18-09-2020

DS Lab: Program 1 (Admission)

```
#include <stdio.h>
struct student
{
  char name[50];
  int age, marks;
};
S[n];
int main()
{
  int i,n;
  printf("Enter the number of students\n");
  scanf("%d",&n);
  printf("Enter student information");
  printf("Enter no. of students\n");
  scanf("%d",&n);
  S[n];
  for(i=0;i<=n;i++)
  {
    printf("Roll number: \n",i);
    printf("Enter age: \n");
    scanf("%d",&S[i].age);
    printf("Enter name: \n");
    scanf("%s",&S[i].name);
    printf("Enter marks: \n");
    scanf("%d",&S[i].marks);
    if(age<20 || marks<65){
      printf("Not Eligible\n");
      break;
```

```
if(age>20 && marks>650){
    printf("selected");
    printf("%s", s[i].name);
}

if(age<0 || marks<0){
    printf("Invalid");
    break;
}

return 0;
}
```

25-09-2020

Week 2: Stack Implementation (Program 2)

1. Soln.

```
#include <stdio.h>
#include <stdlib.h>
#define max 5
int top=-1, stack[max];
void push();
void pop();
void display();
void main()
{
  int ch;
  while(1)
  {
    printf("\n Stack Menu");
    printf("\n 1. Push \n 2. Pop \n 3. Display \n 4. Exit");
    printf("\n Enter your choice \n");
    scanf("%d", &ch);
    switch(ch)
    {
      case 1: push();
              break;
      case 2: pop();
              break;
      case 3: display()
              break;
      case 4: Exit(0)
              break;
      default: printf("\n Wrong Choice");
    }
  }
}
```

```
void push()
{
  int val;
  if (top == max-1)
  {  printf("\n Stack is Full \n");
  else {
       printf("\n Enter Element to push \n");
       scanf("%d", &val);
       top += 1;
       stack[top] = val;
      }
}
void pop()
{
    if (top == -1)
        printf(" Stack is empty \n");
    else {
    {  printf(" Enter element to pop ");
       scanf("%d", &val);
       top -= 1
        printf("Deleted element is: %d", stack[top]);
        top -= 1;
       }
}
void display()
{
   int i;
   if(top == -1)
       printf("Stack is empty \n");
```

```
else {
    printf("Stack is: \n");
    for (i = top; i >= 0; --i);
        printf("%d", stack[i]);
    }
}
```

Week 2: Stack Using Pointer: (Program 3)

Sol^n 2.)

```
#include <stdio.h>
void push (int stack[], int ele, int *top, int *size){
{
    if (*top == *size - 1)
        printf("Stack Overflow \n");
    else {
        (*top)++;
        stack[*top] = ele; }
}
int pop (int stack[], int *top, int *size)
{
    if (*top == -1){
        printf("Stack Underflow \n");
        return -1; }
    else {
        int x = stack[*top];
        (*top)--;
        return x;
    }
}
```

```
void display (int stack[], int *top, int *size)
{
    int i;
    if (*top == -1)
        printf("stack is empty \n");
    else {
        for (i = *top; i >= 0; i--)
            printf("%d \n", stack[i]);
    }
}
int main()
{
    int size = 5;
    int top = -1;
    int stack[size];
    int n=1, ele;
    while (n != 0)
    {
        printf("Menu \n 1: Push \n 2: Pop \n
                3: Display \n 0: Exit \n");
        printf("Enter your choice \n");
        scanf("%d", &n);
        if (n==0)
            break;
        if(n==1) {
            printf("Enter Element: \n");
            scanf("%d", ele);
            push(stack, ele, &top, &size);
        }
```

```
else if (x == 2){
    int popele = pop(stack, &top, &size);
    printf("The element popped out is : %d", popele);
}
else if (x == 3){
    printf("The stack is : \n");
    display(stack, &top, &size);
}
}
return 0;
}
```

Week 3: Lab 3: Infix to Postfix (Program 4)

```
#include <stdio.h>
#define max 50
char stack[max];
int top = -1;
void push(char ch)
{
    if(top == max-1)
        printf("Stack is full\n");
    else{
        top++;
        stack[top] = ch;
    }
}
char pop()
{
    char item;
    if (top == -1)
        printf("Stack is empty\n");
    else{
        item = stack[top];
        top = top-1;
        return item;
    }
}
int empty ()
{
    if (top == -1)
        return 1;
```

```
    else return 0;
}
char stop()
{
    if (top == -1)
        printf("\n Stack is empty \n");
    else
        return stack[top];
}
int pri(char ch)
{
    switch(ch)
    {
        case '+':
        case '-': return (1);
        case '*':
        case '/': return (2);
        case '^': return (3)
        default: return(0);
    }
}
int main(int argc, char **argv)
{
    char infin[100];
    int i, item;
    printf("Enter the infin expression \n");
    scanf("%s", infin);
    printf("Expression is: %s \n", infin);
```

```
printf("Postfix: \n");
int i=0;
while (infix[i]!='\0')
{
    switch(infix[i])
    {
        case '(': push(infix[i]);
                  break;
        case ')': while ((item==pop())!='(')
                      printf("%c", item);
                  break;
        case '+':
        case '-':
        case '*':
        case '/':
        case '^': while (!empty() && pri(infix[i])<=
                                     pri(stop()))
                  {
                      item=pop();
                      printf("%c", item);
                  }
                  push(infix[i]);
                  break;
        default: printf("%c", infix[i]);
                 break;
    }
    i++;
}
```

```
while (! sempty ())
{
    char item;
    item = pop();
    printf("%c", item);
}
printf("\n");
return 0;
}
```

16-10-2020

Week 4: ~~Lab~~ Implement Queue (Program 5)

Sol^n:

```
#include <stdio.h>
#include <stdlib.h>
#define max 5;
int queue[max]
void insert (int val, int *front, int *rear)
{
    if (*rear == max-1)
        printf("The given queue is full \n");
    else {
        if (*front == -1)
            *front = 0;
        (*rear)++;
        queue[*rear] = val;
        printf("Enqueue succesful \n");
    }
void del (int *front, int *rear)
{
    if (*front == *rear)
        printf("Queue empty condition \n");
    else {
        printf("Deleted element is : %d \n", queue
        *front += 1;
        if (*front == *rear)
            *front = *rear = -1;
    }
}
void display (int *front, int *rear)
{
```

```
if (*rear == -1)
    printf("Queue empty condition \n");
else {
    int i;
    printf("Queue elements are\n");
    for(i = *front; i <= *rear; i++)
        printf("%d \t", queue[i]);
}
}
int main()
{
    int val, ch;
    int front = -1, rear = -1;
    while(1)
    {
        printf("Enter the choice \n");
        printf("1. Insert \n 2. Delete \n 3. Display \n");
        scanf("%d", &ch)
        switch(ch) {
            case 1: printf("Enter the value to be inserted \n");
                    scanf("%d", &val);
                    insert(val, &front, &rear);
                    break;
            case 2: del(&front, &rear);
                    break;
            case 3: display(&front, &rear);
                    break;
            case 4: return 0;
```

default: printf("\n Wrong choice");

}

}

return 0;

}

6-11-2020

CircularQueue. (Program 6)

```
#include <stdio.h>
# define size 5
int front = -1;
int rear = -1
int queue[size];
void enqueue (int n){
{
    if (front == 0 && rear == size-1)
        printf ("Queue is full");
    else if (front == rear+1)
        printf ("Queue is full");
    else if (front == -1 && rear == -1)
    {
        front ++;
        rear ++;
        queue[rear] = n;
    }
    else {
        rear = (rear+1) % size;
        queue[rear] = n;
    }
}
int dequeue ()
{
    if (front == -1 && rear == -1)
        return -1;
    else {
        int ele;
```

```
    ele = queue[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    }
    else
        front = (front+1) % size;
    return ele;
}
void display ()
{
    if (front == -1 && rear == -1)
        printf ("Queue is empty");
    else {
        printf ("Queue contents:\n");
        if (front <= rear) {
            for (int i = front; i <= rear; i++)
                printf ("%d\n", queue[i]);
        }
        else {
            for (int i = front; i <= size-1; i++)
                printf ("%d\n", queue[i]);
            for (int i = 0; i <= rear; i++)
                printf ("%d\n", queue[i]);
        }
    }
}
```

```
int main()
{
    int option;
    int item;
    do {
        printf("\n Circular Queue \n");
        printf("1. Insert to Queue \n 2. Delet from Queue
                \n 3. Display Queue \n 4. Exit \n");
        printf("Enter your choice \n");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the element \n");
                    scanf("%d", &item);
                    enqueue(item);
                    break;
            case 2: item = dequeue();
                    if (item == -1)
                        printf("Queue is empty \n");
                    else
                        printf("Removed element from Queue \n", item);
                    break;
            case 3: display();
                    break;
            case 4: exit(0)
        }
    } while (option != 4);
    return 0;
}
```

20-11-2020 (Deletion from Linked List) (Program 7)

```
#include<stdio.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head=NULL;
int length=0;
void insertend(int ele)
{
    struct node *newnode, *temp;
    newnode=(struct node*) malloc(sizeof(struct node));
    newnode->data=ele;
    newnode->next=NULL;
    if(head==NULL)
    {   head=newnode;
        length=1;
    }
    else
    {
        temp=(struct node*) malloc(sizeof(struct node));
        temp=head;
        while(temp->next!=NULL)
            temp=temp->next;
        temp->next=newnode;
        length++;
    }
}
void deletefront()
```

```
{
    if length == 0)
        printf("List is empty \n");
    else {
        struct node * temp;
        temp = (struct node *) malloc (sizeof (struct node));
        temp = head;
        head = head -> next;
        temp -> next = NULL;
        length--;
        printf("\n The element deleted is : %d", temp->data);
    }
}
void deleteend ()
{
    if (length == 0)
        printf("List is empty \n");
    else {
        struct node * temp;
        temp = (struct * node) malloc (sizeof (struct node));
        temp = head;
        while (temp -> next -> next != NULL)
            temp = temp -> next;
        struct node * del;
        del = (struct node*) malloc (sizeof (struct node));
        del = temp -> next;
        temp -> next = NULL;
        length--;
```

```
        printf("Element deleted is \n%d", del->data);
    }
}
void deleterandom(int pos)
{
    if(length ==0)
        printf("\n List is empty \n");
    else if(pos==1)
        deletefront();
    else if(pos >= length+1)
        deleteend();
    else {
        struct node *del;
        del = (struct node *) malloc(sizeof(struct node));
        struct node *temp;
        temp=(struct node*) malloc(sizeof(struct node));
        temp= head;
        for(int i=1;i<pos-1;i++)
            temp = temp->next;
        del= temp->next;
        temp->next = del->next;
        del->next = NULL;
        length--;
        printf("\n The element deleted is: \n%d",del->data);
    }
}
void display()
{
```

```
struct node * temp;
temp = (struct node *) malloc (sizeof(struct node));
temp = head;
if (temp == NULL)
{
    printf("List is empty \n");
else {
    printf("Contents of the list are: \n");
    while (temp != NULL)
    {
        printf("%d \n", temp -> data);
        temp = temp -> next;
    }
}
}
int main()
{
    int choice, ele, pos;
    char ch;
    do {
        printf("\n 1. Insert at end \n 2. Display \n 3. Delete
            from front \n 4. Delete from rear \n 5. Delete
            from random position \n 6. Exit ");
        printf("Enter your choice ");
        scanf("%d", & choice);
        switch (choice)
        {
            case 1: printf("Enter the element to be inserted \n");
                scanf("%d", & ele);
```

```
insertend(ele);
break;
case 2: display();
        break;
case 3: deletefront();
        break
case 4: deleteend();
        break;
case 5: deleteanypos();
        printf("\n Enter the position \n");
        scanf("%d", &pos);
        deleteanypos(pos);
        break;
} while (choice != 6)
return 0;
}
```

(Program 8)

20-11-2020 (Insertion to Linked List)

```
#include <stdio.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head=NULL;
int length=0;
void insertend(int ele)
{
    struct node *newnode, *temp;
    newnode=(struct node *)malloc(sizeof(struct node));
    newnode->data=ele;
    newnode->next=NULL;
    if(head==NULL)
    {
        head=newnode;
        length=1;
    }
    else{
        temp=(struct node*)malloc(sizeof(struct node));
        newnode->data=ele;
        newnode->next=NULL;
        if(head==NULL){
            head=newnode;
            length=1;
        }
        else
```

```
        temp = (struct node *) malloc (sizeof (struct node));
        temp = head;
        while (temp -> next != NULL)
                temp = temp -> next;
        temp -> next = newnode;
        length++;
    }
}
void insertfront (int ele)
{
    struct node * temp;
    temp = (struct node*) malloc (sizeof ( struct node ));
    temp -> data = ele;
    temp -> next = head;
    head = temp;
    length++;
}
void insertrandom (int ele, int pos)
{
    if(pos == 1)
        insertfront (ele);
    else if ( pos >= length + 1)
        insertend (ele);
    else {
        struct node *inst;
        inst = (struct node*) malloc (sizeof (struct node));
        struct node * temp;
        temp = (struct node*) malloc (sizeof (struct node));
```

```
        temp= head;
        for (int i=1; i<pos-1 ; i++)
            temp = temp-> next ;
        inst -> data = ele ;
        inst -> next = temp-> next;
        temp-> next = inst;
        length++;
    }
}
void display ()
{
    struct node * temp;
    temp=(struct node*)malloc (sizeof ( struct node ));
    temp= head;
    if (temp == NULL)
        printf ("\n List is empty \n");
    else {
        printf ("\n The contents of the list are: \n");
    while ( temp != NULL)
    {
        printf ("%d \n", temp-> data );
        temp = temp-> data;
    }
}
}
int main ()
{
    int choice , ele, pos;
```

```
char ch;
do {
    printf("\n 1: Insert at end \n 2: Insert at front \n
            3: Insert at random position \n 4: Display \n
            5: Exit \n");
    printf("Enter your choice \n");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1: printf("Enter element to be inserted \n");
                scanf("%d", &ele);
                insertend(ele);
                break;
        case 2: printf("Enter element to be inserted \n");
                scanf("%d", &ele);
                insertfront(ele);
                break;
        case 3: printf("Enter element to be inserted \n");
                scanf("%d", &ele);
                printf("Enter the position \n");
                scanf("%d", &pos);
                insertrandom(ele, pos);
                break;
        case 4: display();
                break;
    }
} while (choice != 5);
return 0;
}
```

(Program 9)

11-12-2020 Doubly Linked List

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{
        int data;
        struct Node *prev;
        struct Node *next;
} node;
int length (node *head) {
        int count=0;
        while (head!=NULL)
        {
            head = head -> next;
            count++;
        }return count;
}
void insertAtEnd (node **head, int d) {
        node *n, *temp = *head;
        if(*head == NULL){
            *head = (node*)malloc (sizeof (node));
            (*head)-> prev = NULL;
            (*head) -> data = d;
            (*head) -> next = NULL;
        } else {
            while (temp->next != NULL)
                temp = temp -> next;
            n = (node*) malloc (sizeof(node));
            n -> data = d;
```

```
        n->next = NULL;
        n->prev = temp;
        temp->next = n;
    }
    printf("%d was inserted in the list\n", d);
}
void insertleft(node **h, int d, int ele){
    node *head = *h;
    if(head->data == ele){
        node *temp1 = NULL;
        temp1 = (node*) malloc(sizeof(node));
        temp1->prev = NULL;
        temp1->data = d;
        temp1->next = *h;
        (*h)->prev = temp1;
        *h = temp1;
        printf("%d was inserted at the start\n", d);
        return;
    }
    node *temp;
    while(head != NULL){
        if(head->data == ele){
            head = head->prev;
            temp = (node*)malloc(sizeof(node));
            temp->data = d;
            temp->prev = head;
            temp->next = head->next;
            temp->next->prev = temp;
```

```
            head -> next = temp;
            printf("%d was inserted to the left of %d \n", d, ele);
            break;
        }
        else{
            head = head -> next;
        }
        printf("Given element is not present in the list \n");
    }
    void deleteNode(node **head, int d){
        node *temp = *head;
        if(*head == NULL){
            printf("No elements in the list to delete \n");
            return;
        }
        while(temp != NULL){
            if(temp -> data == d){
                if(temp == *head){
                    *head = (*head) -> next
                    (*head) -> prev = NULL;
                }
                else if(temp -> next == NULL){
                    temp -> prev -> next = NULL;
                    free(temp);
                }
                else{
                    temp -> prev -> next = temp -> next;
                    temp -> next -> prev = temp -> prev;
```

```
            free(temp);
        }
        printf("%d was deleted\n", d);
        return;
    }
    temp = temp->next;
}
printf("%d is not present in the list\n", d);
}
void display(node *head){
    if(head == NULL){
        printf("Empty list\n");
        return;
    }
    while(head != NULL){
        printf("<-%d->", head->data);
        head = head->next;
    }
    printf("0\n");
}
int main(){
    node *head = NULL;
    int data, pos, opt;
    printf("Insert few elements in the list (Press -1 to
                stop):\n");
    scanf("%d", &data);
    while(data != -1){
        insertAtEnd(&head, data);
    }
```

```
do{
    printf("Operations of Doubly linked list \n");
    printf("1:Insert at left \n2: Delete specified node
        \n 3: Display \n4: Insert at end \n
        5: Exit \n");
    printf("Enter your choice \n");
    scanf("%d", &opt);
    while(opt != 5) {
    switch(opt) {
    case1: printf("Enter element to be inserted \n");
        scanf("%d", &data);
        printf("Enter the node: ");
        scanf("%d", &pos)
        insertLeft(&head, data, pos);
        break;
    case2: printf("Enter element to be deleted \n");
        scanf("%d", &data);
        deleteNode(&head, data);
        break;
    case 3: display(head);
        break;
    case 4: printf("Enter data to be inserted at end \n");
        scanf("%d", &data);
        insertAtEnd(&head, data);
    }
    printf(while(opt != 5);
}
```

18-12-2020

Program 10 : Binary Search Tree

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
typedef struct binary_node {
        int data
        struct binary_node * left;
        struct binary_node * right;
} node;
void insert (node ** root, int d ) {
        if (*root == NULL) {
          (*root) = (node*) malloc (sizeof (node));
          (*root) -> left = NULL
          (*root) -> data = d;
          (*root) -> right = NULL;
        }
        else {
            if ( d < ((*root) -> data)) {
                insert (&((*root) -> left), d);
            else
                insert (&((*root) -> right), d);
            }
}
void inorder (node *root) {
        if (root == NULL)
            return;
        inorder (root -> left);
        printf ("%d", root -> data);
        inorder (root -> right);
```

```
}
void preorder (node *root) {
    if (root == NULL)
        return;
    printf ("%d", root -> data);
    preorder (root -> left);
    preorder (root -> right);
}
void postorder (node *root) {
    if (root == NULL)
        return;
    postorder (root -> left);
    postorder (root -> right);
    printf ("%d", root -> data);
}
bool search (node *root, int key) {
    if (root == NULL)
        return false;
    if (root -> data == key) {
        return true;
    else if (key < root -> data)
        return search (root -> left, key);
    else
        return search (root -> right, key);
}
int main() {
    node *root = NULL;
    int choice;
```

int d;

do {

printf("1: Insert in BST \n 2: Pre Order \n 3: In Order \n 4: PostOrder \n 5: Search \n 6: Exit \n");

printf("Your Choice: \n");

scanf("%d", &choice);

while (choice != 6) {

switch (choice) {

case 1: printf("Enter element to be inserted \n");

scanf("%d", &d);

insert(&root, d);

printf("%d inserted in tree \n", d);

break;

case 2: printf("Preorder traversal is : \n");

preorder(root);

printf("\n");

break;

case 3: printf("In order traversal is : \n");

inorder(root);

printf("\n");

break;

case 4: printf("Postorder traversal is : \n");

postorder(root);

printf("\n");

break;

case 5: printf("Enter element to be searched \n");

scanf("%d", &d);

if

```
if(search(root, d))
        printf("Element found!\n");
else
        printf("Element not found!\n");
}while(choice != 6);
}
return 0;
}
```