Experiment No. 1

Design, Develop and Implement a menu driven Program in C for the following Array operations
a. understand the concept of array storage, size of a word.
b. Find the address of element and verify it with the theoretical value.
Support the program with functions for each of the above operations.

About the address of array:

Address calculation in 2-D array: suppose A is a 2D array with dimensions M X N. So that here total number of elements is equal to M X N. Address should be calculated using the following method:
Location (A [I, J]) = Base (A) + w [N (I-1) + (J-1)]
Here, I and J indicates index value of the element whose address you want to search. Base (A) is the starting address of the given array. W is the memory word required to store given array's element. Here, array is the of size M(rows) X N(columns).

Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10][10],b[10],s,t,u,i,j,n,m;
unsigned int od,td;
printf("enter 1d array");
for(i=0;i<=10;i++)
scanf("%d",&b[i]);
printf("enter rows n coulmn");
scanf("%d%d",&m,&n);
printf("enter 2d array");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("enter the nth element whose address u want to print in 1d array");
scanf("%d",&s);
printf("enter for 2d array");
scanf("%d%d",&t,&u);
od=b+2*(s-1);
td=a+((t-1)*n+(u-1))*2;
printf("for 1d array=%u",od);
printf("for 2d array=%u",td);
}
```

Experiment No. 2

Design, Develop and Implement a menu driven Program in C for the following Array operations using functions
a. Linear Search
b. Bubble Sort

```
#include<stdio.h>
liner_search(int a[],int n)
{
   int item,i;
   printf("\nEnter item to search: ");
   scanf("%d", &item);
   for (i=0; i<n; i++)
   {
   if (item == a[i])
      {
      printf("\nItem found at location %d", i+1);
      break;
      }
   }
   if (i==n)
   printf("\nItem does not exist.");
}

void bubblesort(int a[],int n)
{
   int temp,i,j;
   for(i=0;i<n;i++)
   {
   for(j=0;j<n-1;j++)
   {
   if(a[j]>a[j+1])
   {
    temp=a[j];
    a[j]=a[j+1];
    a[j+1]=temp;
   }
  }
 }
 printf("\n The sorted array is\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}

}
void main()
```

```
{
int a[10], i,n,ch;
printf("\nEnter number of elements of an array:\n");
scanf("%d",&n);
printf("\nEnter elements: \n");
for (i=0; i<n; i++)
scanf("%d", &a[i]);
do
{
printf("\n\n----MAIN MENU----\n");
printf("\n1. search");
printf("\n2. sort");
printf("\n3. Exit (End the Execution)");
printf("\nEnter Your Choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1:
liner_search(a,n);
break;
case 2:
bubblesort( a,n);
break;
case 3:
exit(0); break;
default:
printf("\nEND OF EXECUTION");
}//end switch
}while (ch != 3);
}
```

Experiment No. 3

Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

a. *Push* an Element on to Stack
b. *Pop* an Element from Stack
c. Demonstrate how Stack can be used to check *Palindrome*
d. Demonstrate *Overflow* and *Underflow* situations on Stack
e. Display the status of Stack
f. Exit

Support the program with appropriate functions for each of the above operations

About Stack:

The stack abstract data type is defined by the following structure and operations. A stack is structured, as described above, as an ordered collection of items where items are added to and removed from the end called the "top." Stacks are ordered LIFO. The stack operations are given below.

- Stack() creates a new stack that is empty. It needs no parameters and returns an empty stack.
- push(item) adds a new item to the top of the stack. It needs the item and returns nothing.
- pop() removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.
- peek() returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.
- isEmpty() tests to see whether the stack is empty. It needs no parameters and returns a boolean value.
- size() returns the number of items on the stack. It needs no parameters and returns an integer.

Code:

```
#include<stdio.h>
#define MAX 4
int stack[MAX], item;
int ch, top = -1, count = 0, status = 0;
/*PUSH FUNCTION*/
void push(int stack[], int item)
{
if (top == (MAX-1))
printf("\n\nStack is Overflow");
else
{
stack[++top] = item;
status++;
}
}
/*POP FUNCTION*/
int pop(int stack[])
{
int ret;
if(top == -1)
printf("\n\nStack is Underflow");
else
{
ret = stack[top--];
status--;
printf("\nPopped element is %d", ret);
```

```
}
return ret;
}
/* FUNCTION TO CHECK STACK IS PALINDROME OR NOT */
void palindrome(int stack[])
{
int i, temp;
temp = status;
for(i=0; i<temp; i++)
{
if(stack[i] == pop(stack))
count++;
}
if(temp==count)
printf("\nStack contents are Palindrome");
else
printf("\nStack contents are not palindrome");
}
/*FUNCTION TO DISPLAY STACK*/
void display(int stack[])
{
int i;
printf("\nThe stack contents are:");
if(top == -1)
printf("\nStack is Empty");
else{
for(i=top; i>=0; i--)
printf("\n ------\n| %d |", stack[i]);
printf("\n");
}
}
/*MAIN PROGRAM*/
void main()
{
do{
printf("\n\n----MAIN MENU----\n");
printf("\n1. PUSH (Insert) in the Stack");
printf("\n2. POP (Delete) from the Stack");
printf("\n3. PALINDROME check using Stack");
printf("\n4. Exit (End the Execution)");
printf("\nEnter Your Choice: ");
scanf("%d", &ch);
switch(ch){
```

```
case 1: printf("\nEnter a element to be pushed: ");
scanf("%d", &item);
push(stack, item);
display(stack);
break;
case 2: item=pop(stack);
display(stack);
break;
case 3:
palindrome(stack);
break;
case 4:
exit(0); break;
default:
printf("\nEND OF EXECUTION");
}//end switch
}while (ch != 4);
}
```

Experiment No. 4

Design, Develop and Implement a menu driven Program in C for the following operations on QUEUE of Characters (Array Implementation of Queue with maximum size MAX)
a. Insert an Element on to QUEUE
b. Delete an Element from QUEUE
c. Demonstrate Overflow and Underflow situations on QUEUE
d. Display the status of QUEUE
e. Exit
Support the program with appropriate

```
#include <stdio.h>
#define MAX 10

int QUEUE[MAX],front=-1,rear=-1;

/** function       :   insert_in_Q(),
to push an item into queue.
**/
void insert_in_Q(int queue[],int ele)
{
   if(rear==-1)
   {
      front=rear=0;
      queue[rear]=ele;
   }
   else if(rear==MAX-1)
   {
      printf("\nQUEUE is full.\n");
      return;
   }
   else
   {
      rear++;
      queue[rear]=ele;
   }
   printf("\nItem inserted..");
}

/** function    :   display_Q(),
to display queue elements
**/

void display_Q(int queue[])
{       int i;
   if(rear==-1) { printf("\nQUEUE is Empty."); return; }
   for(i=front;i<=rear;i++)
   { printf("%d,",queue[i]); }

}
```

```
/** function    :   remove_from_Q(),
to remove (pop) an item from queue.
**/
void remove_from_Q(int queue[])
{
   int ele;
   if(front==-1)
   {
      printf("QUEUE is Empty.");
      return;
   }
   else if(front==rear)
   {
      ele=queue[front];
      front=rear=-1;
   }
   else
   {
      ele=queue[front];
      front++;
   }
   printf("\nItem removed : %d.",ele);
}



int main()
{
   int ele,choice;
while(1)
{
   //clrscr();
   printf("\nQUEUE Elements are :");
   display_Q(QUEUE);
   printf("\n\nEnter choice (1:Insert,2:Display,3:Remove,0:Exit):");
   scanf("%d",&choice);
   switch(choice)
   {
      case 0:
         exit(1);
         break;
      case 1:
         printf("Enter an element to insert:");
         scanf("%d",&ele);
         insert_in_Q(QUEUE,ele);
         break;
      case 2:
         display_Q(QUEUE);
```

```
            break;
        case 3:
            remove_from_Q(QUEUE);
            break;
        default:
            printf("\nInvalid choice\n");
            break;
    }

} //end of while(1)
return 0;
}
```

Experiment No. 6

Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Branch, Sem, PhNo*
a. Create a SLL of N Students Data by using *front insertion*.
b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of SLL
d. Perform Insertion and Deletion at Front of SLL
e. Demonstrate how this SLL can be used as STACK and QUEUE
f. Exit
Code:

```
#include<stdio.h>
#include<stdlib.h>
int MAX=4, count;
struct student
{
char usn[10];
char name[30];
char branch[5];
int sem;
char phno[10];
struct student *next; //Self referential pointer.
};
typedef struct student NODE;
int countnodes(NODE *head)
{
NODE *p;
count=0;

p=head;
while(p!=NULL)
{
p=p->next;
count++;
}
return count;
}
NODE* getnode(NODE *head)
{
NODE *newnode;
newnode=(NODE*)malloc(sizeof(NODE)); //Create first NODE
printf("\nEnter USN, Name, Branch, Sem, Ph.No\n");
scanf("%s",&newnode->usn);
scanf("%s",&newnode->name);
scanf("%s",&newnode->branch);
scanf("%d",&newnode->sem);
scanf("%s",&newnode->phno);
newnode->next=NULL; //Set next to NULL...
head=newnode;
return head;
}
NODE* display(NODE *head)
```

```
{
NODE *p;
if(head == NULL)
printf("\nNo student data\n");
else
{
p=head;
printf("\n----STUDENT DATA----\n");
printf("\nUSN\tNAME\t\tBRANCH\tSEM\tPh.NO.");
while(p!=NULL)
{
printf("\n%s\t%s\t\t%s\t%d\t%s", p->usn, p->name, p->branch, p->sem, p->phno);
p = p->next; //Go to next node...
}
printf("\nThe no. of nodes in list is: %d",countnodes(head));
}

return head;
}
NODE* create(NODE *head)
{
NODE *newnode;
if(head==NULL)
{
newnode=getnode(head);
head=newnode;
}
else
{
newnode=getnode(head);
newnode->next=head;
head=newnode;
}
return head;
}
NODE* insert_front(NODE *head)
{
if(countnodes(head)==MAX)
printf("\nList is Full / Overflow!!");
else
head=create(head); //create()insert nodes at front.
return head;
}
NODE* insert_rear(NODE *head)
{
NODE *p, *newnode;
p=head;
if(countnodes(head) == MAX)
printf("\nList is Full(QUEUE)!!");
else
{
if(head == NULL)
{
newnode=getnode(head);
```

```
head=newnode; //set first node to be head
}
else
{

newnode=getnode(head);
while(p->next!=NULL)
{
p=p->next;
}
p->next=newnode;
}
}
return head;
}
NODE* insert(NODE *head)
{
int ch;
do
{
printf("\n1.Insert at Front(First)\t2.Insert at End(Rear/Last)\t3.Exit");
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1: head=insert_front(head); break;
case 2: head=insert_rear(head); break;
case 3: break;
}
head=display(head);
}while(ch!=3);
return head;
}
NODE* delete_front(NODE *head)
{
NODE *p;
if(head==NULL)
printf("\nList is Empty/Underflow (STACK/QUEUE)");
else
{
p=head;
head=head->next; //Set 2nd NODE as head
free(p);
printf("\nFront(first)node is deleted");
}
return head;
}

NODE* delete_rear(NODE *head)
{
NODE *p, *q;
p=head;
while(p->next!=NULL)
{
```

```
p=p->next; //Go upto -1 NODE which you want to delete
}
q=p->next;
free(q);//Delete last NODE...
p->next=NULL;
printf("\nLast(end) entry is deleted");
return head;
}
NODE* del(NODE *head)
{
int ch;
do{
printf("\n1.Delete from Front(First)\t2. Delete from End(Rear/Last))\t3.Exit");
printf("\nEnter your choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1: head=delete_front(head);
break;
case 2: head=delete_rear(head);
break;
case 3: break;
}
head=display(head);
}while(ch!=3);
return head;
}
NODE* stack(NODE *head)
{
int ch;
do
{
printf("\nSSL used as Stack...");
printf("\n 1.Insert at Front(PUSH) \t 2.Delete from Front(POP))\t3.Exit");
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch)
{

case 1: head=insert_front(head); break;
case 2: head=delete_front(head); break;
case 3: break;
}
head=display(head);
}while(ch!=3);
return head;
}
NODE* queue(NODE *head)
{
int ch;
do
{
printf("\nSSL used as Queue...");
printf("\n 1.Insert at Rear(INSERT) \t 2.Delete from Front(DELETE))\t3.Exit");
```

```
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1: head=insert_rear(head); break;
case 2: head=delete_front(head); break;
case 3: break;
}
head=display(head);
}while(ch!=3);
return head;
}
void main()
{
int ch, i, n;
NODE *head;
head=NULL;

printf("\n*----------Studednt Database----------*");
do
{
printf("\n 1.Create\t 2.Display\t 3.Insert\t 4.Delete\t 5.Stack\t 6.Queue\t 7.Exit");
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1: printf("\nHow many student data you want to create: ");
scanf("%d", &n);

for(i=0;i<n;i++)
head=create(head);//Call to Create NODE...
break;
case 2: head=display(head); //Call to Display...
break;
case 3: head=insert(head); //Call to Insert...
break;
case 4: head=del(head); //Call to delete
break;
case 5: head=stack(head);
break;
case 6: head=queue(head);
break;
case 7: exit(0); //Exit...
}
}while(ch!=7);
}
```