End-Semester Test

Session 1

06-July-2021

09:00am – 10:15am

Sections 2, 7, 8, 9, 10

QA1

Part (a) [2 marks]

Two sequences U(n) and V(n) are mutually recursively defined as:

$U(0) = 0$, $U(n) = V(n-1) + 2$ for $n \geqslant 1$.
$V(0) = 1$, $V(n) = 2\,U(n-1) + 3$ for $n \geqslant 1$.

Complete the following code for recursively computing U(n) for an integer $n \geqslant 0$. You must not use a function for computing V(n).

```
int U ( int n )
{
   if (n == 0) return 0;
   if (n == 1) return 3;
   return __________ ;
}
```

```
--------------------------------------------------------------------------------------------------
ANS: 2 * U(n-2) + 5
--------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given R identical red balls, G identical green balls, and B identical blue balls. Assume that R,G,B are positive integers. Your task is to count the arrangements of all the given balls on a line such that no two red balls appear consecutively. For example, if R = 4, G = 1, and B = 2, the only three allowed arrangements are:

```
red, green, red,  blue, red,  blue, red
red,  blue, red, green, red,  blue, red
red,  blue, red,  blue, red, green, red
```

The following function returns the count of the allowed arrangements. The function recursively considers the balls one after another. The parameters r,g,b stand for the respective counts of red, green, and blue balls considered so far. The function chooses each of the three colors (whichever is allowed), considers putting a ball of that color after the arrangement of the balls chosen so far, and makes a recursive call to obtain the count of all possible arrangements that result from this placement of the ball. A parameter lastadded stores the color of the last ball added (use the #define's below). Before the outermost call noRR(R,G,B,0,0,0,UNDEF) in main(), no balls are added, so r = g = b = 0, and lastadded = UNDEF. Complete the code of the function by filling in the blanks. Write the entire lines for [C] and [D]. You are not permitted to use other variables.

```
      #define UNDEF 0
      #define RED 1
      #define GREEN 2
      #define BLUE 3

      int noRR ( int R, int G, int B, int r, int g, int b, int lastadded )
      {
         int count = 0;

         if ((r == R) && (b == B) && (g == G))  /* All balls are considered */
[A]         return __________ ;

         /* Update count recursively after including a green ball (if allowed) */
[B]      if ( g < G ) __________

         /* Update count recursively after including a blue ball (if allowed) */
[C]      if ( __________ ) __________

         /* Update count recursively after including a red ball (if allowed) */
[D]      if ( __________ ) __________

         return count;
      }
```

```
--------------------------------------------------------------------------------------------------
ANS:
   [A] 1
   [B] count += noRR(R,G,B,r,g+1,b,GREEN);
   [C] if (b < B) count += noRR(R,G,B,r,g,b+1,BLUE);
   [D] if ((lastadded != RED) && (r < R)) count += noRR(R,G,B,r+1,g,b,RED);
--------------------------------------------------------------------------------------------------
```

QA2

Part (a) [2 marks]

Two sequences U(n) and V(n) are mutually recursively defined as:

U(0) = 0, U(n) = V(n-1) + 5 for n ⩾ 1.
V(0) = 1, V(n) = 2 U(n-1) + 2 for n ⩾ 1.

Complete the following code for recursively computing U(n) for an integer n ⩾ 0. You must not use a function for computing V(n).

```
int U ( int n )
{
   if (n == 0) return 0;
   if (n == 1) return 6;
   return __________ ;
}
```

```
-----------------------------------------------------------------------------------------------
ANS: 2 * U(n-2) + 7
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given R identical red balls, G identical green balls, and B identical blue balls. Assume that R,G,B are positive integers. Your task is to count the arrangements of all the given balls on a line such that the arrangement starts with a green ball and ends with a non-green (that is, red or blue) ball. For example, if R = 1, G = 2, and B = 1, only the following four arrangements are allowed.

```
green,   red, green, blue
green,  blue, green,  red
green, green,   red, blue
green, green,  blue,  red
```

The recursive function below returns the count of the allowed arrangements. It takes four parameters: r,g,b stand for the respective numbers of red, green, and blue balls yet to be considered, whereas the parameter greenlast is a flag taking the value 1 if the last ball added was green, 0 otherwise. Since the arrangement must start with a green ball, we make the outermost call in main() as GnoG(R,G-1,B,1). The function recursively considers all possible arrangements of the remaining balls, and counts only those arrangements ending with a non-green ball. Each recursive call counts the possible arrangements of the remaining balls if a ball of a given color (if allowed) is placed at the beginning. Complete the code of the function by filling in the blanks. Write the entire lines for [C] and [D]. You are not permitted to use other variables.

```
      int GnoG ( int r, int g, int b, int greenlast )
      {
         int count;

         if ( (r == 0) && (b == 0) ) {  /* All red and blue balls are considered */
[A]         return ( ( __________ ) ? 1 : 0 );
         }

         /* Initialize count recursively after including a red ball (if allowed) */
[B]      count = (r) ? GnoG( __________ ) : 0;

         /* Update count recursively after including a green ball (if allowed) */
[C]      if ( __________ ) __________

         /* Update count recursively after including a blue ball (if allowed) */
[D]      if ( __________ ) __________

         return count;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] (g == 0) && (!greenlast)
   [B] r-1,g,b,0
   [C] if (g) count += GnoG(r,g-1,b,1);
   [D] if (b) count += GnoG(r,g,b-1,0);
-----------------------------------------------------------------------------------------------
```

QA3

Part (a) [2 marks]

Two sequences U(n) and V(n) are mutually recursively defined as:

U(0) = 0, U(n) = V(n-1) + 4 for n ⩾ 1.
V(0) = 1, V(n) = 2 U(n-1) - 1 for n ⩾ 1.

Complete the following code for recursively computing U(n) for an integer n ⩾ 0. You must not use a function for computing V(n).

```
int U ( int n )
{
   if (n == 0) return 0;
   if (n == 1) return 5;
   return __________ ;
}
```

```
-----------------------------------------------------------------------------------------------
ANS: 2 * U(n-2) + 3
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given R identical red balls, G identical green balls, and B identical blue balls. Assume that R,G,B are positive integers. Your task is to count the arrangements of all the given balls on a line such that all the blue balls appear together. For example, if R = 1, G = 1, and B = 2, only the following six arrangements are allowed.

```
  blue,  blue,   red, green
  blue,  blue, green,   red
   red,  blue,  blue, green
 green,  blue,  blue,   red
   red, green,  blue,  blue
 green,   red,  blue,  blue
```

The recursive function below returns the count of such arrangements. In the function, the respective counts r,g,b of red, green, and blue balls yet to be considered are passed as parameters. Before the outermost call in main(), no balls are added, so r = R, g = G, and b = B, and we make this call as allBlue(R,G,B). Each recursive call is meant for counting the possible arrangements of the remaining balls, that result from putting a ball of a given color (if allowed) at the beginning. Complete the function by filling in the blanks. Write the entire lines for [C] and [D]. You are not permitted to use other variables.

```
      int allBlue ( int r, int g, int b )
      {
         int count = 0;
[A]      if ( __________ ) {  /* If not all balls are added yet */

            /* Update count recursively after including one red ball (if allowed) */
[B]         if ( r > 0 ) __________

            /* Update count recursively after including one green ball (if allowed) */
[C]         if ( __________ ) __________

            /* Update count recursively after including all blue balls together (if allowed) */
[D]         if ( __________ ) __________
            return count;
         }
         return 1;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] r || g || b
   [B] count += allBlue(r-1,g,b);
   [C] if (g) count += allBlue(r,g-1,b);
   [D] if (b) count += allBlue(r,g,0);
-----------------------------------------------------------------------------------------------
```

QA4

Part (a) [2 marks]

Two sequences U(n) and V(n) are mutually recursively defined as:

U(0) = 0, U(n) = V(n-1) + 3 for n ⩾ 1.
V(0) = 1, V(n) = 2 U(n-1) - 2 for n ⩾ 1.

Complete the following code for recursively computing U(n) for an integer n ⩾ 0. You must not use a function for computing V(n).

```
int U ( int n )
{
   if (n == 0) return 0;
   if (n == 1) return 4;
   return __________ ;
}
```

```
-----------------------------------------------------------------------------------------------
ANS: 2 * U(n-2) + 1
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given R identical red balls, G identical green balls, and B identical blue balls. Assume that R,G,B are positive integers. Your task is to count the arrangements of all the given balls on a line such that no blue ball appears after a red ball, that is, the red balls are allowed to be added only after all the blue balls are added. For example, if R = 1, G = 1, and B = 2, the following five are the only arrangements allowed.

```
green,  blue,  blue,   red
 blue, green,  blue,   red
 blue,  blue, green,   red
 blue,  blue,   red, green
```

The following function returns the count of such possible arrangements. In addition to R, G, and B, the respective counts r, g, b of red, green, and blue balls considered so far are passed as parameters to the function. The call in main() is noBafterR(R,G,B,0,0,0) because at the beginning, no balls are added to the arrangement. Each recursive call counts the arrangements of the remaining balls if a ball of a given color is put at the beginning (if allowed). Complete the code by filling in the blanks. Write the entire lines for [B] and [C]. You are not permitted to use other variables.

```
      int noBafterR ( int R, int G, int B, int r, int g, int b )
      {
         int count = 0;

         /* If there is at least one more ball to be considered */
         if ((r < R) || (g < G) || (b < B)) {
            /* Update count recursively after including a green ball (if allowed) */
[A]         if ( g < G ) __________

            /* Update count recursively after including a blue ball (if allowed) */
[B]         if ( __________ ) __________

            /* Update count recursively after including a red ball (if allowed) */
[C]         if ( __________ ) __________
         } else {
[D]         count = __________ ;
         }
         return count;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] count += noBafterR(R,G,B,r,g+1,b);
   [B] if (b < B) count += noBafterR(R,G,B,r,g,b+1);
   [C] if ((b == B) && (r < R)) count += noBafterR(R,G,B,r+1,g,b);
   [D] 1
-----------------------------------------------------------------------------------------------
```

QB1

Part (a) [2 marks]

Consider the following function.

```
void func1 (int *p)
{
   int *a;
   a = (int *)malloc(sizeof(int));
   *a =  10;
   *a = (*a)*5; p = a;
}
```

Suppose that in the main function an int type variable x has a value 20. If the function is called two times as func1(&x), what will be the value of x after the second function call returns?

```
------------------------------------------------------------------------------------------------
ANS: 20
------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Fill in the blanks to complete the function findMinStr() below, for which A and n are two of the parameters, where A is a pointer to a pointer to a char, and n is an int. A contains the starting address of an array of n char pointers, with each pointer storing the address of the first character of a null-terminated string. The function returns (through appropriate return value and additional parameters) two things: (i) a new null-terminated string that contains the lexicographically smallest string among the n strings pointed to by the array of pointers, and (ii) the index in the array of pointers, at which the pointer to this smallest string is stored. You can write more than one statement in each blank if you want. You can use string functions if you want. You cannot declare any other variables. For blank [A], write the complete line filling all the blanks.

```
[A]    __________ findMinStr( __________ )
       {
          int i, k = 0;
          char *new;

          for(i = 1; i < n; i++) {
[B]          if ( _________ )
                k = i;
          }
[C]       __________
[D]       __________
          return(new);
       }
```

```
------------------------------------------------------------------------------------------------
ANS:
   [A] char *findMinStr(char **A, int n, int *idx)
   [B] strcmp(A[k], A[i]) > 0
   [C] new = (char *)malloc((strlen(A[k])+1)*sizeof(char));
   [D] strcpy(new, A[k]); *idx = k;
------------------------------------------------------------------------------------------------
```

QB2

Part (a) [2 marks]

Consider the following function.

```
   void func1 (int **p)
   {
      int *a;
      a = (int *)malloc(sizeof(int));
      *a =  10;
      *a = (*a)*5; *p = a;
   }
```

Suppose that in the main function a pointer x is declared which is a pointer to an int type variable. If the function is called two times as func1(&x), what will be the value of *x after the second function call returns?

```
-----------------------------------------------------------------------------------------------
ANS: 50
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Fill in the blanks to complete the function findMaxStr() below, for which X and n are two of the parameters, where X is a pointer to a pointer to a char, and n is an int. X contains the starting address of an array of n char pointers, with each pointer storing the address of the first character of a null-terminated string. The function returns (through appropriate return value and additional parameters) two things: (i) a new null-terminated string that contains the lexicographically largest string among the n strings pointed to by the array of pointers, and (ii) the index in the array of pointers, at which the pointer to this largest string is stored. You can write more than one statement in each blank if you want. You can use string functions if you want. You cannot declare any other variables. For blank [A], write the complete line filling all the blanks.

```
[A]   __________ findMaxStr( __________ )
      {
         int p, q = 0;
         char *ch;

         for(p = 1; p < n; p++) {
[B]         if ( _________ )
               q = p;
         }
[C]      __________
[D]      __________
         return(ch);
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] char *findMaxStr(char **X, int n, int *idx)
   [B] strcmp(X[q], X[p]) < 0
   [C] ch = (char *)malloc((strlen(X[q])+1)*sizeof(char));
   [D] strcpy(ch, X[q]); *idx = q;
-----------------------------------------------------------------------------------------------
```

QB3

Part (a) [2 marks]

Consider the following function.

```
int func1 (int *p)
{
   int *a;
   a = (int *)malloc(sizeof(int));
   *a =  15;
   *a = (*a)*3; p = a;
   return *p;
}
```

Suppose that in the main function an int type variable x has a value 30. If the function is called three times as x = func1(&x), what will be the value of x after the third function call?

```
-----------------------------------------------------------------------------------------------
ANS: 45
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Fill in the blanks to complete the function findMaxStr() below for which P, n, and S are three of the parameters, where P is a pointer to a pointer to a char, n is an int, and S is a null-terminated string. P contains the starting address of an array of n char pointers, with each pointer storing the address of the first character of a null-terminated string. The function returns (through appropriate return value and additional parameters) two things: (i) a new null-terminated string that contains the last string (starting from index 0 of the array of pointers) among the n strings pointed to by the array of pointers, that is lexicographically larger than S, and (ii) the number of strings in the array of pointers which are lexicographicallly larger than S. If there is no string lexicographically larger than S, the function should return NULL, and the number of strings returned can be any value. You can write more than one statement in each blank if you want. You can use string functions if you want. You cannot declare any other variables. For blank [A], write the complete line filling all blank spaces.

```
[A]     __________ findMaxStr ( __________ )
        {
           int i, j = 0, count = 0;
           char *str = NULL;

           for(i = 0; i < n; i++) {
[B]           if ( _________ ) {
                 j = i; count++;
              }
           }
           if (count > 0) {
[C]           __________
[D]           __________
           }
           return(str);
        }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] char *findMaxStr(char **P, int n, int *cnt, char *S)
   [B] strcmp(P[i], S) > 0
   [C] str = (char *)malloc((strlen(P[j])+1)*sizeof(char));
   [D] strcpy(str, P[j]); *cnt = count;
-----------------------------------------------------------------------------------------------
```

QC1

Part (a) [2 marks]

Consider the structure below, where the rank field stores the class rank of a student. Fill in the blanks of the following code to print the class ranks of the students A and C. Your answer cannot contain the letters A and C.

```
struct student {
   int rank;
   struct student *ptr;
};

int main()
{
   struct student A, B, C;
   A.rank = 1; B.rank = 5; C.rank = 10;
   A.ptr = &B; B.ptr = &C; C.ptr = &A;
   printf("%d,%d", __________ );
}
```

```
-----------------------------------------------------------------------------------------------
ANS: B.ptr->ptr->rank, B.ptr->rank
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let findElements() be a function that takes a matrix X[][] and its dimensions m (number of rows) and n (number of columns) as its three arguments. For each element X[i][j] in the matrix, the function prints X[i][j] along with i and j if and only if X[i][j] is the maximum element in row i and also the maximum element in column j. Assume that the mn elements of X are distinct.

As an example, consider the following 4 x 4 matrix:

```
 27  21  16  13
 42  38  33  47
 51  41   4  32
 31  17   9  25
```

The output is:

```
Element found in row 1 and column 3 = 47
Element found in row 2 and column 0 = 51
```

Fill in the blanks in the following C code that realizes the function findElements().

```
       void findElements(int X[100][100], int m, int n)
       {
           int i, j, k, max_row, col_ind = 0;
           int flag = 0;

           for (i = 0; i < m; i++) {
               max_row = X[i][0];
               col_ind = 0;
               for (j = 1; j < n; j++) {
[A]                if ( __________ ) {
[B]                    __________ ;
                       col_ind = j;
                   }
               }
               for (k = 0; k < m; k++)
[C]                if ( __________ ) break;
               if (k == m) {
                  flag = 1;
[D]               printf("Element found in row %d and column %d = %d\n", __________ );
               }
           }
           if (flag == 0) printf("No elements found\n");
       }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] X[i][j] > max_row
   [B] max_row = X[i][j]
   [C] max_row < X[k][col_ind]
   [D] i, col_ind, max_row   /* The third argument can be X[i][col_ind] */
-----------------------------------------------------------------------------------------------
```

QC2

Part (a) [2 marks]

Consider the structure below, where the rank field stores the class rank of a student. Fill in the blanks of the following code to print the class ranks of the students Y and Z. Your answer cannot contain the letters Y and Z.

```
struct student {
   int rank;
   struct student *linkpt;
};

int main()
{
   struct student X, Y, Z;
   X.rank = 3; Y.rank = 6; Z.rank = 9;
   X.linkpt = &Y; Y.linkpt = &Z; Z.linkpt = &X;
   printf("%d,%d", __________ );
}
```

```
--------------------------------------------------------------------------------------------------
ANS: X.linkpt->rank, X.linkpt->linkpt->rank
--------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let findElements() be a function that takes a matrix A[][] and its dimensions m (number of rows) and n (number of columns) as its three arguments. For each element A[i][j] in the matrix, the function prints A[i][j] along with i and j if and only if A[i][j] is the minimum element in row i and also the minimum element in column j. Assume that the mn elements of A are distinct.

As an example, consider the following 4 x 4 matrix:

```
31  23  20   5
17  12  40  26
45   8  32  13
37  50  55   2
```

The output is:

```
Element found in row 2 and column 1 = 8
Element found in row 3 and column 3 = 2
```

Fill in the blanks in the following C code that realizes the function findElements().

```
      void findElements(int A[100][100], int m, int n)
      {
         int i, j, k, min_row, col_ind = 0;
         int flag = 0;
         for (i = 0; i < m; i++) {
            min_row = A[i][0];
            col_ind = 0;
            for (j = 1; j < n; j++) {
[A]            if ( __________ ) {
[B]               __________ ;
                  col_ind = j;
               }
            }
            for (k = 0; k < m; k++)
[C]            if ( __________ ) break;
            if (k == m) {
              flag = 1;
[D]           printf("Element found in row %d and column %d = %d\n", __________ );
            }
         }
         if (flag == 0) printf("No elements found\n");
      }
```

```
--------------------------------------------------------------------------------------------------
ANS:
   [A] A[i][j] < min_row
   [B] min_row = A[i][j]
   [C] min_row > A[k][col_ind]
   [D] i, col_ind, min_row    /* The third argument can be A[i][col_ind] */
--------------------------------------------------------------------------------------------------
```

QC3

Part (a) [2 marks]

Consider the structure below, where the rank field stores the class rank of a student. Fill in the blanks of the following code to print the class ranks of the students M and N. Your answer cannot contain the letters M and N.

```
struct student {
   int rank;
   struct student *listptr;
};

int main()
{
   struct student M, N, P;
   M.rank = 4; N.rank = 8; P.rank = 12;
   M.listptr = &N; N.listptr = &P; P.listptr = &M;
   printf("%d,%d", __________ );
}
```

```
--------------------------------------------------------------------------------
ANS: P.listptr->rank, P.listptr->listptr->rank
--------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let findElements() be a function that takes a matrix M[][] and its dimensions m (number of rows) and n (number of columns) as its three arguments. For each element M[i][j] in the matrix, the function prints M[i][j] along with i and j if and only if M[i][j] is the maximum element in row i and the minimum element in column j. Assume that the mn elements of M are distinct.

As an example, consider the following 4 x 4 matrix:

```
  5  18  23  27
 35  48  30  36
 44  39  24  17
  8   2  19  13
```

The output is:

```
Element found in row 3 and column 2 = 19
```

Fill in the blanks in the following C code that realizes the function findElements().

```
      void findElements(int M[100][100], int m, int n)
      {
          int i, j, k, min_col, row_ind = 0;
          int flag = 0;

          for (j = 0; j < n; j++) {
              min_col = M[0][j];
              row_ind = 0;
              for (i = 1; i < m; i++) {
[A]                 if ( __________ ) {
[B]                     __________ ;
                        row_ind = i;
                    }
              }
              for (k = 0; k < n; k++)
[C]                 if ( __________ ) break;
              if (k == n) {
                 flag = 1;
[D]              printf("Element found in row %d and column %d = %d\n", __________ );
              }
          }
          if (flag == 0) printf("No elements found\n");
      }
```

```
--------------------------------------------------------------------------------
ANS:
   [A] M[i][j] < min_col
   [B] min_col = M[i][j]
   [C] min_col < M[row_ind][k]
   [D] row_ind, j, min_col   /* The third argument can be M[row_ind][j] */
--------------------------------------------------------------------------------
```

QC4

Part (a) [2 marks]

Consider the structure below, where the rank field stores the class rank of a student. Fill in the blanks of the following code to print the class ranks of the students A and B. Your answer cannot contain the letters A and B.

```
struct student {
   int rank;
   struct student *nextptr;
};

int main()
{
   struct student A, B, C;
   A.rank = 1; B.rank = 5; C.rank = 10;
   A.nextptr = &B; B.nextptr = &C; C.nextptr = &A;
   printf("%d,%d", __________ );
}
```

```
------------------------------------------------------------------------------------------------
ANS: C.nextptr->rank, C.nextptr->nextptr->rank
------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let findElements() be a function that takes a matrix T[][] and its dimensions m (number of rows) and n (number of columns) as its three arguments. For each element T[i][j] in the matrix, the function prints T[i][j] along with i and j if and only if T[i][j] is the minimum element in row i and also the maximum element in column j. Assume that the mn elements of T are distinct.

As an example, consider the following 4 x 4 matrix:

```
 6   8  40  19
25  15  10   9
30  41  31  20
36  29  12  11
```

The output is:

```
Element found in row 2 and column 3 = 20
```

Fill in the blanks in the following C code that realizes the function findElements().

```
      void findElements(int T[100][100], int m, int n)
      {
          int i, j, k, max_col, row_ind = 0;
          int flag = 0;
          for (j = 0; j < n; j++) {
              max_col = T[0][j];
              row_ind = 0;
              for (i = 1; i < m; i++) {
[A]               if ( __________ ) {
[B]                   __________ ;
                      row_ind = i;
                  }
              }
              for (k = 0; k < n; k++)
[C]               if ( __________ ) break;
              if (k == n) {
                 flag = 1;
[D]              printf("Element found in row %d and column %d = %d\n", __________ );
              }
          }
          if (flag == 0) printf("No elements found\n");
      }
```

```
------------------------------------------------------------------------------------------------
ANS:
   [A] T[i][j] > max_col
   [B] max_col = T[i][j]
   [C] max_col > T[row_ind][k]
   [D] row_ind, j, max_col   /* The third argument can be T[row_ind][j] */
------------------------------------------------------------------------------------------------
```

QD1

Part (a) [2 marks]

Consider the following definition and assignment.

```
   struct Person {
     int age;
     float weight;
   } p, *t;

   t = &p;
```

Suppose that you want to read age of p, and then print the same using scanf() and printf(). Fill in the blanks to do this. You can use t (and not p) for this purpose.

```
[A]    scanf("%d", __________ );
[B]    printf("%d", __________ );
```

```
------------------------------------------------------------------------------------------------
ANS:
   [A] &(t -> age)
   [B] t -> age
------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given two unsorted linked lists L1 and L2, either one or both of which may be empty. Assume that the nodes of L1 contain distinct values in the field x, and so too do the nodes of L2. However, the same value may appear in a node in L1 and in a node in L2. You have to create a new list L (unsorted) containing the data values that appear in both the lists (that is, the intersection of the values in the lists). For example, if L1 heads the list 4 -> 12 -> 10 -> 5 -> 2 -> 7 -> 3 -> 15, and L2 heads the list 0 -> 11 -> 19 -> 18 -> 5 -> 6 -> 13 -> 15 -> 8 -> 10, then L would head the list 15 -> 5 -> 10. Fill in the blanks in the function intersection that returns L, given L1 and L2 as arguments. You may write multiple statements in each blank. You should not define any extra variable.

```
      struct Node {
         int x;
         struct Node *link;
      };

      struct Node *intersection (struct Node *L1, struct Node *L2)
      {
         struct Node *L, *tmp1, *tmp2, *new;

         if ((L1 == NULL) || (L2 == NULL)) return NULL;

         L = NULL;
         tmp1 = L1;
         do {
            tmp2 = L2;
            do {
[A]            if ( __________ ) break;
[B]            else __________ ;
            } while (tmp2 != NULL);

[C]         if ( __________ ) {
               new = (struct Node *)malloc(sizeof(struct Node));
               new->x = tmp1->x;
[D]            __________
            }
            tmp1 = tmp1->link;
         } while (tmp1 != NULL);
         return L;
      }
```

```
------------------------------------------------------------------------------------------------
ANS:
   [A] tmp1->x == tmp2->x
   [B] tmp2 = tmp2->link
   [C] tmp2 != NULL
   [D] new->link = L; L = new;
------------------------------------------------------------------------------------------------
```

QD2

Part (a) [2 marks]

Consider the following definition.

```
struct Student {
   char name[20];
   float marks;
} s;
```

Suppose that you want to read name of s, and then print the same using scanf() and printf(). Fill in the blanks to do this.

```
[A]    scanf("%s", ___________ );
[B]    printf("%s", ___________ );
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A]  s.name
   [B]  s.name
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given two unsorted linked lists L1 and L2, either one or both of which may be empty. Assume that the nodes of L1 contain distinct values in the field x, and so too do the nodes of L2. However, the same value may appear in a node in L1 and in a node in L2. You have to create a new list L (unsorted) containing the union of all the data values stored in L1 and L2. For example, if L1 heads the list 3 -> 1 -> 6 -> 5, and L2 heads the list 8 -> 5 -> 1 -> 7 -> 0, then L would head the list 6 -> 3 -> 8 -> 5 -> 1 -> 7 -> 0. Assume that there is a function listcopy() that takes the header (a pointer) to a linked list, makes a node-by-node copy of the input linked list, and returns a header (a pointer) to the new list. Fill in the blanks in the function listunion that performs the union operation. You may write multiple statements in each blank. You should not define any extra variable.

```
      struct Node {
         int x;
         struct Node *next;
      };

      struct Node *listunion ( struct Node *L1, struct Node *L2 )
      {
         struct Node *L, *p1, *p2, *q;

         if (L1 == NULL) return listcopy(L2);
         if (L2 == NULL) return listcopy(L1);

         L = listcopy(L2);
         p1 = L1;
         do {
            p2 = L2;
            do {
[A]            if ( __________ ) break;
[B]            else __________ ;
            } while (p2 != NULL);
[C]         if ( __________ ) {
               q = (struct Node *)malloc(sizeof(struct Node));
               q -> x = p1 -> x;
[D]            __________
            }
            p1 = p1 -> next;
         } while (p1 != NULL);
         return L;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] p1 -> x == p2 -> x
   [B] p2 = p2 -> next
   [C] p2 == NULL
   [D] q -> next = L; L = q;
-----------------------------------------------------------------------------------------------
```

QD3

Part (a) [2 marks]

Consider the following definition. We want to store in z an animal of name zyzzyva and of type insect. Complete the initialization of z by filling in the blank.

```
struct animal {
   char name[50];
   char type[20];
} z = __________ ;
```

```
-------------------------------------------------------------------------------------------------
ANS: { "zyzzyva", "insect" }
-------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given two unsorted linked lists L1 and L2, either one or both of which may be empty. Assume that the nodes of L1 contain distinct values in the field x, and so too do the nodes of L2. However, the same value may appear in a node in L1 and in a node in L2. You have to create a new list L (unsorted) containing all the data values stored in L1, that are not present in L2. For example, if L1 heads the list 3 -> 8 -> 1 -> 2 -> 7 -> 4, and L2 heads the list 6 -> 3 -> 7 -> 2, then L would head the list 4 -> 1 -> 8. Fill in the blanks in the following function that, given L1 and L2 as parameters, returns L. You may write multiple statements in each blank. You should not define any extra variable.

```
      struct Node {
         int x;
         struct Node *ptr;
      };

      struct Node *inFirstOnly ( struct Node *L1, struct Node *L2 )
      {
         struct Node *L = NULL, *p, *n;

         while (L1 != NULL) {
            p = L2;
[A]         while ( __________ ) p = p -> ptr;
[B]         if ( __________ ) {
               n = (struct Node *)malloc(sizeof(struct Node));
               n -> x = L1 -> x;
[C]            __________
            }
[D]         __________
         }
         return L;
      }
```

```
-------------------------------------------------------------------------------------------------
ANS:
   [A] (p != NULL) && (p -> x != L1 -> x)
   [B] p == NULL
   [C] n -> ptr = L; L = n;
   [D] L1 = L1 -> ptr;
-------------------------------------------------------------------------------------------------
```

QE1

Part (a) [2 marks]

We want to sort the array A = {6, 3, 9, 1, 5, 8, 7, 2} in ascending order. We use an algorithm that, for i = 0, 1, 2, ... (in that sequence), finds the index j of the smallest element in A[i ... n-1], and swaps A[i] with A[j]. What does the array become just after the first four iterations (for i = 0, 1, 2, 3) complete?

```
-----------------------------------------------------------------------------------------------
ANS:  {1, 2, 3, 5, 6, 8, 7, 9}
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

A positive integer n is called digisorted if the (decimal) digits of n are in non-decreasing order from left to right. For example, the integers 2, 4667, 5555, and 23679 are digisorted, whereas the integers 21, 3451, and 67787 are not digisorted. We impose an ordering on positive integers a and b as follows.

> i) Any digisorted integer precedes any non-digisorted integer (examples: 777 precedes 664, and 13456 precedes 23411).
>
> ii) If both a and b are digisorted, we say a precedes b if and only if a is smaller than b with respect to their values (example: 1233 precedes 1234).
>
> iii) If both a and b are non-digisorted, we say a precedes b if and only if a is smaller than b with respect to their values (example: 2231 precedes 2232).

In the code snippet below, the function digisorted(n) returns 1 if n is digisorted, or 0 if n is not digisorted. The function precedes(a,b) returns 1 if a precedes b (with respect to the ordering mentioned above), or 0 otherwise. Complete the code by filling in the blanks. Write the entire lines in [C] and [D]. Do not use any other variable.

```
      int digisorted ( int n )
      {
         int lastdigit, oldlastdigit;

[A]      oldlastdigit = __________ ;
         while (n) {
            lastdigit = n % 10;
[B]         if ( __________ ) return 0;
            oldlastdigit = lastdigit;
            n /= 10;
         }
         return 1;
      }

      int precedes ( int a, int b )
      {
         int t;

         t = digisorted(b);
         if (digisorted(a)) {
[C]         if (t) return __________ else return __________ ;
         } else {
[D]         if (t) return __________ else return __________ ;
         }
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] 9 (any integer >= 9 will do)
   [B] lastdigit > oldlastdigit
   [C] if (t) return (a < b); else return 1;
   [D] if (t) return 0; else return (a < b);
-----------------------------------------------------------------------------------------------
```

QE2

Part (a) [2 marks]

We want to sort the array A = {4, 3, 2, 8, 6, 1, 9, 5} in descending order. We use an algorithm that, for i = 0, 1, 2, ... (in that sequence), finds the index j of the largest element in A[i ... n-1], and swaps A[i] with A[j]. What does the array become just after the first four iterations (for i = 0, 1, 2, 3) complete?

```
-----------------------------------------------------------------------------------------------
ANS:  {9, 8, 6, 5, 2, 1, 4, 3}
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

A positive integer a is called digisorted if the (decimal) digits of a are in non-increasing order from left to right. For example, the integers 2, 7664, 5555, and 97632 are digisorted, whereas the integers 12, 1543, and 87767 are not digisorted. We impose an ordering on positive integers m and n as follows.

i) Any digisorted integer precedes any non-digisorted integer (examples: 777 precedes 668, and 87650 precedes 95817).

ii) If both m and n are digisorted, we say m precedes n if and only if m is smaller than n with respect to their values (example: 3321 predeces 4321).

iii) If both m and n are non-digisorted, we say m precedes n if and only if m is smaller than n with respect to their values (example: 3323 precedes 3423).

In the code snippet below, the function digisorted(a) returns 1 if a is digisorted, or 0 if a is not digisorted. The function precedes(m,n) returns 1 if m precedes n (with respect to the ordering mentioned above), or 0 otherwise. Complete the code by filling in the blanks. Do not use any other variable.

```
      int digisorted ( int a )
      {
         int lastdigit, oldlastdigit;

[A]      oldlastdigit = __________ ;
         while (a) {
            lastdigit = a % 10;
[B]         if ( __________ ) return 0;
            oldlastdigit = lastdigit;
            a /= 10;
         }
         return 1;
      }

      int precedes ( int m, int n )
      {
         int dsm, dsn;

         dsm = digisorted(m);
         dsn = digisorted(n);
[C]      if (dsm == dsn) return __________ ;
[D]      else return __________ ;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] 0 (any integer <= 0 will do)
   [B] lastdigit < oldlastdigit
   [C] (m < n)
   [D] (dsm > dsn)  (or (dsm == 1) or (dsn == 0) or the && of these two conditions)
-----------------------------------------------------------------------------------------------
```

QE3

Part (a) [2 marks]

We want to sort the array A = {3, 9, 8, 2, 7, 1, 4, 5} in ascending order. We use an algorithm that, for i = n-1, n-2, n-3, ... (in that sequence), finds the index j of the largest element in A[0 ... i], and swaps A[i] with A[j]. What does the array become just after the first four iterations (for i = 7, 6, 5, 4) complete?

```
-----------------------------------------------------------------------------------------------
ANS:  {3, 1, 4, 2, 5, 7, 8, 9}
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let n be a positive integer. By digiflip(n), we denote the positive integer obtained by reversing the digits of n. For example, digiflip(5) = 5, digiflip(767) = 767, digiflip(32224) = 42223, and digiflip(546000) = 000645 = 645. We impose an ordering on positive integers x and y as follows.

i) Any even integer precedes any odd integer (examples: 1234 precedes 789, and 222 precedes 5555).

ii) If x and y are both odd, then we say x precedes y if and only if x is smaller than y with respect to their values (example: 2349 precedes 2351)

iii) If x and y are both even, then we say x precedes y if and only if digiflip(x) is smaller than digiflip(y) with respect to their values (examples: 2340 precedes 2338, and 5678 precedes 4778 and 6678).

Fill in the blanks to complete the code snippet given below. The function digiflip returns digiflip(n) as defined above. The function precedes(x,y) returns 1 if x precedes y (according to the ordering defined above), or 0 otherwise. Do not use any other variable.

```
      int digiflip ( int n )
      {
         int m = 0;

         while (n) {
[A]         m = __________ ;
            n /= 10;
         }
         return m;
      }

      int precedes ( int x, int y )
      {
         int p;

         p = 2 * (x % 2) + (y % 2);
         switch (p) {
            case 1  : return 1;
[B]         case 2  : return __________ ;
[C]         case 3  : return __________ ;
[D]         default : return __________ ;
         }
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] 10 * m + (n % 10)
   [B] 0
   [C] (x < y)
   [D] (digiflip(x) < digiflip(y))
-----------------------------------------------------------------------------------------------
```

End-Semester Test
Session 2
06-July-2021
10:30am – 11:45am
Sections 1, 3, 4, 5, 6

QF1

Part (a) [2 marks]

Recall that Fibonacci numbers are recursively defined as: F(0) = 0, F(1) = 1, and F(n) = F(n-1) + F(n-2) for n ⩾ 2. Define a new sequence G(n) as G(n) = 5 F(n) + 3 for all n ⩾ 0. Complete the code of the following recursive function to return G(n) for an input integer n ⩾ 0. You must not use any function for calculating F(n).

```
int G ( int n )
{
   if (n == 0) return 3;
   if (n == 1) return 8;
   return __________ ;
}
```

```
-----------------------------------------------------------------------------------------------
ANS: G(n-1) + G(n-2) - 3
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let N be a positive integer. We want to find in how many ways N can be written as a sum of the integers 1,2,3 (referred to as summands). We require the number of summands in the sum to be odd. For example, if N = 5, then we have these seven allowed sums: 1 + 2 + 2, 2 + 1 + 2, 2 + 2 + 1, 1 + 1 + 3, 1 + 3 + 1, 3 + 1 + 1, 1 + 1 + 1 + 1 + 1. Sums like 2 + 3 or 1 + 2 + 1 + 1 which contain even numbers of summands should not be counted. Complete the following recursive function to return the desired count. In the function, n stands for the sum obtained so far, and nsmd is the number of summands chosen so far. We make the outermost call in main() as oddsno(N,0,0). In the function, we find out which of the summands 1,2,3 can be considered next. For each allowed summand, we make a recursive call after adding that summand to the sum obtained so far. Fill in the blanks to complete the code. Write the entire lines for [C] and [D] in your answer. Do not use additional variables.

```
      int oddsno ( int N, int n, int nsmd )
      {
         int count;

         if (n == N) {  /* No more summands needed */
            /* Set count depending on whether the number of summands is odd  */
[A]         count = ________ ;
         } else {
            /* Initialize count by choosing 1 as the next summand */
[B]         count = oddsno ( _________ );

            /* Recursively update count by choosing 2 as the next summand (if allowed) */
[C]         if ( __________ ) __________ ;

            /* Recursively update count by choosing 3 as the next summand (if allowed) */
[D]         if ( __________ ) __________ ;
         }
         return count;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] nsmd % 2
   [B] N,n+1,nsmd+1
   [C] if (n <= N-2) count += oddsno(N,n+2,nsmd+1);
   [D] if (n <= N-3) count += oddsno(N,n+3,nsmd+1);
-----------------------------------------------------------------------------------------------
```

QF2

Part (a) [2 marks]

Recall that Fibonacci numbers are recursively defined as: F(0) = 0, F(1) = 1, and F(n) = F(n-1) + F(n-2) for n ⩾ 2. Define a new sequence G(n) as G(n) = 4 F(n) - 1 for all n ⩾ 0. Complete the code of the following recursive function to return G(n) for an input integer n ⩾ 0. You must not use any function for calculating F(n).

```
int G ( int n )
{
   if (n == 0) return -1;
   if (n == 1) return 3;
   return __________ ;
}
```

```
--------------------------------------------------------------------------------------------------
ANS: G(n-1) + G(n-2) + 1
--------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let N be a positive integer. We want to find in how many ways N can be written as a sum of the integers 1,2,3 (referred to as summands). We require the sum to be palindromic (that is, read the same forward and backward). For example, for N = 6, all the allowed palindromic sums are 1 + 1 + 1 + 1 + 1 + 1, 1 + 1 + 2 + 1 + 1, 1 + 2 + 2 + 1, 2 + 1 + 1 + 2, 2 + 2 + 2, and 3 + 3. Non-palindromic sums like 2 + 1 + 3 or 2 + 2 + 1 + 1 are not to be counted. In the recursive function below, n stands for the remaining sum to be realized recursively by considering more summands in a palindromic fashion. The call in main() should be palinsum(N) (no summands are considered, so n is the entire sum N to be realized). If n is 0 or 1, there is only one possibility. Otherwise, the function chooses summands s for putting both at the beginning and at the end of the current expression (if possible). A recursive call is then made for each allowed s. Finally, a check is made whether n can be a single summand. (For example, the expression 3 + 1 + 2 + 1 + 3 for N = 10 can be realized by first considering the summand 3 to go at the beginning and at the end of the expression, and then considering 1 as the next summand to go inside at both the ends, and finally choosing 2 as a single summand sitting at the center. The expression 3 + 2 + 2 + 3 for N = 10 is obtained without using the option of choosing any summand to appear at the center.) Fill in the blanks to complete the code. Write the entire lines for [B] and [C]. Do not use additional variables.

```
      int palinsum ( int n )
      {
         int count;

         if (n <= 1) return 1;        /* Only one possibility */

         /* Initialize count by choosing 1 as the outer summand */
[A]      count = palinsum ( __________ );

         /* Update count recursively by choosing 2 as the outer summand (if allowed) */
[B]      if ( __________ ) __________

         /* Update count recursively by choosing 3 as the outer summand (if allowed) */
[C]      if ( __________ ) __________

[D]      if ( n <= 3 ) __________ ;  /* Case of single summand */

         return count;
      }
```

```
--------------------------------------------------------------------------------------------------
ANS:
   [A] n - 2
   [B] if (n >= 4) count += palinsum(n-4);
   [C] if (n >= 6) count += palinsum(n-6);
   [D] ++count
--------------------------------------------------------------------------------------------------
```

QF3

Part (a) [2 marks]

Recall that Fibonacci numbers are recursively defined as: F(0) = 0, F(1) = 1, and F(n) = F(n-1) + F(n-2) for n ⩾ 2. Define a new sequence G(n) as G(n) = 6 F(n) + 5 for all n ⩾ 0. Complete the code of the following recursive function to return G(n) for an input integer n ⩾ 0. You must not use any function for calculating F(n).

```
int G ( int n )
{
   if (n == 0) return 5;
   if (n == 1) return 11;
   return __________ ;
}
```

```
-----------------------------------------------------------------------------------------------
ANS: G(n-1) + G(n-2) - 5
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let N be a positive integer. We want to find in how many ways N can be written as a sum of the integers 1,2,3 (referred to as summands). We require each summand 1,2,3 to appear at least once in the sum. For example, for N = 6, there are six allowed expressions: 1 + 2 + 3, 1 + 3 + 2, 2 + 1 + 3, 2 + 3 + 1, 3 + 1 + 2, and 3 + 2 + 1. Expressions like 2 + 2 + 2 or 1 + 1 + 3 + 1 missing one or more of 1,2,3 as summand(s) are not to be counted. In the recursive function below, n is the sum realized so far. Moreover, the three flags (0 for false, 1 for true) added1, added2, added3 are passed as parameters to indicate respectively whether 1,2,3 have already been considered as summands. Before the outermost call all123(N,0,0,0,0) made in main(), no summand is added, so n = 0, and all the three flags are 0 as well. In each recursive call, a single summand is chosen (if allowed) for placing at the end of the sum obtained so far, and the count of expressions resulting from this choice is obtained. Fill in the blanks to complete the code. Write the entire lines for [C] and [D]. Do not use additional variables.

```
      int all123 ( int N, int n, int added1, int added2, int added3 )
      {
         int count = 0;

         if (N == n) {  /* No more summands need to be considered */
            /* Check if all of 1,2,3 have appeared in the sum */
[A]         return ( ( __________ ) ? 1 : 0 );
         }

         /* Initialize count by choosing 1 as the next summand */
[B]      count = all123 ( __________ );

         /* Recursive update of count by choosing 2 as the next summand (if allowed) */
[C]      if ( __________ ) __________ ;

         /* Recursive update of count by choosing 3 as the next summand (if allowed) */
[D]      if ( __________ ) __________ ;

         return count;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] added1 && added2 && added3
   [B] N,n+1,1,added2,added3
   [C] if (n <= N-2) count += all123(N,n+2,added1,1,added3);
   [D] if (n <= N-3) count += all123(N,n+3,added1,added2,1);
-----------------------------------------------------------------------------------------------
```

QF4

Part (a) [2 marks]

Recall that Fibonacci numbers are recursively defined as: F(0) = 0, F(1) = 1, and F(n) = F(n-1) + F(n-2) for n ⩾ 2. Define a new sequence G(n) as G(n) = 3 F(n) - 7 for all n ⩾ 0. Complete the code of the following recursive function to return G(n) for an input integer n ⩾ 0. You must not use any function for calculating F(n).

```
int G ( int n )
{
   if (n == 0) return -7;
   if (n == 1) return -4;
   return __________ ;
}
```

```
ANS: G(n-1) + G(n-2) + 7
```

Part (b) [8 marks]

Let N be a positive integer. We want to find in how many ways N can be written as a sum of the integers 1,2,3 (referred to as summands) such that the summands appear in the non-decreasing order in the sum expression. In other words, we want to count expressions of the form $x_1 + x_2 + \cdots + x_k = N$ for some k, where each $x_i \in \{1,2,3\}$, and $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_k$. For example, for N = 6, the following seven expressions are to be counted: 1 + 1 + 1 + 1 + 1 + 1, 1 + 1 + 1 + 1 + 2, 1 + 1 + 1 + 3, 1 + 1 + 2 + 2, 1 + 2 + 3, 2 + 2 + 2, and 3 + 3. Expressions like 1 + 3 + 2 or 1 + 1 + 1 + 2 + 1 where the summands are not in the non-decreasing order should not be counted. The recursive function given below has two parameters: n stands for the sum left to be realized, and lastsmd stores the last summand added to the sum. The recursive calls consider summands that are not smaller than lastsmd. The outermost call in main() can be made as nondecsmd(N,0), because the entire sum N needs to be realized, and no summands are chosen. Each recursive call is used to obtain the count of possibilities resulting from placing an allowed summand at the beginning of the rest of the sum to be realized. Fill in the blanks to complete the code. Write the entire lines for [B] and [C]. Do not use additional variables.

```
      int nondecsmd ( int n, int lastsmd )
      {
         int count;

         if (n) {  /* Not the entire sum N is realized yet */
            count = 0;

            /* If 1 is allowed to be a next summand, update count recursively */
[A]         if ( lastsmd <= 1 ) count += nondecsmd ( __________ );

            /* Update count recursively if 2 is allowed to be a next summand */
[B]         if ( __________ ) __________ ;

            /* Update count recursively if 3 is allowed to be a next summand */
[C]         if ( __________ ) __________ ;
         } else {  /* The entire sum N is realized so n = N - N = 0 */
[D]         __________
         }
         return count;
      }
```

```
ANS:
   [A] n-1,1
   [B] if ((n >= 2) && (lastsmd <= 2)) count += nondecsmd(n-2,2);
   [C] if (n >= 3) count += nondecsmd(n-3,3);
   [D] count = 1;
```

QG1

Part (a) [2 marks]

What will be the values stored in the array A after the following code is executed?

```
   int A[ ] = {0, 5, 1, 7, 8}, n = 5;
   for (i=0; i<n; i++)
      *(A+i) = *(A+n-1-i) = *(A+i);
```

```
-----------------------------------------------------------------------------------------------
ANS: A = {0, 5, 1, 5, 0}
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Fill in the blanks to complete the following function that takes as parameters a fully dynamically allocated 2-d array A of integers, and the number m of rows and the number n of columns of the array. The function returns (through appropriate additional parameters) the starting address of a new integer 1-d array that contains the sum of each row in the 2-d array (so index 0 of the 1-d array returned will contain the sum of row 0 of the 2-d array, index 1 will contain the sum of row 1, and so on). You cannot use the [][] notation to access any element of the 2-d array. You cannot declare any additional variables.

```
[A]   void findSum ( __________ )
      {
         int i, j, sum, *new;

[B]      __________
         for (i = 0; i<m; i++) {
            sum = 0;
[C]         for (j=0; j<n; j++) __________
            *(new + i) = sum;
         }
[D]      __________
         return;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] int **A, int m, int n, int **newarr
   [B] new = (int *)malloc(m*sizeof(int));
   [C] sum += *(*(A+i) + j);
   [D] *newarr = new;
-----------------------------------------------------------------------------------------------
```

QG2

Part (a) [2 marks]

What will be the values stored in the array A after the following code is executed?

```
int A[ ] = {3, 7, 5, 7, 9, 1}, n = 6;
for (i=0; i<n; i++)
   *(A+i) = *(A+n-1-i) = *(A+i);
```

```
-------------------------------------------------------------------------------------------------
ANS: A = {3, 7, 5, 5, 7, 3}
-------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Fill in the blanks to complete the following function that takes as parameters a fully dynamically allocated 2-d array X of integers, and the number p of rows and the number q of columns of the array. The function returns (through appropriate additional parameters) the starting address of a new integer 1-d array that contains the number of positive integers in each row in the 2-d array (so index 0 of the 1-d array returned will contain the number of positive integers in row 0 of the 2-d array, index 1 will contain the number of positive integers in row 1, and so on). You cannot use the [][] notation to access any element of the 2-d array. You also cannot declare any additional variables.

```
[A]    void countPve ( __________ )
       {
          int a, b, *arr, count;

[B]       __________
          for(a = 0; a < p; a++) {
             count = 0;
             for (b = 0; b < q; b++) {
[C]             if ( __________ ) count++;
             }
             *(arr + a) = count;
          }

[D]       __________
          return;
       }
```

```
-------------------------------------------------------------------------------------------------
ANS:
   [A] int **X, int p, int q, int **newarr
   [B] arr = (int *)malloc(p*sizeof(int));
   [C] *(*(X+a) + b) > 0
   [D] *newarr = arr;
-------------------------------------------------------------------------------------------------
```

QG3

Part (a) [2 marks]

Consider the following code fragment, where A is an array of n integers, and B and i are int type variables initialized to 0. What will be the value of B after the code is executed if A initially contains the integers {1, 2, 3, 4, 5, 6}?

```
   for (i=0; i<n; i++)
      B += (*(A+n-1-i) - *(A+i)) * (&A[n-1-i] - &A[i]);
```

```
-----------------------------------------------------------------------------------------------
ANS: 70
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Suppose that you are already given an implementation of the C function

```
   void copyArr(int *A, int n, int *B);
```

that takes as parameters two distinct arrays A and B, each with n elements. The function, when called, copies the elements of A to B. So B is overwritten by A, and A is left unchanged.

Now, consider the following function that takes as parameters a fully dynamically allocated 2-d array P of positive integers, and the number n of rows and the number m of columns of the array. The function returns (through appropriate additional parameters) a new integer 1-d array that contains the elements of the row with the largest sum (of elements in that row) among all the rows of the 2-d array. You can call the given function copyArr() with appropriate parameters in the blanks wherever needed. You can write more than one statement in a blank, but cannot declare any new variable. Also, you cannot use [][] to access any element of a 2-d array.

```
[A]   void findSum ( __________ )
      {
         int i, j, k, sum = 0, maxsum = 0, *p;

         for (j=0; j<n; j++) {
            sum = 0;
[B]         for(i=0; i<m; i++) __________
            if (sum > maxsum) {
               maxsum = sum;  k = j;
            }
         }
[C]      __________
[D]      __________
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] int **P, int n, int m, int **arr
   [B] sum += *(*(P+j) + i);
   [C] p = (int *)malloc(n*sizeof(int));
   [D] copyArr(P[k], m, p); *arr = p;
-----------------------------------------------------------------------------------------------
```

QH1

Part (a) [2 marks]

Fill in the blanks in the following code, so that it prints 5x^100 + 7x^10. Use only the structure of polyterm A in the printf statement.

```
struct polyterm {
   int coeff;
   int power;
   struct polyterm *ptr;
};

int main()
{
   struct polyterm A, B;
   A.coeff = 5; B.coeff = 7; A.power = 100; B.power = 10 ;
   A.ptr = &B;
   printf("%dx^%d + %dx^%d",  __________ );
}
```

```
---------------------------------------------------------------------------------------------------
ANS: A.coeff, A.power, A.ptr->coeff, A.ptr->power
---------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let snakePrint() be a function that takes a matrix A[][] and its dimensions m (number of rows) and n (number of columns) as its three arguments. The function traverses the matrix, starting from the extreme bottom right element A[m-1][n-1], going from right to left on the last row, then going from left to right along the row before the last, and so on until all the elements are printed.

For example, consider the 4 x 3 matrix:

```
  1   2   3
  4   5   6
  7   8   9
 10  11  12
```

The output is: `12 11 10 7 8 9 6 5 4 1 2 3`

Fill in the blanks in the following C code that realizes the function snakePrint().

```
      void snakePrint(int m, int n, int A[100][100])
      {
          int i, j;

[A]       for ( __________ ) {
[B]          if ( __________ == 0 )
[C]               for ( __________ )          /* rows printed from right to left */
                      printf("%d ", A[i][j]);
               else
[D]               for ( __________ )          /* rows printed from left to right */
                      printf("%d ", A[i][j]);
          }
      }
```

```
---------------------------------------------------------------------------------------------------
ANS:
   [A] i = m-1; i >= 0; --i
   [B] (m - 1 - i) % 2
   [C] j = n-1; j >= 0; --j
   [D] j = 0; j < n; ++j
---------------------------------------------------------------------------------------------------
```

QH2

Part (a) [2 marks]

Fill in the blanks in the following code, so that it prints 7x^80 + 9x^20. Use only the structure of polyterm X in the printf statement.

```
struct polyterm {
   int coval;
   int expt;
   struct polyterm *polyptr;
};

int main()
{
   struct polyterm X, Y;
   X.coval = 7; Y.coval = 9; X.expt = 80; Y.expt = 20;
   X.polyptr = &Y;
   printf("%dx^%d + %dx^%d", __________ );
}
```

```
-------------------------------------------------------------------------------------------------
ANS: X.coval, X.expt, X.polyptr->coval, X.polyptr->expt
-------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let snakePrint() be a function that takes a matrix A[][] and its dimensions rowdim (number of rows) and coldim (number of columns) as its three arguments. The function traverses the matrix, starting from the extreme bottom left element A[rowdim-1][0], going from left to right on the last row, then going from right to left along the row before the last, and so on until all the elements are printed.

For example, consider the following 4 x 3 matrix:

```
 1   2   3
 4   5   6
 7   8   9
10  11  12
```

The output is: `10 11 12 9 8 7 4 5 6 3 2 1`

Fill in the blanks in the following C code that realizes the function snakePrint().

```
      void snakePrint(int rowdim, int coldim, int A[100][100])
      {
          int i, j;

[A]       for ( __________ ) {
[B]           if ( __________ == 0 )
[C]               for ( __________ )          /* rows printed from left to right */
                      printf("%d ", A[i][j]);
              else
[D]               for ( __________ )          /* rows printed from right to left */
                      printf("%d ", A[i][j]);
          }
      }
```

```
-------------------------------------------------------------------------------------------------
ANS:
   [A] i = rowdim-1; i >= 0; --i
   [B] (rowdim - 1 - i) % 2
   [C] j = 0; j < coldim; ++j
   [D] j = coldim-1; j >= 0; --j
-------------------------------------------------------------------------------------------------
```

QH3

Part (a) [2 marks]

Fill in the blanks in the following code, so that it prints 9x^60 + 11x^30. Use only the structure of polyterm M in the printf statement.

```
struct polyterm {
   int termval;
   int termexp;
   struct polyterm *ptr;
};

int main()
{
   struct polyterm M, N;
   M.termval = 9; N. termval = 11; M.termexp = 60; N.termexp = 30 ;
   M.ptr = &N;
   printf("%dx^%d + %dx^%d", __________ );
}
```

```
-----------------------------------------------------------------------------------------------
ANS: M.termval, M.termexp, M.ptr->termval, M.ptr->termexp
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let snakePrint() be a function that takes a matrix M[][] and its dimensions r (number of rows) and c (number of columns) as its three arguments. The function traverses the matrix, starting from the extreme top right element M[0][c-1], going from top to bottom on the last column, then going from bottom to top along the column before the last, and so on until all the elements are printed.

For example, consider the following 4 x 3 matrix:

```
1   2   3   4
5   6   7   8
9  10  11  12
```

The output is: `4 8 12 11 7 3 2 6 10 9 5 1`

Fill in the blanks in the following C code that realizes the function snakePrint().

```
      void snakePrint(int r, int c, int M[100][100])
      {
          int i, j;

[A]       for ( __________ ) {
[B]           if ( __________ == 0 )
[C]               for ( __________ )           /* columns printed from top to bottom */
                      printf("%d ", M[i][j]);
              else
[D]               for ( __________ )           /* columns printed from bottom to top */
                      printf("%d ", M[i][j]);
          }
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] j = c - 1; j >= 0; --j
   [B] (c - 1 - j) % 2
   [C] i = 0; i < r; ++i
   [D] i = r-1; i >= 0; --i
-----------------------------------------------------------------------------------------------
```

QH4

Part (a) [2 marks]

Fill in the blanks in the following code, so that it prints 11x^50 + 13x^40. Use only the structure of polyterm P in the printf statement.

```
struct polyterm {
   int val;
   int expo;
   struct polyterm *polyptr;
};

int main()
{
   struct polyterm P, Q;
   P.val = 11; Q.val= 13; P.expo= 50; Q.expo = 40 ;
   P.polyptr = &Q;
   printf("%dx^%d + %dx^%d",  __________ );
}
```

```
------------------------------------------------------------------------------------------------
ANS: P.val, P.expo, P.polyptr->val, P.polyptr->expo
------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Let snakePrint() be a function that takes a matrix M[][] and and its dimensions nrows (number of rows) and ncols (number of columns) as its three arguments. The function traverses the matrix, starting from the extreme bottom right element M[nrows-1][ncols-1], going from bottom to top on the last column, then going from top to bottom along the column before the last, and so on until all the elements are printed.

As an example, consider the following 3 x 4 matrix:

```
1   2   3   4
5   6   7   8
9  10  11  12
```

The output is: `12 8 4 3 7 11 10 6 2 1 5 9`

Fill in the blanks in the following C code that realizes the function snakePrint().

```
      void snakePrint(int nrows, int ncols, int M[100][100])
      {
          int i, j;

[A]       for ( _________ ) {
[B]           if ( __________ == 0 )
[C]               for ( __________ )          /* columns printed from bottom to top */
                      printf("%d ", M[i][j]);
              else
[D]               for ( __________ )          /* columns printed from top to bottom */
                      printf("%d ", M[i][j]);
          }
      }
```

```
------------------------------------------------------------------------------------------------
ANS:
   [A] j = ncols - 1; j >= 0; --j
   [B] (ncols - 1 - j) % 2
   [C] i = nrows - 1; i >= 0; --i
   [D] i = 0; i < nrows; ++i
------------------------------------------------------------------------------------------------
```

QI1

Part (a) [2 marks]

Consider the following declarations.

```
   struct student {
      char name[100];
      char roll[10];
      float cgpa;
   };

   struct student s;
```

The roll number is in the IITKGP format YYDDPNNNN, where YY is the year of registration, DD is the department code, P is the academic program, and NNNN is the student's personal number. We want to print the year of registration of the student s. Fill in the blank to do that.

```
   printf( __________ );
```

```
-----------------------------------------------------------------------------------------------
ANS: "%c%c", s.roll[0], s.roll[1]
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given a linked list L. Your task is to create a new list T containing every third element from L starting from the third element. For example, if L heads the list 3 -> 5 -> 7 -> 9 -> 5 -> 1 -> 10 -> 6 -> 5 -> 1, then T should head a new list 7 -> 1 -> 5. Fill in the blanks to complete the function which returns T, given L as a parameter. Do not use extra variables. You may write multiple statements in each blank.

```
      typedef struct _node {
         int data;
         struct _node *next;
      } node;

      node *onethird ( node *L )
      {
         node *T, *p, *new;

[A]      if ( __________ ) return NULL;

[B]      L = __________ ;
         T = (node *)malloc(sizeof(node));
         T -> data = L -> data;
         T -> next = NULL;
         p = T;

         while (L != NULL) {
            L = L -> next;
[C]         __________
            if (L != NULL) {
               new = (node *)malloc(sizeof(node));
               new -> data = L -> data;
               new -> next = NULL;
               /* Insert at the end */
[D]            __________
            }
         }
         return T;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] (L == NULL) || (L -> next == NULL) || (L -> next -> next == NULL)
   [B] L -> next -> next
   [C] if (L != NULL) L = L -> next; if (L != NULL) L = L -> next;
   [D] p -> next = new; p = p -> next;
-----------------------------------------------------------------------------------------------
```

QI2

Part (a) [2 marks]

Consider the following declarations.

```
struct student {
   char name[100];
   char roll[10];
   float cgpa;
};

struct student myfriend;
```

The roll number is in the IITKGP format YYDDPNNNN, where YY is the year of registration, DD is the department code, P is the academic program, and NNNN is the student's personal number. We want to print the department code of the student myfriend. Fill in the blank to do that.

```
printf( __________ );
```

```
---------------------------------------------------------------------------------------------------
ANS: "%c%c", myfriend.roll[2], myfriend.roll[3]
---------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given a linked list L. Your task is to create a new list T containing every third element from L starting from the second element. For example, if L heads the list 2 -> 4 -> 5 -> 7 -> 10 -> 1 -> 5 -> 9 -> 6 -> 4, then T should head a new list 4 -> 10 -> 9. Fill in the blanks to complete the function which returns T, given L as a parameter. Do not use extra variables. You may write multiple statements in each blank.

```
      typedef struct _node {
         int item;
         struct _node *link;
      } node;

      node *onethird ( node *L )
      {
         node *T, *p, *new;

[A]      if ( __________ ) return NULL;

         L = L -> link;
         T = (node *)malloc(sizeof(node));
         T -> item = L -> item;
         T -> link = NULL;
         p = T;

         while (L != NULL) {
            L = L -> link;
[B]         __________
[C]         if ( __________ ) return T;
            new = (node *)malloc(sizeof(node));
            new -> item = L -> item;
            new -> link = NULL;
            /* Insert at the end */
[D]         __________
         }
      }
```

```
---------------------------------------------------------------------------------------------------
ANS:
   [A] (L == NULL) || (L -> link == NULL)
   [B] if (L != NULL) L = L -> link; if (L != NULL) L = L -> link;
   [C] L == NULL
   [D] p -> link = new; p = p -> link;
---------------------------------------------------------------------------------------------------
```

QI3

Part (a) [2 marks]

Consider the following declarations.

```
struct student {
   char name[100];
   char roll[10];
   float cgpa;
};

struct student thatstud;
```

The roll number is in the IITKGP format YYDDPNNNN, where YY is the year of registration, DD is the department code, P is the academic program, and NNNN is the student's personal number. We want to print the academic program of the student thatstud. Fill in the blank to do that.

```
switch ( _________ ) {
   case '1': printf("BTech"); break;
   case '2': printf("Integrated MSc"); break;
   case '3': printf("Dual-degree");
}
```

```
-----------------------------------------------------------------------------------------------
ANS: thatstud.roll[4]
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

You are given a linked list L. Your task is to create a new list T containing every third element from L starting from the fourth element. For example, if L heads the list 3 -> 4 -> 2 -> 9 -> 1 -> 10 -> 4 -> 5 -> 7 -> 3 -> 4, then T should head a new list 9 -> 4 -> 3. Fill in the blanks to complete the function which returns T, given L as a parameter. Do not use extra variables. You may write multiple statements in each blank.

```
      typedef struct _node {
         int value;
         struct _node *ptr;
      } node;

      node *onethird ( node *L )
      {
         node *T = NULL, *q = NULL, *new;
         int i;

         for (i=0; i<3; ++i) {
[A]         if ( __________ ) return T;
            L = L -> ptr;
         }

         while (L != NULL) {
            new = (node *)malloc(sizeof(node));
            new -> value = L -> value;
            new -> ptr = NULL;
            /* Insert at the end */
            if (q == NULL) {
[B]            __________
            } else {
[C]            __________
            }
            L = L -> ptr;
[D]         __________
         }
         return T;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] L == NULL
   [B] T = q = new;
   [C] q -> ptr = new; q = q -> ptr;
   [D] if (L != NULL) L = L -> ptr; if (L != NULL) L = L -> ptr;
-----------------------------------------------------------------------------------------------
```

QJ1

Part (a) [2 marks]

Consider the array {20, 7, 2, 13, 5, 10, 7, 9, 1, 6, 5}. Insertion sort is used to sort the array in non-decreasing order. What is the content of the array after 6 iterations of the outer for loop?

```
ANS: 2, 5, 7, 7, 10, 13, 20, 9, 1, 6, 5
```

Part (b) [8 marks]

Consider two 1-d integer arrays A and B, of size m and n, respectively. We call A < B if m = n, and for all i, A[i] ⩽ B[i], and there is at least one i such that A[i] < B[i]. Similarly, we call A > B if m = n, and for all i, A[i] ⩾ B[i], and there is at least one i such that A[i] > B[i]. We say that A and B are incomparable if m is not equal to n, or neither A < B nor B < A is true. For example, {1, 3, 5} < {1, 4, 6}, {1, 3, 5} > {1, 2, 5}, whereas {1, 3, 5} and {1, 7, 3} are incomparable. Note that two arrays of the same size and storing the same sequence of elements are also incomparable. Fill in the blanks in the function checkIncomparable below so that it takes as parameters two int arrays A and B with m and n elements, respectively, and returns 1 if A and B are incomparable, 0 otherwise. You cannot declare any new variables. For [B] and [C], write the complete lines filling all the blanks.

```
        int checkIncomparable ( int *A, int *B, int m, int n )
        {
           int i, gf = 0, lf = 0;

[A]        __________
           for (i=0; i<n; ++i) {
[B]           if ( __________ ) __________ ;
[C]           if ( __________ ) __________ ;
           }
[D]        if ( __________ ) return 0;
           else return 1;
        }
```

```
ANS:
   [A] if (m != n) return 1;
   [B] if (A[i] > B[i]) gf = 1;
   [C] if (A[i] < B[i]) lf = 1;
   [D] ((gf == 0) && (lf == 1)) || ((gf == 1) && (lf == 0))  [Alternate answer: gf + lf == 1]
```

QJ2

Part (a) [2 marks]

Consider the array {11, 5, 12, 1, 6, 11, 7, 9, 1, 11, 5}. Insertion sort is used to sort the array in non-decreasing order. What is the content of the array after 5 iterations of the outer for loop?

```
-----------------------------------------------------------------------------------------------
ANS: 1, 5, 6, 11, 11, 12, 7, 9, 1, 11, 5
-----------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Consider two 1-d integer arrays A and B with m and n elements, respectively. It is given that the elements of A and B can only be the integers in the range 1 to 9. We say that A is k-digit-greater than B for a given k in the range $1 \leqslant k \leqslant m$ if $m \leqslant n$, and for all indices i in the range $0 \leqslant i \leqslant m\text{-}k$, the k-digit number formed by taking the k consecutive elements in A starting from index i is strictly greater than the k-digit number formed by taking the k consecutive elements in B starting from the same index i. For example, take A = {2, 5, 1, 6} and B = {2, 4, 3, 7, 9}. Then, A is 3-digit-greater than B since 251 > 243 and 516 > 437, but A is not 2-digit-greater than B because although 25 > 24 and 51 > 43, we have 16 < 37. Fill in the blanks in the function kDigitGreater below so that it takes as parameters two int arrays A and B with m and n elements, respectively, and a positive integer k, and returns 1 if A is k-digit-greater than B, 0 otherwise. You cannot declare any new variables.

```
      int kDigitGreater(int *A, int *B, int m, int n, int k)
      {
         int i, j, flag = 1;

[A]      __________

         /* Check all k-digit possibilities in A */
[B]      for ( __________ ) {
            j = i;

            /* Check digit by digit */
[C]         while ( __________ ) j++;

[D]         if ( __________ ) { flag = 0; break; }
         }
         return flag;
      }
```

```
-----------------------------------------------------------------------------------------------
ANS:
   [A] if (m > n) return 0;
   [B] i = 0; i <= m-k; ++i
   [C] (j < i+k) && (A[j] == B[j])
   [D] (j == i+k) || (A[j] < B[j])
-----------------------------------------------------------------------------------------------
```

QJ3

Part (a) [2 marks]

Consider the array {5, 3, 12, 1, 4, 12, 7, 1, 15, 12, 10}. Insertion sort is used to sort the array in non-decreasing order. What is the content of the array after 6 iterations of the outer for loop?

```
-------------------------------------------------------------------------------------------------
ANS: 1, 3, 4, 5, 7, 12, 12, 1, 15, 12, 10
-------------------------------------------------------------------------------------------------
```

Part (b) [8 marks]

Consider two character arrays A and B, each containing null-terminated strings. We say A and B are k-conjugal for a given k if the lengths of A and B are the same, k is a factor of that length, and each k-character block of A is either the same as that of B or the reverse of that of B (the k-character blocks are the blocks of characters at indices 0 to k-1, k to 2k-1, and so on). For example, take the strings A = "abxypq" and B = "baxypq". Then, A and B are 2-conjugal (the blocks ab, xy, and pq of A are either the same as or the reverse of the corresponding blocks ba, xy, and pq of B), but not 3-conjugal (because abx is neither the same as nor the reverse of bax). Fill in the blanks in the function kConjugal below so that it takes as parameters two characters arrays A and B holding null-terminated strings, and an integer k, and returns 1 if A and B are k-conjugal, 0 otherwise. You cannot declare any new variables.

```
      int kConjugal ( char *A, char *B, int k )
      {
         int i, j, len, fm, bm;

         len = strlen(A);
[A]      if ( _________ ) return 0;
         for (i = 0; i < len; i += k) {
            /* Check if the blocks are the same or the reverse */
            fm = bm = 0;
            for (j = 0; j < k; ++j) {
[B]            if ( __________ ) ++fm;
[C]            if ( __________ ) ++bm;
            }
[D]         if ( __________ ) return 0;
         }
         return 1;
      }
```

```
-------------------------------------------------------------------------------------------------
ANS:
   [A] (len != strlen(B)) || (len % k != 0)
   [B] A[i+j] == B[i+j]
   [C] A[i+j] == B[i-j+k-1]
   [D] (fm < k) && (bm < k)
-------------------------------------------------------------------------------------------------
```