

SRI SIVASUBRAMANIYA NADAR COLLEGE OF ENGINEERING

**(AN AUTONOMOUS INSTITUTION,
AFFILIATED TO ANNA UNIVERSITY)**

Rajiv Gandhi Salai (OMR), Kalavakkam – 603 110.

LABORATORY RECORD

Name : Shreya Sriram

Reg. No. : 195001106

Dept. : CSE

Sem . : III Sec. : B



**SRI SIVASUBRAMANIYA NADAR
COLLEGE OF ENGINEERING, KALAVAKKAM**

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO ANNA UNIVERSITY)

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of the practical work done in the

.....DATA STRUCTURES..... Laboratory by

NameSHREYA SRIRAM

Register Number195001106.....

SemesterIII.....

BranchCSE.....

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam,

During the academic year2020-2021.....

Faculty B.PRABHAVATHY

Head of the Department

*Submitted for theEND SEMESTER... Practical Examination held at SSNCE
on ...17th November.....*

Internal Examiner

External Examiner

INDEX

Name : Shreya Sriram

Reg. No. 195001106

Sem. : III

Sec. : B

Ex. No.	Date of Expt.	Title of the Experiment	Page No.	Signature of the Faculty	Remarks
1	23.08.2020	Student Information System using Linked List Implementation of Student ListADT	4		
2	10.09.2020	Stack Implementation using Array and Linked List	17		
3	22.09.2020	Evaluation of postfix expression	31		
4	29.09.2020	Implementation of Circular Queue and its Application	38		
5	13.10.2020	Player Profile Maintenance of Cricket players using Binary Search Tree	49		
6	20.10.2020	Check connectivity and Route finding using BFS and DFS	61		
7	19.10.2020	Dictionary Application using AVL Tree	70		
8	27.10.2020	Prioritization of Employees based on salary with Priority Queue Implementation using Binary Heap	78		
9	7.11.2020	Implementation of hash table using separate chaining collision resolution strategy	87		
10	7.11.2020	Implementation of linear and binary searching techniques	93		

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering

III Semester - CSE

UCS 1312 Data Structures Lab Laboratory
Academic Year: 2020-2021 Batch: 2019-2023

(Odd)

**Exercise 1: Student Information System using Linked List Implementation of
Student ListADT**

In order to implement Student Information System,

1. Create a StudentADT.h with the following:
 - a. Data
 - i. Students' information such as Register number, Name and Category (StateBoard, CBSE, LaterEntry, Rural, NRI)
 - b. Operations
 - i. Creation of list of students
`void insertFront(listADT l, student s)`
 - ii. Insert a student after the given Register Number
`void insertAfter(listADT l, int reg_no)`
 - iii. Search the student based on his Register Number
`void search(listADT l, int reg_no)`
 - iv. Delete a student based on his Register Number
`void delete(listADT l, int reg_no)`
 - v. Display the details of the students in the List
`void display(listADT l)`
 - vi. List the students based on the given category
 - vii. List the details of the students who are having same name
2. Create StudentADTImpl.h with the implementations of the above-mentioned operations
3. Create StudentADTAppl.c is menu driven program which utilizes StudentADT and StudentADTImpl to perform the operations.

EXERCISE 1

STUDENT INFORMATION SYSTEM USING LINKED LIST

a) stuADT.h

```

struct student
{
    int regno;
    char name[20];
    char categ[20];
};

struct node
{
    struct student stu;
    struct node *next;
};

void insertFront(struct node *header, struct student s);
void insertAfter(struct node *header, struct student s, int rno);
void search(struct node *header, int rno);
void del(struct node *header, int rno);
void display(struct node *header);
void dispcat(struct node *header, char cat[20]);
void dispname(struct node *header, char name[20]);

```

b) stuADTImpl.h

```

#include "stuADT.h"
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

struct node* createHeader()
{
    struct node *header;
    header=(struct node *)malloc(sizeof(struct node));
    header->next=NULL;
    return header;
}

```

```

void insertFront(struct node *header,struct student s)
{
    struct node *temp;
    temp=(struct node*)malloc(sizeof(struct node));
    temp->stu=s;
    temp->next=header->next;
    header->next=temp;
    printf("\nINSERTED IN FRONT\n");
}

```

```

void insertAfter(struct node *header,struct student s,int rno)
{
    struct node *temp,*ptr;
    temp=(struct node*)malloc(sizeof(struct node*));
    temp->stu=s;
    ptr=header->next;
    while(ptr!=NULL)
    {
        if(ptr->stu.regno==rno)
        {
            temp->next=ptr->next;
            ptr->next=temp;
            break;
        }
        ptr=ptr->next;
    }
    printf("\nINSERTED AFTER REGISTER NUMBER\n");
}

```

```

void search(struct node *header,int rno)
{
    printf("\nSEARCHING RECORD....\n");
    struct node *ptr;
    int f=0;
    ptr=header->next;
    while(ptr!=NULL)
    {
        if(ptr->stu.regno==rno)
        {
            f=1;
            break;
        }
        ptr=ptr->next;
    }
    if(f==0)
    {

```

```

        printf("RECORD NOT FOUND\n");
    }
    else
    {
        printf("\nRECORD FOUND\n");
        printf("Name: %s\n",ptr->stu.name);
        printf("ID: %d\n",ptr->stu.regno);
        printf("Category: %s\n",ptr->stu.categ);
    }
}

```

```

void del(struct node *header,int rno)
{
    struct node *ptr,*prev;
    int flag=0;
    ptr=header->next;
    prev=header;
    while(ptr!=NULL)
    {
        if(ptr->stu.regno==rno)
        {
            flag=1;
            prev->next=ptr->next;
        }
        ptr=ptr->next;
        prev=prev->next;
    }
    if(flag==0)
    {
        printf("\nRECORD TO BE DELETED NOT FOUND!\n");
    }
    else
    {
        printf("\nRECORD REMOVED\n");
    }
}

```

```

void display(struct node *header)
{
    struct node *ptr;
    ptr=header->next;
    printf("\nSTUDENT DETAILS\n");
    while(ptr!=NULL)
    {
        printf("Name: %s\n",ptr->stu.name);
    }
}

```

```

        printf("ID: %d\n",ptr->stu.regno);
        printf("Category: %s\n",ptr->stu.categ);
        printf("\n");
        ptr=ptr->next;
    }
}

void dispcat(struct node *header,char cat[20])
{
    struct node *ptr;
    ptr=header->next;
    printf("\nLIST BY CATEGORY\n");
    printf("\n");
    while(ptr!=NULL)
    {
        if(strcmp(cat,ptr->stu.categ)==0)
        {
            printf("Name: %s\n",ptr->stu.name);
            printf("ID: %d\n",ptr->stu.regno);
            printf("Category: %s\n",ptr->stu.categ);
        }
        ptr=ptr->next;
    }
}

void dispname(struct node *header,char name[20])
{
    struct node *ptr;
    ptr=header->next;
    printf("\nLIST BY NAME\n");
    printf("\n");
    while(ptr!=NULL)
    {
        if(strcmp(name,ptr->stu.name)==0)
        {
            printf("Name: %s\n",ptr->stu.name);
            printf("ID: %d\n",ptr->stu.regno);
            printf("Category: %s\n",ptr->stu.categ);
        }
        ptr=ptr->next;
    }
}

```

c) stuADTAppl.c

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

void main()
{
    struct node *header;
    struct student t;
    char cat[20],name[20];
    int op,rno,ch;
    header=createHeader();
    do
    {
        printf("MENU\n");
        printf("1.Insertion of student record in front\n2.Insert a student after a given register number\n3.Search the student\n4.Delete a student record\n5.Display student details\n6.List students based on category\n7.List students having same name\n");
        printf("\n");
        printf("Enter the menu option: ");
        scanf("%d",&op);
        switch(op)
        {
            case 1:
                printf("\nEnter student name: ");
                scanf("%s",t.name);
                printf("Enter student register number: ");
                scanf("%d",&t.regno);
                printf("Enter student category: ");
                scanf("%s",t.categ);
                insertFront(header,t);
                break;

            case 2:
                printf("Enter register number after which a record has to be added: ");
                scanf("%d",&rno);
                printf("Enter student name: ");
                scanf("%s",t.name);
                printf("Enter student register number: ");
                scanf("%d",&t.regno);
                printf("Enter student category: ");
                scanf("%s",t.categ);
                insertAfter(header,t,rno);
```

```
break;

case 3:
printf("Enter register number for searching: ");
scanf("%d",&rno);
search(header,rno);
break;

case 4:
printf("Enter the register number for deletion: ");
scanf("%d",&rno);
del(header,rno);
break;

case 5:
display(header);
break;

case 6:
printf("Enter the category of student: ");
scanf("%s",cat);
dispcat(header,cat);
break;

case 7:
printf("Enter the name of student: ");
scanf("%s",name);
dispname(header,name);
break;

default:
printf("INVALID OPTION!!\n");
break;
}
printf("\n");
printf("DO YOU WANT TO CONTINUE?\n(PRESS 0 TO TERMINATE): ");
scanf("%d",&ch);
}while(ch!=0);
}
```

OUTPUT SCREENSHOTS

1) INSERT RECORD TO THE FRONT

```
MENU
1.Insertion of student record in front
2.Insert a student after a given register number
3.Search the student
4.Delete a student record
5.Display student details
6.List students based on category
7.List students having same name

Enter the menu option: 1

Enter student name: Shreya
Enter student register number: 1982
Enter student category: CBSE

INSERTED IN FRONT

DO YOU WANT TO CONTINUE?
(PRESS 0 TO TERMINATE): 1
```

2) INSERT RECORD AFTER A GIVEN REGISTER NUMBER

```
Enter the menu option: 2
Enter register number after which a record has to be added:
8273
Enter student name: Shreya
Enter student register number: 9823
Enter student category: NRI

INSERTED AFTER REGISTER NUMBER
```

3) SEARCHING FOR A RECORD (FOUND)

```
Enter the menu option: 3
Enter register number for searching: 1982

SEARCHING RECORD....

RECORD FOUND
Name: Shreya
ID: 1982
Category: CBSE

DO YOU WANT TO CONTINUE?
(PRESS 0 TO TERMINATE) : 1
```

SEARCHING FOR A RECORD (NOT FOUND)

```
Enter the menu option: 3
Enter register number for searching: 9988

SEARCHING RECORD....

RECORD NOT FOUND

DO YOU WANT TO CONTINUE?
(PRESS 0 TO TERMINATE) : 1
```



4) DELETING A RECORD

```
Enter the menu option: 4
Enter the register number for deletion: 9182

RECORD REMOVED

DO YOU WANT TO CONTINUE?
(PRESS 0 TO TERMINATE): 1
```

DELETING A NON EXISTENT RECORD

```
Enter the menu option: 4
Enter the register number for deletion: 1111

RECORD TO BE DELETED NOT FOUND!

DO YOU WANT TO CONTINUE?
(PRESS 0 TO TERMINATE): 1
```

5) DISPLAYING RECORDS

Enter the menu option: 5

STUDENT DETAILS

Name: Sam

ID: 9182

Category: STATE

Name: Shraddha

ID: 8273

Category: NRI

Name: Shreya

ID: 9823

Category: NRI

Name: Shreya

ID: 1982

Category: CBSE

DO YOU WANT TO CONTINUE?

(PRESS 0 TO TERMINATE): 1

6) LISTING BY CATEGORY

```
Enter the menu option: 6
Enter the category of student: NRI

LIST BY CATEGORY

Name: Shraddha
ID: 8273
Category: NRI
Name: Shreya
ID: 9823
Category: NRI

DO YOU WANT TO CONTINUE?
(PRESS 0 TO TERMINATE) : 1
```

7) LISTING BY NAME

```
Enter the menu option: 7
Enter the name of student: Shreya

LIST BY NAME

Name: Shreya
ID: 9823
Category: NRI
Name: Shreya
ID: 1982
Category: CBSE

DO YOU WANT TO CONTINUE?
(PRESS 0 TO TERMINATE) : 1
```

LEARNING OUTCOME:

- This application helped us use the concept of linked list to store and maintain student records.
- Creation of three separate files to incorporate modular programming and for easier understanding and organization of code.
- Creating different versions and applying the concept of incremental coding to expand and debug the code easily.

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021

Batch: 2019-2023

(Odd)

Exercise 1: Implementation of Character Stack using Array and Linked List

In order to implement Stack using Array,

4. Create a StackADT.h with the following:

a. Data

Structure Stack containing a character data array, size and top

b. Operations

i. Initialize the stack

`void initStack(struct stack S)`

ii. Push a character element into the stack

`void push(struct stack S, char c)`

iii. Pop an element from the stack

`char pop(struct stack S)`

iv. Check whether stack is Full

`int isFull(struct stack S)`

v. Check whether stack is Empty

`int isEmpty(struct stack S)`

vi. Display the elements of the stack

`void display(struct stack S)`

5. Create StackADTImpl.h with the implementations of the above-mentioned operations

6. Create StackADTApl.c is menu driven program which utilizes StackADT and StackADTImpl to perform the operations.

In order to implement Stack using Linked List,

1. Create a StackADT.h with the following:

- a. Data

Structure Stack containing a character data array, size and top

Structure containing the variable S of type struct Stack and the address of the next node

- b. Operations

- i. Push a character element into the stack

`void push(struct stack *top, char c)`

- ii. Pop an element from the stack

`char pop(struct stack *top)`

- iii. Check whether stack is Empty

`int isEmpty(struct stack *top)`

- iv. Display the elements of the stack

`void display(struct stack *top)`

2. Create StackADTImpl.h with the implementations of the above-mentioned operations
3. Create StackADTApl.c is menu driven program which utilizes StackADT and StackADTImpl to perform the operations.

EXERCISE 2

IMPLEMENTATION OF CHARACTER STACK USING ARRAY AND LINKED LIST

1) Array implementation

Code:

a) StackADT.h

```
struct stack
{
    int size;
    int top;
    char arr[50];
};

void initStack(struct stack *S);
void push(struct stack *S, char c);
char pop(struct stack *S);
int isFull(struct stack *S);
int isEmpty(struct stack *S);
void display(struct stack *S);
```

b) StackADTImpl.h

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

void initStack(struct stack *S)
{
    printf("\nEnter the size of the character array: ");
    scanf("%d",&S->size);
    S->top=-1;
}

void push(struct stack *S, char c)
{
    int f;
    f=isFull(S);
    if(f==1)
```

```

{
    S->top++;
    S->arr[S->top]=c;
    printf("\nINSERTED SUCCESSFULLY!!!\n");
}
else
{
    printf("\nARRAY IS FULL\n");
}
}
char pop(struct stack *S)
{

    int f=isEmpty(S);
    if(f!=1)
    {
        printf("CHARACTER OUTPUT: ");
        S->top--;
        return S->arr[S->top+1];

    }
    else
    {
        printf("\nARRAY IS EMPTY\n");
    }
}
int isFull(struct stack *S)
{
    if(S->top<S->size-1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
int isEmpty(struct stack *S)
{
    if(S->top==-1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

```

```

    }
}
void display(struct stack *S)
{
    printf("\n\tDISPLAY CHARACTERS\n");
    if(isEmpty(S)==1)
    {
        printf("Array is empty\n");
    }
    else
    {
        for(int i=0;i<S->top+1;i++)
        {
            printf("%c\t",S->arr[i]);
        }
        printf("\n");
    }
}
}

```

c) StackADTAppl.c

```

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

void main()
{
    int n,op;
    char ch,c,a;
    struct stack S;
    initStack(&S);
    do
    {
        printf("\nMENU\n1.Push\n2.Pop\n3.Display");
        printf("\nEnter option: ");
        scanf("%d",&op);
        switch(op)
        {
            case 1:
                printf("\nEnter character to input: ");
                scanf(" %c",&ch);
                push(&S,ch);
                break;

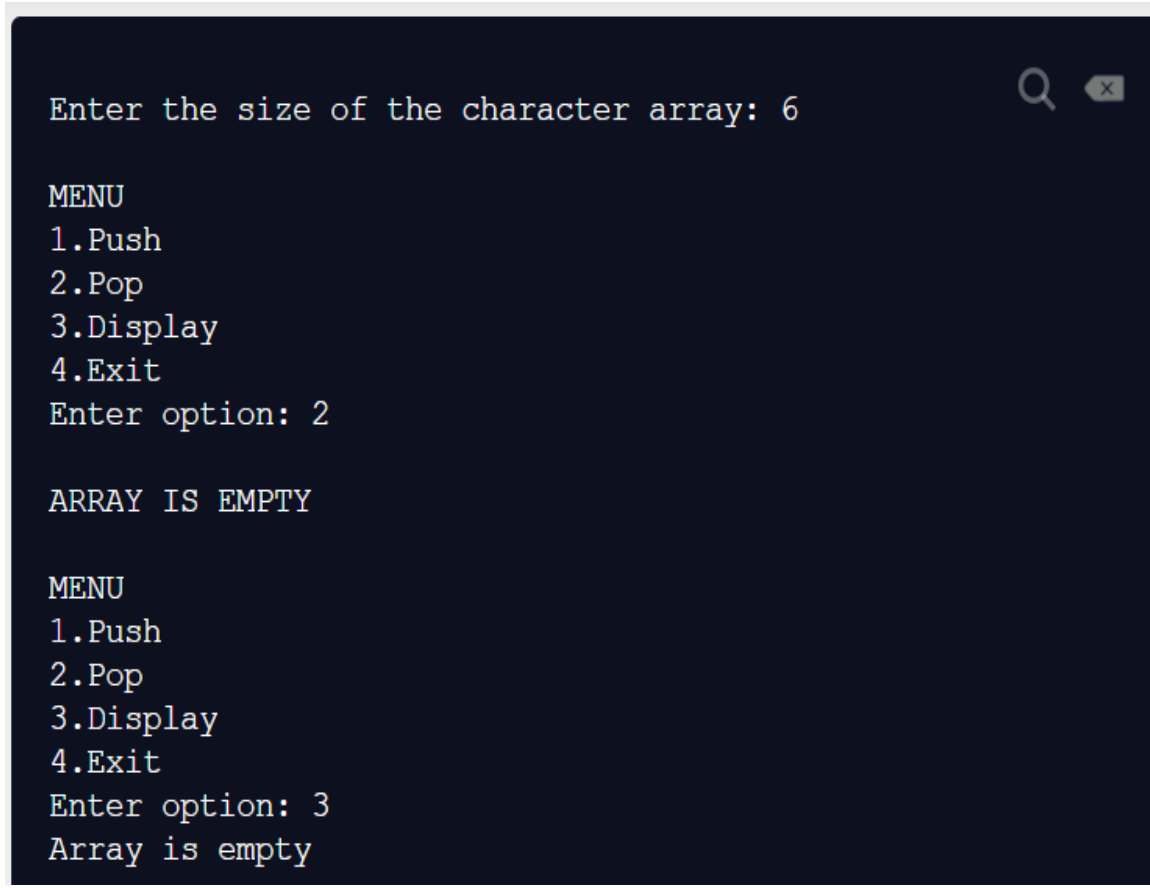
            case 2:
                a=pop(&S);

```

```
        printf("%c",a);  
        break;  
  
        case 3:  
            display(&S);  
            break;  
    }  
}while(op<=4);  
}
```

OUTPUT:

INITIALISING, DISPLAY AND POP FROM AN EMPTY ARRAY



Enter the size of the character array: 6

MENU

1.Push

2.Pop

3.Display

4.Exit

Enter option: 2

ARRAY IS EMPTY

MENU

1.Push

2.Pop

3.Display

4.Exit

Enter option: 3

Array is empty

PUSH TO STACK (a,b,c,d,e,f)

```
MENU
1.Push
2.Pop
3.Display
4.Exit
Enter option: 1

Enter character to input: c
```

PUSH TO A FULL STACK

```
MENU
1.Push
2.Pop
3.Display
4.Exit
Enter option: 1

Enter character to input: g

ARRAY IS FULL
```

DISPLAYING NON EMPTY STACK

```
MENU
1.Push
2.Pop
3.Display
4.Exit
Enter option: 3
a   b   c   d   e   f
```

POP FROM NON EMPTY STACK

```
MENU
1.Push
2.Pop
3.Display
4.Exit
Enter option: 2
a
```


A) Linked list implementation

Code:

StackADT.h

```
struct stack
{
    char arr;
    struct stack *next;
    int size;
};

void push(struct stack *top, char c);
char pop(struct stack *top);
int isEmpty(struct stack *top);
void display(struct stack *top);
```

StackADTImpl.h

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

void initStack(struct stack *top)
{
    top->next=NULL;
    top->size=0;
}

void push(struct stack *top, char c)
{
    struct stack *temp;
    temp=(struct stack*)malloc(sizeof(struct stack));
    temp->arr=c;
    top->size++;
    temp->next=top->next;
    top->next=temp;
    printf("\nPushed to stack\n");
    printf("Size of stack: %d\n",top->size);
}

int isEmpty(struct stack *top)
{

```

```

if(top->next==NULL)
{
    return 1;
}
else
{
    return 0;
}
}

char pop(struct stack *top)
{
    struct stack *ptr;
    ptr=top->next;
    if(!isEmpty(top))
    {
        char a;
        top->size--;
        printf("\nCHARACTER OUTPUT: ");
        a=ptr->arr;
        top->next=ptr->next;
        free(ptr);
        return a;
    }
    else
    {
        printf("\nARRAY IS EMPTY\n");
    }
}

void display(struct stack *top)
{
    printf("\nDISPLAY STACK\n");
    struct stack *ptr;
    ptr=top->next;
    if(isEmpty(top)==1)
    {
        printf("STACK IS EMPTY\n");
    }
    while(ptr!=NULL)
    {
        printf("%c\t",ptr->arr);
        ptr=ptr->next;
    }
    printf("Size of stack: %d\n",top->size);
}

```

```
}
```

StackADTAppl.c

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

void main()
{
    int op;
    char c;
    struct stack top;
    initStack(&top);
    do
    {
        printf("\nMENU\n1.Push to stack\n2.Pop from stack\n3.Check if stack is empty\n4.Display\nEnter option: ");
        scanf("%d",&op);
        switch(op)
        {
            case 1:
                printf("\nEnter element to push: ");
                scanf("%s",&c);
                push(&top,c);
                break;

            case 2:
                c=pop(&top);
                printf("%c",c);
                break;

            case 3:
                c=isEmpty(&top);
                if(c==1)
                {
                    printf("\nStack empty\n");
                }
                else
                {
                    printf("\nStack is not empty\n");
                }
                break;

            case 4:
```

```
        display(&top);  
        break;  
    }  
}while(op<=4);  
}
```

OUTPUT:

PUSH TO STACK

```
MENU  
1.Push to stack  
2.Pop from stack  
3.Check if stack is empty  
4.Display  
Enter option: 1  
  
Enter element to push: b  
  
Pushed to stack  
Size of stack: 2
```

POP FROM EMPTY STACK

```
MENU  
1.Push to stack  
2.Pop from stack  
3.Check if stack is empty  
4.Display  
Enter option: 2  
  
ARRAY IS EMPTY
```

POP FROM NON EMPTY STACK

```
MENU
1.Push to stack
2.Pop from stack
3.Check if stack is empty
4.Display
Enter option: 2

CHARACTER OUTPUT: b
```

CHECKING EMPTINESS OF STACK (EMPTY STACK)

```
MENU
1.Push to stack
2.Pop from stack
3.Check if stack is empty
4.Display
Enter option: 3

Stack empty
```

CHECKING EMPTINESS OF STACK (NON EMPTY STACK)

```
MENU
1.Push to stack
2.Pop from stack
3.Check if stack is empty
4.Display
Enter option: 3

Stack is not empty
```

DISPLAY (EMPTY STACK)

```
MENU
1.Push to stack
2.Pop from stack
3.Check if stack is empty
4.Display
Enter option: 4

DISPLAY STACK
STACK IS EMPTY
Size of stack: 0
```

DISPLAY (NON EMPTY STACK)

```
MENU
1.Push to stack
2.Pop from stack
3.Check if stack is empty
4.Display
Enter option: 4

DISPLAY STACK
e  d  c  b  a  Size of stack: 5
```

LEARNING OUTCOME

- Implementation of character stack using both array and linked list
- Modular programming using 3 files to organize and understand the code better.
- Incremental coding practice for efficient debugging
- Usage of proper prompt messages for user input and to make it more user friendly.

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021

Batch: 2019-2023

(Odd)

Exercise 3: Evaluation of postfix expression using character stack

Implement character stack and do the following:

- i. Convert infix expression into postfix expression using the character stack
- ii. Evaluate the postfix expression using the character stack

In order to implement Character Stack using Array,

7. Create a StackADT.h with the following:

a. Data

Structure Stack containing a character data array, size and top

b. Operations

- i. Initialize the stack
`void initStack(struct stack *S)`
- ii. Push a character element into the stack
`void push(struct stack *S, char c)`
- iii. Pop an element from the stack
`char pop(struct stack *S)`
- iv. Check whether stack is Full
`int isFull(struct stack *S)`
- v. Check whether stack is Empty
`int isEmpty(struct stack *S)`
- vi. Display the elements of the stack
`void display(struct stack *S)`

8. Create StackADTImpl.h with the implementations of the above-mentioned operations

9. Create StackADTAppl.c is menu driven program which utilizes StackADT and StackADTImpl to perform the operations.

EXERCISE 3

EVALUATION OF POSTFIX EXPRESSION USING CHARACTER STACK

EvalPostADTV1.h

```
#include "StackADTImpl.h"

struct evalpost
{
    char infix[100];
    char post[100];
    struct stack *t;
    int val;
};

void convert(struct evalpost *t, char a[100]);
```

EvalPostADTV1Impl.h

```
#include "EvalPostADTV1.h"
#include <string.h>
#include <stdio.h>
int precedence(char ch)
{
    if(ch=='+' || ch=='-')
    {
        return 1;
    }
    else if(ch=='/' || ch=='*' || ch=='%')
    {
        return 2;
    }
    else if(ch=='^')
    {
        return 3;
    }
    return 0;
}

int paranthesis(char ch)
{
    if(ch=='[' || ch=='(' || ch=='{')
```



```

{
    return 1;
}
else if(ch=='}' || ch==']' || ch=='')
{
    return 2;
}
return 0;
}

int isOperand(char ch)
{
    if(ch>='a'&&ch<='z' || ch>='A'&&ch<='Z' || ch>='0'&&ch<='9')
    {
        return 1;
    }
    return 0;
}

void convert(struct evalpost *p, char a[100])
{
    strcpy(p->infix,a);
    int pf=0,pre_curr,pre_top;
    char s;
    p->t=(struct stack*)malloc(sizeof(struct stack));
    p->t->top=-1;
    for(int i=0;p->infix[i];i++)
    {
        pre_curr=precedence(p->infix[i]);
        if(isOperand(p->infix[i])==1)
        {
            p->post[pf++]=p->infix[i];
        }
        else if(parenthesis(p->infix[i])==1)
        {
            push(p->t,p->infix[i]);
        }
        else if(parenthesis(p->infix[i])==2)
        {
            pre_top=precedence(p->t->arr[p->t->top]);
            while(!isEmpty(p->t) && parenthesis(p->t->arr[p->t->top])!=1)
            {
                p->post[pf++]=pop(p->t);
            }
        }
    }
}

```

```

else
{
    pre_top=precedence(p->t->arr[p->t->top]);
    if(pre_top<pre_curr)
    {
        push(p->t,p->infix[i]);
    }
    else
    {
        while(pre_top>=pre_curr && isEmpty(p->t)==0)
        {
            char s = pop(p->t);
            pre_top=precedence(p->t->arr[p->t->top]);
            p->post[pf++]=s;
        }
        push(p->t,p->infix[i]);
    }
}
}
while(!isEmpty(p->t))
{
    char s = pop(p->t);
    p->post[pf++]=s;
}

printf("POSTFIX EXPRESSION IS: ");
for(int i=0;i<pf;i++)
{
    printf("%c",p->post[i]);
}
}

```

EvalPostADTV1Appl.c

```

#include"EvalPostADTV1Impl.h"

#include<stdlib.h>

void main()
{
    struct evalpost *p;
    p=(struct evalpost *)malloc(sizeof(struct evalpost));
    char arr[100];
    printf("Enter expression to convert to postfix: ");
    gets(arr);

```

```

    convert(p,arr);
}

```

OUTPUT:

```

PS D:\01-College\assignments\data structures (lab)\ex3> gcc EvalPostADTV1Appl.c -o appl
PS D:\01-College\assignments\data structures (lab)\ex3> ./appl
Enter expression to convert to postfix: (1+5)*4/2-(2%3)
POSTFIX EXPRESSION IS: 15+4*2/23%-

```

EvalPostADTV2.h

```

#include"StackADTImpl.h"

struct evalpost
{
    char infix[100];
    char post[100];
    struct stack *t;
    int val;
};

void evaluate(struct evalpost *p,char a[]);

```

EvalPostADTV2Impl.h

```

#include"EvalPostADTV2.h"
#include<string.h>
#include<stdio.h>
#include<math.h>

void operation(int a,int b,char c,struct evalpost *p)
{
    switch(c)
    {
        case '+':
            push(p->t,b+a+'0');
            break;

        case '-':
            push(p->t,b-a+'0');
            break;
    }
}

```

```

        case '*':
            push(p->t,b*a+'0');
            break;

        case '/':
            push(p->t,b/a+'0');
            break;

        case '%':
            push(p->t,b%a+'0');
            break;

        default:
            printf("invalid postfix!!\n");
            break;
    }
}

int isOperand(char ch)
{
    if(ch>='0'&&ch<='9')
    {
        return 1;
    }
    return 0;
}

void evaluate(struct evalpost *p,char a[])
{
    strcpy(p->post,a);
    int op;
    p->t=(struct stack*)malloc(sizeof(struct stack));
    p->t->top=-1;
    for(int i=0;i<strlen(p->post);i++)
    {
        if(isOperand(p->post[i])==1)
        {
            push(p->t,p->post[i]);
            //printf("pushed");
        }
        else
        {
            int a = (int)pop(p->t)-'0';
            int b = (int)pop(p->t)-'0';
            operation(a,b,p->post[i],p);
        }
    }
}

```

```

    }
}
p->val=p->t->arr[p->t->top]-'0';
}

```

EvalPostADTV2Appl.c

```

#include "EvalPostADTV2Impl.h"
#include <stdlib.h>

void main()
{
    struct evalpost *p;
    p=(struct evalpost *)malloc(sizeof(struct evalpost));
    char arr[100];
    printf("Enter postfix expression: ");
    gets(arr);
    evaluate(p,arr);
    printf("Value: %d\n",p->val);
}

```

OUTPUT:

```

PS D:\01-College\assignments\data structures (lab)\ex3> gcc EvalPostADTV2Appl.c -o app
PS D:\01-College\assignments\data structures (lab)\ex3> ./app
Enter postfix expression: 15+4*2/23%-
Value: 10

```

LEARNING OUTCOME:

- To apply the concept of stack to evaluate postfix expression and calculate value from a postfix expression
- Creating versions for efficient debugging which saves time and works efficiently when it comes to handling larger codes
- Creating 3 separate files to apply modular programming and organization of code which makes it more presentable and easy to understand

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021 (Odd)

Batch: 2019-2023

Exercise 4: Implementation of Circular Queue and its Application

Implementation of Circular Queue

The structure Circular Queue consists of integer array, front, rear, size, iter. Implement Circular Queue using array with the following methods.

- void enqueue(Queue *Q, int x) – Insert an element into the queue
- int dequeue(Queue *Q) – Dequeue an element from the queue
- void disp(Queue *Q) – Display elements from the Queue
- int isEmpty(Queue *Q) – Check whether the queue is empty
- int isFull(Queue *Q) – Check whether the queue is full

Create CircularQueueADT.h with the data and declarations of the functions

Create CircularQueueADTImpl.h to implement the above-mentioned operations

Create CircularQueueADTAppl.c is menu driven program which utilizes CircularQueueADT and CircularQueueADTImpl to perform the operations.

Application of Circular Queue

1. Modify the circular queue to contain job number and the cpu burst time
2. Instantiate 2 circular queues Q1 and Q2
3. Insert circular queue with the following contents
(J1,2), (J2,4), (J3,8), (J4,5), (J5,2), (J6,7), (J7,4), (J8,3) (J9,6) & (J10,6)
4. Insert the job into the circular queue whichever is empty. If it is not empty, insert the job into the queue whichever is having minimum average time
5. Display the jobs waiting in both the queues along with their cpu burst time.

EXERCISE 4:

IMPLEMENTATION OF CIRCULAR QUEUE AND ITS APPLICATION

CQADTV2.h

```
#include<stdio.h>

#include<stdlib.h>

struct joballoc
{
    int bt,no;
};

struct circq
{
    int f,r,iter,size;
    struct joballoc arr[100];
};

void initQueue(struct circq *c,int size);
int isFull(struct circq *q);
int isEmpty(struct circq *q);
void enqueue(struct circq *c,struct joballoc x);
struct joballoc* dequeue(struct circq *c);
void display(struct circq *c);
```

CQADTImplV2.h

```
#include"CQADTV2.h"
```

```
void initQueue(struct circq *c,int size)
```

```
{
    c->f=c->r=-1;
    c->iter=0;
    c->size=size;
    printf("Initialised queue\n");
}
```

```
int isFull(struct circq *q)
```

```
{
    if(q->iter<q->size)
    {
        return 0;
    }
    return 1;
}
```

```
int isEmpty(struct circq *q)
```

```
{
    if(q->iter==0)
    {
        return 1;
    }
    return 0;
}
```

```
void enqueue(struct circq *c,struct joballoc x)
```



```

{
    if(!isFull(c))
    {
        c->iter++;
        c->r=(c->r+1)%c->size;
        c->arr[c->r]=x;
        printf("Inserted job %d into queue\n",c->arr[c->r].no);
    }
    else
    {
        printf("Queue is full\n");
    }
}

```

```

void display(struct circq *c)
{
    if(isEmpty(c))
    {
        printf("Empty queue\n");
    }
    else
    {
        for(int i=c->f+1;i<=c->r;++i)%c->size)
        {
            printf("Job Number: %d\t",c->arr[i].no);
            printf("Burst Time: %d\n",c->arr[i].bt);
        }
    }
}

```

```

    }
    printf("\n");
}

```

JobADTV3.h

```

#include "CQADTImplV2.h"

void allocate(struct circq *q1, struct circq *q2, struct joballoc j);
void calAvg(struct circq *q);

```

JobADTImplV3.h

```

#include "JobADTV3.h"

float calcAvg(struct circq *q)
{
    float sum=0;
    for(int i=q->f+1; i<=q->r; ++i)
    {
        sum+=q->arr[i].bt;
    }
    return sum/q->iter;
}

```

```

void allocate(struct circq *q1,struct circq *q2,struct joballoc j)
{
    if(isEmpty(q1))
    {
        enqueue(q1,j);
    }
    else if(isEmpty(q2))
    {
        enqueue(q2,j);
    }
    else if(calcAvg(q1)<calcAvg(q2) && !isFull(q1))
    {
        enqueue(q1,j);
    }
    else if(calcAvg(q2)<calcAvg(q1) && !isFull(q2))
    {
        enqueue(q2,j);
    }
    else if(!isFull(q1))
    {
        enqueue(q1,j);
    }
    else if(!isFull(q2))
    {
        enqueue(q2,j);
    }
    printf("QUEUE 1\n");
}

```

```

display(q1);
printf("QUEUE 2\n");
display(q2);
}

```

JobADTApplV3.c

```

#include"JobADTImplV3.h"

void main()
{
    struct circq *q1,*q2;
    q1=(struct circq *)malloc(sizeof(struct circq));
    q2=(struct circq *)malloc(sizeof(struct circq));
    initQueue(q1,5);
    initQueue(q2,5);
    int inp[10][2] = { {1,2},{2,4},{3,8},{4,5},{5,2},{6,7},{7,4},{8,3},{9,6},{10,6}};
    for(int i=0;i<10;i++)
    {
        struct joballoc j;
        j.no = inp[i][0];
        j.bt = inp[i][1];
        allocate(q1,q2,j);
    }

    printf("DISPLAYING QUEUES\n");
}

```

```
printf("QUEUE 1\n");  
printf("\n");  
display(q1);  
printf("QUEUE 2\n");  
printf("\n");  
display(q2);  
}
```

```

PS D:\01-College\assignments\data structures (lab)\ex4> gcc JobADTApp1V3.c -o app
PS D:\01-College\assignments\data structures (lab)\ex4> ./app
Initialised queue
Initialised queue
Inserted job 1 into queue
QUEUE 1
Job Number: 1    Burst Time: 2

QUEUE 2
Empty queue

Inserted job 2 into queue
QUEUE 1
Job Number: 1    Burst Time: 2

QUEUE 2
Job Number: 2    Burst Time: 4

Inserted job 3 into queue
QUEUE 1
Job Number: 1    Burst Time: 2
Job Number: 3    Burst Time: 8

QUEUE 2
Job Number: 2    Burst Time: 4

Inserted job 4 into queue
QUEUE 1
Job Number: 1    Burst Time: 2
Job Number: 3    Burst Time: 8

QUEUE 2
Job Number: 2    Burst Time: 4
Job Number: 4    Burst Time: 5

Inserted job 5 into queue
QUEUE 1
Job Number: 1    Burst Time: 2
Job Number: 3    Burst Time: 8

QUEUE 2
Job Number: 2    Burst Time: 4
Job Number: 4    Burst Time: 5
Job Number: 5    Burst Time: 2

Inserted job 6 into queue
QUEUE 1
Job Number: 1    Burst Time: 2
Job Number: 3    Burst Time: 8

QUEUE 2
Job Number: 2    Burst Time: 4
Job Number: 4    Burst Time: 5
Job Number: 5    Burst Time: 2
Job Number: 6    Burst Time: 7

Inserted job 7 into queue
QUEUE 1
Job Number: 1    Burst Time: 2
Job Number: 3    Burst Time: 8

QUEUE 2
Job Number: 2    Burst Time: 4
Job Number: 4    Burst Time: 5
Job Number: 5    Burst Time: 2
Job Number: 6    Burst Time: 7
Job Number: 7    Burst Time: 4

Inserted job 8 into queue
QUEUE 1
Job Number: 1    Burst Time: 2
Job Number: 3    Burst Time: 8
Job Number: 8    Burst Time: 3

```

```

QUEUE 2
Job Number: 2   Burst Time: 4
Job Number: 4   Burst Time: 5
Job Number: 5   Burst Time: 2
Job Number: 6   Burst Time: 7
Job Number: 7   Burst Time: 4

Inserted job 9 into queue
QUEUE 1
Job Number: 1   Burst Time: 2
Job Number: 3   Burst Time: 8
Job Number: 8   Burst Time: 3
Job Number: 9   Burst Time: 6

QUEUE 2
Job Number: 2   Burst Time: 4
Job Number: 4   Burst Time: 5
Job Number: 5   Burst Time: 2
Job Number: 6   Burst Time: 7
Job Number: 7   Burst Time: 4

Inserted job 10 into queue
QUEUE 1
Job Number: 1   Burst Time: 2
Job Number: 3   Burst Time: 8
Job Number: 8   Burst Time: 3
Job Number: 9   Burst Time: 6
Job Number: 10  Burst Time: 6

QUEUE 2
Job Number: 2   Burst Time: 4
Job Number: 4   Burst Time: 5
Job Number: 5   Burst Time: 2
Job Number: 6   Burst Time: 7
Job Number: 7   Burst Time: 4

DISPLAYING QUEUES
QUEUE 1

Job Number: 1   Burst Time: 2
Job Number: 3   Burst Time: 8
Job Number: 8   Burst Time: 3
Job Number: 9   Burst Time: 6
Job Number: 10  Burst Time: 6

QUEUE 2

Job Number: 2   Burst Time: 4
Job Number: 4   Burst Time: 5
Job Number: 5   Burst Time: 2
Job Number: 6   Burst Time: 7
Job Number: 7   Burst Time: 4

```

LEARNING OUTCOME

- Using the concept of circular queue to assign tasks to two queues and effectively use all the queue functions to create a real life application
- Proper prompt messages and to make it more user friendly and presentable
- Incremental coding and modular programming was implemented in the code for easier debugging and better presentation

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021

Batch: 2019-2023

(Odd)

Exercise 5: Player Profile Maintenance of Cricket players using Binary Search Tree

The structure playerProfile consists of playerInfo, left and right children. The structure playerInfo has the name, role of the player (Batsman or Bowler) and average run rate. Implement the following methods.

- void insert(struct playerProfile *P, struct playerInfo x) – Insert information about a player into profile
- void delete(struct playerProfile *Q, char name[]) – Delete the information about the player given his name
- void disp(struct playerProfile *Q) – Display the information about all the players (Hierarchically)
- struct playerInfo *search(struct playerProfile *P, char name[]) – Search the player for his information
- void preorder(struct playerProfile *P) – Display the player names in preorder
- void inorder(struct playerProfile *P) – Display the player names in inorder
- void postorder(struct playerProfile *P) – Display the player names in postorder

Note:

In order to implement this Player Profile System,

- It is necessary to create a file that has PlayerProfile ADT and implementation of above-mentioned functions
- Another file will be created with only function prototypes
- One more file will be created to write the player profile information system using the PlayerProfile ADT

EXERCISE 5

PLAYER PROFILE MAINTENANCE OF CRICKET PLAYERS USING BINARY SEARCH TREE

PlayerProfileADT.h

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

struct playerInfo
{
    char name[30],role[10];
    float avg;
};

struct playerProfile
{
    struct playerInfo p;
    struct playerProfile *left,*right;
};

struct playerProfile* insert(struct playerProfile *P, struct playerInfo x);
struct playerProfile* deletePlayer(struct playerProfile *Q, char name[]);
void disp(struct playerProfile *Q,int space);
struct playerInfo *search(struct playerProfile *P, char name[]);
void preorder(struct playerProfile *P);
void inorder(struct playerProfile *P);
void postorder(struct playerProfile *P);
```

PlayerProfileADTImpl.h

```
#include"PlayerProfileADT.h"

struct playerProfile* insert(struct playerProfile *P, struct playerInfo x)
{
    if(P == NULL)
    {
        P = (struct playerProfile*)malloc(sizeof(struct playerProfile));
        P->p = x;
        P->left = P->right = NULL;
    }
    if(strcmp(x.name,P->p.name)<0)
```

```

{
    P->left = insert(P->left,x);
}
if(strcmp(x.name,P->p.name)>0)
{
    P->right = insert(P->right,x);
}
return P;
}

```

struct playerProfile *findMin(*struct* playerProfile *P)

```

{
    if(P == NULL)
    {
        printf("Tree empty\n");
        return NULL;
    }
    while(P->left!=NULL)
    {
        P->left = findMin(P->left);
    }
    if(P->left == NULL)
    {
        return P;
    }
}

```

struct playerProfile * deletePlayer(*struct* playerProfile *Q, *char* name[])

```

{
    struct playerProfile *temp;
    if(Q == NULL)
    {
        return Q;
    }
    else if(strcmp(name,Q->p.name)<0)
    {
        Q->left = deletePlayer(Q->left,name);
    }
    else if(strcmp(name,Q->p.name)>0)
    {
        Q->right = deletePlayer(Q->right,name);
    }
    else if(Q->left == NULL && Q->right == NULL)
    {
        temp = Q;
        Q = NULL;
    }
}

```

```

    free(temp);
}
else if(Q->left && Q->right)
{
    temp = findMin(Q->right);
    strcpy(Q->p.name,temp->p.name);
    Q->right = deletePlayer(Q->right,temp->p.name);
    free(temp);
}
else
{
    temp = Q;
    if(Q->left == NULL)
    {
        Q = Q->right;
    }
    if(Q->right == NULL)
    {
        Q = Q->left;
    }
    free(temp);
}
return Q;
}

```

```

void disp(struct playerProfile *Q,int space)

```

```

{
    if(Q == NULL)
    {
        return;
    }
    space += 15;
    disp(Q->right,space);
    printf("\n");
    for(int i=10;i<space;i++)
    {
        printf(" ");
    }
    printf("%s\n",Q->p.name);
    for(int i=10;i<space;i++)
    {
        printf(" ");
    }
    printf("%s\n",Q->p.role);
    for(int i=10;i<space;i++)
    {

```

```

    printf(" ");
}
printf("%f\n",Q->p.avg);
disp(Q->left,space);
}

```

```

struct playerInfo * search(struct playerProfile *P, char name[])
{
    if(P == NULL)
    {
        return NULL;
    }
    else if(strcmp(name,P->p.name)<0)
    {
        return search(P->left,name);
    }
    else if(strcmp(name,P->p.name)>0)
    {
        return search(P->right,name);
    }
    else if(strcmp(name,P->p.name)==0)
    {
        printf("Found\n");
        return &P->p;
    }
    return NULL;
}

```

```

void inorder(struct playerProfile *P)
{
    if(P->left != NULL)
    {
        inorder(P->left);
    }
    printf("Name: %s\n",P->p.name);
    printf("Role: %s\n",P->p.role);
    printf("Average run rate: %f\n",P->p.avg);
    if(P->right != NULL)
    {
        inorder(P->right);
    }
}

```

```

void preorder(struct playerProfile *P)
{
    printf("Name: %s\n",P->p.name);
}

```

```

printf("Role: %s\n",P->p.role);
printf("Average run rate: %f\n",P->p.avg);
if(P->left != NULL)
{
    preorder(P->left);
}
if(P->right != NULL)
{
    preorder(P->right);
}
}

void postorder(struct playerProfile *P)
{
    if(P->left != NULL)
    {
        postorder(P->left);
    }
    if(P->right != NULL)
    {
        postorder(P->right);
    }
    printf("Name: %s\n",P->p.name);
    printf("Role: %s\n",P->p.role);
    printf("Average run rate: %f\n",P->p.avg);
}

```

PlayerProfileADTAppl.c

```

#include "PlayerProfileADTImpl.h"

void main()
{
    struct playerProfile *P = NULL;
    struct playerInfo *x;
    char name[30];
    int ch;
    char c='y';
    do{
        printf("MENU\n1.Insert\n2.Delete\n3.Display in tree structure\n4.Search\n5.Inorder\n6.Pre
order\n7.Postorder\n");
        printf("Enter choice: ");
        scanf("%d",&ch);
        if(ch==1)
        {

```

```

    printf("Enter name: ");
    scanf("%s",x->name);
    printf("Enter role: ");
    scanf("%s",x->role);
    printf("Enter average runs: ");
    scanf("%f",&x->avg);
    P = insert(P,*x);
    printf("INSERTED\n");
}

else if(ch==2)
{
    printf("Enter name: ");
    scanf("%s",name);
    P = deletePlayer(P,name);
}

else if(ch==3)
{
    disp(P,0);
}

else if(ch==4)
{
    printf("Enter name: ");
    scanf("%s",name);
    x = search(P,name);
    if(x == NULL)
    {
        printf("Not found\n");
    }
}

else if(ch==5)
{
    inorder(P);
}

else if(ch==6)
{
    preorder(P);
}
else if(ch==7)
{
    preorder(P);
}

```

```
    printf("DO YOU WANT TO CONTINUE? (y or Y to continue): ");  
    scanf("%s",&c);  
}while(c=='y'||c=='Y');  
}
```

OUTPUT:

INSERT TO BST

```
MENU  
1.Insert  
2.Delete  
3.Display in tree structure  
4.Search  
5.Inorder  
6.Preorder  
7.Postorder  
Enter choice: 1  
Enter name: MSD  
Enter role: Batsman  
Enter average runs: 66.3  
INSERTED  
DO YOU WANT TO CONTINUE? (y or Y to continue): y
```

DELETE FROM BST

```
MENU  
1.Insert  
2.Delete  
3.Display in tree structure  
4.Search  
5.Inorder  
6.Preorder  
7.Postorder  
Enter choice: 2  
Enter name: Kapil  
DO YOU WANT TO CONTINUE? (y or Y to continue): y
```

DISPLAY BST IN TREE HEIRARCHY

```
6.Preorder
7.Postorder
Enter choice: 3
```

```

                                Sehwag
                                Batsman
                                88.400002

                                Sachin
                                Batsman
                                66.699997

                                Raina
                                Batsman
                                99.699997

                                MSD
                                Batsman
                                66.300003

                                Hardik
                                Bowler
                                33.200001

                                Chris
                                Batsman
                                77.300003

                                Bravo
                                Bowler
                                55.400002

                                Aswin
                                Batsman
                                77.300003

DO YOU WANT TO CONTINUE? (y or Y to continue): y
```

SEARCH IN BST

```

MENU
1.Insert
2.Delete
3.Display in tree structure
4.Search
5.Inorder
6.Preorder
7.Postorder
Enter choice: 4
Enter name: Aswin
Found
DO YOU WANT TO CONTINUE? (y or Y to continue): y
```


INORDER IN BST

```
MENU
1.Insert
2.Delete
3.Display in tree structure
4.Search
5.Inorder
6.Preorder
7.Postorder
Enter choice: 4
Enter name: Aswin
Found
DO YOU WANT TO CONTINUE? (y or Y to continue): y
MENU
1.Insert
2.Delete
3.Display in tree structure
4.Search
5.Inorder
6.Preorder
7.Postorder
Enter choice: 5
Name: Aswin
Role: Batsman
Average run rate: 77.300003
Name: Bravo
Role: Bowler
Average run rate: 55.400002
Name: Chris
Role: Batsman
Average run rate: 77.300003
Name: Hardik
Role: Bowler
Average run rate: 33.200001
Name: MSD
Role: Batsman
Average run rate: 66.300003
Name: Raina
Role: Batsman
Average run rate: 99.699997
Name: Sachin
Role: Batsman
Average run rate: 66.699997
Name: Sehwag
Role: Batsman
Average run rate: 88.400002
DO YOU WANT TO CONTINUE? (y or Y to continue): y
```

PREORDER

```
MENU
1.Insert
2.Delete
3.Display in tree structure
4.Search
5.Inorder
6.Preorder
7.Postorder
Enter choice: 6
Name: MSD
Role: Batsman
Average run rate: 66.300003
Name: Chris
Role: Batsman
Average run rate: 77.300003
Name: Aswin
Role: Batsman
Average run rate: 77.300003
Name: Bravo
Role: Bowler
Average run rate: 55.400002
Name: Hardik
Role: Bowler
Average run rate: 33.200001
Name: Raina
Role: Batsman
Average run rate: 99.699997
Name: Sachin
Role: Batsman
Average run rate: 66.699997
Name: Sehwag
Role: Batsman
Average run rate: 88.400002
DO YOU WANT TO CONTINUE? (y or Y to continue): y
```

POSTORDER IN BST

```

DO YOU WANT TO CONTINUE? (y or Y to continue): y
MENU
1.Insert
2.Delete
3.Display in tree structure
4.Search
5.Inorder
6.Preorder
7.Postorder
Enter choice: 7
Name: MSD
Role: Batsman
Average run rate: 66.300003
Name: Chris
Role: Batsman
Average run rate: 77.300003
Name: Aswin
Role: Batsman
Average run rate: 77.300003
Name: Bravo
Role: Bowler
Average run rate: 55.400002
Name: Hardik
Role: Bowler
Average run rate: 33.200001
Name: Raina
Role: Batsman
Average run rate: 99.699997
Name: Sachin
Role: Batsman
Average run rate: 66.699997
Name: Sehwag
Role: Batsman
Average run rate: 88.400002
DO YOU WANT TO CONTINUE? (y or Y to continue): y

```

LEARNING OUTCOME

- Binary search tree has been implemented successfully to maintain player records.
- Creation of three separate files to incorporate modular programming and for easier understanding and organization of code.
- Creating different versions and applying the concept of incremental coding to expand and debug the code easily.

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021 (Odd)

Batch: 2019-2023

Exercise 6: Check connectivity and Route finding using BFS and DFS

Check connectivity application for cities using BFS

The cityADT contains the number of cities and the connectivity information between the cities (adjacency matrix). Write the following methods.

- void create(cityADT *C) – will represent the graph using adjacency matrix
- void disp(cityADT *C) – Display the graph
- void BFS(cityADT *C) – provides the OUTPUT of visiting the cities by following breadth first
- int connect(cityADT *C) – Check whether connection is present between the source and destination cities. Will return 1 if there is connection, otherwise 0.

Note:

Implement cityADT with the specified operations in cityADTImpl.h
 Write a menu driven connectivity application to utilize the cityADT.

Route finding application between cities using DFS

The cityADT contains the number of cities and the connectivity information between the cities (adjacency matrix). Write the following methods.

- void create(cityADT *C) – will represent the graph using adjacency matrix
- void disp(cityADT *C) – Display the graph
- void DFS(cityADT *C) – provides the OUTPUT of visiting the cities by following depth first
- char * path(cityADT *C) – Find the path of the intermediate cities between the source and destination cities.

Note:

Implement cityADT with the specified operations in cityADTImpl.h
 Write a menu driven application to utilize the cityADT.

EXERCISE 6

CHECK CONNECTIVITY AND ROUTE FINDING USING BFS AND DFS

cityADT.h

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

struct cityADT
{
    int adjMat[10][10],visit[10],path[10];
    int nc,nr;
    char name[10];
};

void init(struct cityADT *g);
void display(struct cityADT *g);
void DFS(struct cityADT *g,int x);
void path(struct cityADT *g, int x,int y);
void checkPath(struct cityADT *g, int x,int y,int j);
int checkConnect(struct cityADT *C);
```

cityADTImpl.h

```
#include "cityADT.h"

void init(struct cityADT *g)
{
    int x, y;
    printf("Enter cities: ");
    scanf("%d", &g->nc);
    printf("Enter routes: ");
    scanf("%d", &g->nr);
    for (int i = 1; i <= g->nc; i++)
    {
        g->visit[i] = 0;
        for (int j = 1; j <= g->nc; j++)
        {
            g->adjMat[i][j] = 0;
        }
    }
}
```

```

for (int i = 1; i <= g->nc; i++)
{
    printf("Enter name: ");
    scanf(" %c", &g->name[i]);
}
for (int i = 0; i < g->nr; i++)
{
    printf("Enter source: ");
    scanf("%d", &x);
    printf("Enter destination: ");
    scanf("%d", &y);
    g->adjMat[x][y] = 1;
}
}

void display(struct cityADT *g)
{
    for (int i = 1; i <= g->nc; i++)
    {
        printf("\t%c", g->name[i]);
    }
    printf("\n");
    for (int i = 1; i <= g->nc; i++)
    {
        printf("%c\t", g->name[i]);
        for (int j = 1; j <= g->nc; j++)
        {
            printf("%d\t", g->adjMat[i][j]);
        }
        printf("\n");
    }
}

void DFS(struct cityADT *g, int x)
{
    g->visit[x] = 1;
    for (int i = 1; i <= g->nc; i++)
    {
        if (g->visit[i] == 0 && g->adjMat[x][i] == 1)
        {
            g->visit[i] = 1;
            printf(" %c\t", g->name[i]);
            DFS(g, i);
        }
    }
}

```

```

void BFS(struct cityADT *g, int arr[], int p1, int p2, int visit[])
{
    int x = arr[p1];
    if (x != -1)
    {
        for (int j = 1; j <= g->nc; j++)
        {
            if (g->adjMat[x][j] == 1 && visit[j] == 0)
            {
                visit[j] = 1;
                p2++;
                arr[p2] = j;
            }
        }
        p1++;
        BFS(g, arr, p1, p2, visit);
    }
    else
    {
        for (int i = 0; i < p1; i++)
        {
            printf(" %c\t", g->name[arr[i]]);
        }
    }
}

```

```

int checkConnect(struct cityADT *C)
{
    int x,y;
    printf("Enter the city numbers to check connection\n");
    printf("Enter city 1: ");
    scanf("%d",&x);
    printf("Enter city 2: ");
    scanf("%d",&y);
    if(C->adjMat[x][y]==1)
    {
        return 1;
    }
    return 0;
}

```

```

void checkPath(struct cityADT *g, int x,int y,int j)

```

```

{

```

```

g->visit[x] = 1;
for (int i = 1; i <= g->nc; i++)
{
    if (g->visit[i] == 0 && g->adjMat[x][i] == 1)
    {
        g->visit[i] = 1;
        g->path[j]=i;
        //printf("%d\t",g->path[j]);
        if(g->path[j]==y)
        {
            break;
        }
        j++;
        checkPath(g,i,y,j);
    }
}
}

```

```

void path(struct cityADT *g, int x,int y)
{
    checkPath(g,x,y,0);
    for(int i=0;g->path[i]!=-1;i++)
    {
        printf("%c\t",g->name[g->path[i]]);
    }
}

```

cityADTAppl.c

```
#include "cityADTImpl.h"
```

```

void main()
{
    struct cityADT *g;
    int x,arr[10],p1=0,p2=0,visit[10],n,s,a,b;
    g=(struct cityADT*)malloc(sizeof(struct cityADT));
    init(g);
    printf("MENU\n1.Display\n2.DFS\n3.BFS\n4.Check connection\n5.Cech path\n6.Exit\n");
    do
    {
        printf("Enter option: ");
        scanf("%d",&n);
        switch(n)
        {
            case 1:

```



```
display(g);
break;
```

```
case 2:
printf("Enter starting vertex: ");
scanf("%d",&s);
printf("%c\t",g->name[s]);
DFS(g,s);
printf("\n");
break;
```

```
case 3:
printf("Enter starting city: ");
scanf("%d",&s);
for(int i=1;i<=g->nc;i++)
{
    visit[i]=0;
    arr[i]=-1;
}
visit[s]=1;
arr[0]=s;
BFS(g,arr,p1,p2,visit);
printf("\n");
break;
```

```
case 4:
if(checkConnect(g)==1)
{
    printf("Connection exists!\n");
}
else
{
    printf("Connection doesnt exist!\n");
}
break;
```

```
case 5:
p1=0;
for(int i=1;i<=g->nc;i++)
{
    g->path[i]=-1;
    g->visit[i]=0;
}
printf("Enter starting city: ");
scanf("%d",&a);
printf("Enter ending city: ");
```

```
        scanf("%d",&b);  
        path(g,a,b);  
        printf("\n");  
        break;  
    }  
} while (n!=6);  
}
```

OUTPUT

INITIALISING GRAPH

```
PS D:\01-College\data\GRAPH> ./main  
Enter cities: 6  
Enter routes: 7  
Enter name: A  
Enter name: B  
Enter name: C  
Enter name: D  
Enter name: E  
Enter name: F  
Enter source: 1  
Enter destination: 3  
Enter source: 3  
Enter destination: 5  
Enter source: 5  
Enter destination: 6  
Enter source: 6  
Enter destination: 4  
Enter source: 1  
Enter destination: 4  
Enter source: 2  
Enter destination: 5  
Enter source: 2  
Enter destination: 1
```

DISPLAYING THE ADJACENCY MATRIX

```

MENU
1.Display
2.DFS
3.BFS
4.Check connection
5.Cech path
6.Exit
Enter option: 1

```

	A	B	C	D	E	F
A	0	0	1	1	0	0
B	1	0	0	0	1	0
C	0	0	0	0	1	0
D	0	0	0	0	0	0
E	0	0	0	0	0	1
F	0	0	0	1	0	0

BFS

```

Enter option: 2
Enter starting vertex: 3
C      E      F      D

```

CHECKING CONNECTIONS BETWEEN CITIES

```

Enter option: 4
Enter the city numbers to check connection
Enter city 1: 1
Enter city 2: 2
Connection doesnt exist!
Enter option: 4
Enter the city numbers to check connection
Enter city 1: 1
Enter city 2: 3
Connection exists!

```

DFS

```

Enter option: 3
Enter starting city: 2
  B      A      E      C      D      F

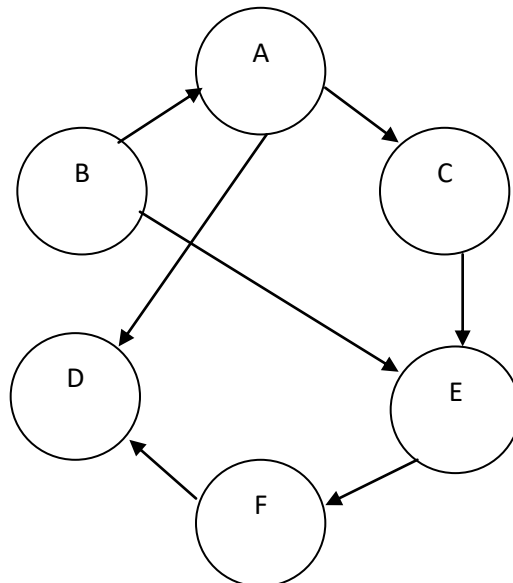
```

CHECKING INTERMEDIATE PATH

```

Enter option: 5
Enter starting city: 2
Enter ending city: 4
  A      C      E      F      D
Enter option: 6

```

SAMPLE GRAPH**LEARNING OUTCOMES**

- Graph traversals are successfully implemented using recursive logic with additional functions to check intermediate path and connectivity between 2 cities.
- Creating versions for efficient debugging which saves time and works efficiently when it comes to handling larger codes
- Creating 3 separate files to apply modular programming and organization of code which makes it more presentable and easy to understand
- Usage of appropriate comments and prompt messages for user friendliness

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021

Batch: 2019-2023

(Odd)

Exercise 7: Dictionary Application using AVL Tree

Develop a dictionary application using AVL Tree. In the tree, each node consists of a word and its meaning. The dictionary application is menu driven and supports the following operations.

- a. Insertion
- b. Display
- c. Search

The structure dictionaryADT consists of wordMeaning, left and right children. The structure wordMeaning has the word and meaning. Implement the following methods.

- struct dictionaryADT* insert(struct dictionaryADT *D, struct wordMeaning x) – Insertion of a new word and meaning into dictionary
- void disp(struct dictionaryADT *D) – Display the information about all the words and meanings (Hierarchically)
- struct dictionaryADT* search(struct dictionaryADT *D, char word[]) – Will search for a word to get its meaning

Note:

In order to implement this dictionary application,

- It is necessary to create a file that has dictionaryADT.h with the data and prototypes of the above-mentioned functions
- Another file dictionaryADTImpl.h will be created with only function definitions
- One more file dictionaryADTAppl.c will be created to develop dictionary application that utilizes the dictionaryADT

EXERCISE 7

DICTIONARY APPLICATION USING AVL TREE

DictionaryADT.h

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

struct wordMeaning
{
    char word[15],meaning[100];
};

struct dictionaryADT
{
    struct wordMeaning w;
    struct dictionaryADT *left,*right;
    int height;
};

struct dictionaryADT* insert(struct dictionaryADT *D, struct wordMeaning x);
void disp(struct dictionaryADT *D,int space);
struct dictionaryADT* search(struct dictionaryADT *D, char word[]);
struct dictionaryADT * singleRotatewithLeft(struct dictionaryADT *D);
struct dictionaryADT * singleRotatewithRight(struct dictionaryADT *D);
struct dictionaryADT * doubleRotatewithLeft(struct dictionaryADT *D);
struct dictionaryADT * doubleRotatewithRight(struct dictionaryADT *D);
```

DictionaryADTImpl.h

```
#include"DictionaryADT.h"

static int treeHeight(struct dictionaryADT *D)
{
    if(D==NULL)
    {
        return -1;
    }
    else
    {
        return D->height;
    }
}
```

```

int max(int x,int y)
{
    if(x>y)
        return x;
    else if(y>x)
        return y;
    else
        return 0;
}

```

```

struct dictionaryADT* insert(struct dictionaryADT *D, struct wordMeaning x)
{
    int hdiff;
    if(D==NULL)
    {
        D = (struct dictionaryADT*)malloc(sizeof(struct dictionaryADT));
        D->w = x;
        D->left=D->right=NULL;
        D->height = 0;
    }
    else if(strcmp(x.word,D->w.word)<0)
    {
        D->left = insert(D->left,x);
        hdiff=max(treeHeight(D->left),treeHeight(D->right));
        if(hdiff==2)
        {
            if(strcmp(x.word,D->left->w.word)<0)
            {
                D = singleRotatewithLeft(D);
            }
            else
            {
                D = doubleRotatewithLeft(D);
            }
        }
    }
    else if((x.word,D->w.word)>0)
    {
        D->right = insert(D->right,x);
        hdiff=max(treeHeight(D->left),treeHeight(D->right));
        if(hdiff==2)
        {
            if(strcmp(x.word,D->right->w.word)>0)
            {
                D = singleRotatewithRight(D);
            }
        }
    }
}

```

```

        else
        {
            D = doubleRotatewithRight(D);
        }
    }
}

D->height = max(treeHeight(D->left),treeHeight(D->right))+1;
return D;
}

struct dictionaryADT * singleRotatewithLeft(struct dictionaryADT *D)
{
    struct dictionaryADT *temp;
    temp = D->left;
    D->left = temp->right;
    temp->right = D;
    D->height=max(treeHeight(D->left),treeHeight(D->right))+1;
    temp->height=max(treeHeight(temp->left),treeHeight(temp->right))+1;
    return temp;
}

struct dictionaryADT * doubleRotatewithLeft(struct dictionaryADT *D)
{
    D->left=singleRotatewithRight(D->left);
    return singleRotatewithLeft(D);
}

struct dictionaryADT * singleRotatewithRight(struct dictionaryADT *D)
{
    struct dictionaryADT *temp;
    temp = D->right;
    D->right = temp->left;
    temp->left = D;
    D->height=max(treeHeight(D->left),treeHeight(D->right))+1;
    temp->height=max(treeHeight(temp->left),treeHeight(temp->right))+1;
    return temp;
}

struct dictionaryADT * doubleRotatewithRight(struct dictionaryADT *D)
{
    D->right=singleRotatewithLeft(D->right);
    return singleRotatewithRight(D);
}

void disp(struct dictionaryADT *D,int space)

```



```

{
    if(D == NULL)
    {
        return;
    }
    space += 15;
    disp(D->right,space);
    printf("\n");
    for(int i=10;i<space;i++)
    {
        printf(" ");
    }
    printf("%s\n",D->w.word);
    for(int i=10;i<space;i++)
    {
        printf(" ");
    }
    printf("%s\n",D->w.meaning);
    disp(D->left,space);
}

```

```

struct dictionaryADT * search(struct dictionaryADT *D,char word[])
{
    if(D == NULL)
    {
        printf("TREE EMPTY\n");
    }
    else if(strcmp(word,D->w.word)<0)
    {
        return search(D->left,word);
    }
    else if(strcmp(word,D->w.word)>0)
    {
        return search(D->right,word);
    }
    else if(strcmp(word,D->w.word)==0)
    {
        printf("Found\n");
        return D;
    }
    printf("Not found\n");
}

```

DictionaryADTAppl.c

```
#include "DictionaryADTImph.h"

void main()
{
    int n;
    struct wordMeaning x;
    char s[20];
    struct dictionaryADT *D = NULL;
    do{
        printf("MENU\n1.Insert\n2.Display\n3.Search\n4.Exit\nEnter option: ");
        scanf("%d",&n);
        switch(n)
        {
            case 1:
                printf("Enter word: ");
                gets(x.word);
                printf("Enter meaning: ");
                gets(x.meaning);
                D = insert(D,x);
                break;

            case 2:
                disp(D,0);
                break;

            case 3:
                printf("Enter word: ");
                gets(s);
                puts(s);
                D=search(D,s);
                break;
        }
    }while(n!=4);
}
```

OUTPUT

INSERT TO AVL TREE

```

MENU
1.Insert
2.Display
3.Search
4.Exit
Enter option: 1
Enter word: ectomorphic
Enter meaning: skinny

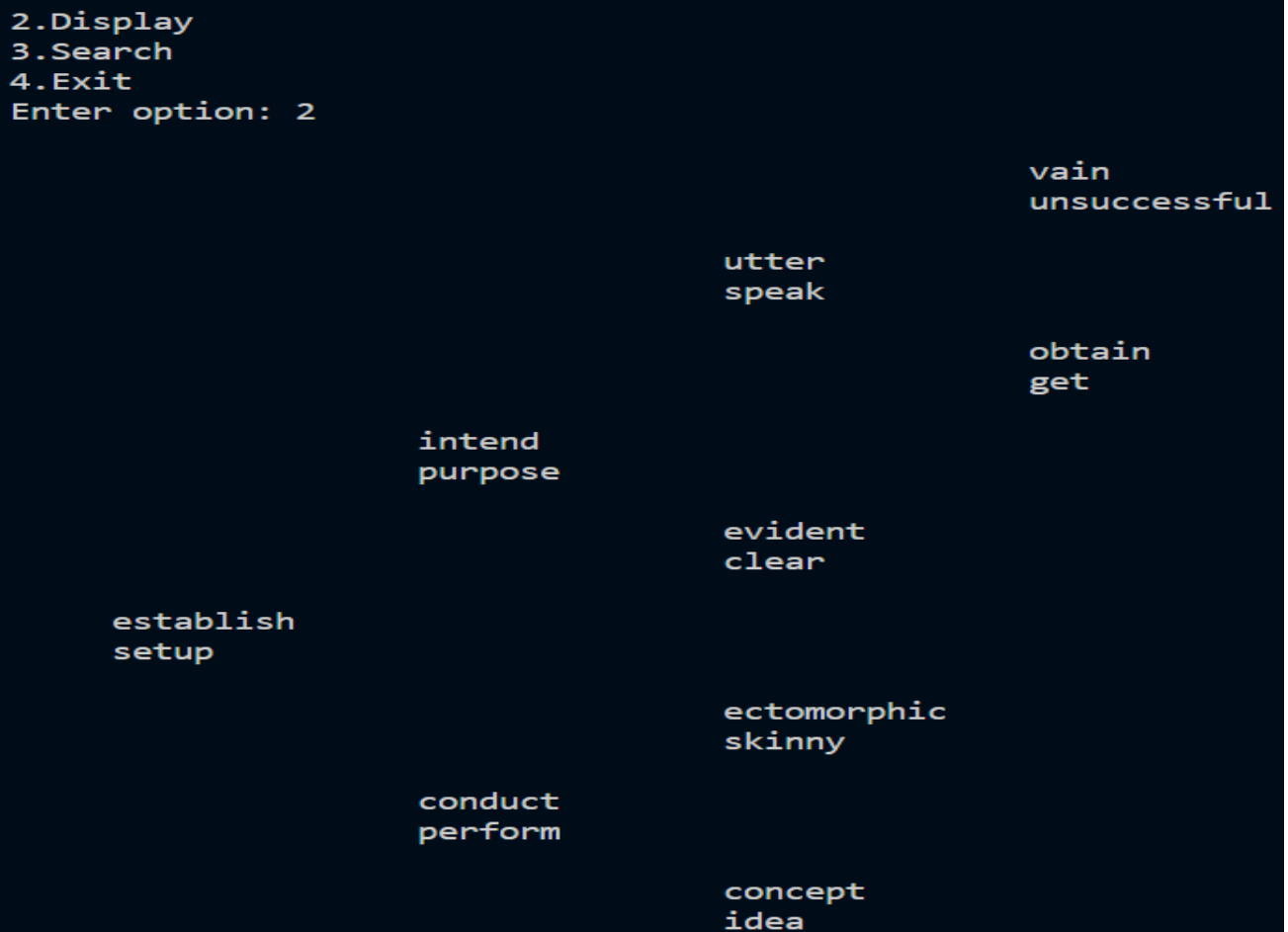
```

DISPLAY AVL TREE

```

2.Display
3.Search
4.Exit
Enter option: 2

```



The diagram shows an AVL tree structure. The root node is 'vain unsuccessful'. Its left child is 'utter speak', and its right child is 'obtain get'. The 'utter speak' node has a left child 'intend purpose'. The 'intend purpose' node has a left child 'establish setup'. The 'intend purpose' node also has a right child 'evident clear'. The 'evident clear' node has a right child 'ectomorphic skinny'. The 'ectomorphic skinny' node has a left child 'conduct perform'. The 'conduct perform' node has a right child 'concept idea'.

```

graph TD
    vain["vain  
unsuccessful"] --> utter["utter  
speak"]
    vain --> obtain["obtain  
get"]
    utter --> intend["intend  
purpose"]
    intend --> establish["establish  
setup"]
    intend --> evident["evident  
clear"]
    evident --> ectomorphic["ectomorphic  
skinny"]
    ectomorphic --> conduct["conduct  
perform"]
    conduct --> concept["concept  
idea"]

```

SEARCH IN AVL TREE (NOT FOUND)

```
MENU
1.Insert
2.Display
3.Search
4.Exit
Enter option: 3
Enter word: conduc
conduc
TREE EMPTY
Not found
```

SEARCH IN AVL TREE (FOUND)

```
MENU
1.Insert
2.Display
3.Search
4.Exit
Enter option: 3
Enter word: vain
vain
Found
```

LEARNING OUTCOME

- Dictionary application has been successfully implemented using AVL tree concepts
- Proper prompt messages and to make it more user friendly and presentable
- Incremental coding and modular programming was implemented in the code for easier debugging and better presentation

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021

Batch: 2019-2023

(Odd)

Exercise 8: Prioritization of Employees based on salary with Priority Queue Implementation using Binary Heap

Design a priority queue using max binary heap. An item in the priority queue consists of employee id, employee name and salary amount. The queue supports two operations, namely, insertion and deletion.

The priorityQueueADT consists of empDetails, left and right children. The structure empDetails has the employee-id, employee-name and salary. Implement the following methods.

- void insert(struct priorityQueueADT *P, struct empDetails x) – Insertion of the details of a new employee into priority queue
- void disp(struct priorityQueueADT *P) – Display the information about the employees (Hierarchically)
- empDetails* delete(struct priorityQueueADT *P) – Will remove the employee at the root of the max binary heap from the queue

Note:

In order to implement this dictionary application,

- It is necessary to create a file that has priorityQueue ADT.h with the data and prototypes of the above-mentioned functions
- Another file priorityQueue ADTImpl.h will be created with only function definitions
- One more file priorityQueue ADTAppl.c will be created to develop dictionary application that utilizes the priorityQueue ADT

EXERCISE 8

PRIORITIZATION OF EMPLOYEES BASED ON SALARY WITH PRIORITY QUEUE IMPLEMENTATION USING BINARY HEAP

Code:

priorityQueueADT.h

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

struct empDetails
{
    int id;
    char name[20];
    float sal;
};

struct priorityQueueADT
{
    int size,index;
    struct empDetails E[20];
};

void insert(struct priorityQueueADT *P, struct empDetails x);
void disp(struct priorityQueueADT *P);
struct empDetails deleteElt(struct priorityQueueADT *P);
```

priorityQueueADTImpl.h

```
#include"priorityQueueADT.h"

void init(struct priorityQueueADT *P,int size)
{
    P->size=size;
    P->index=0;
    P->E[0].sal=9999999999999999;
}
```

```

int isFull(struct priorityQueueADT* H)
{
    if(H->index<=H->size-1)
    {
        return 0;
    }
    return 1;
}

```

```

int isEmpty(struct priorityQueueADT* H)
{
    if(H->index==0)
    {
        return 1;
    }
    return 0;
}

```

```

void insert(struct priorityQueueADT *P, struct empDetails x)
{
    if(!isFull(P))
    {
        P->E[++P->index]=x;
        for(int i=P->index;P->E[i/2].sal<x.sal;i=i/2)
        {
            P->E[i]=P->E[i/2];
            P->E[i/2]=x;
        }
        printf("INSERTED\n");
    }
    else
    {
        printf("FULL!!!!\n");
    }
}

```

```

void disp(struct priorityQueueADT *P)
{
    if(!isEmpty(P))
    {
        for(int i=1;i<P->index+1;i++)
        {
            printf("NAME: %s\n",P->E[i].name);
            printf("EMPLOYEE ID: %d\n",P->E[i].id);
            printf("SALARY: %f\n",P->E[i].sal);
            printf("\n");
        }
    }
}

```

```

    }
}
else
{
    printf("EMPTY\n");
}
}

struct empDetails deleteElt(struct priorityQueueADT *P)
{
    int c= 0;
    struct empDetails t,max;
    max=P->E[1];
    P->E[1]=P->E[P->index];
    P->index--;
    for(int i=1;2*i<=P->index;i=c)
    {
        if(P->E[2*i].sal>P->E[2*i+1].sal)
        {
            c=2*i;
        }
        else
        {
            c=2*i+1;
        }
        if(P->E[i].sal<P->E[c].sal)
        {
            t=P->E[i];
            P->E[i]=P->E[c];
            P->E[c]=t;
        }
    }
    return max;
}

```

priorityQueueADTAppl.c

```

#include"priorityQueueADTImpl.h"
int main()
{
    struct priorityQueueADT *P;
    struct empDetails E;
    int n,o;
    P = (struct priorityQueueADT*)malloc(sizeof(struct priorityQueueADT));
    printf("Enter size of heap: ");
}

```



```

scanf("%d",&n);
init(P,n);
do
{
    printf("\nMENU\n1.INSERT\n2.DELETE\n3.DISPLAY\n4.EXIT\nEnter option: ");
    scanf("%d",&o);
    switch(o)
    {
        case 1:
            printf("\n");
            printf("Enter name of employee: ");
            scanf("%s",E.name);
            printf("Enter employee ID of employee: ");
            scanf("%d",&E.id);
            printf("Enter salary of employee: ");
            scanf("%f",&E.sal);
            insert(P,E);
            break;

            case 2:
                if(!isEmpty(P))
                {
                    E=deleteElt(P);
                    printf("\nName: %s\n",E.name);
                    printf("Employee ID: %d\n",E.id);
                    printf("Salary: %f\n",E.sal);
                    printf("\n");
                    break;
                }
                else
                {
                    printf("EMPTY\n");
                }

            case 3:
                disp(P);
                break;
    }
} while (o!=4);

return 0;
}

```

OUTPUT:**INITIALIZING HEAP, DISPLAY AND DELETE FROM A EMPTY HEAP**

```
Enter size of heap: 7
```

```
MENU
```

```
1.INSERT
```

```
2.DELETE
```

```
3.DISPLAY
```

```
4.EXIT
```

```
Enter option: 2
```

```
EMPTY
```

```
EMPTY
```

```
MENU
```

```
1.INSERT
```

```
2.DELETE
```

```
3.DISPLAY
```

```
4.EXIT
```

```
Enter option: 3
```

```
EMPTY
```

INSERTING TO A HEAP

```
MENU
```

```
1.INSERT
```

```
2.DELETE
```

```
3.DISPLAY
```

```
4.EXIT
```

```
Enter option: 1
```

```
Enter name of employee: Manas
```

```
Enter employee ID of employee: 7722
```

```
Enter salary of employee: 9133222.4
```

```
INSERTED
```

INSERTING TO A FULL HEAP

```
MENU
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter option: 1

Enter name of employee: Ram
Enter employee ID of employee: 5542
Enter salary of employee: 315542.44
FULL!!!!
```

DELETING FROM A HEAP

```
MENU
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
Enter option: 2

Name: Manas
Employee ID: 7722
Salary: 9133222.000000
```

DISPLAYING A HEAP

MENU

1.INSERT

2.DELETE

3.DISPLAY

4.EXIT

Enter option: 3

NAME: Manas

EMPLOYEE ID: 7722

SALARY: 9133222.000000

NAME: Shreya

EMPLOYEE ID: 7162

SALARY: 991882.437500

NAME: Sam

EMPLOYEE ID: 71442

SALARY: 614423.562500

NAME: Mary

EMPLOYEE ID: 8172

SALARY: 41332.500000

NAME: Sharan

EMPLOYEE ID: 9182

SALARY: 112334.500000

NAME: Devika

EMPLOYEE ID: 8777

SALARY: 5442.500000

NAME: Shraddha

EMPLOYEE ID: 8177

SALARY: 72663.437500

LEARNING OUTCOME

- Implementing priority queue using max heap based on salary of the employees
- Creation of three separate files to incorporate modular programming and for easier understanding and organization of code.
- Creating different versions and applying the concept of incremental coding to expand and debug the code easily.
- Proper usage of prompt messages to maintain user friendliness.

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021

Batch: 2019-2023

(Odd)

Exercise 9: Implementation of Hash Table using separate chaining collision resolution

strategy

The HashTableADT contains the size of hash table and list of addresses pointing to the corresponding lists. Hash function to be used for the insertion of elements is $x \bmod \text{tableSize}$.

- void init(HashTableADT *H) – To initialize the size and the header nodes of the lists
- void insertElement (HashTableADT *H, int x)– To insert the input key into the hash table using the hash function
- void searchElement(HashTableADT *H, int key) – Searching an element in the hash table
- void displayHT(HashTableADT *H) – Display the elements in the hash table

Note:

Implement HashTableADT with the specified operations in HashTableADTImpl.h

Write a menu driven application to utilize the HashTableADT

EXERCISE 9

IMPLEMENTATION OF HASH TABLE USING SEPARATE CHAINING COLLISION RESOLUTION STRATEGY

hashTableADT.h

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

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

struct HashTableADT
{
    int size;
    struct ListNode *mainHeader[50];
};

void init(struct HashTableADT *H);
void insertElement (struct HashTableADT *H, int x);
void searchElement(struct HashTableADT *H, int key);
void displayHT(struct HashTableADT *H);
```

hashTableADTImpl.h

```
#include"hashTableADT.h"

void init(struct HashTableADT *H)
{
    printf("Enter size: ");
    scanf("%d",&H->size);
    for(int i=0;i<H->size;i++)
    {
        H->mainHeader[i]=(struct ListNode*)malloc(sizeof(struct ListNode));
        H->mainHeader[i]=NULL;
    }
}
```

```

int hashFunction(struct HashTableADT *H, int x)
{
    return x%H->size;
}

void insertElement (struct HashTableADT *H, int x)
{
    struct ListNode *c,*temp;
    temp=(struct ListNode*)malloc(sizeof(struct ListNode));
    temp->data=x;
    temp->next=NULL;
    if(H->mainHeader[hashFunction(H,x)]==NULL)
    {
        H->mainHeader[hashFunction(H,x)]=temp;
    }
    else
    {
        c=H->mainHeader[hashFunction(H,x)];
        while(c->next!=NULL)
        {
            c=c->next;
        }
        c->next=temp;
    }
}

void searchElement(struct HashTableADT *H, int key)
{
    struct ListNode *c;
    int flag=0;
    if(H->mainHeader[hashFunction(H,key)]==NULL)
    {
        flag=0;
    }
    else
    {
        for(c=H->mainHeader[hashFunction(H,key)];c!=NULL;c=c->next)
        {
            if(H->mainHeader[hashFunction(H,key)]->data==key)
            {
                flag=1;
            }
        }
    }
}

```



```

    if(flag==1)
    {
        printf("FOUND\n");
    }
    else
    {
        printf("NOT FOUND\n");
    }
}

void displayHT(struct HashTableADT *H)
{
    struct ListNode *c;
    for(int i=0;i<H->size;i++)
    {
        printf("%d\t",i);
        c=H->mainHeader[i];
        if(H->mainHeader[i]==NULL)
        {
            printf("-");
        }
        else
        {
            for(c=H->mainHeader[i];c!=NULL;c=c->next)
            {
                printf("%d -> ",c->data);
            }
        }
        printf("null");
        printf("\n");
    }
}

```

hashTableADTAppl.c

```
#include "hashTableADTImpl.h"
```

```

void main()
{
    struct HashTableADT *H=(struct HashTableADT*)malloc(sizeof(struct HashTableADT));
    int n,x;
    init(H);
    printf("\nMENU\n1.Insert\n2.Search\n3.Display\n4.Exit\n");
}

```

```

do
{
    printf("Enter option: ");
    scanf("%d",&n);
    switch(n)
    {
        case 1:
            printf("Enter element to insert to the hashtable: ");
            scanf("%d",&x);
            insertElement(H,x);
            break;

        case 2:
            printf("Enter element to search in the hashtable: ");
            scanf("%d",&x);
            searchElement(H,x);
            break;

        case 3:
            printf("DISPLAY");
            printf("\n");
            displayHT(H);
            break;
    }
} while (n!=4);
}

```

OUTPUT:

INITIALISATION AND INSERTING ELEMENT TO THE HASHTABLE

```

Enter size: 8

MENU
1.Insert
2.Search
3.Display
4.Exit
Enter option: 1
Enter element to insert to the hashtable: 45

```

SEARCHING FOR AN ELEMENT IN HASHTABLE

```

Enter option: 2
Enter element to search in the hashtable: 45
FOUND
Enter option: 2
Enter element to search in the hashtable: 33
NOT FOUND

```

DISPLAYING A HASHTABLE

```

Enter option: 3
DISPLAY
0      32 -> 80 -> 56 -> null
1      -null
2      -null
3      43 -> 11 -> 91 -> null
4      -null
5      45 -> 5 -> null
6      78 -> null
7      23 -> null
Enter option: 4

```

LEARNING OUTCOME

- Implementation of hash table using the function `num%size` is done successfully
- Modular programming using 3 files to organize and understand the code better.
- Incremental coding practice for efficient debugging
- Usage of proper prompt messages for user input and to make it more user friendly.

SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
III Semester - CSE
UCS 1312 Data Structures Lab Laboratory

Academic Year: 2020-2021

Batch: 2019-2023

(Odd)

Exercise 10: Implementation of linear and binary searching techniques

The numberADT contains the size of integer array and an array of integers.

- void init(numberADT *N) – To initialize the size of the array
- void insertElements (numberADT *N, int x[10])– To copy the elements from x to the
array in numberADT
- void lsearchElement(numberADT *N, int key) – Searching an element in the
array
using linear search
technique
- void bsearchElement(numberADT *N, int key) – Searching an element in
the array
using binary search
technique
- void display(numberADT *N) – Display the elements in the array

Note:

Implement numberADT with the specified operations in numberADTImpl.h
 Write a menu driven application to utilize the numberADT

EXERCISE 10

IMPLEMENTATION OF LINEAR AND BINARY SEARCHING TECHNIQUES

numberADT.h

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

struct numberADT
{
    int size,arr[10];
};

void init(struct numberADT *N);
void insertElements (struct numberADT *N, int x[10]);
void lsearchElement(struct numberADT *N, int key);
void bsearchElement(struct numberADT *N, int key);
void display(struct numberADT *N);
```

numberADTImpl.h

```
#include"numberADT.h"

void init(struct numberADT *N)
{
    printf("Enter size of array: ");
    scanf("%d",&N->size);
}

void insertElements (struct numberADT *N, int x[10])
{
    for(int i=0;i<N->size;i++)
    {
        N->arr[i]=x[i];
    }
}

void lsearchElement(struct numberADT *N, int key)
{
    int f=0,pos;
```

```

for(int i=0;i<N->size;i++)
{
    if(N->arr[i]==key)
    {
        f=1;
        pos=i;
        break;
    }
}
if(f==1)
{
    printf("Element found at %d\n",pos);
}
else
{
    printf("Element not found\n");
}
}

void sort(struct numberADT *N)
{
    int temp;
    for(int i=0;i<N->size;i++)
    {
        for(int j=i+1;j<N->size;j++)
        {
            if(N->arr[i]>N->arr[j])
            {
                temp=N->arr[i];
                N->arr[i]=N->arr[j];
                N->arr[j]=temp;
            }
        }
    }
}

void bsearchElement(struct numberADT *N, int key)
{
    printf("SORTED ARRAY\n");
    sort(N);
    display(N);
    int f=0,mid,l=0,h=N->size-1,pos;
    while(l<=h)
    {
        mid=l+(h-l)/2;
        if(N->arr[mid]==key)

```

```

    {
        f=1;
        pos=mid;
        break;
    }
    else if(N->arr[mid]>key)
    {
        h=mid-1;
    }
    else if(N->arr[mid]<key)
    {
        l=mid+1;
    }
}
if(f==1)
{
    printf("Element found at %d\n",pos);
}
else
{
    printf("Element not found");
}
}

void display(struct numberADT *N)
{
    for(int i=0;i<N->size;i++)
    {
        printf("%d\t",N->arr[i]);
    }
}

```

numberADTAppl.c

```

#include"numberADTImpl.h"
void main()
{
    int n,x[10],key;
    struct numberADT* N=(struct numberADT*)malloc(sizeof(struct numberADT));
    printf("\nMENU\n1.Initialise\n2.Insert\n3.Display\n4.Linear search\n5.Binary search\n6.Exit\n");
    do
    {
        printf("\nEnter option: ");

```

```

scanf("%d",&n);
switch(n)
{
    case 1:
        init(N);
        break;

    case 2:
        for(int i=0;i<N->size;i++)
        {
            printf("Enter the array element %d: ",i+1);
            scanf("%d",&x[i]);
        }
        insertElements(N,x);
        break;

    case 3:
        display(N);
        break;

    case 4:
        printf("Enter key: ");
        scanf("%d",&key);
        lsearchElement(N,key);
        break;

    case 5:
        printf("Enter key: ");
        scanf("%d",&key);
        bsearchElement(N,key);
    }
} while (n!=6);
}

```

OUTPUT:

INITIALISING AN ARRAY


```
MENU
1.Initialise
2.Insert
3.Display
4.Linear search
5.Binary search
6.Exit

Enter option: 1
Enter size of array: 9
```

INSERING TO ARRAY

```
Enter option: 2
Enter the array element 1: 3
Enter the array element 2: 7
Enter the array element 3: 8
Enter the array element 4: 1
Enter the array element 5: 2
Enter the array element 6: 9
Enter the array element 7: 5
Enter the array element 8: 0
Enter the array element 9: 4
```

DISPLAYING ARRAY

```
h
Enter option: 3
3  7  8  1  2  9  5  0  4
```

LINEAR SEARCH

```
Enter option: 4
Enter key: 3
Element found at 0

Enter option: 4
Enter key: 11
Element not found
```

BINARY SEARCH

```
Enter option: 5
Enter key: 1
SORTED ARRAY
0      1      2      3      4      5      7      8      9      Element found at 1

Enter option: 5
Enter key: 11
SORTED ARRAY
0      1      2      3      4      5      7      8      9      Element not found
Enter option: 6
```

LEARNING OUTCOME

- Implementing linear and binary search using array concept successfully
- Creating versions for efficient debugging which saves time and works efficiently when it comes to handling larger codes
- Creating 3 separate files to apply modular programming and organization of code which makes it more presentable and easy to understand
- Usage of appropriate comments and prompt messages for user friendliness