> push, pop, display for stack pseudocode

```
void push(int val)
{
   if top==size-1
        printf("underflow(stack is full")
   else
      {  top=top+1
         stack[top]=value
         printf("Insertion successful")
      }
}
void pop()
{
   if top==-1
      printf("stack is empty(underflow")
   else{
      printf("deleted elements is %d", stack[top]);
      top=top-1;
   }
}
```

```
void display()
{
   int i
   if top==-1
    printf("stack is
           empty")
  for(i=top;i>=0;i--)
  {
   printf("%d", stack[i])
  }
}
```

> Infix to postfix pseudocode

```
Evaluvate_postfix(exp)
{
   Create a stack s
   for i<-0 to (length(exp)-1
    {
        if exp[i] is operand
            push(exp[i])
        else if (exp(i) is operator)
            {
               op2<-pop()
               op1<-pop()
               res<-compute(exp[i], op1, op2)
               push(res)
            }
    }
    return top of stack
}
```

Sun
11/1/24

```
// stack implementation
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10

void push(int);
void pop();
void display();

int stack[SIZE], top = -1;

int main()
{
    int value, choice;
    while(1) {
        printf("\n\n ***** MENU *****\n");
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        scanf("%d", &choice);
        switch(choice) {
            case 1: printf("Enter value to be inserted: ");
                    scanf("%d", &value);
                    push(value);
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
            default: printf("\n Wrong selection!!! Try again");
        }
    }
    return 0;
}
```

```
void push(int value) {
    if(top == SIZE-1){
        printf("\n Stack is Full!!! Insertion is not possible");
    }
    else {
        top++;
        stack[top] = value;
        printf("\n Insertion success!!!");
    }
}

void pop() {
    if(top == -1){
        printf("\n Stack is empty!!! Deletion is not possible!!!");
    }
    else {
        printf("\n Deleted : %d", stack[top]);
        top--;
    }
}

void display() {
    if(top == -1){
        printf("\n Stack is empty!!!");
    }
    else {
        int i;
        printf("Stack elements are:\n");
        for(i=top; i>=0; i++){
            printf("%d\n", stack[i]);
        }
    }
}
```

output -

```
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit

1. Enter value to be inserted: 3

Insertion Success!!!

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit

2. Deleted
Deleted: 3

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit

1.
Enter value to be inserted: 5

Insertion Success!!!

***** MENU *****
1. Push
2. Pop
3. Display
4. exit

3.
stack elements are:
9
7

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit

4.
```

> Infix to Postfix

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

struct stack
{
   int size;
   int top;
   char *arr;
};
int stackTop(struct stack *sp){
        return sp->arr[sp->top];
};
int isEmpty(struct stack *ptr)
{
    if(ptr->top==-1)
    {
      return 1;
    }
    else
    {
      return 0;
    }
}
int isFull(struct stack *ptr)
{
      if(ptr->top==ptr->size-1)
      {
        return 1;
      }
      else
      {
        return 0;
      }
}
```

```
void push (struct stack *, char val) {
    if (is full (ptr)){
        printf ("stack overflow! cannot push %d to the stack!\n",
                val);
    }
    else{
        ptr->top++;
        ptr->arr[ptr->top]= val;
    }
}
char pop(struct stack *ptr){
    if (is empty(ptr)) {
        printf ("stack underflow! cannot pop from the stack\n");
        return -1;
    }
    else{
    char val = ptr->arr[ptr->top];
    ptr->top--;
    return val;
  }
};
int precedence(char ch){
    if(ch=='*' || ch=='/')
        return 3;
    else if (ch=='+' || ch=='-')
        return 2
    else
        return 0;
  }
}
int isoperator(char ch){
    if (ch=='+' || ch=='-' || ch=='*' || ch=='/')
      return 1;
  else
      return 0;
};
```

```
char *infix to postfix(char *infix){
  struct stack *sp = (struct stack *)malloc(sizeof(struct stack));
  sp->size=10;
  sp->top=-1
  sp->arr = (char *)malloc(sp->size*sizeof(char));
  char *postfix = (char *) malloc (strlen(infix)+1) *sizeof(char);
  int i=0;
  int j= 0;
  while(infix[i] != '\0')
  {
      if (! isoperator(infix[i])) {
          postfix[j] = infix[i];
          j++;
          i++;
      }
      else{
          if (precedence(infix[i])>precedence(stackTop(sp))){
              push(sp,infix[i]);
              i++;
          }
          else{
              postfix[j]= pop(sp);
              j++;
          }
      }
  }
  while (! isEmpty(sp)){
      postfix[j]= pop(sp);
      j++;
  }
  postfix[j]='\0';
  return postfix;
}
```

```
int main()
{
    char *infix = "x-y/z-k*d";
    printf("postfix is %s", infixtopostfix(infix));
    return 0;
}
```

output:

expression

~~postfix is xyz/-kd*-~~

01/01/24

~~output:~~ ~~expression = "x-y/z-k*d";~~

~~postfix is xyz/-kd*-~~ 8/1/2024

Queue

```
void display(){
    if(front == -1 || front > rear){
        printf("\n Queue is empty \n");
    }
    else{
        printf("Queue is :\n");
        for(int i=front; i<=rear; i++){
            printf("%d", Queue_array[i]);
        }
        printf("\n");
    }
}

void insert(){
    int add_item;
    if(rear==MAX-1){
        printf("Queue overflow\n");
    }
    else{
        if(front==-1){
            front=0;
        }
        printf("enter the element\n");
        scanf("%d", &add_item);
        rear++;
        Queue_array[rear]=add_item;
    }
}
```

```
void delete(){
    if(front==-1 || front>rear){
        printf("Queue underflow\n");
        return;
    }
    else{
        printf("deleted element: %d\n",queue_array[front]);
        front++;
    }
}
```

output -

```
1. Insert 2 Delete 3. Display 4. Exit
enter your choice
1
enter the element
2
1. Insert 2 Delete 3. Display 4 Exit
enter your choice
1
enter the element
3
1. Insert 2. Delete 3. Display 4. Exit
Enter your choice
3
Queue is
2 3
1. Insert 2. Delete 3. Display 4. Exit
enter your choice
4.
```

circular queue

```
int isEmpty(){
    return (front==-1);
}
int isFull(){
    return((rear+1)%MAX==front);
}
void display(){
    if(isEmpty()){
        printf("\n queue is empty\n");
    }else{
        printf("front-->%d\n",front);
        printf("Queue is:\n");
        int i=front;
        do{
            printf("%d",queue_array[i]);
            i=(i+1)%MAX;
        }while(i!=(rear+1)%MAX);
        printf("\n Rear-->%d\n",rear);
    }
}

void insert(){
    if(isFull()){
        printf("Queue Overflow\n");
    }else{
        if(isEmpty()){
            front=0;
        }
        printf("Enter the element\n");
        scanf("%d",&queue_array[(rear+1)%MAX]);
        rear=(rear+1)%MAX;
    }
}
```

```
void delete()
{
  if(isEmpty())
  {
    printf("Queue underflow\n");
  }
  else
  {
    printf("Deleted element: %d\n",Queue_array[front]);
    if(front == rear)
    {
      front = -1;
      rear = -1;
    }
    else
    {
      front = (front+1) % MAX;
    }
  }
}
```

Output -

1. Insert 2. Delete 3. Display 4. Exit
Enter your choice
1
Enter the element
2
1. Insert 2. Delete 3. Display 4. Exit
Enter your choice
1
Enter the element
3
1. Insert 2. Delete 3. Display 4. Exit
Enter your choice
1
Enter the element
4
1. Insert 2. Delete 3. Display 4. Exit
Enter your choice
1
Queue overflow

1. Insert 2. Delete 3. Display 4. Exit
enter your choice
2
Deleted element: 43
1. Insert 2. Delete 3. Display 4. Exit
Enter your choice
2
Deleted element: 4
1. Insert 2. Delete 3. Display 4. Exit
enter your choice
2
Queue underflow

08/01/24

Linked List

```
struct Node
{
    int data;
    struct Node *Next;
}*head = NULL;

void insert_first(struct Node *p, int x)
{
    struct Node *t;
    t = (struct Node *)Malloc(sizeof(struct Node));
    t->next = head;
    t->data = x;
    head = t;
}

void insert_last(struct Node *p, int x)
{
    while(p->next != NULL)
        p = p->next;
    struct Node *t;
    t = (struct Node *)Malloc(sizeof(struct Node *));
    t->data = x;
    p->next = t;
    t->next = NULL;
}

void insert_pos(struct Node *p, int x, int pos)
{
    struct Node *q, *t;
    t = (struct Node *)Malloc(sizeof(struct Node*));
    t->data = x;
    for(int i=1; i<pos; i++){
        q = p;
        p = p->next;
    }
    t->next = p;
    q->next = t;
}
```

```
void delete_pos(struct Node *p)
{
    if(head->next == NULL)
    {
        free(head);
        return;
    }
    head = head->next;
    free(p);
}

void delete_last(struct Node *p)
{
    struct Node *q;
    while(p->next != NULL){
        q = p;
        p = p->next;
    }
    q->next = NULL;
    free(p);
}

void delete_pos(struct Node *p, int pos)
{
    struct Node *q;
    for(int i=0; i<pos; i++){
        q = p;
        p = p->next;
    }
    q->next = p->next;
    free(p);
}

void display(struct Node *p)
{
    while(p->next != NULL)
    {
        int x = p->data;
        printf("%d\t", x);
        p = p->next;
    }
```

output -

Original Linked List : 1 -> 2 -> 3 -> 4 -> 5 -> NULL

After deleting the first element: Linked List : 2 -> 3 -> 4 -> 5 -> NULL

After deleting element 3 : Linked List: 2 -> 4 -> 5 -> NULL

After deleting last element: Linked list : 2 -> 4 -> NULL

22/01/24

concatenation

29/1/2024

```
void concatenate(struct node *a, struct node *b)
{
    if(a != NULL && b != NULL)
    {
        if (a->next == NULL)
            a->next = b;
        else
            concatenate(a->next, b);
    }
    else
    {
        printf("Either a or b is NULL\n");
    }
}
struct node * concat(struct node *start1, struct node *start2)
{
    struct node *ptr;
    if (start1 == NULL)
    {
        start1 = start2;
        return start1;
    }
    if (start2 == NULL)
        return start1;
    ptr = start1;
    while(ptr->link != NULL)
        ptr = ptr->link;
    ptr->link = start2;
    return start1;
}
```

Output -

First linked list : 1 2 3
second linked list : 4 5
concatenated linked list: 1 2 3 4 5

~~original linked list~~: 5 3 8 1 6
sorted linked list: 1 3 5 6 8

> Reversing

```
struct Node *prev = NULL;
struct Node *current = head;
struct Node *next = NULL;
while(current != NULL)
{
    next = current->next;
    current->next = prev;
    prev = current;
    current = next;
}
*head = prev;
```

```
struct Node *reverse(struct Node *head)
```

output

original linked list : 5 4 3 2 1

reversed linked list : 1 2 3 4 5

> sorting

```
void insertionSort(struct Node *head)
{
    struct Node *sorted = NULL;
    struct Node *current = head;
    while(current != NULL){
        struct Node *next = current->next;
        sortedInsert(sorted, current);
        current = next;
    }
    head = sorted;
}
```

output

original linked list: 5 4 8 1 6

sorted linked list : 1 4 5 6 8

> stack implementation using singly linked list

```
struct Node *push(struct Node *head, int value)
{
    struct Node *temp = (struct node *)malloc(size of(struct Node));
    temp->data = value;
    temp->next = head;
    head = temp;
    return head;
}

struct Node *pop(struct Node *head)
{
    if(head == NULL){
        printf("stack is empty\n");
    }
    return head;
    struct node *temp = head;
    head = temp->next;
    free(temp);
    return head;
}

void display(struct Node *head){
    struct node *d = head;
    while(d != NULL){
        printf("%d -->", d->data);
        d = d->next;
    }
    printf("NULL\n");
}
```

> queue implementation using linked list

```
struct Node *enqueue(struct Node *head, int value){
    struct Node *temp = (struct Node*)malloc(size of(struct node));
    temp->data = value;
    if(head == NULL){
        temp->next = head;
        head = temp;
    }else{
        struct Node *newN = head;
        while(newN->next != NULL){
            newN = newN->next;
        }
        newN->next = temp;
        temp->next = NULL;
    }
    return head;
}

struct Node *dequeue(struct Node *head){
    if(head == NULL){
        printf("Queue is empty\n");
        return head;
    }
```

```
        struct node *tp, *head;
        head = top->next;
        free(top);
        return head;
    }

void display(struct Node *head){
    struct Node *d = head;
    while(d != NULL){
        printf("%d ", d->data);
        d = d->next;
    }
    printf("NULL\n");
}
```

output

```
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
1
Enter value to enque 5
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
3
Queue : 5 -> NULL
1. Enqueue 2. Dequeue 3. Display 4. Exit
Enter your choice
4
Exiting
```

stack output

```
1. push 2. pop 3. Display 4. Exit
Enter your choice 1
Enter value to push: 4
1. push 2. pop 3. Display 4. Exit
Enter your choice : 1
Enter value to push. 5
1. push 2 pop 3. Display 4. Exit
Enter your choice. 3
stack : 5 -> 4 -> NULL
1. push 2. pop 3. Display 4. Exit
Enter your choice: 2
1. push 2 pop 3. Display 4. Exit
Enter your choice: 2
1. push 2 pop 3. Display 4. Exit
Enter your choice: 3
stack : 4 -> NULL
1. push 2. pop 3. Display 4. Exit
Enter your choice: 4
Exiting ....
```

29.01.24

5-2-2024

Linked List

```
struct Node *insertAtFirst(struct Node *head, int data){
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if(newNode == NULL){
        printf("cannot insert as full\n");
        return NULL;
    }
    newNode->data = data;
    newNode->next = head;
    newNode->prev = NULL;
    if(head != NULL){
        head->prev = newNode;
    }
    head = newNode;
    return head;
}

struct Node *deleteNode(struct Node *head, int value){
    if(head == NULL){
        printf("List is empty\n");
        return NULL;
    }
    if(temp->data == value){
        head = temp->next;
        if(head != NULL){
            head->prev = NULL;
        }
        free(temp);
        printf("Node with value %d deleted\n", value);
        return head;
    }
    while(temp != NULL && temp->data != value){
        temp = temp->next;
    }
    if(temp == NULL){
        printf("The given number is not in list\n");
        return head;
    }
    if(temp->prev != NULL){
        temp->prev->next = temp->next;
    }
    if(temp->next != NULL){
        temp->next->prev = temp->prev;
    }
    printf("Node with value %d deleted\n", value);
    free(temp);
    return head;
}
```

output -

1 2 3 to 4

5 1 2 3 4

Node with value 3 deleted

5 1 2 4

```
1. Insert at left 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter value: 1
1. Insert at left 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter value to be inserted: 2
1. Insert at left 2. Delete 3. Display 4. Exit
Enter your choice: 1
Enter value to be inserted: 3
1. Insert at left 2. Delete 3. Display 4. Exit
Enter your choice: 3
3->2->1->NULL
1. Insert at left 2. Delete 3. Display 4. Exit
Enter your choice: 2
Enter value to be deleted: 2
1. Insert at left 2. Delete 3. Display 4. Exit
Enter your choice: 3
3->1->NULL
1. Insert at left 2. Delete 3. Display 4. Exit
Enter your choice: 4
```

$9/2/2024

Binary Search Tree

```
struct Node *createNode(int data){
    struct Node *n = (struct Node *) malloc(sizeof(struct Node));
    n->data = data;
    n->left = NULL;
    n->right = NULL;
    return n;
}

void preOrder(struct Node *root){
    if(root != NULL){
        printf("%d", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(struct Node *root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        printf("%d", root->data);
    }
}

void inOrder(struct Node *root){
    if(root != NULL){
        inOrder(root->left);
        printf("%d", root->data);
        inOrder(root->right);
    }
}
```

```
void insert(struct Node *root, int key){
    struct Node *prev = NULL;
    while(root != NULL){
        prev = root;
        if(key == root->data){
            printf("cannot insert %d, already in BST", key);
            return;
        }
        else if(key < root->data){
            root = root->left;
        }
        else{
            root = root->right;
        }
    }
}
```

output -

5 3 1 4 6 } for preOrder

1 4 3 6 5 } for postOrder

1 3 4 5 6 } for inOrder.

19.02.24

BFS

```
void bfs(int a[10][10], int n, int u) {
    int f, r, q[10], v;
    int s[10] = {0};
    printf("The nodes visited from %d: ", u);
    f = 0;
    r = -1;
    q[++r] = u;
    s[u] = 1;
    printf("%d", u);
    while (f <= r) {
        u = q[f++];
        for (v = 0; v < n; v++) {
            if (a[u][v] == 1 && s[v] == 0) {
                printf(" %d", v);
                s[v] = 1;
                q[++r] = v;
            }
        }
    }
    printf("\n");
}
```

output - Enter no. of nodes : 4

Enter the adjacency Matrix:

0	1	1	0
1	0	1	0
1	1	0	1
0	0	1	0

The nodes visited from 0: 0 1 2 3
The nodes visited from 1: 1 0 2 3
The nodes visited from 2: 2 0 1 3
The nodes visited from 3: 3 2 0 1

DFS

```
void dfs(int a[10][10], int n, int u, int visited[]) {
    int v;
    printf("%d ", u);
    visited[u] = 1;
    for (v = 0; v < n; v++) {
        if (a[u][v] == 1 && !visited[v]) {
            dfs(a, n, v, visited);
        }
    }
}
```

output -

Enter number of vertices : 4
Enter the adjacency matrix :

0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

DFS Traversal : 0 1 2 3

```
int main() {
    int n, a[10][10], source, i, j;
    int visited[10] = {0};
    printf("Enter the no. of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency - Matrix: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }
}
```

```
printf("DFS Traversal");
for(source=0; source<n; source++){
    if(!visited[source]){
        dfs(a,n,source,visited);
    }
}
return 0;
}
```

BFS

```
int main(){
  int n,a[10][10],i,j;
  printf("\n Enter no. of nodes: ");
  scanf("%d",&n);
  printf("\nEnter adjacency Matrix\n");
  for(i=0;i<n;i++){
      for(j=0;j<n;j++){
          scanf("%d",&a[i][j]);
      }
  }
for(int source=0;source<n;source++){
    bfs(a,n,source);
  }
}
```

26.02.24