INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Signature of the Invigilator

Please fill up carefully the boxes provided below

EXAMINATION (Mid Semester)										SEMESTER (Autumn)			
Roll Number										Section		Name	
Subject Number	C	S	1	1	0	0	1	Subject Name		Programming and Data Structures			
Name of the Department / Center of the Student													

Instructions and Guidelines to Students Appearing in the Examination

1. Ensure that you have occupied the seat as per the examination schedule.
2. Ensure that you do not have a mobile phone or a similar gadget with you even in switched off mode. Note that loose papers, notes, books should not be in your possession, even if those are irrelevant to the paper you are writing.
3. Data book, codes or any other materials are allowed only under the instruction of the paper-setter.
4. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items is not permitted.
5. Additional sheets, graph papers and relevant tables will be provided on request.
6. Write on both sides of the answer script and do not tear off any page. Report to the invigilator if the answer script has torn page(s).
7. Show the identity card whenever asked for by the invigilator. It is your responsibility to ensure that your attendance is recorded by the invigilator.
8. You may leave the examination hall for wash room or for drinking water, but not before one hour after the commencement of the examination. Record your absence from the examination hall in the register provided. Smoking and consumption of any kind of beverages is not allowed inside the examination hall.
9. After the completion of the examination, do not leave the seat until the invigilator collects the answer script.
10. During the examination, either inside the examination hall or outside the examination hall, gathering information from any kind of sources or any such attempts, exchange or helping in exchange of information with others or any such attempts will be treated as adopting ‘unfair means’. Do not adopt ‘unfair means’ and do not indulge in unseemly behavior as well.

Violation of any of the instructions may lead to disciplinary action of varied nature.

To be filled in by the examiner

Question	1	2	3	4	5	6	7	8	9	10	**Total**
Marks Obtained											

Marks obtained (in words)	Signature of Examiner	Signature of Scrutineer

CS11001 Programming and Data Structures, Autumn 2014–2015

Mid Semester Examination

Date: 22–September–2014 Time: 9:00–11:00am Maximum Marks: 60

Write your answers in the question paper itself. Be neat and tidy.
Answer all questions. Not all blanks carry equal marks.

1. Supply short answers to the following questions. **(2 × 7)**

(a) What is the output of the following program?

```
main()
{
    int s = 0;
    while (s++ < 10) {
        if ((s < 4) && (s < 9)) continue;
        printf("%d,", s);
    }
}
```

4,5,6,7,8,9,10,

(b) What is the output of the following program?

```
main()
{
    int a[5] = { 5, 1, 15, 20, 25 };
    int i, j, k;
    i = ++a[1] ;
    j = a[1]++ ;
    k = a[i++] ;
    printf ("%d,%d,%d", i, j, k);
}
```

3,2,15

(c) What is printed by the following program?

```
main()
{
    int i, j, x, A[3] = {1,-1,1};
    for (x=1; x<6; x++) {
        j = 0;
        for (i=0; i<3; i++) j = j*x + A[i];
        printf("%d,", j);
    }
}
```

1,3,7,13,21,

(d) What is printed by the following program?

```
int A (int m, int n)
{
    if (!m) return n + 1;
    if (!n) return A(m - 1, 1);
    return A(m - 1, A(m, n - 1));
}

main()
{
    printf("A(1,2) = %d", A(1,2));
}
```

A(1,2) = 4

(e) What is the output of the following program?

```
void f0 (unsigned int);
void f1 (unsigned int);

void f0 (unsigned int a)
{
    if (!a) printf("No"); else f1(a-1);
}

void f1 (unsigned int a)
{
    if (!a) printf("Yes"); else f0(a-1);
}

main ()
{
    f1(6);
}
```

Yes

(f) Mathematically express the return value $f(n)$ of the function supplied below. Your mathematical expression for $f(n)$ must hold for all integers $n \geqslant 0$.

```
unsigned int f ( unsigned int n )
{
    if (n == 0) return 0;
    return 2 * f(n - 1) + 1;
}
```

$f(n) = 2^n - 1$

(g) The Fibonacci sequence is defined as

$$\begin{aligned} F(0) &= 0, \\ F(1) &= 1, \\ F(n) &= F(n-2) + F(n-1) \text{ for } n \geqslant 2. \end{aligned}$$

The following function defines another sequence $G(n)$ for $n = 0, 1, 2, 3, \ldots$. How is the sequence $G(n)$ mathematically related to the Fibonacci sequence $F(n)$? (Express $G(n)$ in terms of $F(n)$. Your expression should hold for all integers $n \geqslant 0$.)

```
int G ( unsigned int n )
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return G(n-2) - G(n-1);
}
```

$G(n) = (-1)^{n+1} F(n)$

2. Fibonacci numbers can be used to compute the *golden ratio* $\rho = \frac{1+\sqrt{5}}{2}$. Recall that the Fibonacci numbers are defined as $F(0) = 0$, $F(1) = 1$, and $F(n) = F(n-1) + F(n-2)$ for $n \geqslant 2$. A property of the Fibonacci numbers is that $\lim_{n\to\infty} F(n)/F(n-1) = \rho$. In practice, the sequence $F(n)/F(n-1)$ rapidly converges to ρ. In this exercise, you use this idea to compute a good approximation of ρ using Fibonacci numbers. Your approximation should be within an error bound of 10^{-10}. You are not allowed to use any math or stdlib library calls (like **sqrt**, **pow**, or **fabs**). Complete the following program.

The actual value of ρ is unavailable to you, so it is not possible to compute the error exactly. This problem is bypassed in the following way. For $n \geqslant 2$, let $\rho_n = F(n)/F(n-1)$ be the n-th approximation of ρ. It is known that $\rho_2 < \rho_4 < \rho_6 < \cdots < \rho < \cdots < \rho_7 < \rho_5 < \rho_3$. Moreover, each approximation is closer to ρ than the previous approximation. Therefore, in each iteration, we have one of the following two possibilities:

We stop as soon as we get $|\rho_n - \rho_{n-1}| < 10^{-10}$. This condition will automatically imply $|\rho_n - \rho| < 10^{-10}$.

At the beginning of each iteration of the **while** loop below, **F** should store $F(n)$, **F_prev** the value $F(n-1)$, **rho** the ratio $F(n)/F(n-1)$, **rho_prev** the ratio $F(n-1)/F(n-2)$, and **error** the absolute difference of **rho** and **rho_prev**. Do not change these meanings of the variables, that is, the initialization step and the loop body must ensure that precisely these values are stored at the beginning of each iteration. **(12)**

```
/* Define an approximation error of 10^-10 */

#define APPROX_ERROR ____1e-10____

main ()
{
   int F, F_prev, n; /* Variables for Fibonacci numbers */
   double rho, rho_prev, error; /* Variables to store approximations and error */

   /* Initialize the Fibonacci-number variables for n = 3 */

   n = 3; F = ____2____ ; F_prev = ____1____ ;
   /* Initialize the approximation variables to ρ3 and ρ2 */

   rho = ____2____ ; rho_prev = ____1____ ;
   /* Initialize the error to |ρ2 − ρ3| */
   error = rho - rho_prev;
   /* Repeat so long as the error is more than the allowed approximation error */

   while ( ____error >= APPROX_ERROR____ ) {
      /* Increment n and calculate the next (a closer) approximation of ρ */
      ++n;
      /* Update the Fibonacci numbers */

      F = ____F + F_prev____ ; F_prev = ____F - F_prev____ ;
      /* Update the approximation ratios */

      rho_prev = ____rho____ ; rho = ____(double)F / (double)F_prev____ ;
      /* Compute the new error */

      ____error = rho - rho_prev; if (error < 0) error = -error;____
   }
   /* Print the approximate value and which approximation it is */

   printf("rho_%d = %lf = %d/%d\n", ____n, rho, F, F_prev____ );
}
```

3. You write a program which runs a loop for the user to enter a sequence of zeros and ones. Entering an integer other than $0, 1$ terminates the loop. Your task is to count how many times the user enters the sequence 1010. For example, if the user enters $1,0,0,1,0,1,0,1,0,0,1,0,1,0,1,1,2$, then the sequence 1010 occurs thrice:

$$1\ 0\ 0\ \overline{1\ 0\ 1\ 0}\ 1\ 0\ 0\ \overline{1\ 0\ 1\ 0}\ 1\ 1$$

In order to solve this problem, you maintain a progress indicator *preflen* to remember the length of the longest prefix of 1010, that you have read in the last *preflen* iterations (including the current one). For example, at the beginning of the loop, or after reading two consecutive zeros, *preflen* is zero (indicating that we cannot be somewhere inside a match of 1010 at this point of time). On the other hand, if you read two consecutive ones, *preflen* should be one, because if the user enters $0, 1, 0$ in the next three iterations, then you locate one occurrence of the pattern 1010. For the user input in the example above, *preflen* changes as follows. The *preflen* value *after* processing each scanned integer is shown below.

Input	1	0	0	1	0	1	0	1	0	0	1	0	1	0	1	1
preflen	1	2	0	1	2	3	2	3	2	0	1	2	3	2	3	1

Here, *preflen* starts with the value 0 before any user input. After the first input (1), match occurs with the first symbol of 1010, so *preflen* becomes 1. After the second input (0), the length of the matched prefix increases to two. The third input (0) destroys the partial match, so *preflen* reduces to zero.

The following program implements this algorithm. Fill out the missing details. In each **`case`** of the **`switch`** statements, change *preflen* appropriately, and increment **`count`** if a complete match is located. **(12)**

```
main ()
{
    int a, count = 0, preflen = 0;

    while (1) {
        scanf("%d", &a);
        if (a == 0) {
            switch (preflen) {

                case 0:  ____________________________ break; ____________________________

                case 1:  ____________________ preflen = 2; break; ____________________

                case 2:  ____________________ preflen = 0; break; ____________________

                case 3:  ____________ ++count; preflen = 2; break; ____________

            }
        } else if (a == 1) {
            switch (preflen) {

                case 0:  ____________________ preflen = 1; break; ____________________

                case 1:  ____________________________ break; ____________________________

                case 2:  ____________________ preflen = 3; break; ____________________

                case 3:  ____________________ preflen = 1; break; ____________________

            }

        } else ____________ break; ____________
    }

    printf("The number of occurrences of 1010 is %d\n", count);
}
```

4. An array A stores a permutation of the integers $0, 1, 2, 3, \ldots, n-1$. Choose an index $i_1 \in \{0, 1, 2, \ldots, n-1\}$. We have $A[i_1] = i_2, A[i_2] = i_3, \ldots, A[i_k] = i_1$ for some $k \geqslant 1$. We say that $(i_1, i_2, i_3, \ldots, i_k)$ is a *cycle* (of length k) of the permutation. A permutation can always be written as a collection of pairwise disjoint cycles. Your task is to complete the following function which prints all the cycles of the permutation stored in A.

As an example, let $n = 10$, and consider the following array:

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	2	7	8	9	0	5	3	6	4	1

Here, $A[0] = 2, A[2] = 8, A[8] = 4$ and $A[4] = 0$, so $(0, 2, 8, 4)$ is a cycle. The complete cycle decomposition of this permutation is $(0, 2, 8, 4)(1, 7, 6, 3, 9)(5)$.

The function below prints all the cycles one after another in a single line. Each index $i \in \{0, 1, 2, \ldots, n-1\}$ appears in one and exactly one cycle. Therefore, if i is once printed, it will never be printed again. We use an array `printed[]` to store the information about the indices which are printed. The array is initialized to zero. Whenever an i is printed, the corresponding entry in the array is set to a non-zero value. The function stops when all indices are printed. The number of indices printed is stored in the count `nprinted`.

Assume that the array `A[]`, when passed to the function, already stores a permutation of $0, 1, 2, \ldots, n-1$. **(12)**

```
#define MAXSIZE 1000

void cycledecomposition ( int A[], int n )
{
   int printed[MAXSIZE], nprinted, i, j;

   /* Initialize printed[] to all zeros */

                        for (j=0; j<n; ++j) printed[j] = 0;
   /* Initialize the number of indices printed */

   nprinted = ____________________ 0 ____________________ ;
   /* Initialize the first index to start with */
   i = 0;
   /* Repeat until all the indices are printed */

   while ( _____________ nprinted < n _____________ ) {
      /* Find the next unprinted index i by searching the array printed[] */

                                  while (printed[i]) ++i;
      /* Print the cycle starting from index i */

      printf("(%d",i); printed[i] = ____ 1 ____ ; nprinted = ____ nprinted + 1 ____ ;
      /* Now let j iterate over all other indices in the current cycle */
      j = A[i]; /* Initialize j to the index next to i in the cycle */

      while ( _____________ j != i _____________ ) { /* Loop on j */
         /* Print */

                                  printf(",%d",j);
         /* Update nprinted and an appropriate element in printed[] */

                              ++nprinted; printed[j] = 1;
         /* Prepare for the next iteration */

                                  j = A[j];
      }
      printf(")");
   }
   printf("\n");
}
```

5. **(a)** In this exercise, you write a recursive function to print all sequences of $0, 1$ of length n, in which two zeros never appear in consecutive positions. For example, if $n = 3$, then the sequences to be printed are 010, 011, 101, 110, and 111. The length-three sequences 000, 001 and 100 contain consecutive zeros and must not be printed. You are required to print each allowed sequence only once.

The recursive function takes three arguments: an array A, the length n of A, and an index i. The call assumes that zeros and ones are placed appropriately (that is, without violating the constraint on consecutive zeros) in the first i cells of A (that is, from `A[0]` to `A[i-1]`). If all of the n elements are placed, the sequence is printed in a line. Otherwise, `A[i]` is assigned all possible values such that the first $i+1$ cells (`A[0]` to `A[i]`) satisfy the constraint mentioned above. For each allowed value of `A[i]`, a call to `printseq()` is made with appropriate arguments in order to recursively populate the remaining cells (`A[i+1]` through `A[n-1]`). Complete the following code. **(8)**

```
void printseq ( int A[], int n, int i )
{
    int j; /* Needed for printing */

    /* Check the terminating condition */

    if ( ____________i == n____________ ) {
        /* Write a loop to print the sequence stored in A[] */

        ____________for (j=0; j<n; ++j) printf("%d", A[j]);____________
        printf("\n");

        ____________return;____________ /* No task left */
    }

    /* Printing condition is not reached here */
    /* Check whether A[i] is allowed to be 0 */

    if ( ____________(i == 0) || (A[i-1] != 0)____________ ) {
        /* Make a recursive call */

        A[i] = 0; printseq( ____________A, n, i+1____________ );
    }

    /* Now make an unconditional recursive call */

    ____________A[i] = 1; printseq(A, n, i+1);____________
}
```

(b) Suppose that the `main()` function makes only one call to accomplish the desired printing task. How should `printseq()` be called from `main()`? **(2)**

```
#define MAXSIZE 1000

main ()
{
    int A[MAXSIZE], n;

    scanf("%d", &n);

    printseq( ____________A, n, 0____________ );
}
```