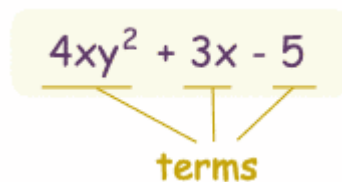


Program 1: Using circular representation for a polynomial, design, develop, and execute a program in C to accept two polynomials, add them, and then print the resulting polynomial.

About the Program

Polynomials

A polynomial looks like this:



The diagram shows the polynomial $4xy^2 + 3x - 5$ inside a light yellow rounded rectangle. Three yellow lines extend from the bottom of the rectangle, each pointing to one of the three terms: $4xy^2$, $3x$, and -5 . Below these lines, the word "terms" is written in a bold, yellow, sans-serif font.

example of a polynomial

this one has 3 terms

Polynomial comes from poly- (meaning "many") and -nomial (in this case meaning "term") ... so it says "many terms". A polynomial can have constants, variables and exponents, but never division by a variable.

To add two polynomials, we examine their terms starting at the nodes pointed to by a and b. if the exponents of the two terms are equal, we add two coefficients and create a new term for the result. We also move the pointers to the next nodes in a and b.

If the exponent of the current term in a is less than the exponent of the current term in b, then we create a duplicate term of b, attach this term to the result, called c, and advances the pointer to the next term in b. we take similar action on a if $a \rightarrow \text{expon} > b \rightarrow \text{expon}$.

Each time we generate a new node, we set its coef and expon fields and append it to the end of c. to avoid having to search for the last node in c each time we add a new node, we keep a pointer, rear, which points to the current last node in c. to create a new node and append it to the end of c, padd uses attach function.

We introduce a header node into each polynomial with circular list, that is, each polynomial, zero or nonzero, contains one additional node. The expon and coef fields of this node are irrelevant.

```
//Code: Add two Polynomials using circular list

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

typedef struct polynomial{
    int coef;
    int expon;
    struct polynomial *link;
}*POLYNODE;

int terms;

POLYNODE getNode ( int x, int y )
{
    POLYNODE temp;
    temp = ( POLYNODE ) malloc ( sizeof ( *temp ) );
    if ( !temp )
    {
        printf ( "\nAllocation Failed" );
        getchar ( );
        exit ( EXIT_FAILURE ); /* exit ( 1 ); */
    }
    else
    {
        {
            temp->coef = x;
            temp->expon = y;
            temp->link = temp;
            return temp;
        }
    }
}

void attach ( int coefficient, int exponent, POLYNODE *ptr )
{
    POLYNODE temp;
    temp = getNode ( coefficient, exponent );

    (*ptr)->link = temp;
    *ptr = temp;
}

POLYNODE makePoly ( void )
{
    POLYNODE c, lastC;

    int coef, expon;
    register int i;

    c = getNode( -1, -1 );
```

```

    lastC = c;

    for ( i = 0; i < terms; ++ i )
    {
        printf ( "\nGive Coef(%d) : ?\b", i + 1 );
        scanf ( "%d", &coef );
        printf ( "Give Expon(%d): ?\b", i + 1 );
        scanf ( "%d", &expon );

        attach ( coef, expon, &lastC ); /* Reusing attach function in order
to make circular list */

    }
    lastC->link = c;

    return c;
}

void displayPoly( POLYNODE poly )
{
    POLYNODE temp = poly->link; /* skip the header node */
    int c = 0;
    while ( poly != temp )
    {
        if ( ( c > 0 ) && ( temp->coef > 0 ) )
        {
            printf(" +");
        }
        printf ( " %dx^%d", temp->coef, temp->expon );
        temp = temp->link;
        c = 1;
    }
}

int COMPARE ( int a, int b )
{
    if ( a < b ) return -1;
    else if ( a == b ) return 0;
    else return 1;
}

POLYNODE cpadd ( POLYNODE a, POLYNODE b )
{
    /*
    Polynomials a and b are singly linked circular lists with a
header node.
    Return a polynomial which is the sum of a and b
    */
    POLYNODE startA, c, lastC;
    int sum, done = 0;
    startA = a; /* record start a */
    a = a->link; /* skip header node for a and b */
    b = b->link;
    c = getNode ( -1, -1 ); /* get a header node for sum */
    lastC = c;
    do

```

```

{
    switch ( COMPARE ( a->expon, b->expon ) )
    {
        case -1:    /*a->expon < b->expon) */
                    attach ( b->coef, b->expon, &lastC );
                    b = b->link;
                    break;

        case 0 :    /*a a-> expon = b->expon */
                    if ( startA == a )
                    {
                        done = 1;
                    }
                    else
                    {
                        sum = a->coef + b->coef;
                        if ( sum )
                        {
                            attach ( sum, a->expon, &lastC );
                        }
                        a = a->link;
                        b = b->link;
                    }
                    break;

        case 1 :    /* a->expon > b-> expon */
                    attach ( a->coef, a->expon, &lastC );
                    a = a->link;
                }
    }while ( !done );
    lastC->link = c;
    putchar('\n');
    return c;
}

int main ( void )
{
    POLYNODE poly1, poly2, poly3; /* Header nodes to hold the actual nodes */

    poly1 = poly2 = poly3 = NULL; /*header nodes*/

    printf ( "\n\nEnter the NO. of TERMS in the POLY-1: ?\b" );
    scanf ( "%d", &terms );
    poly1 = makePoly ( );

    printf ( "\n\nEnter the NO. of TERMS in the POLY-2: ?\b" );
    scanf ( "%d", &terms );
    poly2 = makePoly ( );

    printf ( "\n\nPOLYNOMIAL REPRESENTATION:" );

    printf ( "\n\nPOLY-1: " );
    displayPoly ( poly1 );

    printf ( "\n\nPOLY-2: " );
    displayPoly ( poly2 );

    poly3 = cpadd ( poly1, poly2 );
}

```

```

printf ( "\n\nPOLY-3: " );
displayPoly ( poly3 );

fflush ( stdin );
getchar ( );
return EXIT_SUCCESS; /* exit ( 0 ); */
}

```

Output:-

Enter the NO. of TERMS in the POLY-1: 3

Give Coef(1) : 3

Give Expon(1): 14

Give Coef(2) : 2

Give Expon(2): 8

Give Coef(3) : 1

Give Expon(3): 0

Enter the NO. of TERMS in the POLY-2: 3

Give Coef(1) : 8

Give Expon(1): 14

Give Coef(2) : -3

Give Expon(2): 10

Give Coef(3) : 10

Give Expon(3): 6

POLYNOMIAL REPRESENTATION:

POLY-1: $3x^{14} + 2x^8 + 1x^0$

POLY-2: $8x^{14} - 3x^{10} + 10x^6$

POLY-3: $11x^{14} - 3x^{10} + 2x^8 + 10x^6 + 1x^0$

Program 2: Design, develop, and execute a program in C to convert a given valid parenthesized infix arithmetic expression to postfix expression and then to print both

*the expressions. The expression consists of single character operands and the binary operators + (plus), - (minus), *(multiply) and / (divide).*

About the Program

Infix notation is the common arithmetic and logical formula notation, in which operators are written infix-style between the operands they act on (e.g. $2 + 2$). Infix notation is more difficult to parse by computers than prefix notation (e.g. $+ 2 2$) or postfix notation (e.g. $2 2 +$).

In infix notation, unlike in prefix or postfix notations, parentheses surrounding groups of operands and operators are necessary to indicate the intended order in which operations are to be performed. In the absence of parentheses, certain precedence rules determine the order of operations.

Reverse Polish notation (RPN) is a mathematical notation in which every operator follows all of its operands, in contrast to Polish notation, which puts the operator in the prefix position. It is also known as postfix notation and is parenthesis-free as long as operator arities are fixed.

The function `postfix` converts an infix expression into a postfix expression. This function invokes a function, `print_token`, to print out the character associated with the enumerated type. That is, `print_token` reverses the process used in `get_token`.

Analysis of postfix:

Let n be the number of token in the expression. $\Theta(n)$ time is spent extracting tokens and outputting them. Besides this, time is spent in the two while loops. The total time spent here is $\Theta(n)$ as the number of tokens that get stacked and unstacked is linear in n . so, the complexity of function `postfix` is $\Theta(n)$.

```
//Code: Infix to Postfix conversion
```

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

/*--Global variables--*/
int stack[50];
int top = 0;
char infix[50];
```

```

typedef enum {lparen, rparen, plus, minus, times, divide, mod, eos, operand}
PRECEDENCE;

/* isp and icp arrays -- index is value of precedence
   lparen, rparen, plus, minus, times, divide, mod, eos */
int isp[] = {0, 19, 12, 12, 13, 13, 13, 0};
int icp[] = {20, 19, 12, 12, 13, 13, 13, 0};

/*-- Function definition--*/
void getInfix ( void )
{
    printf ( "\nEnter the valid Infix expression\n" );
    gets ( infix );
}

void push ( int symbol )
{
    /* add an item to the global stack */
    stack[++ top] = symbol;
}

int pop ( void )
{
    /* delete and return the top element from the stack */
    return stack[top --];
}

PRECEDENCE getToken ( char *symbol, int *n )
{
    *symbol = infix[(*n)++];

    switch ( *symbol )
    {
        case '(' : return lparen;
        case ')' : return rparen;
        case '+' : return plus;
        case '-' : return minus;
        case '/' : return divide;
        case '*' : return times;
        case '%' : return mod;
        case '\\0' : return eos;
        case ' ' : return eos;
        default : return operand; /* no error checking,
                                   default is operand */
    }
}

void printToken ( int symbol )
{
    switch ( symbol )
    {
        case plus : putchar ( '+' ); break;
        case minus : putchar ( '-' ); break;
        case divide : putchar ( '/' ); break;
        case times : putchar ( '*' ); break;
        case mod : putchar ( '%' );
    }
}

```

```

    }
}

void postfix ( void )
{
    char symbol;
    PRECEDENCE token;
    int n = 0;
    stack[0] = eos;

    printf ( "\n\nEquivalent Postfix expression\n" );

    for ( token = getToken ( &symbol, &n ); token != eos; token = getToken
( &symbol, &n ) )
    {
        if ( token == operand )
        {
            putchar ( symbol );
        }
        else if ( token == rparen )
        {
            /* unstack tokens until left parenthesis */
            while ( stack[top] != lparen )
            {
                printToken ( pop ( ) );
            }
            pop ( ); /* discard the left parenthesis */
        }
        else
        {
            /* remove and print symbols whose isp is greater
            than or equal to the current token's icp */
            while ( isp[stack[top]] >= icp[token] )
            {
                printToken ( pop ( ) );
            }
            push ( token );
        }
    }
    while ( ( token = ( PRECEDENCE ) pop ( ) ) != eos )
    {
        printToken ( token );
    }
    putchar ( '\n' );
}

int main ( void )
{
    system ( "cls" );//clrscr();
    getInfix ( );
    postfix ( );
    fflush ( stdin );
    getchar();
    return 0;
}

```

Output:- 1

Enter the valid Infix expression

$((a*b)+(c/d))$

Equivalent Postfix expression

$ab*cd/+$

Output:- 2

Enter the valid Infix expression

$(a+b)*d+e/(f+a*d)+c$

Equivalent Postfix expression

$ab+d*efad*+ / +c+$

*Program 3: Design, develop, and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are + (add), - (subtract), * (multiply), / (divide).*

About the Program

The function eval contains the code to evaluate a postfix expression. Since an operand (symbol) is initially a character, we must convert it into a single digit integer. we use the statement symbol-'0', to accomplish this task.

We use an auxiliary function, getToken, to obtain tokens from the expression string. If the token is an operand, we convert it to a number and add it to the stack. Otherwise, we remove two operands from the stack, perform the specified operation, and place the result back on the stack.

When we have reached the end of expression, we remove the result from the stack.

Code: Evaluate a valid postfix expression using stack.

```
/* Preprocessor directives */
#include <stdio.h>
#include <stdlib.h>

/*--Global variables--*/
int stack[50];
int top = 0;
char postfix[20];

typedef enum {lparen, rparen, plus, minus, times, divide, mod, eos, operand}
PRECEDENCE;

/*-- Function defination--*/
void getPostfix ( void )
{
    printf ( "\nEnter the valid POSTFIX expression: ?\b" );
    gets ( postfix );
}

void push ( int symbol )
{

```

```

        /* add an item to the global stack */
        stack[++ top] = symbol;
    }

int pop ( void )
{
    /* delete and return the top element from the stack */
    return stack[top --];
}

PRECEDENCE getToken ( char *symbol, int *n )
{
    *symbol = postfix[(*n)++];
    switch ( *symbol )
    {
        case '(' : return lparen;
        case ')' : return rparen;
        case '+' : return plus;
        case '-' : return minus;
        case '/' : return divide;
        case '*' : return times;
        case '%' : return mod;
        case '\\0' : return eos;
        case ' ' : return eos;
        default : return operand; /* no error checking,
                                   default is operand */
    }
}

int eval ( void )
{
    PRECEDENCE token;
    char symbol;
    int op1, op2;
    int n = 0;
    token = getToken ( &symbol, &n );
    while ( token != eos )
    {
        if ( token == operand )
        {
            push ( symbol - '0' );
        }
        else
        {
            op2 = pop ( );
            op1 = pop ( );
            switch ( token )
            {
                case plus : push ( op1 + op2 ); break;
                case minus : push ( op1 - op2 ); break;
                case times : push ( op1 * op2 ); break;
                case divide : push ( op1 / op2 ); break;
                case mod : push ( op1 % op2 );
            }
        }
        token = getToken ( &symbol, &n );
    }
}

```

```

    }
    return pop ( );
}

void printResult ( void )
{
    printf ( "\n\nEvaluated POSTFIX expression: %d\n\n", eval ( ) );
}

int main ( void )
{
    system ( "cls" );

    getPostfix ( );

    printResult ( );

    fflush ( stdin );
    getchar ( );
    return 0;
}

```

Output:- 1

Enter the valid POSTFIX expression: 62/3-42*+

Evaluated POSTFIX expression: 8

Output:- 2

Enter the valid POSTFIX expression: 231*+9-

Evaluated POSTFIX expression: -4

Program 4: Design, develop, and execute a program in C to simulate the working of a Queue of integers using array. Provide the following operations:

a. Insert b.Delete c.Display

About the Program

Queue is an abstract data type or a linear data structure, in which the first element is inserted from one end called REAR (also called tail), and the deletion of existing element takes place from the other end called as FRONT (also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed first.

The process to add an element into queue is called ENQUEUE and the process of removal of an element from queue is called DEQUEUE.

Applications of Queue

Queue, as the name suggests is used whenever we need to have any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios:

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life, call center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, first come first served.

Code: Queue of integers using array.

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

typedef struct{
    int ele;
}ELEMENT;
ELEMENT *queue;
int rear = -1;
int front = 0;
int max;
```

```

void allocQueue ( void )
{
    queue = ( ELEMENT * ) malloc ( max * sizeof ( ELEMENT ) );
}

void queueFull ( void )
{
    printf ( "\n\nERROR: Queue is Full\n\n" );
}

void queueEmpty ( void )
{
    printf("\n\nERROR: Queue is Empty\n\n");
}

void addq ( ELEMENT item )
{
    /* add an item to the queue */
    if ( rear == max - 1 )
    {
        queueFull ( );
        return;
    }
    queue[++ rear] = item;
}

void deleteq ( void )
{
    /* remove element at the front of the queue */
    if ( front > rear )
    {
        queueEmpty ( );
        return;
    }
    printf ( "\n\nDeleted: %d", queue[front ++].ele );
    if ( front > rear )
    {
        front = 0;
        rear = -1;
    }
}

void display ( void )
{
    register int i;
    if ( front > rear )
    {
        queueEmpty ( );
        return;
    }
    printf( "\n\nThe Queue elements are..\n\n" );
    for ( i = front; i <= rear; ++ i )
    {
        printf ( "%d\t", queue[i].ele );
    }
}

```

```

int main ( void )
{
    ELEMENT temp;
    int ch;
    printf ( "\n\nEnter the size of the Queue: ?\b" );
    scanf ( "%d", &max );

    allocQueue ( );

    while ( 1 )
    {
        printf("\n\n\t\t -- QUEUE OPERATIONS --\n\n");
        printf("\t\t1.INSERT\t2.DELETE\t\n\t\t3.DISPLAY\t4.EXIT\n\n");
        printf("\nEnter your choice: ?\b");
        scanf("%d", &ch);
        system ( "cls" );
        switch ( ch )
        {
            case 1: display ( );
                    printf("\n\n\nEnter an element to Insert:
?\b");

                    scanf("%d", &temp.ele);
                    addq( temp );
                    break;

            case 2: display ( );
                    deleteq();
                    break;

            case 3: display();
                    break;

            case 4: return 0;
        }
        fflush ( stdin );
        getchar();
        system ( "cls" );
    }
}

```

Program 11: Design, develop, and execute a program in C to implement a doubly linked list where each node consists of integers. The program should support the following operations:

- i. Create a doubly linked list by adding each node at the front.*
- ii. Insert a new node to the left of the node whose key value is read as an input.*
- iii. Delete the node of a given data if it is found, otherwise display appropriate message.*
- iv. Display the contents of the list.*

About the Program

So far we have been working chiefly with chains and singly linked circular lists. One difficult with these lists is that if we are pointing to a specific node, say p, then we can move only in the direction of the links. The same problem arises when one wishes to delete an arbitrary node from a singly linked list.

The solution for this is to have doubly linked lists. Each node now has two link fields, one linking in the forward direction and the other linking in the backward direction.

A node in a doubly linked list has at least three fields, a left link field (llink), a data field (data), and a right link field (rlink).

The necessary declarations are:

```
struct node{
    struct node *llink;
    int data;
    struct node *rlink;
};
typedef struct node *DNODE;
```

In our doubly linked list program we have used a simply doubly linked circular list with three nodes. Besides three nodes, we have added a header node which allows us to implement our operations more easily. The data field of the header node usually contains no information.

To use doubly linked lists we must be able to insert and delete nodes. Assume that we have two nodes, targetNode and newNode, targetNode may be either header node or an interior node in a list. The function dinsert performs the insertion operation in constant time.

Deletion from a double linked list is equally easy. To accomplish this deletion, we only need to change the link field of the nodes that precede (deleted->llink->rlink) and follow (deleted->rlink->llink) the node we want to delete.

We have reused the dinsert function, when we want to insert a new node to the left of the node by pointing to left node of the key node.

Code: Doubly Linked List.

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

struct node{
    struct node *llink;
    int data;
    struct node *rlink;
};

typedef struct node *DNODE;

DNODE createNode ( int item );
void dInsert ( DNODE targetNode, DNODE newNode );
void insertItem ( DNODE head, int item );
DNODE findNode ( DNODE head, int key );
void insertLeftToKey ( DNODE head, int key, int item );
void ddelete ( DNODE deleted );
void delItem ( DNODE head, int item );
void display ( DNODE head );

DNODE createNode ( int item )
{
    DNODE temp;
    temp = ( DNODE ) malloc ( sizeof ( *temp ) );
```

```

    if ( !temp )
    {
        printf ( "\n\nInsufficient memory\n\n" );
        getchar ( );
        exit ( -1 );
    }
    else
    {
        temp->data = item;
        temp->llink = temp->rlink = temp;
        return temp;
    }
}

/** newNode will be added to right of the targetNode **/
void dInsert ( DNODE targetNode, DNODE newNode )
{
    newNode->llink = targetNode;
    newNode->rlink = targetNode->rlink;
    targetNode->rlink->llink = newNode;
    targetNode->rlink = newNode;
}

void insertItem ( DNODE head, int item )
{
    DNODE newNode = createNode( item );
    dInsert ( head, newNode );
}

DNODE findNode ( DNODE head, int key )
{
    DNODE node;
    for
    (
        node = head->rlink; /* Skip the Header Node */
        node != head; /* Condition */
        node = node->rlink /* Points to Next node */
    )

```

```

    {
        if ( node->data == key ){
            return node;
        }
    }
    return ( node = NULL );
}

void insertLeftToKey ( DNODE head, int key, int item )
{
    DNODE newNode;
    DNODE node = findNode ( head, key );
    if ( node )
    {
        newNode = createNode ( item );
        /* reusing the "dInsert" function by pointing to left node*/
        dInsert ( node->llink, newNode );
    }
    else
    {
        printf ("\n\nKey is not present\n");
    }
}

void ddelete ( DNODE deleted )
{
    deleted->llink->rlink = deleted->rlink;
    deleted->rlink->llink = deleted->llink;
    free ( deleted );
}

void delItem ( DNODE head, int item )
{
    DNODE node = findNode ( head, item );
    if ( node )
    {
        ddelete ( node );
        printf ( "\n\nTarget Item is deleted\n" );
    }
}

```

```

    }
    else
    {
        printf ( "\n\nTarget Item is not present\n" );
    }
}

void display ( DNODE head )
{
    DNODE temp;

    if ( head == head->rlink )
    {
        printf ( "\n\nDoubly Linked List is empty.\n" );
        return;
    }
    printf ( "\n\nDoubly Linked List Nodes...\n\n" );
    for
    (
        temp = head->rlink; /* Initilization */
        temp != head; /* Condition */
        temp = temp->rlink /* Points to Next node */
    )
    {
        printf ( "%3d", temp->data );
    }
}

int main ( void )
{
    int ch, item, key;
    DNODE head = NULL;
    system ( "cls" );//clrscr();
    head = createNode ( -1 );
    while ( 1 )
    {
        system ( "cls" );

```

```

printf ( "\n\n\t\t -- OPERATIONS --\n\n" );
printf ( "\t\t1.Insert Front\t2.Insert Left with
Key\t\n\t\t3.Display\t4.Delete wit Key\n\n" );
printf ( "\t\t\t5.Any Other key to Exit\n\n" );
printf ( "\nEnter your choice: ?\b" );
scanf ( "%d", &ch );
system ( "cls" );//clrscr();
switch ( ch )
{
    case 1: printf ( "\nElement to be Inserted: ?\b" );
            scanf ( "%d", &item );
            insertItem ( head, item );
            break;
    case 2: printf ( "\nElement to be Inserted: ?\b" );
            scanf ( "%d", &item );
            printf ( "\nKey to be Inserted Left: ?\b" );
            scanf ( "%d", &key );
            insertLeftToKey ( head, key, item );
            break;
    case 3: display ( head );
            break;
    case 4: display ( head );
            printf ( "\n\nElement to be deleted: ?\b" );
            scanf ( "%d", &item );
            delItem ( head, item );
            break;
    default: return 0;
}
fflush ( stdin );
getchar ( );
}
}

```

Output:-

```
C:\Windows\system32\cmd.exe - a.exe

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display      4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 1
Element to be Inserted: 100

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display      4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 1
Element to be Inserted: 200

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display      4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 1
Element to be Inserted: 300

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display      4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 2
```

```
C:\Windows\system32\cmd.exe - a.exe

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display       4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 3

Doubly Linked List Nodes...
300  200  100

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display       4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 2

Doubly Linked List Nodes...
300  200  100
Element to be Inserted: 150
Key to be Inserted Left: 100

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display       4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 3

Doubly Linked List Nodes...
300  200  150  100  _
```

```
C:\Windows\system32\cmd.exe - a.exe

      1.Insert Front  2.Insert Left with Key
      3.Display       4.Delete wit Key

      5.Any Other key to Exit

Enter your choice: 4

Doubly Linked List Nodes...
300  200  150  100
Element to be deleted: 200

Target Item is deleted

      -- OPERATIONS --

      1.Insert Front  2.Insert Left with Key
      3.Display       4.Delete wit Key

      5.Any Other key to Exit

Enter your choice: 3

Doubly Linked List Nodes...
300  150  100

      -- OPERATIONS --

      1.Insert Front  2.Insert Left with Key
      3.Display       4.Delete wit Key

      5.Any Other key to Exit

Enter your choice: 4

Doubly Linked List Nodes...
300  150  100
Element to be deleted: 100

Target Item is deleted
```



```
C:\Windows\system32\cmd.exe - a.exe

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display       4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 3

Doubly Linked List Nodes...
150

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display       4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 4

Doubly Linked List Nodes...
150
Element to be deleted: 150
Target Item is deleted

-- OPERATIONS --
1.Insert Front  2.Insert Left with Key
3.Display       4.Delete wit Key
5.Any Other key to Exit

Enter your choice: 3

Doubly Linked List is empty.
```

Program 9: Design, develop, and execute a program in C to read a sparse matrix of integer value and to search the sparse matrix for an element specified by the user. Print the result of the search appropriately. Use the triple <row, column, value> to represent an element in the sparse matrix.

Code: Sparse Matrix.

```
#include <stdio.h>
#include <conio.h>
struct element
{
    int row;
    int col;
    int value;
};
typedef struct element node;
node a[100];

void main ( )
{
    int keyval, b[50][50], i, j, k, m, n, flag;
    clrscr();
    printf ( "\nEnter the size of the matrix:" );
    scanf ( "%d %d", &m, &n );
    printf ( "\nEnter the elements of the two dimensional array\n" );
    for ( i = 0; i < m; i ++ )
        for ( j = 0; j < n; j ++ )
            scanf ( "%d", &b[i][j] );

    /*the non-zero elements are taken into triple representation*/
    k = 0;
    a[k].row = m;
    a[k].col = n;
    for ( i = 0; i < m; i ++ )
    {
        for ( j = 0; j < n; j ++ )
        {
```

```

        if ( b[i][j] != 0)
        {
            k ++;
            a[k].row = i;
            a[k].col = j;
            a[k].value = b[i][j];
        }
    }
}

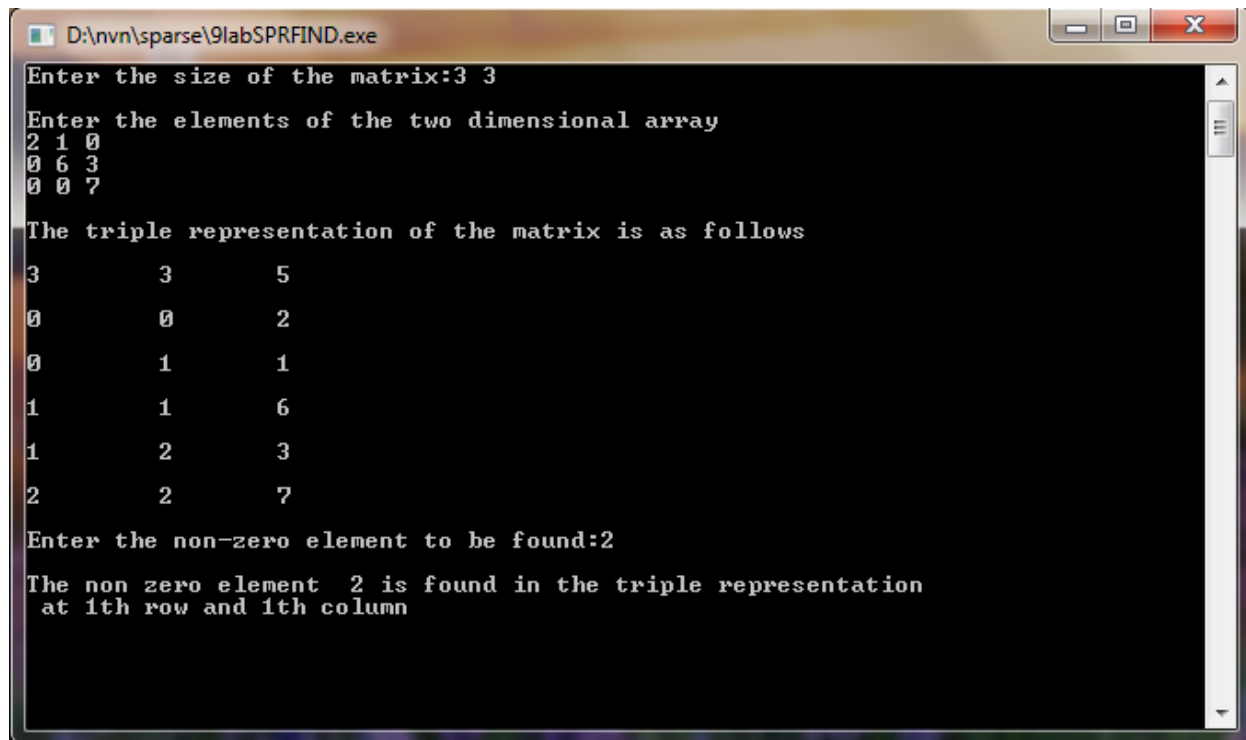
a[0].value = k;
printf ( "\nThe triple representation of the matrix is as follows\n" );
for ( i = 0; i <= k; i ++)
    printf ( "\n%d \t %d \t %d",a[i].row,a[i].col,a[i].value );
printf ( "\nEnter the non-zero element to be found:" );
scanf ( "%d", &keyval );
flag = 0;
for ( i = 1; i <= k; i ++)
{
    if ( a[i].value == keyval)
    {
        flag = 1;
        break;
    }
}

if ( flag == 1 )
    printf ( "\nThe non zero element  %d is found in the triple
representation \n at %dth row and %dth column",keyval,a[i].row+1,a[i].col+1
);

else if ( flag == 0 )
    printf ( "\nThe keyval %d is not present in the triple
representation", keyval );
getch ( );
}

```

OUTPUT:-



```
D:\nvn\sparse\9labSPRFIND.exe
Enter the size of the matrix:3 3
Enter the elements of the two dimensional array
2 1 0
0 6 3
0 0 7

The triple representation of the matrix is as follows
3      3      5
0      0      2
0      1      1
1      1      6
1      2      3
2      2      7

Enter the non-zero element to be found:2
The non zero element 2 is found in the triple representation
at 1th row and 1th column
```

Program 10: Design, develop, and execute a program in C to create a max heap of integers by accepting one element at a time and by inserting it immediately in to the heap. Use the array representation for the heap. Display the array at the end of insertion phase.

Code: Max heap of integers.

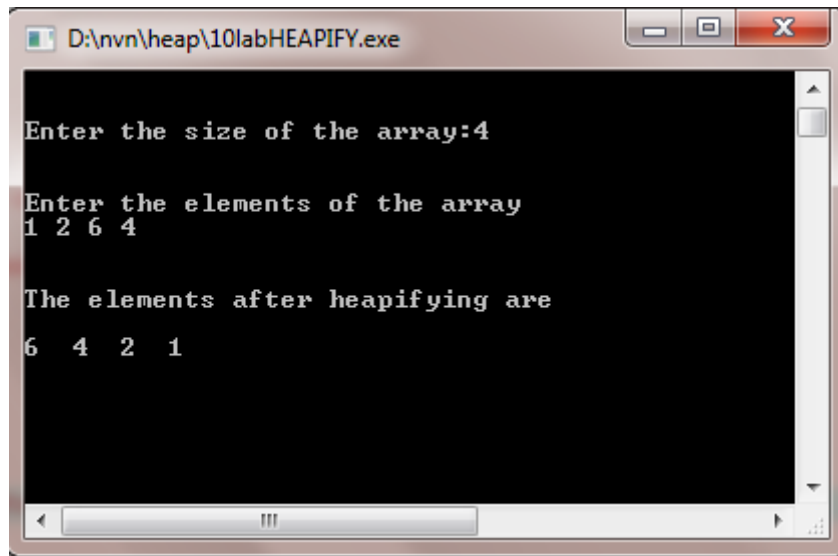
```
#include <stdio.h>

int main ( )
{
    int a[50], i, n;
    void insert ( int a[], int );
    printf ( "\n\nEnter the size of the array:" );
    scanf ( "%d", &n );
    printf ( "\n\nEnter the elements of the array\n" );
    for ( i = 1; i <= n; i ++ )
        scanf ( "%d", &a[i] );
    for ( i = 1; i <= n; i ++ )
    {
        insert ( a, i );
    }
    printf ( "\n\nThe elements after heapifying are\n\n" );
    for ( i = 1; i <= n; i ++ )
        printf ( "%-3d", a[i] );
    fflush ( stdin );
    getchar ( );
    return 0;
}

void insert ( int a[50], int n)
{
    int item;
    item = a[n];
    while ( n > 1 && item > a[n/2] )
    {
        a[n] = a[n/2];
```

```
        n = n / 2;  
    }  
    a[n] = item;  
}
```

OUTPUT:-



```
D:\nvn\heap\10labHEAPIFY.exe  
Enter the size of the array:4  
Enter the elements of the array  
1 2 6 4  
The elements after heapifying are  
6 4 2 1
```

