CMR INSTITUTE OF TECHNOLOGY

Solution/Model Answer of IAT-1 (Sept 2018)
Data Structures & Applications (17CS33)
III Sem – CSE
Dr. P. N. Singh, Professor(CSE)

1.

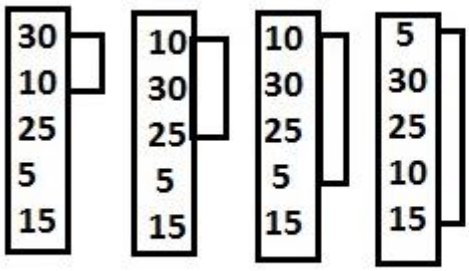
a. Demonstrate selection sort with array of 5 integers **5**

Ans:

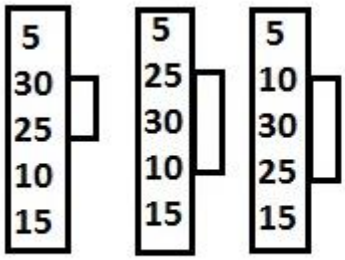
In selection sort theme of algorithm is to select minimum from the array and replace with first element. Then, it finds the minimum from rest of the element and replaces with 2nd element and so on.

The algorithm begins with comparing 1st with 2nd and if 1st is greater than there will be an exchange, then 1st element is compared with 3rd (for the same), 1st with 4th, 1st with 5th and after first pass the minimum element comes at first position. This is repeated for rest of the elements. After n-1 passes and n*(n-1)/2 comparisons the array is sorted.

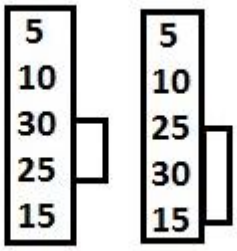
Demo: Let the array of 5 integers be: 30,10,25,5,15



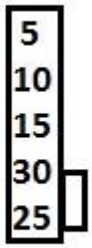
After first pass the array:



After 2nd pass the array



3rd pass



4th pass

Sorted Array

b. Write a C Program to sort n strings in alphabetical order **5**

Ans:

```
/* sorting strings */
#include <stdio.h>
#include <string.h>

void sortstr(char str[][20], int n)
{
   int x,y;
   char temp[20];
   for(x=0;x<n;x++)
     for(y=x+1;y<n;y++) /* using selection sort */
   if(strcmp(str[x],str[y]) > 0)
   {
      strcpy(temp,str[x]);
      strcpy(str[x],str[y]);
      strcpy(str[y],temp);
   }
}

int main()
{
   char str[10][20];
   int x;

   for(x=0;x<10;x++)
   {
      printf("Enter string %d : ",x+1);
      gets(str[x]);
   }
   sortstr(str,10);
   printf("Sorted strings:\n");
   for(x=0;x<10;x++)
      printf("%s\n",str[x]);

   return (0);
 }
```

2. Write a program using nested structure to read & display information about 60 employees. Consider the field names empid, name, salary and doj(date month and year) 10

Ans:

```
/*nested structure*/
#include <stdio.h>
struct employee
{
   int empid;
   char name[20];
   int salary;
   struct date { int dd,mm,yy; }doj; /* nesting of structure */
```

```
}e[60]; /* array of structure */


 int main()
 {
    int x;
    for(x=0;x<5;x++)
    {
      printf("Enter empid, name, salary & doj for emp %d : ", x+1);
      scanf("%d%s%d%d%d%d",&e[x].empid,e[x].name,&e[x].salary,
                    &e[x].doj.dd,&e[x].doj.mm,&e[x].doj.yy);
    }
    printf("Empid       Name        Salary  date of joining:\n");
    for(x=0;x<5;x++)
       printf("%5d%-20s%5d%5d/%2d/%4d\n",
                   e[x].empid,e[x].name,e[x].salary,
                     e[x].doj.dd,e[x].doj.mm,e[x].doj.yy);

    return (0);
 }
```

%-20s for left alignment of names

Expected output for 5 employees:

Enter empid, name, salary & doj for emp 1 : 111 paras 30000 20 8 2018
Enter empid, name, salary & doj for emp 2 : 222 kumar 25000 13 4 17
Enter empid, name, salary & doj for emp 3 : 333 ramesh 13000 15 3 2019
Enter empid, name, salary & doj for emp 4 : 444 dhannu 16000 17 4 2017
Enter empid, name, salary & doj for emp 5 : 555 kushum 20000 14 8 1992

Empid	Name	Salary	date of joining:
111	paras	30000	20/ 8/2018
222	kumar	25000	13/ 4/ 17
333	ramesh	13000	15/ 3/2019
444	dhannu	16000	17/ 4/2017
555	kushum	20000	14/ 8/1992

3.

a. How a sparse matrix is converted to triplets? Write with example. 5

Ans:

Let the sparse matrix be:

$$\begin{pmatrix} 10 & 0 & 0 & 25 & 0 \\ 0 & 23 & 0 & 0 & 45 \\ 0 & 0 & 0 & 0 & 32 \\ 42 & 0 & 0 & 31 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 30 & 0 & 0 \end{pmatrix}$$

Converting in triplets:

First/header row of triplets will contain total number of rows, total number of columns and total number of non-zero elements of the matrix i.e. 6, 5 and 8 respectively and rest of the triplets will have corresponding row number, column number of non-zero element and non-zero element itself

6	5	8
0	0	10
0	3	25
1	1	23
1	4	45
2	4	32
3	0	42
3	3	31
5	2	30

b. Write a C program or fast transpose algorithm to transpose a sparse matrix using triplets. 5

Ans:

Writing function with input triplets trip1[][3] to be transposed to trip2[][3]

```
void transpose(int trip1[][3],int trip2[][3])
{
    int x,y,z,n;
    trip2[0][0] = trip1[0][1];
    trip2[0][1] = trip1[0][0];
    trip2[0][2] = trip1[0][2];

    z=1;
    n=trip1[0][2];

     for(x=0;x<trip1[0][1];x++)
   for(y=1;y<=n;y++)
       //if a column number of current triple==i
       //then insert the current triple in b2
       if(x==trip1[y][1])
       {
         trip2[z][0]=x;
         trip2[z][1]=trip1[y][0];
         trip2[z][2]=trip1[y][2];
         z++;
       }
}
```

4.

a. Explain solution of Tower of Hanoi problem with example of 3 discs. 5

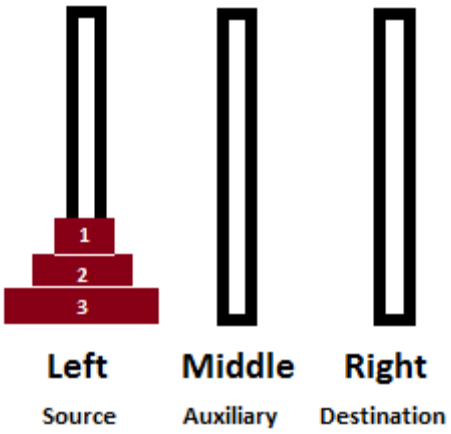
Ans:

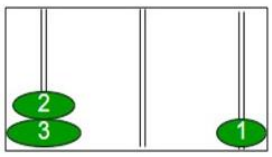
The Problem:
Lord Brahma has given 3 gold pegs and 64 gold disks of lowered size inserted in one peg to his 7 monks (Saptarishi). The task was to shift all diskettes from "left/source" to "right/destination" peg taking help of "middle/auxiliary" peg with conditions:

- Move 1 disk at a time
- There must not be a larger disk on smaller 1

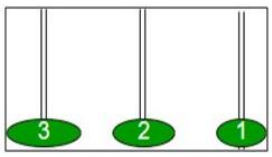
This is world-wide famous problem where complexity grows exponentially – $O(2^n)$

Solution for 3 discs:

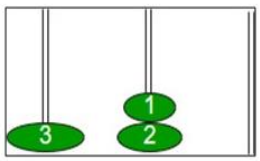




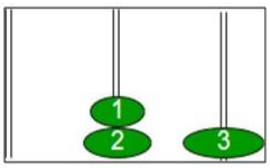
1. Move disk 1 from Left to Right



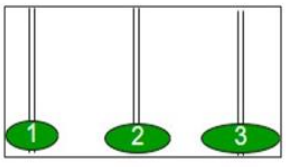
2. Move disk 2 from Left to Middle



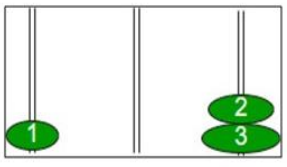
3. Move disk 1 from right to Middle



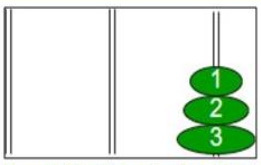
4. Move disk 3 from Left to Right



5. Move disk 1 from Middle to Leftt



6. Move disk 1 from Middle to Leftt



7. Move disk 1 from Left to Right

b. Write a C program to solve the "Tower of Hanoi" problem using recursion. 5

Ans:

```
/* Tower of Hanoi - Solution by recursion   - Dr. P. N. Singh */

#include<stdio.h>
long int k=0;
void hanoi(int d,char l[],char r[],char m[])
{
     if(d>0)
     {
      hanoi(d-1,l,m,r);
      printf("%7ld. Move disk %d from %s to %s\n",++k,d,l,r);
      hanoi(d-1,m,r,l);
     }
}

 main()

  {
      int d;
      printf("How many disks ? ");
      scanf("%d",&d);
      hanoi(d,"Left","Right","Middle");
      return(0);
  }
```

Output:
How many disks ? 3

1. Move disk 1 from Left to Right
2. Move disk 2 from Left to Middle
3. Move disk 1 from Right to Middle
4. Move disk 3 from Left to Right
5. Move disk 1 from Middle to Left
6. Move disk 2 from Middle to Right
7. Move disk 1 from Left to Right

5. Write the Knuth Morris Pratt pattern matching algorithm and apply the same to search the pattern 'abcdabcy' in the text 'abcxabcdabxabcdabcdabcy'. Demonstrate steps also. 10

Ans:

Knuth Morris Pratt Pattern Matching Algorithm:
algorithm kunth_morris_pratt
input:
S // array of characters (String/text to be searched)
W // array of characters (sub-string/pattern to find)
output:
an integer (starting position where W is found in S)

1. m=0 // the beginning of the current match in S

```
2.              i=0  // the position of the current character in W)
3.              T[]  //  array of integers (to track and record position)

4.              while m + i < length(S) do
4.1 if W[i] = S[m + i] then
     if i = length(W) - 1 then
return m
    endif
      i = i+1
        4.2 else if T[i] > -1 then
          m = m + i - T[i]
            i =  T[i]
        4.3      else
         m = m + 1
           i = 0
4.4    endif
4.5 endif
5.  Endwhile
6            return the length of S  // if reached here
```

Applying the algorithm to **search the pattern 'abcdabcy' in the text 'abcxabcdabxabcdabcdabcy'**

```
                    1         2
m:      012345678901234567890123
S:      abcxabcdabxabcdabcdabcy
W:      abcdabcy
i:      01234567
                    //Mismatches on 3 so, next comparison after overlapping from 3

                    1         2
m:      012345678901234567890123
S:      abcxabcdabxabcdabcdabcy
W:        abcdabcy
i:        01234567
                        //Mismatches on 3(first char) so, next comparison from 4

                    1         2
m:      012345678901234567890123
S:      abcxabcdabxabcdabcdabcy
W:          abcdabcy
i:          0123456

        //Mismatches on 10 but next ab starts at 8 so, next comparison after overlapping from 8
                    1         2
m:      012345678901234567890123
S:      abcxabcdabxabcdabcdabcy
W:              abcdabcy
i:              01234567

                    //Mismatches on 10 so, next comparison after overlapping from 10

                    1         2
m:      012345678901234567890123
```

```
S:      abcxabcdabxabcdabcdabcy
W:              abcdabcy
i:               01234567
```

//Mismatches on 10(first char) so, next comparison from 11

```
              1         2
m:    01234567890123456789012
S:    abcxabcdabxabcdabcdabcy
W:              abcdabcy
i:               0123456 7
```

//Mismatches on 18 but next ab starts at 15 so, next comparison after overlapping from 15

```
              1         2
m:    01234567890123456789012
S:    abcxabcdabxabcdabcdabcy
W:                   abcdabcy
i:                    01234567
```

//All characters of pattern match starting from 15 – Found

6.

a. Write algorithms or functions to insert and delete elements from a linear queue with alerts when queue is full or empty. 5

Ans:

```
#define MAX 10

int  rear=-1,front=-1;

void insertq(int q[], int item)
{
  if(rear==MAX-1)
   printf("Queue is full\n");
  else
   {
     rear++;
     q[rear]=item;
     if(front==-1) front=0; /*for first time*/
   }
 }

int deleteq(int q[])
 {
   int d;
   if(front==-1)
    { printf("Queue is empty\n"); return (NULL); }
   else
    {
       d=q[front]; q[front]=0; front++;
       if(front==rear+1)
```

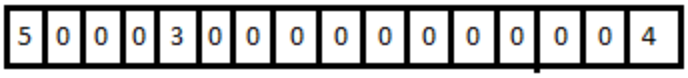
```
                front=rear=-1; /*when all items deleted */
            return (d);
        }
    }
```

b. Consider two polynomials $A(x) = 4x^{15} + 3x^4 + 5$ and $B(x) = x^4 + 10x^2 + 1$
Show diagrammatically how these two polynomials can be stored in a 1- D array. Also give its C representation for initialization. 6

Ans:

Diagrammatic representation of given polynomial in one-dimensional array:
Array representation assumes that the exponents of the given expression are arranged from 0 to the highest value (degree), which is represented by the index/subscript (element number of the array) of the array beginning with 0. The coefficients of the respective exponent are placed at an appropriate index in the array.

$A(x) = 4x^{15} + 3x^4 + 5$

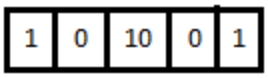


Exponents/Indices 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Now C representation:

int Ax [16] = { 5,0,0,0,3,0,0,0,0,0,0,0,0,0,0,4};

$B(x) = x^4 + 10x^2 + 1$



Exponents/Indices 0 1 2 3 4

And C representation:

int Bx [5] = { 1,0,10,0,1};

7.

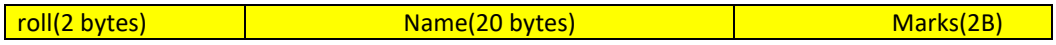
a. Differentiate structure and union with examples. 4

Ans:

Union is similar to structure but it differs in terms of memory allocation. Unions allow a portion of memory to be accessed as different data types, since all of them are in fact the same location in memory. Its declaration and use is similar to the one of structures but its functionality is totally different. The keyword is **union**.
For structure memory allocation is done separately for each member according to their size. For union, only one memory location is provided according to largest size of member and all members can use that location one by one. The keyword is **struct**.

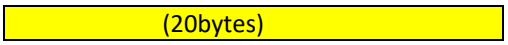
Structure :

```
struct student
{
      int roll;
      char name[20];
      int marks;
};
```



Union:

```
union student
{
      int roll;
      char name[20];
      int marks;
};
```



We know that all the members of a union use the same location although it may contain many members of different type.

A union may be a member of a structure, and a structure may be a member of an union. Moreover, structures and unions may be freely mixed with arrays.

b. Explain dynamic memory allocation/deallocation in C with syntax/example. 6

Ans:

Dynamic Memory Allocation/De-allocation functions in C are:

malloc( ), calloc( ), realloc() and free()

Related header file is stdlib.h>

These functions provide a complete set of memory allocation, reallocation, deallocation, and heap management tools.

malloc()

Syntax:

ptr = (cast-type*) malloc(byte-size);

Here, *ptr* is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

Example:

ptr = (int*) malloc(100 * sizeof(int));

calloc()

The name calloc stands for "contiguous allocation". The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

Syntax:

ptr = (cast-type*)calloc(n, element-size);

Example:

ptr = (float*) calloc(25, sizeof(float));

realloc()

If the previously allocated memory is insufficient or more than required, you can change the previously allocated memory size using realloc().

Syntax

ptr = realloc(ptr, newsize);

Example:

ptr = realloc(ptr, newsize);

free():

The free subroutine deallocates a block of memory previously allocated by the malloc subsystem. Undefined results occur if Pointer is not an address that has previously been allocated by the malloc subsystem, or if Pointer has already been deallocated. If Pointer is NULL, no action occurs.

Syntax

void free (Pointer)

Example:

free(p); /* p is pointer */

8.

a. Convert the infix expression ((a/(b-c+d))*(e-a)*c) to postfix expression and evaluate that postfix expression for given data a=6, b=3, c=1, d=2, e=4. 5

Ans:

Given Infix expression:

((a/(b-c+d))*(e-a)*c)

For postfix in steps: underlined expression becomes operand

((a/(**bc-** +d)) * (e-a) * c)

((a/ **bc-d+** **) * (e-a)*c)**

(abc-d+/ * (e-a)*c)

(abc-d+/ * ea- * c)

(abc-d+/ea-* * c)

abc-d+/ea-*c*

Trace for given data a=6, b=3, c=1, d=2, e=4

a b c – d + / e a - * c *

6 3 1 – 2 + / 4 6 - * 1 *

6 (3-1) 2 + / 4 6 - * 1 *

6 (2+2) / 4 6 - * 1 *

(6 / 4) 4 6 - * 1 *

1 (4-6) * 1 *

(1 * -2) 1 *
(-2 *1)
-2 Ans

b. Write a function to evaluate postfix expression **5**

Ans:

Function to evaluate postfix:

```
#define SIZE 50            /* Size of Stack */
int s[SIZE]; int top=-1;       /* Global declarations */
push(int elem) { s[++top]=elem; }
int pop() {  return(s[top--]); }
void evalpostfix(char pofx[50])
{
  char ch;
  int i=0,op1,op2;
  while( (ch=pofx[i++]) != '\0')
  {
    if(isdigit(ch)) push(ch-'0'); /* Push the operand */
    else
      {        /* Operator,pop two  operands */
        op2=pop();    op1=pop();
        switch(ch)
         {
              case '+':push(op1+op2);break;
              case '-':push(op1-op2);break;
              case '*':push(op1*op2);break;
              case '/':push(op1/op2);break;
              case '%':push(op1%op2);break;
              case '^':push(pow(op1,op2));break;
             default : printf("Invalid operator\n");
         }
      }
  }
  printf("\n Given Postfix Expn: %s\n",pofx);
  printf("\n Result after Evaluation: %d\n",s[top]);
}
```