classmate
Date
Page

Index

Sign

1)	21/12/2023	Stack Implementation	21/12
2)	28/12/2023	• Infix to Postfix • Evaluation of Postfix • Queue Implementation. • Circular Queue	8/1/24

INDEX

Name Pannaga R. Bhat Std III Sem Sec D

Roll No. 189 Subject DS School/College BMSCE

Sl. No.	Date	Title	Page No.	Teacher Sign/ Remarks
1	11/1/2024	Insertion in Linked List	10	18/1/24
2	18/01/2024	Deletion of Linked List Minstack [Leet code] Reverse [leet code]	10	18/1
3.	25/01/2024	Link list operations, stack & Queue implementation	10	25/1
4.	1/2/2024	Doubly Linked List. Linked list into parts	10	1/2
5.	15/2/2024	Binary search Tree Rotate Linked list	10	20/2
6.	22/2/2024	BFS, DFS	10	1/3
7.	29/2/2024	Linear Probing	10	

10

* Stack Implementation. (Push, pop, display).

```
#include<stdio.h>
#include<stdlib.h>
#define size 10

void push(int i; int a[]);
void pop( );
void display(int a[]);

int top = -1;
int a[size]; int i;

void main( ) {
int choice;

while(1) {
    printf("Enter 1 for push operations\n");
    printf("Enter 2 for pop operations\n");
    printf("Enter 3 for display\n");
    printf("Enter 4 to exit\n");
    scanf("%d", &choice);
```

```
switch(choice) {
    case 1:
        printf("Enter the element to be pushed\n");
        scanf("%d", &i);
        push(i, a);
        break;

    case 2:
        pop();
        break;

    case 3:
        display(a);
        break;
    case 4:
        printf("Program ends succesfully\n");
        exit(0);
    }
  }
}

void push(int i, int a[]) {
    if (top == size-1) {
        printf("Stack overflow, cannot insert a element
                into a stack\n");
        exit(0);
        }
    else {
        top=top+1
        a[top] = i;
        printf("Element pushed succesfully\n");
        return;
    }
}
```

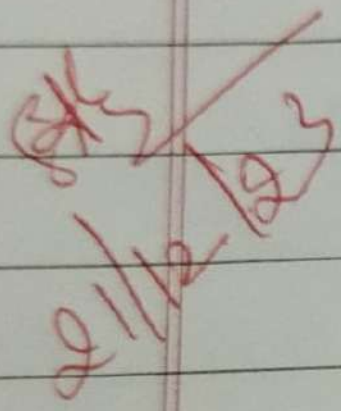
```
void pop() {
  if (top == -1) {
    printf("stack underflow, cannot pop a
            element from a stack\n");
    exit(0);
  }
  else {
    top=top-1;
    printf("Element popped successfully\n");
    return;
  }
}

void display(int a[]) {
    if (top == -1){
      printf("Stack Underflow");
      exit(0);
    }
    else {
      printf("Elements in the stack\n");
      for(int j=0; j<= top; j++)
        {
          printf("%d\n", a[j]);
        }
        return;
    }
}
```

output:

```
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
1
Enter the element to be pushed
10
Element pushed succesfully
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
1
Enter the element to be pushed
20
Element pushed succesfully
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
2
Element popped succesfully.
Enter 1 for push operations
Enter 2 for pop operations
Enter 3 for display
Enter 4 to exit
3
Elements in the stack
10
Enter 1 for push operations
Enter 2 for pop operations
```

Enter 3 for display
Enter 4 to exit
4
Program ends succesfully



28/12/2023

* Program to convert infix expression to postfix expression.

```
#include<stdio.h>
#include<ctype.h>
#define max 15

void push(char a);
char pop();
int precedence(char b);
```

```
char stack[max];
int top=-1;

void main()
{
  char infix[max], p, postfix[max];
  printf("Enter the infix expression: ");
  scanf("%s", infix);
  int j=0;
  push('(');
  for(int i=0; i<strlen(infix); i++)
  {
    if(isalnum(infix[i])) {
      postfix[j] = infix[i];
      j++;
    }
    else if(infix[i]=='+' || infix[i] == '-' || infix[i]
          == '*' || infix[i] == '\' || infix[i] == '^'
    {
      if(precedence(infix[i]) > precedence(stack[top])
        push(infix[i]);
      else if(precedence(infix[i]) <= precedence(
                                    stack[top])
      {
        while(1) {
        p= pop();
          if(p=='(') {
            push(p);
            break;
          }
          postfix[j]=p;
        } j++;
```

```
            push(infix[i]);
        }
    }
}
while(top != -1) {
    char y = pop();
    if(y == '(')
        break;
    postfix[j] = y;
    j++;
}
postfix[j] = '\0';
printf("%s", postfix);
}

void push(char a) {
    if(top == max-1) {
        printf("stack overflow");
        exit(0);
    }
    else {
        stack[++top] = a;
    }
}

char pop() {
    if(top == -1) {
        printf("stack underflow");
        exit(0);
    }
    else
        return stack[top--];
}
```

```
int precedence(char b){
    if (b=='^')
        return 3;
    else if (b=='/' || b=='*')
        return 2;
    else if(b=='+' || b=='-')
        return 1;
    else
        return 0;
}
```

- Output

Enter the infix expression: A+B*C-D+E/F
ABC*+D-EF/+

* Program to evaluate postfix expression.

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#define max 20

void push(char b);
char pop();
int eval_postfix(char a[]);
int compute(char c, int m, int n);

int top = -1;
char stack[max];
```

```
void main() {
  char postfix[max];
  printf("Enter numerical postfix expression:");
  scanf("%s", postfix);
  int r = eval_postfix(postfix);
  printf("Solution for the above postfix
          expression is %d", r);
}

int eval_postfix(char a[]) {
  int a1, b1, res;
  for(int i=0; i<strlen(a); i++) {
    if(isalnum(a[i]))
      push(a[i]);
    else if(a[i]=='+' || a[i]=='-' || a[i]=='*' ||
            a[i]=='/' || a[i]=='%') {
      a1 = pop() - '0';
      b1 = pop() - '0';

      if(a[i]=='/' && b1==0) {
        printf("Error: Dirision by zero");
        exit(1);
      }

      res = compute(a[i], b1, a1);
      push(res + '0');
    }
  }

  return stack[top] - '0';
}

void push(char b) {
  if(top == max-1) {
```

```
printf("stack overflow");
exit(0);
}
else
    stack[++top] = b;
}

char pop() {
    if(top == -1) {
        printf("stack underflow");
        exit(0);
    }
    else
        return stack[top--];
}

int compute(char c, int m, int n) {
    switch(c) {
        case '+':
            return m+n;
        case '-':
            return m-n;
        case '*':
            return m*n;
        case '/':
            return m/n;
        case '%':
            return m%n;
        default:
            return 0;
    }
}
```

Output:
Enter numerical postfix expression: 12*34*+5-
Solution for the above postfix expression is 9

* Queue Implementation

```
#include<stdio.h>
#include<stdlib.h>
# define max 6

void enqueue(int e);
int dequeue( );
void display( );

int q[max], front =-1, rear =-1, j =0;

void main( )
{
   int choice, ele;
   while (1) {
      printf("\n Enter 1 for inserting element
                  into the queue\n");
      printf("Enter 2 for deleting element from
                  the queue\n");
      printf("Enter 3 for displaying of the queue
      printf("Enter 4 to exit\n");                    \n");
      scanf("%d", &choice);
      switch(choice)
      {
         case 1:
            printf("\nEnter the element to be
                  inserted into the queue\n");
```

```
scanf("%d", &ele);
enqueue(ele);
break;

case 2:
printf("Element %d is deleted from the queue
        \n", dequeue());
break;
case 3:
display();
break;
case 4:
printf("Program ends successfully!!");
exit(0);
break;
default:
printf("Enter a valid choice.\n");
continue;
}
continue;
}
}

void enqueue(int e)
{
if(rear == max-1)
{
printf("Queue is full\n");
exit(0);
}
else {
if(rear == -1)
```

```
    {
      front = rear = 0;
      q[rear] = B;
    }
    else
    {
      rear += 1;
      q[rear] = c;
    }
    printf("Element inserted\n");
    return;
  }
}

int dequeue()
{
  if(front > rear || front == -1) {
    printf("Queue is empty");
    exit(0);
  }
  else {
    return q[front++];
  }
}

void display() {
  if(front == -1 || front > rear) {
    printf("Queue is empty\n");
    exit(0);
  }
  else {
  printf("Elements of the queue:\n");
```

```
for(int i= front; i<=rear; i++)
    printf("%d\n",q[i]);
  }
}
```

Output:

```
Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
1

Enter the element to be inserted into the queue
10
Element inserted!

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
1

Enter the element to be inserted into the queue
20
Element inserted!

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
3
```

Elements of the queue:
10
20

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
2
Element 10 is deleted from the queue

Enter 1 for inserting element into the queue
Enter 2 for deleting elemunt from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
2
Elemunt 20 is deleted from the queue

Enter 1 for inserting element into the queue
Enter 2 for deleting element from the queue
Enter 3 for displaying of the queue
Enter 4 to exit
2
Queue is empty

Sakshi
11/12/24

circular Queue Implementation.

```
#include<stdio.h>
#include<stdlib.h>
#define max 6
void enqueue(int e);
int dequeue();
void display();
int q[max], front = -1, rear = -1, j = 0;
void main() {
   int choice, ele;
   while(1) {
      printf("\n1. Insert\n");
      printf("2. Delete\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      scanf("%d", &choice);
      switch(choice)
      {
        case 1:
          printf("\nEnter the element to be
                  inserted into the queue\n");
          scanf("%d", &ele);
          enqueue(ele);
          break;
        case 2:
          printf("Element %d is deleted from
                  the queue\n", dequeue());
          break;
        case 3:
          display();
          break;
```

```
        case 4:
            printf("Program ends succesfully!!");
            exit(0);
            break;
        default:
            printf("Enter a valid choice.");
            continue;
        }
    }
}

void enqueue(int e) {
    if((rear-front) == max-1 || (front == (rear+1))) {
        printf("Queue is full.");
        exit(0);
    }
    else {
        if(front == -1)
            rear = front = 0;
        else
            rear = (rear+1) % (max);
    }
    q[rear] = e;
    printf("Element inserted!");
}

int dequeue() {
    int item;
    if(front == -1) {
        printf("Queue is empty\n");
        exit(0);
    }
```

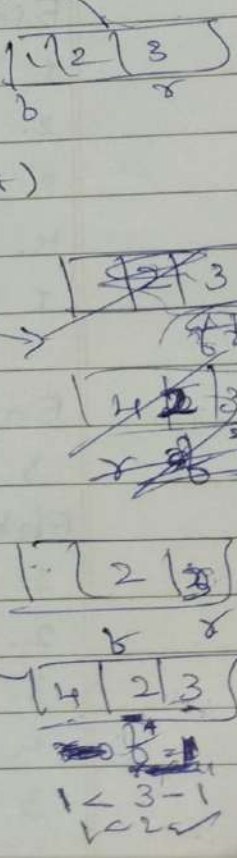
```
    item = q[front];
    if (front == rear)
        front = rear = -1;
    else
        front = (front + 1) % max;

    return item;
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    if (rear >= front)
    {
        printf("Elements of the queue:\n");
        for(int i=front; i<=rear; i++)
            printf("%d\n", q[i]);
    }
    else {
        for(int i=front; i<=max-1; i++)
            printf("%d\n", q[i]);
        for(int i=0; i<=rear; i++)
            printf("%d\n", q[i]);
    }
}
```

// if only one element is present so init to -1

then from index 0 to rear



Output:-

1. Insert
2. Delete
3. Display
4. Exit

1

Enter the element to be inserted into the queue

1

Element inserted!

1. Insert
2. Delete
3. Display
4. Exit

1

Enter the element to be inserted into the queue

2

Element inserted!

1. Insert
2. Delete
3. Display
4. Exit

1

Enter the element to be inserted into the queue

3

Element inserted!

1. Insert
2. Delete
3. Display
4. Exit

3

Elements of the queue:
1
2
3

1. Insert
2. Delete
3. Display
4. Exit

2
Element 1 is deleted from the queue

1. Insert
2. Delete
3. Display
4. Exit

1
Enter the element to be inserted into the queue
4
Element inserted!

1. Insert
2. Delete
3. Display
4. Exit

3
Elements of the queue
2
3
4

1. Insert
2. Delete
3. Display
4. Exit

2
Element 2 is deleted from the queue

1. Insert
2. Delete
3. Display
4. Exit
2
Element 3 is deleted from the queue

1. Insert
2. Delte
3. Display
4. Exit
2
Element 4 is deleted from the queue

1. Insert
2. Delete
3. Display
4. Exit
2.
Queue is empty.

11/1/2024

Sak 1/1/24

* Insertion in LinkedList.

```
#include<stdio.h>
void push(int);
void append(int);
void insert_at_pos(int);

struct Node
{
int data;
struct Node *next;
};

struct Node *head = NULL;

int choice,pos,n;

void main()
{
    while(1) {
        printf("1. Insert from beginning\n");
        printf("2. Insert at end\n");
        printf("3. Insert at specific position\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
        case 1:
            printf("Enter the insert element\n");
```

```
        scanf("%d", &n);
        push(n);
        break;
    case 2:
        printf("Enter the insert element\n");
        scanf("%d", &n);
        append(n);
        break;
    case 3:
        printf("Enter the insert element\n");
        scanf("%d", &n);
        insert_at_pos(n);
        break;
    case 4:
        display();
        break;
    case 5:
        exit(0);
    default:
        printf("Enter correct choice");
        break;
    }
    continue;
  }
}

void push(int n)
{
  struct Node* new_node = (struct Node*) malloc
                          (sizeof(struct Node));
  new_node -> data = n;
  new_node -> next = head;
```

```
    head = new_node
}

void append(int n)
{
    struct Node* new_node = (struct Node*) malloc
                            (sizeof(struct Node));
    struct Node* last = head;
    new_node -> data = n;
    new_node -> next = NULL;

    if(head == NULL)
    {
        head = new_node;
        return;
    }

    else {
        while(last -> next != NULL)
        {
            last = last -> next;
        }
        last -> next = new_node;
    }
}

void insert_at_pos(int n)
{
    int pos;
    printf("Enter the position\n");
    scanf("%d", &pos);
    struct Node* ptr = (struct Node*) malloc
                       (sizeof(struct Node));
```

```
struct Node* temp = head;
ptr -> data = n;

if(pos == 1) {
    ptr -> next = temp;
    head = ptr;
}
else {
    for(int i=1; i<pos-1 && temp != NULL; i++)
    {
        temp = temp -> next;
    }
    ptr -> next = temp -> next;
    temp -> next = ptr;
}
}

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

Output:-

1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit

Enter your choice: 1
Enter the insert element
10

1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit

Enter your choice: 2
Enter the insert element
20

1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit

Enter your choice: 2
Enter the insert element
30

1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. DEsplay
5. Exit

Enter your choice: 3
Enter the insert element

2
Enter the position
3
~~Enter your~~
1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit
Enter your choice : 4
10
20
2
30
1. Insert from beginning
2. Insert at end
3. Insert at specific position
4. Display
5. Exit
Enter your choice : 5

12/1/24

Date: / /

18/11/2024. Deletion of Singly Linked List.

```
#include<stdio.h>
void append(int);
void bpop();
void lpop();
void mpop();
struct Node
{
    int data;
    struct Node *next;
};

struct Node *head = NULL;
int choice, pos, n;

void main()
{
    while(1)
    {
    printf("1. Insert at end\n");
    printf("2. Delete from beginning\n");
    printf("3. Delete from last\n");
    printf("4. Delete at particular position\n");
    printf("5. Display\n");
    printf("6. Exit\n");
}
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch(choice)
    {
     case 1:
        printf("Enter the insert element\n");
        scanf("%d", &n);
```

```
        append(n);
        break;
    case 2:
        pop();
        break;
    case 3:
        pop();
        break;
    case 4:
        mpop();
        break;
    case 5:
        display();
        break;
    case 6:
        exit(0);
    default:
        printf("Enter correct choice\n");
        break;
    }
    continue;
  }
}

void append(int n)
{
  struct Node* new_node = (struct Node*) malloc
                (sizeof(struct Node));
  struct Node * last = head;
  new_node -> data = n;
  new_node -> next = NULL;
  if(head == NULL)
```

```
    {
      head = new_node;
      return;
    }

  else {
    while(last->next != NULL)
    {
      last = last->next;
    }
    last->next = new_node;
  }
}

void pop()
{
struct Node *ptr;
  if(head == NULL)
     printf("List is empty\n");
  else
  {
    ptr = head;
    head = head->next;
    free(ptr);
    printf("1st element deleted!\n");
  }
}

void rpop()
{
  struct Node *ptr, *p;
  if(head == NULL)
```

```
        printf("List is empty \n");

    else
    {
        if(head->next == NULL)
        {
            free(head);
            head = NULL;
        }
        else
        {
            ptr = head;
            while(ptr->next != NULL)
            {
                p1 = ptr;
                ptr = ptr->next;
            }
            p1->next = NULL;
            free(ptr);
        }
        printf("Last element deleted! \n");
    }
}

void mpop()
{
    int pos;
    printf("Enter the position to be deleted\n");
    scanf("%d", &pos);
    struct Node *ptr, *ptr1;
    if(pos == -1)
    {
```

```
    ptr = head;
    free(ptr);
    head = NULL;
}
else
{
    ptr = head;
    while(pos-1 != 0)
    {
    ptr1 = ptr;
    ptr = ptr->next;
    pos--;
    }
    ptr1->next = ptr->next;
    free(ptr);
}
printf("element deleted!\n");
}

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

Output:

1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit

Enter your choice: 1

Enter the insert element

10

1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit

Enter your choice: 1

Enter the insert element

20

1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit

Enter your choice: 1

Enter the insert element

30

1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display

6. Exit
Enter your choice: 2
1st element deleted!
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 3
Last element deleted.
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 1
Enter the insert element
40
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 4
Enter the position to be deleted
2
element deleted!
1. Insert at end
2. Delete from beginning

3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 5
20
1. Insert at end
2. Delete from beginning
3. Delete from last
4. Delete at particular position
5. Display
6. Exit
Enter your choice: 6

* Minimum Stack Leet Code.

```
#include<stdio.h>
#include<stdlib.h>
#define max 1000

typedef struct {
    int top;
    int st[max];
    int min[max];
} MinStack;

MinStack * minStackCreate() {
    MinStack * stack = (MinStack*) malloc(sizeof
                                (MinStack));
    stack -> top = -1;
    return stack;
}
```

```
void minStackPush(MinStack * obj, int val) {
    if(obj->top == max-1) {
        printf("Stack full\n");
        return;
    }
    obj->st[++obj->top] = val;

    if(obj->top >=0)
    {
        if(obj->min[obj->top-1] < val)
            obj->min[obj->top] = obj->min
                                 [obj->top-1];
    else
        obj->min[obj->top] = val;
    }
    else
        obj->min[obj->top] = val;
}

void minStackPop(MinStack * obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return;
    }
    else {
        obj->top--;
    }
}

int minStackTop(MinStack * obj) {
    if(obj->top == -1)
```

// val is greater than top ele retain as it is

```
    {
        printf("Stack empty\n");
        return -1;
    }
    return obj->st[obj->top];
}

int minStackGetMin(MinStack * obj) {
    if(obj->top == -1)
    {
        printf("min stack empty");
        return -1;
    }
    return obj->min[obj->top];
}

void minStackFree(MinStack * obj) {
    free(obj);
}

int main()
{
    MinStack *, obj = minStackCreate();
    minStackPush(obj, 3);
    minStackPush(obj, 5);
    minStackPush(obj, 2);
    minStackPush(obj, 1);
    printf("Min: %d\n", minStackGetMin(obj));
    printf("Top: %d\n", minStackTop(obj));
    minStackPop(obj);
    printf("Min: %d\n", minStackGetMin(obj));
    minStackFree(obj);
    return 0;
}
```

Date: / /

top = ~~x~~ 0 ~~x~~ 2 3

Min: 1

Top: 1

Min: 2

	st	min
val = 3	3	3
val = 5	5	3
val = 2	2	2
3 < 5 val = 1	1 (top stack)	1 -> getMin

After popping top becomes 2 automatically 1, gets deleted from stk min

then freeelch()

Now min = 2

* Reverse Linked List.

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
  int data;
  struct Node *next;
};
struct Node *head;

void create()
{
  int a[] = {1,2,3,4,5};
  struct Node *t, *last;
  head = (struct Node*) malloc(sizeof(struct Node));
  head -> data = a[0];
  head -> next = NULL;
  last = head;
```

```
for(int i=0; i<sizeof(a)/sizeof(a[0]); i++)
 {
  t=(struct Node*) malloc(sizeof(struct Node));
  t->data = a[i];
  t->next = NULL;
  last->next = t;          // linking
 }
}

void reverse(int l, int r)
{
 struct Node *t, *t1, *pre, *cur, *aft;
 int left = l;
 int right = r;
 t = head;
 if(t == NULL)                          → Empty
 {
  printf("List is empty");
  exit(0);
 }
 else
 {
  if(left==1 && right==1)               → For Fi
  {
   printf("%d", head->data);
   exit(0);
  }
  else {                                → To get left
   while(left-1 != 0)                     position
   { t1 = t;
     t = t->next;
     left--;
   }

   pre = NULL;                          → Reversing
   cur = aft = t;
```

```
for(int i=1; i<=right; i++)
{
    aft = aft->next;
    cur->next = pre;
    pre = cur
    cur = aft;
}
    t1->next = pre;
    t->next = cur;
}
}
```

→ Reversing between range [l,r]

} Joining remaining list

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

```
void main()
{
create();
reverse(2,4);
display();
}
```

output:-

1
4
3
2
5

25-01-2024

Lab 6

1) WAP to implement Single Linked List with following operations: Sort the linked list, reverse, concatenation of 2 lists.

```
#include<stdio.h>

struct node {
    int data;
    struct node *next;
};
struct node *head, *head1;

void create()
{
  head = (struct node*) malloc(sizeof(struct node));
  head1 = (struct node*) malloc(sizeof(struct node));
  struct node *last, *last1;
  int a[] = {1,2,3,4};
  int b[] = {5,6,7,8};

  head -> data = a[0];
```

```
head->next = NULL;

head1->data = b[0];
head1->next = NULL;

last = head;
last1 = head1;

for(int i=1; i<sizeof(a)/sizeof(a[0]); i++)
{
  struct node *t;
  t = (struct node*)malloc(sizeof(struct node));
  t->data = a[i];
  t->next = NULL;
  last->next = t;
  last = t;
}
}
```

- ```
  void sort()
  {
    struct node *p, *p1;
    int temp;
    for(p=head1; p!=NULL; p=p->next)
    {
      for(p1=p->next; p1!=NULL; p1=p1->next)
      {
        if(p->data < p1->data)
        {
          temp = p->data;
          p->data = p1->data;
          p1->data = temp;
        }
      }
  ```
  ```

```
}
display_b();
}

• void concat()
{
struct node *n, *n1;
if(head==NULL || head1==NULL)
{
  if(head==NULL)
    display_b();
  else
    display_a();
}
else
{
  n=head
  while(n!=NULL)
  {
    n1=n;
    n=n->next;
  }
    n1->next=head1;
    display_a();
}
}

void reverse()
{
struct node *t, *pre, *cur, *aft;
printf("Elements in B before reversing");
display_b();
```

```
t = head1;
pre = NULL;
cur = aft = t;
while(cur != NULL)
{
    aft = aft -> next;
    cur -> next = pre;
    pre = cur;
    cur = aft;
}
    head1 = pre;
    printf("\n");
    printf("Elements in B after reversing\n");
    display_b();
}
```

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

-
```
void display_b()
{
    struct node *n = head1;
    printf("B:\n");
```

```
    while(n != NULL)
    {
      printf("%d\n", n->data);
      n = n->next;
    }
}

void main()
{
  create();
  display_a();
  printf("\n");
  display_b();
  printf("After Concatenation A and B:\n");
  concat();
  reverse();
  printf("After sorting\n");
  sort();
}
```

output:

A:
1
2
3
4

B:
5
6
7
8

After concatenation A and B
A:
1
2
3
4
5
6
7
8
Elements in B before reversing
B:
5
6
7
8
Elements in B after reversing
B:
8
7
6
5
After sorting
B:
8
7
6
5

2) Write to implement single Linked list to stimulate stack & queue operations.

Stack :

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();

struct node {
  int data;
  struct node *next;
};

struct node *head. = NULL;
void main()
{
int ch;
printf("Stack Implementation using linked list");
while(1)
 {
    printf("1. Insert \n");
    printf("2. Delete \n");
    printf("3. Display \n");
    printf("4. Exit\n");
    printf("Enter your choice: \n");
    scanf("%d", &ch);

    switch(ch)
    {
```

```
case 1:
    push();
    break;
case 2:
    pop();
    break;
case 3:
    display();
    break;
case 4:
    printf("Program ends successfully!");
    exit(0);
default:
    printf("Enter a valid number...\n");
}
continue;
}
}

void push()
{
int n;
printf("Enter the insert element\n");
scanf("%d", &n);
struct node *new_node = (struct node*)
                malloc(sizeof(struct node));
new_node -> data = n;
new_node -> next = NULL;
if(head == NULL)
    head = new_node;
else
{
```

```
struct node *p;
p = head;
while(p->next != NULL)
{
    p = p->next
}
p->next = new_node;
}
return;
}

void pop()
{
struct node *ptr, *p2;
if(head == NULL)
{
printf("List is empty\n");
exit(0);
}
else
{
    if(head->next == NULL)
    {
    printf("Element % deleted\n", head->data);
    free(head);
    head = NULL;
}
    else
    {
    ptr = head;
    while(ptr->next != NULL)
    {   p2 = ptr;
        ptr = ptr->next;
```

```
    }
    printf("Element %d deleted\n", ptr->data);
    p2->next = NULL;
    free(ptr);
  }
  return;
 }
}

void display()
{
  struct node *n;
  if(head == NULL)
  {
   printf("List is empty");
   exit(0);
  }
  else
  { n = head;
    while(n != NULL)
    {
      printf("%d\n", n->data);
      n = n->next;
    }
  }
}
```

Output:-

Stack Implementation using linked list

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

1

Enter the insert element

10

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

1

Enter the insert element

20

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

3

10

20

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

2

Date: / /

```
Element 10 deleted
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
3
10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
4
Program ends succesfully!
```

* Queue Implementation

```
#include<stdio.h>
#include<stdlib.h>
void enqueue();
void dequeue();
void display();

struct node {
    int data;
    struct node *next;
};

struct node *head = NULL;
```

```
void main()
{
 int ch;
 printf("Queue Implementation using linked list");
 while(1){
  printf("1. Insert \n");
  printf("2. Delete \n");
  printf("3. Display \n");
  printf("4. Exit \n");
  printf("Enter your choice: \n");
  scanf("%d", &ch);
  switch(ch)
  {
   case 1:
        enqueue();
        break;
   case 2:
        dequeue();
        break;
   case 3:
        display();
        break;
   case 4:
        exit(0);
   default:
      printf("Enter a valid number... \n");
  }
  continue;
 }
}
```

head

```
void enqueue()
{
  int n;
  printf("Enter the insert element\n");
  scanf("%d", &n);
  struct node *new_node = (struct node*) malloc
                          (sizeof(struct node));
  new_node -> data = n;
  new_node -> next = head;
  head = new_node;
}

void dequeue()
{
  if(head == NULL)
  {
    printf("List is empty");
    exit(0);
  }
  else
  {
    if(head -> next == NULL)
    {
      printf("Element %d deleted\n", head->data);
      free(head);
      head = NULL;
    }
    else {
      struct node *p, *p1;
      p = head;
      while(p->next != NULL)
      {
```

```
        p1 = p;
        p = p->next;
    }
    p1->next = NULL;
    printf("Element %d deleted\n", p->data);
    free(p);
  }
 }
}

void display()
{
  struct node *n;
  if(head == NULL)
  {
   printf("List is empty");
   exit(0);
  }
  else {
  n = head;
  while(n != NULL)
  {
     printf("%d\n", n->data);
     n = n->next;
  }
 }
}
```

Sakshi
21/1/24

* Lab Program

WAP to implement doubly link list, create a doubly linked list, insert a new node to the left of the node, delete the node based on a specific value. Display the contents of the list.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   struct node *pre;
   struct node *next;
   int data;
};
struct node *head = NULL;

void insert_left();
void delete_val();
void display();
```

```
int main()
{
int ch;
printf("1. Insert \n");
printf("2. Delete \n");
printf("3. Display \n");
printf("4. Exit \n");
while(1)
{
printf("Enter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1:
insert_left();
break;
case 2:
delete_val();
break;
case 3:
display();
break;
case 4:
exit(0);
default:
printf("Enter correct choice: \n");
break;
}
}
}
```

```
void insert_left()
{
  int val, pos;
  printf("Enter the value: ");
  scanf("%d", &val);
  printf("Enter the position: ");
  scanf("%d", &pos);
  struct node *new_node = (struct node*)
        malloc(sizeof(struct node));
  new_node->data = val;

  if(pos == 1)
  {
    new_node->pre = NULL;
    new_node->next = head;
    if(head != NULL)
    {
      head->pre = new_node;
// set the "pre" pointer of the current head
    to the new node, if it has one node
      already.
    }
    head = new_node;
}
  else
  {
    struct node *t;
    if(pos > 1)
    {
      t = head;
      while(pos-1 != 0)
      {
```

```
        t = t->next;
        pos--;
      }
      new_node->pre = t->pre;
      new_node->next = t;
      t->pre->next = new_node;
      t->pre = new_node;
    }
  }
}

void delete_val()
{
  int d, c=1;
  printf("Enter the value to be deleted: ");
  scanf("%d", &d);
  struct node *t = head;
  if(head == NULL)
  {
    printf("List is empty");
    return;
  }
  else
  {
    while(t != NULL)
    {
      if(t->data == d)
        break;
      t = t->next;
      c++;
    }
    if(t != NULL)
```

```
{
    if(c==1)
    {
      head = t->next;
      if(head != NULL)
         t->next->pre = NULL;

      free(t);
    }
    else
    {
    t->pre->next = t->next;
     if( t->next != NULL)
        t->next->pre = t->pre;

     free(t);
     }
  }
  else
     printf("Element not found\n");
  }
}

void display()
{
  struct node *t = head;
  if ( head == NULL) {
   printf("List empty");
    return;
  }
  else {
     while(t != NULL)
```

```
    {
    printf("%d\n", t->data);
    t = t->next;
    }
  }
}
```

Output

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 1
Enter the value: 1
Enter the position: 1
Enter your choice: 1
Enter the value: 2
Enter the position: 1
Enter your choice: 3
2
1
Enter your choice: 2
Enter the value to be deleted: 2
Enter your choice: 3
1
Enter your choice: 1
Enter the value: 3
Enter the position: 1
Enter your choice: 1
Enter the value: 4
Enter the position: 2
Enter your choice: 3

3
4
1
Enter your choice: 2
Enter the value to be deleted: 4
Enter your choice: 3
3
1
Enter your choice: 4

* Leet code 3 Singly Linked_list into parts

```
struct ListNode {
    int val;
    struct ListNode *next;
};

struct ListNode ** splitListToParts(struct
        ListNode * head, int k, int *returnSize)
{
    struct ListNode **arr = (struct ListNode*) calloc
            (k+1), sizeof(struct ListNode));
    int size = 0;
    struct ListNode *ptr = head, *nextl;
    *returnSize = k; int i=0;
    while(ptr) {
        size ++;
        ptr = ptr->next;
    }
    ptr = head;
    int j=0, col;
```

```
while(ptr && i<k && size) {
    col = size % (k-i) == 0 ? size / (k-i) :
                size / (k-i) + 1;
    size -= col;
    head = ptr;
    j = 0;

    while(ptr && j<col-1) {
        ptr = ptr->next;
        j++;
    }
    next1 = ptr->next;
    arr[i++] = head, ptr->next = NULL;
    ptr = next;
}
return arr;
}
```

15/2/2024 WAP to construct a binary search Tree and traverse the tree using all the methods (in-order, post-order, pre-order). and display.

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

struct node
{
    int data;
    struct node *l;
    struct node *r;
};
```

```
struct node *root = NULL;

struct node *create(int d)
{
    struct node *t = (struct node*) malloc(sizeof
                     (struct node));
    t -> data = d;
    t -> l = t -> r = NULL;
    return t;
}

struct node *insert(struct node *t, int val)
{
    if(t == NULL)
    {
        return create(val);
    }
    if(val < t -> data)
        t -> l = insert(t -> l, val);
    else if(val > t -> data)
        t -> r = insert(t -> r, val);
    return t;
}

struct node *inorder(struct node *root)
{
    if(root != NULL)
    {
        inorder(root -> l);
        printf("%d \t", (root -> data));
        inorder(root -> r);
    }
}
```

```
struct node* postorder(struct node* root)
{
    if (root != NULL)
    {
        postorder(root->l);
        postorder(root->r);
        printf("%d \t", root->data);
    }
}

struct node* preorder(struct node* root)
{
    if(root != NULL)
    {
        printf("%d \t", root->data);
        preorder(root->l);
        preorder(root->r);
    }
}

void main()
{
    int ch, d, d1;
    printf("1. Insert into BST \n");
    printf("2. Display \n");
    printf("3. Exit \n");
    while(1)
    {
    printf("\n Enter your choice: ");
    scanf("%d", &ch);
    switch(ch)
    {
```

```
case 1:
    printf("Enter the element: ");
    scanf("%d", &ch);
    switch root = insert(root, d);
    break;

case 2:
    printf("1. Preorder\n");
    printf("2. Inorder\n");
    printf("3. Postorder\n");
    printf("Choice: ");
    scanf("%d", &d1);
    switch(d1)
    {
     case 1:
        printf("Preorder Display\n");
        preorder(root);
        break;
    case 2:
        printf("Inorder Display\n");
        inorder(root);
        postor
         break;
    case 3:
        printf("Postorder Display\n");
        postorder(root);
        break;
    default:
        printf("Enter correct choice.");
        break;
    }
     break;
```

```
        case 3:
            printf("Programs ends successfully");
            exit(0);
        default:
            printf("Enter correct choice");
            break;
    }
  }
}
```

Output:-

1. Insert into BST
2. Display
3. Exit

Enter your choice : 1
Enter the element : 10

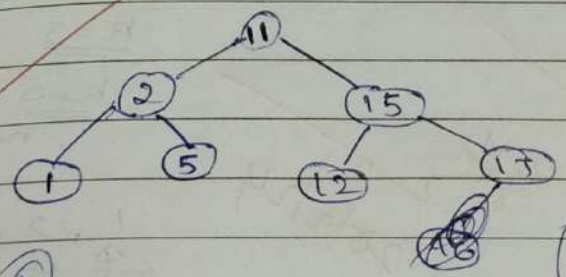
Enter your choice : 1
Enter the element : 20

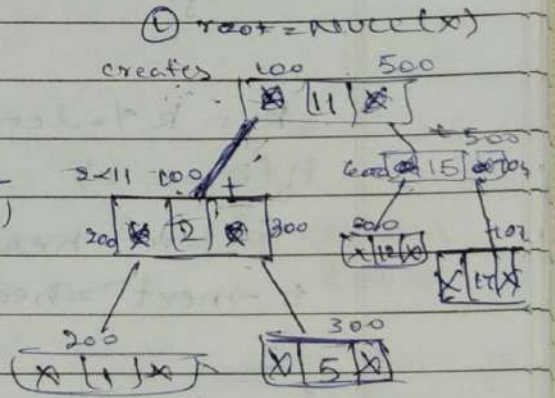
Enter your choice : 1
Enter the element : 5

Enter your choice : 1
Enter the elemunt : 2

Enter your choice : 2
1. Preorder
2. Inorder
3. Postorder
choice : 1

Preorder Display
10 2 5 20
Enter your choice: 2
1. Preorder
2. Inorder
3. Postorder
choice: 2
Inorder Display
2 5 10 20
Enter your choice: 2
1. Preorder
2. Inorder
3. Postorder
choice: 3
Postorder Display
2 5 20 10
Enter your choice: 3
Program ends succesfully.





(2) ins, 2
ins(X, 2)

(3) ins(1)
1<11
1<2
ins(X, 1)
t==X

(4) ins 5 5<11
5>2 →

(5) ins 15 15>11 r
ins(X, 15)

(6) ins 12 12>11 right
12<15 left
ins(X, 12)
t==X

(9) ins 17
17>11 right
17>15 right
ins(X, 17)
t==X create

* Rotate linked list (Leet code)

```
struct ListNode
{
  int val;
  struct ListNode *next;
};

struct ListNode* rotateRight(struct ListNode* head,
                             int k)
{   if(head == NULL || k == 0)
        return head;
    struct ListNode *t = head;
    int len = 1;
    while(t->next != NULL) {
        t = t->next;
        len++;
    }

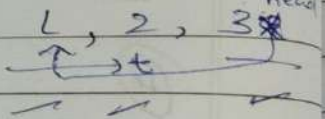
    k = k % len;
    if(k == 0)
      return head;
    t->next = head;

    t = head;
    for(int i = 0; i < len-k-1; i++)
      t = t->next;

// new head
struct ListNode *newHead = t->next;
  t->next = NULL;

return newHead;
}
```

len = 3
k = 3
k = 0, so no rotati



20/2/24

* Breadth First Search

```
#include <stdio.h>
#define r 5

int fr = 0;
int rear = 0;
int v[r];

int a[r][r] = {{0,1,0,0,1}, {1,0,1,1,0},
               {0,1,0,1,0}, {0,1,1,0,1},
               {1,0,0,1,1}};

int q[r];

void bfs(int n)
{
    int roo;
    while(fr < rear) {
        roo = q[fr++];
        printf("%d", roo);
        for(int j=0; j<r; j++)
        {
            if(a[roo][j] && !v[j]) {
                v[j] = 1;
                q[rear++] = j;
            }
        }
    }
}
```

```
int main()
{
  int root;
  for(int i=0; i<r; i++)
  {
    v[i] = 0;
  }
  printf("Enter the root: ");
  scanf("%d", &root);
  q[rear++] = root;
  v[root] = 1;
  bfs(root);
  return 0;
}
```

output:-

* Depth First Search

```
#include<stdio.h>
#define r 5

int a[r][r] = {{1,0,1,0},{0,0,1,1},{0,1,0,1}
               {1,1,1,1}};
int v[r];

void dfs(int root)
{
  printf("%d\t", root);
  v[root] = 1;
  for(int i=0; i<r; i++)
  {
    if(a[root][i] && !v[i])
    {
```

```
            dfs(i);
        }
    }
}

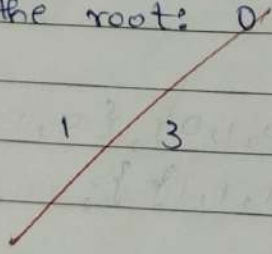
void main()
{
    int root;
    for(int i=0; i<n; i++)
    {
        v[i] = 0;
    }
    printf("Enter the root: ");
    scanf("%d", &root); printf("Path\n");
    dfs(root);
}
```

Output:-

Enter the root: 0

Path

0 2 1 3



Date: / /

* Hashing function

```
#include <stdio.h>
#define ts 3

int hash_array[ts];

void hashing(int key)
{
    int hkey, index;
    int i = 0;
    hkey = key % ts;
    while(i < ts)
    {
        index = (hkey + i) % ts;
        if(hash_array[index] == -1)
        { hash_array[index] = key;
          break;
        }
        i = i+1;
    }
}

void print()
{
    for(int i=0; i<ts ;i++)
    {
     printf("%d\n", hash_array[i]);
    }
}
```

hkey = 1 % 5 = 1

1 + 0 % 5

1

i = 1

1 < 5

0 < 1

1 < 1

1

1

i = 0

1

0 < 1

i = 1

```
void search(int s)
{
int hkey = s % ts;
int i=0, flag = 0;
int index;
while(i<ts)
{
  index = (hkey + i) % ts;
  if(hash_array[index] == s)
  {
    flag = 1;
    printf("Element found at pos %d",
                                index);
    break;
  }
  i+=1;
}
if(flag == 0)
  printf("Not found");
}

void main()
{
  int key, s;
  for(int j=0; j<ts ;j++)
  {
    hash_array[j] = -1;
  }
  for(int c=0; c<ts; c++)
  {
    printf("Enter the value: ");
    scanf("%d", &key);
```

```
    hashing(key);
}
print();

printf("Enter the key to be searched: ");
scanf("%d",&s);
search(s)
}
```

Output:-

Enter the value: 1
Enter the value: 2
Enter the value: 5
5
1
2
Enter the key to be searched: 2
Element found at pos 2