/* 1. Write a menu driven C Program to create a dynamic array of n elements and

Perform the following operations

a) insert a new element at a specified Position

b) delete an element at a specified position

c) Display

d) 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; // update new size
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;
}
```

```
/*
2. Write a menu driven program for the following operations
a. Create a sparse Matrix
b. Transpose of sparse Matrix
c. 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("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;
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);
}
}
```

```
/*
3.Develop a menu driven Program in C for the following operations on STACK of
Integers (Array Implementation of Stack with maximum size MAX)
a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate Overflow and Underflow situations on Stack
d. Display the status of Stack
e. 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)
{
printf("\n Stack is Overflow...\n");
}
else
{
stack[++top] = ele;
}
}
int pop()
{
if (top == -1)
{
```

```
printf("\n Stack is underflow! \n");
}
else
{
return stack[top–];
}
}
void stakstatus()
{
if (top == MAX - 1)
{
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]);
}
}
}
```

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);
else if(sym=='(')
push(sym);
else if(sym==')')
{
while((ele=pop())!='(')
printf("%c ",ele);
```

```
output
}
else
{
while(priority(stack[top])>=priority(sym))
printf("%c ",pop());
push(sym);
}
}
while(top!=-1)
{
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;
}
```

/*

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))
{
push(sym-'0');
}
else{
op2=pop();
op1=pop();
```

```
eval(op1,sym,op2);
}
}
printf("Result of given expression=%d",pop());
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;
push(res);
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;
}
}
```

/*

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)

a. Insert an Element on to Circular QUEUE

b. Delete an Element from Circular QUEUE

c. Demonstrate Overflow and Underflow situations on Circular QUEUE

d. Display the status of Circular QUEUE

e. Exit

Support the program with appropriate functions for each of the above operations.*/

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX 5
char cqueue[MAX], element;
int front = 0, rear = -1, count = 0;
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);
```

```
insert(element);
break;
case 2:
delete ();
break;
case 3:
display();
break;
case 4:
return;
}
}
return 0;
}
void insert(char ele)
{
if (count == MAX)
{
printf("Circular Queue is full\n");
return;
}
rear = (rear + 1) % MAX;
cqueue[rear] = ele;
count++;
printf("Element inserted into C-Queue. \n");
}
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;
count-=1;
}
void display()
{
int i;
if (count == 0)
{
printf("\n Circular Queue is empty!\n");
return;
}
else
{
for (i=front;i!=rear;i=(i+1)%MAX)
{
printf("%c \n", cqueue[i]);
}
printf("%c",cqueue[i]);
}
}
```

```
/*
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
a.Create a SLL of N Students Data by using front insertion.
b.Display the status of SLL and count the number of nodes in it
c.Perform Insertion / Deletion at End of SLL
d.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;
NODE createNode();
void CreateSLL();
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;
count++;
return temp;
}
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();
if (first == NULL)
{
first = temp;
}
else
{
temp->link = first;
```

```
first = temp;
}
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();
}
```

```
void InsertEnd()
{
NODE temp, cur;
cur = first;
while (cur->link != NULL)
{
cur = cur->link;
}
temp = createNode();
cur->link = temp;
printf("Studnet Node inserted at end of the list!\n");
DisplaySLL();
}
void DeleteFront()
{
NODE cur;
cur = first->link;
free(first);
first = cur;
printf("Node deleted front of the list!\n");
count–;
DisplaySLL();
}
void DeleteEnd()
{
NODE cur, prev;
cur = first;
if (first == NULL)
{
printf("Student list is empty!\n");
return;
```

```
}
if (first->link == NULL)
{
free(first);
}
while (cur->link != NULL)
{
prev = cur;
cur = cur->link;
}
free(cur);
prev->link = NULL;
printf("Node deleted end of the list!\n");
count–;
DisplaySLL();
}
```

```
/*
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
a. Create a DLL of N Employees Data by using end insertion.
b. Display the status of DLL and count the number of nodes in it.
c. Perform Insertion and Deletion at End of DLL.
d. Perform Insertion and Deletion at Front of DLL.
e. Exit
*/

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
struct Employee
{
struct Employee *llink;
int ssn;
char name[50];
float sal;
struct Employee *rlink;
};
typedef struct Employee *NODE;
int count = 0;
NODE first = NULL;
NODE createNode();
void createDll();
void insertFront();
void insertEnd();
void deleteFront();
void deleteEnd();
void displayDll();
```

```
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();
break;
case 2:
insertFront();
break;
case 3:
insertEnd();
break;
case 4:
deleteFront();
break;
case 5:
deleteEnd();
break;
case 6:
displayDll();
break;
case 7:
exit(0);
}
```

```
}
return 0;
}
NODE createNode()
{
NODE temp;
temp = malloc(sizeof(struct Employee));
printf("Enter emp SSN: \t");
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;
temp->rlink = NULL;
count++;
return temp;
}
void createDll()
{
int i, n;
NODE temp, cur;
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();
if (first == NULL)
{
first = temp;
```

```
}
else
{
cur = first;
while (cur->rlink != NULL)
{
cur = cur->rlink;
}
cur->rlink = temp;
temp->llink = cur;
}
}
displayDll();
}
void insertFront()
{
NODE temp;
temp = createNode();
first->llink = temp;
temp->rlink = first;
first = temp;
displayDll();
}
void insertEnd()
{
NODE cur = first;
NODE temp = createNode();
while (cur->rlink != NULL)
{
cur = cur->rlink;
}
```

```
cur->rlink = temp;
temp->llink = cur;
count++;
displayDll();
}
void deleteFront()
{
NODE cur;
cur = first->rlink;
free(first);
first = cur;
first->llink = NULL;
count–;
displayDll();
}
void deleteEnd()
{
NODE cur, prev;
cur = first;
while (cur->rlink != NULL)
{
prev = cur;
cur = cur->rlink;
}
free(cur);
prev->rlink = NULL;
count–;
displayDll();
}
void displayDll()
{
```

```
int count = 0;
NODE cur;
cur = first;
if (first == NULL)
{
printf("\n List is Empty!");
return;
}
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;
count++;
}
printf("\n Total Num of employees:%d\n ", count);
}
```

```
/*
9.Develop a menu driven Program in C for the following operations on
Binary Search Tree (BST) of Integers .
a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Exit
*/

#include <stdio.h>
#include <stdlib.h>
struct node
{
struct node *leftchild;
int data;
struct node *rightchild;
};
typedef struct node *treePointer;
treePointer root = NULL;
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;
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);
printf("%d ", root->data);
inorder(root->rightchild);
}
```

```
}
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)
{
printf("Key found!\n");
return;
}
else if (key < temp->data)
```

```
subtree
{
temp = temp->leftchild;
}
else
{
temp = temp->rightchild;
}
}
}
```

/*

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

a. Create a Graph of N cities using Adjacency Matrix.

b. 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;
```

```
for (i = 0; i < n; i++)
{
if (!visited[i] && graph[start][i] == 1)
{
DFS(i, n);
}
}
}
void BFS(int start, int n)
{
int i, vertex;
printf("%d ", start);
visited[start] = 1;
queue[++rear] = start;
while (front <= rear)
{
vertex = queue[front++];
for (i = 0; i < n; i++)
{
if (!visited[i] && graph[vertex][i] == 1)
{
printf("%d ", i);
visited[i] = 1;
queue[++rear] = i;
vertex
}
}
}
}
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 yourchoice:\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;
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;
}
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;
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;
}
```