**Ex. No. 1a Array implementation of Stack ADT**

**Date:**

**Aim**

To implement stack operations using arrays.

**TUTORIAL:**

A stack data structure can be implemented using a one dimensional array. But a stack implemented using arrays can store only a fixed number of data values. This implementation is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable 'top'. Initially the top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

**Algorithm**

1. Start
2. Define a array *stack* of size *max* = 5
3. Initialize *top* = -1
4. Display a menu listing stack operations
5. Accept choice
6. If choice = 1 then

If top < max -1

Increment top

Store element at current position of top

Else

Print Stack overflow

Else If choice = 2 then

If top < 0 then

Print Stack underflow

Else

Display current top element Decrement top

Else If choice = 3 then

Display stack elements starting from top

1. Stop

**Program**

**/\* Stack Operation using Arrays \*/**

#include <stdio.h>

#include conio.h>

#define max 5

static int stack[max];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return (stack[top--]);

}

void view()

{

int i;

if (top < 0)

printf("\n Stack Empty \n"); else

{

printf("\n Top-->");

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

{

printf("%4d", stack[i]);

}

printf("\n");

}

}

main()

{

int ch=0, val; clrscr();

while(ch != 4)

{

printf("\n STACK OPERATION \n"); printf("1.PUSH ");

printf("2.POP ");

printf("3.VIEW ");

printf("4.QUIT \n"); printf("Enter Choice : ");

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

{

case 1:

if(top < max-1)

{

printf("\nEnter Stack element : "); scanf("%d", &val);

push(val);

}

else

printf("\n Stack Overflow \n"); break;

case 2:

if(top < 0)

printf("\n Stack Underflow \n");

else

{

val = pop();

printf("\n Popped element is %d\n", val);

}

break;

case 3:

view(); break;

case 4:

exit(0); default:

printf("\n Invalid Choice \n");

}

}

}

**Output**

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 1

Enter Stack element : 12 STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 1

Enter Stack element : 23

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 1

Enter Stack element : 34 STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 1

Enter Stack element : 45 STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 3

Top--> 45 34 23 12 STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 2 Popped element is 45

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 3

Top--> 34 23 12 STACK OPERATION

1. PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 4

**Result**

Thus push and pop operations of a stack were demonstrated using arrays.

**REVIEW QUESTIONS:**

1. Write down the definition of data structures?
2. What is meant by an Abstract Data Type(ADT)?
3. What is meant by stack ADT?
4. List out the operation in Stack ADT.
5. Write about push and pop operation in stack?
6. What is the use of Top of the stack?

**Ex. No. 1b Array implementation of Queue ADT**

**Date:**

**Aim**

To implement queue operations using arrays.

**TUTORIAL:**

A queue data structure can be implemented using a one dimensional array. But, a queue implemented using an array can store only a fixed number of data values. The implementation of queue data structure using arrays is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using FIFO (First In First Out) principle with the help of variables 'front' and 'rear'. Initially both 'front' and 'rear' are set to -1. Whenever, we want to insert a new value into the queue, increment the 'rear' value by one and then insert at that position. Whenever we want to delete a value from the queue, then increment the 'front' value by one and then display the value at 'front' position as deleted element.

**Algorithm**

* 1. Start
  2. Define a array *queue* of size *max* = 5
  3. Initialize *front = rear* = –1
  4. Display a menu listing queue operations
  5. Accept choice
  6. If choice = 1 then If rear < max -1

Increment rear

Store element at current position of rear

Else

Print Queue Full

Else If choice = 2 then If front = –1 then

Print Queue empty

Else Display Current front Element

Increment front

Else If choice = 3 then

Display queue elements starting from front to rear.

* 1. Stop

**Program**

**/\* Queue Operation using Arrays \*/**

#include <stdio.h>

#include <conio.h>

#define max 5

static int queue[max];

int front = -1;

int rear = -1;

void insert(int x)

{

queue[++rear] = x;

if (front == -1)

front = 0;

}

int remove()

{

int val;

val = queue[front];

if (front==rear && rear==max-1)

front = rear = -1;

else

front ++; return (val);

}

void view()

{

int i;

if (front == -1)

printf("\n Queue Empty \n");

else

{

printf("\n Front-->");

for(i=front; i<=rear; i++)

printf("%4d", queue[i]);

printf(" <--Rear\n");

}

}

main()

{

int ch= 0,val;

clrscr();

while(ch != 4)

{

printf("\n QUEUE OPERATION \n"); printf("1.INSERT ");

printf("2.DELETE ");

printf("3.VIEW ");

printf("4.QUIT\n");

printf("Enter Choice : "); scanf("%d", &ch);

switch(ch)

{

case 1:

if(rear < max-1)

{

printf("\n Enter element to be inserted : ");

scanf("%d", &val);

insert(val);

}

else

printf("\n Queue Full \n");

break;

case 2:

if(front == -1)

printf("\n Queue Empty \n");

else

{

val = remove();

printf("\n Element deleted : %d \n", val);

}

break; case 3:

view(); break;

case 4:

exit(0); default:

printf("\n Invalid Choice \n");

}

}

}

**Output**

QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 12 QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 23 QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 34 QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 45 QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 56 QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1 Queue Full

QUEUE OPERATION

1. INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 3

Front--> 12 23 34 45 56 <--Rear

**Result**

**Thus insert and delete operations of a queue was demonstrated using arrays.**

**REVIEW QUESTIONS:**

1. What is a Queue?
2. Mention applications of queue.
3. Define Priority Queue.
4. Write about enqueue and dequeue operation.

**Ex. No. 1.c Array implementation of Circular Queue ADT**

**Date:**

**Aim:**

To Write a c program for implementing Circular Queue ADT using array.

**TUTORIAL**

There was one limitation in the array implementation of [Queue.](https://www.javatpoint.com/data-structure-queue) If the rear reaches to the end position of the Queue then there might be the possibility that some vacant spaces are left in the beginning which cannot be utilized. So, to overcome such limitations, the concept of the circular queue was introduced. A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a Ring Buffer.

**Algorithm:**

1. Check whether the Queue is full or not.
2. Initially the front and rear are set to -1. When we insert the first element in a Queue, front and rear both are set to 0.
3. When we insert a new element, the rear gets incremented, i.e., rear=rear+1.
4. If rear != max - 1, then rear will be incremented to mod(maxsize) and the new value will be inserted at the rear end of the queue.
5. If front != 0 and rear = max - 1, it means that the queue is not full, then set the value of rear to 0 and insert the new element there.
6. When front ==0 && rear = max-1, which means that front is at the first position of the Queue and rear is at the last position of the Queue.
7. front== rear + 1;
8. Check whether the Queue is empty or not. If the queue is empty, we cannot perform the dequeue operation.
9. When the element is deleted, the value of front gets decremented by 1.
10. If there is only one element left which is to be deleted, then the front and rear are reset to -1.

**Program:**

#include <stdio.h>

# define max 6

**int** queue[max];  // array declaration

**int** front=-1;

**int** rear=-1;

// function to insert an element in a circular queue

**void** enqueue(**int** element)

{

**if**(front==-1 && rear==-1)   // condition to check queue is empty

    {

        front=0;

        rear=0;

        queue[rear]=element;

    }

**else** **if**((rear+1)%max==front)  // condition to check queue is full

    {

        printf("Queue is overflow..");

    }

**else**

    {

        rear=(rear+1)%max;       // rear is incremented

        queue[rear]=element;     // assigning a value to the queue at the rear position.

    }

}

// function to delete the element from the queue

**int** dequeue()

{

**if**((front==-1) && (rear==-1))  // condition to check queue is empty

    {

        printf("\nQueue is underflow..");

    }

**else** **if**(front==rear)

{

   printf("\nThe dequeued element is %d", queue[front]);

   front=-1;

   rear=-1;

}

**else**

{

    printf("\nThe dequeued element is %d", queue[front]);

   front=(front+1)%max;

}

}

// function to display the elements of a queue

**void** display()

{

**int** i=front;

**if**(front==-1 && rear==-1)

    {

        printf("\n Queue is empty..");

    }

**else**

    {

        printf("\nElements in a Queue are :");

**while**(i<=rear)

        {

            printf("%d,", queue[i]);

            i=(i+1)%max;

        }

    }

}

**int** main()

{

**int** choice=1,x;   // variables declaration

**while**(choice<4 && choice!=0)   // while loop

    {

    printf("\n Press 1: Insert an element");

    printf("\nPress 2: Delete an element");

    printf("\nPress 3: Display the element");

    printf("\nEnter your choice");

    scanf("%d", &choice);

**switch**(choice)

    {

**case** 1:

        printf("Enter the element which is to be inserted");

        scanf("%d", &x);

        enqueue(x);

**break**;

**case** 2:

        dequeue();

**break**;

**case** 3:

        display();

    }}

**return** 0;

}

**Output:**

Press 1: Insert an element

Press 2: Delete an element

Press 3: Display the element

Enter your choice

1

Enter the element which is to be inserted

10

Press 1: Insert an element

Press 2: Delete an element

Press 3: Display the element

Enter your choice

1

Enter the element which is to be inserted

20

Press 1: Insert an element

Press 2: Delete an element

Press 3: Display the element

Enter your choice

1

Enter the element which is to be inserted

30

Press 1: Insert an element

Press 2: Delete an element

Press 3: Display the element

Enter your choice

3

Elements in a Queue are :10,20,30,

Press 1: Insert an element

Press 2: Delete an element

Press 3: Display the element

Enter your choice

2

The dequeued element is 10

**Result**

**Thus operation on the Circular queue using an array is performed.**

**REVIEW QUESTIONS:**

1. **What is meant by circular queue**
2. **What is the advantage of using circular queue**
3. **Mention the applications of circular queue**

**Ex. No. 2 Implementation of Singly Linked List**

**Date:**

**Aim**

**To define a singly linked list node and perform operations such as insertions and deletions dynamically.**

**TUTORIAL:**

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Following are the important terms to understand the concept of Linked List.

* + Link − each link of a linked list can store a data called an element.
  + Next − each link of a linked list contains a link to the next link called Next.
  + Linked List − A Linked List contains the connection link to the first link called First.

Linked list can be visualized as a chain of nodes, where every node points to the next node.

* + Linked List contains a link element called first.
  + Each link carries a data field(s) and a link field called next.
  + Each link is linked with its next link using its next link.
  + Last link carries a link as null to mark the end of the list. Types of Linked List

Following are the various types of linked lists.

* + Simple Linked List − Item navigation is forward only.
  + Doubly Linked List − Items can be navigated forward and backward.
  + Circular Linked List − Last item contains a link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

Following are the basic operations supported by a list.

* + Insertion − Adds an element at the beginning of the list.
  + Deletion − Deletes an element at the beginning of the list.
  + Display − Displays the complete list.
  + Search − Searches an element using the given key.
  + Delete − Deletes an element using the given key.

**Algorithm**

* 1. Start
  2. Define single linked list *node* as self referential structure
  3. Create *Head* node with label = -1 and next = NULL using
  4. Display menu on list operation
  5. Accept user choice
  6. If choice = 1 then

Locate node after which insertion is to be done Create a new node and get data part

Insert new node at appropriate position by manipulating address Else if choice = 2

Get node's data to be deleted. Locate the node and delink the node Rearrange the links

Else

Traverse the list from Head node to node which points to null

* 1. Stop

**Program**

/\* Single Linked List \*/

#include <stdio.h> #include <conio.h> #include <process.h> #include <alloc.h> #include <string.h>

struct node

{

int label;

struct node \*next;

};

main()

{

int ch, fou=0; int k;

struct node \*h, \*temp, \*head, \*h1;

/\* Head node construction \*/

head = (struct node\*) malloc(sizeof(struct node)); head->label = -1;

head->next = NULL;

while(-1)

{

clrscr();

printf("\n\n SINGLY LINKED LIST OPERATIONS \n");

printf("1->Add "); printf("2->Delete "); printf("3->View "); printf("4->Exit \n");

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

switch(ch)

{

/\* Add a node at any intermediate location \*/ case 1:

printf("\n Enter label after which to add : "); scanf("%d", &k);

h = head; fou = 0;

if (h->label == k) fou = 1;

while(h->next != NULL)

{

if (h->label == k)

{

fou=1; break;

}

h = h->next;

}

if (h->label == k) fou = 1;

if (fou != 1)

printf("Node not found\n"); else

{

temp=(struct node \*)(malloc(sizeof(struct node))); printf("Enter label for new node : "); scanf("%d", &temp->label);

temp->next = h->next; h->next = temp;

}

break;

/\* Delete any intermediate node \*/ case 2:

printf("Enter label of node to be deleted\n"); scanf("%d", &k);

fou = 0;

h = h1 = head;

while (h->next != NULL)

{

h = h->next;

if (h->label == k)

{

fou = 1; break;

}

}

if (fou == 0)

printf("Sorry Node not found\n"); else

{

while (h1->next != h) h1 = h1->next;

h1->next = h->next; free(h);

printf("Node deleted successfully \n");

}

break;

case 3:

printf("\n\n HEAD -> "); h=head;

while (h->next != NULL)

{

h = h->next;

printf("%d -> ",h->label);

}

printf("NULL"); break;

case 4:

exit(0);

}

}

}

**Output**

SINGLY LINKED LIST OPERATIONS

1->Add 2->Delete 3->View 4->Exit Enter your choice : 1

Enter label after which new node is to be added : -1 Enter label for new node : 23

SINGLY LINKED LIST OPERATIONS

1->Add 2->Delete 3->View 4->Exit Enter your choice : 1

Enter label after which new node is to be added : 23 Enter label for new node : 67

SINGLY LINKED LIST OPERATIONS

1. >Add 2->Delete 3->View 4->Exit Enter your choice : 3

HEAD -> 23 -> 67 -> NULL

**Result**

**Thus operation on single linked list is performed.**

**REVIEW QUESTIONS:**

1. Define singly linked list with neat diagram.
2. Define doubly linked list with neat diagram.
3. Define circularly linked list with neat diagram.
4. Write the difference between singly and doubly linked lists.
5. List three examples that uses linked list.

**Ex. No. 3.a Stack Using Linked List**

**Date:**

**Aim**

To implement stack operations using linked lists.

**TUTORIAL:**

The major problem with the stack implemented using arrays is, it works only for a fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using arrays is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using a linked list can work for an unlimited number of values. That means, a stack implemented using a linked list works for variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked lists can organize as many data values as we want.

In the linked list implementation of a stack, every new element is inserted as a 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its next node in the list. The next field of the first element must be always NULL.

**Algorithm**

* 1. Start
  2. Define a singly linked list node for stack
  3. Create Head node
  4. Display a menu listing stack operations
  5. Accept choice
  6. If choice = 1 then

Create a new node with data Make new node point to first node

Make head node point to new node Else If choice = 2 then

Make temp node point to first node

Make head node point to next of temp node Release memory

Else If choice = 3 then

Display stack elements starting from head node till null

* 1. Stop

**Program**

/\* Stack using Single Linked List \*/

#include <stdio.h>

#include <conio.h> #include <process.h> #include <alloc.h>

struct node

{

int label;

struct node \*next;

};

main()

{

int ch = 0; int k;

struct node \*h, \*temp, \*head;

/\* Head node construction \*/

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

head->next = NULL;

while(1)

{

printf("\n Stack using Linked List \n");

printf("1->Push ");

printf("2->Pop ");

printf("3->View "); printf("4->Exit \n");

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

switch(ch)

{

case 1:

/\* Create a new node \*/

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

printf("Enter label for new node : ");

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

h = head;

temp->next = h->next;

h->next = temp; break;

case 2:

/\* Delink the first node \*/

h = head->next;

head->next = h->next;

printf("Node %s deleted\n", h->label);

free(h);

break;

case 3:

printf("\n HEAD -> ");

h = head;

/\* Loop till last node \*/

while(h->next != NULL)

{

h = h->next;

printf("%d -> ",h->label);

}

printf("NULL \n"); break;

case 4:

exit(0);

}

}

}

**Output**

Stack using Linked List

1->Push 2->Pop 3->View 4->Exit Enter your choice : 1

Enter label for new node : 23 New node added

Stack using Linked List

1->Push 2->Pop 3->View 4->Exit Enter your choice : 1

Enter label for new node : 34

Stack using Linked List

1. >Push 2->Pop 3->View 4->Exit Enter your choice : 3

HEAD -> 34 -> 23 -> NULL

**Result**

**Thus push and pop operations of a stack were demonstrated using a linked list.**

**REVIEW QUESTIONS:**

1. What is a Stack?
2. Mention applications of stack.
3. Define Infix, prefix and postfix notations.
4. Convert the expression ((A + B) \* C – (D – E) ^ (F + G)) to equivalent Postfix notations
5. What are the three ways to represent arithmetic expression?

**Ex. No. 3.b Queue Using Linked List**

**Date:**

**Aim**

To implement queue operations using linked lists.

**TUTORIAL:**

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data(enqueue) and the other is used to remove data(dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues.

* + enqueue () − add (store) an item to the queue.
  + dequeue () − remove (access) an item from the queue.

**Algorithm**

* 1. **Start**
  2. **Define a singly linked list node for queue**
  3. **Create Head node**
  4. **Display a menu listing queue operations**
  5. **Accept choice**
  6. **If choice = 1 then**

Create a new node with data

    If queue is empty, then

      new node is front and rear both

         Add the new node at

      the end of queue and change rear

**Else If choice = 2 then**

**If the queue is empty, return NULL.**

Store previous front and

     move front one node ahead

If front becomes NULL, then

     change rear also as NULL

**Else If choice = 3 then**

**Display queue elements starting from head node till null**

* 1. **Stop**

**Program**

/\* Queue using Single Linked List \*/

#include <stdio.h>

#include <conio.h> #include <process.h> #include <alloc.h>

struct node

{

int label;

struct node \*next;

};

main()

{

int ch=0; int k;

struct node \*h, \*temp, \*head;

/\* Head node construction \*/

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

head->next = NULL;

while(1)

{

printf("\n Queue using Linked List \n");

printf("1->Insert ");

printf("2->Delete "); printf("3->View "); printf("4->Exit \n");

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

switch(ch)

{

case 1:

/\* Create a new node \*/

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

printf("Enter label for new node : ");

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

/\* Reorganize the links \*/ h = head;

while (h->next != NULL) h = h->next;

h->next = temp; temp->next = NULL; break;

case 2:

/\* Delink the first node \*/

h = head->next;

head->next = h->next; printf("Node deleted \n"); free(h);

break;

case 3:

printf("\n\nHEAD -> "); h=head;

while (h->next!=NULL)

{

h = h->next;

printf("%d -> ",h->label);

}

printf("NULL \n"); break;

case 4:

exit(0);

}

}

}

**Output**

Queue using Linked List

1->Insert 2->Delete 3->View 4->Exit

Enter your choice : 1

Enter label for new node : 12

Queue using Linked List

1->Insert 2->Delete 3->View 4->Exit

Enter your choice : 1

Enter label for new node : 23

Queue using Linked List

1. >Insert 2->Delete 3->View 4->Exit

Enter your choice : 3

HEAD -> 12 -> 23 -> NULL

**Result**

**Thus insert and delete operations of a queue were demonstrated using a linked list.**

**REVIEW QUESTIONS:**

1. What is meant by Dequeue? Why is it used?
2. List out queue ADT Operations.
3. What is the use of front and rear end in Queue ADT?
4. What are the advantages of Circular Queue?
5. List out the application of Queue.

**Ex. No. 4 Implementation of Polynomial Manipulation using Linked List**

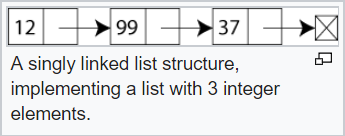
**Date:**

**Aim**

To add any two given polynomials using linked lists.

**TUTORIAL:**

A list or sequence is an abstract data type that represents a countable number of ordered values, where the same value may occur more than once. An instance of a list is a computer representation of the mathematical concept of a finite sequence.

Lists are typically implemented either as linked lists (either singly or doubly linked) or as arrays. As the name implies, lists can be used to store a list of elements. However, unlike in traditional arrays, lists can expand and shrink, and are stored dynamically in memory.Lists also form the basis for other abstract data types including the queue, the stack, and their variations.

**Algorithm**

1. Create a structure for polynomials with exp and coeff terms.
2. Read the coefficient and exponent of two polynomials p and q.
3. While p and q are not null, repeat step 4.

If powers of the two terms are equal then

Insert the sum of the terms into the sum Polynomial Advance p and q

Else if the power of the first polynomial> power of second then Insert the term from first polynomial into sum polynomial Advance p

Else

Insert the term from second polynomial into sum polynomial Advance q

1. Copy the remaining terms from the non empty polynomial into the sum polynomial
2. Stop

**Program**

/\* Polynomial Addition \*/

/\* Add two polynomials \*/

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

struct link

{

int coeff; int pow;

struct link \*next;

};

struct link \*poly1=NULL,\*poly2=NULL,\*poly=NULL;

void create(struct link \*node)

{

char ch; do

{

printf("\nEnter coefficient: "); scanf("%d", &node->coeff); printf("Enter exponent: "); scanf("%d", &node->pow);

node->next = (struct link\*)malloc(sizeof(struct link));

node = node->next; node->next = NULL;

printf("\n continue(y/n): "); fflush(stdin);

ch=getch();

} while(ch=='y' || ch=='Y');

}

void show(struct link \*node)

{

while(node->next!=NULL)

{

printf("%dx^%d", node->coeff, node->pow); node=node->next;

if(node->next!=NULL)

printf(" + ");

}

}

void polyadd(struct link \*poly1, struct link \*poly2, struct link \*poly)

{

while(poly1->next && poly2->next)

{

if(poly1->pow > poly2->pow)

{

poly->pow = poly1->pow;

poly->coeff = poly1->coeff;

poly1 = poly1->next;

}

else if(poly1->pow < poly2->pow)

{

poly->pow = poly2->pow;

poly->coeff = poly2->coeff;

poly2 = poly2->next;

}

else

{

poly->pow = poly1->pow;

poly->coeff = poly1->coeff + poly2->coeff;

poly1 = poly1->next;

poly2 = poly2->next;

}

poly->next=(struct link \*)malloc(sizeof(struct link));

poly=poly->next;

poly->next=NULL;

}

while(poly1->next || poly2->next)

{

if(poly1->next)

{

poly->pow = poly1->pow;

poly->coeff = poly1->coeff;

poly1 = poly1->next;

}

if(poly2->next)

{

poly->pow = poly2->pow;

poly->coeff = poly2->coeff;

poly2 = poly2->next;

}

poly->next = (struct link \*)malloc(sizeof(struct link));

poly = poly->next;

poly->next = NULL;

}

}

main()

{

poly1 = (struct link \*)malloc(sizeof(struct link));

poly2 = (struct link \*)malloc(sizeof(struct link));

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

printf("Enter 1st Polynomial:");

create(poly1);

printf("\nEnter 2nd Polynomial:"); create(poly2);

printf("\nPoly1: "); show(poly1); printf("\nPoly2: "); show(poly2);

polyadd(poly1, poly2, poly); printf("\nAdded Polynomial: ");

show(poly);

}

**SAMPLE INPUT & OUTPUT:**

Enter 1st Polynomial:

Enter coefficient: 5

Enter exponent: 2 continue(y/n): y

Enter coefficient: 4

Enter exponent: 1 continue(y/n): y

Enter coefficient: 2

Enter exponent: 0 continue(y/n): n

Enter 2nd Polynomial:

Enter coefficient: 5

Enter exponent: 1 continue(y/n): y

Enter coefficient: 5

Enter exponent: 0 continue(y/n): n

Poly1: 5x^2 + 4x^1 + 2x^0 Poly2: 5x^1 + 5x^0

Added Polynomial: 5x^2 + 9x^1 + 7x^0

**Result**

**Thus the two given polynomials were added using linked lists.**

**REVIEW QUESTIONS:**

1. What is a list ADT?
2. Give some applications of List ADT.
3. What are the operations that can be done on List ADT?
4. What are the two ways in which List ADT can be implemented?
5. Define Abstract Data Type