अध्याय 5

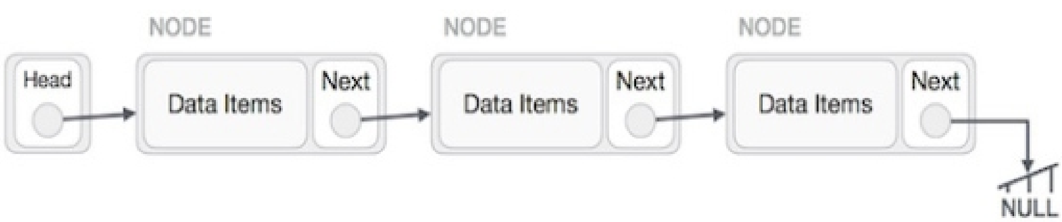
लिंक्ड लिस्ट(LinkedList)

लिंक लिस्ट एक लीनियर डाटा स्ट्रक्चर होता है जिसमे तत्वो की सीरीज इस तरह होती है कि प्रत्येक तत्व अपने अगले तत्व को पॉइंट करता है लिंक लिस्ट में प्रत्येक तत्व को नोड कहते है।आसान भाषा में लिस्ट एक तत्वो कि सीरीज है जिसमे तत्व एक दूसरे से जुड़े हुए है। लिंक लिस्ट ऐरे के बाद सबसे अधिक काम आने वाला डाटा स्ट्रक्चर है लिंक्ड लिस्ट की अवधारणा को समझने के लिए निम्नलिखित महत्वपूर्ण शब्द हैं ।

नोड(Node)– प्रत्येक नोड में डाटा आइटम और अगले नोड का एड्रेस होता है।

नेक्स्ट(Next)– एक पॉइंटर फील्ड होता है जिसमे नेक्स्ट नोड का एड्रेस होता है।

लिंक्ड लिस्ट का प्रेजेंटेशन(Representation):लिंक लिस्ट को नोड्स कि चैन के रूप में प्रदर्शित कर सकते है जहां पर हरेक नोड अगले नोड को पॉइन्ट करता है



लिंक्ड लिस्ट के प्रकार:

लिंक लिस्ट के निम्नलिखित विभिन्न प्रकार हैं।

सिंगल लिंक लिस्ट (Single Linked List) –केवल फॉरवर्ड पॉइंटर होता है

डबल लिंक लिस्ट(Doubly Linked)–फॉरवर्ड और बैकवर्ड पॉइंटर होता है

सर्कुलर लिंक लिस्ट(List Circular Linked List)– अंतिम तत्व पहले तत्व को पॉइन्ट करता है

लिंक लिस्ट के लाभ:निम्नलिखित लिंक लिस्ट के लाभ हैं

(क) लिंक्ड लिस्ट डायनामिक डाटा स्ट्रक्चर है।

(ख) लिंक लिस्ट रन टाइम के दौरान विकसित और सिकुड़ सकती है।

(ग) लिंक लिस्ट में इंसर्शन और डिलेशन आसान होता हैं।

(घ) कुशल स्मृति उपयोग, यानी स्मृति के पूर्व आवंटित कि कोई जरूरत नहीं।

(ङ) एक्सेस टाइम फास्ट होता है मेमोरी ओवरहेड के बिना कॉन्स्टेन्ट टाइम में बढ़ सकती है।

(च) लीनियर डाटा स्ट्रक्चर जैसे स्टैक,क्यू को लिंक लिस्ट कि मदद से आसानी से इम्पलीमेंट कर सकते है।

लिंक लिस्ट के नुकसान:निम्नलिखित लिंक लिस्ट के नुकसान हैं

(क) यदि आवश्यक मेमोरी का पता हो तो मेमोरी वेस्टेज होता है।

(ख) सर्च कर पाना मुश्किल है।

बुनियादी आपरेशन:लिंक लिस्ट के निम्नलिखित बुनियादी आपरेशन है

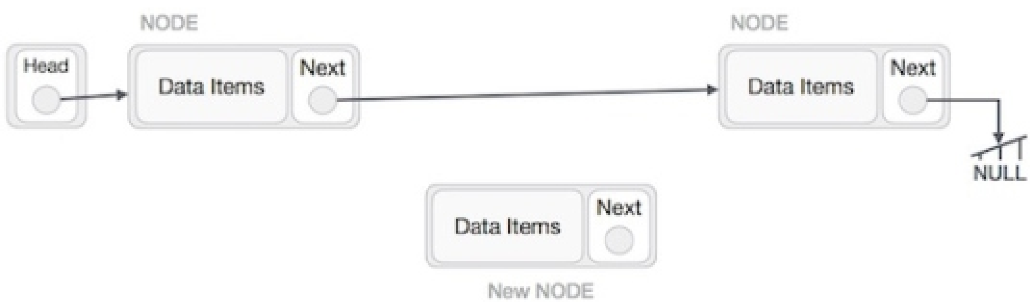
इनर्सशन (Insertion): इनर्सशन का अर्थ एक डाटा स्ट्रक्चर में एक नये डेटा तत्व को जोडना।

डिलिशन (deletion):डिलिशन का अर्थ एक डाटा स्ट्रक्चर में एक डेटा तत्व को हटाना यदि वह मौजूद है।

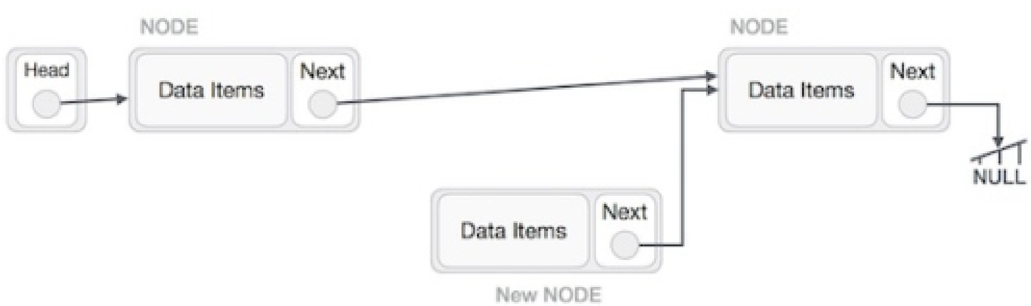
सर्च (Search):एक डाटा स्ट्रक्चर में निर्दिष्ट डेटा तत्व को खोजने को सर्च कहते है।

डिस्प्ले (Display): पूर्ण लिस्ट को डिस्प्ले करता है

इनर्सशन(Insertion) ऑपरेशन:इनर्सशन का अर्थ एक डाटा स्ट्रक्चर में एक नये नोड को जोड़ना है। हम यहाँ निम्नलिखित चित्र कि मदद से समझेंगे। सबसे पहले एक नया नोड बनाइये और इन्सर्ट करने के लिए लोकेशन पता करिये

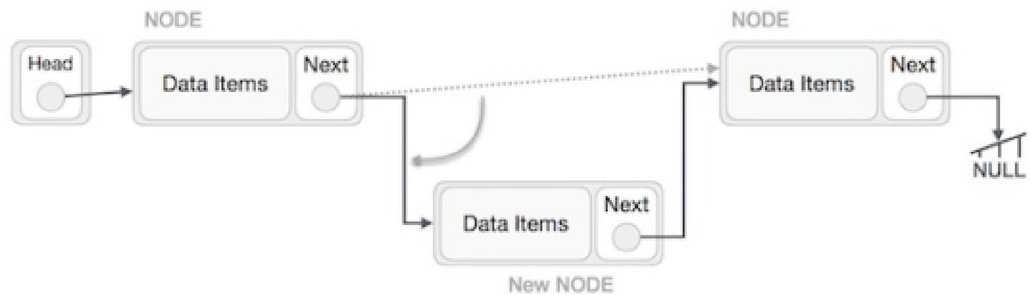


कल्पना कीजिए कि हम एक नोड B(NewNode),A(LeftNode) और C(RightNode) के बीच इन्सर्ट करना चाहते हैं। तब B.next C को पॉइन्ट करेगा और NewNode.next -> RightNode;अब यह इस तरह दिखेगा

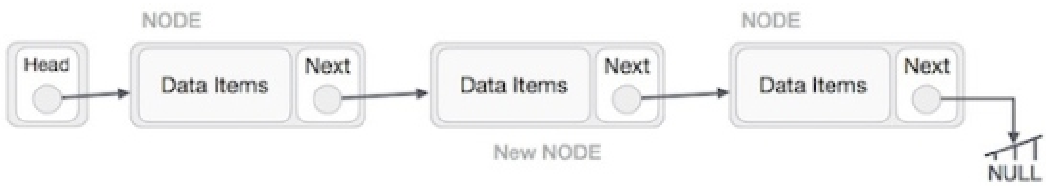


अब लेफ्ट नोड नए नोड को पॉइन्ट करेगा।

LeftNode.next -> NewNode;

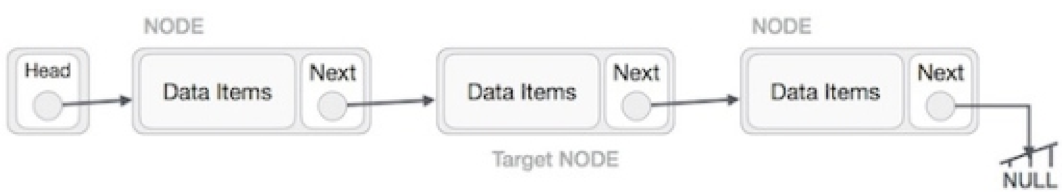


अब दोनों नोड्स के बिच में नए नोड को इन्सर्ट कर देंगे फिर नयी लिस्ट इस प्रकार होगी–

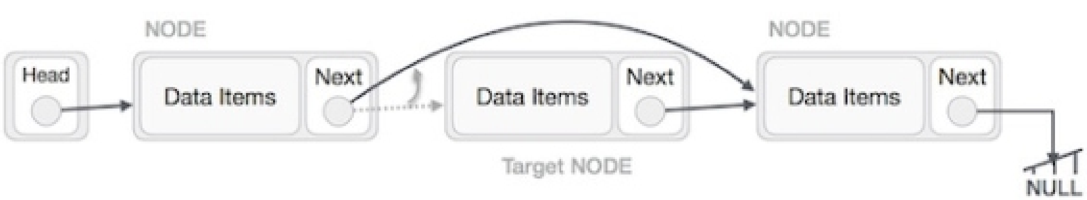


यदि नोड लिस्ट के शुरू में इन्सर्ट करना हो तो समान विधि अपनानी होगी और अंत में इन्सर्ट करना हो तो अंतिम नोड नए नोड को पॉइंट करेगा और नया नोड Null को पॉइंट करेगा

डिलिशन (deletion) ऑपरेशन:डिलिशन का अर्थ एक डाटा स्ट्रक्चर में एक डेटा तत्व को हटाना यदि वह मौजूद है। डिलेशन भी एक से अधिक स्टेप्स का प्रोसेस है चित्र की मदद से देखते है कि सबसे पहले सर्चिंग का उपयोग कर डिलीट करने वाले तत्व को खोजते है।



अब टारगेट नोड के पहले वाला नोड उसके बाद वाले नोड को पॉइंट करेगा–LeftNode.next -> TargetNode.next;



अब टारगेट नोड जिस नोड को पॉइंट कर रहा था वो लिंक निम्नलिखित कोड से हट जायेगा।

TargetNode.next –> NULL;

अगर हमें जरुरत है तो डिलीट किये गए नोड को रख सकते है अन्यथा हम मेमोरी को deallocate कर सकते है।

लिंक लिस्ट के ऑपरेशन्स के लिए C प्रोग्राम:

```
#include <stdio.h>

#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
int data;
int key;
struct node *next;
};

struct node *head = NULL;
struct node *current = NULL;

//display the list
void printList() {
struct node *ptr = head;
printf"\n[ ");

//start from the beginning
while(ptr != NULL) {
printf("(%d,%d) ",ptr->key,ptr->data);
ptr = ptr->next;
}

printf(" ]");
}
```

```
//insert link at the first location
void insertFirst(int key, int data) {
//create a link
struct node *link = (struct node*) malloc(sizeof(struct node));

link->key = key;
link->data = data;

//point it to old first node
link->next = head;

//point first to new first node
head = link;
}

//delete first item
struct node* deleteFirst() {

//save reference to first link
struct node *tempLink = head;

//mark next to first link as first
head = head->next;

//return the deleted link
return tempLink;
}

//is list empty
bool isEmpty() {
return head == NULL;
}

int length() {
int length = 0;
struct node *current;

for(current = head; current != NULL; current = current->next) {
```

```
length++;
}

return length;
}

//find a link with given key
struct node* find(int key) {

//start from the first link
struct node* current = head;

//if list is empty
if(head == NULL) {
return NULL;
}

//navigate through list
while(current->key != key) {

//if it is last node
if(current->next == NULL) {
return NULL;
} else {
//go to next link
current = current->next;
}
}

//if data found, return the current Link
return current;
}

//delete a link with given key
struct node* delete(int key) {

//start from the first link
struct node* current = head;
struct node* previous = NULL;
```

```
//if list is empty
if(head == NULL) {
return NULL;
}

//navigate through list
while(current->key != key) {

//if it is last node
if(current->next == NULL) {
return NULL;
} else {
//store reference to current link
previous = current;
//move to next link
current = current->next;
}
}

//found a match, update the link
if(current == head) {
//change first to point to next link
head = head->next;
} else {
//bypass the current link
previous->next = current->next;
}

return current;
}

void sort() {

int i, j, k, tempKey, tempData;
struct node *current;
struct node *next;

int size = length();
```

```
k = size ;

for ( i = 0 ; i < size - 1 ; i++, k-- ) {
current = head;
next = head->next;

for ( j = 1 ; j < k ; j++ ) {

if ( current->data > next->data ) {
tempData = current->data;
current->data = next->data;
next->data = tempData;

tempKey = current->key;
current->key = next->key;
next->key = tempKey;
}

current = current->next;
next = next->next;
}
}
}

void reverse(struct node** head_ref) {
struct node* prev   = NULL;
struct node* current = *head_ref;
struct node* next;

while (current != NULL) {
next  = current->next;
current->next = prev;
prev = current;
current = next;
}

*head_ref = prev;
}
```

```
main() {
insertFirst(1,10);
insertFirst(2,20);
insertFirst(3,30);
insertFirst(4,1);
insertFirst(5,40);
insertFirst(6,56);

printf("Original List: ");

//print list
printList();

while(!isEmpty()) {
struct node *temp = deleteFirst();
printf("\nDeleted value:");
printf("(%d,%d) ",temp->key,temp->data);
}

printf("\nList after deleting all items: ");
printList();
insertFirst(1,10);
insertFirst(2,20);
insertFirst(3,30);
insertFirst(4,1);
insertFirst(5,40);
insertFirst(6,56);

printf("\nRestored List: ");
printList();
printf("\n");

struct node *foundLink = find(4);

if(foundLink != NULL) {
printf("Element found: ");
printf("(%d,%d) ",foundLink->key,foundLink->data);
printf("\n");
} else {
```

```
printf("Element not found.");
}
delete(4);
printf("List after deleting an item: ");
printList();
printf("\n");
foundLink = find(4);
if(foundLink != NULL) {
printf("Element found: ");
printf("(%d,%d) ",foundLink->key,foundLink->data);
printf("\n");
} else {
printf("Element not found.");
}
printf("\n");
sort();
printf("List after sorting the data: ");
printList();
reverse(&head);
printf("\nList after reversing the data: ");
printList();
}
```

If we compile and run the above program, it will produce the following result –

Output

```
Original List:
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10) ]
Deleted value:(6,56)
Deleted value:(5,40)
Deleted value:(4,1)
Deleted value:(3,30)
Deleted value:(2,20)
```

Deleted value:(1,10)
List after deleting all items:
[ ]
Restored List:
[(6,56) (5,40) (4,1) (3,30) (2,20) (1,10)]
Element found: (4,1)
List after deleting an item:
[(6,56) (5,40) (3,30) (2,20) (1,10)]
Element not found.
List after sorting the data:
[(1,10) (2,20) (3,30) (5,40) (6,56)]
List after reversing the data:
[(6,56) (5,40) (3,30) (2,20) (1,10)]

महत्वपूर्ण बिंदु

- लिंक लिस्ट एक लीनियर डाटा स्ट्रक्चर होता है जिसमे तत्वो की सीरीज इस तरह होती है कि प्रत्येक तत्व अपने अगले तत्व को पॉइंट करता है लिंक लिस्ट में प्रत्येक तत्व को नोड कहते है।
- लीनियर डाटा स्ट्रक्चर जैसे स्टैक,क्यू को लिंक लिस्ट कि मदद से आसानी से इम्पलीमेंट कर सकते है।
- लिंक्ड लिस्ट डायनामिक डाटा स्ट्रक्चर है

अभ्यासार्थ प्रश्न

वस्तुनिष्ठ प्रश्न

प्र1 लिंक लिस्ट सबसे उपयुक्त हैं

(अ) डेटा के स्थायी संग्रह के लिए

(ब) लगातार बदल रहे स्ट्रक्चर के आकार और डेटा के लिए

(स) ऊपर की दोनों स्थिति के लिए

(द) उपरोक्त में से कोई नहीं

प्र2 आम तौर पर नोड्स के संग्रह को ____________ कहा जाता है।

(अ) स्टैक (ब) लिंक्ड लिस्ट

(स) स्टैक (द) पोंइन्टर

प्र3 निम्न में से कौन सा लिंक्ड लिस्ट का एक प्रकार नहीं है

(अ) डबल लिंक लिस्टम (ब) सिंगल लिंक्ड लिस्ट

(स) सरक्युलर लिंक्ड लिस्ट (द) हाइब्रिड लिंक्ड लिस्ट

प्र4 लिंक लिस्ट आम तौर पर ___________ स्मृति आवंटन के उदाहरण के रूप में जाना जाता है।

(अ) स्थिर (ब) डायनेमिक

(स) कम्पाईल टाईम (द) इनमे से कोई नहीं

प्र5 एक सरक्युलर लिंक लिस्ट में

(अ) सभी तत्व सिक्वैंशियल तरीके से जुड़े होते हैं।

(ब) इसमे कोई शुरुआत और कोई अंत नहीं होता है।

(स) अवयव पदानुक्रम में व्यवस्थित होते हैं।

(द) सूची के भीतर आगे और पीछे चंक्रमण की अनुमति होती है।

लघुत्तरात्मक प्रश्न

प्र1 लिंक लिस्ट को परिभाषित करें।

प्र2 हैडर लिंक लिस्ट क्या है?

प्र3 ऐरे और लिंक लिस्ट में कौन बेहतर है?

प्र4 सरक्युलर लिंक लिस्ट को परिभाषित करें।

निबंधात्मक प्रश्न

प्र1 डबल लिंक लिस्टको समझाओ।

प्र2 सिंगल और डबल लिंक लिस्ट के बीच अंतर को बताओ।

प्र3 लिंक लिस्ट किस प्रकार की स्मृति आवंटन से जुड़ा हुआ हैं?

प्र4 लिंक लिस्ट का उपयोग समझाओ।

उत्तरमाला

उत्तर 1: ब उत्तर 2: ब उत्तर 3: द

उत्तर 4: ब उत्तर 5: ब