/\* 1. Write a menu driven C Program to create a dynamic array of n elements and Perform the following operations

1. insert a new element at a specified Position
2. delete an element at a specified position
3. Display
4. Exit

\*/

#include <stdio.h> int main()

{

int \*p, n, ele, ch, i, pos;

printf("Enter number of elements to create an Array:\t"); scanf("%d", &n);

p = malloc(n \* sizeof(int));

printf("Dynamic Array Created.\n");

printf("Enter %d elements\n ", n);

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

{

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

}

while (1)

{

printf("\n 1.Insert\n 2.delete\n 3.display \n 4.Exit\n Enter your choice:\t");

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

{

case 1:

printf("\n Enter element & Pos(0 to %d) to insert:\t", n - 1); scanf("%d%d", &ele, &pos);

realloc(p, (n+1) \* sizeof(int));

n = n + 1;

for (i = n - 1; i >= pos; i--)

{

p[i] = p[i - 1];

}

p[pos] = ele;

break;

case 2:

printf("Enter Position(0 to %d) to delete:\t", n - 1); scanf("%d", &pos);

for (i = pos + 1; i < n; i++)

{

p[i - 1] = p[i];

}

n = n - 1;

break;

case 3:

printf("\n Array Elements Are:\n"); for (i = 0; i < n; i++)

{

printf("%d\t", p[i]);

}

break; case 4:

exit(0);

}

}

return 0;

}

**OUTPUT**

/\*

1. Write a menu driven program for the following operations
   1. Create a sparse Matrix
   2. Transpose of sparse Matrix
   3. Exit

\*/

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

#define MAX 100

struct term {

int row; int col; int value;

};

struct term sparse[MAX],trans[MAX]; int size;

void create(); void transpose();

void display(int values,struct term matrix[]); int main() {

int choice; while(1) {

printf("\nMenu:\n");

printf("1. Create Sparse Matrix\n"); printf("2. Transpose of Sparse Matrix\n");

// printf(". Display Sparse Matrix\n"); printf("3. Exit\n");

printf("Enter your choice: "); scanf("%d", &choice);

switch (choice) { case 1:

create(); break;

case 2:

transpose(); break;

case 3:

exit(0);

}

}

return 0;

}

void create() {

int matrix[10][10];

int i,rows,cols,values;

printf("\nEnter number of Rows,Columns and number of Values :"); scanf("%d%d%d",&rows,&cols,&values);

sparse[0].row = rows; sparse[0].col = cols;

sparse[0].value = values;

for(i=1;i<=values;i++)

{

printf("\n Enter row,col and value:"); scanf("%d%d%d",&sparse[i].row,&sparse[i].col,&sparse[i].value);

}

display(values,sparse);

}

void transpose() { int i,values;

// trans[0].row = sparse[0].col;

// trans[0].col = sparse[0].row;

// trans[0].value = sparse[0].value; values=sparse[0].value; for(i=0;i<=sparse[0].value;i++)

{

trans[i].row=sparse[i].col; trans[i].col=sparse[i].row; trans[i].value=sparse[i].value;

}

display(values,trans);

}

void display(int values,struct term a[]) { int i;

printf("\n\t Row\tColumn\tValue\n"); for (i = 0; i <= values; i++) {

printf("a[%d]: %d\t%d\t%d\n",i, a[i].row, a[i].col, a[i].value);

}

}

/\*

1. Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)
   1. Push an Element on to Stack
   2. Pop an Element from Stack
   3. Demonstrate Overflow and Underflow situations on Stack
   4. Display the status of Stack
   5. Exit

Support the program with appropriate functions for each of the above operations

\*/

#include <stdio.h> #define MAX 6

int stack[MAX], ele, num, top = -1; void push(int);

int pop();

void stakstatus(); void display(); int main()

{

int ch; while (1)

{

printf("\n1.Push \n2.Pop \n3.Stack Status \n4.Display\n 5.Exit \n Enter Your choice: ");

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

{

case 1:

printf("\n Enter element to Push: "); scanf("%d", &ele);

push(ele); break;

case 2:

ele = pop();

printf("\n Popped element from stack: %d", ele); break;

case 3:

stakstatus(); break;

case 4:

display(); break;

case 5:

exit(0);

}

}

}

void push(int ele)

{

if (top == MAX - 1) // if top==MAX-1 stack is full ...

{

printf("\n Stack is Overflow...\n");

}

else

{

stack[++top] = ele; // Increment top and push element to stack

}

}

int pop()

{

if (top == -1) // if top=-1 stack is empty you cannot pop element

{

}

else

{

}

}

printf("\n Stack is underflow! \n");

return stack[top--]; // pop last element inserted from stack

void stakstatus()

{

if (top == MAX - 1) // Check the condition to stack full or not

{

printf("Stack is Full!");

}

display();

}

void display()

{

int i;

if (top == -1)

{

printf("Stack is empty!\n");

}

else

{

printf(“Stack eles are \n”);

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

{

Printf(“%d \n”,stack[i]);

}

}

}

**O/P:**

4. Develop a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, \*, /, % (Remainder), ^ (Power) and alphanumeric operands.

#include<stdio.h> #include<ctype.h> char stack[20]; int top=-1;

void push(char ele); char pop();

int priority(char sym); int main()

{

int i=0; char exp[20]; char sym,ele;

printf("Enter valid Infix expression:"); scanf("%s",exp);

printf("\n Postfix:");

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

{

sym=exp[i];

if(isalnum(sym))

printf("%c ",sym); // print output if it is operand alphabet else if(sym=='(')

push(sym); // push symbol if it lparenthsis else if(sym==')') // if rparenthesis encountered

{

while((ele=pop())!='(') // Pop all the elements from stack till printf("%c ",ele);

}

else

{

while(priority(stack[top])>=priority(sym))

printf("%c ",pop()); // and stack[top] symbol print on output push(sym); // Push the symbol

}

}

while(top!=-1) // Pop remaining all the ele from stack and print

{

printf("%c ",pop());

}

return 0;

}

void push(char ele)

{

stack[++top]=ele;

}

char pop()

{

return stack[top--];

}

int priority(char sym)

{

if(sym=='(')

return 0;

if(sym=='+'|| sym=='-')

return 1;

if(sym=='\*'|| sym=='/'|| sym=='%')

return 2;

if(sym=='^')

return 3;

return 0;

}

O/P:

/\*

5.Develop a Program in C for the following Stack Applications

a. Evaluation of Suffix expression with single digit operands and operators:

+, -, \*, /, %,^

\*/

#include<stdio.h> #define MAX 10

int stack[MAX],top=-1; void push(int);

int pop();

void eval(int op1,char sym,int op2); int main()

{

int i=0,op1,op2; char exp[20]; char sym;

printf("Enter postfix expression:\t"); scanf("%s",exp); for(i=0;exp[i]!='\0';i++)

{

sym=exp[i];

if(isdigit(sym)) // Check given symbol is a digit or operator

{

push(sym-'0'); // Symbol converted to int

}

else{

op2=pop(); // Pop the top ele as op2 op1=pop(); // Pop the next top ele as op1

eval(op1,sym,op2); // Call eval function to find res..

}

}

printf("Result of given expression=%d",pop()); **// Last ele in stack is res**

return 0;

}

void push(int ele)

{

stack[++top]=ele;

}

int pop()

{

return stack[top--];

}

void eval(int op1,char sym,int op2)

{

int res; switch(sym)

{

case '+':res= op1+op2; // Calculate addition of Op1+Op2 push(res); // Result Push it into stack break;

case '-':res= op1-op2;

push(res); break;

case '\*':res= op1\*op2;

push(res); break;

case '/':res= op1/op2;

push(res); break;

case '%':res= op1%op2;

push(res); break;

}

}

**O/P**

/\*

6.Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

1. Insert an Element on to Circular QUEUE
2. Delete an Element from Circular QUEUE
3. Demonstrate Overflow and Underflow situations on Circular QUEUE
4. Display the status of Circular QUEUE
5. Exit

#include <stdio.h> #include <string.h> #include <stdlib.h> #define MAX 5

char cqueue[MAX], element;

int front = 0, rear = -1, count = 0; // count set to 0 Initially Q Empty. void insert(char ele);

void delete(); void display(); int main()

{

int ch; while(1)

{

printf("\n1.insert\n2.delete \n3.display \n4.exit\n Enter Your Choice:"); scanf("%d", &ch);

switch (ch)

{

case 1:printf("\n Enter a char element to insert:");

scanf(" %c",&element); // Space is mandatory Before %c insert(element); // Call insert function

break; case 2:

delete (); break;

case 3:

display(); break;

case 4:

return;

}

}

return 0;

}

void insert(char ele)

{

if (count == MAX) // if count is MAX then queue is full

{

printf("Circular Queue is full\n"); return;

}

rear = (rear + 1) % MAX; //Find rear and insert element cqueue[rear] = ele; // Inserting element into queue count++; // Increment the count of elements printf("Element inserted into C-Queue. \n");

// display();

}

void delete()

{

if (count == 0)

{

printf("Circular Queue is empty\n"); return;

}

printf("Element deleted from C-Queue: %c\n", cqueue[front]); front = (front + 1) % MAX; // Change front position after deletion count-=1; // Decrement count after deleting an element.

}

void display()

{

int i;

if (count == 0)

{

}

else

{

printf("\n Circular Queue is empty!\n"); return;

printf("\n Circular Queue is empty!\n"); return;

for (i=front;i!=rear;i=(i+1)%MAX)

{

printf("%c \n", cqueue[i]);

}

printf("%c",cqueue[i]); // print last element

}

}

/\*

7.Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL)

of Student Data with the fields : USN, Name, Branch, rollno, PhNo

1. Create a SLL of N Students Data by using front insertion.
2. Display the status of SLL and count the number of nodes in it
3. Perform Insertion / Deletion at End of SLL
4. Perform Insertion / Deletion at Front of SLL(Demonstration of stack) e.Exit

\*/

#include <stdio.h> struct student

{

int rollno; char usn[20]; char name[50];

struct student \*link;

};

typedef struct student \*NODE;

int count = 0; // To count total no of nodes in a list.. NODE createNode();

void CreateSLL(); // Creating list by inserting fornt void DisplaySLL();

void InsertFront(); void InsertEnd(); void DeleteFront(); void DeleteEnd(); NODE first = NULL; int main()

{

int ch;

while(1)

{

printf("1.CreateSLL \n2.DisplaySLL \n3.Insert front \n4.Insert End \n5.Delete Front\n6.Delete End \n7.Exit\nEnter Your Choice:");

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

{

case 1:

CreateSLL(); break;

case 2:

DisplaySLL(); break;

case 3:

InsertFront(); break;

case 4:

InsertEnd(); break;

case 5:

DeleteFront(); break;

case 6:

DeleteEnd(); break;

case 7:

exit(0);

}

}

return 0;

}

NODE createNode()

{

NODE temp;

temp = malloc(sizeof(struct student)); printf("\n Enter Roll Number:"); scanf("%d", &temp->rollno);

printf("\n Enter Usn:"); scanf("%s", temp->usn); printf("\n Enter Student Name:"); scanf("%s", temp->name);

temp->link = NULL; // Make temp link null default

count++; // Increment the count new node is created return temp; // return newly created node

}

void CreateSLL()

{

int i, n; NODE temp;

printf("\n Enter number of students:"); scanf("%d", &n);

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

{

printf("Enter Student %d details:", i + 1);

temp = createNode(); // Copy student node to temp

if (first == NULL) // If first is NULL no nodes in the list

{

}

else

{

}

first = temp; // Make first node as Temp

temp->link = first;

first = temp; // Make temp as first node

printf("Student node inserted at Front of List \n");

}

DisplaySLL();

}

void DisplaySLL()

{

NODE cur = first; if (first == NULL)

{

printf("\n Student List is empty!"); return;

}

printf("\nStudents List:\n");

printf(" \n");

while (cur != NULL)

{

printf("%d\t%s\t%s\n",cur->rollno, cur->usn, cur->name); cur = cur->link;

}

printf("\n");

printf("Total Number of students: %d \n", count);

}

void InsertFront()

{

NODE temp = createNode(); temp->link = first;

first = temp;

DisplaySLL(); // Call CreateSLL to insert front of the list

}

void InsertEnd()

{

NODE temp, cur; // To create node temp node taken as reference cur = first; // make current as first

while (cur->link != NULL) // Go to end of the list

{

cur = cur->link; // Move current node to next node

}

temp = createNode(); // Create Node and copy to temp cur->link = temp; // Link new node to last node printf("Studnet Node inserted at end of the list!\n"); DisplaySLL(); // Display List

/\*

8.Develop a menu driven program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: ssn, Name, Salary

1. Create a DLL of N Employees Data by using end insertion.
2. Display the status of DLL and count the number of nodes in it.
3. Perform Insertion and Deletion at End of DLL.
4. Perform Insertion and Deletion at Front of DLL.
5. Exit

\*/

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

// C Structure for employee... struct Employee

{

struct Employee \*llink; // Left link of node int ssn; // Info.. of employee char name[50];

float sal;

struct Employee \*rlink; // Right link of node

};

typedef struct Employee \*NODE;

int count = 0; // to count total number of employees NODE first = NULL; // Start with empty list..

NODE createNode(); // Creating temp node with data

void createDll(); // Creating emp DLL by inserting at end void insertFront(); // Inserting at front of the list

void insertEnd(); // inserting at the end of the list void deleteFront(); // Deleting node at front of list void deleteEnd(); // delelting node at end of list void displayDll(); // displaying the list of employees int main()

{

int ch;

while (1)

{

printf("\n1.Create Emp DLL \n2.insert Front \n3.Insert End \n4.Delete Front \n5.Delete End \n6.Display DLL \n7.Exit\n Enter Your Choice: ");

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

{

case 1:

createDll(); // Create a DLL by inserting end break;

case 2:

insertFront(); // inserting front of list break;

case 3:

insertEnd(); // Inserting end of the list break;

case 4:

deleteFront(); // delete front of the list break;

case 5:

deleteEnd(); // delete end of the list break;

case 6:

displayDll(); break;

case 7:

exit(0);

}

}

return 0;

}

NODE createNode()

{

NODE temp;

temp = malloc(sizeof(struct Employee)); // allocate memory for new node printf("Enter emp SSN: \t"); // read all the details of emp scanf("%d", &temp->ssn);

printf("Enter emp Name: \t"); scanf("%s", temp->name); printf("Enter emp Salary: \t"); scanf("%f", &temp->sal);

temp->llink = NULL; // make new node left and right link NULL temp->rlink = NULL;

count++; // increment the count when new node is created.. return temp; // return newly created node with data

}

void createDll()

{

int i, n;

NODE temp, cur; // creating DLL by inserting end of the list

printf("\n Enter number of Employees:"); scanf("%d", &n);

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

{

printf("Enter Employee[%d] Details:\n", i + 1); temp = createNode(); // Get new node with data

if (first == NULL) // if no nodes in list temp node is first node

{

}

else

{

first = temp;

cur = first; // take cur is reference node to insert at last while (cur->rlink != NULL) // repeat till end of list

{

cur = cur->rlink; // traverse to next node

}

cur->rlink = temp; // attach temp to last node with rLink

temp->llink = cur; // store second last node address in last node

}

}

// call diplay of the list displayDll();

}

void insertFront()

{

NODE temp;

temp = createNode(); // Get new node with data

first->llink = temp; // store address of temp node in first node left link

temp->rlink = first; // insert front of the list by linking right of temp with first first = temp; // Make temp as first node

displayDll();

}

void insertEnd()

{

NODE cur = first;

// Create a new node with data NODE temp = createNode();

// Insert at end of the function while (cur->rlink != NULL)

{

cur = cur->rlink; // Move to Last node

}

cur->rlink = temp; // Insert last rlink

temp->llink = cur; // store prev node llink in last node

count++; // Increment the count when new node inserted.. displayDll();

}

void deleteFront()

{

NODE cur;

cur = first->rlink; // store second node in current node free(first); // delete front i.e first node

first = cur; // make second node as first node after deletion

// (or) use the below statements

// first->llink=NULL;

// first=first->rlink;

first->llink = NULL; // Make first node left link NULL count--; // when node is deleted displayDll();

}

void deleteEnd()

{

NODE cur, prev; // Prev stores the second last node

cur = first; // take cur as reference to delete last node while (cur->rlink != NULL) // repeat till end of the list

{

prev = cur; // Store second last node in cur node cur = cur->rlink; // Move to next node

}

free(cur); // delete the last Node

prev->rlink = NULL; // Make last node rlink null to get end of the list. count--; // decrement the count when node is deleted displayDll();

}

void displayDll()

{

int count = 0;

NODE cur; // take cur node as reference to populate data cur = first; // Make use of cur as first

if (first == NULL) // if first is null list is empty...

{

printf("\n List is Empty!"); return;

}

// Display the heading part of details.... printf("\n SSN \t Name \t\t Salary \n"); while (cur != NULL)

{

printf("%d \t %s \t\t %f \n ", cur->ssn, cur->name, cur->sal); cur = cur->rlink; // move to the next node till end of list..

count++; // increment the count to get total count of nodes

}

printf("\n Total Num of employees:%d\n ", count); // Dispaly total count of Emps...

}

/\*

9.Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers .

1. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
2. Traverse the BST in Inorder, Preorder and Post Order
3. Search the BST for a given element (KEY) and report the appropriate message
4. Exit

\*/

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

struct node

{

struct node \*leftchild; int data;

struct node \*rightchild;

};

typedef struct node \*treePointer;

treePointer root = NULL; // Initialize an empty tree treePointer createNode(int value);

treePointer insertBST(treePointer root, int value); void inorder(treePointer root);

void preorder(treePointer root); void postorder(treePointer root);

void search(treePointer root,int key); int main()

{

int values[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 10};

int i, ch, key, n = 10; // n is Size of the above array... while (1)

{

printf("1.Create BST\n 2.Traversals \n3.Search \n4.Exit \nEnter Your Choice:"); scanf("%d", &ch);

switch (ch)

{

case 1:

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

{

root = insertBST(root, values[i]);

}

printf("Binary Search Tree Constructed.\n "); break;

case 2:

printf("\n Inorder: "); inorder(root); printf("\n Pre order: "); preorder(root);

printf("\n Post order: "); postorder(root); printf("\n");

break; case 3:

printf("\n Enter the key to Search: "); scanf("%d", &key);

search(root, key); case 4:

exit(0);

}

}

return 0;

}

treePointer createNode(int value)

{

treePointer temp = malloc(sizeof(struct node)); temp->data = value;

temp->leftchild = NULL; temp->rightchild = NULL; return temp;

}

treePointer insertBST(treePointer root, int value)

{

if (root == NULL)

{

root = createNode(value);

}

else if (value < root->data)

{

root->leftchild = insertBST(root->leftchild, value);

}

else

{

root->rightchild = insertBST(root->rightchild, value);

}

return root;

}

void inorder(treePointer root)

{

if (root != NULL)

{

inorder(root->leftchild); // Visit Left Child printf("%d ", root->data); // Visit Root inorder(root->rightchild); // Visit Right Child

}

}

void preorder(treePointer root)

{

if (root != NULL)

{

printf("%d ", root->data); preorder(root->leftchild); preorder(root->rightchild);

}

}

void postorder(treePointer root)

{

if (root != NULL)

{

postorder(root->leftchild); postorder(root->rightchild); printf("%d ", root->data);

}

}

void search(treePointer root, int key)

{

treePointer temp ; temp= root;

while (temp != NULL)

{

if (key == temp->data) // Compare the node data with key

{

printf("Key found!\n"); return;

}

else if (key < temp->data) **// If key less the node data search left**

{

temp = temp->leftchild;

}

else // If key greater than the node data search right subtree

{

temp = temp->rightchild;

}

}

} O/P:

/\*

10.Develop a Program in C for the following operations on Graph(G) of Cities

1. Create a Graph of N cities using Adjacency Matrix.
2. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method \*/

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

#define MAX 10

int graph[MAX][MAX]; int visited[MAX]; int queue[MAX];

int front = -1, rear = -1;

void createGraph(int n)

{

int i, j;

printf("Enter the adjacency matrix:\n"); for (i = 0; i < n; i++)

{

printf("Enter row %d: ", i + 1); for (j = 0; j < n; j++)

{

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

}

}

}

void DFS(int start, int n)

{

int i;

printf("%d ", start);

visited[start] = 1; // Put 1 for strating vertex that is visited for (i = 0; i < n; i++)

{

if (!visited[i] && graph[start][i] == 1)

{

DFS(i, n); // Call DFS recursively to reach next vertex

}

}

}

void BFS(int start, int n)

{

int i, vertex; printf("%d ", start);

visited[start] = 1; // Put Starting vertex visited queue[++rear] = start; // Insert into queue visited first vertex while (front <= rear)

{

vertex = queue[front++]; // Pop the vertex visited for (i = 0; i < n; i++)

{

if (!visited[i] && graph[vertex][i] == 1)

{

printf("%d ", i);

visited[i] = 1;

queue[++rear] = i; // Insert all the vertices connected to Current

}

}

}

}

int main()

{

int ch,i;

int n, start; while (1)

{

printf("\n1.Create a Graph\n.2.DFS \n 3.BFS \n 4.Exit \n Enter your choice:\n");

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

{

case 1:

printf("Enter the number of cities: "); scanf("%d", &n);

createGraph(n); break;

case 2:

printf("\nEnter the starting node: "); scanf("%d", &start);

printf("\nNodes reachable from node %d using DFS: ", start); DFS(start, n);

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

{

visited[i] = 0;

}

break;

case 3:front = rear = -1; // Reset queue for BFS

printf("\nNodes reachable from node %d using BFS: ", start); BFS(start, n);

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

{

visited[i] = 0;

}

printf("\n"); case 4:exit(0);

break;

}

}

return 0;

}

/\*

11.Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers.

Develop a Program in C that uses Hash function H:

K →L as H(K)=K mod m (remainder method), and implement hashing

technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing. \*/

#include <stdio.h> #include <stdlib.h> #define MAX\_EMPLOYEES 5

#define HT\_SIZE 10

struct Employee

{

int key;

char name[30];

};

struct EmployeeHashTable

{

struct Employee \*employees[MAX\_EMPLOYEES];

};

int hash(int key, int m)

{

return key % m;

}

void insert(struct EmployeeHashTable \*ht, struct Employee \*emp, int m)

{

int index = hash(emp->key, m); // Find index to insert empin hash table while (ht->employees[index] != NULL)

{

index = (index + 1) % m; //**Linear probing finding next position to insert**

**}**

ht->employees[index] = emp;

}

void display(struct EmployeeHashTable \*ht, int m)

{

int i;

printf("Hash Table:\n"); for (i = 0; i < m; i++)

{

if (ht->employees[i] != NULL)

{

printf("Index %d: Key=%d, Name=%s\n", i, ht->employees[i]->key, ht->employees[i]->name);

}

else

{

printf("Index %d: Empty\n", i);

}

}

}

int main()

{

int i,m;

struct EmployeeHashTable ht;

for ( i = 0; i < MAX\_EMPLOYEES; i++)

{

ht.employees[i] = NULL;

}

m = HT\_SIZE;

// Create 4 emps key,values to insert into hash table struct Employee e1 = {1000, "Ram"};

struct Employee e2 = {1001, "Naga"};

struct Employee e3 = {1002, "Lakshmi"}; struct Employee e4 = {2002, "Sontosh"}; insert(&ht, &e1, m);

insert(&ht, &e2, m); insert(&ht, &e3, m); insert(&ht, &e4, m); display(&ht, m);

return 0;

}