

1. Write a program to Append 2 arrays.

Algorithm

Step 1) Initialize

Let Array1 'a' and Array2 'b' be the input arrays.

Create a new ResultArray 'c' of size $n1 + n2$.

Step 2) Copy elements from Array1:

For each i from 0 to $n1 - 1$, set $c[i] = a[i]$.

Step 3) Copy elements from Array2:

For each j from 0 to $n2 - 1$, set $c[n1 + j] = b[j]$.

Step 4) Print ResultArray 'c'.

Code

```
#include<stdio.h>
int main()
{
    int aSize, bSize, mSize, i, j;
    int a[10], b[10], Merged[20];
    printf("\n Please Enter the First Array Size : ");
    scanf("%d", &aSize);
    printf("\n Please Enter the First Array Elements : ");
    for(i = 0; i < aSize; i++)
    {
        scanf("%d", &a[i]);
    }
    printf("\n Please Enter the Second Array Size : ");
    scanf("%d", &bSize);
    printf("\n Please Enter the Second Array Elements : ");
    for(i = 0; i < bSize; i++)
    {
        scanf("%d", &b[i]);
    }

    for(i = 0; i < aSize; i++)
    {
        Merged[i] = a[i];
    }

    mSize = aSize + bSize;
    for(i = 0, j = aSize; j < mSize && i < bSize; i++, j++)
    {
        Merged[j] = b[i];
    }
    printf("\n a[%d] Array Elements After Merging \n", mSize);
    for(i = 0; i < mSize; i++)
    {
        printf(" %d \t ", Merged[i]);
    }
}
```

```
return 0;
}
```

2. Implement a Stack using an array.

Algorithm

1. Check if the stack is full.
If(top==max-1)
2. If the stack is full, produce an overflow message and exit.
3. If the stack is not full, increments top to point next space.
top=top+1
4. Adds data element to the stack location, where the top points.
stack[top]=data
5. Returns success.
6. Check if the stack is empty
If(top==-1)
7. if the stack is empty produce a message underflow and exit.
8. decrement the top, top=top-1
9. display the stack

Code

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
    //clrscr();
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t-----");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                push();
```

```

break;
case 2:
pop();
break;
case 3:
display();
break;

case 4:
printf("\n\t EXIT POINT ");
break;
default:

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
}
}
while(choice!=4);
return 0;
}
void push()
{
if(top>=n-1)
{
printf("\n\tSTACK is over flow");

}
else
{
printf(" Enter a value to be pushed:");
scanf("%d",&x);
top++;
stack[top]=x;
}
}
void pop()
{
if(top<=-1)
{
printf("\n\t Stack is under flow");
}
else
{
printf("\n\t The popped elements is %d",stack[top]);
top--;
}
}
void display()
{
if(top>=0)
{

```

```

printf("\n The elements in STACK \n");
for(i=top; i>=0; i--)
printf("\n%d",stack[i]);
printf("\n Press Next Choice");
}
else
{
printf("\n The STACK is empty");
}
}

```

3. Write a program to implement stack using LinkedList

Algorithm

Push operation

1. Initialise a node
2. Update the value of that node by data i.e. **node->data = data**
3. Now link this node to the top of the linked list
4. And update the top pointer to the current node **top=top->ptr;**

Pop operation

5. First Check whether there is any node present in the linked list or not, if not then return if (top == NULL)
Stack is empty
6. Otherwise make a pointer let's say **temp** to the top node and move forward the top node by 1 step
7. Now free this temp node

Peek element

8. **top->info**

Display operation

9. print the value of node
print top->info;
10. update pointer
top1 = top->ptr;

code

```

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

```

```

struct node

```

```

{
int info;
struct node *ptr;
}*top,*top1,*temp;

```

```
int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();
int count = 0;
void main()
{
    int no, ch, e;
    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Top");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Dipslay");
    printf("\n 7 - Stack Count");
    printf("\n 8 - Destroy stack");
    create();
    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1: printf("Enter data : ");
                    scanf("%d", &no);
                    push(no);
                    break;
            case 2: pop();
                    break;
            case 3: if (top == NULL)
                    printf("No elements in stack");
                    else
                    {
                        e = topelement();
                        printf("\n Top element : %d", e);
                    }
                    break;
            case 4: empty();
                    break;
            case 5: exit(0);
            case 6: display();
                    break;
            case 7: stack_count();
                    break;
            case 8: destroy();
                    break;
        }
    }
}
```

```

default :
printf(" Wrong choice, Please enter correct choice ");
break;
}
}
}
void create()
{
    top = NULL;
}
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
}
void display()
{
    top1 = top;
    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }
    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
}
void pop()
{
    top1 = top;
    if (top1 == NULL)
    {

```

```

printf("\n Error : Trying to pop from empty stack");
return;
}
else
top1 = top1->ptr;
printf("\n Popped value : %d", top->info);
free(top);
top = top1;
count--;
}
int topelement()
{
return(top->info);
}
void empty()
{
if (top == NULL)
printf("\n Stack is empty");
else
printf("\n Stack is not empty with %d elements", count);
}
void destroy()
{
top1 = top;
while (top1 != NULL)
{
top1 = top->ptr;
free(top);
top = top1;
top1 = top1->ptr;
}
free(top1);
top = NULL;
printf("\n All stack elements destroyed");
count = 0;
}

```

4. Implement a sparse matrix using an array

Algorithm

The triplet (row, column, value) format stores only the non-zero elements.

The row index of the non-zero element.

The column index of the non-zero element.

The non-zero element itself.

Steps:

1. Input the Matrix: Get the size and elements of the matrix.

2. Count Non-Zero Elements: Traverse the matrix to find the non-zero elements.
3. Create Triplet Representation: Store the row, column, and value of each non-zero element in an array.

```
if (matrix[i][j] != 0) {  
    sparse[nonZeroCount].row = i;  
    sparse[nonZeroCount].col = j;  
    sparse[nonZeroCount].value = matrix[i][j];  
    nonZeroCount++;  
}
```

4. Output the Triplet Representation: Display the sparse matrix in triplet form.

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

```
#define MAX 100
```

```
int row; // Row index  
int col; // Column index  
int value; // Value at (row, col)  
};
```

```
int main() {  
    int matrix[10][10];  
    struct SparseMatrix sparse[MAX];  
    int i, j, rows, cols, nonZeroCount = 0;  
  
    printf("Enter the number of rows and columns of the matrix:\n");  
    scanf("%d%d", &rows, &cols);  
  
    printf("Enter the elements of the matrix:\n");  
    for (i = 0; i < rows; i++) {  
        for (j = 0; j < cols; j++) {  
            scanf("%d", &matrix[i][j]);  
  
            if (matrix[i][j] != 0) {  
                sparse[nonZeroCount].row = i;  
                sparse[nonZeroCount].col = j;  
                sparse[nonZeroCount].value = matrix[i][j];  
                nonZeroCount++;  
            }  
        }  
    }  
  
    printf("\nOriginal Matrix:\n");  
    for (i = 0; i < rows; i++) {  
        for (j = 0; j < cols; j++) {  
            printf("%d ", matrix[i][j]);  
        }  
        printf("\n");  
    }  
}
```



```

printf("\nSparse Matrix Representation (Triplet Format):\n");
printf("Row  Column  Value\n");
for (i = 0; i < nonZeroCount; i++) {
    printf("%d    %d    %d\n", sparse[i].row, sparse[i].col, sparse[i].value);
}

return 0;
}

```

5. Implement a Queue using an array

Algorithm

1. Enqueue(x):

Check if the queue is full (rear == MAX_SIZE - 1).
 If full, return "Queue Overflow".
 If the queue is empty, set both front and rear to 0.
 Otherwise, increment rear and insert x at queue[rear].

2. Dequeue():

Check if the queue is empty (front == -1 or front > rear).
 If empty, return "Queue Underflow".
 Retrieve and return the element at queue[front].
 Increment front. If front exceeds rear, reset front and rear to -1.

3. Peek():

Check if the queue is empty.
 If not, return the element at queue[front].

4. Empty():

Returns True if front == -1.

5. Full():

Returns True if rear == MAX_SIZE - 1.

Code

```

#include<stdio.h>
#include<conio.h>
#define SIZE 10
void enQueue(int);
void deQueue();
void display();
int queue[SIZE], front = -1, rear = -1;
void main()
{
    int value, choice;
    clrscr();
    while(1){
        printf("\n\n***** MENU *****\n");
        printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter the value to be insert: ");

```

```

scanf("%d",&value);
enQueue(value);
break;
case 2: deQueue();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nWrong selection!!! Try again!!!");
}
}
}
void enQueue(int value){
if(rear == SIZE-1)
printf("\nQueue is Full!!! Insertion is not possible!!!");
else{
if(front == -1)
front = 0;
rear++;
queue[rear] = value;
printf("\nInsertion success!!!");
}
}
void deQueue(){
if(front == rear)
printf("\nQueue is Empty!!! Deletion is not possible!!!");
else{
printf("\nDeleted : %d", queue[front]);
front++;
if(front == rear)
front = rear = -1;
}
}
void display()
{
if(rear == -1)
printf("\nQueue is Empty!!!");
else{
int i;
printf("\nQueue elements are:\n");
for(i=front; i<=rear; i++)
printf("%d\t",queue[i]);
}
}
}

```



```

break;
case 4: exit(0);
default: printf("\nPlease select the correct choice!!!\n");
}
}
}
void enQueue(int value)
{
if((front == 0 && rear == SIZE - 1) || (front == rear+1))
printf("\nCircular Queue is Full! Insertion not possible!!!\n");
else{
if(rear == SIZE-1 && front != 0)
rear = -1;
cQueue[++rear] = value;
printf("\nInsertion Success!!!\n");
if(front == -1)
front = 0;
}
}
void deQueue()
{
if(front == -1 && rear == -1)
printf("\nCircular Queue is Empty! Deletion is not possible!!!\n");
else{
printf("\nDeleted element : %d\n",cQueue[front++]);
if(front == SIZE)
front = 0;
if(front-1 == rear)
front = rear = -1;
}
}
void display()
{
if(front == -1)
printf("\nCircular Queue is Empty!!!\n");
else{
int i = front;
printf("\nCircular Queue Elements are : \n");
if(front <= rear){
while(i <= rear)
printf("%d\t",cQueue[i++]);
}
else{
while(i <= SIZE - 1)
printf("%d\t", cQueue[i++]);
i = 0;
while(i <= rear)
printf("%d\t",cQueue[i++]);
}
}
}

```

}

ANSHID BABUTM