Indian Institute of Technology Kharagpur
Programming and Data Structures (CS10001)
Autumn 2017-18: Mid-Semester Examination

Time: 2 Hours **Full Marks: 60**

INSTRUCTIONS

1. Answer ALL questions
2. Please write the answers either within the boxes provided or on the blank lines to be filled up. Any answer written elsewhere will not be evaluated.
3. You may use the last two blank pages for your rough works.

Q.1. Answer the following questions as directed.

a) What will get displayed when the following program is executed? [1]

```
#include <stdio.h>
int main() {
  int x = 2, y = 17, result = 5;
  result -= x/5 * 13 * y/3 * x;
  printf("result=%d\n", result);
  return 0;
}
```

result=5

b) What will get displayed when the following program is executed? [1]

```
#include <stdio.h>
int main() {
  int x = -5, y = 10;
  if (x > y) x = 1;
  else if (y < 0)  x = x * (-1);
  else x = 2 * x;
  printf("x=%d\n", x);
  return 0;
}
```

x=-10

c) What is the 8-bit two's complement representation of the decimal number −37? [2]

11011011

d) What will get displayed when the following program is executed? [1]

```
#include <stdio.h>
int main() {
  char a = 'a';
  while ((a > 'a') && (a <= 'c'))  a++;
  printf("%c\n", a);
  return 0;
}
```

a

e) What will get displayed when the following program is executed? [2]

```
#include <stdio.h>
int main() {
  int sum = 1, index = 9;
  do {
        index = index - 1;
        sum = 2 * sum;
   } while (index > 9);
  printf("sum=%d, index=%d\n", sum, index);
  return 0;
}
```

sum=2, index=8

f) What value will the following function return when called as ***recur*(3)**? [2]

```
int recur(int data) {
   if (data > 2)
        return (recur(data - 1) - recur(data - 2));
   else return 1;
}
```

0

g) What value will the following function return when called as ***g*(1024)**? [1]

```
int g(int n) {
     if (n < 2) return n;
     return g(n/2);
}
```

1

h) What is the binary number corresponding to the hexadecimal number **C5.75?** [2]

11000101.01110101

i) Consider the program segment given below to read a letter from a..z and A..Z from the keyboard and convert it to uppercase if not already so. It is assumed that the user will only input a character from a..z and A..Z. Fill up the missing line with a single C expression so that the variable ***ch*** will contain the character in uppercase. Do not use any library functions. [2]

```
char ch;
ch = getchar();

ch = (ch>='A')&&(ch<='Z')?ch:'A'+ch-'a';
```

j) The following program segment is supposed to check whether the values stored by three integer variables ***a***, ***b***, and ***c*** are in ascending order. However, it contains an error. Encircle the part of the program that contains the error and write only that part corrected. [1]

```
if (a < b < c)
   printf("Numbers in ascending order \n");
else printf("Not in ascending order\n");
```

((a<b) && (b<c))

Q.2. Answer the following questions as directed.

a) What will get displayed when the following program is executed? [2]

```
#include <stdio.h>
int main() {
  int i;
  for (i = 1; i = -1; i++)
     if (i < 5) break;
  printf("%d\n", i);
  return 0;
}
```

-1

b) What will get displayed when the following program is executed? [2]

```
#include <stdio.h>
void increment(int i) {
   i++;
}
int main() {
  int i = 0, j = 0;
  while (i++ < 10)  increment(j);
  printf("i=%d, j=%d\n", i, j);
  return 0;
}
```

i=11, j=0

c) What will get displayed when the following program is executed? [2]

```
#include <stdio.h>
int main() {
  float  j = 1.0, i = 2.0;
  int n = 0;
  while (i/j > 0.05) {
      j = j + j;
      n++;
  }
  printf("n=%d, j=%f\n", n, j);
  return 0;
}
```

n=6, j=64.000000

d) What will get displayed when the following function is called as ***f*(2, 8)**? [2]

```
int f(int x, int y) {
    int sum = 0;
    y--;
    if (x == 0) return 0;
    else {
          printf("%d : ", x);
          sum = y + f(x – 1, y);
          printf ("%d : ", sum);
     }
     return sum;
}
```

2 : 1 : 6 : 13 :

e) What will get displayed when the following program is executed? [2]

```
#include <stdio.h>
int main() {
  int sum = 0, i = 3;
  while (i < 100) {
     sum = sum + i;
     i = i + 3;
  }
  printf("sum=%d, i=%d\n", sum, i);
  return 0;
}
```

sum=1683, i=102

Q.3.

a) A number is said to be *perfect* if it is the sum of all its factors (except itself). For example, 6 has factors 1, 2, 3 and 1+2+3 = 6, hence it is perfect. Also, 28 = 1+2+4+7+14 is perfect. In the following function ***checkPerfect*** fill up the missing lines so that it returns 1 if ***n*** is a perfect number and 0 if ***n*** is not a perfect number. [2 + 2]

```
int checkPerfect(int n) {
   int i, sum = 0;
   for (i = 1; i < n; i++) {

       if (n % i == 0 )

            sum += i;
   }
    return (sum == n );
```

b) The following function ***strEqual*** takes two strings ***S*1** and ***S*2** as parameters. Fill up the missing lines in the function so that it returns 1 if the two strings are the same, 0 otherwise.

[1 + 2 + 2]

```
int strEqual(char S1[], char S2[])
{
    int i = 0;

    /* Go on until the end of either of the strings */

    while (s1[i] != '\0' && s2[i] != '\0')

    {
        if (S1[i] != S2[i])  return 0;
        i++;
    }
    if (s1[i] == '\0'  && s2[i] =='\0')
       return 1;
    else
       return 0;
}
```

Q.4.

a) The following **recursive** function ***find_power*** should return x^n when called as ***find_power(x,n),*** ***n*** being a non-negative integer. Fill up the missing lines in the function so that it returns x^n**.** [1 + 1 + 2]

```
float find_power(float x, int n) {
    if (n ==  0 )
        return 1;
    else
        return x*find_power(x,n-1);
}
```

b) Fill up the missing lines in the following program so that it will display the sum of the elements of the array ***A*** when executed. [2 + 2]

```
#include <stdio.h>
int main() {
    int i, n, k = 0, A[10], lim;
    printf("Enter number of elements  ");
    scanf("%d", &n);
    printf("Enter the elements  ");
    for (i = 0; i < n; i++)
          scanf("%d", &A[i]);
    for (i = 0, lim = n/2; i < lim; i++) {
         /* Accumulate Sum */
         k = k + A[i]+A[n-i-1];
    }
    if (lim <(n-lim))   /* if middle element left out */
        k = k + A[i];
    printf("%d\n", k);
    return 0;
}
```

Q.5.

a) The following program is supposed to insert a new integer value ***x*** into an already sorted (in ascending order) array ***A*** containing ***n*** distinct integers. You can assume that ***x*** does not already exist in ***A***, and there is space available to insert ***x*** in ***A***. For example, assume that ***n*** is 10, and ***A*** has the elements 10, 20, 30, 40, 60, 70, 80, 90, 100, 110, and ***x*** is 56. After insertion of ***x***, the array would become 10, 20, 30, 40, 56, 60, 70, 80, 90, 100, 110, and ***n*** would be 11. Fill up the missing lines in the program so that the program inserts ***x*** in the sorted array ***A***.

[2 + 2 + 2 + 2]

```
#include <stdio.h>
int main(){
  int x, i = 0, j, n, A[100];
  scanf("%d%d", &n, &x);
  for (j = 0; j < n; j++) scanf("%d", &A[j]);

  while (x > A[i] && i < n) i++; /* find position after which to insert */

  for (j=n; j>= i+1; j--) /* make space for inserting x */

        A[j] = A[j-1];

  n++;

  A[i] = x; /* insert  the element at the required place */

  for (i = 0; i < n; i++)  printf("%d  ", A[i]);
  return 0;
```

b) The following **recursive** function ***reverse*** takes as parameters an integer array ***A*** and two other integers ***leftIndex*** and ***rightIndex*** which are indices of ***A***. After the function returns, only the part of the array from ***leftIndex*** to ***rightIndex*** (including both) should be reversed if ***leftIndex*** < ***rightIndex*****.** For example, if the elements of the array ***A*** are 1, 2, 3, 4, 5, 6, 7, and the function call ***reverse*****(*A*,2,6)** is made, then on return, the array ***A*** will contain 1, 2, 7, 6, 5, 4, 3 (i.e., ***A***[0] and *A*[1] remain unchanged, and ***A***[2] to ***A***[6] get reversed). Fill up the missing lines in the function so that it reverses the part of the array ***A*** between ***leftIndex*** and ***rightIndex***.

[1 + 3]

```
void reverse(int A[], int leftIndex, int rightIndex) {
    int temp;
    if (leftIndex < rightIndex) {
        temp = A[leftIndex];
        A[leftIndex] = A[rightIndex];
        A[rightIndex] = temp;
        reverse(A, leftIndex+1, rightIndex-1);
    }
}
```

Q.6.

The function ***closest*** given below takes as parameters an integer array ***A***, the number of elements ***n*** in ***A***, and an integer ***val***. Assume that all integers in the array ***A*** are distinct and ***A*** is already sorted in ascending order. The function returns the index of the element in ***A*** with minimum absolute difference with ***val*** (i.e., it returns the index ***i*** such that |***A***[***i***] – ***val***| is minimum). If more than one element has the same minimum absolute difference with ***val***, then it returns the smallest index. For example, if ***A*** contains the elements 10, 13, 15, 19, 110 and ***val*** is 18, the function returns 3, which is the index of 19 (as |19 – 18| is the minimum). However, if ***val*** is 14, it returns 1 (as both |15 – 14| and |13 – 14| are the minimum, 13 occurs at index 1 and 15 at index 2, and 1 is the smaller index). Fill up the missing lines in the function so that it does the above. [2 + 2 + 2]

```
int closest(int A[], int n, int val) {
    int index, i;
    if (val < A[0]) /* smaller than the smallest element */
       index = 0;
    else if (val > A[n-1]) /* larger than the largest element */
           index = n - 1;
    else {
           /* find the elements closest to val */

           for (i = 0; A[i]<val ; i++) ;

           if ((A[i]-val)<(val-A[i-1]))

                /* if current  element closest */
                 index = i;
           else index= i-1 ; /* set index to the closest element */

        }
    return index;
}
```

--- The End---