लिंक्ड लिस्ट

पाठ्यपुस्तक के प्रश्न

वस्तुनिष्ठ प्रश्न

प्रश्न 1. 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

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

प्रश्न 2. आमतौर पर नोड्स के संग्रह को ..... कहा जाता है
(अ) स्टैक
(ब) लिंक्ड लिस्ट
(स) क्यू
(द) पॉइन्टर

उत्तर: (ब) लिंक्ड लिस्ट

प्रश्न 3. निम्न में से कौन सा लिंक्ड लिस्ट का एक प्रकार नहीं है
(अ) डबल लिंक लिस्टम,
(ब) सिंगल लिंक्ड लिस्ट
(स) सरक्यूलर लिंक्ड लिस्ट
(द) हाइब्रिड लिंक्ड लिस्ट

उत्तर: (द) हाइब्रिड लिंक्ड लिस्ट

प्रश्न 4. लिंक लिस्ट आमतौर पर ..... स्मृति आवंटन के उदाहरण के रूप में जाना जाता है।
(ब) डायनेमिक
(स) कम्पाईल टाईम
(द) इनमें से कोई नहीं
(अ) स्थिर

उत्तर: (ब) डायनेमिक

प्रश्न 5. एक सरक्युलर लिंक लिस्ट में
(अ) सभी तत्त्व सिक्वेंशियल तरीके से जुड़े होते हैं।
(ब) इसमें 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।

उत्तर: (ब) इसमें कोई शुरुआत और कोई अंत नहीं होता है।

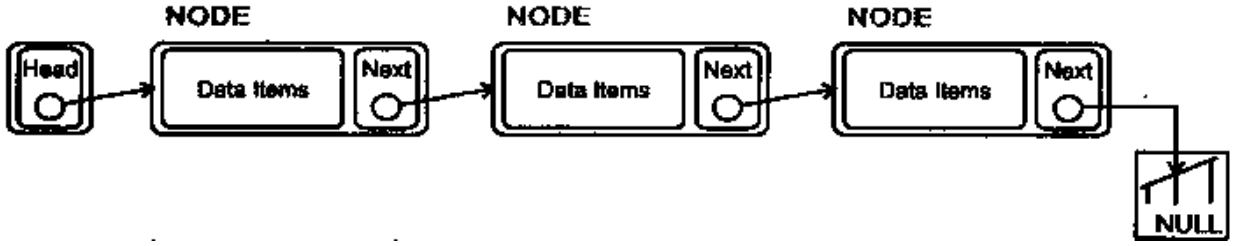
लघु उत्तरीय प्रश्न

प्रश्न 1. लिंक लिस्ट को परिभाषित करें।

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

नोड (Node) – प्रत्येक नोड में डाटा आइटम और अगले नोड का एड्स होता है।
नेक्स्ट (Next) – एक पॉइन्टर फील्ड होता है जिसमें नेक्स्ट का एड्रेस होता है।

लिंक्ड लिस्ट का प्रेजेंटेशन (Presentation of Linked list)-लिंक लिस्ट को नोड्स के चैन के रूप में प्रदर्शित कर सकते हैं जहाँ पर हर एक नोड को पॉइन्ट करती है।

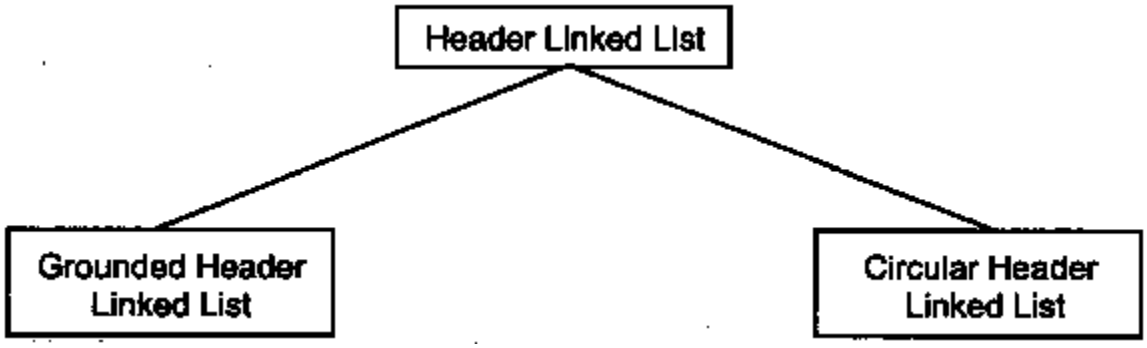


प्रश्न 2. हैडर लिंक लिस्ट क्या है?

उत्तर- हैडर लिंक्ड लिस्ट (Header Linked List)-हैडर लिंक्ड लिस्ट एक ऐसी लिंक्ड लिस्ट है जो हमेशा एक स्पेशल नोड लिस्ट की शुरुआत में रखती है जिसे हैडर नोड (Header Node) कहते हैं।
यह दो प्रकार की होती हैं

- Grounded Header List – इसमें Last Node, NULL Pointer रखता है।

- Circular Header List – इसमें Last Node वापिस, Header Node को पॉइंट (Point) करता है।



प्रश्न 3. ऐरे और लिंक लिस्ट में कौन बेहतर है?

उत्तर- ऐरे में प्रत्येक तत्त्व independent होता है, उसका अपने से पहले तत्त्व से या उसकी location से कोई connection नहीं होंता है। लिंक्ड लिस्ट में तत्त्वों के location या एड्रेस link part में स्टोर रहते हैं।

ऐरे में सारे तत्त्व consecutive manner में मैमोरी में स्टोर रहते हैं। लिंक्ड लिस्ट में तत्त्व किसी भी उपलब्ध जगह में स्टोर हो जाते हैं क्योंकि एक node का एड्रेस पहले वाले. एड्रेस में स्टोर रहता है।

ऐरे single dimensional, double dimensional या multidimensional हो सकता है। लिंक्ड लिस्ट singly, doubly या circular linked list में होते हैं। लिंक्ड लिस्ट में इन्सर्शन और deletion ऑपरेशन fast और easy होते हैं क्योंकि लिंक्ड लिस्ट में केवल पॉइन्टर की value ही change करनी होती है। अतः लिंक लिस्ट, ऐरे से बेहतर होती है।

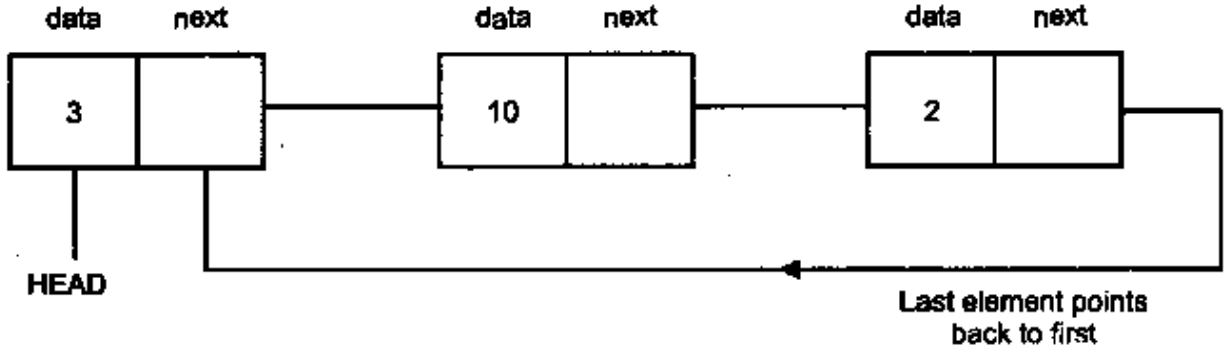
प्रश्न 4. सर्कुलर लिंक लिस्ट को परिभाषित करें।

उत्तर- सर्कुलर लिंक्ड लिस्ट (Circular Linked List)—सामान्य लिंक्ड लिस्ट में अन्तिम नोड का next पॉइन्टर सदैव NULL को दर्शाता है परन्तु सर्कुलर लिंक्ड लिस्ट (Circular Linked List) में लिस्ट की अन्तिम नोड सदैव start को ही दर्शाता है अर्थात् सर्कुलर लिंक्ड लिस्ट एक इस प्रकार की लिस्ट है जिसकी आखिरी नोड सदैव लिस्ट की पहली नोड को दर्शाती है।

यहाँ पर हम जिस सर्कुलर लिंक्ड लिस्ट को समझाने जा रहे हैं, वह सिंगल सर्कुलर लिस्ट (Single Circular List) कहलाती है क्योंकि इस लिस्ट को हम केवल आगे की ओर बढ़ते हुए ही प्रिन्ट करा सकते हैं।

एक विशेष प्रकार की लिस्ट को हम एक अलग तकनीक द्वारा दोनों से अर्थात् आगे से बढ़ते हुए और पीछे से लौटते हुए प्रिन्ट करा सकते हैं, इस प्रकार की लिस्ट को डबली लिंक्ड लिस्ट (Doubly Linked List) कहते हैं।

सामान्य लिस्ट की भांति ही सिंगल सर्कुलर लिस्ट में नई नोड़ जोड़ने, किसी नोड को मिटाने, किसी नोड को खोजने तथा प्रिन्ट कराने के लिए चारों फंक्शन को चलाया जा सकता है।



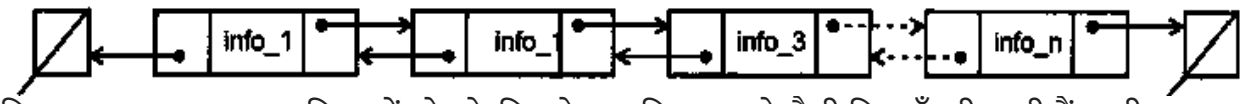
निबंधात्मक प्रश्न

प्रश्न 1. डबल लिंक लिस्ट को समझाओ।

उत्तर- डबल लिंक लिस्ट (Double Link List) डबल लिंक्ड लिस्ट अथवा टू वे लिंक्ड लिस्ट में एक सूचना के दोनों तरफ पॉइन्टर सुरक्षित किए जाते हैं। इनमें से सूचना के बाईं ओर वाला पॉइन्टर सूचना से पहले वाली सूचना को दर्शाता है तथा दाईं ओर वाला पॉइन्टर सूचना अगली सूचना को दर्शाता है। हमने बाईं ओर वाले पॉइन्टर को back तथा दाईं ओर वाले पॉइन्टर को next नाम देकर प्रयुक्त किया है। चूँकि इस लिस्ट में सुरक्षित नोड में एक नहीं बल्कि दो पॉइन्टर हैं, इसलिए इसकी node का structure बाकी सामान्य लिस्ट तथा सर्कुलर लिंक्ड लिस्ट से भिन्न होता है। डबली लिस्ट में प्रयुक्त नोड का structure निम्नलिखित है

```
struct node
{
struct node *back;
int info;
struct node*next;
```

डबली लिंक्ड लिस्ट में दो पॉइन्टर्स की उपस्थिति के कारण हम डबली लिंक्ड लिस्ट में सुरक्षित सूचनाओं को दोनों ओर से प्रिन्ट करा सकते हैं। इसलिए डबली लिंक्ड लिस्ट का प्रयोग साधारणतया उन्हीं स्थानों पर किया जाता है, जहाँ हमें दोनों ओर अर्थात् आगे से और पीछे से सूचनाओं को प्रिन्ट कराना होता है। चूंकि इस लिस्ट में सुरक्षित सूचना को दोनों ओर से प्रिन्ट करा सकते हैं, इसलिए इस लिस्ट को डबली लिंक्ड लिस्ट या टू वे लिंक्ड लिस्ट कहते हैं। डबली लिंक्ड लिस्ट की रूपरेखा निम्न प्रकार से होगी



जिस प्रकार एक साधारण लिस्ट में जोड़ने, मिटाने तथा प्रिन्ट कराने जैसी क्रियाएँ की जाती हैं, उसी प्रकार इस लिस्ट में भी जोड़ना, मिटाना तथा प्रिन्ट कराना होता है।

डबली लिंक्ड लिस्ट में नई नोड को जोड़ना (Adding New Node in Doubly Linked List) – डबली लिंक्ड लिस्ट में भी नोड को लिस्ट में चार स्थानों पर जोड़ा जा सकता है, इसे हम इस प्रकार से भी कह सकते हैं कि इस लिस्ट में भी नोड जोड़ने का कार्य निम्नलिखित चार प्रकार से किया जाता है

- लिस्ट के प्रारम्भ में
- लिस्ट के अन्त में
- लिस्ट में दी गई किसी सूचना के बाद।।
- लिस्ट में वांछित स्थान (Desired Location) के बाद।

प्रश्न 2. सिंगल और डबल लिंक लिस्ट के बीच अन्तर को बताओ।

उत्तर- सिंगल लिंक्ड लिस्ट में हर एक एलीमेंट अगले एलीमेंट का reference रखता है। जबकि डबल लिंक्ड लिस्ट में हर एक एलीमेंट अगले एलीमेंट के साथ पहले वाले एलीमेंट का भी reference रखता है। डबल लिंक्ड लिस्ट को प्रत्येक एलीमेंट के लिए ज्यादा जगह की जरूरत होती है। डबल लिंक्ड लिस्ट में इनसर्शन और डिलीशन जैसे ऑपरेशन ज्यादा कॉम्प्लेक्स होते हैं क्योंकि इन्हें दो references के साथ deal करना होता है।

सिंगल लिंक्ड लिस्ट केवल एक तरफ ही traversal allow करता है जबकि डबल लिंक्ड लिस्ट दोनों तरफ traversal allow करता है।

अगर हमें memory बेचानी है और searching की जरूरत नहीं है तो हम सिंगल लिंक्ड लिस्ट का प्रयोग करते हैं, परन्तु यदि हमें searching में अच्छी performance चाहिए तो हम डबल लिंक्ड लिस्ट का प्रयोग करते हैं।

सिंगल लिंक्ड लिस्ट को ज्यादातर stacks के लिए प्रयोग करते हैं। डबल लिंक्ड लिस्ट को stacks, heaps और binary trees के लिए प्रयोग करते हैं।

प्रश्न 3. लिंक लिस्ट किस प्रकार की स्मृति भावंटन से जुड़ा हुआ है?

उत्तर- लिंक लिस्ट आमतौर पर डायनेमिक स्मृति आवंटन से जुड़ा होता है।

उदाहरण C program to create and display singly linked list

```
#include
#include
struct node
{
int data;
```

```
struct node* next;
}
* head;
void createList(int n) ;
void traverselist();
int main()
{
int n;
printf("Enter the total no. of noder:");
Scanf("%d", &n);
createList(n);
printf("\n Data in the list \n");
traverseList ();
return 0;
}
void createList > (int n)
{
struct node*newNode, *temp;
int data, i;
head = (struct node *) malloc (size of (struct node));
if (head = = NULL)
{
printf("unable to allocate memory.");
}
else
{
printf("Enter the data of node 1 :");
scanf("%d", &data);
head → data = data;
head → next = NULL;
temp = head;
for (i = 2; i<=n; i++)
{
newNode = (struct node *) malloc(sizcof (struct node));
if (newNode = = NULL)
{
printf("Unable to allocate memory")
break;
}
else
```

```
{
printf ("Enter the data of node %d: ", i);
scanf("%d", 4 data);
newNode→data = data;
new node→next = NULL;
temp → next = newNode;
temp = temp→next;
}
}
}
}
Void travere List ()
{
struct node * temp;
if (head==NULL)
{
printf ("List is empty");
}
else
{
temp = head;
while (temp!=NULL)
{
printf("data = %d\n", temp ® data); .
}
}
}
Output :
Enter the total no. of nodes : 5
Enter the data of node 1 : 10
Enter the data of node 2 : 20
Enter the data of node 3 : 30
Enter the data of node 4 : 40
Enter the data of node 5:50
Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50
```

प्रश्न 4. लिंक लिस्ट का उपयोग समझाओ।

उत्तर- लिंक लिस्ट के उपयोग निम्नलिखित उपयोग हैं।

- लिंक लिस्ट डायनामिक डाटा स्ट्रक्चर है।
- लिंक लिस्ट रन टाइम के दौरान विकसित और सिकुड़ सकती है।
- लिंक लिस्ट में इंसर्शन और डिलीशन आसान होता है।
- कुशल स्मृति उपयोग, यानी स्मृति के पूर्व आवंटित की कोई जरूरत नहीं।
- एक्सेस टाम फास्ट होता है, मैमोरी ओवरहेड के बिना कॉन्स्टेन्ट टाइम में बढ़ सकती है।
- लीनियर डाटा स्ट्रक्चर जैसे स्टैक, क्यू को लिंक लिस्ट की मदद से आसानी से इम्पलीमेंट कर सकते हैं।
- लिंक्ड लिस्ट का प्रयोग एक डाटा स्ट्रक्चर के रूप में कई कम्प्यूटर प्राब्लमस को सुलझाने में किया जाता है।
- लिंक्ड लिस्ट का महत्त्वपूर्ण उपयोग निम्न को विकासित करने में हुआ है
  (a) Artificial Intelligence
  (b) Chess Program
  (c) General Problem Solver

अन्य महत्त्वपूर्ण प्रश्न

अतिलघु उत्तरीय प्रश्न

प्रश्न 1. लिंक लिस्ट से आप क्या समझते हैं?

उत्तर- लिंक लिस्ट एक लीनियर डाटा स्ट्रक्चर होता है जिसमें तत्त्वों की सीरीज इस तरह होती है कि प्रत्येक तत्त्व अपने अगले तत्त्व को पॉइन्ट करता है।

प्रश्न 2. नोड़ किसे कहते हैं?

उत्तर- लिंक लिस्ट में प्रत्येक तत्त्व को नोड कहते हैं।

प्रश्न 3. नोड़ में क्या अन्तर है?

उत्तर- प्रत्येक नोड में डाटा आइटम और अगले नोड का एड्रेस होता है।

प्रश्न 4. नेक्स्ट (next) क्या होता है?

उत्तर- नेक्स्ट (Next) एक पॉइन्टर फील्ड होता है जिसमें अगले नोड़ का एड्रेस होता है।

प्रश्न 5. इन्सर्शन (Insertion) का क्या अर्थ है?

उत्तर- इन्सर्शन (Insertion) को अर्थ एक डाटा स्ट्रक्चर में एक नये तत्त्व को जोड़ना होता है।

लघु उत्तरीय प्रश्न

प्रश्न 1. लिंक लिस्ट के नुकसान बताइए।

उत्तर- लिंक लिस्ट के नुकसान: निम्नलिखित लिंक लिस्ट के नुकसान हैं।
(क) यदि आवश्यक मैमोरी का पता हो तो मैमोरी वेस्टेज होता है।
(ख) सर्च कर पाना मुश्किल है।

प्रश्न 2. लिंक लिस्ट के प्रकार बताइए।

उत्तर- लिंक लिस्ट के प्रकारः लिंक लिस्ट के निम्नलिखित विभिन्न प्रकार हैं
सिंगल लिंक लिस्ट (Single Linked List) – केवल फॉरवर्ड पॉइन्टर होता है।
डबल लिंक लिस्ट (Double Linked List) – फॉरवर्ड और बैकवर्ड पॉइंटर होता है।
सर्कुलर लिंक लिस्ट (List Circular Linked List) – अंतिम तत्त्व पहले तत्त्व को पॉइन्ट करता है।

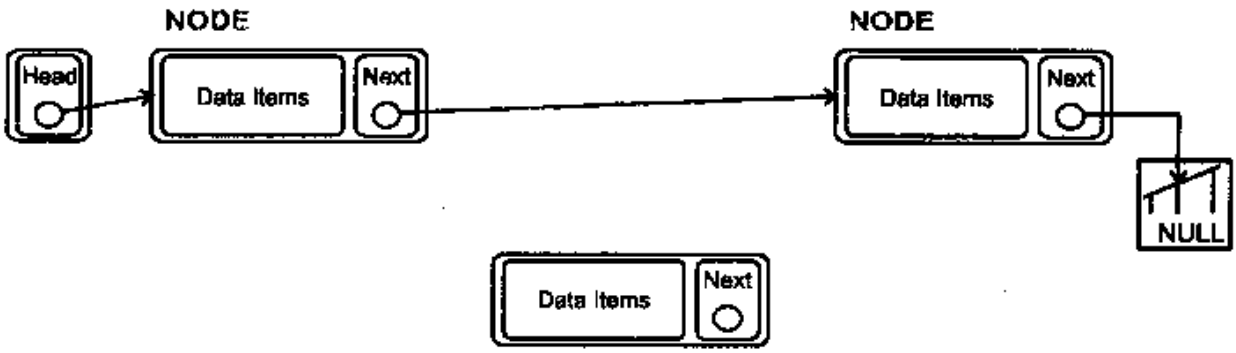
प्रश्न 3. लिंक लिस्ट के प्रमुख बुनियादी आपरेशन के विषय में बताइए।

उत्तर – बुनियादी आपरेशन: लिंक लिस्ट के निम्नलिखित बुनियादी ऑपरेशन है।
इनसर्शन (Insertion) – इनसर्शन का अर्थ एक डाटा स्ट्रक्चर में एक नये डेटा तत्त्व को जोड़ना।
डिलिशन (deletion) – डिलीशन का अर्थ एक डाटा स्ट्रक्चर में एक डेटा तत्त्व हो हटाना यदि यह मौजूद है।
सर्च (Search) – एक डाटा स्ट्रक्चर में निर्दिष्ट डेटा तत्त्व को खोजने को सर्च कहते हैं।
डिस्प्ले (Display) – पूर्ण लिस्ट को डिस्प्ले करता है।

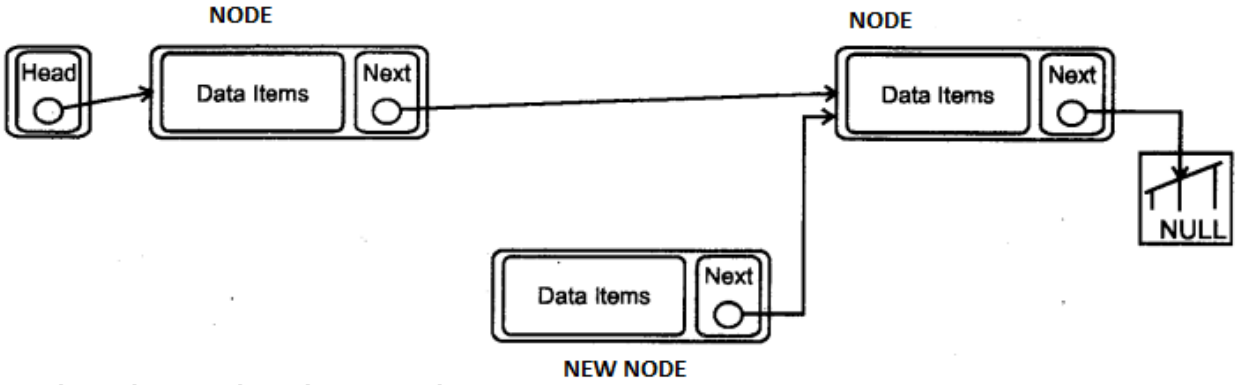
निबंधात्मक प्रश्न

प्रश्न 1. इन्सर्शन ऑपरेशन के विषय में विस्तार से बताइए।

उत्तर- इनसर्शन (Insertion) ऑपरेशन-इनर्सशन का अर्थ एक डाटा स्ट्रक्चर में एक नये नोड को जोड़ना है। हम यहाँ निम्नलिखित चित्र की मदद से समझेंगे। सबसे पहले एक नया नोड बनाते हैं और इन्सर्ट करने के लिए लोकेशन पता करते हैं।

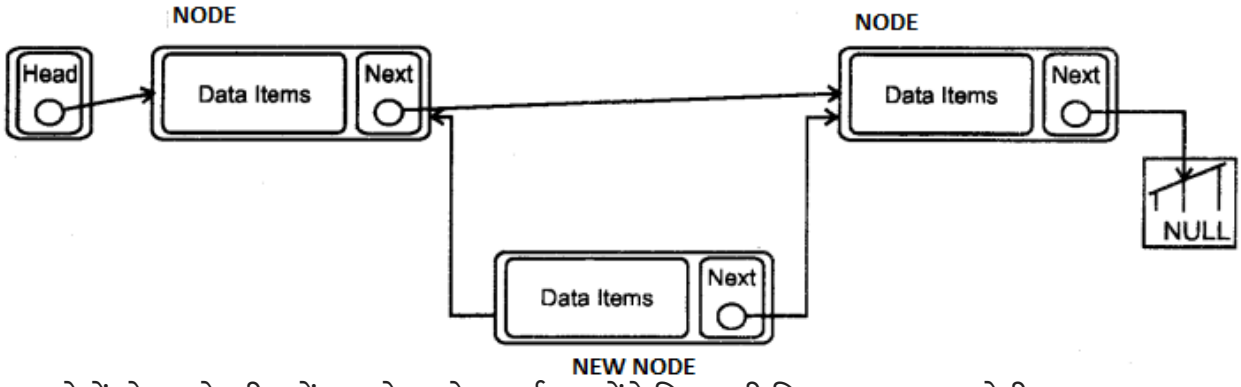


कल्पना कीजिए कि हम एक नोड B(NewNode),A(LeftNode) और C(RightNode) के बीच इन्सर्ट करना चाहते हैं। तब B.next C को पॉइन्ट करेगा और NewNode.next->RightNode: अब यह इस तरह दिखेगा।

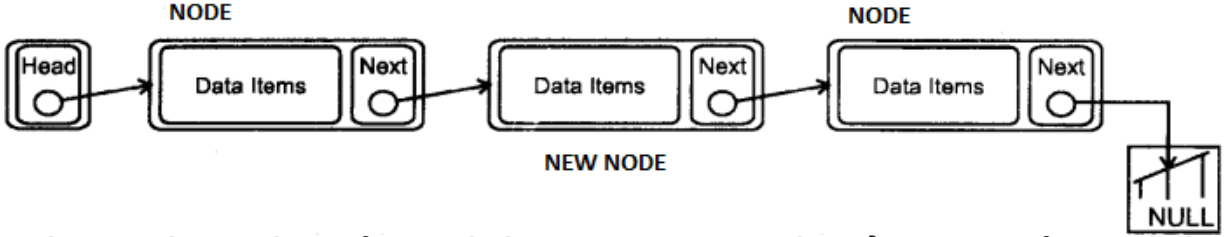


अब लेफ्ट नोड नए नोड को पॉइन्ट करेगा।

Left Node. next -> New Node;



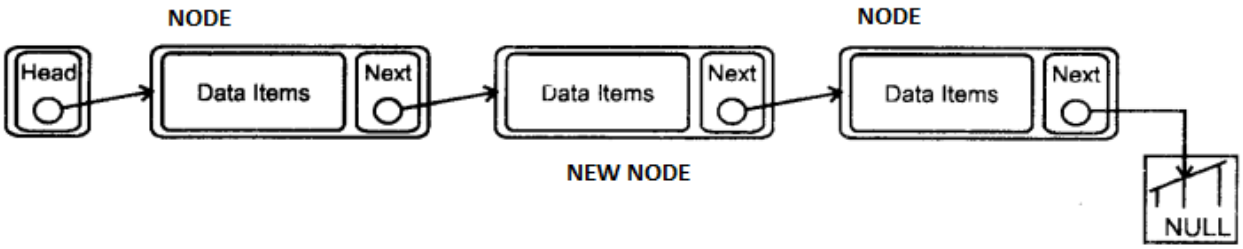
अब दोनों नोड्स के बीच में नए नोड को इन्सर्ट कर देंगे फिर नयी लिस्ट इस प्रकार होगी



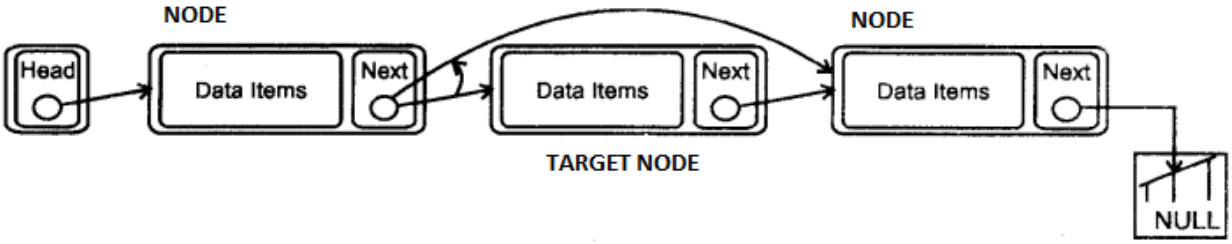
यदि नोड लिस्ट के शुरू में इन्सर्ट करना हो तो समान विधि अपनानी होगी और अंत : इन्सर्ट करना हो तो अंतिम नोड नए नोड को पॉइन्ट करेगा और नया नोड Null को पॉइन्ट करेगा।

प्रश्न 2. डिलीशन ऑपरेशन के विषय में विस्तार से बताइए।

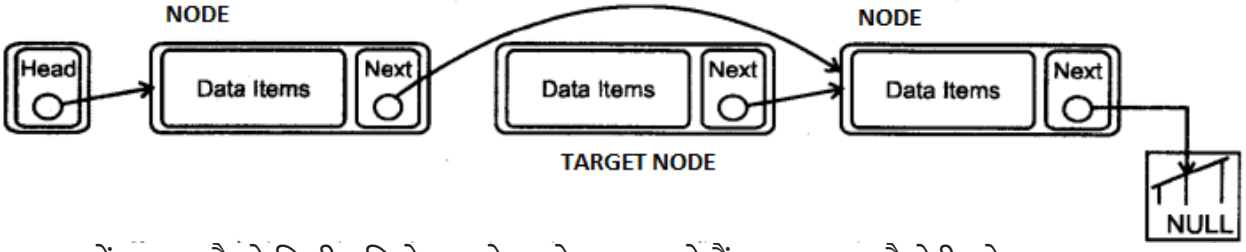
उत्तर- डिलीशन (Deletion) ऑपरेशन-डिलीशन का अर्थ एक डाटा स्ट्रक्चर में एक डेटा तत्त्व को हटाना यदि वह मौजूद है। डिलीशन भी एक से अधिक स्टेप्स का प्रोसेस है चित्र की मदद से देखते हैं कि सबसे पहले सर्चिंग का उपयोग कर डिलीट करने वाले तत्त्व को खोजते हैं।



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



अब टारगेट जिस नोड को पॉइन्ट कर रहा था वो लिंक निम्नलिखित कोड से हट जायेगा।
TargeNode. next -> NULL;



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

प्रश्न 3. लिंक लिस्ट के विभिन्न ऑपरेशन के लिए C प्रोग्राम लिखिए।

उत्तर- लिंक लिस्ट के ऑपरेशन के लिए C प्रोग्राम:

```
#include
#include
#include
#include
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);
}
printf("]");
}
//insert link at the first location
void insert First (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 firt 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;
//is 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, temp Data;
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 ("Orignal List:");
//print list
printlist();
while (!isEmpty())
{
struct node *temp = deleteFirst();
printf("\n Delete value:");
printf("%d,%d)", temp-> key, temp->data);
}
printf("\n List after deleting all items:");
printfList();
insertFirst (1,10);
insertFirst (2,20);
insertFirst (3, 30);
insertFirst (4,1);
insertFirst (5,40);
insert First (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 found Link->data);
printf("\n");
}
else
{
printf ("Element not found.");
}
printf("\n");
sort ();
printf("List after sorting the data:");
prinfList();
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)]
```