

DHARAMSINH DESAI UNIVERSITY , NADIAD.

Faculty of management and information

**DATA**

**STRUCTRE**

**USING ‘C’**

**ASSIGNMENT**

SUBMITTED BY : KARNIKA SHUKLA.

SEMESTER : MCA-II

ROLL NO.: MA021 (18MAPOG025)

SUBMITTED TO: Prof. HIMANSHU PUROHIT.

1

**1. Create function called swap ( ), which swaps the number values. Create a function pointer which points to a swap () function and call function using pointer. Write a program which also checks whether the two number entered by user is palindrome or not after swaping.**

**>>**

**#include<stdio.h>**

**#include<string.h>**

**int number[2];**

**void swap (int \*num1, int \*num2)**

**{**

**int temp;**

**temp=\*num1;**

**\*num1=\*num2;**

**\*num2=temp;**

**}**

**void palindrome (int num1)**

**{**

**int n1=num1;**

**int d=0;**

**printf ("value of num1 : %d\n", num1);**

**while (num1 != 0)**

**{**

**d = d \* 10;**

**d = d + num1%10;**

**num1= num1/10;**

**}**

**if (n1 == d)**

**{**

**printf (" %d is Palindrome \n", n1);**

**}**

**else**

**{**

**printf (" %d is not palindrome \n", n1);**

**}**

**}**

**void read()**

**{**

**FILE \*fptr;**

**int n;**

**fptr = fopen("e:\\swap.txt","r");**

**if(fptr == NULL)**

**{**

**printf("error\n");**

**}**

**int i=0;**

**while (fscanf (fptr, "%d", &n)!= EOF)**

**{**

**number[i]=n;**

**i++;**

**}**

**fclose(fptr);**

**}**

**int main()**

**{**

**int num1, num2;**

**read();**

**num1=number [0];**

**num2=number [1];**

**void (\*p) (int, int) =&swap;**

**(\*p) (&num1, &num2);**

**printf("\n");**

**printf ("Check This Two numbers are palindrome or not:\n");**

**printf ("Number 1: \n");**

**palindrome(num1);**

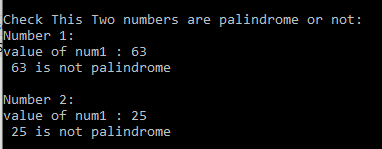
**printf("\n");**

**printf ("Number 2: \n");**

**palindrome(num2);**

**}**

**OUTPUT:**

****

**2. Implement linked list to create and manage a set of**

**elements. Set of elements contains integer values i.e. S =**

**{4,5,6}. Also implement a method which shows all possible subsets of the created set by user i.e. {{4}, {5}, {6}, {4,5}, {4,6},{5,6}, {4,5,6}, {Ø}}.**

**OUTPUT:**

**3. Write a program to check the balance of parenthesis if an expression. Implement required data structure for the same.**

**>>**

**#include <stdio.h>**

**#include <string.h>**

**#define MAX\_SIZE 100**

**int top=-1;**

**char arr[MAX\_SIZE];**

**int isEmpty()**

**{**

**if(top == -1)**

**return 1;**

**else**

**return 0;**

**}**

**int isFull(){**

**if(top == MAX\_SIZE-1)**

**return 1;**

**else**

**return 0;**

**}**

**void push(char item){**

**if(isFull())**

**printf("\nStack is full.");**

**else**

**{**

**top++;**

**arr[top] = item;**

**}**

**}**

**void pop(){**

**if(isEmpty()){**

**printf("\NEmpty Stack.");**

**}else{**

**top--;**

**}**

**}**

**char gettop(){**

**return arr[top];**

**}**

**int ArePair(char opening,char closing)**

**{**

**if(opening == '(' && closing == ')') return 1;**

**else if(opening == '{' && closing == '}') return 1;**

**else if(opening == '[' && closing == ']') return 1;**

**return 0;**

**}**

**//read data from file**

**void read\_from\_file()**

**{**

**FILE \*in\_file;**

**char in\_expr;**

**in\_file = fopen("parentheses.txt","r");**

**if(in\_file == NULL)**

**{**

**printf("error\n");**

**}**

**// printf("\nGiven Code Below to find Any mis Paranthesis \n\n");**

**while(fscanf(in\_file,"%c",&in\_expr) != EOF)**

**{**

**printf("%c",in\_expr);**

**if(in\_expr == '(' || in\_expr == '{' || in\_expr == '[')**

**{**

**push(in\_expr);**

**}**

**else if(in\_expr == ')' || in\_expr == '}' || in\_expr == ']')**

**{**

**char a = gettop();**

**if(isEmpty() || !ArePair(gettop(),in\_expr))**

**{**

**printf("\nResult : Invalid expression - Not a Balanced one !");**

**return 0;**

**}**

**else**

**{**

**pop();**

**}**

**}**

**}**

**fclose(in\_file);**

**}**

**void main()**

**{**

**read\_from\_file ();**

**if(isEmpty()){**

**printf ("\n\nResult : Valid expression - Perfectly Balanced!");**

**}else{**

**printf ("\n\nResult : Invalid expression - Not a Balanced one!");**

**}**

**printf("\n");**

**}**

**OUTPUT:**

**ss3.PNG**

**4. Implement a program to generate a linked list. For an unsorted linked list, write a method that will delete any duplicates from the linked list without using a temporary buffer.**

**>>**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node{**

**int data;**

**struct node\*next;**

**}\*head=NULL;**

**void insert(int n)**

**{**

**struct node \*temp,\*newnode;**

**newnode=(struct node\*)malloc(sizeof(struct node));**

**newnode->data=n;**

**if(head == NULL)**

**{**

**head=newnode;**

**newnode->next=NULL;**

**}**

**else**

**{**

**temp=head;**

**while(temp->next != NULL)**

**temp=temp->next;**

**temp->next=newnode;**

**newnode->next=NULL;**

**}**

**}**

**//read data from file**

**void read\_from\_file()**

**{**

**FILE \*in\_file;**

**int n;**

**in\_file = fopen("Link\_list.txt","r");**

**if(in\_file == NULL)**

**{**

**printf("error\n");**

**}**

**while(fscanf(in\_file,"%d",&n) != EOF)**

**{**

**// fscanf(in\_file,"%d",&n1);**

**insert(n);**

**}**

**fclose(in\_file);**

**}**

**void write\_into\_file()**

**{**

**FILE \*out\_file;**

**out\_file = fopen("Link\_list.txt","w");**

**if(out\_file == NULL)**

**printf("error\n");**

**struct node \* temp;**

**temp=head;**

**while(temp != NULL)**

**{**

**fprintf(out\_file,"%d\n",temp->data);**

**temp=temp->next;**

**}**

**fclose(out\_file);**

**}**

**void find\_duplicate()**

**{**

**struct node \*temp ,\*temp1 ,\*dup;**

**temp=head;**

**while(temp != NULL)**

**{**

**temp1=temp;**

**while(temp1->next != NULL)**

**{**

**if(temp->data == temp1->next->data)**

**{**

**dup=temp1->next;**

**temp1->next=temp1->next->next;**

**free(dup);**

**}**

**else**

**{**

**temp1=temp1->next;**

**}**

**}**

**temp=temp->next;**

**}**

**}**

**void display()**

**{**

**struct node \*temp;**

**if(head == NULL)**

**{**

**printf("List is Empty \n");**

**return;**

**}**

**temp=head;**

**while(temp != NULL)**

**{**

**printf("| %d | -> ",temp->data);**

**temp=temp->next;**

**}**

**}**

**void main()**

**{**

**read\_from\_file();**

**printf("Link List : \n");**

**display();**

**find\_duplicate();**

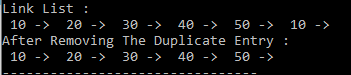
**write\_into\_file();**

**printf("\nAfter Removing The Duplicate Entry :\n");**

**display();**

**}**

**OUTPUT:**

****

**5. Write a program to create a binary tree. Implemen required method to generate a binary tree from user inputs and to display binary tree using level order and pre order traversals**.

**>>**

**#include<stdio.h>S**

**#include<stdlib.h>**

**struct node**

**{**

**int data;**

**struct node \*left;**

**struct node \*right;**

**};**

**struct node\* root;**

**struct node\* insert(struct node\* r, int data)**

**{**

**if(r==NULL)**

**{**

**r = (struct node\*) malloc(sizeof(struct node));**

**r->data = data;**

**r->left = NULL;**

**r->right = NULL;**

**return r;**

**}**

**else if(data < r->data){**

**r->left = insert(r->left, data);**

**}**

**else {**

**r->right = insert(r->right, data);**

**}**

**return r;**

**}**

**void Preorder(struct node \*root)**

**{**

**if(root != NULL)**

**{**

**printf("[ %d ] -> ",root->data);**

**Preorder(root->left);**

**Preorder(root->right);**

**}**

**}**

**//read data from file**

**void read\_from\_file()**

**{**

**FILE \*in\_file;**

**int n;**

**in\_file = fopen("e:\\BSTtree.txt","r");**

**if(in\_file == NULL)**

**{**

**printf("error101\n");**

**}**

**while(fscanf(in\_file,"%d",&n) != EOF)**

**{**

**root=insert(root,n);**

**}**

**fclose(in\_file);**

**}**

**int queue[100];**

**int front=0;**

**int rear=-1;**

**void enQueue(struct node \*new\_node)**

**{**

**queue[rear++] = new\_node;**

**}**

**struct node \*deQueue()**

**{**

**if(front == rear)**

**{**

**return NULL;**

**}**

**else**

**{**

**return queue[front++];**

**}**

**}**

**void printLevelOrder(struct node\* root)**

**{**

**struct node \*temp\_node = root;**

**enQueue(root);**

**while (temp\_node != NULL)**

**{**

**printf("[ %d ] - > ", temp\_node->data);**

**if (temp\_node->left != NULL)**

**{**

**enQueue(temp\_node->left);**

**}**

**if (temp\_node->right != NULL)**

**{**

**enQueue(temp\_node->right);**

**}**

**temp\_node = deQueue();**

**}**

**}**

**int main()**

**{**

**read\_from\_file();**

**printf("\n PreOrder Binary Tree : \n");**

**Preorder(root);**

**printf("\n\n");**

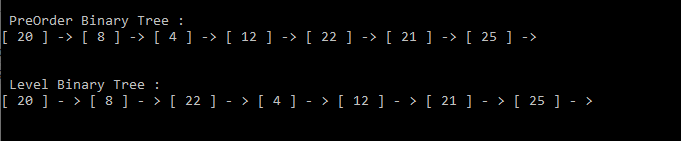
**printf("\n Level Binary Tree : \n");**

**printLevelOrder(root);**

**printf("\n\n");**

**}**

**OUTPUT:**

****

**6. Given two values v1 and v2 (where v1 < v2) within a Binary Search Tree. Print all the keys of tree in range v1 to v2. i.e. print all x such that v1<=x<=v2 and x is a element of given BST. (Create a Binary Search Tree by any method).**

**>>**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node**

**{**

**int data;**

**struct node \*left;**

**struct node \*right;**

**};**

**struct node\* root;**

**struct node\* insert(struct node\* r, int data)**

**{**

**if(r==NULL)**

**{**

**r = (struct node\*) malloc(sizeof(struct node));**

**r->data = data;**

**r->left = NULL;**

**r->right = NULL;**

**return r;**

**}**

**else if(data < r->data){**

**r->left = insert(r->left, data);**

**}**

**else {**

**r->right = insert(r->right, data);**

**}**

**return r;**

**}**

**void Print(struct node \*root, int k1, int k2)**

**{**

**if ( NULL == root )**

**return;**

**if ( k1 < root->data )**

**Print(root->left, k1, k2);**

**if ( k1 <= root->data && k2 >= root->data )**

**printf("%d ", root->data );**

**if ( k2 > root->data )**

**Print(root->right, k1, k2);**

**}**

**//read data from file**

**void read\_from\_file()**

**{**

**FILE \*in\_file;**

**int n;**

**in\_file = fopen("e:\\BSTtree.txt","r");**

**if(in\_file == NULL)**

**{**

**printf("error101\n");**

**}**

**while(fscanf(in\_file,"%d",&n) != EOF)**

**{**

**root=insert(root,n);**

**}**

**fclose(in\_file);**

**}**

**void Display(struct node\* root)**

**{**

**if(root != NULL)**

**{**

**printf("%d \t",root->data);**

**Display(root->left);**

**Display(root->right);**

**}**

**}**

**int main()**

**{**

**int k1,k2;**

**read\_from\_file();**

**printf("\nBinary Tree : \n");**

**Display(root);**

**printf("\n\nEnter First Number : ");**

**scanf("%d",&k1);**

**printf("\nEnter second Number : ");**

**scanf("%d",&k2);**

**printf("\n");**

**printf("Possible Keys Range Between %d and %d \n",k1,k2);**

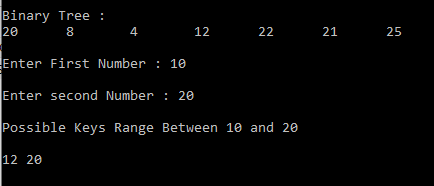
**printf("\n");**

**Print(root,k1,k2);**

**printf("\n\n");**

**}**

**OUTPUT:**

****

**7. Write a program to create a binary tree. Implement**

**required method to generate a binary tree from user inputs and check whether the Binary Tree is a perfect binary tree**.

**>>**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct tnode{**

**int data;**

**struct tnode \*lchiled;**

**struct tnode \*rchiled;**

**};**

**struct tnode \*Create(struct tnode \*p,int value)**

**{**

**struct tnode \*temp,\*temp1;**

**if(p == NULL)**

**{**

**p=(struct tnode\*)malloc(sizeof(struct tnode));**

**if(p == NULL)**

**{**

**printf("Error : Allocating Memory \n");**

**exit(0);**

**}**

**else**

**{**

**p->data = value;**

**p->rchiled=NULL;**

**p->lchiled=NULL;**

**}**

**}**

**//if Root Node Exit**

**else**

**{**

**temp=p;**

**while(temp != NULL)**

**{**

**temp1=temp;**

**if(temp1->data > value)**

**{**

**temp=temp->lchiled;**

**}**

**else**

**{**

**temp=temp->rchiled;**

**}**

**}**

**if(temp1->data > value)**

**{**

**temp1->lchiled=(struct tnode\*)malloc(sizeof(struct tnode));**

**temp1=temp1->lchiled;**

**if(temp1 == NULL)**

**{**

**printf("Error : Allocating Memory \n");**

**exit(0);**

**}**

**else**

**{**

**temp1->data=value;**

**temp1->rchiled=temp1->lchiled=NULL;**

**}**

**}**

**else**

**{**

**temp1->rchiled=(struct tnode\*)malloc(sizeof(struct tnode));**

**temp1=temp1->rchiled;**

**if(temp1 == NULL)**

**{**

**printf("Error : Allocating Memory \n");**

**exit(0);**

**}**

**else**

**{**

**temp1->data=value;**

**temp1->rchiled=temp1->lchiled=NULL;**

**}**

**}**

**}**

**return(p);**

**}**

**int findADepth(struct tnode \*node)**

**{**

**int d = 0;**

**while (node != NULL)**

**{**

**d++;**

**node = node->lchiled;**

**}**

**return d;**

**}**

**int isPerfectRec(struct tnode\* root, int d, int level)**

**{**

**if (root == NULL)**

**return 1;**

**if (root->lchiled == NULL && root->rchiled == NULL)**

**return (d == level+1);**

**if (root->lchiled == NULL || root->rchiled == NULL)**

**return 0;**

**return isPerfectRec(root->lchiled, d, level+1) &&**

**isPerfectRec(root->rchiled, d, level+1);**

**}**

**int isPerfect(struct tnode \*root)**

**{**

**int d = findADepth(root);**

**return isPerfectRec(root,d,0);**

**}**

**struct tnode \*root;**

**//read data from file**

**void read\_from\_file()**

**{**

**FILE \*in\_file;**

**int n;**

**in\_file = fopen("e:\\BSTtree.txt","r");**

**if(in\_file == NULL)**

**{**

**printf("error\n");**

**}**

**printf("Given Binary Tree : \n\n");**

**while(fscanf(in\_file,"%d",&n) != EOF)**

**{**

**printf("%d\t",n);**

**root=Create(root,n);**

**}**

**fclose(in\_file);**

**}**

**void main()**

**{**

**read\_from\_file();**

**printf("\n\n");**

**if (isPerfect(root))**

**printf("This Binary Tree is Perfect : \n");**

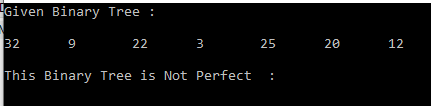
**else**

**printf("This Binary Tree is Not Perfect : \n");**

**printf("\n");**

**}**

**OUTPUT:**

****

**8. Write a program to implement stack with all basic operations using linked list.**

**>>**

**// C program to Implement a stack**

**//using singly linked list**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node\* head;**

**}\*top=NULL;**

**int write()**

**{**

**int num;**

**FILE \*fptr;**

**fptr = fopen("e:\\linknum.txt","w");**

**if(fptr == NULL)**

**{**

**printf("Error!");**

**return;**

**}**

**struct Node \*temp;**

**temp = top;**

**while (temp != NULL)**

**{**

**fprintf(fptr,"%d->", temp->data);**

**temp = temp->head;**

**}**

**fclose(fptr);**

**}**

**int read()**

**{**

**int num;**

**FILE \*fptr;**

**if ((fptr = fopen("e:\\linknum.txt","r")) == NULL){**

**printf("Error! opening file");**

**exit(1);**

**}**

**while(fscanf(fptr,"%d",&num) == 1)**

**push(num);**

**fclose(fptr);**

**}**

**void push(int data)**

**{**

**struct Node\* temp;**

**temp = (struct Node\*)malloc(sizeof(struct Node));**

**if (!temp) {**

**printf("\nStack Overflow");**

**exit(1);**

**}**

**temp->data = data;**

**temp->head = top;**

**top = temp;**

**}**

**int isEmpty()**

**{**

**return top == NULL;**

**}**

**void pop()**

**{**

**struct Node\* temp;**

**if (top == NULL) {**

**printf("\nStack Underflow");**

**exit(1);**

**}**

**else {**

**temp = top;**

**top = top->head;**

**temp->head = NULL;**

**free(temp);**

**}**

**}**

**void display()**

**{**

**struct Node\* temp;**

**if (top == NULL) {**

**printf("\nStack Underflow");**

**exit(1);**

**}**

**else {**

**temp = top;**

**while (temp != NULL) {**

**printf("%d->", temp->data);**

**temp = temp->head;**

**}**

**}**

**}**

**int main()**

**{**

**read();**

**printf("\nData in file : \n");**

**display();**

**int ch,data;**

**printf("\n\t\*\*\*\*MENU\*\*\*\*\n");**

**printf("\t 1. PUSH\n");**

**printf("\t 2. POP\n");**

**printf("\t 3. DISPLAY\n");**

**printf("\t 4. EXIT\n");**

**do{**

**printf("\n\t Enter Your Choice : ");**

**scanf("\t %d",&ch);**

**switch(ch)**

**{**

**case 1:**

**{**

**printf("\t Enter Data : ");**

**scanf("%d",&data);**

**push(data);**

**write();**

**break;**

**}**

**case 2:**

**pop();**

**write();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**break;**

**default :**

**printf("\t Enter Proper Choice \n");**

**break;**

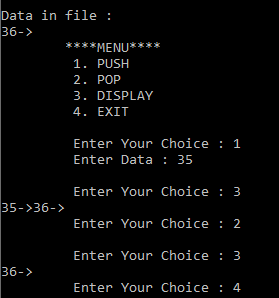
**}**

**}while(ch!=4);**

**return 0;**

**}**

**OUTPUT:**

****

**9. Write a program to implement Queue with all basic operations using linked list.**

**>>**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node{**

**int data;**

**struct node \*next;**

**}\*head=NULL;**

**int write()**

**{**

**int num;**

**FILE \*fptr;**

**fptr = fopen("e:\\linknum.txt","w");**

**if(fptr == NULL)**

**{**

**printf("Error!");**

**return;**

**}**

**struct node \*temp;**

**while(temp != NULL)**

**{**

**fprintf(fptr,"| %d | -> ",temp->data);**

**temp=temp->next;**

**}**

**printf(" |\n");**

**printf("\n");**

**fclose(fptr);**

**}**

**int read()**

**{**

**int num;**

**FILE \*fptr;**

**if ((fptr = fopen("e:\\linknum.txt","r")) == NULL){**

**printf("Error! opening file");**

**exit(1);**

**}**

**while(fscanf(fptr,"%d",&num) == 1)**

**insert(num);**

**fclose(fptr);**

**}**

**void insert(int item)**

**{**

**struct node \*newnode;**

**newnode=(struct node\*)malloc(sizeof(struct node));**

**newnode->data=item;**

**if(head == NULL)**

**{**

**head=newnode;**

**newnode->next=NULL;**

**}**

**else**

**{**

**newnode->next=head;**

**head=newnode;**

**}**

**}**

**void Delete()**

**{**

**struct node \*temp;**

**if(head == NULL)**

**{**

**printf("\tQueue is Empty \n");**

**}**

**temp=head;**

**while(temp->next->next != NULL)**

**{**

**temp=temp->next;**

**}**

**temp->next=NULL;**

**}**

**void display()**

**{**

**struct node \*temp;**

**temp=head;**

**if(head==NULL)**

**{**

**printf("\tQueue is Empty \n");**

**}**

**printf("\t");**

**while(temp != NULL)**

**{**

**printf("| %d | -> ",temp->data);**

**temp=temp->next;**

**}**

**printf(" |\n");**

**printf("\n");**

**}**

**void main()**

**{**

**read();**

**printf("\nData in file : \n");**

**display();**

**int ch;**

**int item;**

**struct node \*newnode;**

**newnode=(struct node\*)malloc(sizeof(struct node));**

**printf(“\n\t\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*”);**

**printf("\t1.INSERT\n");**

**printf("\t2.DELETE\n");**

**printf("\t3.DISPLAY\n");**

**printf("\t4.EXIT\n");**

**do{**

**printf("\n\tEnter Your Choice : ");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**printf("\tEnter Data : ");**

**scanf("%d",&item);**

**insert(item);**

**write();**

**break;**

**case 2:**

**Delete();**

**write();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**break;**

**default:**

**printf("\tEnter proper choice :\n");**

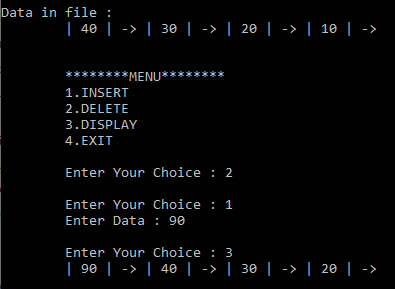
**break;**

**}**

**}while(ch!=4);**

**}**

**OUTPUT:**

****

**10. Write a program to implement stack with required operations using array.**

**>>**

**#include<stdio.h>**

**#include<stdlib.h>**

**#define SIZE 100**

**int stack[SIZE];**

**int top=-1;**

**void push(int);**

**int pop();**

**void display();**

**int write()**

**{**

**int num;**

**FILE \*fptr;**

**fptr = fopen("e:\\numbers.txt","w");**

**if(fptr == NULL)**

**{**

**printf("Error!");**

**exit(1);**

**}**

**int i;**

**for(i=top;i>=0;i--)**

**fprintf(fptr,"%d\t",stack[i]);**

**fclose(fptr);**

**}**

**int read()**

**{**

**int num;**

**FILE \*fptr;**

**if ((fptr = fopen("e:\\numbers.txt","r")) == NULL){**

**printf("Error! opening file");**

**exit(1);**

**}**

**while(fscanf(fptr,"%d",&num) == 1)**

**push(num);**

**fclose(fptr);**

**}**

**void main()**

**{**

**read();**

**printf("\nData in file : \n");**

**display();**

**int ch,data;**

**printf("\t\*\*\*\*MENU\*\*\*\*\n");**

**printf("\t 1. PUSH\n");**

**printf("\t 2. POP\n");**

**printf("\t 3. DISPLAY\n");**

**printf("\t 4. EXIT\n");**

**do{**

**printf("\t Enter Your Choice : ");**

**scanf("\t %d",&ch);**

**switch(ch)**

**{**

**case 1:**

**printf("\t Enter Data : ");**

**scanf("%d",&data);**

**push(data);**

**write();**

**break;**

**case 2:**

**pop();**

**write();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**break;**

**default :**

**printf("\t Enter Proper Choice \n");**

**break;**

**}**

**}while(ch!=4);**

**}**

**void push(int item)**

**{**

**if(top >= SIZE-1)**

**{**

**printf("\nStack Overflow.");**

**}**

**else**

**{**

**top = top+1;**

**stack[top] = item;**

**}**

**}**

**int pop()**

**{**

**int item;**

**if(top <0)**

**{**

**printf("stack under flow:");**

**}**

**else**

**{**

**item = stack[top];**

**printf("\t %d : DELETE\n",item);**

**top = top-1;**

**return(item);**

**}**

**}**

**void display()**

**{**

**if(top==-1)**

**{**

**printf("\tSTACK IS EMPTY\n");**

**}**

**int i;**

**for(i=top;i>=0;i--)**

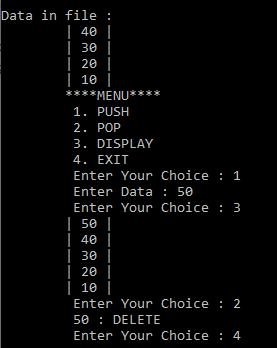
**{**

**printf("\t| %d |\n",stack[i]);**

**}**

**}**

**OUTPUT:**

****

**11. Write a program to implement Queue with required operations using array.**

**>>**

**#include<stdio.h>**

**#include<stdlib.h>**

**#define SIZE 100**

**int queue[SIZE];**

**int front=-1;**

**int rear=-1;**

**void insert(int);**

**int write()**

**{**

**int num;**

**FILE \*fptr;**

**fptr = fopen("e:\\numbers.txt","w");**

**if(fptr == NULL)**

**{**

**printf("Error!");**

**return;**

**}**

**int i;**

**for(i=front;i<=rear;i++)**

**fprintf(fptr,"| %d | ",queue[i]);**

**printf("\n");**

**fclose(fptr);**

**}**

**int read()**

**{**

**int num;**

**FILE \*fptr;**

**if ((fptr = fopen("e:\\numbers.txt","r")) == NULL){**

**printf("Error! opening file");**

**exit(1);**

**}**

**while(fscanf(fptr,"%d",&num) == 1)**

**insert(num);**

**fclose(fptr);**

**}**

**void insert(int data)**

**{**

**if(rear == -1)**

**front=0;**

**if(rear > SIZE)**

**printf("\tQueue is Overflow : \n");**

**else**

**{**

**rear++;**

**queue[rear]=data;**

**}**

**}**

**void Delete()**

**{**

**if(front > rear)**

**printf("\tQueue is Underflow : \n");**

**else**

**{**

**printf("\tDelete : %d\n",queue[front]);**

**front++;**

**}**

**}**

**void display()**

**{**

**int i;**

**if(rear == -1 || front > rear)**

**printf("\tQueue is Empty \n");**

**//printf("Rear : %d\n",rear);**

**//printf("Front : %d\n",front);**

**printf("\t");**

**for(i=front;i<=rear;i++)**

**printf("| %d | ",queue[i]);**

**printf("\n");**

**}**

**void main()**

**{**

**read();**

**printf("\nData in file : ");**

**display();**

**int ch;**

**int data;**

**printf("\t\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\n");**

**printf("\t 1. INSERT\n");**

**printf("\t 2. DELETE\n");**

**printf("\t 3. DISPLAY\n");**

**printf("\t 4. EXIT\n");**

**do{**

**printf("\tEnter Your Choice : ");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**printf("\t Enter Data : ");**

**scanf("%d",&data);**

**insert(data);**

**write();**

**break;**

**case 2:**

**Delete();**

**write();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**return;**

**default:**

**printf("\tEnter Proper Choice \n");**

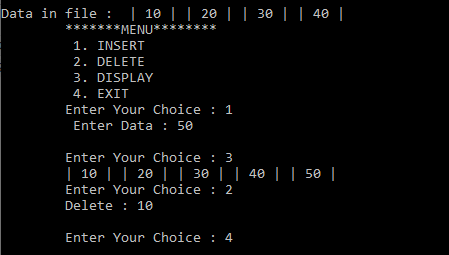
**break;**

**}**

**}while(ch !=4);**

**}**

**OUTPUT:**

****

**12. Write a program to check whether the string is palindrome or not. Use Stack Data Structure for the same.**

**>> /\* C Program to Identify whether the String is Palindrome or not using Stack \*/**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#define MAX 50**

**int top = -1, front = 0;**

**int stack[MAX];**

**void push(char);**

**void pop();**

**int write()**

**{**

**char c[25];**

**FILE \*fptr;**

**fptr = fopen("e:\\string.txt","w");**

**if(fptr == NULL)**

**{**

**printf("Error!");**

**exit(1);**

**}**

**int i;**

**while(stack[i] != '\0')**

**{**

**printf("%c",stack[i]);**

**i++;**

**}**

**fclose(fptr);**

**}**

**int read()**

**{**

**char c[25],b;**

**int i;**

**FILE \*fptr;**

**if ((fptr = fopen("e:\\string.txt","r")) == NULL){**

**printf("Error! opening file");**

**exit(1);**

**}**

**while(fscanf(fptr,"%c",&c) == 1)**

**push(c);**

**fclose(fptr);**

**}**

**void display()**

**{**

**int i=0;**

**char c[25];**

**while(stack[i] != '\0')**

**{**

**printf("%c",stack[i]);**

**i++;**

**}**

**}**

**void main()**

**{**

**read();**

**printf("\nString in file : ");**

**display();**

**int i, choice;**

**char s[MAX], b;**

**while (1)**

**{**

**printf("\n1-read string\n2-exit\n");**

**printf("enter your choice\n");**

**scanf("%d", &choice);**

**switch (choice)**

**{**

**case 1:**

**for (i = 0;s[i] != '\0';i++)**

**{**

**b = s[i];**

**push(b);**

**write();**

**}**

**for (i = 0;i < (strlen(s) / 2);i++)**

**{**

**if (stack[top] == stack[front])**

**{**

**pop();**

**front++;**

**write();**

**}**

**else**

**{**

**printf("%s is not a palindrome\n", s);**

**break;**

**}**

**}**

**if ((strlen(s) / 2) == front)**

**printf("%s is palindrome\n", s);**

**front = 0;**

**top = -1;**

**break;**

**case 2:**

**exit(0);**

**default:**

**printf("enter correct choice\n");**

**}**

**}**

**}**

**void push(char a)**

**{**

**top++;**

**stack[top] = a;**

**}**

**void pop()**

**{**

**top--;**

**}**

**OUTPUT:**

**SS12.PNG**

**13. Write a program to implement Doubly Linked List.**

**>> /\* C Program to Implement a Doubly Linked List & provide Insertion, Deletion & Display Operations \*/**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node**

**{**

**struct node \*prev;**

**int n;**

**struct node \*next;**

**}\*h,\*temp,\*temp1,\*temp2;**

**int write()**

**{**

**int num;**

**FILE \*fptr;**

**fptr = fopen("e:\\linknum.txt","w");**

**if(fptr == NULL)**

**{**

**printf("Error!");**

**return;**

**}**

**//printf("\n Linked list elements from begining : ");**

**temp=h;**

**while (temp != NULL)**

**{**

**fprintf(fptr," %d ", temp->n);**

**temp= temp->next;**

**}**

**// printf(" %d ", temp2->n);**

**fclose(fptr);**

**}**

**int read()**

**{**

**int num;**

**FILE \*fptr;**

**if ((fptr = fopen("e:\\linknum.txt","r")) == NULL){**

**printf("Error! opening file");**

**exit(1);**

**}**

**while(fscanf(fptr,"%d",&num) == 1)**

**{**

**insert1(num);**

**}**

**fclose(fptr);**

**}**

**void insert1(int);**

**void insert2(int);**

**void insert3(int);**

**void delete();**

**void display();**

**int count = 0;**

**void main()**

**{**

**//read();**

**//printf("\nData in file : \n");**

**//display();**

**int ch,n;**

**h = NULL;**

**temp = temp1 = NULL;**

**printf("\n\*\*\*\*\*\*MENU\*\*\*\*\*\*\n");**

**printf("\n 1 - Insert at beginning");**

**printf("\n 2 - Insert at end");**

**printf("\n 3 - Insert at position i");**

**printf("\n 4 - Delete at i");**

**printf("\n 5 - Display");**

**printf("\n 6 - Exit");**

**do**

**{**

**printf("\n Enter choice : ");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**printf("\n Enter value to node : ");**

**scanf("%d", &n);**

**insert1(n);**

**write();**

**break;**

**case 2:**

**printf("\n Enter value to node : ");**

**scanf("%d", &n);**

**insert2(n);**

**write();**

**break;**

**case 3:**

**printf("\n Enter value to node : ");**

**scanf("%d", &n);**

**insert3(n);**

**write();**

**break;**

**case 4:**

**delete();**

**write();**

**break;**

**case 5:**

**display();**

**write();**

**break;**

**case 6:**

**exit(0);**

**default:**

**printf("\n Wrong choice menu");**

**}**

**}while(ch!=6);**

**}**

**/\* TO create an empty node \*/**

**void create(int data)**

**{**

**temp =(struct node \*)malloc(1\*sizeof(struct node));**

**temp->prev = NULL;**

**temp->next = NULL;**

**temp->n = data;**

**count++;**

**}**

**/\* TO insert at beginning \*/**

**void insert1(int x)**

**{**

**if (h == NULL)**

**{**

**create(x);**

**h = temp;**

**temp1 = h;**

**}**

**else**

**{**

**create(x);**

**temp->next = h;**

**h->prev = temp;**

**h = temp;**

**}**

**}**

**/\* To insert at end \*/**

**void insert2(int x)**

**{**

**if (h == NULL)**

**{**

**create(x);**

**h = temp;**

**temp1 = h;**

**}**

**else**

**{**

**create(x);**

**temp1->next = temp;**

**temp->prev = temp1;**

**temp1 = temp;**

**}**

**}**

**/\* To insert at any position \*/**

**void insert3(int x)**

**{**

**int pos, i = 2;**

**printf("\n Enter position to be inserted : ");**

**scanf("%d", &pos);**

**temp2 = h;**

**if ((pos < 1) || (pos >= count + 1))**

**{**

**printf("\n Position out of range to insert");**

**return;**

**}**

**if ((h == NULL) && (pos != 1))**

**{**

**printf("\n Empty list cannot insert other than 1st position");**

**return;**

**}**

**if ((h == NULL) && (pos == 1))**

**{**

**create(x);**

**h = temp;**

**temp1 = h;**

**return;**

**}**

**else**

**{**

**while (i < pos)**

**{**

**temp2 = temp2->next;**

**i++;**

**}**

**create(x);**

**temp->prev = temp2;**

**temp->next = temp2->next;**

**temp2->next->prev = temp;**

**temp2->next = temp;**

**}**

**}**

**/\* To delete an element \*/**

**void delete()**

**{**

**int i = 1, pos;**

**printf("\n Enter position to be deleted : ");**

**scanf("%d", &pos);**

**temp2 = h;**

**if ((pos < 1) || (pos >= count + 1))**

**{**

**printf("\n Error : Position out of range to delete");**

**return;**

**}**

**if (h == NULL)**

**{**

**printf("\n Error : Empty list no elements to delete");**

**return;**

**}**

**else**

**{**

**while (i < pos)**

**{**

**temp2 = temp2->next;**

**i++;**

**}**

**if (i == 1)**

**{**

**if (temp2->next == NULL)**

**{**

**printf("Node deleted from list");**

**free(temp2);**

**temp2 = h = NULL;**

**return;**

**}**

**}**

**if (temp2->next == NULL)**

**{**

**temp2->prev->next = NULL;**

**free(temp2);**

**printf("Node deleted from list");**

**return;**

**}**

**temp2->next->prev = temp2->prev;**

**if (i != 1)**

**temp2->prev->next = temp2->next; /\* Might not need this statement if i == 1 check \*/**

**if (i == 1)**

**h = temp2->next;**

**printf("\n Node deleted");**

**free(temp2);**

**}**

**count--;**

**}**

**void display()**

**{**

**temp2 = h;**

**if (temp2 == NULL)**

**{**

**printf("List empty to display \n");**

**return;**

**}**

**printf("\n Linked list elements from begining : ");**

**while (temp2->next != NULL)**

**{**

**printf(" %d ", temp2->n);**

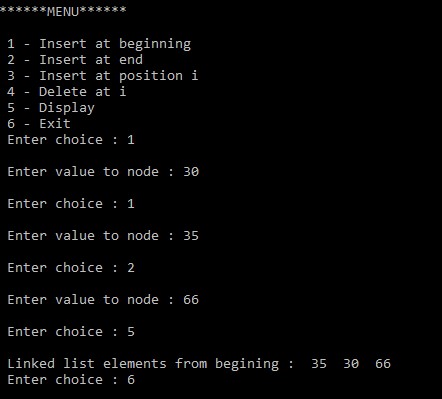
**temp2 = temp2->next;**

**}**

**printf(" %d ", temp2->n);**

**}**

**OUTPUT :**

****