[DATA STRUCTURES]

[LABSETS]

Abstract

Labset programs for semester-3 Informaton science and engineering.

harikesh bg

[harikeshbyrandurga21@gmail.com]

LABSET-1 **[Program to demonstrate file operations.]**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct employee
{
   unsigned int id;
   char name[25];
   char dept[25];
   unsigned int sal,age;
};
void readData(char filename[]);
void displayData(char filename[]);
void search(char filename[]);
void searchAge(char filename[],unsigned int sage);
void searchSalary(char filename[],unsigned int ssalary);
void searchId(char filename[],unsigned int sid);
void searchDept(char filename[],char sdept[]);
int main()
{
  char filename[30];
  int choice;
  printf("Enter the file name\n");
  scanf("%s",filename);

  do{
        printf("1.READ RECORD\n2.DISPLAY RECORD\n3.SEARCH RECORD\n4.EXIT\n");
        printf("Enter the choice:");
        scanf("%d",&choice);
        switch(choice)
```

```
        {
            case 1 : readData(filename);
                    break;
            case 2 : displayData(filename);
                    break;
            case 3 : search(filename);
                    break;
            case 4 : exit(0);
        }
  }while(1);
  return(0);
}
void readData(char filename[])
{
   FILE *fp;
   struct employee emp;
   char ch;
   if((fp=fopen(filename,"w"))!=NULL)
   {
      do{
            printf("Enter the details of the employee\n");
            printf("Employee id,Name,Department,Salary,Age\n");
            scanf("%d%s%s%d%d",&emp.id,emp.name,emp.dept,&emp.sal,&emp.age);
            fwrite(&emp,sizeof(struct employee),1,fp);
            printf("Do you want to continue?(y/n)\n");
            ch=getche();
            //scanf("%c",&ch);
       }while(ch=='y'||ch=='Y');
            //fprintf(fp,"%d%s%s%d%d",emp.id,emp.name,emp.dept,emp.sal,emp.age);

   }
```

```
   fclose(fp);
}
void displayData(char filename[])
{
   FILE*fp=NULL;
   struct employee emp;
   int ch;
   if((fp=fopen(filename,"r"))!=NULL)
   {
       printf(" EMPLOYEE ID   NAME                  DEPARTMENT           SALARY    AGE\n ");
      do
      {
          ch=fread(&emp,sizeof(struct employee),1,fp);
          if(ch==1)
            printf("%-11d   %-25s   %-25s %-6d    %-
3d\n",emp.id,emp.name,emp.dept,emp.sal,emp.age);
          else
            exit(1);
      }while(1);
   }
   fclose(fp);
}
void search(char filename[])
{
   int choice;
   unsigned int sage,ssal,sid;
   char sdept[25];
   do
   {
      printf("1.age\n 2.salary\n3.employee id\n4.dept\n5.exit\n");
      printf("Enter a key data for us to fetch the details of the employee\n");
```

```
        scanf("%d",&choice);
        switch(choice)
        {
           case 1 : printf("Enter age:");
                    scanf("%d",&sage);
                    searchAge(filename,sage);
                    break;
           case 2 : printf("Enter salary:");
                    scanf("%d",&ssal);
                    searchSalary(filename,ssal);
                    break;
           case 3 : printf("Enter employee ID:");
                    scanf("%d",&sid);
                    searchId(filename,sid);
                    break;
           case 4 : printf("Enter department:");
                    scanf("%s",sdept);
                    searchDept(filename,sdept);
                    break;
           default : return;
        }
    }while(1);
}
void searchAge(char filename[],unsigned int sage)
{
    FILE*fp=NULL;
    struct employee emp;
    int found=0;
    int ch;
    if((fp=fopen(filename,"r"))!=NULL)
    {
```

```
        do
        {
           ch=fread(&emp,sizeof(struct employee),1,fp);
           if(ch==1)
           {
              if(emp.age==sage)
              {
                 found=1;
                 printf("Here is the details of the employee of age %d\n",sage);
                 printf("Employee ID : %d\n",emp.id);
                 printf("Name : %-25s\n",emp.name);
                 printf("Department : %-25s\n",emp.dept);
                 printf("Salary : %d\n",emp.sal);
                 printf("Age : %d\n",emp.age);
              }
           }
           else
              break;
        }while(1);
        if(found==0)
           printf("Record not found\n");
        fclose(fp);
     }
}
void searchId(char filename[],unsigned int sid)
{
   FILE*fp=NULL;
   struct employee emp;
   int found=0;
   int ch;
   if((fp=fopen(filename,"r"))!=NULL)
```

```
    {
        do
        {
            ch=fread(&emp,sizeof(struct employee),1,fp);
            if(ch==1)
            {
                if(emp.id==sid)
                {
                    found=1;
                    printf("Here is the details of the employee of id %d\n",sid);
                    printf("Employee ID : %d\n",emp.id);
                    printf("Name : %-25s\n",emp.name);
                    printf("Department : %-25s\n",emp.dept);
                    printf("Salary : %d\n",emp.sal);
                    printf("Age : %d\n",emp.age);
                }
            }
            else
                break;
        }while(1);
        if(found==0)
            printf("Record not found\n");
        fclose(fp);
    }
}
void searchSalary(char filename[],unsigned int ssal)
{
    FILE*fp=NULL;
    struct employee emp;
    int found=0;
    int ch;
```

```
    if((fp=fopen(filename,"r"))!=NULL)
    {
       do
       {
          ch=fread(&emp,sizeof(struct employee),1,fp);
          if(ch==1)
          {
             if(emp.sal==ssal)
             {
                found=1;
                printf("Here is the details of the employee of salary %d\n",ssal);
                printf("Employee ID : %d\n",emp.id);
                printf("Name : %-25s\n",emp.name);
                printf("Department : %-25s\n",emp.dept);
                printf("Salary : %d\n",emp.sal);
                printf("Age : %d\n",emp.age);
             }
          }
          else
             break;
       }while(1);
       if(found==0)
          printf("Record not found\n");
       fclose(fp);
    }
}
void searchDept(char filename[],char sdept[])
{
    FILE*fp=NULL;
    struct employee emp;
    int found=0;
```

```
    int ch;
    if((fp=fopen(filename,"r"))!=NULL)
    {
       do
       {
          ch=fread(&emp,sizeof(struct employee),1,fp);
          if(ch==1)
          {
             if(strcmp(emp.dept,sdept)==0)
             {
                found=1;
                printf("Here is the details of the employee of department %s\n",sdept);
                printf("Employee ID : %d\n",emp.id);
                printf("Name : %-25s\n",emp.name);
                printf("Department : %-25s\n",emp.dept);
                printf("Salary : %d\n",emp.sal);
                printf("Age : %d\n",emp.age);
             }
          }
          else
             break;
       }while(1);
       if(found==0)
          printf("Record not found\n");
       fclose(fp);
    }
}
```

LABSET-2 **[Program to perform stack operations.]**

```
#include <stdio.h>
#include <stdlib.h>
#define size 7
struct stack
{
   int data[size];
   int top;
};
struct stack s;
int isempty();
int isoverflow();
int isunderflow();
void push(int element);
int pop();
void display();
int isempty()
{
   if(s.top==-1)
      return 1;
   else
      return 0;
}
int isoverflow()
{
   if(s.top>=size-1)
      return 1;
   else
      return 0;
}
```

```
int isunderflow()
{
   if(s.top==-1)
      return 1;
   else
      return 0;
}
void push(int element)
{
   if(isoverflow())
   {
      printf("stack overflown...\n");
      return;
   }
   s.data[++s.top]=element;
}
int pop()
{
   if(isunderflow())
   {
      printf("stack is underflown....\n");
      return -1;
   }
   return(s.data[s.top--]);
}
void display()
{
   int index;
   if(isempty())
   {
      printf("stack is empty...\n");
```

```
        return;
    }
    printf("stack elements are:\n");
    for(index=s.top;index>=0;index--)
        printf("%d\n",s.data[index]);
}
int main()
{
    int element,choice;
    s.top=-1;
    do
    {
        printf("1.push\n2.pop\n3.Display\n4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("enter the value to be pushed:");
                     scanf("%d",&element);
                     push(element);
                     break;
            case 2 : element=pop();
                     if(element!=-1)
                       printf("Popped item is %d",element);
                     break;
            case 3 : display();
                     break;
            case 4 : exit(0);
                     break;
            default: printf("Invalid choice....\n");
        }
```

```
    }while(1);
}
```

LABSET-3 **[Program to convert infix expression to it's equivalent postfix expression.]**

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<math.h>
#define size 50
int precedence(char ch);
void infix_postfix();
void push(int e);
char pop();
struct stack
{
   char data[size];
   int top;
};
struct stack s;
char infix[100];
char postfix[100];
void push(int e)
{
   s.data[++s.top]=e;
}
char  pop()
{
   return(s.data[s.top--]);
}
main()
{
   int k=0;
```

```
    s.top=0;
    s.data[s.top]='@';
    printf("Enter the valid infix expression:\n");
    gets(infix);
    infix_postfix();
}
int precedence(char ch)
{
    switch(ch)
    {
        case '@' : return 0;break;
        case '(' : return 1;break;
        case '+' :
        case '-' : return 2;break;
        case '*' :
        case '%' :
        case '/' : return 3;break;
        case '^' : return 4;break;

    }
}
void infix_postfix()
{
    int i=0,j=0;
    char dat,ch;
    while(infix[i]!='\0')
    {
        dat=infix[i];
        if(dat=='(')
            push(dat);
        else
```

```
    if(dat==')')
    {
       ch=pop();
       while(ch!='(')
          {
             postfix[j++]=ch;
             ch=pop();
          }
    }
    else
    if(isalpha(dat)||isdigit(dat))
       {
          postfix[j++]=dat;
       }

    else
    {
       if(precedence(dat)>precedence(s.data[s.top]))
             push(dat);
       else
          if(precedence(dat)<=precedence(s.data[s.top]))
             {
                while(precedence(dat)<=precedence(s.data[s.top]))
                      postfix[j++]=pop();
                push(dat);
             }
    }
    i++;
}
while(s.data[s.top]!='@')
    postfix[j++]=pop();
```

```
    postfix[j]='\0';
    printf("Postfix expression:");
    puts(postfix);
}
```

LABSET-4 **[Program to evaluate the given prefix expression.]**

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define size 50

struct stack
{
   int data[size];
   int top;
};
struct stack s;
int values[26][2];
void push(int e)
{
   s.data[++s.top]=e;
}
int pop()
{
   return(s.data[s.top--]);
}
char prefix[50];
main()
{
   s.top=0;
   int i=0,op1,op2,res;
   char ch;
   printf("Enter a valid prefix expression:");
   gets(prefix);
```

```
puts(prefix);
strrev(prefix);
while(prefix[i]!='\0')
{
   ch=prefix[i];
   if((isdigit(ch)))
   {
     push(ch-'0');
     i++;
   }
    else
   if(isalpha(ch))
   {
         if(isupper(ch))
            ch=tolower(ch);
         if(values[ch-97][1]==0)
            {
               printf("Enter the value of %c:",ch);
               scanf("%d",&values[ch-97][0]);
               printf("%d",values[ch-97][0]);
               values[ch-97][1]=1;
               push(values[ch-97][0]);
               printf("stacktop=%d\n",s.data[s.top]);
               i++;

            }
         else
            {
               push(values[ch-97][0]);
               i++;
            }
```

```
        }
        else
        {
          op1=pop();
          op2=pop();
          switch(ch)
          {
            case '+': res=op1+op2;
                    break;
            case '-': res=op1-op2;
                    break;
            case '*': res=op1*op2;
                    break;
            case '/': res=op1/op2;
                    break;
            case '%': res=op1%op2;
                    break;
          }
          push(res);
          i++;

        }

    }
    printf("result=%d",res);
    i++;
}
```

LABSET-5 **[Program to demonstrate queue operations.]**

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define size 3
void insert(int element);
void del();
void display();
int overflow();
int underflow();
struct queue
{
   int data[size];
   int front;
   int rear;
}s;
main()
{
   s.front=-1;
   s.rear=-1;
   int choice,element;
   do
      {
         printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
         printf("Enter your choice:");
         scanf("%d",&choice);
         switch(choice)
         {
            case 1 : printf("Enter the element to be inserted:");
```

```
                scanf("%d",&element);
                insert(element);
                break;
            case 2 : del();
                break;
            case 3 : display();
                break;
            case 4 : exit(0);
                break;
            default: printf("Invalid choice\n");
                break;
          }
       }while(1);
}

void insert(int element)
{
   if(overflow())
     {
      printf("Queue is overflown..\n");
      return;
     }
   else
     {
        s.data[++s.rear]=element;
        if(s.front==-1)
          s.front=0;
        printf("front=%d\n",s.front);
        printf("rear=%d\n",s.rear);
     }
}
```

```
void del()
{
   if(underflow())
   {
      printf("queue under flown...\n");
      return;
   }
   else
      {
         printf("deleted item is:%d\n",s.data[s.front]);
         s.front++;
         printf("front=%d\n",s.front);
         printf("rear=%d\n",s.rear);
      }
}
void display()
{
   int n=s.rear;
   if(s.rear==-1)
      printf("queue is empty\n");
   else
   {
      printf("Queue elements are:\n");
      while(n>=s.front)
         printf("%d\n",s.data[n--]);
   }
}
int overflow()
{
   if(s.rear>=size-1)
      return 1;
```

```
    else
        return 0;
}
int underflow()
{
    if(s.rear==-1&&s.front==-1)
        return 1;
    else
        return 0;
}
```

LABSET-6 **[Program to implement circular queues.]**

```
#include<stdio.h>
#include<stdlib.h>
#define size 3
struct queue
{
   int data[size];
   int front,rear;
};
struct queue q;
void insert()
{
   if((q.front==0&&q.rear==size-1)||(q.rear==q.front-1))
     {
        printf("Queue overflown...\n");
        return;
     }
   if(q.front==-1)
      q.front=q.rear=0;
   else
     {
        q.rear=(q.rear+1)%size;
     }
   printf("Enter the element to be inserted:");
   scanf("%d",&q.data[q.rear]);
   printf("Inserted element is:%d\n",q.data[q.rear]);
}
void del()
{
   if(q.rear==-1)
```

```
    {
        printf("Queue underflown..\n");
        return;
    }
    else
    {
        if(q.front==q.rear)
            {
                printf("Deleted element is:%d\n",q.data[q.front]);
                q.front=q.rear=-1;
            }
        else
        {
            printf("Deleted element is:%d\n",q.data[q.front]);
            q.front=(q.front+1)%size;
        }
    }
}
void display()
{
    int temp;
    if(q.rear==-1)
        printf("Queue is empty..\n");
    else
    {
        temp=q.front;
        printf("Queue elements are:\n");
        while((temp%size)!=q.rear)
        {
            printf("temp=%d\n",temp);
            printf("%d\t",q.data[temp%size]);
```

```
            temp++;
        }
        temp=temp%size;
        printf("temp=%d\n",temp);
        printf("%d",q.data[temp]);
        printf("\n");
    }
}
int main()
{
    q.front=-1;
    q.rear=-1;
    int choice;
    do
    {
        printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:insert();break;
            case 2:del();break;
            case 3:display();break;
            case 4:exit(0);break;
            default:printf("Invalid choice..");break;
        }
    }while(1);
    return 0;
}
```

LABSET-7 **[Program to implement linked list.]**

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

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

typedef struct Node *NODE;

void display(NODE list)
{
   NODE temp;
   if(list==NULL)
   {
      printf("List is Empty...\n");
      return;
   }
   temp=list;
   printf("The list elements are...\n");
   while(temp!=NULL)
   {
      printf("%d\t",temp->info);
      temp=temp->next;
   }
   printf("\n");
}

NODE deletion(NODE list)
{
   NODE temp;
   if(list==NULL)
   {
      printf("List doesn't contains any elements...\n");
      return(list);
   }
   temp=list;
   list=list->next;
   printf("Deleted element is %d\n",temp->info);
   free(temp);
   return(list);
}

NODE insert(NODE list,int ele)
{
   NODE newn,temp;
   newn=(NODE)malloc(sizeof(struct Node));
   if(newn==NULL)
```

```
    {
       printf("Memory allocation error...\n");
       return(list);
    }
    newn->info=ele;
    newn->next=NULL;
    if(list==NULL)
    {
       list=newn;
       return(list);
    }
    else
    {
       temp=list;
       while(temp->next!=NULL)
          temp=temp->next;
       temp->next=newn;
    }
    return(list);
}

NODE insertAtPos(NODE list,int ele,int pos)
{
    NODE newn,temp,prev;
    int c=2;
    newn=(NODE)malloc(sizeof(struct Node));
    if(newn==NULL)
    {
       printf("Memory allocation error...\n");
       return(list);
    }
    newn->info=ele;
    newn->next=NULL;
    if(list==NULL&&pos==1)
       return(newn);
    if(pos==1)
    {
       newn->next=list;
       return(newn);
    }
    temp=list->next;
    prev=list;
    while(temp!=NULL&&c!=pos)
    {
       prev=temp;
       temp=temp->next;
       c++;
    }
    if(temp==NULL&&pos==c)
       prev->next=newn;
    else if(pos>=c&&temp!=NULL)
```

```
        {
           prev->next=newn;
           newn->next=temp;
        }
        else
          printf("Invalid position...\n");
   return(list);
}

NODE reverse(NODE list)
{
   NODE temp,prev;
   if(list==NULL||list->next==NULL)
      return(list);
   temp=list->next;
   prev=list;
   list->next=NULL;
   while(temp!=NULL)
   {
      prev=temp;
      temp=temp->next;
      prev->next=list;
      list=prev;
   }
   return(list);
}

int main()
{
   int ele,in,pos,ch;
   NODE start=NULL;
   do
   {
      printf("1.Insertion\n2.Delete\n3.Display\n4.Reverse the list\n5.Exit\nEnter your choice:");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1 :printf("1.Inserting a node at end\n2.Inserting a node at the specified position\nEnter
your choice:");
                 scanf("%d",&in);
                 switch(in)
                 {
                    case 1 :printf("Enter element to be insert:");
                          scanf("%d",&ele);
                          start=insert(start,ele);
                          break;
                    case 2 :printf("Enter an element to be insert:");
                          scanf("%d",&ele);
                          printf("Enter the position where the element %d is to be insert:",ele);
                          scanf("%d",&pos);
                          start=insertAtPos(start,ele,pos);
```

```
                    break;
                default:printf("Invalid choice...\n");
                    break;
                }
                break;
        case 2 :start=deletion(start);
                break;
        case 3 :display(start);
                break;
        case 4 :start=reverse(start);
                break;
        case 5 :exit(1);
                break;
        default:printf("Invalid choice...\n");
                break;
      }
   }while(1);
   return(0);
}
```

LABSET-8 **[Program to perform union and intersection of two lists.]**

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int info;
   struct node *next;
};
typedef struct node *NODE;
NODE ulist=NULL,ilist=NULL;
NODE insert_sort(NODE list,int e)
{
   NODE newN,temp=NULL,prev=NULL;
   newN=(NODE)malloc(sizeof(struct node));
   newN->info=e;
   newN->next=NULL;
   temp=list;
   if(newN==NULL)
   {
      printf("Memory allocation error..");
      return list;
   }
   if(list==NULL)
   {
      list=newN;
      return list;

   }
   else
   {
      temp=list;
      prev=list;
      if(newN->info<list->info)
      {
         newN->next=list;
         list=newN;
         return list;
      }
      else
      {
         while(temp!=NULL&&newN->info>temp->info)
         {
            prev=temp;
            temp=temp->next;
         }
         newN->next=prev->next;
         prev->next=newN;
         return list;
```

```
        }
    }
}
void display(NODE list)
{
    NODE temp;
    if(list==NULL)
    {
        printf("List is empty..");
        return;
    }
    temp=list;
    while(temp!=NULL)
    {
        printf("%d\t",temp->info);
        temp=temp->next;
    }
}
void uni(NODE l1,NODE l2)
{
    int status;
    if(l1==NULL)
    {
        ulist=l2;
        return;
    }
    if(l2==NULL)
    {
        ulist=l1;
        return;
    }
    while(l1!=NULL)
    {
        status=search(ulist,l1->info);
        if(status==0)
        {
            ulist=insert_sort(ulist,l1->info);
            l1=l1->next;
        }
        else
            l1=l1->next;
    }
     while(l2!=NULL)
    {
        status=search(ulist,l2->info);
        if(status==0)
        {
            ulist=insert_sort(ulist,l2->info);
            l2=l2->next;
        }
        else
```

```
            l2=l2->next;
        }
    }
    void inter(NODE l1,NODE l2)
    {
        int status;
        if(l1==NULL||l2==NULL)
        {
            ilist=NULL;
            return;
        }
        while(l1!=NULL)
        {
            status=search(l2,l1->info);
            if(status==1&&!(search(ilist,l1->info)))
                ilist=insert_sort(ilist,l1->info);
                l1=l1->next;
        }
         while(l2!=NULL)
        {
            status=search(l1,l2->info);
            if(status==1&&!(search(ilist,l2->info)))
                ilist=insert_sort(ilist,l2->info);
                l2=l2->next;
        }

    }
    int search(NODE list,int e)
    {
        NODE temp;
        temp=list;
        while(temp!=NULL)
        {
            if(temp->info==e)
                return 1;
            else
                temp=temp->next;
        }
        return 0;
    }
    main()
    {

        int choice,e;
        NODE l1=NULL,l2=NULL;
        do
        {
            printf("\n1.Insert-l1\n2.Insert-l2\n3.Display-l1\n4.Display-
    l2\n5.Union\n6.Intersection\n7.Display-ulist\n8.Display-ilist\n9.Exit\n");
            printf("Enter your choice:");
            scanf("%d",&choice);
```

```
        switch(choice)
        {
        case 1:printf("Enter the element:");
               scanf("%d",&e);
               l1=insert_sort(l1,e);break;
        case 2:printf("Enter the element:");
               scanf("%d",&e);
               l2=insert_sort(l2,e);break;
        case 3:printf("\nElements of list-1:\n");
               display(l1);break;
        case 4:printf("\nElements of list-2:\n");
               display(l2);break;
        case 5:uni(l1,l2);break;
        case 6:inter(l1,l2);break;
        case 7:printf("\nUnion of two lists:\n");
               display(ulist);break;
        case 8:printf("\nIntersection of two lists:\n");
               display(ilist);break;
        case 9:exit(0);break;
        default:printf("invalid choice..");break;

        }
      }while(1);

}
```