**/\***

**\* C Program to Perform Binary Search using Recursion**

**\*/**

#include <stdio.h>

void binary\_search(int [], int, int, int);

void bubble\_sort(int [], int);

int main()

{

int key, size, i;

int list[25];

printf("Enter size of a list: ");

scanf("%d", &size);

printf("Generating random numbers\n");

for(i = 0; i < size; i++)

{

list[i] = rand() % 100;

printf("%d ", list[i]);

}

bubble\_sort(list, size);

printf("\n\n");

printf("Enter key to search\n");

scanf("%d", &key);

binary\_search(list, 0, size, key);

}

void bubble\_sort(int list[], int size)

{

int temp, i, j;

for (i = 0; i < size; i++)

{

for (j = i; j < size; j++)

{

if (list[i] > list[j])

{

temp = list[i];

list[i] = list[j];

list[j] = temp;

}

}

}

}

void binary\_search(int list[], int lo, int hi, int key)

{

int mid;

if (lo > hi)

{

printf("Key not found\n");

return;

}

mid = (lo + hi) / 2;

if (list[mid] == key)

{

printf("Key found\n");

}

else if (list[mid] > key)

{

binary\_search(list, lo, mid - 1, key);

}

else if (list[mid] < key)

{

binary\_search(list, mid + 1, hi, key);

}

}

**/\***

**\* C Program to Implement BogoSort in an Integer Array**

**\*/**

#include <stdio.h>

#include <stdlib.h>

#define size 7

/\* Function Prototypes \*/

int is\_sorted(int \*, int);

void shuffle(int \*, int);

void bogosort(int \*, int);

int main()

{

int numbers[size];

int i;

printf("Enter the elements of array:");

for (i = 0; i < size;i++)

{

scanf("%d", &numbers[i]);

}

bogosort(numbers, size);

printf("The array after sorting is:");

for (i = 0;i < size;i++)

{

printf("%d\n", numbers[i]);

}

printf("\n");

}

/\* Code to check if the array is sorted or not \*/

int is\_sorted(int \*a, int n)

{

while (--n >= 1)

{

if (a[n] < a[n - 1])

{

return 0;

}

}

return 1;

}

/\* Code to shuffle the array elements \*/

void shuffle(int \*a, int n)

{

int i, t, temp;

for (i = 0;i < n;i++)

{

t = a[i];

temp = rand() % n; /\* Shuffles the given array using Random function \*/

a[i] = a[temp];

a[temp] = t;

}

}

/\* Code to check if the array is sorted or not and if not sorted calls the shuffle function to shuffle the array elements \*/

void bogosort(int \*a, int n)

{

while (!is\_sorted(a, n))

{

shuffle(a, n);

}

}

**/\***

**\* C Program to Implement a Heap & provide Insertion & Deletion Operation**

**\*/**

#include <stdio.h>

#include <stdlib.h>

int array[100], n;

main()

{

int choice, num;

n = 0;/\*Represents number of nodes in the heap\*/

while(1)

{

printf("1.Insert the element \n");

printf("2.Delete the element \n");

printf("3.Display all elements \n");

printf("4.Quit \n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter the element to be inserted to the list : ");

scanf("%d", &num);

insert(num, n);

n = n + 1;

break;

case 2:

printf("Enter the elements to be deleted from the list: ");

scanf("%d", &num);

delete(num);

break;

case 3:

display();

break;

case 4:

exit(1);

default:

printf("Invalid choice \n");

}/\*End of switch \*/

}/\*End of while \*/

}/\*End of main()\*/

display()

{

int i;

if (n == 0)

{

printf("Heap is empty \n");

return;

}

for (i = 0; i < n; i++)

printf("%d ", array[i]);

printf("\n");

}/\*End of display()\*/

insert(int num, int location)

{

int parentnode;

while (location > 0)

{

parentnode =(location - 1)/2;

if (num <= array[parentnode])

{

array[location] = num;

return;

}

array[location] = array[parentnode];

location = parentnode;

}/\*End of while\*/

array[0] = num; /\*assign number to the root node \*/

}/\*End of insert()\*/

delete(int num)

{

int left, right, i, temp, parentnode;

for (i = 0; i < num; i++) {

if (num == array[i])

break;

}

if (num != array[i])

{

printf("%d not found in heap list\n", num);

return;

}

array[i] = array[n - 1];

n = n - 1;

parentnode =(i - 1) / 2; /\*find parentnode of node i \*/

if (array[i] > array[parentnode])

{

insert(array[i], i);

return;

}

left = 2 \* i + 1; /\*left child of i\*/

right = 2 \* i + 2; /\* right child of i\*/

while (right < n)

{

if (array[i] >= array[left] && array[i] >= array[right])

return;

if (array[right] <= array[left])

{

temp = array[i];

array[i] = array[left];

array[left] = temp;

i = left;

}

else

{

temp = array[i];

array[i] = array[right];

array[right] = temp;

i = right;

}

left = 2 \* i + 1;

right = 2 \* i + 2;

}/\*End of while\*/

if (left == n - 1 && array[i]) {

temp = array[i];

array[i] = array[left];

array[left] = temp;

}

}

**/\***

**\* C Program to Implement Circular Linked List using Struct Pointers**

**\*/**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct circular

{

int i;

struct circular \*next;

};

struct circular \*temp;

struct circular \*head;

struct circular \*p;

struct circular \*mid;

struct circular \*move;

int cnt=0;

void create(void);

void insert(void);

void display(void);

void del(void);

void main()

{

int ch=0;

clrscr();

while(ch!=5)

{

printf("\n1.CREATE");

printf("\n2.INSERT");

printf("\n3.DELETE");

printf("\n4.DISPLAY");

printf("\n5.EXIT");

scanf("%d",&ch);

if(ch==1)

{

create();

cnt++;

cnt++;

}

if(ch==2)

{

insert();

cnt++;

}

if(ch==3)

{

del();

cnt--;

}

if(ch==4)

{

display();

}

if(ch==5)

{

break;

}

}

getch();

}

void create()

{

head=(struct circular \*)malloc(sizeof(struct circular));

head->next=head;

printf("ENETER THE DATA");

scanf("%d",&head->i);

temp=head;

temp->next=(struct circular \*)malloc(sizeof(struct circular));

temp=temp->next;

temp->next=head;

printf("ENETER THE DATA");

scanf("%d",&temp->i);

}

void insert()

{

int add,t;

printf("\n\t ENTER ANY NUMBER BETWEEN 1 AND %d",cnt);

scanf("%d",&add);

p=head;

t=1;

while(t<add)

{

p=p->next;

t++;

}

printf("%d",p->i);

clrscr();

mid=(struct circular \*)malloc(sizeof(struct circular));

printf("ENETER THE DATA");

scanf("%d",&mid->i);

mid->next=p->next;

p->next=mid;

}

void display()

{

p=head;

printf("%d-->",p->i);

p=p->next;

while(p!=head)

{

printf("%d-->",p->i);

p=p->next;

}

}

void del(void)

{

int add,t;

printf("\n\t ENTER ANY NUMBER BETWEEN 1 AND %d",cnt);

scanf("%d",&add);

p=head;

t=1;

while(t<add-1)

{

p=p->next;

t++;

}

printf("%d",p->i);

clrscr();

mid=p->next;

p->next=mid->next;

}

**/\***

**\* C Program to Implement Circular Linked List normal**

**\*/**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

struct node {

int data;

int key;

struct node \*next;

};

struct node \*head = NULL;

struct node \*current = NULL;

bool isEmpty() {

return head == NULL;

}

int length() {

int length = 0;

//if list is empty

if(head == NULL) {

return 0;

}

current = head->next;

while(current != head) {

length++;

current = current->next;

}

return length;

}

//insert link at the first location

void insertFirst(int key, int data) {

//create a link

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

link->key = key;

link->data = data;

if (isEmpty()) {

head = link;

head->next = head;

} else {

//point it to old first node

link->next = head;

//point first to new first node

head = link;

}

}

//delete first item

struct node \* deleteFirst() {

//save reference to first link

struct node \*tempLink = head;

if(head->next == head) {

head = NULL;

return tempLink;

}

//mark next to first link as first

head = head->next;

//return the deleted link

return tempLink;

}

//display the list

void printList() {

struct node \*ptr = head;

printf("\n[ ");

//start from the beginning

if(head != NULL) {

while(ptr->next != ptr) {

printf("(%d,%d) ",ptr->key,ptr->data);

ptr = ptr->next;

}

}

printf(" ]");

}

main() {

insertFirst(1,10);

insertFirst(2,20);

insertFirst(3,30);

insertFirst(4,1);

insertFirst(5,40);

insertFirst(6,56);

printf("Original List: ");

//print list

printList();

while(!isEmpty()) {

struct node \*temp = deleteFirst();

printf("\nDeleted value:");

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

}

printf("\nList after deleting all items: ");

printList();

}

**/\***

**\* 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,\*temp4;

void insert1();

void insert2();

void insert3();

void traversebeg();

void traverseend(int);

void sort();

void search();

void update();

void delete();

int count = 0;

void main()

{

int ch;

h = NULL;

temp = temp1 = NULL;

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 from beginning");

printf("\n 6 - Display from end");

printf("\n 7 - Search for element");

printf("\n 8 - Sort the list");

printf("\n 9 - Update an element");

printf("\n 10 - Exit");

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert1();

break;

case 2:

insert2();

break;

case 3:

insert3();

break;

case 4:

delete();

break;

case 5:

traversebeg();

break;

case 6:

temp2 = h;

if (temp2 == NULL)

printf("\n Error : List empty to display ");

else

{

printf("\n Reverse order of linked list is : ");

traverseend(temp2->n);

}

break;

case 7:

search();

break;

case 8:

sort();

break;

case 9:

update();

break;

case 10:

exit(0);

default:

printf("\n Wrong choice menu");

}

}

}

/\* TO create an empty node \*/

void create()

{

int data;

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

temp->prev = NULL;

temp->next = NULL;

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

scanf("%d", &data);

temp->n = data;

count++;

}

/\* TO insert at beginning \*/

void insert1()

{

if (h == NULL)

{

create();

h = temp;

temp1 = h;

}

else

{

create();

temp->next = h;

h->prev = temp;

h = temp;

}

}

/\* To insert at end \*/

void insert2()

{

if (h == NULL)

{

create();

h = temp;

temp1 = h;

}

else

{

create();

temp1->next = temp;

temp->prev = temp1;

temp1 = temp;

}

}

/\* To insert at any position \*/

void insert3()

{

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();

h = temp;

temp1 = h;

return;

}

else

{

while (i < pos)

{

temp2 = temp2->next;

i++;

}

create();

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--;

}

/\* Traverse from beginning \*/

void traversebeg()

{

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);

}

/\* To traverse from end recursively \*/

void traverseend(int i)

{

if (temp2 != NULL)

{

i = temp2->n;

temp2 = temp2->next;

traverseend(i);

printf(" %d ", i);

}

}

/\* To search for an element in the list \*/

void search()

{

int data, count = 0;

temp2 = h;

if (temp2 == NULL)

{

printf("\n Error : List empty to search for data");

return;

}

printf("\n Enter value to search : ");

scanf("%d", &data);

while (temp2 != NULL)

{

if (temp2->n == data)

{

printf("\n Data found in %d position",count + 1);

return;

}

else

temp2 = temp2->next;

count++;

}

printf("\n Error : %d not found in list", data);

}

/\* To update a node value in the list \*/

void update()

{

int data, data1;

printf("\n Enter node data to be updated : ");

scanf("%d", &data);

printf("\n Enter new data : ");

scanf("%d", &data1);

temp2 = h;

if (temp2 == NULL)

{

printf("\n Error : List empty no node to update");

return;

}

while (temp2 != NULL)

{

if (temp2->n == data)

{

temp2->n = data1;

traversebeg();

return;

}

else

temp2 = temp2->next;

}

printf("\n Error : %d not found in list to update", data);

}

/\* To sort the linked list \*/

void sort()

{

int i, j, x;

temp2 = h;

temp4 = h;

if (temp2 == NULL)

{

printf("\n List empty to sort");

return;

}

for (temp2 = h; temp2 != NULL; temp2 = temp2->next)

{

for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)

{

if (temp2->n > temp4->n)

{

x = temp2->n;

temp2->n = temp4->n;

temp4->n = x;

}

}

}

traversebeg();

}

**/\***

**\* C Program to Implement Pancake Sort on Array of Integers**

**\*/**

#include <stdio.h>

#include <stdlib.h>

void do\_flip(int \*, int, int);

/\*Function to implement the pancake sort\*/

int pancake\_sort(int \*list, unsigned int length)

{

if (length < 2)

return 0;

int i, a, max\_num\_pos, moves;

moves = 0;

for (i = length;i > 1;i--)

{

max\_num\_pos = 0;

for (a = 0;a < i;a++)

{

if (list[a] > list[max\_num\_pos])

max\_num\_pos = a;

}

if (max\_num\_pos == i - 1)

continue;

if (max\_num\_pos)

{

moves++;

do\_flip(list, length, max\_num\_pos + 1);

}

do\_flip(list, length, i);

}

return moves;

}

/\*Function to do flips in the elements\*/

void do\_flip(int \*list, int length, int num)

{

int swap;

int i = 0;

for (i;i < --num;i++)

{

swap = list[i];

list[i] = list[num];

list[num] = swap;

}

}

/\*Function to print the array\*/

void print\_array(int list[], int length)

{

int i;

for (i = 0;i < length;i++)

{

printf("%d ", list[i]);

}

}

int main(int argc, char \*\*argv)

{

int list[9];

int i;

printf("enter the 9 elements of array:\n");

for (i = 0;i < 9;i++)

scanf("%d", &list[i]);

printf("\nOriginal: ");

print\_array(list, 9);

int moves = pancake\_sort(list, 9);

printf("\nSorted: ");

print\_array(list, 9);

printf(" - with a total of %d moves\n", moves);

}

**/\***

**\* C Program to Traverse the Tree Recursively**

**\*/**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int a;

struct node \*left;

struct node \*right;

};

void generate(struct node \*\*, int);

void infix(struct node \*);

void postfix(struct node \*);

void prefix(struct node \*);

void delete(struct node \*\*);

int main()

{

struct node \*head = NULL;

int choice = 0, num, flag = 0, key;

do

{

printf("\nEnter your choice:\n1. Insert\n2. Traverse via infix\n3.Traverse via prefix\n4. Traverse via postfix\n5. Exit\nChoice: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter element to insert: ");

scanf("%d", &num);

generate(&head, num);

break;

case 2:

infix(head);

break;

case 3:

prefix(head);

break;

case 4:

postfix(head);

break;

case 5:

delete(&head);

printf("Memory Cleared\nPROGRAM TERMINATED\n");

break;

default: printf("Not a valid input, try again\n");

}

} while (choice != 5);

return 0;

}

void generate(struct node \*\*head, int num)

{

struct node \*temp = \*head, \*prev = \*head;

if (\*head == NULL)

{

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

(\*head)->a = num;

(\*head)->left = (\*head)->right = NULL;

}

else

{

while (temp != NULL)

{

if (num > temp->a)

{

prev = temp;

temp = temp->right;

}

else

{

prev = temp;

temp = temp->left;

}

}

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

temp->a = num;

if (num >= prev->a)

{

prev->right = temp;

}

else

{

prev->left = temp;

}

}

}

void infix(struct node \*head)

{

if (head)

{

infix(head->left);

printf("%d ", head->a);

infix(head->right);

}

}

void prefix(struct node \*head)

{

if (head)

{

printf("%d ", head->a);

prefix(head->left);

prefix(head->right);

}

}

void postfix(struct node \*head)

{

if (head)

{

postfix(head->left);

postfix(head->right);

printf("%d ", head->a);

}

}

void delete(struct node \*\*head)

{

if (\*head != NULL)

{

if ((\*head)->left)

{

delete(&(\*head)->left);

}

if ((\*head)->right)

{

delete(&(\*head)->right);

}

free(\*head);

}

}