Name: Section: Rollno:

ESC101A: Fundamentals of Computing (Mid Semester Exam)

21st February, 2014

Total Number of Pages: 15

Total Points 170

Instructions

1. Read these instructions carefully.
2. Write you name, section and roll number on all the pages of the answer book.
3. Write the answers cleanly in the space provided. There is space left on the back of the answer book for rough work.
4. Do not exchange question books or change the seat after obtaining question paper.
5. Using pens (blue/black ink) and not pencils. Do not use red pens for answering.
6. Even if no answers are written, the answer book has to be returned back with name and roll number written.

Question	Points	Score
1	15	
2	15	
3	20	
4	20	
5	20	
6	25	
7	30	
8	25	
Total:	170	

Helpful hints

1. The questions are *not* arranged according to the increasing order of difficulty. Do a quick first round where you answer the easy ones and leave the difficult ones for the subsequent rounds.
2. All blanks may NOT carry equal marks.

Question 1. (15 points) A *proper divisor* of a positive integer n is an integer d such that $1 \leq d < n$, and d divides n. An *abundant number* is a positive integer for which the sum of its proper divisors is strictly greater than the number itself.

For example, the proper divisors of 28 are 14, 7, 4, 2 and 1, which sum to 28. Hence 28 is not an abundant number. On the other hand, 12 is an abundant number since the proper divisors of 12 are 6, 4, 3, 2 and 1, which sum to 16.

The program given below is a partially filled program that given an integer `n` as input, prints **Abundant** if `n` is an abundant number and displays **Not Abundant** otherwise.

Fill in the missing blanks and complete the program. Note that there are a total of **8 blanks** that you need to fill.

```
#include<stdio.h>

int is_abundant(int);

int main() {
   int n, ans;
   printf("Enter the number: ");

   scanf("%d", ________________);
   ans = is_abundant(n);
   if(ans) {
      printf("________________\n");
   }
   else {
      printf("________________\n");
   }
   return 0;
}

int is_abundant(int n) {
   int i;

   int ________________;

   for(i=1; ________________; ________________) {
      if(n%i == 0) {

         sum = ________________;
      }
   }
   if(sum > ________________)
     return 1;
   else
     return 0;
}
```

Solution:

```
#include<stdio.h>

int is_abundant(int);

int main() {
  int n, ans;
   printf("Enter the number: ");
   scanf("%d", &n);
   ans = is_abundant(n);
   if(ans) {
      printf("Abundant\n");
   }
   else {
      printf("Not Abundant\n");
   }
   return 0;
}

int is_abundant(int n) {
   int i;
   int sum = 0;
   for(i=1; 2*i <= n; i++) {
      if(n%i == 0) {
         sum = sum + i;
      }
   }
   if(sum > n)
     return 1;
   else
     return 0;
}
```

Question 2. (15 points) Consider the following program.

```
#include <stdio.h>

int main(){
  int ar[101];
  int n, i, j, out;
  scanf("%d", &n);
  for(i=0;i<n;i++)
    scanf("%d", &ar[i]);

  for(i=0;i<n;i++){
    out = 1;
    for(j=0;j<n;j++)
      if(ar[j]>ar[i])
        out++;
    printf("%d\n", out);
  }
  return 0;
}
```

What is the output of the above program for the following two set of inputs?

(a) (5 points) **Input 1**

```
10
9 8 5 4 1 2 3 10 7 0
```

Answer

```
2
3
5
6
9
8
7
1
4
10
```

(b) (10 points) **Input 2**

```
20
8 7 79 5 -9 758 78 123 80 805 78
-8954 789 75 5 -894 784 725 123 321
```

Answer

```
14
15
10
16
18
4
11
7
9
1
11
20
2
13
16
19
3
5
7
6
```

Name: Section: Rollno:

Question 3. (20 points) Consider the following program.

```
 1 #include<stdio.h>
 2
 3 int fun1(int x) {
 4    if(x < 0) return -x;
 5    else return x;
 6 }
 7
 8 int fun2(int n) {
 9   int val = 0, p = 1;
10    while(n>0) {
11       val = val + fun1((n%10) - p);
12       p++;
13       n = n/10;
14    }
15    return val;
16 }
17
18 int main() {
19   int i, arr[]={38, 59, 999, 13, 5124, 853, 925, 1111111, 74, 198};
20   int x, y=0, z;
21
22   for(i=0; i<10; i++) {
23     z = fun2(arr[i]);
24     if(z > y) {
25       y = z;
26       x = arr[i];
27     }
28     printf ("%d %d\n", z, x);
29   }
30   printf("%d\n", x);
31   return 0;
32 }
```

(a) What does the above program display when executed?

Answer

```
8 38
11 59
21 999
3 999
6 999
10 999
10 999
21 999
8 999
16 999
999
```

(b) What would be the output of the program, if in `Line 24` if we change the condition of the `if` statement to `(z >= y)`?

Answer

```
8 38
11 59
21 999
3 999
6 999
10 999
10 999
21 1111111
8 1111111
16 1111111
1111111
```

Question 4. (a) (12 points) Consider the following variables and the values assigned to them.

```
int x=6, y=4;
float z=0.5;
char ch = 'A';
```

What would be the value of the following expressions based on the above values

Expression	**Value**
`2*x` (an example)	`12`
`x/y/z`	`2.000000`
`x/(y/z)`	`0.750000`
`(float) x/y/z`	`3.000000`
`x%y/z`	`4.000000`
`ch + y`	`E` (as a character)
`ch + (int)z + x/y`	`B` (as a character)

(b) (8 points) Consider the following code segment. Assume that all variables have been declared.

```
j=0;
for (i=k; i<=64; i++){
  i = i*i;
  i--;
  j++;
}
printf("%d\n%d\n",i,j);
```

What is the output of the above code segment for the following values of `k`?

`k = 2`	`k = -3`
256 3	81 2

Name: Section: Rollno:

Question 5. (20 points) A *balanced cell* in a 2-dimensional array `ar[]` having `n` rows and `m` columns, is a cell having index `(i,j)` such that `ar[i][j]` is the maximum over all elements in row `i` and minimum over all elements in column `j`.

The program given below is a partially filled program that takes as input a 2-dimensional integer array `ar` and displays the number of balanced cells in the array.

More formally, the program first reads the values of `n` and then `m` as input from the user. It then reads the entries of the array `ar[]`, one row at a time from left to right, starting from the topmost row. Finally, the program computes and then prints the number of balanced cells in the array `ar`.

Fill in the missing blanks and complete the program. Note that there are a total of **11 blanks** that you need to fill.

```
#include <stdio.h>
#define MAX 100
int main(){
  int ar[MAX][MAX], n, m, i, j, k;

  int row_flag, col_flag, ans = ________________;

  scanf("%d", ________________);

  scanf("%d", ________________);

  for(i=0 ; i<n ; i++)
    for(j=0;j<m;j++)

      scanf("________________", &ar[i][j]);

  for(i=0 ; i<n ; i++){
    for(j=0 ; j<m ; j++){

      ________________ = 1;

      ________________ = 1;

      for(k=0 ; ________________ ; k++)

        if( ________________ > ar[i][j] )
          row_flag = 0;

      for(k=0 ; k<n ; k++)

        if(________________ < ar[i][j])
          col_flag = 0;

      if(row_flag && col_flag)

        ________________;
    }
  }
  printf("%d\n", ________________);
  return 0;
}
```

Solution:

```
#include <stdio.h>
#define MAX 100

int main(){
  int ar[MAX][MAX], n, m, i, j, k;
  int row_flag, col_flag, ans = 0;

  scanf("%d", &n);
  scanf("%d", &m);

  for(i=0 ; i<n ; i++)
    for(j=0;j<m;j++)
      scanf("%d", &ar[i][j]);

  for(i=0 ; i<n ; i++){
    for(j=0 ; j<m ; j++){

      row_flag = 1;
      col_flag = 1;
      for(k=0 ; k<m ; k++)
        if(ar[i][k] > ar[i][j])
          row_flag = 0;

      for(k=0 ; k<n ; k++)
        if(ar[k][j] < ar[i][j])
          col_flag = 0;

      if(row_flag && col_flag)
        ans++;
    }
  }
  printf("%d\n", ans);
  return 0;
}
```

Question 6. (25 points) We want to write a program that takes two numbers as input and computes their product. The first number is extremely large and can contain up to 100 digits. Therefore it will not fit even in the datatype `long long int`. The second number is of type `int`. To solve this problem we will design two functions (apart from the `main` function) whose description are given below.

Name	**Arguments**	**Return Type**	**Description**
`get_number`	`int ar[]`	`int`	Reads and stores the large number in this array - one digit in every index of the array, with the left most digit in index 0 and so on. Returns the number of digits in the number.
`display_mult`	`int b[], int len, int a`	`void`	Array `b` contains the large number, `len` is the number of digits in the large number and `a` is the smaller number. It computes and displays the product of the two numbers.

The program is partially filled. Complete the program based on the description given above by filling the missing blanks. Note that there are a total of **15 blanks** that you need to fill.

```
#include <stdio.h>
#define MAX_LENGTH 101

// reads a number as a sequence of characters, terminated by \n
// and returns the number of digits in the number.

int get_number(int ar[]){
  char c;
  int len = ________________;

  scanf("%c", ________________);
  while(c!='\n'){

    ar[len] = ________________;
    len++;
    scanf("%c", ________________);
  }
  return ________________;
}


```

```

// takes a large number in the array 'b', its length 'len' and
// a small number 'a', and displays the product of a and b

void display_mult(int b[], int len, int a){
  int i, carry = 0, temp,  prod[MAX_LENGTH];

  for(i=_______________; _______________ ; _______________){
    temp = b[i]*a + carry;

    prod[i] = _______________;

    carry = _______________;
  }
  // displaying the product

  if (carry > 0)
    printf("%d", carry);
  for(i=0;i<len;i++)
    printf("%d", prod[i]);
  printf("\n");
}

int main(){
  int len, num2, num1[MAX_LENGTH];

  len = get_number(_______________);

  scanf("%d", _______________);

  display_mult(_______________, _______________, _______________);
  return 0;
}
```

Solution:

```
#include <stdio.h>
#define MAX_LENGTH 101
// reads a number as a sequence of characters, terminated by \n and
   returns the number of digits in the number.
int get_number(int ar[]){
  char c;
  int len = 0;

  scanf("%c", &c);
  while(c!='\n'){
    ar[len] = c - '0';
    len++;
    scanf("%c", &c);
  }
  return len;
}

void display_mult(int b[], int len, int a){
  int i, carry = 0, temp,  prod[MAX_LENGTH];

  for(i=len-1; i>=0; i--){
    temp = b[i]*a + carry;
    prod[i] = temp%10;
    carry = temp/10;
  }
  if (carry > 0)
    printf("%d", carry);
  for(i=0;i<len;i++)
    printf("%d", prod[i]);
  printf("\n");
}

int main(){
  int len, num2, num1[MAX_LENGTH];
  len = get_number(num1);
  scanf("%d", &num2);
  display_mult(num1, len, num2);
  return 0;
}
```

Question 7. Consider the following program.

```
#include <stdio.h>
int fun(int[], int, int);

int fun(int a[], int m, int s){
  if (a[m] < s) return 1;
  else if (a[m] > s) return -1;
  else return 0;
}

int main()
{
  int array[100],i, a=0, b, m, n, k, ans, z=0;

  scanf("%d",&n);
  b = n - 1;
  for (i = 0; i < n ; i++ )
    scanf("%d",&array[i]);
  scanf("%d",&k);

  while(a <= b){
    z++;
    printf("%d %d\n",a,b);
    m = (a + b)/2;
    ans = fun(array, m, k);
    if (ans == 1)
      a = m + 1;
    else if (ans == -1)
      b = m - 1;
    else{
      printf("%d %d %d\n", k, m+1, z);
      return 0;
    }
  }
  printf("%d %d %d\n", k, -1, z);
  return 0;
}
```

What is the output of the above program for the following set of inputs (see next page)?

(a) (9 points) **Input 1**

```
8
-1 2 5 6 7 10 50 55
0
```

Output for Input 1

```
0 7
0 2
0 0
0 -1 3
```

(b) (10 points) **Input 2**

```
20
-10 -8 -7 -5 -2 0 5 6 7 10 12 15 16 17 20 22 24 26 27 30
22
```

Output for Input 2

```
0 19
10 19
15 19
15 16
22 16 4
```

(c) (11 points) **Input 3**

```
20
76 89 10 18 20 1 -12 3 -33 87 100 0 21 27 98 12 988 1000 -32 21
21
```

Output for Input 3

```
0 19
0 8
5 8
7 8
8 8
21 -1 5
```

Name: Section: Rollno:

Question 8. (25 points) You are given an integer array `ar[]`, whose each index is filled with either `0` or `1`. Now this array is repeated `rep` times to create a sequence (say `A`).

For example, if `ar[] = {0,1,0}` and `rep = 2`, you will consider the sequence `A = (0,1,0,0,1,0)`.

You start from the first integer of the sequence `A` and move till the end of `A`. Whenever, you land on an integer `0`, you change the `0`'s to `1`'s and all the `1`'s to `0`'s from the next position onwards till the end of the sequence.

The program given below is a partially filled program that takes as input an integer `len` (the length of the array), followed by `len` number of `0`'s and `1`'s (the array `ar[]`) and `rep` (the number of times `ar[]` is repeated to make the sequence `A`). It then prints the number of elements in the sequence `A` that were `1` when you landed on them. Consider the following examples.

`len`	`ar[]`	`rep`	sequence `A`	Value printed
3	{1,0,1}	2	(1,0,1,1,0,1)	2
4	{0,1,1,1}	3	(0,1,1,1,0,1,1,1,0,1,1,1)	6

Fill in the missing blanks and complete the program. Note that there are a total of **10 blanks** that you need to fill.

```
#include <stdio.h>

int main() {
  int ar[100], i, rep, len, start = 0, end, pos, ans = 0, flips = 0;

  scanf("_______________", &len);
  for (i=0; i<len; i++)

    scanf("%d", _______________);

  scanf("%d", _______________);

  end = ____________________;

  while( start <= end ){

    pos = ____________________;

    if( ar[pos] == _______________ ){
      if(!flips){
        flips = !flips;
      }else
        ans++;
    }
    else{
      if(____________________){
        ans++;
      }else
        flips = ____________________;
    }
    ____________________;
  }
  printf("%d\n", _______________);
  return 0;
}
```

Solution:

```
#include <stdio.h>

int main() {
  int ar[100], i, rep, len, start = 0, end, pos, ans = 0, flips = 0;

  scanf("%d", &len);
  for (i=0; i<len; i++)
    scanf("%d", &ar[i]);

  scanf("%d", &rep);
  end = len*rep - 1;

  while(start <= end){
    pos = start%len;
    if(ar[pos] == 0){
      if(!flips){
        flips = !flips;
      }else
        ans++;
    }
    else{
      if(!flips){
        ans++;
      }else
        flips = !flips;
    }
    start++;
  }
  printf("%d\n", ans);
  return 0;
}
```