INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Programming and Data Structures (CS10003) **Spring Semester 2021-22**
Long Test 2 - Part A **Marks=50**

21 June 2022 from 09:00 to 10:10

Instructions:

1. *Write your answers on paper. Answers must be handwritten. Typed or written answers using an electronic device are not allowed.*
2. *Write your name and roll number on each page. Write page numbers for each page.*
3. *Answer all five questions. All parts of a question must be answered in the same place.*
4. *Scan all pages and collate. Create a single PDF file (of size <= 10 MB). You could also take pictures of different pages, combine them to make a single pdf file.*
5. *The name of the file should be <RollNumber>_LT2A. Ensure your roll number in the filename has only digits and uppercase characters.*
6. *Upload/Attach your file. Make sure you click on the 'Turn in' button to submit your file.*
7. *10:10 AM is a strict deadline after which no submissions will be allowed.*
8. *We will not accept submission by any other means.*

---

1. What is the output of the following C program when the input is the last two digits of your roll number? [6 marks]

```
#include <stdio.h>

int k=50;

int mystery(int a[])  {
    return(a[0] + a[1]);
}

int f(int k)  {
  k=k+1; return k;
}

int g(int x)  {
  return (x*k);
}

int main()  {
      int a, x[6], j, k;
      scanf("%d",&k); printf("k=%d\n",k);

      for (j=0; j<6; j++) x[j] = k+j;
      printf("\n%d %d %d\n",
                  mystery(x), mystery(&x[2]), mystery(x+3));

      a = f(k);
      printf("\n%d %d %d", k, a, g(k));
}
```

Solution:

If the input is Z, then the program prints Z + (Z+1), (Z+2) + (Z+3), and (Z+3) + (Z+4). For example, if the input is 30, then the output is 30+31=61, 32+33=65, and 33+34=67.

In the next part, the output is Z, Z + 1, and 50 x Z. For example, if the input is 30, then the output is 30 31 1500.

2. The following code shows the bubble-sort function and the associated function for swapping.
   (a) Indicate what code should fill the blanks with the red letters in the function `swap()`.

```
     #include <stdio.h>
L1:  void bubbleSort(int arr[], int n)
L2:  {
L3:        int i, j;
L4:        for (i = 0; i < n - 1; i++)
L5:              for (j = 0; j < n - i - 1; j++)
L6:                    if (arr[j] > arr[j + 1])
L7:                          swap(&arr[j], &arr[j + 1]);
L8:  }
     void swap(____A___)
     {
           int temp;
           temp = ____B____;
           ___C___  = *b ;
           ___D___  = temp;
     }
```

[4 marks]

   (b) As you may know, the last element of the array is exchanged at most once by the function `bubbleSort()`. For some reason we want to use the philosophy of the algorithm in a way that the first element is exchanged at most once. This can be done by rewriting only the line L5. How will you rewrite L5 to make this possible?

[2 marks]

Solution:

(a) Swap function:

```
void swap(int *a, int *b)
{
int temp;
temp = *a;
*a = *b;
*b = temp;
}
```

(b) Modified bubbleSort:

```
L1:  void bubbleSort(int arr[], int n)
L2:  {
L3:        int i, j;
L4:         for (i = 0; i < n - 1; i++)
L5:               for (j = n-2; j >= i; j--)
L6:                     if (arr[j] > arr[j + 1])
L7:                           swap(&arr[j], &arr[j + 1]);
L8:    }
```

3. What does the following program print when the input is the last digit of your roll number?

```
#include <stdio.h>
#include <stdlib.h>

struct node {
    int key;
    struct node * next;
};

int main()
{
    struct node *x, *y, *t;
    int N;
    x = (struct node *) malloc(sizeof(struct node));
    y = (struct node *) malloc(sizeof(struct node));
    x->next = y; y->next = x;
    scanf("%d",&N); printf("N=%d\n",N);
    x->key = y->key = N+1;




    for (t=x; t->key < 25; t=t->next){
           t->key = x->key + y->key;
           printf("%d ", t->key);
    }
    printf("\n");
}
```

[6 marks]

Solution:

The program generates a Fibonacci sequence seeded by $f_0 = N+1$, $f_1 = N+1$. The program prints N and then f_2, f_3, … where each $f_k = f_{k-1} + f_{k-2}$. The program stops after two successive numbers greater than or equal to 25 are printed.

For N=4, the output is:

```
4
N=4
10 15 25 40
```

For N=3, the output is:

```
3
N=3
8 12 20 32 52
```

4. Given the type definitions as follows:

```
struct list
{
        int data;
        struct list *next;
};
typedef struct list ELEMENT;
typedef ELEMENT *LINK;
```

Write a non-recursive function **`LINK reverse_unique (LINK head)`** that takes in the start pointer to an existing linked list of elements of the given type mentioned above and returns the start pointer to the same list with the elements reversed after removal of duplicate copies of elements. During deletion in case of duplicate copies, the SECOND COPY of an element in the original forward list is to be retained and the other copies deleted. If there is only one copy of an element, it is retained in its proper order. For example, if the original list is <2, 3, 3, 2, 4, 5, 2, 4, 2, 3, 1> then the reversed list pointed to by the returned pointer will be <1, 4, 5, 2, 3>. Free the deleted elements. You are not allowed to use any additional array. You are also not allowed to dynamically allocate any new element. Do not write any other auxiliary function nor define any new user-defined data type. You can use only at most 10 local variables in the function which could be pointers (of type `LINK`) or integers only. Use only standard input output library. Write suitable comments in your code to explain the steps and method.

[16 marks]

Sample Solution:

```
#include<stdio.h>
#include<stdlib.h>
struct list
{
    int data;
    struct list *next;
};
typedef struct list ELEMENT;
typedef ELEMENT *LINK;
LINK create_node(int val)
{
    LINK newp;
    newp = (LINK)malloc(sizeof(ELEMENT));
    newp->data = val;

    newp->next = NULL;

    return (newp);

}


LINK insert (int value, LINK ptr)

{

    LINK newp, prev, first;

    newp = create_node(value);

    if (ptr == NULL)

        return (newp);


    else

    {

        first = ptr;

        prev = ptr;

        while (prev->next!=NULL) prev = prev->next;

        prev->next = newp;

        return (first);


    }

}



void print_list(LINK head)

{

    if (head ==NULL) printf("\n");

    else

    {

        printf("%d, ", head->data);

        print_list(head->next);

    }
```

```
}



LINK reverse_unique(LINK head)
{
    LINK ptr, start, prev, place, tail, qtemp;
    int flag, value,temp;
    // REMOVE DUPLICATES KEEPING SECOND IF THERE ARE
DUPLICATES
    start = head;
    while (start!=NULL)
    {
        ptr = start;
        value = start->data;
        flag = 0;
        while(ptr!=NULL)
        {
            if(ptr->data == value) flag++;
            if (flag == 2) {place = ptr; break;}
            ptr = ptr->next;
        }
        start = start->next;
        if (flag ==1) continue;
        if ( head->data == value)
        {
            ptr = head;
            head = head->next;
            ptr->next = NULL;
            free(ptr);

        }
        else
        {
            prev = head;
            ptr = head->next;
            while(ptr->data!=value)
            {
                prev = ptr;
                ptr = ptr->next;
            }
            prev->next = ptr->next;
            ptr->next = NULL;
            free(ptr);
        }
        prev = place;
        ptr = place->next;
        while(ptr!=NULL)
        {
            if(ptr->data == value)
            {
                prev->next = ptr->next;
                ptr->next = NULL;
                free(ptr);
                ptr = prev->next;
            }
            else
            {
                prev = ptr;
                ptr = ptr->next;
            }
        }

}

// REVERSE
```

```
    if (head == NULL) return head;
    tail = head;
    while (tail->next != NULL) tail = tail -> next;
    ptr = head;
    while(ptr != tail)
    {
        temp = ptr-> data;
        ptr -> data = tail -> data;
        tail -> data = temp;
        if (ptr-> next == tail) break;
        qtemp = ptr;
        while(qtemp->next != tail) qtemp = qtemp ->next;
        tail = qtemp;
        ptr = ptr->next;
    }
    return head;
}


main()
{
    int n, i, val, m, result;
    LINK pointer;
    printf("Give number of Elements: \n");
    scanf("%d", &n);
    printf("Number of Elements = %d \n", n);
    for (i = 1; i <=n; i++)
    {
        printf("Give the %d-th element: ", i);
        scanf("%d", &val);
        printf("%d-th value read = %d \n", i, val);
        if (i == 1) pointer = create_node(val);
```

```
                else pointer = insert(val, pointer);
            }
            printf("The Read List is: \n");
            print_list(pointer);
            pointer = reverse_unique(pointer);
            printf("The Reversed List after duplicate removal is: \n");
            print_list(pointer);
        }
```

5. A 2n × 2n matrix ‘a’ may be written as

$$\begin{bmatrix} lua & rua \\ lla & rla \end{bmatrix}$$

where *lua*, *rua*, *lla* and *rla* are respectively the four n×n left upper, right upper, left lower and right lower sub-matrices of ‘a’.

Write a C program that would take a 2n × 2n matrix ‘a’ as input and print the following 2n × 2n matrix

$$\begin{bmatrix} lua & lla \\ rua & rla \end{bmatrix}$$

Example: If n=2 and the input is the following 4×4 matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

then the final output should be

$$\begin{bmatrix} 1 & 2 & 9 & 10 \\ 5 & 6 & 13 & 14 \\ 3 & 4 & 11 & 12 \\ 7 & 8 & 15 & 16 \end{bmatrix}$$

The program must be written strictly as follows.

- Given n as input, only one 2n×2n matrix of integers must be dynamically allocated in the program.
- Only two n × n matrices must be dynamically allocated in the entire program.
- In order to generate any of the 4 n×n submatrices calls to the function having prototype

  ```
  void copysubmatrix(int** b, int** a, int I, int J, int n);
  ```

  where the n×n submatrix of the 2n×2n matrix a[][] (with uppermost leftmost entry a[I][J]), would be copied into the n×n matrix b[][]. For instance the function call for computing rla would be `copysubmatrix(rla,a,n,n,n)` and for computing lla would be `copysubmatrix(lla,a,n,0,n)`.

- In order to transform the 2n×2n matrix a[][] to

  $$\begin{bmatrix} lua & lla \\ rua & rla \end{bmatrix}$$

  make two function calls to the function having prototyp

  `void revcopysubmatrix (int** b, int** a, int I, int J, int n);`

  where the n×n matrix b[][] will be placed in the 2n×2n matrix a[][] with the element b[0][0] as a[I][J]. So, you must call `revcopysubmatrix(rua,a,n,0,n)` to place rua at the correct destination in a[][].
- You must write the functions `copysubmatrix` and `revcopysubmatrix`, and the entire main program. The main program must read a[][] in the beginning and print a[][] at the end.

[16 marks]

Sample Solution:

```
#include <stdio.h>

#include <stdlib.h>


int ** arraymalloc(int n);

void readarray(int **a,int n);

void copysubmatrix (int **A,int **a,int I, int J,int n);

void revcopysubmatrix (int **A,int **c,int I, int J, int n);

void printCall(int **a, int n);


int main() {

  int **a,n;

  int **lua,**rua,**lla,**rla;

  scanf("%d",&n);

  a = arraymalloc(2*n);

  rua = arraymalloc(n);

  lla = arraymalloc(n);


  readarray(a,2*n);

  printCall(a,2*n);


  copysubmatrix(rua,a,0,n,n);

  copysubmatrix(lla,a,n,0,n);
  printCall(rua,n);
  printCall(lla,n);

  revcopysubmatrix(rua,a,n,0,n);
  revcopysubmatrix(lla,a,0,n,n);

  printCall(a,2*n);
}


int ** arraymalloc(int n){
  int **a,i;

  a = (int **)malloc(n* sizeof(int *));
  for(i = 0;i < n;i ++)
    a[i] = (int *)malloc(n * sizeof(int));
  return(a);
}

void readarray(int **a,int n){
  int i,j;
  printf("\nStart entering the values of the  matrix row by
row...\n");
  for(i = 0;i < n;i ++) {
    for(j = 0;j < n;j ++)
      scanf("%d", &a[i][j]);
  }
}

void copysubmatrix (int **A,int **a,int I, int J,int n){
  int i,j;
  for(i = I;i < I+n; i ++)
    for(j = J;j < J+n; j ++){
      A[i-I][j-J]=a[i][j];
    }
}

void revcopysubmatrix (int **A,int **c,int I, int J,int n){
  int i,j;

  for(i = I;i < I+n; i ++)
    for(j = J;j < J+n; j ++){
      c[i][j]=A[i-I][j-J];
    }
}

void printCall(int **a, int n) {
  int i, j, k;
  for(i = 0;i < n;i ++) {
    for(j = 0;j < n;j ++)
      printf("%d ", a[i][j]);
      printf("\n");
  }
  printf("\n\n");
}
```