DATA STRUCTURES LABORATORY MANUAL

18CSL38

Prof. Ganesh Pai
P. A. College of Engineering, Mangaluru

2019

Table of Contents

```
/*
* FILENAME          : ArrayOperation.c
* PROBLEM STATEMENT:
* Menu driven Program for the following Array operations
*     a. Creating an Array of N Integer Elements
*     b. Display of Array Elements with Suitable Headings
*     c. Inserting an Element at a given valid Position
*     d. Deleting an Element at a given valid Position
*     e. Exit.
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/

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

void createArray (int **a, int capacity);
void printArray (int *a, int capacity);
void insert (int *a, int *n, int element, int position);
int  delete (int *a, int *n, int position);

int main()
{
    int choice, *a = NULL, element, position, noOfElements = 0, capacity;

    while(1)
    {
        printf("\n------------- Menu -------------\n"
               "1. Create Array\n"
               "2. Insert Element at a position\n"
               "3. Delete an element at a position\n"
               "4. Display array elements\n"
               "5. Exit\n"
               "Enter your choice: ");
        scanf("%d", &choice);
        printf("--------------------------------\n");

        switch(choice)
        {
            case 1: printf("\nEnter the array capacity: ");
                    scanf("%d", &capacity);

                    createArray(&a, capacity);
                    break;

            case 2: if(a == NULL)
                        printf("\nError: Array not created.\n");
                    else if(noOfElements == capacity)
                        printf("\nArray Overflow.\n");
                    else
                    {
                        printf("\nEnter an integer element & its insert position(0-%d): ", noOfElements);
                        scanf("%d %d", &element, &position);

                        insert(a, &noOfElements, element, position);
                    }
                    break;

            case 3: if(a == NULL)
                        printf("\nError: Array not created.\n");
                    else if(noOfElements == 0)
                        printf("\nEmpty Array.\n");
                    else
                    {
                        printf("\nEnter the delete position(0-%d): ", noOfElements-1);
                        scanf("%d", &position);

                        element = delete(a, &noOfElements, position);

                        if(element != -1)
                            printf("Deleted element: %d\n", element);
                    }
                    break;

            case 4: if(a == NULL)
                        printf("\nError: Array not created.\n");
                    else
                        printArray(a, noOfElements);
                    break;

            case 5: return 0;

            default: printf("\nInvalid choice. Please re-enter your choice.\n");
        }
    }
}

void createArray(int **a, int capacity)
{
    *a = (int *)calloc(capacity, sizeof(int));   //create a dynamic array of size capacity
    if(*a == NULL)                               //print error message if memory allocation failed
        printf("Error: Memory allocation failed.");
}

void printArray(int *a, int n)
{
    printf("\nArray [ ");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("]\n");
}

void insert(int *a, int *n, int element, int position)
{
    if(position < 0)                    //print error message for negative value of position
    {
        printf("Error: Invalid position.\n");
        return;
    }
    if(position > *n)                  //if position entered is > n, set position to n
        position = *n;

    for(int i = *n; i > position; i--)  //shift elements one position right
        a[i] = a[i-1];
    a[position] = element;              //insert the elements in the array

    (*n)++;                             //increment number of elements in the array
}

int delete(int *a, int *n, int position)
{
    if(position < 0 || position > *n-1)       //print error message for position outside range.
    {
        printf("\nError: Invalid position.\n");
        return -1;
    }

    int element = a[position];          //backup deleting element
    for(int i = position; i < *n; i++)  //shift elements one position left
        a[i] = a[i + 1];
    (*n)--;                            //decrement number of elements in array

    return element;                    //return deleted element
}
```

Compilation Procedure:

```
> gcc -Wall filename.c -o filename
```

This generates an executable file named **`filename`**.

The **`-Wall`** flag to the compiler enables warning messages to be displayed by the compiler.

The **`-o filename`** option to the compiler says that the output/executable file need to be stored in **`filename.`**

If the program uses *math.h* library, then while compiling you need to use `–lm` flag.

Eg:

```
> gcc -Wall filename.c -o filename -lm
```

Execution Procedure:

```
> ./filename
```

```
/*
* FILENAME          : StringReplace.c
* PROBLEM STATEMENT:
* A program for the following string operations
*     a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
*     b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with
        REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR
* ----------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ----------------------------------------------------------------------------
*/

#include <stdio.h>
#define MAX 100

void readString (char src[], char pattern[], char replaceString[]);
int length (char string[]);
int indexOf (char string[], char pattern[], int startIndex);
int replaceAll (char src[], char dest[], char pattern[], char replaceString[]);

int main()
{
    char src[MAX], dest[MAX], pattern[MAX], replaceString[MAX];

    readString(src, pattern, replaceString);
    int count = replaceAll(src, dest, pattern, replaceString);
    if(count == 0)
    {
       printf("Pattern '%s' not found in source string '%s'.\n", pattern, src);
    }
    else
    {
       printf("\n%d match of pattern '%s' found in source string '%s'.", count, pattern, src);
       printf("\nNew String: '%s'\n", dest);
    }

    return 0;
}

//Reads the 3 input strings
void readString(char src[], char pattern[], char replaceString[])
{
    printf("Enter a Source string : ");
    fgets(src, MAX, stdin);
    src[length(src) - 1] = '\0';

    printf("Enter a Pattern string: ");
    fgets(pattern, MAX, stdin);
    pattern[length(pattern) - 1] = '\0';

    printf("Enter a Replace string: ");
    fgets(replaceString, MAX, stdin);
    replaceString[length(replaceString) - 1] = '\0';
}

//computes string length
int length(char string[])
{
    int length = 0;
    while(string[length++] != '\0');
    return (length - 1);
}
```

```
// Returns index if found, -1 otherwise
int indexOf(char string[], char pattern[], int startIndex)
{
    //if reached end of string or length of pattern is 0, return search failed
    if(length(string) == 0 || length(pattern) == 0)
        return -1;

    int maxIndex = length(string) - length(pattern);    //search for pattern from startIndex
    for (int i = startIndex; i <= maxIndex; i++)
    {
        int pi = 0;
        //run over matching characters
        for(int si = i; (pattern[pi] != '\0' && pattern[pi] == string[si]); pi++, si++);

        if(pattern[pi] == '\0')                         //return match position
            return i;
    }

    return -1;                                          //return match failed
}

// Returns count of match if found, 0 otherwise
int replaceAll(char src[], char dest[], char pattern[], char replaceString[])
{
    int matchIndex, si = 0, di = 0, count = 0;

    while((matchIndex = indexOf(src, pattern, si)) != -1)   //replace for each occurrences of pattern
    {
        count++;                                //increment match counter
        while(si < matchIndex)                  //copy initial char of source to destination string
            dest[di++] = src[si++];

        for(int j = 0; replaceString[j] != '\0'; j++)   //copy replace string to destination string
            dest[di++] = replaceString[j];

        si += length(pattern);                  //update index of source string
    }

    do dest[di++] = src[si];                    //copy remaining chars. of source to destination
    while(src[si++] != '\0');

    return count;                               //return number of matching strings
}
```

```
/*
* FILENAME         : stack.h
* PROBLEM STATEMENT:
*   Array implementation of Integer Stack
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/

#include<stdio.h>
#include<stdlib.h>
#define MAX 50

typedef struct
{
    int top, data[MAX];
} Stack;

int isEmpty(Stack s)                //returns 1 if empty, 0 otherwise
{   return (s.top == -1);   }

int isFull(Stack s)                 //returns 1 if full, 0 otherwise
{   return (s.top == MAX-1);    }

void push(Stack *s, int num)        //pushes an element to stack
{
    if(isFull(*s))
    {
        printf("\nError: Stack overflow\n");
        return;
    }
    s->data[++s->top] = num;
}

int pop(Stack *s)                 //removes an element from stack and
{                                 //returns element if exist, -1 otherwise
    if(isEmpty(*s))
    {
        printf("\nError: Stack underflow\n");
        return -1;
    }
    return s->data[s->top--];
}

void printStack(Stack s)            //prints all the elements of the stack
{
    printf("\nStack: [ ");
    for(int i = 0; i <= s.top; i++)
    {
        printf("%d ", s.data[i]);
    }
    printf("] <--TOP\n");
}

int topOf(Stack s)                  //returns top element of stack
{   return s.data[s.top];   }
```

```
/*
* FILENAME          : Stack_Palindrome.c
* PROBLEM STATEMENT:
* A menu driven Program for the following operations on STACK of Integers (Array Implementation)
*    a. Push an Element on to Stack
*    b. Pop an Element from Stack
*    c. Demonstrate how Stack can be used to check Palindrome
*    d. Demonstrate Overflow and Underflow situations on Stack
*    e. Display the status of Stack
*    f. Exit
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/
#include<stdio.h>
#include<math.h>
#include "stack.h"

int isPalindrome(int number);

int main()
{
    int choice, element, number;
    Stack stack = {-1};

    while(1)
    {
        printf( "\n------------ MENU ------------"
                "\n1. Push integer into Stack"
                "\n2. Pop integer from Stack"
                "\n3. Display the Stack"
                "\n4. Palindrome check"
                "\n5. Exit"
                "\nEnter a choice: ");
        scanf("%d", &choice);
        printf("------------------------------\n");

        switch(choice)
        {
            case 1: printf("\nEnter an integer to be inserted: ");
                    scanf("%d", &element);

                    push(&stack, element);
                    break;

            case 2: if( (element = pop(&stack)) != -1)
                        printf("\nPopped element: %d\n", element);
                    break;

            case 3: printStack(stack);
                    break;

            case 4: printf("\nEnter a number to check for palindrome: ");
                    scanf("%d", &number);

                    if(isPalindrome(number))
                        printf("%d is a palindrome.\n", number);
                    else
                        printf("%d is NOT a palindrome.\n", number);
                    break;

            case 5: return 0;

            default: printf("\nInvalid choice. Please re-enter your choice.\n");
}   }   }
```

```
int isPalindrome(int number)                          // check if array elements are in palindrome
{
    int reverse = 0, exp = 0;
    Stack s = {-1};

    for (int num = number; num != 0; num /= 10)       // push each digit of the number to the stack
        push(&s, num % 10);                           // push remainder to stack

    while(!isEmpty(s))                                // pop each digit from stack and
        reverse += pow(10, exp++) * pop(&s);          // generate reverse number

    return (number == reverse);                       // return 1 if equal, 0 otherwise
}
```

```
/*
* FILENAME          : InfixToPostfix.c
* PROBLEM STATEMENT:
* A program to convert and print a given valid parenthesized infix arithmetic
* expression to postfix expression. The expression consists of single character operands
* and the binary operators +, -, *, %, / and ^(exponent).
*
* Sample i/p:((A+B)*C-(D/E))^(F+G)
* o/p: AB+C*DE/-FG+^
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/

#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include "stack.h"

void infixToPostfix (char *infix, char *postfix);
int getPriority (char, char);
int priority(char);

int main()
{
    char infix[MAX] = {}, postfix[MAX] = {};

    printf("Enter a valid Infix expression with operators (+, -, *, /, %%, ^): ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("\nInfix Expression  : %s", infix);
    printf("\nPostfix Expression: %s\n", postfix);
}

void infixToPostfix (char *infix, char *postfix)
{
    char token;
    int i = 0, j = 0;
    Stack stack = {-1};

    while((token = infix[i++]) != '\0')
    {
        if(isalnum(token))
            postfix[j++] = token;
        else if(token == '(')
            push(&stack, token);
        else if(token == ')')
        {
            while((token = pop(&stack)) != '(')
                postfix[j++] = token;
        }
        else
        {
            while(!isEmpty(stack) && getPriority(topOf(stack), token))
                postfix[j++] = pop(&stack);
            push(&stack, token);
        }
    }
    while(!isEmpty(stack))
        postfix[j++] = pop(&stack);
}
```

```
int getPriority (char stkTop, char token)
{
    if (stkTop == '^' && token == '^')
        return 0;
    return (priority(stkTop) >= priority(token));
}

int priority(char token)
{
    switch(token)
    {
        case '(': return 1;
        case '+': case '-': return 2;
        case '*': case '/': case '%': return 3;
        case '^': return 4;
        default : printf("Error:Unknown symbol %c", token),
                  exit(0);
    }
}
```

```
/*
* FILENAME         : PostfixEvaluation.c
* PROBLEM STATEMENT:
* Program to evaluate Postfix Expression with operators:
* +, -, *, /, %, ^(Power).
*
* I/p = 21+3*52/-11+^    O/p: 49
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/

#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>
#include "stack.h"

int  evaluatePostfix (char *postfix);
int  calculate (int operand1, char operator, int operand2);

int main()
{
    char postfix[MAX];

    printf("\nEnter a valid Postfix Expression containing operators (+, -, *, /, %%, ^): ");
    scanf("%s", postfix);

    printf("\n%s = %d\n", postfix, evaluatePostfix(postfix));
}

int evaluatePostfix(char *postfix)
{
    char    token;
    int     i = 0, op1, op2, result;
    Stack   stack = {-1};

    while((token = postfix[i++]) != '\0')
    {
        if(isdigit(token))
            push(&stack, token - '0');
        else
        {
            op2 = pop(&stack);
            op1 = pop(&stack);
            result = calculate(op1, token, op2);
            push(&stack, result);
        }
    }

    return pop(&stack);
}

int calculate(int operand1, char operator, int operand2)
{
      switch(operator)
      {
          case '+': return operand1 + operand2;
          case '-': return operand1 - operand2;
          case '*': return operand1 * operand2;
          case '/': if(operand2 == 0)
                    {
                        printf("\nError: Division By Zero");
                        exit(0);
                    }
                    return operand1 / operand2;
          case '%': if(operand2 == 0)
                    {
                        printf("\nError:Floating point error.");
                        exit(0);
                    }
                    return operand1 % operand2;
          case '^': return pow(operand1, operand2);
          default : printf("\nError: Unknown Symbol '%c'", operator);
                    exit(0);
      }
}
```

```
/*
* FILENAME         : TowerOfHonai.c
* PROBLEM STATEMENT:
* Solving the Towers of Hanoi problem with n disks.
* -------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* -------------------------------------------------------------------------------
*/

#include<stdio.h>

int counter;
void tower(int disc, char *src, char *dest, char *aux);

int main()
{
    int noOfDisc;

    printf("Enter number of discs in the tower: ");
    scanf("%d", &noOfDisc);

    tower(noOfDisc, "Source", "Destination", "Auxilary");
    printf("\n\nTotal moves done: %d\n", counter);
}

void tower(int disc, char *src, char *dest, char *aux)
{
    if(disc > 0)
    {
        tower(disc-1, src, aux, dest);
        printf("\nMove disc %d from %s to %s", disc, src, dest);
        counter++;
        tower(disc-1, aux, dest, src);
    }
}
```

```
/*
* FILENAME         : CircularQueue.c
* PROBLEM STATEMENT:
* A menu driven Program for the following operations on Circular QUEUE of Characters
*  (Array Implementation)
*   a. Insert an Element to Circular QUEUE
*   b. Delete an Element from Circular QUEUE
*   c. Demonstrate Overflow and Underflow situations
*   d. Display the status of Circular QUEUE
*   e. Exit
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/
#include<stdio.h>
#define MAX 5

typedef struct cqueue
{
    int  rear, front;
    char data[MAX];
}CQueue;

void insert(CQueue *, char);
char delete(CQueue *);
int isEmpty(CQueue);
int isFull (CQueue);
void printQueue(CQueue);

int main()
{
    int   choice;
    char  element;
    CQueue q = {0, 0};

    while(1)
    {
        printf("\n--- Circular Queue Menu ---\n"
               "1. Insert an Element\n"
               "2. Delete an Element\n"
               "3. Display the status\n"
               "4. Exit\n"
               "Enter your choice : ");
        scanf("%d", &choice);
        printf("--------------------------\n");
        switch(choice)
        {
            case 1: printf("\nEnter the element to be inserted: ");
                    getc(stdin);    //skip \n character from previous input
                    scanf("%c", &element);
                    insert(&q, element);
                    break;

            case 2: element = delete(&q);
                    if(element != -1)
                        printf("\nDeleted element: %c\n", element);
                    break;

            case 3: printQueue(q);
                    break;

            case 4: return 0;

            default: printf("\nError: Invalid choice. Please re-enter your choice.");
}   }   }
```

```
void insert(CQueue *q, char element)
{
    if(isFull(*q))
    {
        printf("\nError: Circular Queue Overflow\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX;
    q->data[q->rear] = element;
}

char delete(CQueue *q)
{
    if(isEmpty(*q))
    {
        printf("\nError: Circular Queue Underflow\n");
        return -1;
    }
    q->front = (q->front + 1) % MAX;
    return q->data[q->front];
}

int isEmpty(CQueue q)
{   return (q.front == q.rear);    }

int isFull(CQueue q)
{   return (q.front == (q.rear + 1) % MAX);    }

void printQueue(CQueue q)
{
    if(isEmpty(q))
    {
        printf("\nEmpty Queue.\n");
        return;
    }

    int i = q.front;
    printf("\nCircular Queue [ ");
    do
    {
        i = (i + 1) % MAX;
        printf("%c ", q.data[i]);
    } while(i != q.rear);
    printf("]");
}
```

```
/*
* FILENAME         : SinglyLinkedList.c
* PROBLEM STATEMENT:
* A menu driven Program in C for the following operations on
* Singly Linked List (SLL) of Student Data with the fields:
*       USN, Name, Branch, Sem, 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
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/

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

typedef struct student
{
    char usn[11], name[50], branch[30], phNo[15];
    int sem;
    struct student *next;
}Node;

typedef Node * NodePtr;

NodePtr list = NULL;
int nodeCount;

void createList();
Node getStudentDetails();
void insertNode(Node, char);
void deleteNode(char);
void printList();

int main()
{
    int choice;

    while(1)
    {
        printf("\n--------- SLL Menu ---------"
                "\n\t"
                "\n[1]  Create Initial List"
                "\n[2]  Insert at Front"
                "\n[3]  Delete at Front"
                "\n[4]  Insert at End"
                "\n[5]  Delete at End"
                "\n[6]  Display all nodes"
                "\n[7]  Exit"
                "\nEnter a choice: ");
        scanf("%d", &choice);
        printf("--------------------------\n");

        switch(choice)
        {
            case 1: createList();
                    break;
            case 2: insertNode(getStudentDetails(), 'f');
                    break;
            case 3: deleteNode('f');
                    break;
            case 4: insertNode(getStudentDetails(), 'e');
                    break;
            case 5: deleteNode('e');
                    break;
            case 6: printList();
                    break;
            case 7: return 0;
            default: printf("Error: Invalid choice. Please re-enter your choice.");
        }//end of switch
    }//end of while
}

void createList()
{
    int n, i;

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

    printf("\nEnter %d student(s) details:\n\n", n);
    for(i = 1; i <= n; i++)
    {
        printf("Student %d\n", i);
        insertNode(getStudentDetails(), 'f');
        printf("\n");
}   }

Node getStudentDetails()
{
    Node s;
    s.next = NULL;

    printf("USN\t:");
    scanf("%s", s.usn);

    printf("Name\t:");
    scanf("%s", s.name);

    printf("Branch\t:");
    scanf("%s", s.branch);

    printf("Sem\t:");
    scanf("%d", &s.sem);

    getc(stdin);
    printf("Ph. No.\t:");
    scanf("%s", s.phNo);

    return s;
}

void insertNode(Node node, char ch)
{
    NodePtr newNode = (NodePtr) malloc(sizeof (Node));

    *newNode = node;
    newNode->next = NULL;

    if(list == NULL)
    {
        list = newNode;
    }
    else if(ch == 'f')  //insert at front of list
    {
        newNode->next = list;
        list    = newNode;
    }
    else                //insert at end of list
    {
        NodePtr p;
        for(p = list; p->next != NULL; p = p->next);  //move to end of list
        p->next = newNode;
    }
    nodeCount++;
}

void deleteNode(char ch)
{
    if(list == NULL)
    {
        printf("Error: Empty List!\n");
        return;
    }

    NodePtr p = list;
    if (ch == 'f')
    {
        list = list->next;
    }
    else
    {
        NodePtr q;
        for(; p->next != NULL; p = p->next)
            q = p;
        (p == list)? (list = NULL) : (q->next = NULL);
    }

    free(p);
    nodeCount--;
}

void printList()
{
    if(list == NULL)
    {
        printf("\nEmpty List!\n");
        return;
    }

    printf("Node Count: %d\n", nodeCount);
    printf("\nStatus of List: [USN,Name,Branch,Semester,Phone No.]\n");
    for(NodePtr p = list; p != NULL; p = p->next)
    {
        printf("[%s,%s,%s,%d,%s]", p->usn, p->name, p->branch, p->sem, p->phNo);
        if(p->next != NULL)
            printf("-->");
}   }
```

```
/*
* FILENAME         : DoublyLinkedList.c
* PROBLEM STATEMENT:
* A menu driven Program for the following operations on Doubly Linked List (DLL) of Employee Data
*  with the fields: SSN, Name, Dept, Designation, Sal, PhNo
*   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. Demonstrate how this DLL can be used as Double Ended Queue
*   f. Exit
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/

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

typedef struct employee
{
    char ssn[11], name[50], dept[30], designation[20], phNo[15];
    int salary;
    struct employee *left, *right;
}Node;

typedef Node * NodePtr;
NodePtr list = NULL, lastNode = NULL;
int nodeCount;

void createList();
Node getEmployeeDetails();
void insertNode(Node, char);
void deleteNode(char);
void printList();

int main()
{
    int choice;

    while(1)
    {
        printf("\n--------- DLL Menu ---------"
                "\n\t"
                "\n[1]  Create Initial List"
                "\n[2]  Insert at Front"
                "\n[3]  Delete at Front"
                "\n[4]  Insert at End"
                "\n[5]  Delete at End"
                "\n[6]  Display all nodes"
                "\n[7]  Exit"
                "\nEnter a choice: ");
        scanf("%d", &choice);
        printf("------------------------------\n");

        switch(choice)
        {
            case 1: createList();
                    break;
            case 2: insertNode(getEmployeeDetails(), 'f');
                    break;
            case 3: deleteNode('f');
                    break;
            case 4: insertNode(getEmployeeDetails(), 'e');
                    break;
            case 5: deleteNode('e');
                    break;
            case 6: printList();
                    break;
            case 7: return 0;
            default: printf("Error: Invalid choice. Please re-enter your choice.");
        } //end of switch
    } //end of while
}

void createList()
{
    int n, i;

    printf("\nEnter the number of employees: ");
    scanf("%d", &n);

    printf("\nEnter %d employee details:\n\n", n);
    for(i = 1; i <= n; i++)
    {
        printf("Employee %d\n", i);
        insertNode(getEmployeeDetails(), 'e');
        printf("\n");
}   }

Node getEmployeeDetails()
{
    Node emp;
    emp.left = emp.right = NULL;

    printf("SSN\t:");
    scanf("%s", emp.ssn);

    printf("Name\t:");
    scanf("%s", emp.name);

    printf("Department\t:");
    scanf("%s", emp.dept);

    printf("Designation\t:");
    scanf("%s", emp.designation);

    printf("Salary\t:");
    scanf("%d", &emp.salary);

    printf("Ph. No.\t:");
    scanf("%s", emp.phNo);

    return emp;
}

void insertNode(Node node, char ch)
{
    NodePtr newNode = (NodePtr) malloc (sizeof(Node));
    *newNode = node;

    if(list == NULL)
    {
        list = lastNode = newNode;
    }
    else if(ch == 'f')  //insertion at front of list
    {
        newNode->right = list;
        list->left = newNode;
        list = newNode;
    }
    else                //insertion at end of list
    {
        lastNode->right = newNode;
        newNode->left = lastNode;
        lastNode = newNode;
    }
    nodeCount++;
}

void deleteNode(char ch)
{
    NodePtr p = list;

    if(list == NULL)
    {
        printf("\nEmpty List");
        return;
    }

    if(lastNode == list)   //Only one node in the list
    {
        list = lastNode = NULL;
    }
    else if(ch == 'f')  //delete node at front of list
    {
        list = list->right;
        list->left = NULL;
    }
    else                    //delete node at end of list
    {
        p = lastNode;
        lastNode = lastNode->left;
        lastNode->right = NULL;
    }

    free(p);
    nodeCount--;
}

void printList()
{
    if(list == NULL)
    {
        printf("\nEmpty List.\n");
        return;
    }

    printf("Node Count: %d\n", nodeCount);
    printf("\nStatus of List: [SSN,Name,Dept.,Designation,Salary,Ph. No.]\n");
    for(NodePtr p = list; p != NULL; p = p->right)
    {
        printf("[%s,%s,%s,%s,%d,%s]", p->ssn, p->name, p->dept, p->designation, p->salary, p->phNo);
        if(p->right != NULL)
            printf("<=>");
}   }
```

```
/*
* FILENAME         : PolynomialEvaluationAddition.c
* PROBLEM STATEMENT:
* A Program for the following operations on Singly Circular Linked List (SLL)
* with header nodes
*    a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
*    b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store
*       the result in POLYSUM(x,y,z)
* -----------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* -----------------------------------------------------------------------------
*/

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

typedef struct node
{
    int coef, exp_x, exp_y, exp_z;
    struct node *next;
}Polynomial;

Polynomial * getHeaderNode();
void readPolynomial(Polynomial *head);
void printPolynomial(Polynomial *head);
void insertNode(Polynomial *head, Polynomial node);
int evaluatePolynomial(Polynomial *head, int x, int y, int z);
void addPolynomial(Polynomial *head1, Polynomial *head2, Polynomial *head3);

int main()
{
    Polynomial *poly, *poly1, *poly2, *polySum;
    int x, y, z, choice;

    while(1)
    {
        printf("\n-------- Menu --------\n"
               "1. Polynomial Evaluation\n"
               "2. Polynomial Addition\n"
               "3. Exit\n"
               "Enter a choice: ");
        scanf("%d", &choice);
        printf("----------------------\n\n");

        switch(choice)
        {
        case 1: poly = getHeaderNode();
                printf("Enter a Polynomial with variables (x, y, z) to evaluate:\n");
                readPolynomial(poly);

                printf("Enter the value of variables x, y, z: ");
                scanf("%d %d %d", &x, &y, &z);

                int value = evaluatePolynomial(poly, x, y, z);

                printf("\nFor x = %d, y = %d, z = %d\n", x, y, z);
                printPolynomial(poly);
                printf("Value = %d\n", value);
                break;

        case 2: poly1 = getHeaderNode(); poly2 = getHeaderNode(); polySum = getHeaderNode();
                printf("\nEnter Polynomial 1:\n");
                readPolynomial(poly1);
                printf("\nEnter Polynomial 2:\n");
                readPolynomial(poly2);

                printf("\nPolynomial 1 ");
                printPolynomial(poly1);

                printf("\nPolynomial 2 ");
                printPolynomial(poly2);

                addPolynomial(poly1, poly2, polySum);

                printf("\nPolynomial Sum ");
                printPolynomial(polySum);
                break;

        case 3: return 0;
        default: printf("\nError: Invalid choice. Please re-enter your choice.");
}   }   }

Polynomial * getHeaderNode()
{
    Polynomial *headNode = (Polynomial *) malloc(sizeof(Polynomial));
    headNode->exp_x = headNode->exp_y = headNode->exp_z = -1;
    headNode->coef = 0;
    headNode->next = headNode;
    return headNode;
}

void readPolynomial(Polynomial *head)
{
    int i, n;
    Polynomial newNode;

    printf("Enter no. of terms in the polynomial: ");
    scanf("%d", &n);
    printf("Enter polynomial:\n");
    for(i = 0; i < n; i++)
    {
        printf("Term %d <coefficient x_exponent y_exponent z_exponent>: ", i+1);
        scanf("%d %d %d %d", &newNode.coef, &newNode.exp_x, &newNode.exp_y, &newNode.exp_z);
        insertNode(head, newNode);
}   }

void insertNode(Polynomial *head, Polynomial node)
{
    Polynomial *newNode = (Polynomial *) malloc(sizeof(Polynomial));
    *newNode = node;

    newNode->next = head->next;     //insert node at front of list
    head->next = newNode;
}

void printPolynomial(Polynomial *head)
{
    Polynomial *p;
    if(head->next == head)
    {
        printf ("Empty polynomial.");
        return;
    }

    printf("f(x,y,z) =");
    for(p = head->next; p != head; p = p->next)
    {
        printf(" %+d", p->coef);
        if(p->exp_x != 0)   printf("x^%d", p->exp_x);
        if(p->exp_y != 0)   printf("y^%d", p->exp_y);
        if(p->exp_z != 0)   printf("z^%d", p->exp_z);
    }
    printf("\n");
}

int evaluatePolynomial(Polynomial *poly, int x, int y, int z)
{
    int sum = 0;

    for(Polynomial *p = poly->next; p != poly; p = p->next)
    {
        sum += p->coef * pow(x, p->exp_x) * pow(y, p->exp_y) * pow(z, p->exp_z);
    }
    return sum;
}

void addPolynomial(Polynomial *polyHead1, Polynomial *polyHead2, Polynomial *polySum)
{
    Polynomial *poly1 = getHeaderNode(), *poly2 = getHeaderNode();  //create 2 new list
    Polynomial *p1, *p2, *prev_p1, *prev_p2;

    for(p1 = polyHead1->next; p1 != polyHead1; p1 = p1->next)       //duplicate the 2 lists
        insertNode(poly1, *p1);
    for(p2 = polyHead2->next; p2 != polyHead2; p2 = p2->next)
        insertNode(poly2, *p2);

    prev_p1 = poly1;
    for(p1 = poly1->next; p1 != poly1; p1 = p1->next)   //for each term in poly1
    {
        prev_p2 = poly2;
        for(p2 = poly2->next; p2 != poly2; p2 = p2->next)   //and for each term in poly2
        {
            //if degrees are same, add coefficients and insert to polySum
            if(p1->exp_x == p2->exp_x && p1->exp_y == p2->exp_y && p1->exp_z == p2->exp_z)
            {
                p1->coef += p2->coef;
                insertNode(polySum, *p1);

                //delete p1 & p2
                prev_p1->next = p1->next;
                prev_p2->next = p2->next;
                free(p1); free(p2);
                p1 = prev_p1;
                break;
            }
            prev_p2 = p2;
        }
        prev_p1 = p1;
    }

    //insert remaining nodes of poly1 & poly2 to polysum
    for(p1 = poly1->next; p1 != poly1; p1 = p1->next)
        insertNode(polySum, *p1);

    for(p2 = poly2->next; p2 != poly2; p2 = p2->next)
        insertNode(polySum, *p2);
}
```

```
/*
* FILENAME         : BinarySearchTree.c
* PROBLEM STATEMENT:
*  A menu driven Program 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 key element and report the appropriate message
*     d.Exit
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/

#include <stdio.h>
#include <stdlib.h>
#define MAX 50

typedef struct treenode
{
    int data;
    struct treenode *left, *right;
}* NODE;

NODE root = NULL;

void constructBinTree(int a[], int n);
void preOrderTraverse(NODE n);
void inOrderTraverse(NODE n);
void postOrderTraverse(NODE n);
int search(int key);

int main()
{
    int choice, a[MAX], key, n;

    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("\nEnter %d elements of the tree: ", n);
    for(int i=0; i<n; i++)
        scanf("%d", &a[i]);

    constructBinTree(a, n);

    while(1)
    {
        printf("\n-------------- Menu --------------- "
               "\n[1] Inorder Traversal "
               "\n[2] Preorder Traversal "
               "\n[3] Postorder Traversal "
               "\n[4] Search an integer in the tree "
               "\n[5] Exit "
               "\nEnter your choice: ");
        scanf("%d", &choice);
        printf("----------------------------------\n");

        switch(choice)
        {
            case 1: printf("\nInorder Traversal: ");
                    inOrderTraverse(root);
                    break;
            case 2: printf("\nPreorder Traversal: ");
                    preOrderTraverse(root);
                    break;
            case 3: printf("\nPostorder Traversal: ");
                    postOrderTraverse(root);
                    break;
            case 4: printf("\nEnter the search key:");
                    scanf("%d", &key);
                    if(search(key) == 1)
                        printf("\nKey '%d' found in the tree\n", key);
                    else
                        printf("\nKey '%d' not found in the tree\n", key);

                    break;
            case 5: return 0;
            default: printf("\nError: Invalid choice. Please re-enter your choice.");
}   }   }

void constructBinTree(int a[], int n)
{
    NODE    newNode, p, q;

    for(int i = 0; i < n; i++)
    {
        newNode = (NODE) malloc(sizeof(struct treenode));
        newNode->data = a[i];
        newNode->left = newNode->right = NULL;

        if(root == NULL)
            root = newNode;
        else
        {
            p = root;
            while(p != NULL)
            {
                q = p;
                if(a[i] < p->data)
                    p = p->left;
                else if(a[i] > p->data)
                    p = p->right;
                else
                {
                    free(newNode);
                    break;
                }
            }
            if(p == NULL)
                (a[i] < q->data)? (q->left = newNode) : (q->right = newNode);
        }
    }
}

void preOrderTraverse(NODE n)
{
    if(n != NULL)
    {
        printf("%d ", n->data);
        preOrderTraverse(n->left);
        preOrderTraverse(n->right);
}   }

void inOrderTraverse(NODE n)
{
    if(n != NULL)
    {
        inOrderTraverse(n->left);
        printf("%d ", n->data);
        inOrderTraverse(n->right);
}   }
```

```
void postOrderTraverse(NODE n)
{
    if(n != NULL)
    {
        postOrderTraverse(n->left);
        postOrderTraverse(n->right);
        printf("%d ", n->data);
}   }

int search(int key)
{
    NODE n = root;
    while(n != NULL && n->data != key)
        n = (key > n->data)? n->right : n->left;

    return (n != NULL);
}
```

```
/*
* FILENAME         : DFS.c
* PROBLEM STATEMENT:
*  A Program 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
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/

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

int readGraph(int graph[][MAX]);
void dfs(int graph[][MAX], int startVertex, int noOfVertices);

int main()
{
    int  graph[MAX][MAX] = {0}, startVertex;

    int noOfVertices = readGraph(graph);

    printf("\nEnter start vertex [1-%d]: ", noOfVertices);
    scanf("%d", &startVertex);

    dfs(graph, startVertex, noOfVertices);
} /* end of main */

/* Input function */
int readGraph(int graph[][MAX])
{
    int i, v1, v2, noOfEdges, noOfVertices;

    printf("Enter number of graph Vertices & Edges: ");
    scanf("%d%d", &noOfVertices, &noOfEdges);

    for(i = 1; i <= noOfEdges; i++)
    {
        printf("Enter start & end vertex of edge %d: ", i);
        scanf("%d %d", &v1, &v2);
        graph[v1][v2] = 1;
    }

    return noOfVertices;
}

void dfs(int a[][MAX], int startVertex, int noOfVertices)
{
    int i, v, visited[MAX] = {0};
    Stack s = {-1};

    visited[startVertex] = 1;
    v = startVertex ;

    printf("\nNodes reachable from start vertex %d: ", startVertex);
    while(1)
    {
        for (i = 1; i <= noOfVertices; i++)
        {
            if (v != i && a[v][i] == 1 && visited[i] == 0)
            {
                push(&s, i);
                visited[i] = 1;
            }
        }
        if (isEmpty(s))
            break;
        v = pop(&s);
        printf("%d  ", v);
    } /* end of while */
} /* end dfs */
```

```
/*
* FILENAME          : Hash.c
* PROBLEM STATEMENT:
* 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.
* ------------------------------------------------------------------------------
* Author      : Prof. Ganesh Pai, P.A.C.E.
* Version     : 2019
* ------------------------------------------------------------------------------
*/
#include <stdio.h>
#include <stdlib.h>
#define M 11
#define hash(key) (key % M)

typedef struct
{
    int key;
    char name[50];
}Employee;

int insert(Employee *ht[], Employee *emp);
void printHashTable(Employee *ht[]);

int main()
{
    Employee *ht[M] = {NULL}, *emp;
    int cntr = 0;

    while(++cntr <= M)
    {
        emp = (Employee *) malloc(sizeof(Employee));    //create employee record
        printf("Enter employee key & name (-1 q to exit): ");
        scanf("%d %s", &emp->key, emp->name);
        if(emp->key == -1) break;
        printf("Inserted at address %d\n", insert(ht, emp));
    }
    printHashTable(ht);
}

int insert(Employee *ht[], Employee *emp)
{
    int addr = hash(emp->key);          //get hash key
    while(ht[addr] != NULL)             //find for unused space
        addr = (addr + 1) % M;
    ht[addr] = emp;                     //insert record to empty space
    return addr;
}
void printHashTable(Employee *ht[])
{
    printf("\nAddr: Key , Name\n");
    for(int i = 0; i < M; i++)
    {
        if (ht[i] == NULL)
            printf("%3d : ---\n", i);
        else
            printf("%3d : %d, %s\n", i, ht[i]->key, ht[i]->name);
}   }
```