);
84                  k--;
85          }
86          printf("\n");
87          return 0;
88  }
```

Name: Section: Rollno:

Question 5. (10 points) In this problem, you have to read a positive integer from the user and check if it is a **perfect** number or not. A number is a perfect if it is equal to the sum of its proper positive divisors.

For example, Consider number 6. Its proper divisors are 1, 2, and 3. Sum of the divisors is 1+2+3=6. Hence, 6 is perfect number.

INPUT:

- One line of input: the positive integer.

OUTPUT:
Output will be **YES** if the number is perfect, **NO** if it is not perfect.

EXAMPLES:

Example 1	**Example 2**	**Example 3**	**Example 4**
INPUT	INPUT	INPUT	INPUT
6	497	496	27
OUTPUT	OUTPUT	OUTPUT	OUTPUT
YES	NO	YES	NO

Solution:

```
#include<stdio.h>
int main(){
    int num,i=1,sum=0;
    scanf("%d",&num);
    while(i<num){
        if(num%i==0)
            sum=sum+i;
        i++;
    }
    printf("%s\n", sum==num ? "YES" : "NO");
    return 0;
}
```

Name: Section: Rollno:

Question 6. (10 points) You are given an array of integer elements and an integer **S**. No element in the array is repeated. Write a program to find all **pairs** of integers in the array that sum up to S.

INPUT:

- First line contains the size of the array, **N**. ($N \leq 10^4$)
- Next line contains N integers. Each element will be in the range $[-10^5 \ldots 10^5]$.
- Next line contains **S**, the desired sum. S will be in the range $[-10^5 \ldots 10^5]$.

OUTPUT:

You have to print all pairs, one per line, that sum up to the given integer S. The pairs are to be printed in parantheses, separated by comma, for example: **(a,b)**.

The order of elements in a pair must be same as their order in the array.

EXAMPLES:

Example 1	INPUT	OUTPUT
	12 4 0 1 3 2 6 9 21 10 11 5 7 8	(1,7) (3,5) (2,6)
Example 2	INPUT	OUTPUT
	8 -1 2 3 4 5 6 7 8 4	(-1,5)
Example 3	INPUT	OUTPUT
	5 -1 3 2 -3 1 0	(-1,1) (3, -3)
Example 4	INPUT	OUTPUT
	1 1 1	**no output**

Solution:

```
#include <stdio.h>
#define MAX_SZ 10000
int main()
{
    int array[MAX_SZ];
    int size, s, i, j;
    scanf("%d", &size);
    for(i=0; i<size; i++) {
        scanf("%d", &array[i]);
    }
    scanf("%d", &s);

    for(i=0; i<size; i++) {
        for(j=i+1; j<size; j++) {
            if(array[i]+array[j] == s) {
                printf("(%d,%d)\n", array[i], array[j]);
```

```
            }
        }
    }
    return 0;
}
```

Name: Section: Rollno:

Question 7. (20 points) Given two functions $f(x)$ and $g(x)$, we compute the area of the region that is between the graphs of $f(x)$ and $g(x)$ in the interval $[a, b]$ as follows:

We divide the interval $[a, b]$ into n subintervals of length Δx (where Δx must be $(b - a)/n$). We label the endpoints of the subintervals by $x_0, x_1, \ldots, x_n$, so that the leftmost point is $a = x_0$ and the rightmost point is $b = x_n$.

To estimate the area between the graphs of f and g, we just need to add up the areas of all the *rectangular* subintervels. The sum of the areas of all n rectangles for $i = 0, 1, \ldots, n - 1$ is

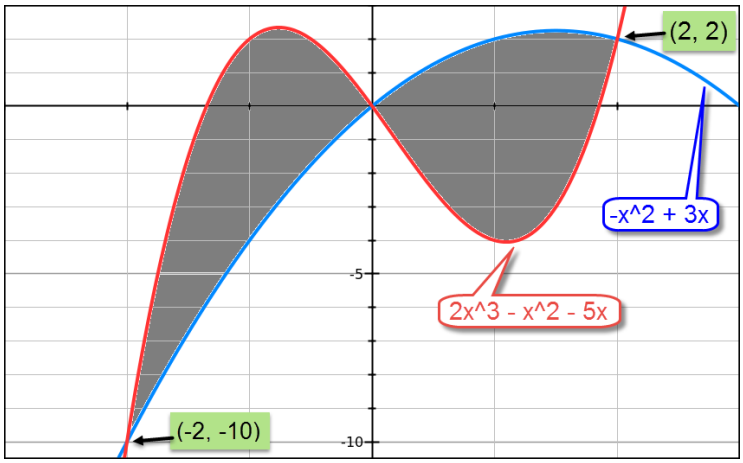
$$\sum_{i=0}^{n-1} |f(x_i) - g(x_i)| \times \Delta x$$

where $|y|$ denotes the absolute value of y.

In this problem, you have to write a program to compute the area between the functions $f(x)$ and $g(x)$ for a given interval $[a, b]$. The functions are defined as:

$$\begin{aligned} f(x) &= -x^2 + 3x \\ g(x) &= 2x^3 - x^2 - 5x \end{aligned}$$

A diagrammatic representation of these curves is below, where the shaded region corresponds to the area between the curves in the interval [-2, 2].



INPUT: There will be three inputs: start of interval a, end of interval b and number of subintervals n. The type of a and b is **double**, while n will be an integer. Also, you can assume $b \geq a$.

OUTPUT: The output will be a **double** value, the desired area. The output should be printed upto 4 places of decimal.

EXAMPLES:

Example 1	**Example 2**	**Example 3**
INPUT	INPUT	INPUT
1.0 1.0 5	-2.0 2.0 10	-1.0 1.0 400
OUTPUT	OUTPUT	OUTPUT
0.0000	15.3600	7.0000

NOTE: For floating point comparison, use a small value (**const double epsilon = 0.000001**) as error margin.

Solution:

```c
#include<stdio.h>
const double epsilon = 0.000001;

double f_x(double x) {
    return -1*x*x + 3*x;
}

double g_x(double x) {
    return 2*x*x*x - x*x - 5*x;
}

double mod(double x){
    if(x<0)
        return -1*x;
    else
        return x;
}

int main(){
    int n; double a, b, i;
    double del_x; //width of strip
    double area = 0;
    scanf("%lf %lf %d", &a, &b, &n);

    if (n==0) {
        printf("Error\n");
        return -1;
    }

    del_x = (b-a)/n;
    for(i=a; b-i > epsilon; i+=del_x) {
        area += mod(f_x(i)-g_x(i))*del_x;
    }
    printf("%.4lf\n",area);

    return 0;
}
```

Name: Section: Rollno:

Question 8. (30 points) A queue is a first in, first out (fifo) data structure. Queue has an inherent notion of *head* and *tail*. Elements can be read and added only from the head of the queue and removed only from the tail of the queue. That is, **not all the elements can be accessed directly** like in the case of arrays.

A queue is characterised by three operations - enqueue, dequeue and is_empty.

a) **enqueue** – this operation adds an item to the head of the queue.

b) **dequeue** – this operation removes an item from the tail of the queue. Dequeue from an empty queue is ignored.

c) **is_empty** – this tells if the queue is empty or not.

An integer queue can be implemented using an **array of int**.

Write a C program which implements a queue of **positive integers**. Implement the **enqueue**, **dequeue** and **is_empty** operations. Note the following:

- For dequeue, if the queue is empty, nothing happens.
- Also implement a **print** function to display the queue elements *from tail to head*. The print function prints an **X** to mark end of queue (This is specially important in case of empty queue).

You may like to define a global integer array as your queue and a front and rear index to keep track of head and tail of queue. Prototypes (declarations) of the operations are as follows:

```
1 void enqueue(int item);
2 void dequeue();
3 void print();
4 int is_empty(); // return 1 if queue is empty, 0 if non-empty
```

INPUT:
First line of input will contain integer **N**, the number of operations to be performed on queue (**assume** $N \leq 10^4$). The next N lines will contain one of the following three operations each:

```
1 e <value>        // e.g. e 5 : enqueue 5 to list
2 d                // e.g. d : dequeue element from list
3 p                // e.g. p : prints elements of list, at the end add X
```

OUTPUT:
The output is the result of **p** (print) operations specified on input. Each **p** operation will result in one line of output, showing the content of the queue at that point.

EXAMPLES:

Example 1 INPUT	**Example 2** INPUT	**Example 3** INPUT	**Example 4** INPUT
5 e 1 p e 4 d p	8 e 1 e 2 e 3 e 4 p d e 9 p	5 d e 1 p d p	1 p
OUTPUT	OUTPUT	OUTPUT	OUTPUT
1 X 4 X	1 2 3 4 X 2 3 4 9 X	1 X X	X

Solution:

```c
#include <stdio.h>

#define MAX_SZ 10000

int queue_array[MAX_SZ];
int back = -1;
int front = -1;

void enqueue(int item)
{
        if(back==MAX_SZ)
        {
                // printf("Queue overflow\n");
                return;
        }
        if(front==-1)
                front=0;
        back++;
        queue_array[back]=item;
        return;
}

void dequeue()
{
        if(front==-1 || front>back)
        {
                // printf("Queue underflow\n");
                return;
        }
        front=front+1;
        return;
}

void print()
{
        if(front==-1 || front > back )
        {
                printf("X\n");
                return;
        }
        int i;
        for(i=front;i<=back;i++)
                printf("%d ", queue_array[i]);
        printf("X\n");
        return;
}

int is_empty()
{
        if(front==-1 || front > back )
        {
                return 1;
        }
    return 0;
}
```

```
56
57
58 int main()
59 {
60         int N=0,i,item;
61         char choice;
62         scanf("%d", &N);
63         getchar();
64         for(i=1;i<=N;i++)
65         {
66                 scanf("%c", &choice);
67                 switch(choice)
68                 {
69                         case 'e': scanf("%d", &item);
70                                 enqueue(item);
71                                 break;
72                         case 'd': dequeue();
73                                 break;
74                         case 'p': print();
75                                 break;
76                 }
77                 getchar();
78         }
79     return 0;
80 }
```