جَامعة المَنارة
MANARA UNIVERSITY

كلية الهندسة قسم المعلوماتية

بنى معطيات 1

Data Structure 1

ا. د. علي عمران سليمان

محاضرات الأسبوع السادس والسابع
اللوائح 2
lists 2

الفصل الثاني 2023-2024

محتوى الفصل

اللوائح 2	Lists 2
1- مقدمة	
2- بعض الأشكال الأخرى للو ائح أحادية الارتباط.	
3- التحقيق المتر ابط لكثيرات الحدود.	
4- hash tables	
5- اللو ائح المتر ابطة المضاعفة والصنف القياسي list في C++.	
6- لو ائح أخرى متعددة الارتباط.	

References

- Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)

- د.علي سليمان، بنى معطيات بلغة JAVA، بنى معطيات بلغة C++، بنى معطيات بلغة Pascal جامعة تشرين 2014، 2007، 1998

3- مدخل إلى اللوائح المترابطة (بنية العقد) 1

بنية العقدة:

نذكر بأن العقدة في لائحة مترابطة تتألف من جزئين: جزء بيانات data part يخزن عنصر من اللائحة وجزء مؤشر next part يشير إلى العقدة الحاوية على القيمة التالية لهذه القيمة أو على القيمة null إذا كانت هي آخر عقده في اللائحة.

تمثل هذه العقد كسجل struct أو صنف class واللائحة المترابطة هي مصفوفة منها. كل منها في هذه الحالة سيتضمن عضوين: عضو بيانات يخزن قيمة عنصر اللائحة والعضو التالي الذي يشير إلى التالي من خلال تخزين دليله في المصفوفة.

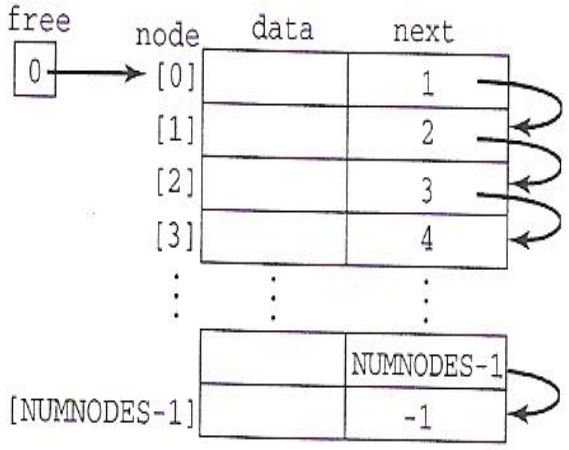
وبالتالي يمكن التصريح عن اللائحة المترابطة كبنية تخزينية باستخدام المصفوفات كما يلي:

```
/*****   Node declaration ******/
struct NodeType
{DataType  data; int next;};
 const int NULL_VALUE=-1;  //a nonexistent location
/**** The Storage Pool ******/
const int NUMNODES=10;          NodeType  node[NUMNODES];
int free;          // points to a free node
```

3- مدخل إلى اللوائح المترابطة (مصفوفات العقد) 1

الآن سنتعرف على بنية مجمع التخزين للعقد المتاحة. إحدى الطرق البسيطة لتنظيم هذا المجمع للعقد الحرة هي كمكدس مترابط linked stack، سيكون محتوى الأجزاء data لهذه العقد غير محدد. الأجزاء next تستخدم ببساطة لربط هذه العقد مع بعضها.

في اللحظة البدائية، جميع العقد تكون متاحة ويجب أن تكون مرتبطة مع بعضها لتشكل مكدس التخزين، إحدى الطرق الطبيعية للقيام بذلك هي جعل العقدة الأولى تشير إلى الثانية والثانية إلى الثالثة وهكذا. وآخر عقدة ستشير إلى null والمؤشر free مساوي للصفر للوصول إلى العقدة الأولى في مجمع التخزين.

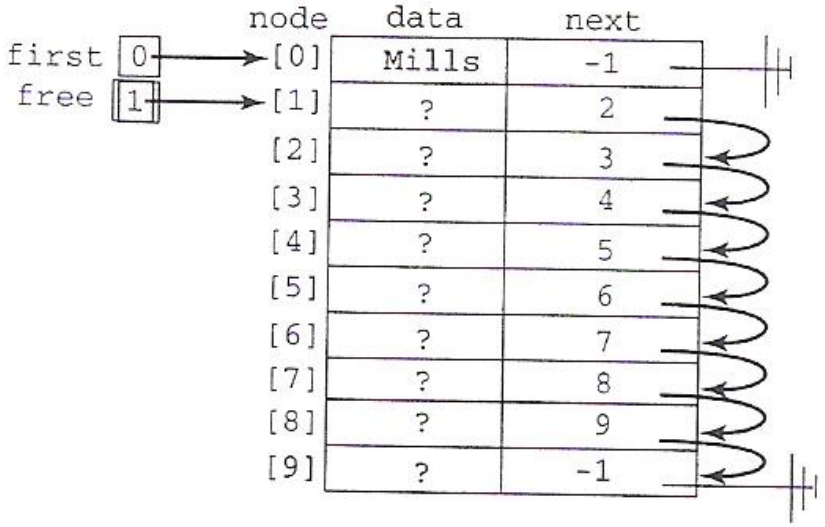



```
//initialize storage pool  each node points to the next one
for (int i=0 ; i<NUMNODES -1; i++)    node[i].next = i+1;
node[NUMNODES – 1].next=NULL_VALUE;  free=0;
```

الاستدعاء ptr=new( ) يعيد موقع عقدة حرة بإسناد العنوان اللذي يشر عليه المؤشر free للمؤشر ptr وحذف العقدة من لائحة المواقع المتاحة بجعل free مساوية لـ node[free].next.

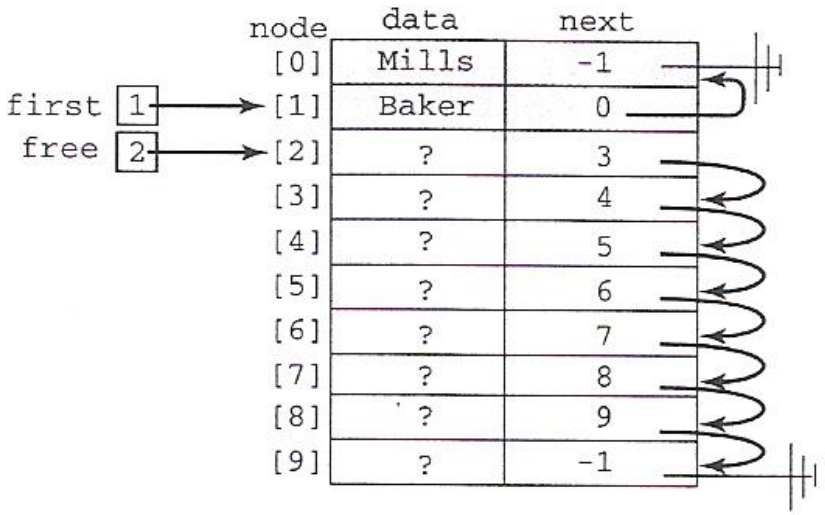
3- مدخل إلى اللوائح المترابطة (مصفوفات العقد) 2

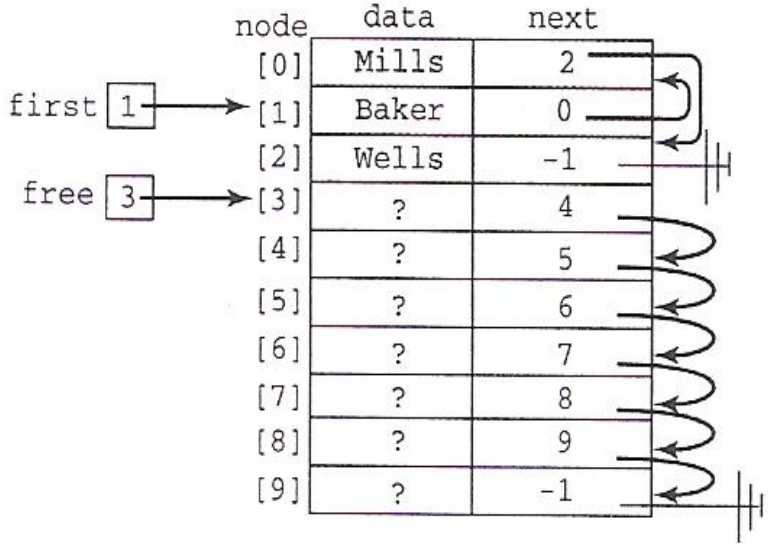
```
//maintain storage pool as a stack
//new operation
Pointer New( ){Pointer p = free ;
        if (free != NULL_VALUE)     free = node[free].next;
        else                        cerr<<"**** Storage pool empty *****\n";
        return p;   }
```



وبالتالي، إذا كان "Mills" هو الاسم الأول المراد حشره إلى لائحة مترابطة، سيخزن في الموقع الأول من المصفوفة node، لأن free تملك القيمة 0 فإن first ستعطى القيمة 0 و free ستصبح مساوية لـ 1.
إذا كان "Baker" هو الاسم التالي المراد حشره، سيخزن في الموقع 1 لأن هذه هي قيمة free. و free ستجعل مساوية لـ 2. إذا كان علينا الحفاظ على هذه اللائحة مرتبة أبجدياً فإن first ستجعل مساوية لـ 1 و node[1].next ستصبح مساوية لـ 0. و node[0].next مساوية لـ -1.

3- مدخل إلى اللوائح المترابطة (مصفوفات العقد) 3



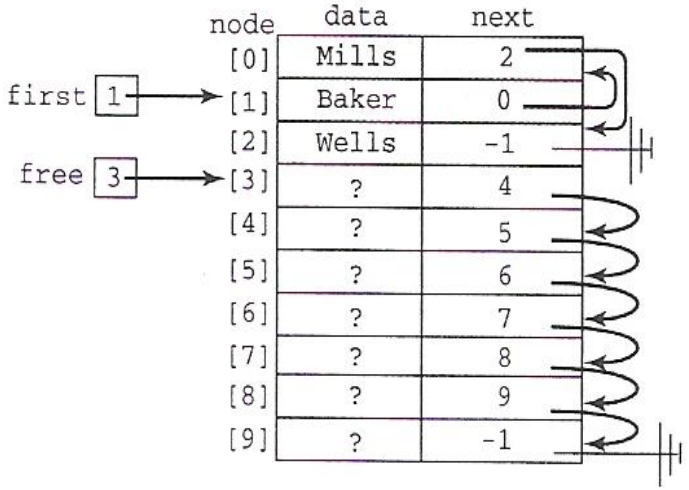


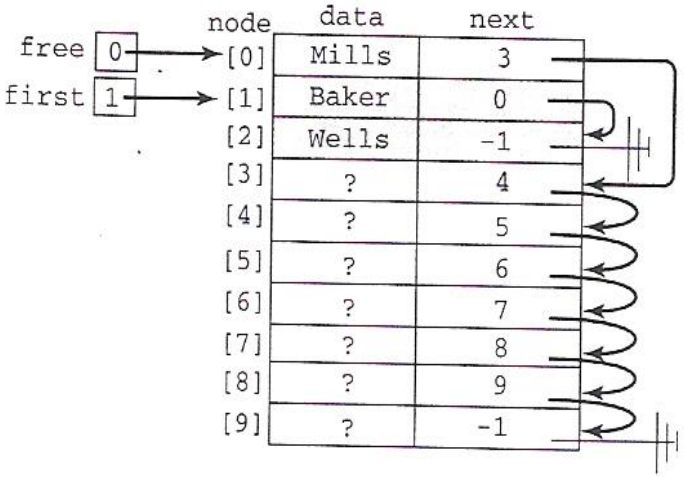
وبنفس الأسلوب لحشر Wells نجدأنها ستكون في اخر اللائحة.

عند حذف عقدة من لائحة مترابطة، يجب أن تتم إعادتها إلى مجمع التخزين للعقد الحرة وبالتالي يمكن أن تتم إعادة استخدامها لاحقاً لتخزين عناصر أخرى. استدعاء التابع Delete(ptr) يقوم بكل بساطة بحشر العقدة المشار إليها بالمؤشر ptr في بداية اللائحة الفارغة بجعل node[ptr].next مساوية لـ free بداية ومن ثم جعل free مساوية لـ ptr:

```
//delete operation
void Delete(Pointer p)
{       node[p].next=free;
        free=p;
}
```

3- مدخل إلى اللوائح المترابطة (مصفوفات العقد) 4





مثالاً: حذف الاسم "Mills" من اللائحة السابقة ينتج الشكل التالي:

هنا يجب أن نشير على العقدة التي تملك Mills بمؤشر ليكن Ptr عندها يجب على العقده التي تسبق العقدة المراد حذفها وهي بحالتنا Baker أن تشير إلي ماتشير هى إليه وفي حالتنا هى Wells وبعدها تتبيعها إلى العقد المتاحة للحجز. أي Ptr ->next= free; And free=Ptr;

يمكن ملاحظة أنه ليس بالضرورة فعلياً لحذف السلسلة "Mills" من جزء البيانات لهذه العقدة لأن تغيير الرابط للعقدة السابقة لها قام بحذفها منطقياً من اللائحة المترابطة ويمكن تخزين اسم جديد عند إعادة استخدامها من جديد.

يمكن تجميع التصريحات والشيفرة السابقين في مكتبة أو صنف أعضاؤه البيانات Node و free بالإضافة إلى تابع باني يعطي قيماً ابتدائية لـ Node و free وتابع ( )New وتابع ( )Delete.

5- تخصيص وإلغاء تخصيص الذاكرة وقت التنفيذ 1

run-time allocation and deallocation 1

يتم تخصيص الذاكرة للمصفوفات compile-time allocation – عند ترجمة البرنامج، هذا يعني أن حجم قطاع الذاكرة المخصص للمصفوفة يبقى ثابتاً خلال تنفيذ البرنامج، ولتغيير سعة المصفوفة نستطيع تغيير قيمة السعة في ملف الشيفرة وإعادة ترجمة البرنامج.

لقد حلت من خلال قالب الصنف vector من مكتبة القوالب القياسية والتي تقوم بتخصيص الذاكرة لغرض من الصنف vector خلال تنفيذ البرنامج (التخصيص وقت التنفيذ run-time allocation).

سنذكر بالآليه التي تقدمها لغة C++ لتخصيص الذاكرة وقت التنفيذ، تخصيص الذاكرة وقت التنفيذ لها عمليتين أساسيتين:

الأولى: الحصول على مو اقع ذاكرة إضافية عند الحاجة من مخزن العقد المتاحة للحجز.
الثانية: تحرير مو اقع الذاكرة عندما لا يعود هناك أي حاجة لها واعادتها إلى مخزن العقد المتاحة للحجز.
تقدم لغة C++ عمليتين مسبقتي التعريف للقيام بذلك وقت التنفيذ هما new لتخصيص الذاكرة و delete لإلغاء التخصيص.

العملية new

الهدف: إرسال طلب وقت التنفيذ للحصول على قطاع ذاكرة كاف لاحتواء غرض من النمط المحدد Type، إذا تمت الاستجابة للطلب تعيد العملية new عنوان بداية قطاع الذاكرة وإلا تعيد العنوان الصفري null.

5- تخصيص وإلغاء تخصيص الذاكرة وقت التنفيذ 1

run-time allocation and deallocation 1

بما أن العملية new تعيد عنوان ذاكرة، وبما أن عناوين الذاكرة يمكن أن تخزن في مؤشرات،

عند تنفيذ التعبير التالي: int *intPtr; intPtr = new int;

التعبير new int يرسل طلباً إلى نظام التشغيل لتخصيص قطاع ذاكرة كاف لاحتواء قيمة صحيحة (أي sizeof(int) بايت) .

إذا كان نظام التشغيل قادراً على الاستجابة للطلب فسيتم إسناد عنوان بداية قطاع الذاكرة إلى intPtr، وإلا فإن كافة مو اقع الذاكرة المتاحة تكون مشغولة وبالتالي سيأخذ intPtr القيمة 0 أو null، وبسبب وجود هذه الإمكانية يجب اختبار العنوان المخصص من خلال التابع new باستخدام التعبير: assert(intPtr !=0);

إذا تم إسناد قيمة غير صفرية لـ intPtr، فإن عنوان موقع الذاكرة المخصص هو مرجع وبالتالي يجب أن يسند لمؤشر.

ويمكن إجراء عدة العمليات عليه نذكر منها:

```
cin>>*intPtr;              //store input value in the new integer
if (*intPtr<100)           //apply relational ops to the new integer
        (*intPtr)++;       //apply arithmetic ops to the new integer
else
        *intPtr=100;       //assign values to the new integer
```

نظرا لتميز اللوائح الديناميكية في عمليات الحشر والحذف في اللائحة في مواقع عشوائية من اللائحة، وتحقيق ذلك بثلاثة أمور:

1. تقسيم الذاكرة إلى عقد، كل منها تتألف من جزء بيانات data part و جزء التالي next part وبناء مؤشر إلى تلك العقد.
2. تعريف عمليات للوصول إلى القيم المخزنة في جزء البيانات و جزء التالي من العقدة المشار إليها بمؤشر ما.
3. المحافظة على ارتباط بين العقد قيد الاستخدام والعقد الحرة المتاحة، وتبادل العقد بين العقد المستخدمة والعقد الحرة.

في التحقيق باستخدام المصفوفات استخدمنا مصفوفة من الأصناف لتحقيق البند (1) وقمنا بإدارة المصفوفة وبالتالي حققنا البندين (2) و (3). نستخدم المؤشرات وميزة التخصيص و إلغاء التخصيص الديناميكي لتقديم تحقيق أفضل للوائح.

بنية العقدة node structure:

إن البنية الأساسية لعقد اللوائح المترابطة ستتضمن حقلين هما data و next. العضو data من نمط مناسب لتخزين عنصر اللائحة، أما العضو next سيخزن رابط للإشارة إلى العضو اللاحق. في حين التصريح الملائم لمثل هذا التحقيق للائحة المترابطة:

```
class Node {   public:          DataType   data;          Node  *next;
        ......
}
```

إن التعريف Node *next; هو تعريف عودي لأنه يستخدم الاسم Node في تعريفه العضو next معرف كمؤشر إلى Node.

6- تحقيق اللوائح المترابطة في لغة C++ باستخدام المؤشرات 2

a pointer-based implementation of linked lists in C++ 2

لسنا مضطرين في هذا التحقيق لتهيئة و المحافظة على مجمع التخزين للعقد الحرة – كما في المصفوفات – فهذا سيتم تلقائياً من قبل مدير الكومة في النظام وباستخدام التوابع المسبقة التعريف في C++ أي new و delete للحصول على العقد أو إعادتها إلى الكومة. للتصريح عن مؤشر إلى العقد نكتب:

```
Node  *ptr;
```

أو

```
typedef Node *NodePointer;
NodePointer   ptr;
```

للحصول على العقدة المشار إليها بـ ptr:

```
ptr=new Node;
ptr=new Node(dataVal);                  //default Node constructor
ptr=new Node(dataVal,linkVal);          //uses Node constructor to set data part  to dataVal and next part to null
//uses Node constructor to set data part  to dataVal and next part to linkVal
```

لإلغاء تخصيص العقدة المشار إليها بالمؤشر ptr:

```
delete ptr;
```

للوصل إلى الجزء data والجزء next من العقدة المشار إليها بالمؤشر ptr:

```
ptr->data    and    ptr->next
```

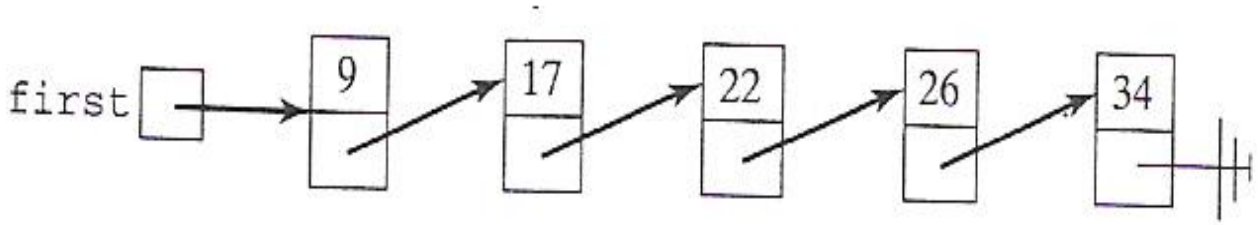
6- تحقيق اللوائح المترابطة في لغة C++ باستخدام المؤشرات3

a pointer-based implementation of linked lists in C++ 3

```
#ifndef  LINKEDLIST
#define LINKEDLIST
template <typename DataType>
class LinkedList
{ / ***** Node class ******/
private:
class Node
{public:          DataType   data;      Node      *next;
        .........
};
typedef Node *NodePointer;
/****** function members *******/
public:           ......
        /****** data members *******/
private:          .....
}
#endif
```

6- تحقيق اللوائح المترابطة في لغة C++ باستخدام المؤشرات4

a pointer-based implementation of linked lists in C++ 4



البيانات الأعضاء لـLinkedList:

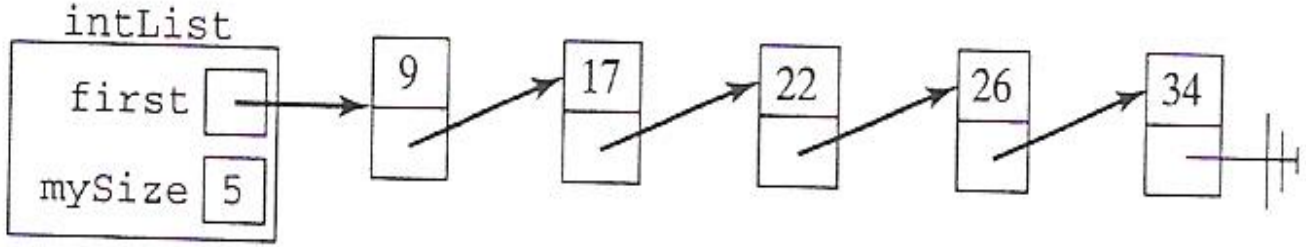
اللوائح المترابطة المبينة في الفقرتين الثانية والثالثة من الشكل:

موصفة بـ:

1. هناك مؤشر إلى العقدة الأولى في اللائحة.
2. كل عقدة تحتوي مؤشراً إلى العقدة التالية في اللائحة.
3. العقدة الأخيرة تتضمن مؤشراً صفرياً.

تعرف هذه اللوائح باسم اللوائح المترابطة البسيطة simple linked lists لتمييزها عن الأشكال الأخرى مثل اللوائح الدائرية، ثنائية الترابط doubly linked واللوائح بعقد رأسية lists with head nodes.

في اللوائح المترابطة البسيطة هناك عضو بيانات وحيد وهو المؤشر إلى العقدة الأولى، ولكن يضاف عادة عضو بيانات آخر يحوي عدد العناصر ضمن اللائحة.



إذا استخدمنا عضو بيانات واحد فقط، عندئذ فإننا كلما احتجنا معرفة طول اللائحة يجب أن نتجول عبر اللائحة ونعد العناصر وفق الخوارزميات المعروفه:

7- قالب الصنف list القياسي 1

يستخدم قالب الصنف list المعرف في مكتبة القوالب القياسية لائحة مترابطة لتخزين العناصر في لائحة. ولكن بنية اللائحة المترابطة أكثر تعقيداً بقليل من اللوائح المترابطة البسيطة ولكن في هذه الفقرة سنتعرف على كيفية استخدام القالب وعملياته الأساسية وأهم مزاياه.

العمليات الأساسية على القالب list:

يبين الجدول التالي التوابع والمعاملات الأعضاء الأكثر

أهمية المعرفة في القالب list:

the standard list class template 1

العملية	الوصف
list<T> l;	بناء l كلائحة فارغة.
list<T> l(n);	بناء l كلائحة مؤلفة من n عنصر.
list<T> l(n , initVal);	بناء l كلائحة تحوي n عنصر كل منها قيمته initVal.
list<T> l(fPtr , lPtr);	بناء لائحة l كلائحة تحوي نسخاً من العناصر الموجودة في مواقع الذاكرة بدءاً من العنوان fPtr وحتى العنوان lPtr.
~list( )	تدمير مكونات وحذف قيم كامل العناصر.
l.empty( )	يعيد true إذا وفقط إذا كانت l لا تحتوي أي قيمة.
l.size( )	يعيد عدد القيم المحتواة في l.
l.push_back(value);	إضافة القيمة value إلى نهاية l.
l.push_front(value);	إضافة القيمة value إلى بداية l.
l.insert(pos,value)	حشر القيمة value في l في الموقع pos.
l.insert(pos,n,value);	حشر n نسخة من القيمة value في l في الموقع pos.
l.insert(pos,fPtr,lPtr);	حشر نسخ من جميع العناصر ضمن المجال [fPtr,lPtr] في الموقع pos.

7- قالب الصنف list القياسي 2

the standard list class template 2

l.pop_back( );	حذف آخر عنصر في اللائحة.
l.pop_front( );	حذف أول عنصر في اللائحة.
l.erase(pos);	حذف القيمة الموجودة في الموقع pos.
l.erase(pos1,pos2);	حذف قيم اللائحة من الموقع pos1 وحتى الموقع pos2.
l.remove(value);	حذف جميع العناصر في اللائحة التي قيمتها value.
l.unique( )	استبدال جميع القيم المتكررة من قيمة ما بقيمة لمرة واحدة فقط.
l.front( )	الإشارة إلى العنصر الأول.
l.back( )	الإشارة إلى العنصر الأخير.
l.begin( )	إعطاء مؤشر متوضع على العنصر الأول في اللائحة.
l.end( )	إعطاء مؤشر متوضع على الموقع التالي للعنصر الأخير في اللائحة.
l.rbegin( )	إعطاء مؤشر عكسي على العنصر الأخير في اللائحة.
l.rend( )	إعطاء مؤشر عكسي على الموقع السابق للعنصر الأول في اللائحة.
l.sort( );	ترتيب العناصر في اللائحة ترتيباً تنازلياً.
l.reverse( );	عكس ترتيب عناصر اللائحة.

l1.merge(l2);	حذف جميع العناصر في l2 ودمجها في l1.
l1.splice(pos,l2);	حذف جميع العناصر في l2 ودمجها في l1 بدءاً من الموقع pos.
l1.splice(to,l2,from);	حذف عناصر اللائحة l2 بدءاً من الموقع from ودمجها في l1 بدءاً من الموقع to.
l1.splice(pos.l2,first,last)	حذف عناصر اللائحة l2 ضمن المجال [first,last] ودمجها في l1 بدءاً من الموقع pos.
l1.swap(l2);	تبديل محتوى اللائحتين l1، l2.
l1=l2	إسناد نسخة من l2 إلى l1.
l1==l2	يعيد true إذا وفقط إذا كانت l1 تحوي نفس العناصر المحتواة في l2 وبنفس الترتيب.
l1<l2	يعيد true إذا وفقط إذا كانت l1 أصغر من l2 بنيوياً.

1- مقدمة الأشكال الأخرى للوائح أحادية الارتباط

1- مقدمة:

إن اللوائح المترابطة القياسية المشروحة سابقاً تتمتع بالصفتين:

- يمكن الوصول بشكل مباشر إلى العقدة الأولى فقط.
- كل عقدة تتألف من جزء بيانات ورابط وحيد يصل هذه العقدة بالعقدة التالية (إن وجدت).

هي عبارة عن بنى خطية linear structures ويجب معالجتها بشكل تتابعي بالترتيب المربوطة فيه العقد، من الأولى وحتى الأخيرة.

بعض الأشكال الأخرى من اللوائح المترابطة،

المترابطة الحلقية، المكدسات والأرتال المترابطة، اللوائح المترابطة الثنائية والمتعددة.

1- بعض الأشكال الأخرى للوائح أحادية الارتباط 0

some variants of singly-linked lists0

نسعى لجعلها خوارزميات بعض العمليات الأساسية أبسط وأكثر فعالية.

المكدسات والأرتال المترابطة linked stacks and queues

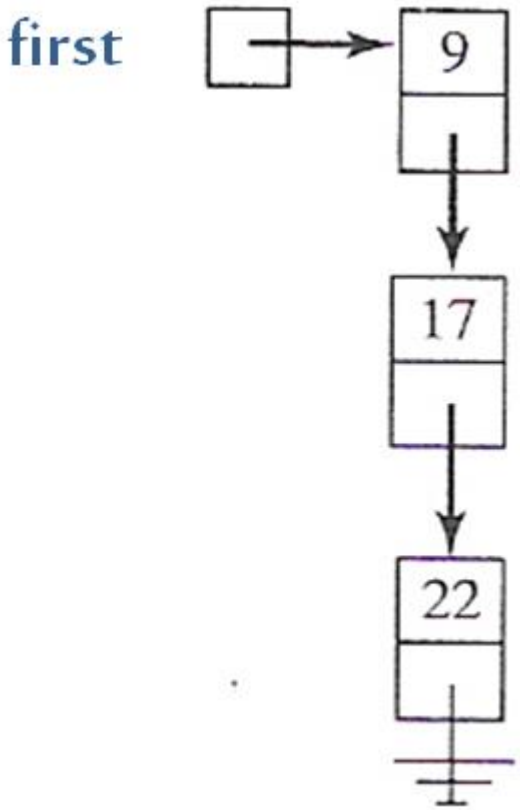
إن أحد أهم نقاط الضعف في تحقيق اللوائح باستخدام المصفوفات كبنية تخزين أساسية هي أن السعة الثابتة للمصفوفة والزمن اللازم لعمليات الازاحة عند الحشر والحذف.

إن استخدام لائحة مترابطة بدلاً من المصفوفه يتيح للمكدس أو الرتل أن يكبر بدون حدود (باستثناء حدود الذاكرة المتاحة) وأن تتقلص بدون خسارة مواقع التخزين غير المستخدمة واختصار الزمن.

سيحتوي الصنف Stack الذي يستخدم لائحة مترابطة لتخزين عناصر المكدس على عضو بيانات واحد فقط وهو المؤشر first للعقدة الموجودة في قمة المكدس.

The stack as single-linked list

المكدس كلائمة متربطة واحدة الاتباط



- في هذا التحقيق، التابع الباني يحتاج فقط لأن يقوم بتهيئة first ليكون مؤشراً صفرياً.
- وأن تختبر العملية empty فيما إذا كان first يساوي العنوان الصفري.
- أما الباني الناسخ ومعامل الإسناد فيجب أن تتجول عبر اللائحة التي تخزن عناصر المكدس لإنشاء نسخة.
- أما الهادم سيحتاج فقط لأن يتجول عبر اللائحة وإعادة العقدة المراد خذفها إلى الكومة.

عملية الإضافة push هي عبارة عن عملية حشر إلى بداية اللائحة المترابطة:

```
Node *iNN=new Node(n, first) {      iNN->data=n;
                                    iNN->next= first;          first=iNN;}
```

حيث يقوم باني الصنف Node بإعطاء الجزء data من العقدة الجديدة القيمة n والجزء next القيمة first ويكون الحصول على العنصر الموجود في أعلى المكدس بديهياً:

```
return first ->data;
```

أما علمية الإخراج pop فهي عملية حذف بسيطة للعقدة الأولى من اللائحة المترابطة:

```
ptr = first;
first = first >next;
delete ptr;
```

Implement the stack as a single-linked list

تنفيذ الكدسة كلائحة أحادية الارتباط 1

```
// link list and display LIFO
#include<iostream>
using namespace std;
class Node { public: Node(): next(NULL){}; int data; Node *next; };

class linklist
{private: Node *first;
public: linklist():first(NULL){};

void additem() {int n,i=1;
      cout<<"enter items and 0 to end:"<<endl;
      while(1){    cout<<i<<" : "; cin>>n;
                   if(n==0)break;
                   i++;
      Node *insertNewNode=new Node;    insertNewNode->data=n;
      insertNewNode->next=first;       first=insertNewNode;}
}
```

Implement the stack as a single-linked list

تنفيذ الكدسة كلائحة أحادية الارتباط 2

```
void display()
      {     cout<<"items are : ";     Node *current=first;
            while(current!=NULL)
                  {cout<<current->data<<"  ";     current=current->next;
                  }     cout<<endl;
      }
};
void main()  {     linklist li;     li.additem();   li.display();
                  system("pause");
            }
enter items and 0 to end:
1 : 4
2 : 77
3 : 99
4 : 0
items are : 99  77  4
Press any key to continue . . .
```

```
// // end link list and add Items sorting insert in orderlist and display
#include<iostream>
using namespace std;
class Node
{ public: Node(): next(NULL){}; int data; Node *next; };
class linklist{
private:        Node *first;
public:         linklist():first(NULL){}
void addItem(){         Node *current;
        while(1)
                {int n;cin>>n;if(n==0)break;
                Node *insertNewNode=new Node ;      insertNewNode->data=n;
                if(first==NULL)
                        {first=insertNewNode; current=insertNewNode;}
                else
                        {current->next=insertNewNode;current=insertNewNode;}
                }
}
```

Adding, arranging, and restricting to a single-linked list2

الاضافة والترتيب والحشر في الائحة أحادية الارتباط 2

```
void sort()
    {bool swa=0;Node *fNN=new Node ;Node *sNN=new Node ;Node *sWB=new Node ;fNN=first;
      while(!swa && fNN != NULL)
       {sNN=fNN->next;
          while (sNN !=NULL){swa=0;
               if(sNN->data<fNN->data)
               {sWB->data = sNN->data;sNN->data=fNN->data;fNN->data=sWB->data; swa=1;}
               sNN=sNN->next;
       }fNN = fNN->next;
       }
     }
void insert(int m)
    {Node *insertNewNode=new Node ; Node *befor,*after=first; insertNewNode->data=m;
       if(insertNewNode->data<first->data)
          {insertNewNode->next=first;first=insertNewNode;}
       else      {while(after!=NULL)
               {if(insertNewNode->data<after->data)break;
               {befor=after;after=after->next; }
}
```

```
if(after!=NULL) {insertNewNode->next=after; befor->next=insertNewNode; }
else   {befor->next=insertNewNode;}
 }
}
void display(){Node *current=first;
   while(current!=NULL) {cout<<current->data<<"  "; current=current->next;} cout<<endl;
}
};
void main() { linklist li;
        int t;char ch;cout<<"enter items and 0 to end:"<<endl;
        li.addItem();          li.display();  li.sort();     li.display();
       while(1)
        {cout<<"enter item to insert :";    cin>>t;
        li.insert(t);  cout<<"items are : "; li.display();
cout<<"press n to finish and anther key to continue: "; cin>>ch; if(ch=='n')break;
}
system("pause");
}// end link list and add Items sorting insert in orderlist and display
```

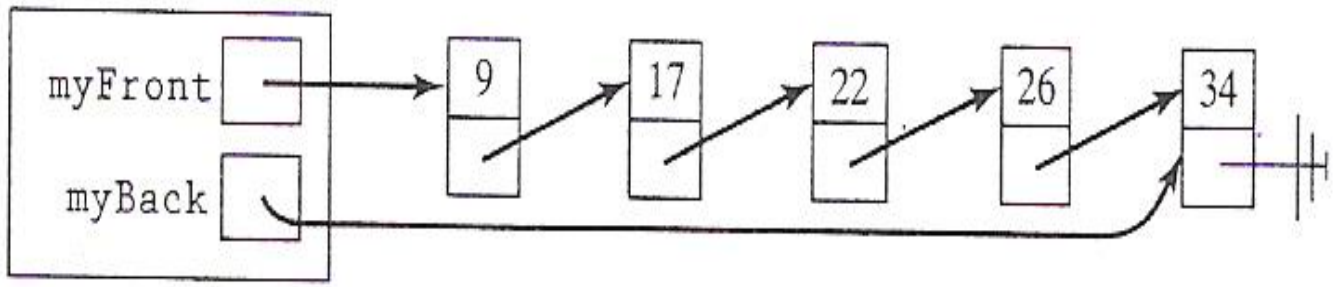
خرج البرنامج

```
enter items and 0 to end:
44
55
66
33
77
22
88
0
44  55  66  33  77  22  88
22  33  44  55  66  77  88
enter item to insert :45
items are : 22  33  44  45  55  66  77  88
press n to finish and anther key to continue: a
enter item to insert :11
items are : 11  22  33  44  45  55  66  77  88
press n to finish and anther key to continue: a
enter item to insert :99
items are : 11  22  33  44  45  55  66  77  88  99
press n to finish and anther key to continue: n
Press any key to continue . . .
```

1- الرتل كلائحه أحادية الارتباط

The Queue as single-linked list

أما التحقيق المترابط للأرتال فهو عبارة عن توسيع بسيط للتحقيق المترابط للمكدسات. حيث يبدو طبيعياً أن نعرف العنصر الأول في اللائحة كمقدمة للرتل، عندئذ يمكن تحقيق عملية الإخراج pop بنفس الطريقة التي حققت فيها عملية الإخراج للمكدس، لكن عملية الإضافة push تتطلب التجول عبر اللائحة بأكملها لإيجاد نهاية الرتل، إن هذا التجول يمكن تجنبه إذا اقتبسنا من التحقيق باستخدام المصفوفات استخدام مؤشرين، myFront يشير إلى العقدة في مقدمة الرتل و myBack يشير إلى العقدة التي في المؤخرة. على سبيل المثال الرتل المترابط الذي يحوي الأعداد الصحيحة 9,17,22,26,34 يمكن تمثيله بالشكل التالي:



وكبديل بسيط يمكن استخدام لائحة مترابطة حلقية ،

نترك أمر إجراء التعديلات على الصنف Queue لكي تستخدم اللوائح المترابطة للتخزين كتمرين للطالب.

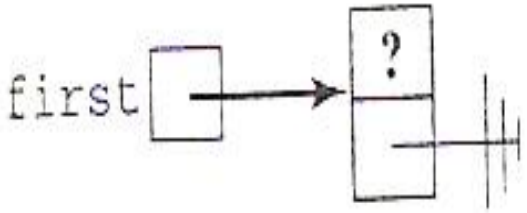
1- بعض الأشكال الأخرى للوائح أحادية الارتباط 1

some variants of singly-linked lists1

اللوائح المترابطة التي تملك عقدة رأس و/أو عقد مقطورة

linked lists with head nodes and/or trailer nodes

إضافة عقدة أولى وهمية dummy first node تدعى العقدة الرأسية head node إلى بداية اللائحة المترابطة. لا يتم تخزين عنصر لائحة فعلي في جزء البيانات من العقدة الرأسية، أي عقدة سابقة للعقدة الأولى التي تخزن العنصر الفعلي الأول في اللائحة لأن حقل الرابط الخاص بها يشير إلى هذه العقدة الأولى الحقيقية.



هذا يعني أنه بدلاً من تهيئة المؤشر first ببساطة ليكون مؤشراً صفرياً، سيكون على باني الصنف LinkedList الذي يستخدم هذا التحقيق أن يحصل على عقدة رأسية مشار إليها بالمؤشر first الذي يملك رابطاً صفرياً. وبشكل مشابه على التابع الذي يقوم باختبار فيما إذا كانت اللائحة فارغة أن يختبر الشرط first->next==0 بدلاً من first==0.

قد يستخدم جزء البيانات للعقدة الرأسية لتخزين بعض المعلومات عن اللائحة.

يمكن أن نخزن حجم اللائحة في العقدة الرأسية، في لائحة بأسماء الأشخاص يمكن أن نخزن اسم الفريق أو المنظمة التي ينتمون إليها:

1- بعض الأشكال الأخرى للوائح أحادية الارتباط 2

some variants of singly-linked lists2

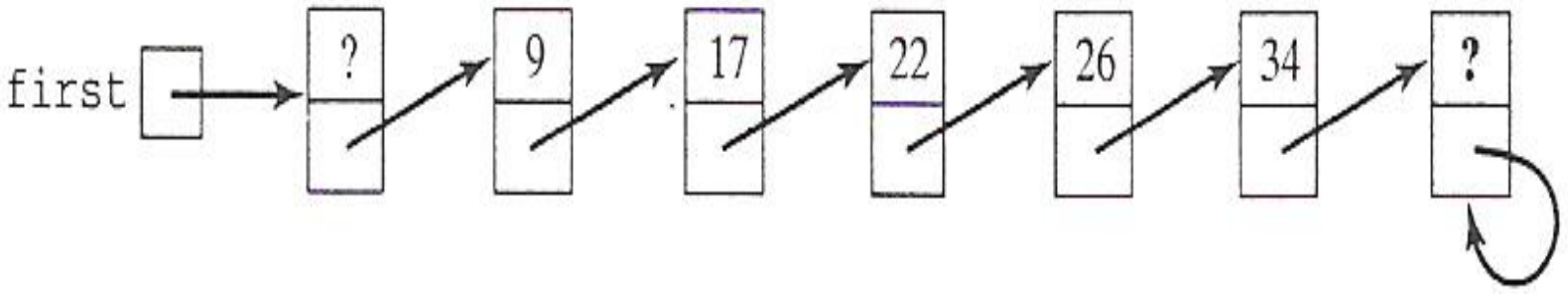
إن حقيقة أن كل عقدة في لائحة مترابطة تملك الآن عقدة سابقة يبسط خوارزميات عمليات الحشر والحذف لأنه لا تطلب لاعتبارات خاصة للعقد التي ليس لها سابق. على سبيل المثال حشر العنصر item بعد العقدة المشار إليها بالمؤشر predptr (الذي قد يكون العقدة الراسية إذا كنا نحشر في بداية اللائحة) يتم ببساطة كما يلي:

```
newptr=new Node(item,predptr->next);
predptr->next=newptr;
```

وعملية الحذف تبسط بنفس الطريقة.

يمكن ببساطة تعديل خوارزميات التجول عبر لائحة مترابطة قياسية لكي تستخدم مع اللوائح المترابطة التي تملك عقدة رأسية، عادة نحتاج فقط إلى تعديل تعليمات التهيئة لمؤشر مساعد إلى العقدة الأولى في اللائحة.

أحياناً، يمكن أن نجد لوائح مترابطة تملك أيضاً عقدة مقطورة trailer node بحيث تكون كل عقدة تملك عقدة تالية successor، مثالاً:



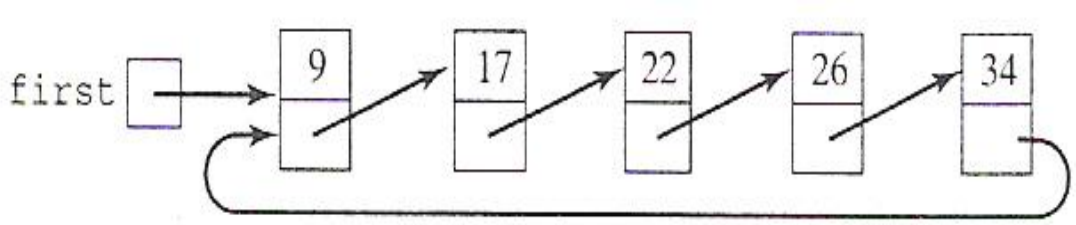
أن مؤشر الجزء التالي للعقدة المقطورة يشير عادة إلى العقدة المقطورة نفسها. ويمكن من أجل فعالية أكبر أن تتشارك اللوائح في التطبيق نفسه نفس العقدة المقطورة.

اللوائح المترابطة الحلقية 1

circular linked lists

اللوائح المترابطة الحلقية circular linked lists

عندما يشير العنصر الأخير إلى العنصر الأول . بجعل الرابط في العقدة الأخيرة يشير إلى العقدة الأولى:



كل عقدة في اللائحة المترابطة الحلقية لها سابق ولاحق (في حال كون اللائحة غير الفارغة). وبالتالي فإن خوارزميات الحذف والحشر – كما في حال اللوائح المترابطة بعقدة رأسية – لا تتطلب اعتبارات خاصة لعدم وجود عقدة سابقة.

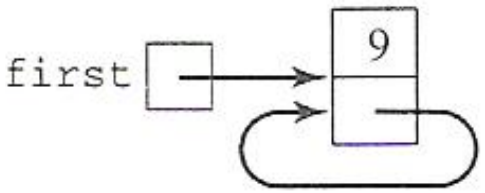
حشر العنصر item كما يلي:

```
newptr=new Node(item,0);
if ( first==0)       //list is empty
{       newptr->next=newptr;       first=newptr;  }
else              //nonempty list
{       newptr->next=predptr->next;          predptr->next=newptr;}
```

لاحظ أن الحشر في لائحة فارغة لا يتطلب اعتبارات خاصة لأن رابط العقدة الوحيدة في هذه الحالة يشير إلى العقدة نفسها:

اللوائح المترابطة الحلقية 2

circular linked lists



بالنسبة للحذف من اللائحة الحلقية، فبالإضافة للائحة الفارغة، فإن اللائحة ذات العنصر الوحيد تتطلب معاملة خاصة، لأن اللائحة في هذه الحالة ستصبح فارغة بعد حذف العقدة. يتم كشف هذه الحالة باختبار فيما إذا كانت العقدة هي السابقة لنفسها، أي حقل الرابط يشير إلى نفسه:

```
if(first==0)                    //list is empty
   cout<<"list is empty \n";
else {  ptr=predptr->next;
        if (ptr==predptr)     //one- node list
                first=0;
        else          //list with 2 or more nodes
                predptr->next=ptr->next; delete ptr;
}
```

إن كلا من خوارزميات عمليتي الحذف والحشر ستبسط إذا استخدمنا لائحة مترابطة حلقية بعقدة رأس كما يلي:

2- التحقيق المترابط لكثيرات الحدود 1

linked implementation of sparse polynomials1

يملك كثير الحدود P(x) بمتحول واحد x الشكل التالي:

$$P(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots\ldots+a_nx^n$$

حيث $a_0 , a_1 , a_2 ,\ldots,a_n$ هي معاملات coefficients كثير الحدود، درجة degree كثير الحدود P(x) هي القوة (الأس exponent) الأعلى لـ x التي تظهر في كثير الحدود بمعامل غير الصفري، على سبيل المثال: $P(x)=5+7x-8x^3+4x^5$

هو من الدرجة 5 ومعاملاته هي 5,7,0,-8,0,4. أما كثير الحدود الثابت Q(x)=3.78 فهو من الدرجة 0 وكذلك كثير الحدود الصفري هو من الدرجة صفر. يمكن النظر إلى كثير الحدود على أنه لائحة من المعاملات: $(a_0 , a_1 , a_2 ,\ldots,a_n)$

ويمكن تمثيله بواسطة تحقيقات اللوائح التي تعرفنا عليها. على سبيل المثال كثير الحدود $P(x)=5+7x-8x^3+4x^5$ يمكن كتابته أيضاً كما يلي: $P(x)=5+7x+0x^2-8x^3+0x^4+4x^5+0x^6+0x^7+0x^8+0x^9+0x^{10}$

والذي يمكن تعريفه بلائحة المعاملات: (5,7,0,-8,0,4,0,0,0,0,0)

ويمكن أن تخزن في مصفوفة P حجمها 11 كما يلي:

I	0	1	2	3	4	5	6	7	8	9	10
P[i]	5	7	0	-8	0	4	0	0	0	0	0

إذا كان الفرق بين القوة العليا والقوة الدنيا في كثير الحدود صغيراً وكانت القوى أصغر من السقف الأعلى المفروض وفق حجم المصفوفة وليس فيه معاملات صفرية كثيرة فإن التحقيق السابق قد يكون ناجحاً تماماً، وغير ذلك يصبح غير فعالاً، فمثلاً لتخزين كثير الحدود التالي:

$$Q(x)=5+x^{99}$$

أو بشكل مكافئ:

$$Q(x)=5+0x+0x^2+0x^3+.....+0x^{98}+1x^{99}$$

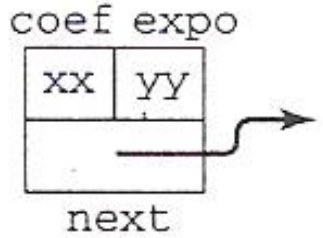
والذي يتطلب مصفوفة بعنصرين غير صفريين، وعلى 98 عنصر صفري فإنه يمكن الحد من الهدر الواضح للذاكرة الناتج عن تخزين كل المعاملات الصفرية وذلك بتخزين المعاملات غير الصفرية فقط.في مثل هذا التحقيق من الواضح أنه من غير الضروري كثير الحدود بلائحة معاملاته، حيث نستطيع تمثيله كلائحة عناصرها عبارة عن أزواج معاملات، على سبيل المثال:

$$P(x)=5+7x-8x^3+4x^5\leftrightarrow((5,0),(7,1),(-8,3),(4,5))$$

لاحظ أن الأزواج مرتبة وفق ترتيب القوى المتزايدة. إن مثل هذه اللوائح يمكن تحقيقها بواسطة مصفوفات السجلات أو الاصناف، كل منها يحتوي حقل المعامل وحقل القوة (الأس)،

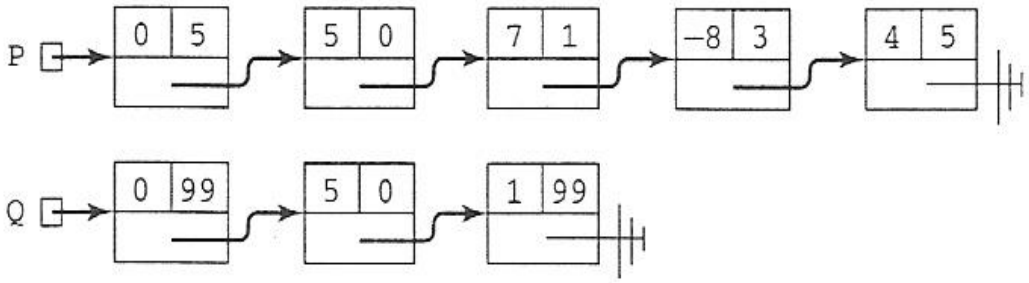
2- التحقيق المترابط لكثيرات الحدود 2

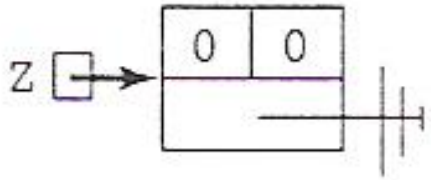
linked implementation of sparse polynomials 2



في هذه التطبيقات، فإن التحقيق باستخدام اللائحة المترابطة مناسب أكثر، كل عقدة تأخذ الشكل:

وفيها تخزن الأجزاء الثلاثة coef,expo,next معاملات غير الصفرية، الأس المقابل،
ومؤشر إلى العقدة التالية الممثلة للحد التالي. على سبيل المثال كثيري الحدود السابقين P(x) و Q(x) يمكن تمثيلهما باللائحتين المترابطتين اللتين تملكان عقدة راسية تخزن درجة كثير الحدود في الحقل expo:





أما كثير الحدود الصفري يمكن تمثيله ببساطة بعقدة رأس كما يلي:

نستطيع الآن البدء بتعريف قالب الصنف Polynomial لمثل هذه الكثيرات الحدود المترابطة:

linked implementation of sparse polynomials 3

```
#ifndef POLYNOMIAL
#define POLYNOMIAL
template <typename CoefType>
class Polynomial          {/**** Node structure ****/
private: class PolyNode
        {       public:         CoefType coef;   int   expo;       PolyNode *next;
                PolyNode(CoefType co=0,int ex=0,PolyNode *ptr=0)
                        {       coef=co;        expo=ex;        next=ptr;       }
   };
typedef PolyNode  *PolyPointer;n   /**** function members ****/
public:         .....
/**** data members ****/
private:        PolyPointer  myFirst;       int  myDegree;     ........
};
#endif
```

2- التحقيق المترابط لكثيرات الحدود 4

linked implementation of sparse polynomials 4

بالتأكيد، كان بإمكاننا استخدام الشكل القياسي العقدة المؤلفة من جزء data وجزء next بجعل نمط الجزء data عبارة عن سجل مؤلف من جزء coef وجزء expo لكن ذلك لا يحقق فوائد حقيقية إذ قد يتطلب اختيار مضاعف للعضو للوصول إلى العناصر أي P->data.coef بدلاً من الكتابة ببساطة P->coef.

لتوضيح كيف تتم معالجة مثل هذه اللوائح المترابطة من كثيرات الحدود، سندرس عملية جمع كثيرات الحدود. على سبيل المثال لجمع كثيري الحدود A(x) و B(x) التاليين:

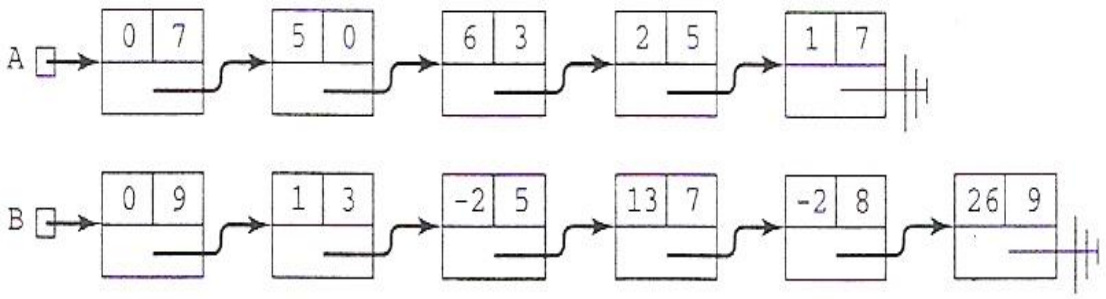
$A(x)=5+6x^3+2x^5+x^7$

$B(x)=x^3-2x^5+13x^7+2x^8+26x^9$

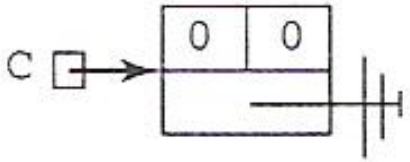
إن عملية الجمع تتم بجمع معاملات الحدود المتشابهة في درجة x وبالتالي فإن ناتج جمع كثيري الحدود A(x) و B(x) هو:

$C(x)=A(x)+B(x)=5+7x^3+14x^7-2x^8+26x^9$

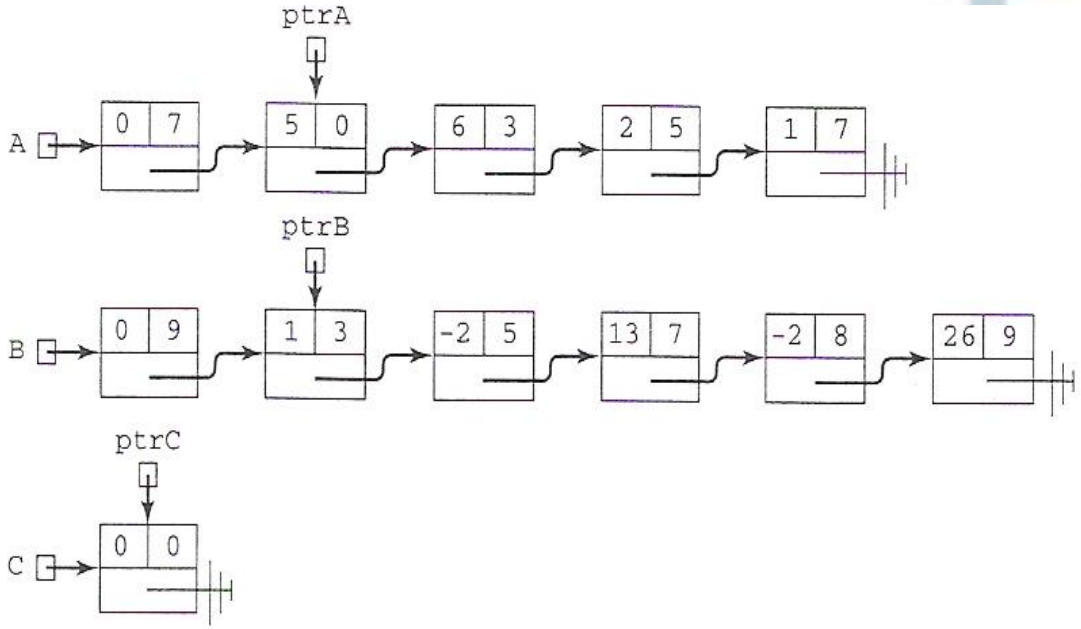
يمكن توضيح التمثيل المترابط لكثيري الحدود كما في الشكل:



عند استخدام لوائح مترابطة بعقدة رأس، سنهيئ C ليشير إلى عقدة رأس:



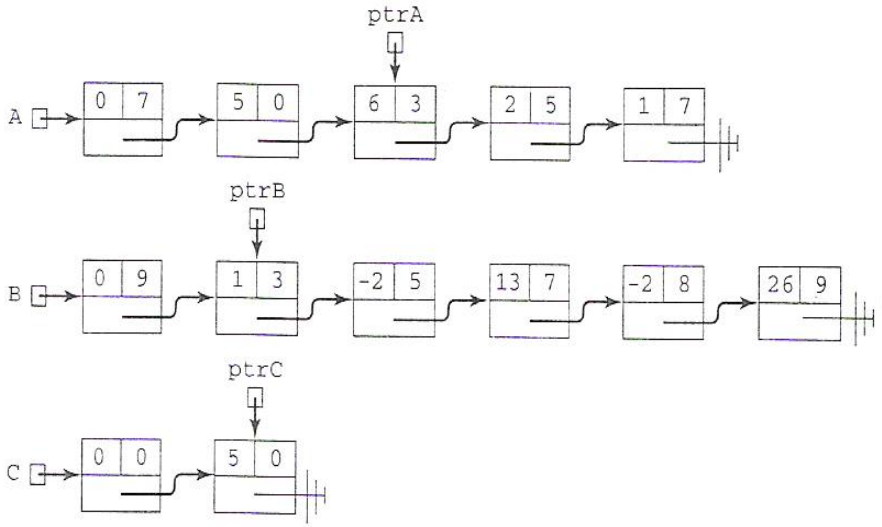
المؤشرات المساعدة ptrA,ptrB,ptrC ستتجول عبر اللوائح A,B,C على الترتيب، سيشير المؤشران ptrA,ptrB إلى العقد الحالية التي تتم معالجتها أما المؤشر ptrC فسيشير إلى آخر عقدة مرتبطة بـ C. وبالتالي فإن ptrA,ptrB,ptrC سيتم إعطاؤها القيم الابتدائية التالية A->next، B->next و C على التوالي:



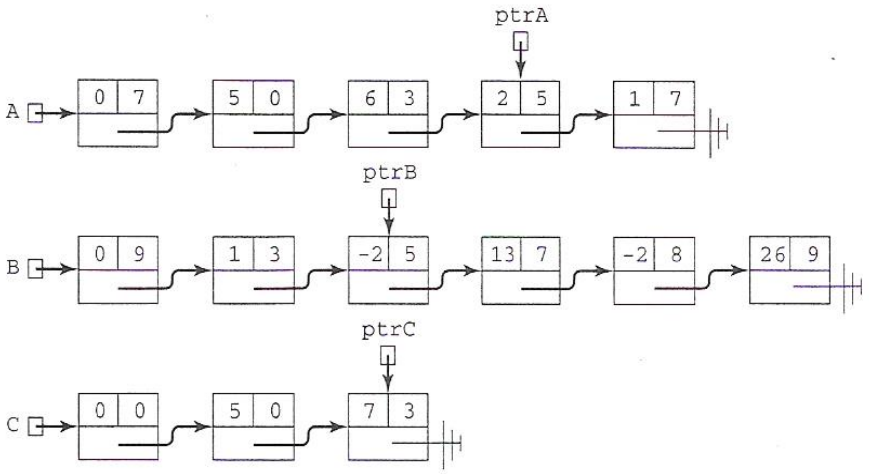
في كل خطوة، تتم مقارنة قوى العقد المشار إليها بـ ptrA و ptrB فإن كانتا مختلفتين فإن العقدة التي تحوي القوة الأصغر والمعامل المقابل يتم ربطها باللائحة C ويتم تقديم المؤشر ptrCوالمؤشر إلى اللائحة:

2- التحقيق المترابط لكثيرات الحدود 6

linked implementation of sparse polynomials 6



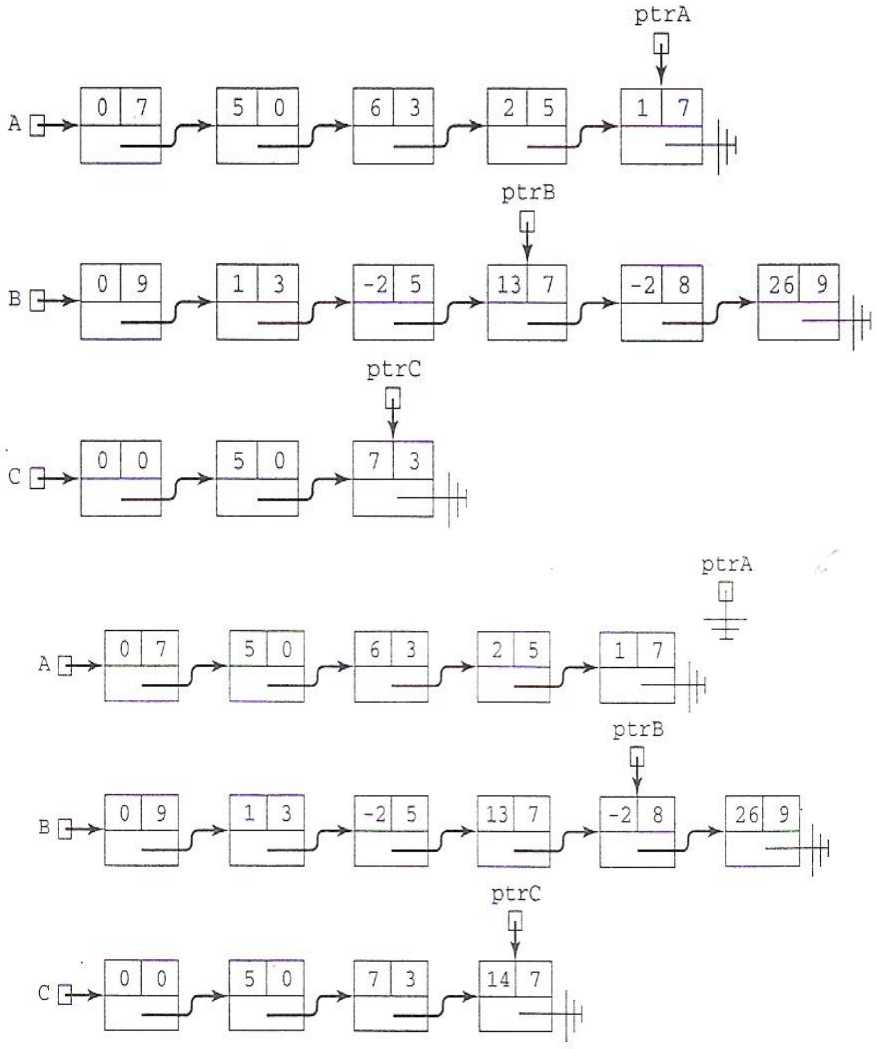
أما إذا كانت القوى المشار إليها بالمؤشرين ptrA، ptrB متساويتين فإن المعاملات في العقدتين يتم جمعهما فإن كان المجموع غير الصفري يتم إنشاء عقدة جديدة فيها جزء المعامل مساوي إلى هذا المجموع وجزء القوة مساوي للقوة المشتركة ويتم ربط هذه العقدة بـ C ومن ثم تقديم المؤشرات في اللوائح الثلاث:



أما إذا كان المجموع صفراً فإن ptrA وptrB يتم تقديمهما ولا تضاف أي عقدة إلى C:

2- التحقيق المترابط لكثيرات الحدود 7

linked implementation of sparse polynomials 7

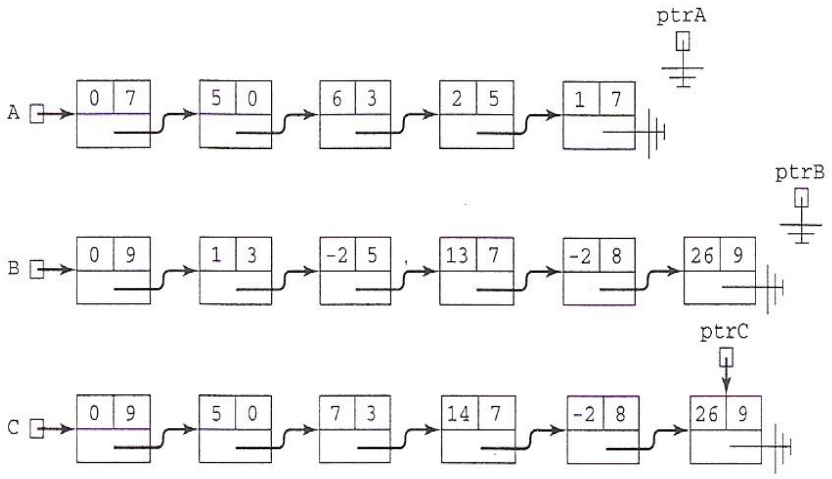


ونتابع بهذه الطريقة لحين الوصول إلى نهاية A أو B، أي إلى أن يشير أحد المؤشرين ptrA أو ptrB إلى العنوان الصفري null:

إذا تم الوصول إلى نهاية إحدى اللائحتين ولم يتم الوصول إلى نهاية الأخرى، نقوم ببساطة بنسخ العقد المتبقية فيها وربطها إلى C. بعدئذ ننهي بناء اللائحة المترابطة C التي تمثل المجموع A(x)+B(x) بجعل الجزء Next من العقدة الأخيرة لـ C يشير إلى العنوان الصفري null و الحقل expo في العقدة الرأسية لـ C يأخذ قوة آخر عقدة:

2- التحقيق المترابط لكثيرات الحدود 8

linked implementation of sparse polynomials 8



تبين الشيفرة التالية تحقيق هذه الخوارزمية لجمع كثيرات الحدود ويمكن إضافتها إلى قالب الصنف Polynomial كتابع صديق:

```
// add polynomial   in list

Node *ptra=firsta, *ptrb=firstb, *ptrc=firstc;

while ( ptra !=NULL && ptrb !=NULL)
{   if( ptra->expo < ptrb-> expo)     { Node *INN= new Node;  INN->data = ptra->data; INN->expo = ptra-> expo;
        if ( ptrc==NULL)          { INN->next = firstc; firstc=INN; ptrc=INN;ptra=ptra->next;}
        else { ptrc->next = INN;  ptrc=INN; ptra=ptra->next;}          }          // end   if( ptra->expo < ptrb-> expo)
```

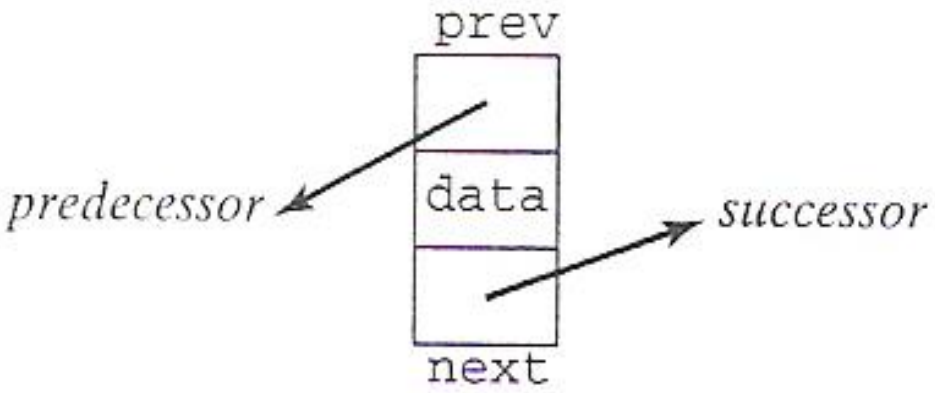
```
else if( ptrb-> expo <ptra-> expo)     { Node *INN= new Node;  INN->data = ptrb->data;INN->expo = ptrb-> expo;
        { if ( ptrc==NULL)  { INN->next = firstc; firstc=INN; ptrc=INN;ptrb=ptrb->next;}
         else { ptrc->next = INN;  ptrc=INN; ptrb=ptrb->next;}    }       // end if( ptrb-> expo <ptra-> expo)

        int m = ptrb->data+ ptra->data;
         if (m != 0 )      { Node *INN= new Node;  INN->data = m; INN->expo = ptrb-> expo;
         if ( ptrc==NULL)      { INN->next = firstc; firstc=INN; ptrc=INN; ptrb=ptrb->next; ptra=ptra->next;}
        else {  ptrc->next = INN;  ptrc=INN; ptrb=ptrb->next; ptra=ptra->next;}     }            // end m !=0
        else { ptrb=ptrb->next; ptra=ptra->next;}  // m=0;  }       //end while ( ptra !=NULL && ptrb !=NULL)

       while( ptra !=NULL)         { Node *INN= new Node;  INN->data = ptra->data;              INN->expo = ptra-> expo;
          if ( ptrc==NULL)          { INN->next = firstc; firstc=INN; ptrc=INN;ptra=ptra->next;}
           else { ptrc->next = INN;  ptrc=INN; ptra=ptra->next;}       }//end   while( ptra !=NULL)

        while( ptrb !=NULL)         { Node *INN= new Node;  INN->data = ptrb->data;              INN->expo = ptrb-> expo;
              if ( ptrc==NULL)              { INN->next = firstc; firstc=INN; ptrc=INN;ptrb=ptrb->next;}
           else { ptrc->next = INN;  ptrc=INN; ptrb=ptrb->next;} }
         }//end  while( ptrb !=NULL)
```

4 -doubly-linked lists and the standard C++ list

4- اللوائح المترابطة المضاعفة والصنف القياسي list1 في C++

إن جميع اللوائح التي قمنا بدراستها حتى الآن أحادية الاتجاه unidirectional، وهذا يعني أنه من الممكن الانتقال ببساطة من العقدة إلى العقدة التالية لها. تتطلب بعض المسائل تحديد العنصر السابق بنفس الطريقة التي نحدد فيها العنصر اللاحق، وبالتالي فإن القدرة على التحرك ثنائي الاتجاه bidirectional ضرورية لأن إيجاد العنصر السابق في لائحة مترابطة أحادية غير مجدي على الإطلاق لأنه يتطلب البحث من بداية اللائحة.

نتعرف في هذه الفقرة على كيفية بناء اللوائح ثنائية الاتجاه ومعالجتها، ثم نتعرف على كيفية استخدام الصنف الحاوي القياسي list في لغة C++ ونقوم باستخدامها في مسألة إجراء العمليات الحسابية على الأعداد الصحيحة الضخمة



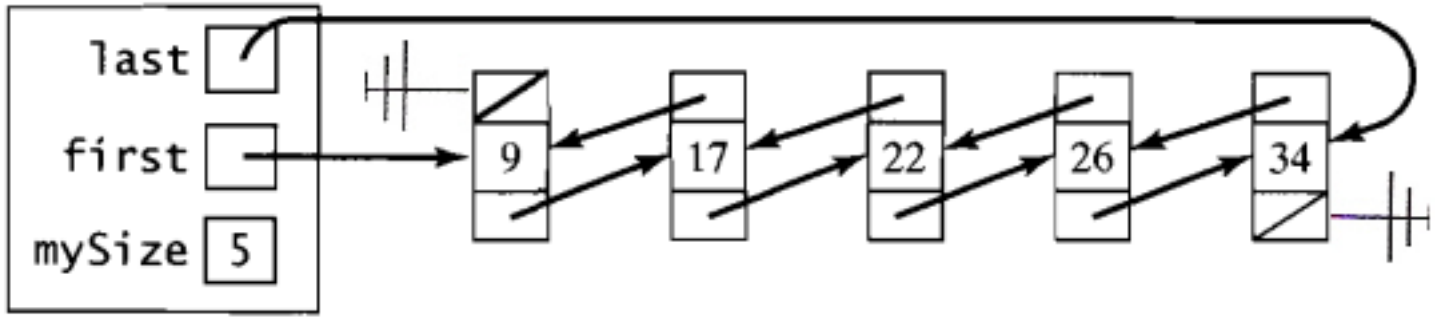
اللوائح المترابطة المضاعفة doubly-linked list:

يمكن بناء اللوائح ثنائية الاتجاه ببساطة باستخدام عقد تحتوي، بالإضافة إلى جزء البيانات، على رابطين رابط أمامي next يشير إلى التالي للعقدة ورابط خلفي prev يشير إلى السابق لها:

4 -doubly-linked lists and the standard C++ list

4- اللوائح المترابطة المضاعفة والصنف القياسي list2 في C++

اللائحة المترابطة التي تبنى من مثل هذه العقد تدعى اللائحة المترابطة المضاعفة doubly-linked أو اللائحة المترابطة المتناظرة symmetrically-linked. لتسهيل العبور الأمامي والخلفي يستخدم مؤشر first يتيح الوصول إلى العقدة الأولى ومؤشر آخر last يتيح الوصول إلى العقدة الأخيرة، على سبيل المثال، اللائحة المترابطة المضاعفة من الأعداد الصحيحة 9,17,22,26,34 يمكن تمثيلها بالشكل التالي:

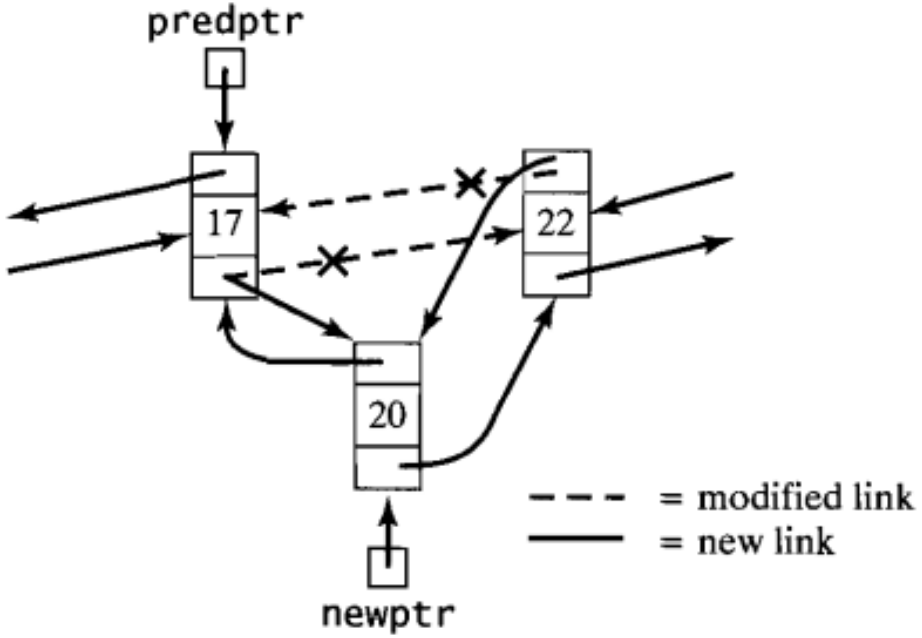


وكما في حال اللائحة المترابطة الأحادية، فإن استخدام عقدة رأسية في اللوائح المترابطة المضاعفة يخلصنا من بعض الحالات الخاصة (مثل اللائحة الفارغة و العقدة الأولى) كما أن جعل اللائحة حلقية يتيح إمكانية الوصول بسهولة إلى أي من نهايتي اللائحة. وكما سنرى لاحقاً في هذه الفقرة فإن هذه البنية مستخدمة في النمط القياسي في لغة C++ المسمى list.

4 -doubly-linked lists and the standard C++ list

4- اللوائح المترابطة المضاعفة والصنف القياسي list 3 في C++

خوارزميات العمليات الأساسية للائحة شبيهة بتلك الخاصة بالحالة أحادية الارتباط، الفارق الأساسي هو الحاجة لبعض الروابط الإضافية، على سبيل المثال، عملية حشر عقدة في لائحة مترابطة مضاعفة يتضمن أولاً جعل رابطها الأمامي والخلفي يشيران إلى العقدة السابقة لها والعقدة اللاحقة، على التوالي، ومن ثم إعادة تعيين الرابط الأمامي للعقدة السابقة لها والرابط الخلفي للعقدة اللاحقة لها يشيران إلى العقدة الجديدة:



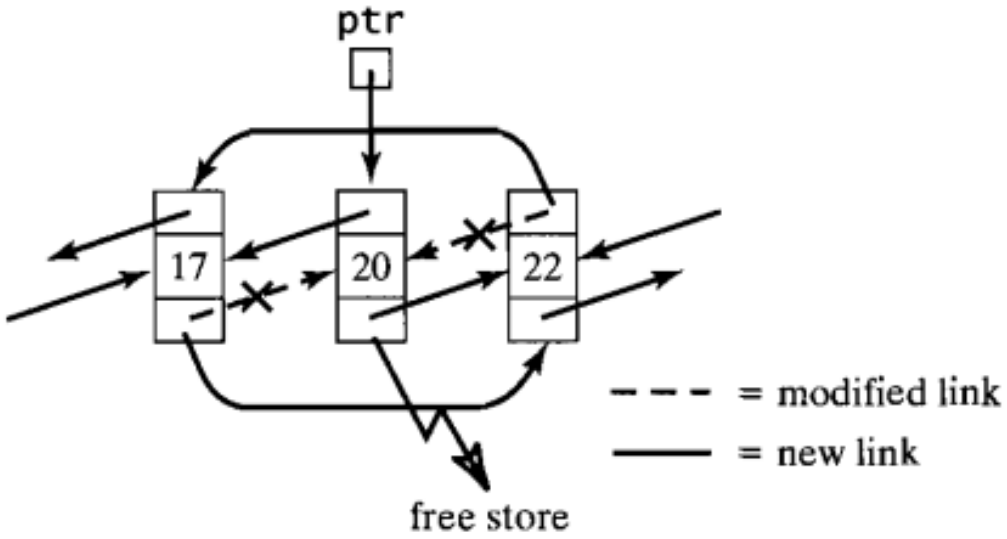
تقوم التعليمات التالية بإنجاز عملية الحشر هذه:

```
newptr->prev=predptr;
newptr->next=predptr->next;
predptr->next->prev =newptr;
predptr->next=newptr;
```

من المهم جداً أن يتم هذا بالترتيب الصحيح.

4 -doubly-linked lists and the standard C++ list

4- اللوائح المترابطة المضاعفة والصنف القياسي list 4 في C++



يمكن حذف عقدة ببساطة بإعادة تعيين الرابط الأمامي للعقدة السابقة لها والرابط الخلفي للعقدة التالية لها لتجاوز العقدة:

يمكن تحقيق ذلك بثلاث تعليمات:

```
ptr->next->prev=ptr->prev;
ptr->prev->next=ptr->next;
delete ptr;
```

يتم إجراء عملية التجول بنفس الطريقة كما في اللائحة أحادية الارتباط. التجول الأمامي يتبع الرابط next والتجول الخلفي يتبع الرابط prev. وكذلك التوابع البانية، الهادمة، البانية الناسخة، الإسناد وغيرها من العمليات الأخرى هي نسخ معدلة من تلك الخاصة باللوائح أحادية الارتباط.

5 - لوائح أخرى متعددة الارتباط 1

other multiply-linked lists

رأينا أن اللوائح المترابطة المضاعفة هي عبارة عن بنى معطيات مفيدة في التطبيقات التي تتطلب التنقل في كلا الاتجاهين، في هذه الفقرة نتعرف على مجموعة من الأشكال الأخرى من معالجة اللوائح وهي تلك التي تحتوي عقد اللائحة المترابطة على أكثر من رابط ، إن هذه الأشكال مدروسة هنا بشكل مختصرويمكن للراغبين بالتوسع فيها العودة إلى مراجع اللغة.

اللوائح متعددة الترتيب multiply-ordered lists:

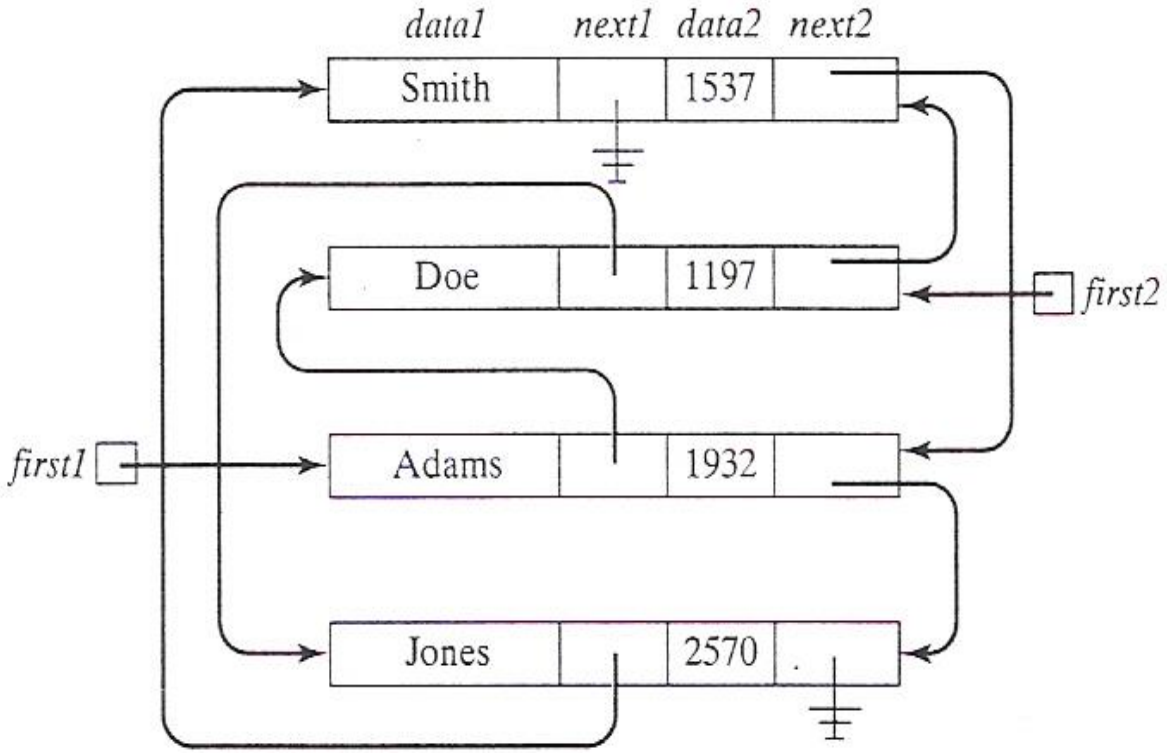
رأينا في الفصل السابق اللوائح المرتبة والتي رتبت فيها العقد بحيث أن عناصر البيانات (أو القيم في بعض الحقول المفتاحية لعناصر البيانات) مخزنة في هذه العقد بترتيب تصاعدي، في حين في بعض التطبيقات، يكون من الضروري الإبقاء على اعلى مجموعة مرتبة بطريقتين مختلفتين أو أكثر، على سبيل المثال، قد نرغب بالحصول على مجموعة من سجلات الطلاب مرتبة بحسب كل من الاسم والرقم الجامعي.

إن إحدى الطرق لتحقيق هذا الترتيب المتعدد هو المحافظة على لوائح مترابطة مرتبة منفصلة، واحدة من أجل كل معيار ترتيب مرغوب. لكن هذه الطريقة غير فعالة وخاصة من أجل السجلات الضخمة، لأن هذا يتطلب على نسخ متعددة من كل سجل.

يمكن استخدام أسلوب أفضل وذلك باستخدام لائحة واحدة تستخدم فيها عدة روابط لربط العقد مع بعضها بتراتيب مختلفة، على سبيل المثال، لتخزين مجموعة من السجلات تحوي أسماء الطلاب وأرقامهم الجامعية حيث الأسماء مرتبة ترتيباً أبجدياً والأرقام الجامعية مرتبة ترتيباً تصاعدياً، يمكن استخدام اللائحة المترابطة المتعددة التالية باستخدام رابطين في كل عقدة:

other multiply-linked lists

5 - لوائح أخرى متعددة الارتباط 2



إذا تم التجول وعرض حقول البيانات باستخدام المؤشر first1 للإشارة إلى العقدة الأولى و تتبع المؤشرات في الحقل next1 فإن الأسماء ستعطى بترتيب أبجدي:

أما إذا تم التجول باستخدام المؤشر first2 للإشارة إلى العقدة الأولى وتتبع المؤشرات في الحقل next2 يعطي الأرقام الجامعية مرتبة تصاعدياً:

next1			next2	
Adams	1932		Doe	1197
Doe	1197		Smith	1537
Jones	2570		Adams	1932
Smith	1537		Jones	2570

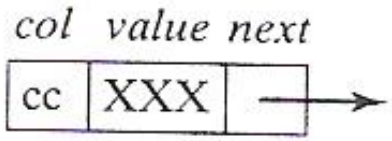
5 - لوائح أخرى متعددة الارتباط 3

مصفوفات متفرقة sparse matrices:

المصفوفة m×n هي عبارة عن مصفوفة مستطيلة تحتوي m سطر و n عمود، البنية التخزينية العادية للمصفوفات هي المصفوفات ثنائية البعد (أو شعاع vector ثنائي البعد) بما أن المصفوفات موجودة في جميع لغات البرمجة.

في بعض التطبيقات (مثل حل المعادلات التفاضلية) من الضروري معالجة مصفوفات كبيرة جداً تحوي بعض القيم غير الصفرية. إن استخدام مصفوفة ثنائية البعد لتخزين جميع القيم (بما في ذلك الصفرية) لمثل هذه المصفوفات المتفرقة ليس أمراً فعالاً. إن هذه المصفوفات يمكن تخزينها بشكل أكثر فعالية باستخدام بنية مترابطة مشابهة لتلك الخاصة بكثيرات الحدود المتفرقة.

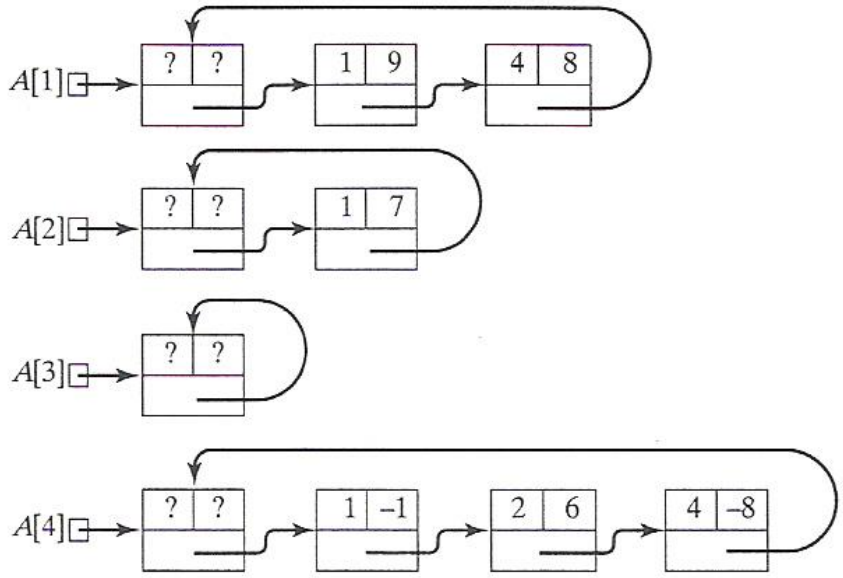
أحد التحقيقات المترابطة الشائعة يتمثل بتمثيل كل صف من المصفوفة كلائحة مترابطة تخزن فقط القيم غير الصفرية في كل صف. تخزن المصفوفة في هذا الأسلوب كمصفوفة من المؤشرات A[1],A[2],.....,A[m] واحد لكل سطر من المصفوفة. كل عنصر في المصفوفة A[i] يشير إلى لائحة مترابطة من العقد، كل منها تخزن قيمة غير صفرية في ذلك السطر ورقم العمود الموجودة فيه، ورابط إلى العقدة التي تحوي القيمة التالية في ذلك السطر: المصفوفة 5×4:

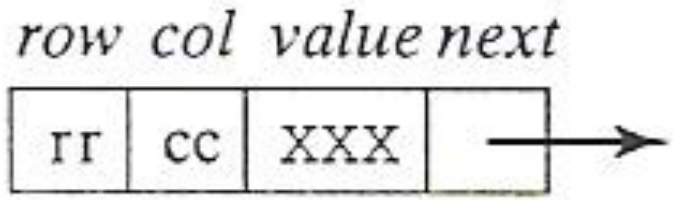


$$A=\begin{vmatrix} 9 & 0 & 0 & 8 & 0 \\ 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 6 & 0 & -8 & 0 \end{vmatrix}$$

5 - لوائح أخرى متعددة الارتباط 4

other multiply-linked lists





يمكن تمثيلها بالشكل:

على الرغم من أن استخدام البنية التخزينية المترابطة مع المصفوفات مفيد، فإن حجم المصفوفة يحد من عدد السطور التي يمكن لمثل هذه المصفوفة أن تحتويها، وأكثر من ذلك، فمن أجل المصفوفات الأصغر و/أو تلك التي تحوي عدداً كبيراً من السطور التي تحوي جميعها على قيم صفرية، فإن العديد من العناصر في المصفوفة ستهدر.

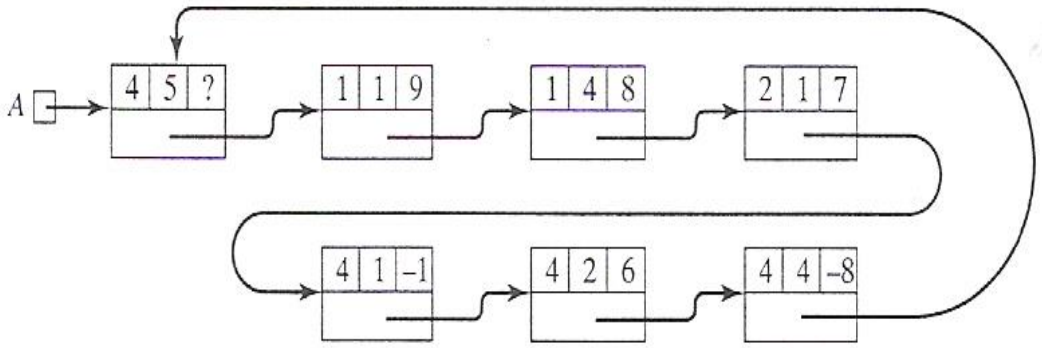
يمكن استخدام أسلوب آخر وذلك بإنشاء لائحة مترابطة أحادية، كل منها تحتوي رقم السطر، رقم العمود، القيمة غير الصفرية في ذلك السطر والعمود ورابط إلى العقدة التالية:

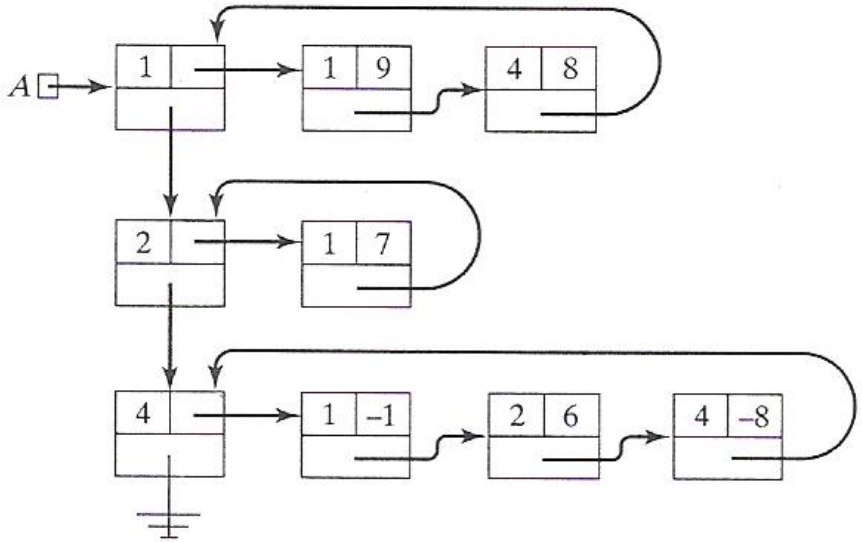
إن هذه العقد مرتبة عادة في اللائحة بحيث أن التجول عبر اللائحة يزور القيم في المصفوفة حسب ترتيب السطور

مثالاً المصفوفة 5×4 يمكن تمثيلها باللائحة المترابطة الحلقية لكل عقده مؤشر وثلاث حقول بيانات (رقم سطر، رقم عمود ، القيمة المخزنه)، مع عقدة رأس لتخزين أبعاد المصفوفة:

other multiply-linked lists

5 - لوائح أخرى متعددة الارتباط 5





بالمقابل، فإن هذا التحقيق يفقد إمكانية الوصول المباشر لكل سطر في المصفوفة.

إذا كانت المعالجة بحسب الأسطر مهمة في بعض التطبيقات الخاصة، مثل جمع المصفوفات.

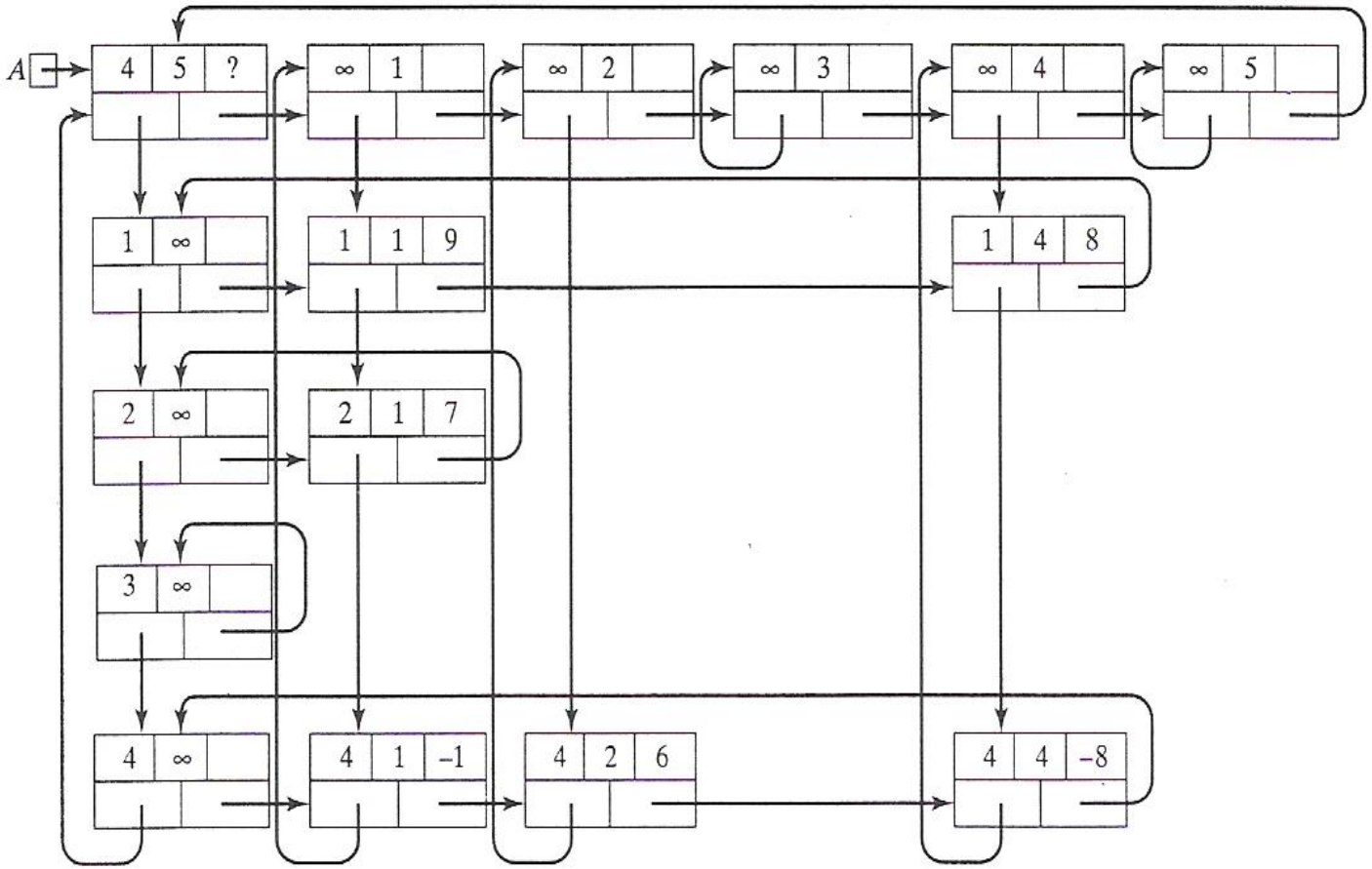
- نعتمد عقدة رأس للصف تملك رقمه ومؤشر للعقدة التالية ضمنه ومؤشر للصف التالي.
- العقد الأخرى تملك رقم العمود الحاوي للقيمة وحقل معطيات يملك القيمة . ولوائح الاسطر دائريه.

يمكن تمثيل المصفوفة 5×4 السابقة بهذه الطريقة كما يلي:

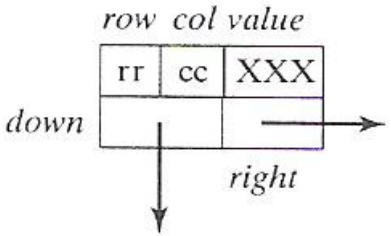
أحد عيوب هذا التحقيق المترابط هي أنه من الصعب معالجة المصفوفة بشكل عمودي عند الضرورة،

other multiply-linked lists

5 - لوائح أخرى متعددة الارتباط 6



مثالاً: عند ضرب مصفوفتين. أحد البنى المترابطة التي تتيح وصولاً سهلاً لكل من الأسطر والأعمدة من المصفوفة تدعى اللائحة المتعامدة orthogonal list. كل عقدة تخزن رقم السطر، رقم العمود والقيمة غير الصفرية في ذلك السطر والعمود وتظهر على شكل لائحة سطرولائحة عمود.



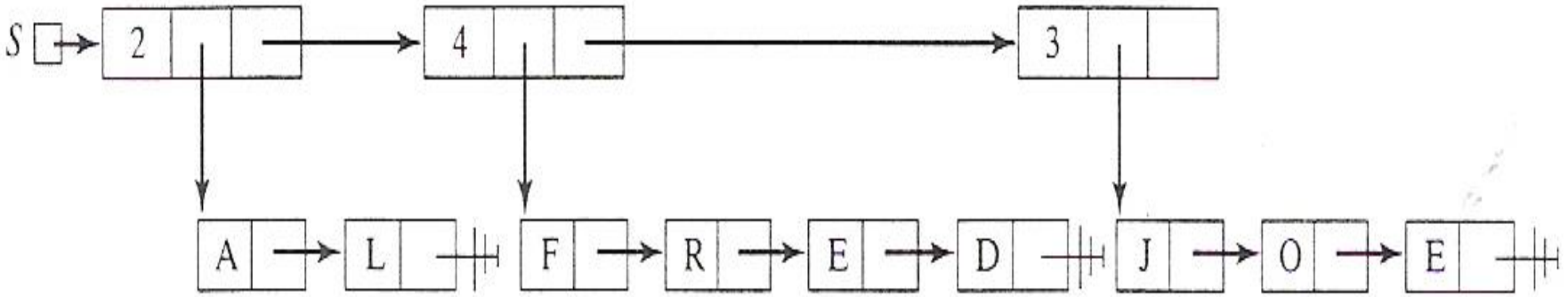
تحقيق باستخدام رابطين في كل عقدة، أحدهما يشير إلى العقدة التالية في لائحة السطر والآخر يشير إلى العقدة التالية في لائحة العمود, ولوائح اسطروالاعمدة دائرية.

generalized lists

اللوائح العامة 1

في جميع الأمثلة التي قمنا بدراستها تقريباً، كانت عناصر اللائحة وحدويه atomic الأمر الذي يعني أنها بحد ذاتها ليست لوائح. فقد درسنا، على سبيل المثال، لوائح الأعداد الصحيحة ولوائح سجلات الطلاب. في حين درسنا تمثيل المصفوفات المتفرقة التي هي لائحة من لوائح السطور.

في الحقيقة، يمكن تخزين السلسلة المحرفية في لائحة مترابطة من الأحرف، واللائحة المترابطة من السلاسل المحرفية يمكن أن تكون عندئذ لائحة مترابطة من اللوائح المترابطة. مثلاً: اللائحة S من الأسماء AL,FRED,JOE يمكن تمثيلها بالشكل:



تدعى اللوائح التي عناصرها يمكن أن تكون لوائح باسم اللوائح العامة generalized lists، وكتوضيح، لنفرض الأمثلة التالية من اللوائح:

A=(4,6) B=((4,6),8) C=(((4)),6) D=(2,A,A) E=(2,4,E)

A هي لائحة عادية تحتوي قيمتين عنصريتين هما العدد الصحيح 4 والعدد الصحيح 6.

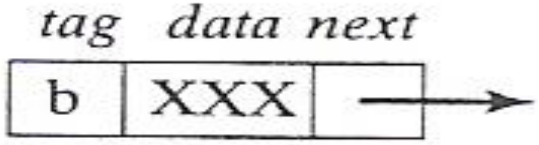
B لائحة من عنصرين، لكن العنصر الأول (4,6) هو بحد ذاته لائحة من عنصرين،

C لائحة من عنصرين، عنصرها الأول (((4)) هو لائحة مؤلفة من عنصر واحد (4)، وهذا العنصر بحد ذاته لائحة من عنصرواحد هو العدد الصحيح 4.

D لائحة من ثلاثة عناصر فيها العنصرين الثاني والثالث هما بحد ذاتهما لوائح،

E تحوي ثلاثة عناصر، ولكنه تختلف عن D في أنها تحوي نفسها كعضو. مثل هذه اللوائح تدعى اللوائح العودية recursive lists.

يمكن تمثيل اللوائح العامة كلائحة مترابطة تحوي العقدة فيها حقل مميز tag بالإضافة إلى جزء البيانات وجزء الرابط:



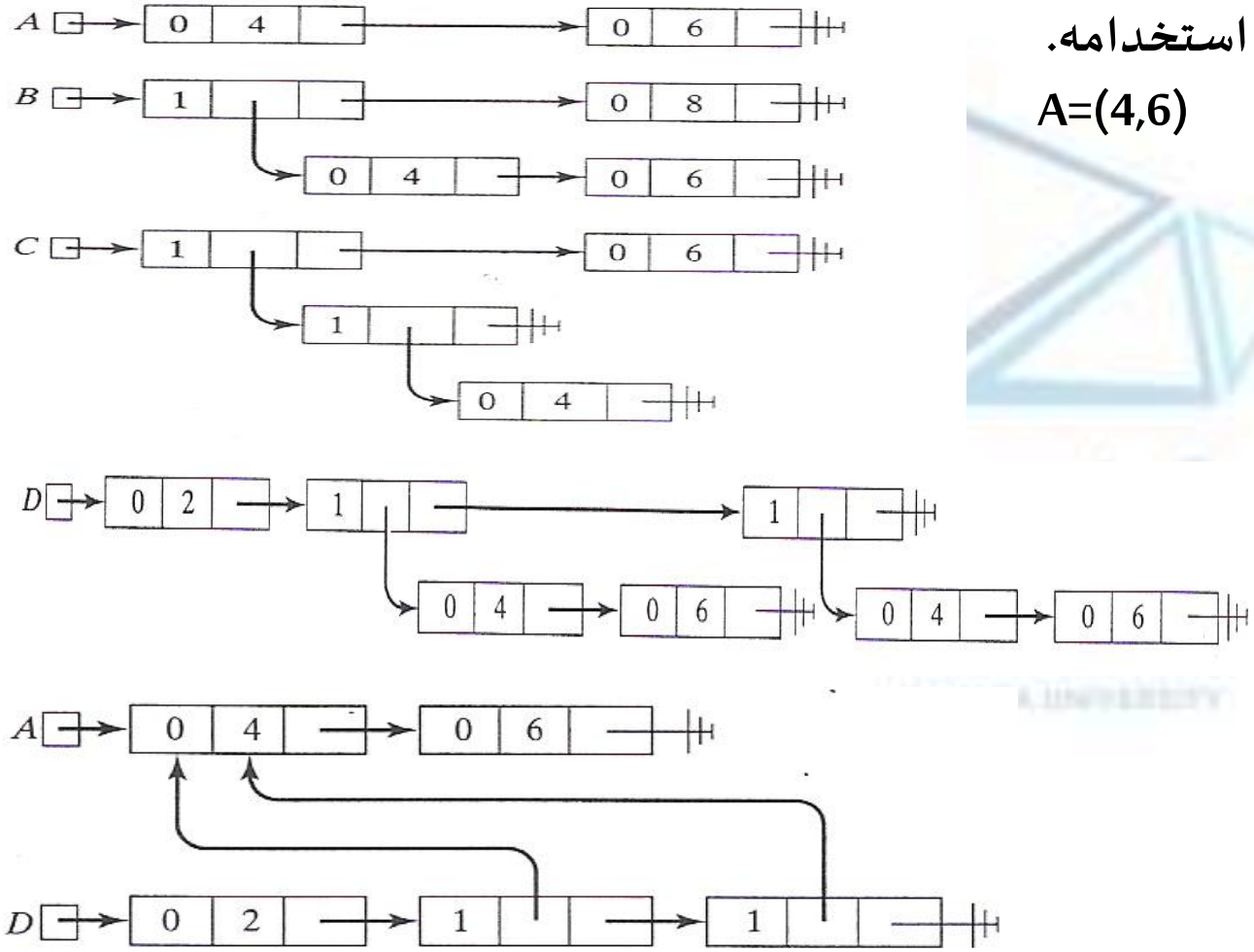
هذا المميز tag يستخدم للإشارة فيما إذا كان حقل البيانات يخزن قيمة وحدوية أو مؤشر إلى لائحة، ويمكن تمثيله كخانة وحيدة single bit، حيث 0 تعبر عن قيمة عنصرية و 1 تعبر عن مؤشر، أو كمتحول منطقي حيث true و false تقابل 0,1 وبالتالي يمكن تمثيل اللوائح A,B,C كما يلي:

generalized lists

اللوائح العامة 3

هذه اللوائح بعقدة رأس، أو حلقية، أو أي شكل آخر تريد استخدامه.

A=(4,6) B=((4,6),8) C=(((4)),6)



هناك طريقتان ممكنتان لتحقيق اللائحة D=(2,A,A) :

بما أن A هي اللائحة (4,6) يمكن اعتبار اللائحة D

كما يلي: D=(2,(4,6),(4,6))

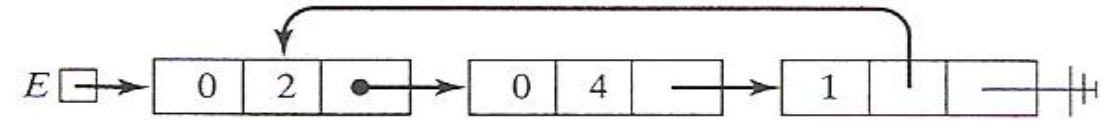
وتمثلها بالشكل التالي:

الطريقة الثانية هي السماح باستخدام اللوائح المشتركة shared lists وتمثيل D وA كما يلي:

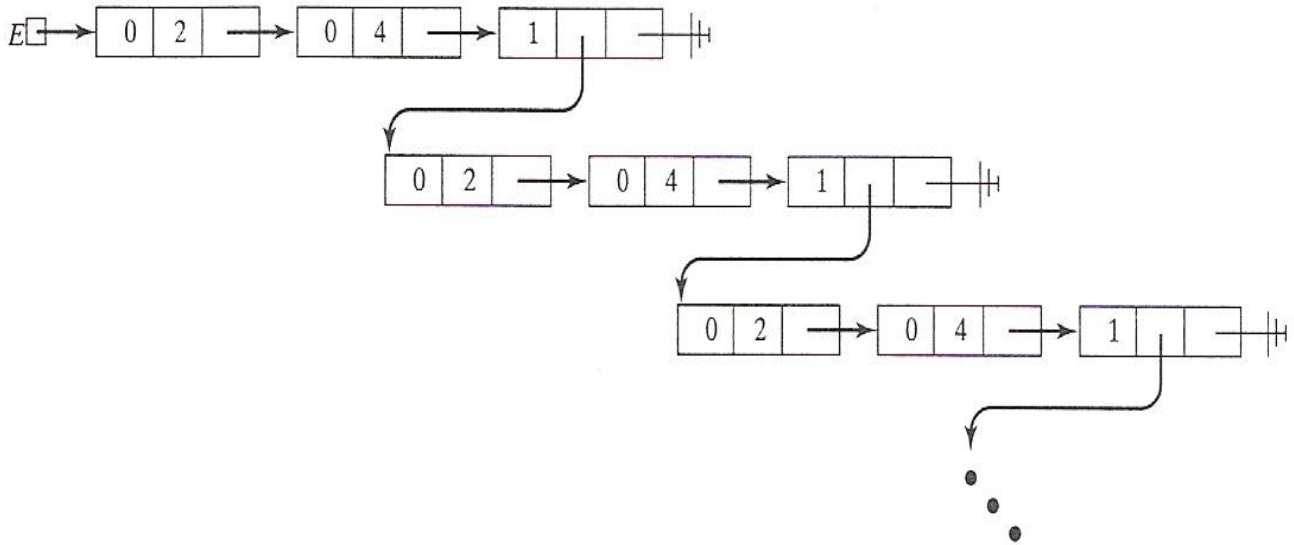
generalized lists

اللوائح العامة 4

اللائحة العودية E يمكن أن تمثل كبنية حلقية كما يلي:



وهي مكافئة للبنية اللانهائية التالية: E=(2,4,E)



قمنا في فقرة سابقة بتمثيل كثيرات الحدود بمتحول واحد بشكل مترابط، كثيرات الحدود ذات أكثر من متحول واحد يمكن تمثيلها باستخدام اللوائح العامة ويمكن بالتالي تحقيقها باستخدام بنى مترابطة شبيهة لهذه البنى.

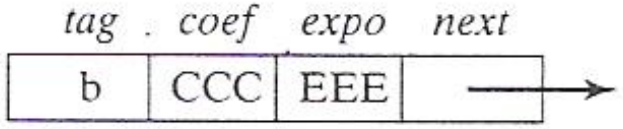
على سبيل المثال، ليكن كثير الحدود P(x,y) بمتحولين:

$P(x,y)=3+7x+14y^2+25y^7 9x^2y^7+18x^6y^7$

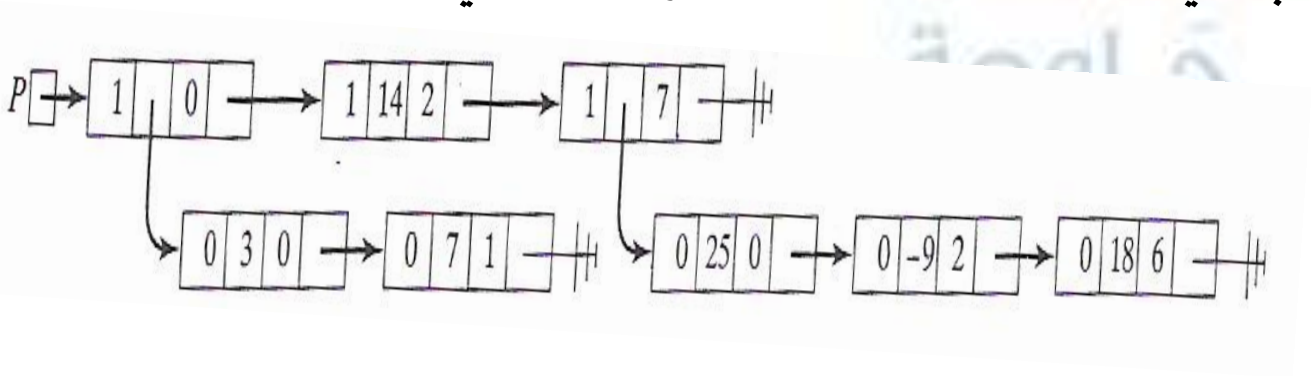
كثير الحدود هذه يمكن كتابته ككثير حدود بمتحول y معاملاته هي كثيرات حدود بدلالة x

$P(x,y)=(3+7x)+14y^2+(25-9x^2+16x^6)y^7$

إذا استخدمنا عقداً من الشكل:

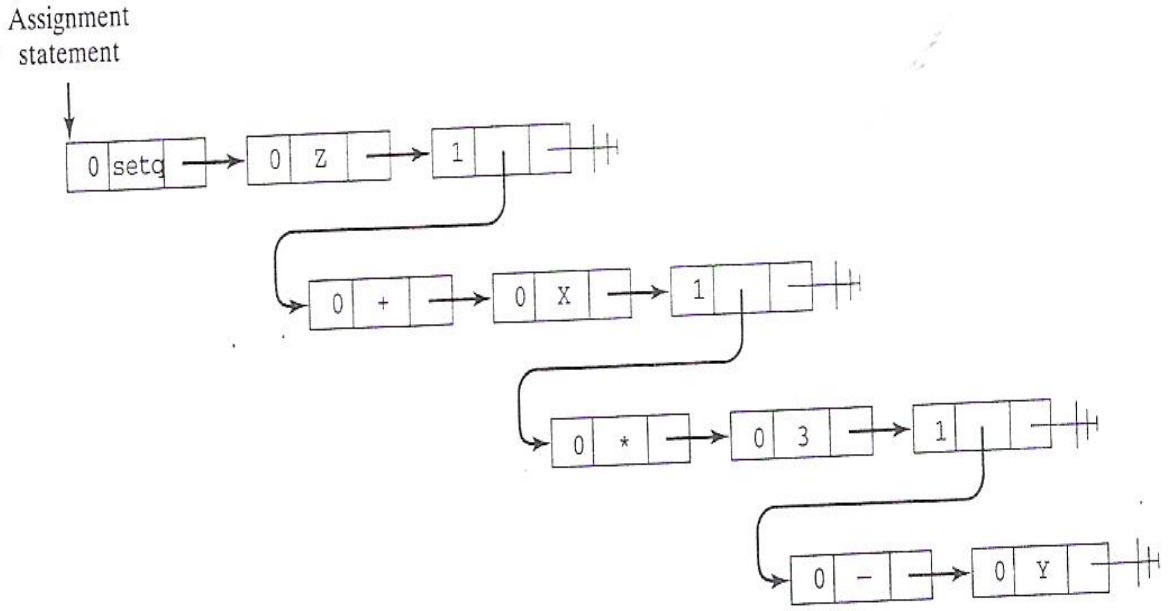


حيث tag=0 تشير إلى أن الحقل coef يخزن رقماً و tag=1 تشير إلى أنه يخزن مؤشراً إلى لائحة مترابطة تمثل كثير حدود بدلالة x. وبالتالي يمكن تمثيل كثير الحدود P(x,y) كما يلي:



إن التمثيل المترابط للوائح العامة يستخدم بكثرة في تحقيق لغة البرمجة LISP (LISt Processing) وكتوضيح تعليمة الإسناد التي يمكن كتابتها بلغة C++ كما يلي:

اللوائح العامة 5



Z=X+3*(-Y) يكتب بلغة LISP كما يلي: (setq Z(+X(*3(-Y))))

إن هذه في الحقيقة لائحة بثلاثة عناصر، العنصر الأول هو الكلمة المفتاحية setq التي تعني معامل الإسناد في لغة LISP والعنصر الثاني هو المتحول Z الذي تسند إليه قيمة والعنصر الثالث في اللائحة هو (+X(*3(-Y))) والذي يمكن كتابته بلغة C++ كما يلي: X+3*(-Y) تحتوي هذه العناصر الثلاثة على معامل الجمع +، المتحول X واللائحة الممثلة للتعبير الجزئي 3*(-Y).

وبشكل مشابه تحتوي هذه اللائحة على المعامل * والعدد الصحيح الثابت 3 وعنصري لائحة يمثلان التعبير الجزئي Y–.

هذه اللائحة تحتوي على معامل الطرح الأحادي كعنصر أول والمتحول Y كنصر ثاني.

إذا استخدمنا عقداً فيها القيمة tag=0 تشير إلى قيمة لائحة عنصرية، مثل الكلمات المفتاحية، أسماء المتحولات والثوابت.

والقيمة tag=1 تشير إلى لائحة. فإن عملية الإسناد السابقة يمكن تمثيلها بالمخطط المجاور.

انتهت المحاضرة السابعة