جَامعة المَنارة
MANARA UNIVERSITY

كلية الهندسة قسم المعلوماتية

بنى معطيات 1

Data Structure 1

ا. د. علي عمران سليمان

محاضرات الأسبوع السادس
اللوائح
lists1

الفصل الثاني 2024-2025

محتوى الفصل

اللوائح 1	Lists
1- مقدمة	
1- تحقيق اللوائح باستخدام التخزين المتتالي	
2- مدخل إلى اللوائح المترابطة	
3- تحقيق اللوائح المترابطة باستخدام المصفوفات	
4- المؤشرات في C++	
5- تخصيص وإلغاء تخصيص الذاكرة وقت التنفيذ	
6- تحقيق اللوائح المترابطة في لغة C++ باستخدام المؤشرات	
7- قالب الصنف list القياسي	

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

1- Introduction

1- مقدمة:

1- مقدمة:

- إن المكدسات، الأرتال والأرتال ثنائية الطرف التي قمنا بدراستها في الفصول السابقة هي أنواع خاصة من اللوائح،
- كل من هذه الأنماط المجردة للبيانات هو عبارة عن تتالي من عناصر البيانات والعمليات الأساسية عليها درسنا منها الإضافة والحذف.
- إن عمليات الإضافة والحذف في هذه البنى مقيدة بنهايات اللوائح، في حين لا توجد مثل هذه القيود على اللوائح العامة حيث يمكن إضافة العناصر أو حذفها في أي مكان من اللائحة.
- ندرس في هذا الفصل بشكل أكثر تفصيلاً هذه اللوائح العامة وتحقيقاتها المختلفة.

1- تحقيق اللوائح باستخدام التخزين المتتالي 1

sequential storage implementation of lists 1

يعد استخدام اللوائح بأشكالها المختلفة أمراً شائعاً في حياتنا اليومية، فهناك لوائح المشتريات، لوائح الديون، لوائح الصف، لوائح الموظفين، لوائح البريد.. وغيرها، أو حتى لوائح اللوائح. وجميع هذه الأشكال من اللوائح تحمل صفات مشتركة ويمكن من خلالها وضع التعريف التالي للائحة:

اللائحة كنمط بيانات مجرد

مجموعة من عناصر البيانات:

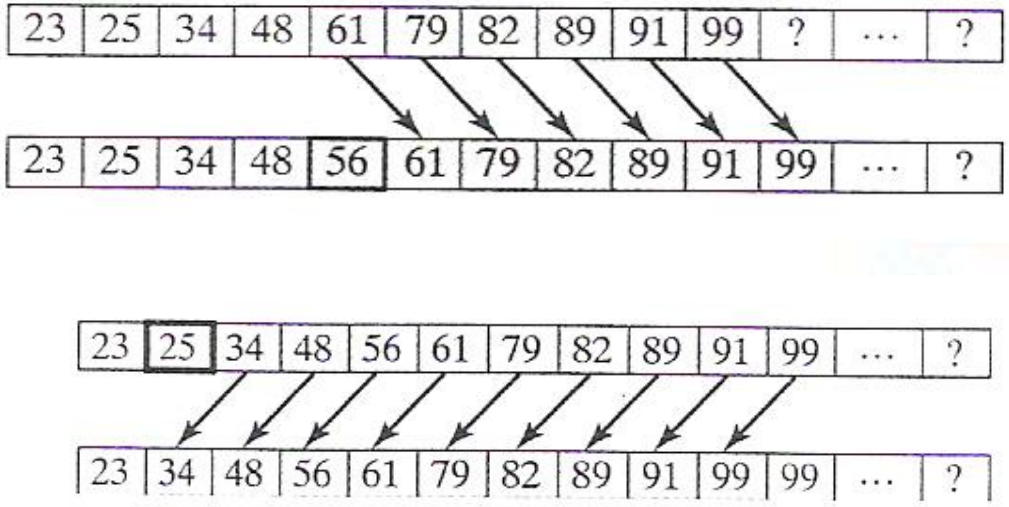
تتال منته (مجموعة مرتبة) من عناصر البيانات.

العمليات الأساسية:

- البناء construction: إنشاء لائحة فارغة.
- اختبار كون اللائحة فارغة empty.
- التجول traverse: التجول عبر اللائحة أو عبر جزء منها O(n) ،
- البحث search الوصول إلى العناصر ومعالجتها بالترتيب O(n) .
- الإدراج insert: إضافة عنصر في أي مكان من اللائحة O(n) .
- الحذف delete: حذف عنصر من أي مكان من اللائحة O(n) .

البنية التخزينية:

إن تخزين عناصر اللائحة ضمن مصفوفة أمر غير مناسب نظراً لعيوب المصفوفات من حجم ثابت وأهم العيوب هي التعديل عليها من إضافة وحذف.



نريد القيمة 56 بعد العنصر 48 في لائحة الأعداد الصحيحة:

23,25,34,48,61,79,82,89,91,99

بحيث تصبح اللائحة الجديدة

23,25,34,48,56,61,79,82,89,91,99

عند الرغبة بحذف العنصر 25 سنجد العملية المعاكسة وهي الإزاحة نجو اليسار لتاي هذا العنصر.

نظراً لهذه العيوب أن درجة التعقيد من المرتبه O(n) تطلب الأمر البحث عن بنية أكثر فعالية.

3- مدخل إلى اللوائح المترابطة 1

introduction to linked lists 1

اللائحة هي تتالي من عناصر البيانات، وهذا يعني أن هناك ترتيباً مرتبطاً بالعناصر في اللائحة: فهي تحوي عنصر أول، عنصر ثاني، وهكذا، وبالتالي فإن أي تحقيق لنمط البيانات المجرد هذا يجب أن يحتوي طريقة لتحديد هذا الترتيب.

عملية الترقيم هذه لعناصر اللائحة محققة ضمنياً **implicitly** من خلال الترتيب الطبيعي لعناصر المصفوفة، حيث العنصر الأول مخزن في الموقع الأول في المصفوفة، العنصر الثاني مخزن في الموقع الثاني للمصفوفة، وهكذا. إن هذا التوصيف الضمني للترتيب عناصر اللائحة هو الذي يتطلب إزاحتها في المصفوفة عند حشر العناصر أو حذفها مسبباً عدم فعالية التحقيق للوائح التي تتغير بكثرة بسبب عمليات الحشر والحذف.

نتعرف في هذه الفقرة على طريقة أخرى لتحقيق اللوائح تتخلص من هذا العيب من خلال التحديد الصريح **explicitly** لترتيب عناصر اللائحة.

ما هي عمليات اللوائح المترابطة:

في أي بنية مستخدمة لتخزين عناصر لائحة ما، و المحافظة على ترتيب عناصر اللائحة، يجب أن يتحقق على الأقل أداء العمليات التالية :

- تحديد موقع العنصر الأول.
- بمعرفة موقع أي عنصر في اللائحة،
- إيجاد العنصر التالي.
- تحديد موقع نهاية اللائحة.

3- مدخل إلى اللوائح المترابطة 2

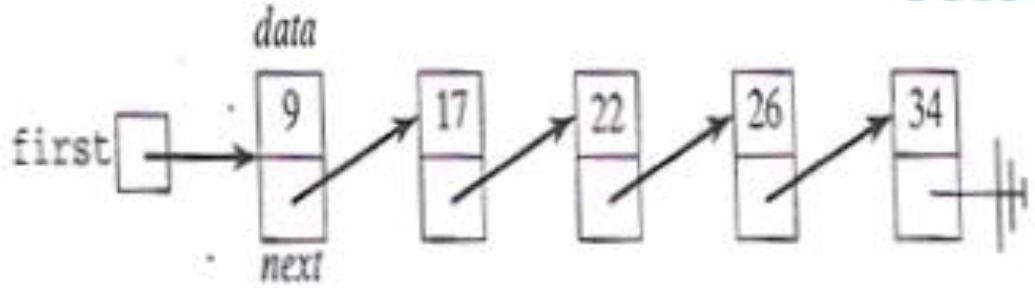
introduction to linked lists 2

إن الرغبة في تحسين الفعالية والتبسيط لهذه العملية هو من قاد إلى اللوائح المترابطة.

اللائحة المترابطة linked list هي مجموعة مرتبة من العناصر تدعى العقد nodes كل منها تتكون من جزئين:

1. جزء بيانات data part لتخزين عنصر من اللائحة.
2. جزء التالي next part لتخزين رابط link أو مؤشر pointer يشير إلى موقع العقدة الحاوية للعنصر التالي في اللائحة، إذا لم يكن هناك عنصر تالي، عندها تستخدم القيمة الخاصة null value.

على الرغم من أن موقع العقدة التي تحوي العنصر الأول يجب أن يكون محدداً، فإنه سيكون القيمة null إذا كانت اللائحة فارغة.
للتوضيح، اللائحة المترابطة التي تحتوي القيم 9,17,22,26,34 يمكن تمثيلها كما في الشكل التالي:



في هذا الشكل، تمثل الأسهم الروابط، المؤشر first يشير إلى العقدة الأولى في اللائحة. جزء البيانات في كل عقدة يخزن عدداً صحيحاً من اللائحة، ورمز الأرضي ground symbol في العقدة الأخيرة يمثل رابطاً فارغاً للإشارة إلى أن عنصر اللائحة هذا ليس له تالي.

3- مدخل إلى اللوائح المترابطة 3

تحقيق العمليات الأساسية للائحة:

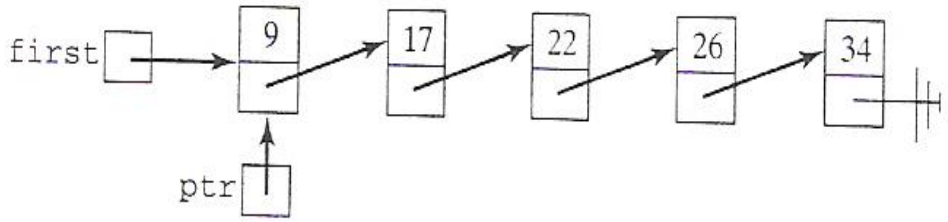
سنبين فيما يلي كيفية تحقيق العمليات الأساسية المبينة في الفقرة السابقة على اللوائح المترابطة:

عملية البناء construction: لبناء لائحة فارغة، نستطيع ببساطة جعل المؤشر first يشير إلى القيمة null للإشارة إلى أنها لا تشير إلى أي عقدة:

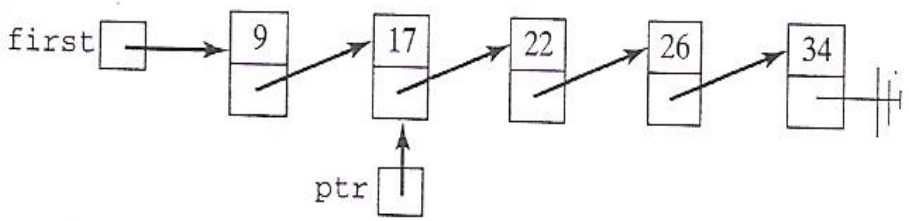
```
first=null_value;
```

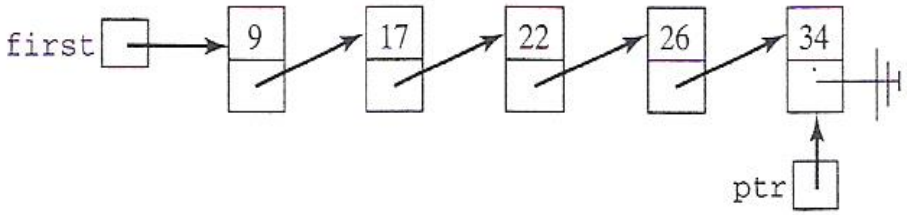
العملية empty: يمكن ببساطة تنفيذ العملية الثانية من عمليات اللائحة وهي تحديد فيما إذا كانت اللائحة فارغة من خلال اختبار كون first يشير إلى null كما يلي: first==null_value ?

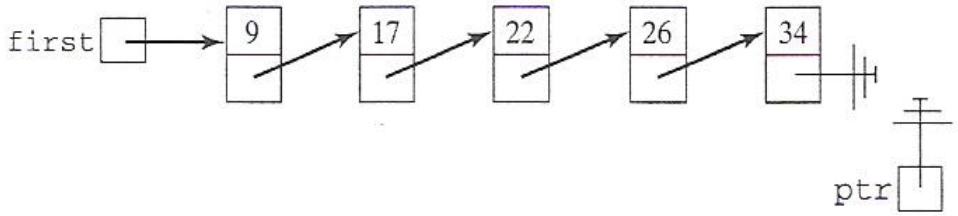
العملية traverse: العملية الأساسية الثالثة هي التجول عبر اللائحة. للتجول عبر لائحة مترابطة (كلائحة الأعداد الصحيحة المشار إليها سابقاً) نبدأ بتهيئة مؤشر مساعد ptr ليشير إلى العقدة الأولى ونعالج قيمة عنصر اللائحة 9 المخزنة في هذه العقدة.



3- مدخل إلى اللوائح المترابطة 4







للانتقال إلى العقدة التالية، نتبع الرابط من العقدة الحالية وذلك بجعل ptr مساوي للرابط في العقدة التي يشير إليها ptr (هذه العملية شبيهة بزيادة الدليل بمقدار 1 في التحقيق باستخدام التخزين المتتالي) ptr= next part ومن ثم معالجة العدد الصحيح 17 المخزن في تلك العقدة.

نستمر في هذه العملية إلى أن نصل إلى العقدة الحاوية على القيمة 34:

إذا حاولنا الانتقال إلى العقدة التالية، سيصبح ptr مشيراً إلى null معبراً عن نهاية اللائحة.

3- مدخل إلى اللوائح المترابطة 5

وكخلاصة، يمكن التجول عبر اللائحة المترابطة كما يلي:

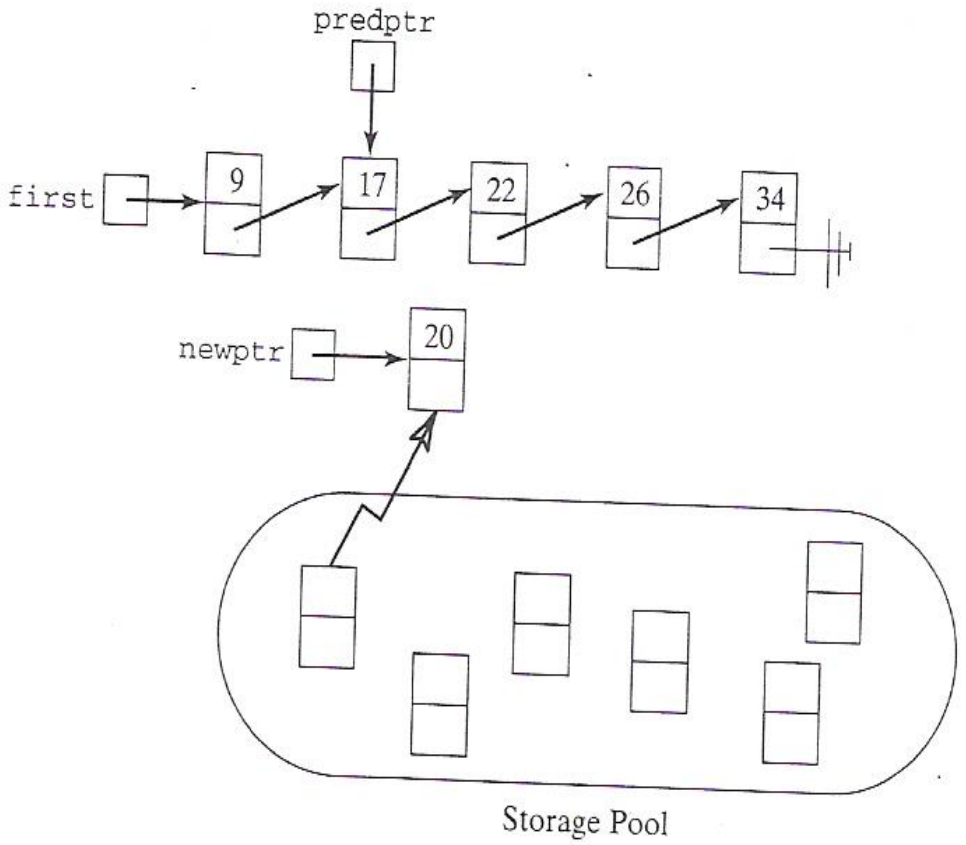
```
ptr=first;
while(ptr!=null_value)
{
        process data part of node pointed to by ptr.
        ptr= next part of node pointed to by ptr.
}
```

إن هذه الخوارزمية صحيحة حتى من أجل اللائحة الفارغة، حيث تشير في هذه الحالة first إلى القيمة null ويتم تجاوز الحلقة. لعرض محتويات اللائحة فإن المعالجة في هذه الحالة هي ببساطة إخراج جزء البيانات من العقدة. أما للبحث عن قيمة معطاة item فستكون المعالجة في هذه الحالة:

```
if (item==data part of node pointed to by ptr)
terminate the loop  //ptr points to the node containing item
```

أما إذا بدلنا المعامل == بالمعامل => فإن المقطع السابق سيحدد المكان الذي يجب أن نقوم بحشر العنصر فيه للحفاظ على اللائحة المترابطة مرتبة.

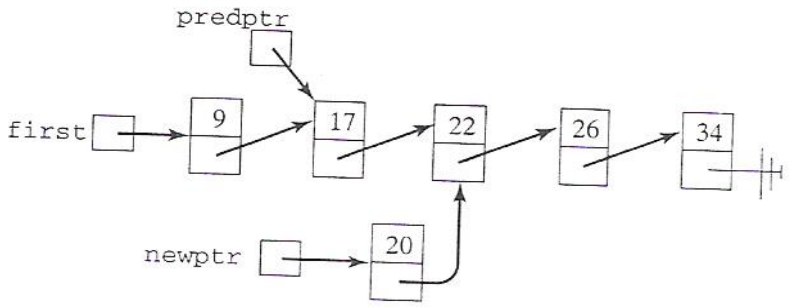
3- مدخل إلى اللوائح المترابطة 6



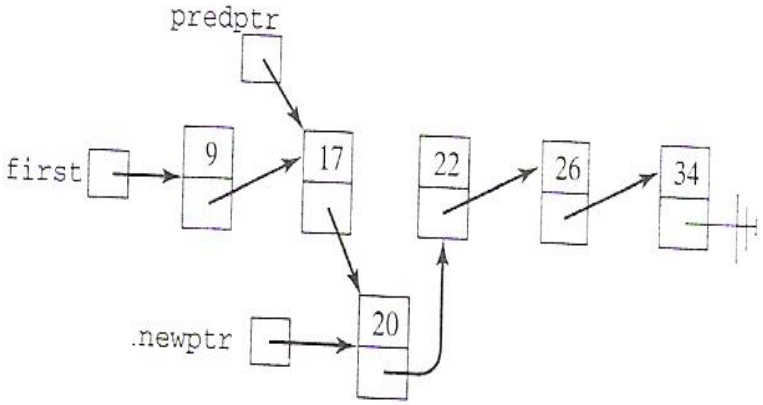
العملية insert: لحشر قيمة بيانات جديدة إلى لائحة مترابطة، يجب أولاً إنشاء عقدة جديدة وتخزين هذه القيمة في جزء البيانات الخاص بها. سنفرض أن هناك عملية ما يمكن استخدامها لإنشاء عقدة من مجمع التخزين storage pool. الخطوة التالية هي ربط هذه العقدة الجديدة باللائحة الموجودة، للقيام بذلك هناك حالتان يجب الانتباه إليهما: الأولى الحشر بعد عنصر ما ضمن اللائحة O(n)، والثانية هي الحشر في بداية اللائحة O(1).

لتوضيح الحالة الأولى، لنفرض أننا نريد حشر 20 بعد 17 في اللائحة السابقة، وبفرض أن المؤشر predptr يشير إلى العقدة الحاوية على القيمة 17. بداية علينا الحصول على عقدة جديدة ندعوها newptr ونقوم بتخزين 20 في جزء البيانات الخاص بها.

3- مدخل إلى اللوائح المترابطة 7



نقوم بحشرها في اللائحة بجعل الجزء next الخاص بها يساوي الجزء التالي للعقدة المشار إليها بالمؤشر predptr



حالياً نجعل الجزء next للعقدة المشار إليها بالمؤشر predptr يشير إلى العقدة الجديدة.

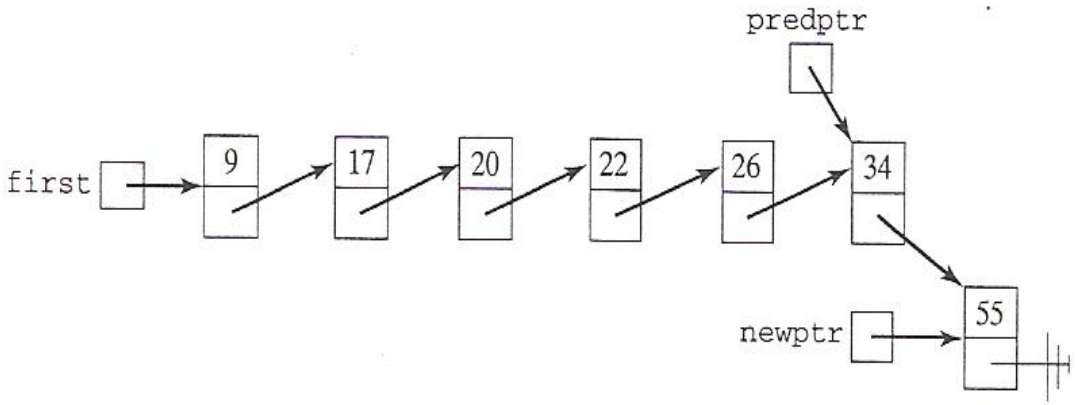
لاحظ أن عملية الحشر هذه تعمل في حالة الحشر إلى نهاية اللائحة ايضاً.

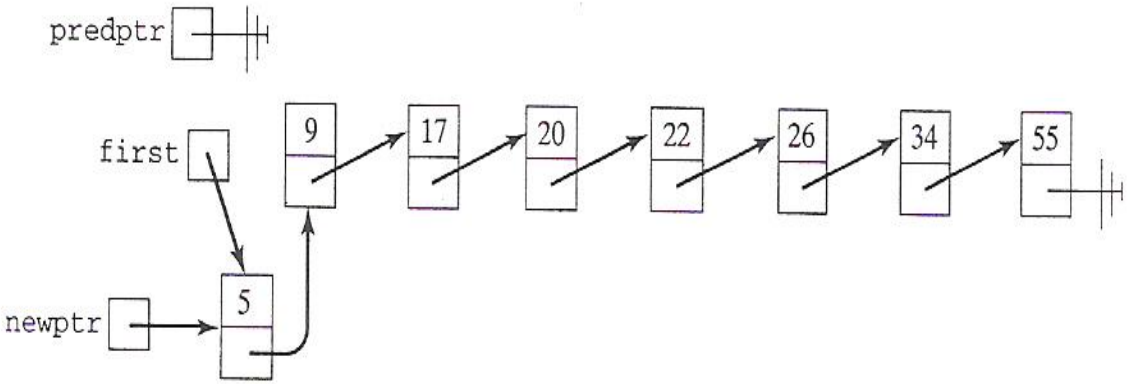
3- مدخل إلى اللوائح المترابطة 8

الشكل التالي يوضح عملية إضافة القيمة 55 إلى نهاية اللائحة.

لنفرض أننا نريد حشر عقدة تحوي القيمة 5 إلى بداية اللائحة.

لتوضيح الحالة الثانية، الخطوتان الأولى والثانية هما نفسهما كما في الحالة الأولى أما الخطوة الثالثة فتتمثل بجعل المؤشر first يشير إلى العقدة الجديدة.





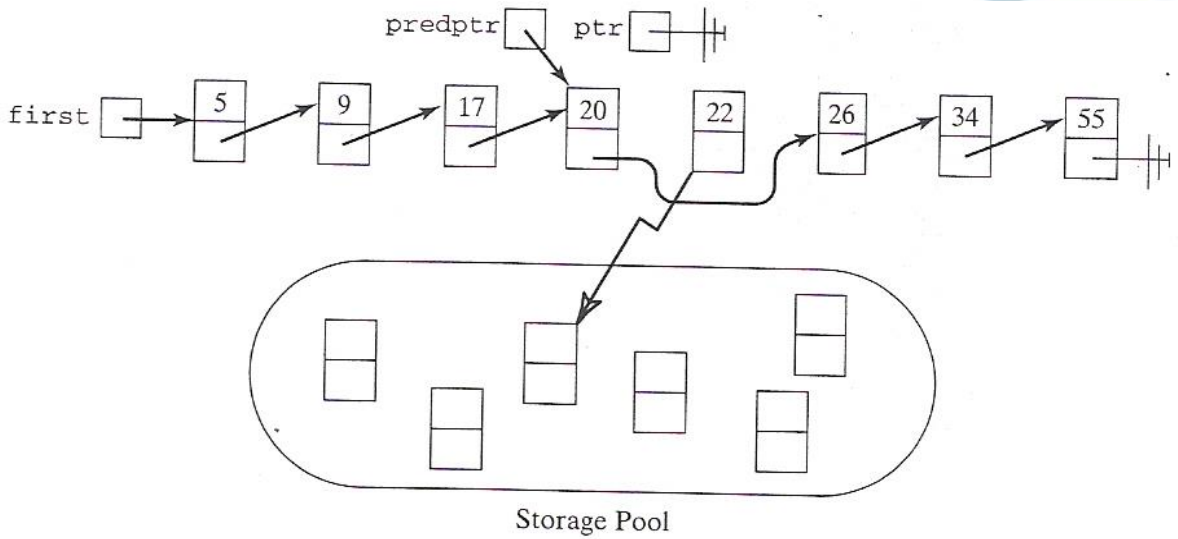
العملية **delete**: للحذف هناك أيضاً حالتان يمكن تمييزهما:

الأولى حذف عنصر له عنصر سابق O(n)

الثانية حذف العنصر الأول في اللائحة O(1).

سنفرض أن هناك عملية ما يمكن استخدامها لإعادة عقدة مشار إليها بمؤشر محدد إلى مجمع التخزين storage pool.

3- مدخل إلى اللوائح المترابطة 9



الحالة الأولى عنصر له عنصر سابق:

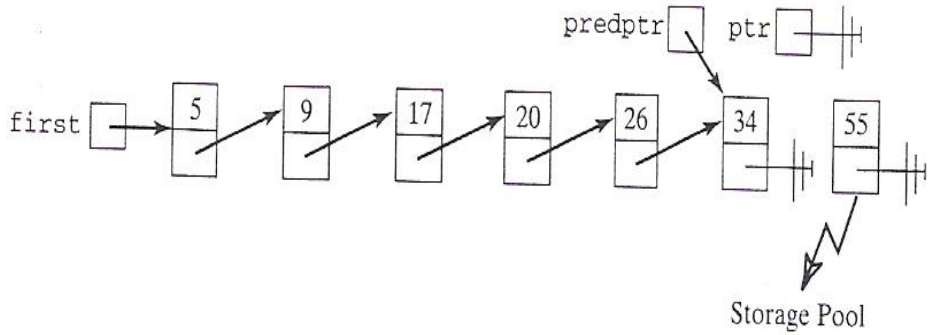
نفرض أننا نرغب بحذف العقدة الحاوية على القيمة 22 من اللائحة المترابطة السابقة،

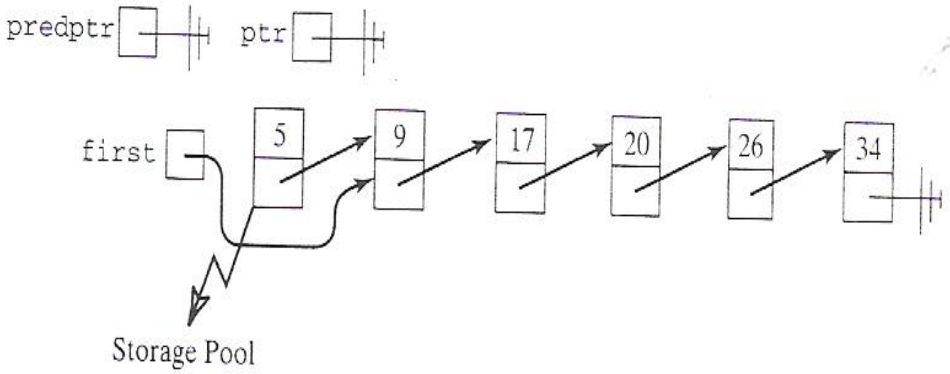
يشير المؤشر ptr إلى العقدة المراد حذفها، والمؤشر predptr يشير إلى العقدة السابقة لها (العقدة الحاوية على 20).

يمكن القيام بعملية الحذف من خلال عملية تمرير تجعل الرابط في العنصر السابق يشير إلى العنصر اللاحق للعقدة المراد حذفها.

ومن ثم إعادة العقدة المشار إليها بالمؤشر ptr إلى مجمع التخزين.

3- مدخل إلى اللوائح المترابطة 9





لاحظ أن هذه الخوارزمية تعمل في حال الحذف من نهاية اللائحة ايضاً.

الحالة الثانية حذف العقدة الأولى:

سهلة وتقتصر على جعل المؤشر first يشير إلى العقدة الثانية في اللائحة .

ومن ثم إعادة العقدة المحذوفة من اللائحة إلى مجمع التخزين.

إن العرض السابق يبين أنه من الممكن حشر عنصر في لائحة مترابطة في موقع محدد أو حذف عنصر من موقع محدد بدون إزاحة عناصر اللائحة. إن هذا يعني أنه بخلاف التحقيق باستخدام التخزين المتتالي فإن هذه العمليات تتم خلال مدة ثابتة.

تعرفنا حتى هذه اللحظة على اللوائح المترابطة بشكل مجرد فقط ولكننا لم ندرس تحقيقها. لتحقيق اللوائح المترابطة يجب أن نمتلك على الأقل المقدرات التالية:

1. بعض الوسائل لتقسيم الذاكرة إلى عقد، كل منها تتألف من جزء بيانات وجزء رابط، وبعض التحقيقات للمؤشرات.
2. عمليات للوصول إلى قيم مخزنة في كل عقدة، أي عمليات للوصول إلى جزء البيانات وجزء التالي من العقدة المشار إليها بمؤشر ما.
3. بعض الوسائل للإمساك بالعقد التي هي قيد الاستخدام بالإضافة إلى العقد الحرة ولتبادل العقد بين تلك التي هي قيد الاستخدام ومجمع العقد الحرة.

3- مدخل إلى اللوائح المترابطة (بنية العقد) 1

بنية العقدة:

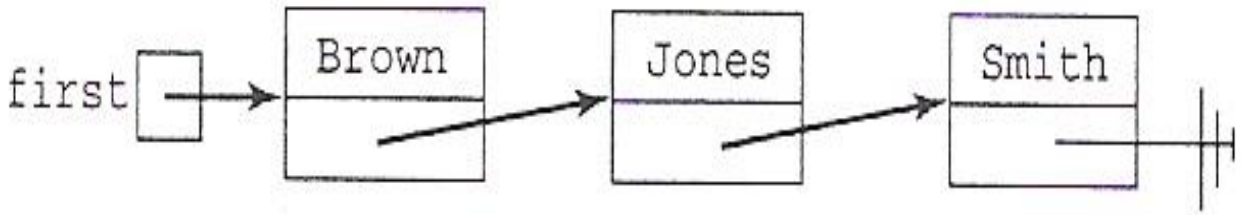
نذكر بأن العقدة في لائحة مترابطة تتألف من جزئين: جزء بيانات data part يخزن عنصر من اللائحة وجزء التالي next part يشير إلى العقدة الحاوية على القيمة التالية لهذه القيمة أو على القيمة null إذا كانت هي آخر عقده في اللائحة.

إن هذا يفترض بأن كل عقدة يمكن تمثيلها كسجل struct واللائحة المترابطة هي مصفوفة من السجلات. كل سجل 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=2048;        NodeType  node[NUMNODES];
int free;            // points to a free node
```

3- مدخل إلى اللوائح المترابطة (بنية العقد) 2

للتوضيح، لتكن لدينا اللائحة المترابطة التي تحتوي على الأسماء Brown، Jones وSmith كما يلي:



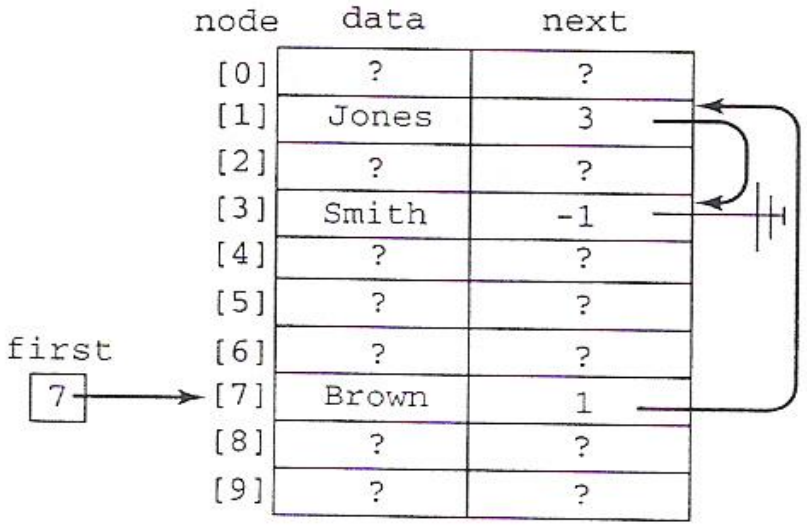
إن first هو متحول من النمط int ويشير إلى العقدة الأولى بتخزين موقعها في مجمع التخزين node. بفرض أن NUMNODES هو 10 وبالتالي فإن المصفوفة node التي تم بناؤها بالتصريح السابق تتألف من 10 سجلات.

هذه السجلات تتألف من عضو بيانات data لتخزين الاسم وعضو next لتخزين موقع العقدة التالية.

العقد في اللائحة المترابطة يمكن أن تخزن في أي ثلاثة مواقع من المصفوفة، بالإضافة إلى أن الروابط تم ضبطها بشكل مناسب و first تبقى كمؤشر إلى العقدة الأولى. فمثالاً بفرض أن العقدة الأولى خزنت في الموقع 7 من المصفوفة والعقدة الثانية في الموقع 1 والثالثة في الموقع 3 وبالتالي first سيكون مساوياً لـ 7، أما node[7].data ستخزن السلسلة Brown و node[7].next ستأخذ القيمة 1. وبشكل مشابه " و node[1].next=3، العقدة الأخيرة "node[3].data="Smith وبما أن هذه العقدة ليست لها عقدة تالية فإن node[3].next ستأخذ قيمة لا تمثل دليلاً لموقع من مواقع المصفوفة وبالتالي ستأخذ القيمة NULL_VALUE المساوية لـ 1-.

3- مدخل إلى اللوائح المترابطة (بنية العقد) 3

علامة الاستفهام في بعض مواقع المصفوفة تشير إلى قيم غير محددة لأن هذه العقد غير مستخدمة لتخزين هذه اللائحة المترابطة.



المخطط التالي يظهر محتوى المصفوفة node ويوضح كيفية قيام الأجزاء next بربط هذه العقد.

للتجول عبر اللائحة وعرض الأسماء بالترتيب، نبدأ بتحديد موقع العقدة الأولى باستخدام المؤشر first، بما أن first تملك القيمة 7 فإن الاسم الأول الذي يتم عرضه هو "Brown" المخزن في node[7].data، بتتبع العضو next نحصل على node[7].next=1 حيث الاسم "Jones" مخزن في node[1].data.... وهكذا.

يمكن تعميم عملية التجول عبر اللائحة المترابطة بالخوارزمية التالية:

3- مدخل إلى اللوائح المترابطة (مصفوفات العقد) 1

1-initialize ptr to first.

2-while ptr ≠ NULL_VALUE do the following:

a.process the data part of the node pointed to by ptr.

b. set ptr equal to the next part of the node pointed by ptr.

في التحقيق الحالي للوائح المترابطة باستخدام المصفوفات، يمكن التعبير عن هذه الخوارزمية بلغة C++ كما يلي (حيث ptr من النمط int):

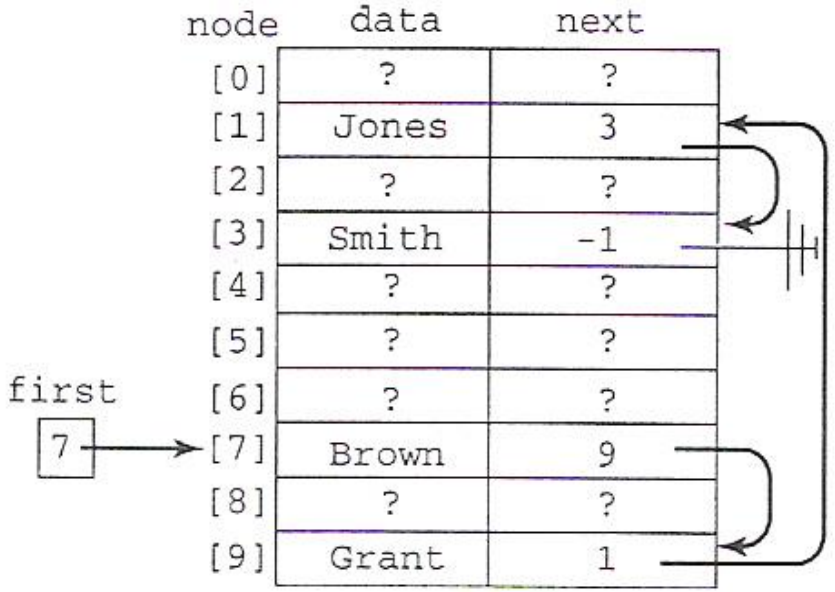
```
ptr=first;
while (ptr!=NULL_VALUE)
{       /* appropriate statements to process
   node[ptr].data are inserted here */
   ptr=node[ptr].next;
}
```

المخطط التالي يظهر محتوى المصفوفة node ويوضح كيفية قيام الأجزاء next بربط هذه العقد.

بفرض أننا نرغب بحشر اسم جديد في هذه اللائحة، على سبيل المثال لحشر "Grant" بعد "Brown"، يجب أن نحصل على عقدة جديدة لكي نخزن فيها هذا الاسم.

3- مدخل إلى اللوائح المترابطة (مصفوفات العقد) 2

هناك سبع مواقع متاحة هي المواقع 0,2,4,5,6,8,9. سنفترض أن هذه العقد منظمة بحيث أن استدعاء المعامل ( )new يعيد الدليل 9 كموقع لعقدة متاحة، يتم حشر الاسم الجديد إلى اللائحة بالطريقة المستخدمة سابقاً أي node[9].data="Grant" و node[9].next=2 ويشير إلى "Brown" وحقل الرابط node[7].next للسابق يساوي للقيمة 9



organizing the storage pool تنظيم مجمع التخزين

هذا المثال يبين أن عناصر المصفوفة node هي من نوع عقده.

مواقع المصفوفة مشغوله – بالتحديد 1، 3، 7 و 9.

العقد الآخر تمثل عقد حرة أو غير مستخدمة متاحة لتخزين العناصر الجديدة عند حشرها في اللائحة.

تعرفنا سابقاً كيف تنظم العقد المستخدمة لتخزين العناصر، والآن سنتعرف على بنية مجمع التخزين للعقد المتاحة. إحدى الطرق البسيطة لتنظيم هذا المجمع للعقد الحرة هي كمكدس مترابط linked stack، في هذه الحالة سيكون محتوى الأجزاء data لهذه العقد غير محدد. الأجزاء next تستخدم ببساطة لربط هذه العقد مع بعضها.

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 بعد 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 الشكل العام: new Type

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
```

6- تحقيق اللوائح المترابطة في لغة C++ باستخدام المؤشرات1

a pointer-based implementation of linked lists in C++ 1

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 المؤشرات1

a pointer-based implementation of linked lists in C++ 1

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 بـptr:

```
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
```

أن الصنف Node سيتم التصريح عن مكوناته عامة للسماح بالوصول العام إليها، وعند وضعه ضمن الجزء private للصنف LinkedList مما سيجدد قابلة الوصول إليه من قبل جميع التوابع الأعضاء والأصدقاء للصنف LinkedList فقط ولن يكون ممكناً الوصول إليها خارج الصنفLinkedList :

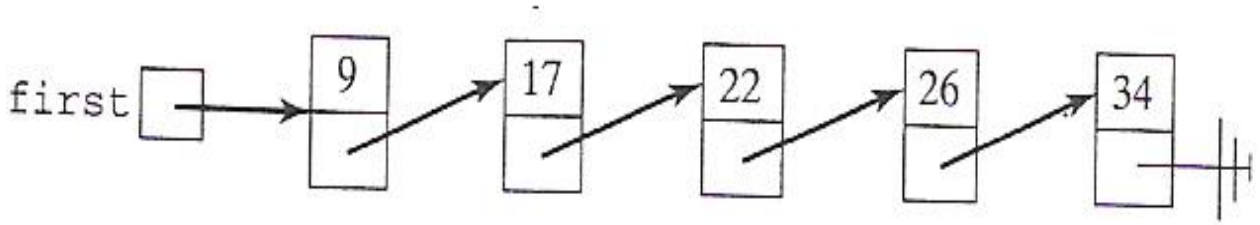
6- تحقيق اللوائح المترابطة في لغة C++ باستخدام المؤشرات1

a pointer-based implementation of linked lists in C++ 1

```
#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 المؤشرات1

a pointer-based implementation of linked lists in C++ 1



البيانات الأعضاء لـLinkedList:

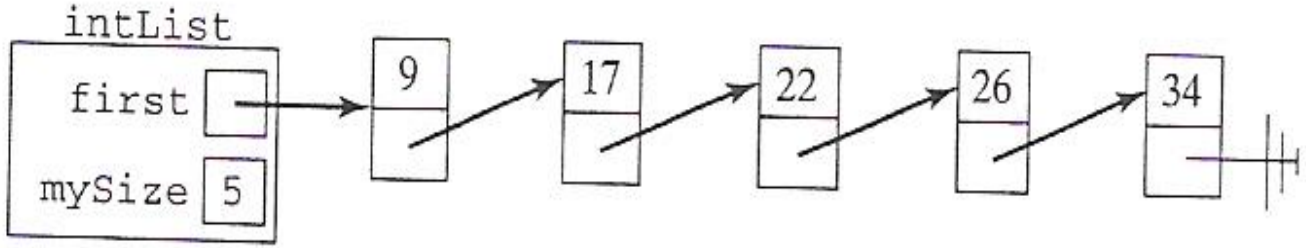
اللوائح المترابطة المبينة في الفقرتين الثانية والثالثة من الشكل:

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 الخوارزمية التالية :

6- تحقيق اللوائح المترابطة في لغة C++ باستخدام المؤشرات1

1. set count to 0
2. make ptr point at the first node
3. while ptr is not null:
   a. increment count
   b. make ptr point to the next node
4. return count

التوابع الأعضاء لـ LinkedList

التابع الباني constructor: يقوم بإنشاء لائحة فارغة، بجعل المؤشر first يشير إلى العنوان الصفري و mySize مساوية للصفر:

```
// constructor
LinkedList( ) {   first=0;   mySize=0;}
```

التابع الهادم destructor: نحتاج لإنشاء تابع هادم لنفس الأسباب التي أشرنا إليها من أجل مصفوفات وقت التنفيذ. إذا لم نقم بتعريفه، سيتم استخدام الهادم الافتراضي من قبل المترجم مما قد يسبب نضوباً في الذاكرة حيث سيقوم المترجم بإلغاء تخصيص الذاكرة للبيانات الأعضاء first و mySize ولكن العقد في اللائحة ستبقى.

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 عنصركل منها قيمته initVal.
list<T> l(fPtr , lPtr);	بناء لائحة l كلائحة تحوي نسخاً من العناصر الموجودة في مو اقع الذاكرة بدءاً من العنوان fPtr وحتى العنوان lPtr.
~list( )	تدمير مكونات وحذف قيم كامل العناصر.
l.empty( )	يعيد trueإذا 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 1

the standard list class template 1

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.

7- قالب الصنف list القياسي 1

the standard list class template 1

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.

انتهت المحاضرة السادسة