Solution to the questions of 3rd Sem CSE-C-DSA-IAT1

Faculty : Dr. P. N. Singh, Professor(CSE)

1. **Write a C program to multiply 2 matrices & to display the resultant matrix .**

```c
#include<stdio.h>
#define m 4
#define n 3
#define p 4

int main()
{
	int i,j,k,m1[m][n],m2[n][p],m3[m][p];
	printf("Enter matrix 1 : %d x %d\n",m,n);
	for(i=0;i<m;i++)
		for(j=0;j<n;j++)
			scanf("%d",&m1[i][j]);

	printf("Enter matrix 2 : %d x %d\n",n,p);
	for(i=0;i<n;i++)
		for(j=0;j<p;j++)
			scanf("%d",&m2[i][j]);
	/* Now multiplying */
	for(i=0;i<m;i++)
		for(j=0;j<p;j++)
		{
			m3[i][j]=0;
			for(k=0;k<n;k++)
				m3[i][j]+=m1[i][k]* m2[k][j];
		}

	printf("Resultant matrix:\n");
	for(i=0;i<m;i++)
	{
		for(j=0;j<p;j++)
			printf(" %d",m3[i][j]);
		printf("\n");
	}
	return(0);
}
```

2. **Explain Dynamic Memory Allocation/De-allocation with in-built functions malloc(), calloc(), realloc() and free().**

   Dynamic Memory allocation and de-allocation:
   Memory can be allocated/ de-allocated at run-time as and when required. C has four in-built functions for the same with header file stdlib.h
   malloc(), calloc() and realloc() to allocate and free() to de-allocate

   malloc()
   The name malloc stands for "memory allocation". The function malloc() reserves a block of memory of specified size in bytes and return a pointer of type void which can be casted into pointer of any form.
   Syntax of malloc()
   ```
   ptr=(cast-type*)malloc(byte-size);
   ```
   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 for calloc():
   ```
   ptr=(cast-type*)calloc(n,element-size);
   ```
   example:
   ```
   ptr=(int*)calloc(25,sizeof(int));
   ```

   realloc()
   If the previously allocated memory is insufficient or more than sufficient. Then, you can change memory size previously allocated using realloc().
   Syntax/example:
   ```
   ptr=realloc(ptr, newsize);
   ```

   free()
   Dynamically allocated memory with either calloc() or malloc() does not get return on its own. The programmer must use free() explicitly to release space.
   Syntax/example:
   ```
   free(ptr);
   ```

3. **What is a sparse matrix? How it can be converted to a concise matrix? Explain with example.**

   Sparse matrix is a 2-dimentional array where relative number of elements are zero (approx 2/3rd). It takes more storage (sometimes without useful elements)

	0	1	2	3	4	5
0	0	0	0	2	0	0
1	0	7	0	0	0	0
2	0	0	0	0	0	0
3	3	0	4	0	0	0
4	0	0	0	0	5	0
5	0	0	0	0	0	9
6	0	0	0	0	8	0

Sparse matrix

A sparse matrix can be represented by a concise/dense matrix by following way.

- Number of columns will be 3
- Number of rows will be number of non-zero elements + 1

In converted matrix:
1st row will contain total number of rows, total number of columns and total number of non-zero elements of sparse matrix
Rest of the row will contain corresponding row number, column number containing the non-zero element in sparse matrix and non-zero elements itself

Now sparse matrix converted to concise matrix

	0	1	2
0	7	6	7
1	0	3	2
2	1	1	7
3	3	0	3
4	3	2	4
5	4	4	5
6	5	5	9
7	6	4	8

4. A) Convert following infix notation **a+(b-c)/d*e-f** to prefix & postfix notations.
   B) Evaluate postfix expression **8 2 6 + 9 3 / - *** (having single digit operands) showing stack status.

   A)
   a + (b-c) / d * e - f
   For prefix
   ➔ a + **-bc** / d * e - f
   ➔ a + **/-bcd** * e – f
   ➔ a + ***/-bcde** – f
   ➔ **+a*/-bcde** – f
   ➔ -+a*/-bcdef Ans

a + (b-c) / d * e - f
For postfix

- ➔ a + **bc-** / d * e - f
- ➔ a + **bc-d/** * e – f
- ➔ a + **bc-d/e*** - f
- ➔ **abc-d/e*+** - f
- ➔ **abc-d/e*+f-** **Ans**

B) Evaluating **8 2 6 + 9 3 / - ***

8 2 6 + 9 3 / - *

Operand /Operator	Action	Operation	Result	Statck status
8	push 8			8
2	push 2			8 2
6	push 6			8 2 6
+	pop 6 and 2	2 + 6	8	8
	push 8			8 8
9	push 9			8 8 9
3	push 3			8 8 9 3
/	pop 3 and 9	9 / 3	3	8 8
	push 3			8 8 3
-	pop 3 and 8	8 - 3	5	8
	push 5			8 5
*	pop 5 and 8	8 * 5	40	
	push 40			40

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

```
long int moves=0; /* to count the moves */
void hanoi(int d, char l[ ], char r[ ], char m[ ]) /* or char *l, char *r, char *m */
{
        if(d>0)
        {
                hanoi(d-1,l,m,r);
               /* recursion with sequence left, middle & right */
                printf("%ld. Move disk %d from %s to %s\n",++moves,d,l,r);
                hanoi(d-1,m,r,l);
                  /* recursion with sequence middle, right & left */
        }
}
```

```
main()

{

        int d;

        printf(“Enter number of disks : “);

        scanf(“%d”, &d);

        hanoi(d,"left","right","middle");

        return (0);

}
```

Expected Output:

```
Enter number of 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
```

6. What is **Ackermann’s function?** Write with example program to solve it.

Ackermann Function – Ackermann function is one of the simplest example of a well defined total function which can be solved by recursive function and so it is compatible but not primitive recursive.
It grows faster than an exponential function
Ackerman function ack(x,y) can be defined for integer x and y as following:

$$\text{ack}(x , y) = \boxed{\begin{array}{ll} y + 1 & \text{if } x=0 \\ \text{ack}(x-1,1) & \text{if } y=0 \\ \text{ack}(x-1,\text{ack}(x,y-1)) & \text{otherwise,} \end{array}}$$

Now implementation in C programming language

```
#include<stdio.h>

#include<stdlib.h>

int count = 0;

int ack(int x, int y)

{

count++;

if(x==0)  return y+1;

if(y==0) return ack(x-1,1);

return ack(x-1,ack(x,y-1));

}
```

```
int main()
{
int x,y;
            printf("Enter values for x and y : ");
            scanf("%d%d",&x,&y);
            printf("Result is %d\n",ack(x,y));
printf("\nFunction called %d times.\n",count);
            return (0);
}
```

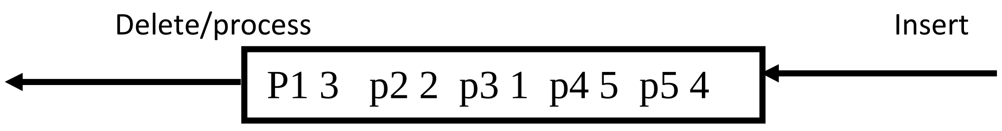
7. **Write a C function/algorithm to insert an element in a circular queue showing an alert when Circular Queue is full.**

```
#define MAX 10
int rear=-1,front=-1;

void insertCQ ( int arr[], int item )
{
            if ( ( rear == MAX - 1 && front == 0 ) || (  rear + 1 == front ) )
             /* when rear comes just behind front – adjacent */
            {
               puts("\nQueue is full" );
                return ;
                /* to return the control only so else will not be written*/
            }
            rear=(rear+1)%MAX;
             arr[rear] = item ;
            if ( front == -1 )
                        front = 0 ;
             /* when queue has been inserted an item */
}
```

8. **Explain priority queue & write algorithm/code to sort the jobs with priority number in a structure.**

Priority queue is like a queue, but where additionally each element has a “priority” associated with it. For example jobs/processes in operating system can be processed according to its priority. In real life example also some objects has priority to get itself processed/operated/deleted first.



A priority Queue with processes and its priority number

In a priority queue, an element with high priority (generally the least priority number will have highest priority) is served before an element with low priority. So, items in priority queue are sorted according its priority. If two elements have the same priority, they are served according to their order in the queue.

```
void add ( struct data dt )
{
        struct data temp;
        int i,j;
        if ( rear == MAX - 1 )   { printf("\nQueue is full") ;    return ;         }
        rear++ ;
        d[rear] = dt ;
        if ( front == -1 )          front = 0 ;
/* Now sorting according to priority number */
        for ( i = front ; i <= rear ; i++ )
            for ( j = i + 1 ; j <= rear ; j++ )
          if ( d[i].prno > d[j].prno )
                  {
                    temp = d[i] ;
                    d[i] = d[j] ;
                    d[j] = temp ;
                   }
}
```