**Applications of Linked List**

Linked list is used in many fields as it has many applications. It is also used to implementing a stack, queue. Also, it is used to add the polynomial equation. During the static implementation of stack and queue there was memory limitation but with linked list implementation we can add any number of data to our stack or queue as memory is dynamically allocated.

**Stack using Linked List**

Every data in the stack is represented by a node in linked list. In linked stack, nodes are linked to each other not by their physical placement in memory like an array memory location, but by logical linked of memory location using pointer.

**Algorithms for operations in stack**

* **Push operation**

Push (data, head)

1. Start
2. Dynamically allocate memory for new node, Node \*n=new Node
3. Set n->data=data and n->next=NULL
4. If head==NULL then set head=n
5. Else set ptr=head
6. Traverse ptr until ptr->next!=NULL
7. Set ptr->next=n
8. Exit

* **Pop operation**

Pop(head)

1. Start
2. If head==NULL display empty stack and exit
3. Else set ptr=head and preptr=head
4. ptr=ptr->next
5. Traverse the ptr and preptr until ptr->next!=NULL
6. Set preptr->next=NULL
7. Delete ptr
8. Exit

**Queue using Linked List**

With the linked list implementation of queue, the size of the queue can grow and shrink according to the execution of the program. In linked queue, nodes are linked to each other not by their physical placement in memory like an array memory location, but by logical linked of memory location using pointer.

**Algorithms for operations in queue**

* **Enqueue operation**

Enqueue (data, head)

1. Start
2. Dynamically allocate memory for new node, Node \*n=new Node
3. Set n->data=data and n->next=NULL
4. If head==NULL then set head=n
5. Else set ptr=head
6. Traverse ptr until ptr->next!=NULL
7. Set ptr->next=n
8. Exit

* **Dequeue operation**

Dequeue(head)

1. Start
2. If head==NULL display empty queue and exit
3. Else set ptr=head
4. Set head=ptr->next
5. Delete ptr
6. Exit

**Addition of polynomials using Linked List**

While implementing polynomial equation using linked list the node must have three fields in it. One for the coefficient another one for the power and last one for the address of next field. Coefficient field stores the coefficient of the polynomial equation, power field stores the power of the polynomial and the next field point to the next node that is another term of polynomial.

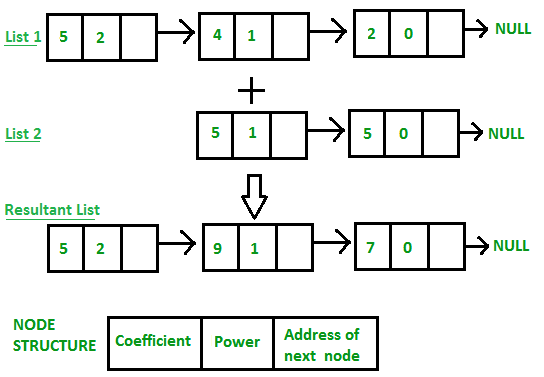


Fig: Addition of polynomial using linked list

**Algorithm for inserting a node in linked list**

Insert (coefficient, power, head)

1. Start
2. Dynamically allocate memory for new node i.e Node \*n=new Node
3. Set n->coefficient=coefficient and n->power=power
4. If head==NULL then set head=n
5. Else set ptr=head
6. Traverse the ptr until ptr->next!= NULL and set ptr->next=n
7. Exit

**Algorithm for addition of polynomial**

add\_polynomial (p, p1, p2)

1. Start
2. Create two Node pointers ptr and ptr1
3. Set ptr=p.head and ptr1=p1.head
4. Repeat step 5 to 7 until ptr1!=NULL and ptr!=NULL
5. If ptr->power>ptr1->power then p2.insert(ptr->coefficient,ptr->power) and set ptr=ptr->next
6. Else if ptr1->power>ptr->power then p2.insert(ptr1->coefficient,ptr1->power) and set ptr1=ptr1->next
7. Else p2.insert(ptr1->coefficient+ptr->coefficient,ptr->power) and set ptr=ptr->next and ptr1=ptr1->next
8. Repeat step 9 to 10 until ptr1!=NULL or ptr!=NULL
9. If ptr1!=NULL then p2.insert(ptr1->coefficient,ptr1->power) and set ptr1=ptr1->next
10. If ptr!=NULL then p2.insert(ptr->coefficient,ptr->power) and set ptr=ptr->next
11. Display p2 which is addition of two polynomials
12. Exit