قسم المعلوماتية -كلية الهندسة

بنى معطيات 1

Data Structure 1

ا. د. علي عمران سليمان

محاضرات الأسبوع الخامس
الأرتال واللوائح
QUEUES & LISTS

الفصل الأول 2025-2026

محتوى الفصل

المكدسات	stacks
1- مقدمة	
2- مدخل إلى الأرتال	
3- تحقيق الأرتال باستخدام المصفوفات	
4- تطبيقات على الأرتال	
اللوائح 1 Lists	
1- مقدمة	
1- تحقيق اللوائح باستخدام التخزين المتتالي	

References:

- **ADTs, Data Structures, and Problem Solving with C++: International Edition, by Larry R. Nyhoff, Publisher:** Pearson , 2004
- Data Structures and Algorithms in Java, 6 th, Roberto Tamassia, Michael T. Goodrich, Michael H. Goldwasser, Pub. Wiley 2014
- د.علي سليمان، بنى معطيات بلغة JAVA، بنى معطيات بلغة C++، بنى معطيات بلغة Pascal جامعة تشرين 2014، 2007، 1998

1- مقدمة:

الأرتال queues, وهي بنية شبيهة إلى حد بعيد ببنية المكدسات, وتستخدم في طيف واسع من التطبيقات .
يمكن تحقيق الأرتال باستخدام المصفوفات كبنى تخزين أساسية, لكن وكما سنرى يحتاج تحقيقها إلى جهد أكبر قليلاً من المكدسات.

نركز الآن على بناء الصنف queues وسنرى في فصل لاحق بعض التحسينات على هذه البنية وأهمها:

- تحويله إلى قالب صنف وبالتالي يصبح نمط حاوي عام يمكن أن يعالج أي نمط من عناصر الرتل ADT.
- استخدام vector لتخزين الحاوي وبالتالي سعة الرتل يمكن أن تكبر بحسب الحاجة.
- التعرف على الحاوي queueالمعرف في مكتبة القوالب القياسية (Standard Template Library) STL.

لننظر إلى المسائل التالية:

يمكن تعريف الرتل queue على أنه صف انتظار.

- مثل صف من الناس ينتظرون الدفع على صندوق المشتريات .
- صف من العربات على حاجز عبور .
- رتل من الطائرات ينتظرون الهبوط في المطار .
- رتل من المهام في نظام حاسوبي ينتظرون استخدام جهاز خرج كالطابعة مثلاً.

1- مدخل إلى الأرتال

introduction to queues

- في كل من الأمثلة السابقة تتم خدمة العناصر في نفس الترتيب التي ترد فيه, أي أول عنصر في الرتل هو أول عنصر ستتم خدمته .
- يعتمد الرتل على بنية تقوم على مبدأ الداخل أولاً يخرج أولاً " First-In-First-Out FIFO" أو القادم أولاً يخدم أولاً "First-Come-First-Served FCFS".
- يمكن تلخيص تعريف الرتل كنمط بيانات مجرد كما يلي:

الرتل كنمط بيانات مجرد ADT

مجموعة مرتبة من عناصر البيانات تتمتع بخاصية أن العناصر يمكن أن تحذف فقط من طرف واحد يدعى بداية الرتل front, ويمكن أن تضاف العناصر فقط من الطرف الآخر ويدعى نهاية الرتل back.

العمليات الأساسية:

- بناء الرتل (عادة فارغ).
- اختبار فيما إذا كان الرتل فارغ (EmptyQ) .
- إضافة عنصر إلى نهاية الرتل (AddQ).
- معرفة العنصر الموجود في بداية الرتل (FrontQ).
- حذف العنصر الموجود في بداية الرتل (RemoveQ)

بما أن الأرتال تشبه إلى حد ما المكدسات مع اختلاف الية العمل, فإننا سنقوم بمحاكاة تحقيق الأرتال باستخدام المصفوفات:

- myArray: لتخزين عناصر الرتل.
- متحولين من النوع الصحيح:

i. myFront: الموقع في مصفوفة العناصر الذي يمكن أن تزال العناصر منه, أي موقع العنصر في مقدمة الرتل.

ii. myBack: الموقع في مصفوفة العناصر الذي يمكن أن تضاف العناصر إليه, أي موقع العنصر التالي لآخر عنصر في الرتل.

وبالتالي يمكن أن نمثل هذه البنية بالشكل التالي:

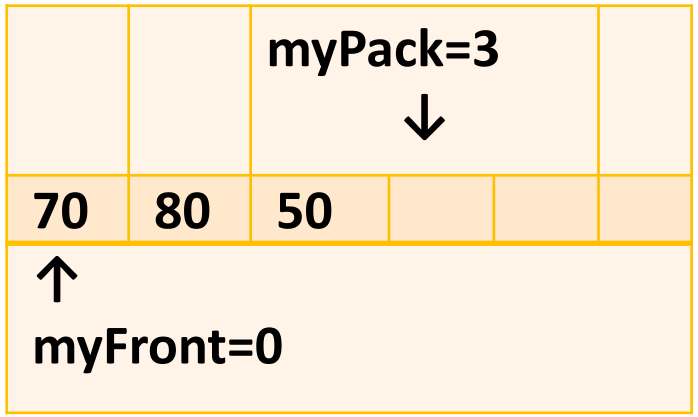
		0	1	2	3	4	5	.....	
myArray								.....	
		←front							
		back→							

1- مدخل إلى الأرتال 2

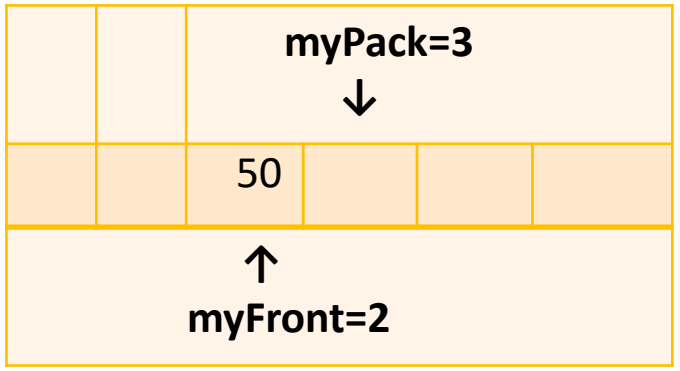
تتم إضافة عنصر إلى الرتل من خلال تخزينه في الموقع myBack من المصفوفة ثم تتم زيادة قيمة myPack بمقدار 1, حيث أن قيمة myBack لا تتجاوز قيمة عظمى هي QUEUE_CAPACITY للمصفوفة.

إن مصدر الصعوبة في هذا التحقيق للرتل يتمثل في احتمال أن يصل مؤشر الزيل لنهاية الحجز وتوجد أماكن فارغة في البداية وعندها يجب أن تزاح العناصر إلى بداية المصفوفة.

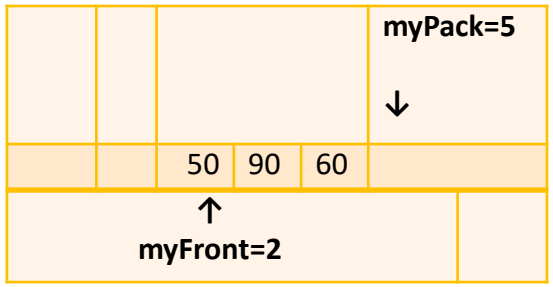
لتوضيح هذه الفكرة, لنفرض أن لدينا رتلاً سعته QUEUE_CAPACITY=5 وعناصره من النوع الصحيح,عندئذ تتالي العمليات التالي: AddQ 50, AddQ 80, AddQ 70 سينتج الوضع المجاور الأعلى في الرتل:

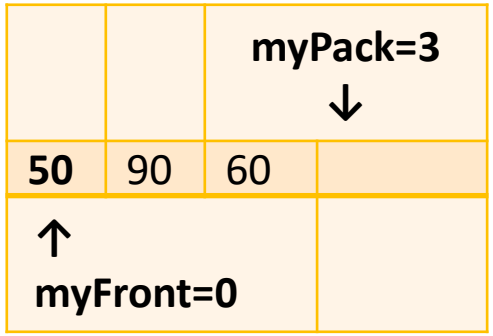


لنفرض الآن أننا أزلنا عنصرين من الرتل سنحصلعلى الرتل المجاور:



1- مدخل إلى الأرتال 2



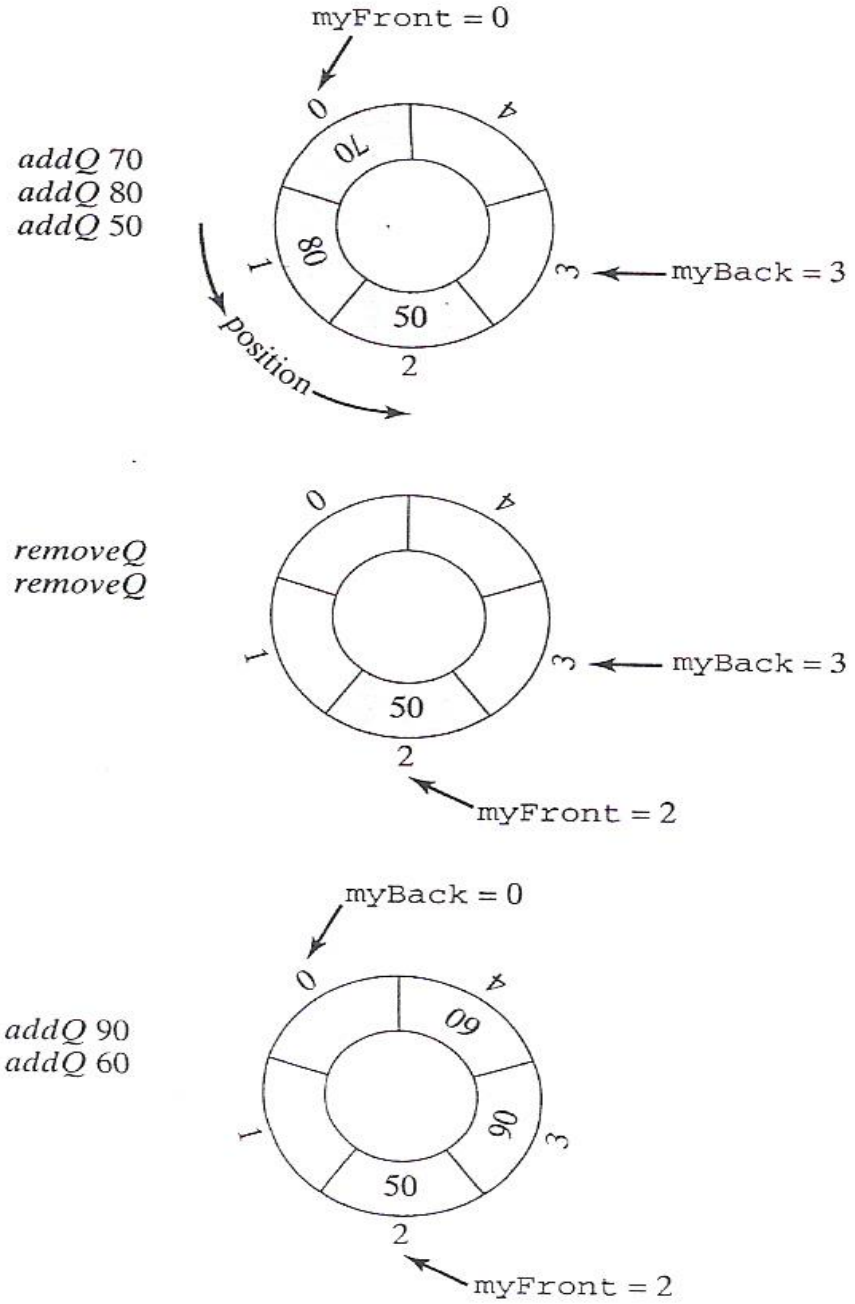


وبإضافة العنصرين AddQ 60, AddQ 90 يصبح الوضع كما يلي:

ملاحظة: عند محاولة إضافة عنصر سيخرج دليل النهاية من نطاق حجز المصفوفة.

وقبل أن يكون لدينا الإمكانية لإضافة عناصر أخرى إلى الرتل, يجب أن تزاح العناصر إلى بداية المصفوفة.

عملية الإزاحة تلك لعناصر المصفوفة غير فعالة على الإطلاق, خاصة عندما تكون العناصر عبارة عن سجلات ضخمة وعددها كبير.
يمكن تجنب تلك العملية بجعل المصفوفة دائرية circular, يكون العنصر الأول يلي العنصر الأخير, وهذا الأمر يمكن تحقيقه بفهرسة المصفوفة بدءاً من الصفر, وزيادة myFront و myPack وبإضافة باقي القسمة على QUEUE_CAPACITY.
إذا أردنا تمثيل العمليات السابقة وفق هذا الأسلوب ينتج ما يلي:



إن عملية إضافة ثلاث عناصر تالية تصبح الآن متاحة بدون الحاجة لتحريك أي عنصر من مكانه لأن myPack يشير الآن إلى الموقع 3.

لنبدأ الآن بتعريف عملية اختبار فيما إذا كان الرتل فارغاً. إذا احتوى الرتل على عنصر واحد, سيكون مثلاً في الموقع myFront=2 من المصفوفة, وسيكون myPack =3 هو الموقع الشاغر التالي له.

إذا تم حذف هذا العنصر, تتم زيادة myFront بمقدار 1 وبالتالي سيكون myFront, myBack لهما نفس القيمة, وبالتالي لتحديد فيما إذا كان الرتل فارغاً يجب أن نختبر الشرط التالي (myFront==myBack).

أما في عملية البناء, فإن باني الرتل سيقوم بتهيئة كلا من قيمتي myFront و myBack إلى الصفر.

إن هذا التحقيق للرتل يطرح إمكانية حصول امتلاء مو اقع الرتل, لتبيان هذه الحالة نفرض أن الرتل يحتوي على موقع واحد فارغ هو myPack=4 و myFront=0, إذا قمنا بتخزين عنصر في هذا الموقع فإن قيمة myPack ستزاد بمقدار 1 وبالتالي ستصبح مساوية لقيمة myFront, وهذا هو نفس الشرط الذي يشير إلى حالة كون الرتل فارغاً, أي لا نستطيع في هذه الحالة التفريق بين حالتي كون الرتل فارغاً أو ممتلئاً. يمكن تجنب هذه العملية بالإبقاء دائماً على موقع فارغ دائماً ضمن الرتل, وبالتالي يصبح شـرط اعتبار الرتل ممتلئاً هو (myBack+1)%QUEUE_CAPACITY==myFront).

هناك العديد من الطرق للإستفادة من الموقع الفارغ في بداية المصفوفة, إحدها هي تخزين عدد العناصر الموجودة حالياً في الرتل.

طريقة أخرى تتمثل في استخدام عنصر البيانات للدلالة على إمكانية الإضافة full بدلاً من عدد العناصر, إذا أصبح الرتل ممتلئاً يأخذ القيمة 0 للدلالله عن false وإلا يأخذ هذا العنصر أية قيمة للدلالة عن true. نلخيص هذا التعريف للصنف بالملف Queue.h المبين فيما يلي:

```
// queue2.cpp : main project file.

#include<iostream>
using namespace std;

#ifndef QUEUE
#define QUEUE
const int QUEUE_CAPACITY=128;
typedef int QueueElement;
class Queue
{ /****  function members ****/
public:
Queue( );
```

```
bool empty( ) const ;                    void addQ(const QueueElement & value);
QueueElement front( ) const;             void removeQ();   void display() const ;
/****  data members ****/
private:   QueueElement myArray[QUEUE_CAPACITY]; int myFront, myBack;
}; //end of class declaration

inline Queue::Queue( ) { myFront=myBack=0; }
inline bool Queue::empty( ) const { return (myFront==myBack)  ;}
void Queue::addQ(const QueueElement & value)
{   int newBack=(myBack+1)%QUEUE_CAPACITY;
  if (newBack==myFront)cout<<"**** QUEUE IS FULL ****"<<endl;
 else{      myArray[myBack]=value;  myBack=newBack; } }
QueueElement Queue::front( ) const
{ if (!empty())          return myArray[myFront];          cerr<<"*** QUEUE IS EMPTY *** \n";          }
```

```
void Queue::removeQ() {    if (empty())          cout<<"**** QUEUE IS EMPTY ****\n";
                           else    myFront=(myFront+1)%QUEUE_CAPACITY;}
void Queue::display() const { for (int i=myFront ; i<myBack ; i++) cout<<myArray[i]<<"  "; }
#endif

void main() {        Queue q1;          cout<<"Queue is empty "<<q1.empty( )<<endl;
          q1.addQ(70); q1.addQ(80);q1.addQ(50);       cout<<"Queue is empty "<<q1.empty( )<<endl;
          q1.display();       q1.removeQ();q1.removeQ();cout<<"\n";     q1.display(); cout<<"\n";
           q1.addQ(90);       q1.addQ(60);q1.addQ(30);  q1.display();   cout<<"\n\n";system("pause");}
```

```
Queue is empty 1
Queue is empty 0
70  80  50
50
50 90 60 30
Press any key to continue . . .
```

محتوى الفصل

اللوائح 1	Lists
1- مقدمة اللوائح	Lists
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- مقدمة:

- إن المكدسات، والأرتال 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.

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.

1- تحقيق اللوائح باستخدام التخزين المتتالي 1

sequential storage implementation of lists 1

جميع هذه الأشكال من اللوائح تحمل صفات مشتركة ويمكن من خلالها وضع التعريف التالي للائحة:

اللائحة كنمط بيانات مجرد

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) .

إن تخزين عناصر اللائحة ضمن مصفوفة أمر غير مناسب نظراً لعيوب المصفوفات :

1. حجم ثابت .
2. بشكل عام التعديل عليها من إضافة وحذف يتطلب عدد كبير من الازاحات.

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

3- مدخل إلى اللوائح المترابطة 2

introduction to linked lists 2

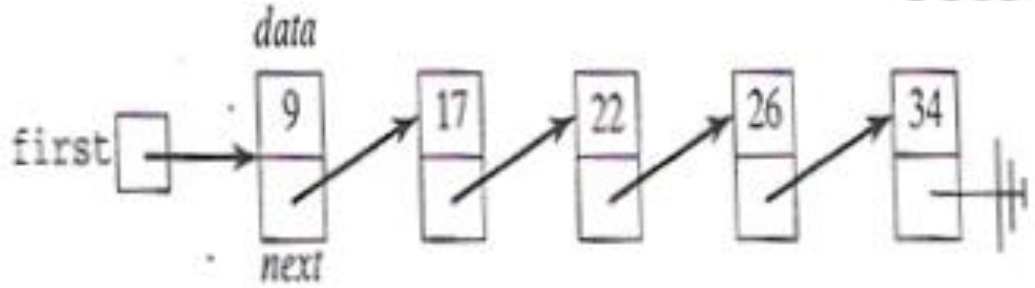
إن الرغبة في تحسين الفعالية والتبسيط لهذه العملية هو من قاد إلى اللوائح المترابطة.

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

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.

```
class Node { public: Node(): next(NULL){}; int data; Node *next; };
```

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 العناصر 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 وتوضع القيمة NULL للدلالة على النهاية.

تحقيق العمليات الأساسية للائحة:

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

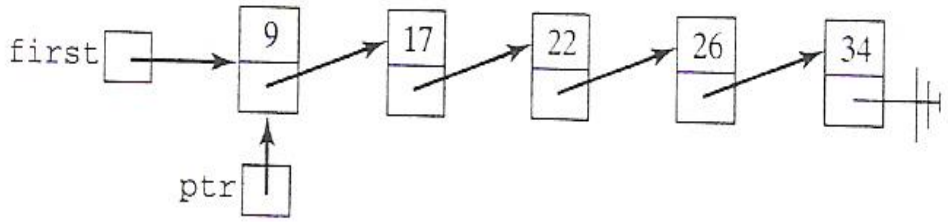
عملية البناء construction: لبناء لائحة فارغة، نستطيع ببساطة جعل المؤشر first يشير إلى القيمة NULL للإشارة إلى أنها لا تشير إلى أي عقدة:

```
first=NULL; //NULL_value;
```

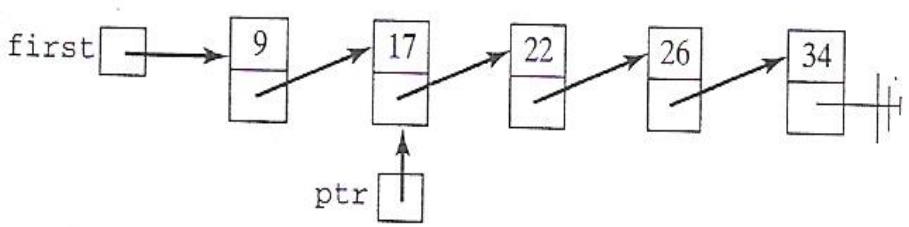
العملية empty: ببساطة تنفذ العملية الثانية من عمليات اللائحة وهي تحديد فيما إذا كانت اللائحة فارغة من خلال اختبار كون first يشير إلى NULL كما يلي:

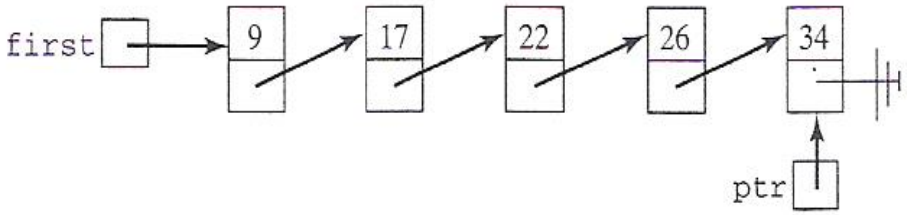
```
if(first==NULL) /                    /first==NULL_value ?
```

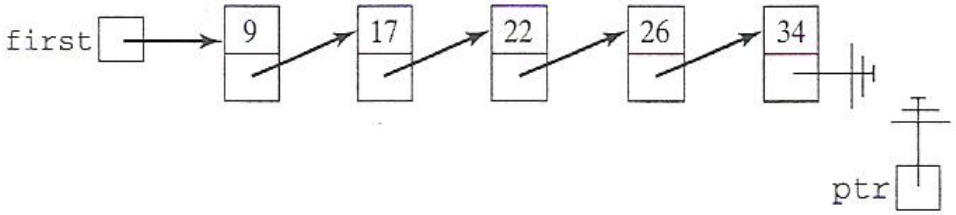
العملية traverse: العملية الأساسية الثالثة هي التجول عبر اللائحة. للتجول عبر لائحة مترابطة، نبدأ بتهيئة مؤشر مساعد ptr ليشير إلى العقدة الأولى ونعالج قيمة عنصر اللائحة 9 المخزنة في هذه العقدة.



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 العدد الصحيح 17 المخزن في تلك العقدة.وتكتب بلغة C++ وفق التالي:

```
ptr = ptr -> next;}
```

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.

وكخلاصة، يمكن التجول عبر اللائحة المترابطة وطباعة محتواها كما يلي:

```
void display(){Node *current=first;
        while(current!=NULL) {cout<<current->data<<" ";        current=current->next;}
        cout<<endl;
}
```

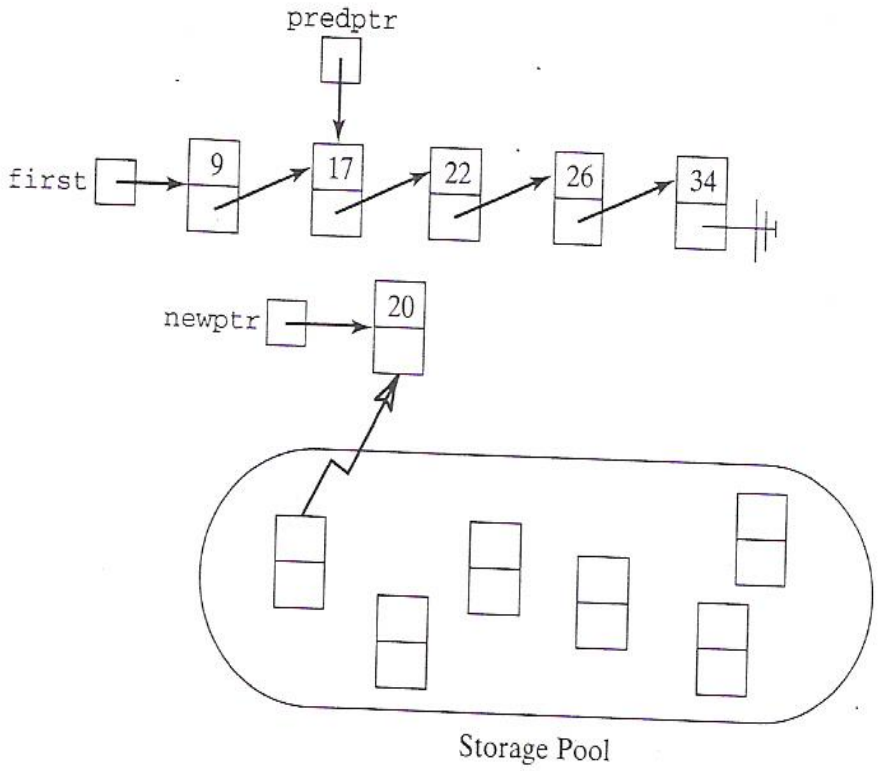
إن هذه الخوارزمية صحيحة حتى من أجل اللائحة الفارغة، حيث تشير في هذه الحالة first إلى القيمة NULL ويتم تجاوز الحلقة.
لعرض محتويات اللائحة فإن المعالجة في هذه الحالة هي ببساطة إخراج جزء البيانات من العقدة.

عملية البحثsearch: لتكن قيمة معطاة item فستكون المعالجة باستبدال التعليمة الأولى من الحلقة بالشرط التالي:

```
if (item== current->data)          // data part of node pointed to by current
Break;                             // terminate the loop  current points to the node containing item
```

عملية الحشرinsert : لحشر عقدة قبل عقدة ما يتطلب استخدم مؤشر اخر predptr متأخر عقدة عن المؤشر الدال على مكان الحشر 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 القيمة المرغوب حشرها في جزء البيانات الخاص بها. إن الباني يعبر عن عملية إنشاء عقدة من مجمع التخزين storage pool.

الخطوة التالية هي ربط هذه العقدة الجديدة باللائحة الموجودة، للقيام بذلك هناك حالتان يجب الانتباه إليهما:

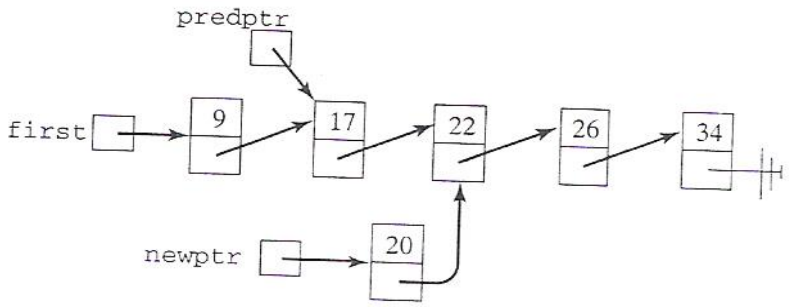
- الأولى الحشر بعد عنصر ما ضمن اللائحة تعقيدها O(n)،
- الثانية هي الحشر في بداية اللائحة تعقيدها O(1).

لتوضيح الحالة الأولى، لنفرض أننا نريد حشر 20 بعد 17 في اللائحة السابقة، وبفرض أن المؤشر predptr يشير إلى العقدة السابقة لموقع الحشر، العقدة الحاوية على القيمة 17.

1- بداية علينا الحصول على عقدة جديدة لندعوها newptr ونخزن 20 في جزء البيانات الخاص بها أي n=20.

```
Node * newptr =new Node ;        newptr ->data=n;        Node *before,*after=first;
```

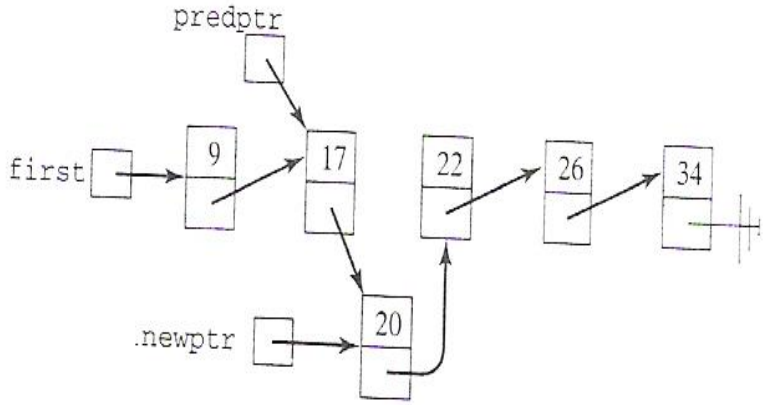
3- مدخل إلى اللوائح المترابطة 7



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

```
newptr ->next = predptr ->next ; //newptr ->next=after;
```

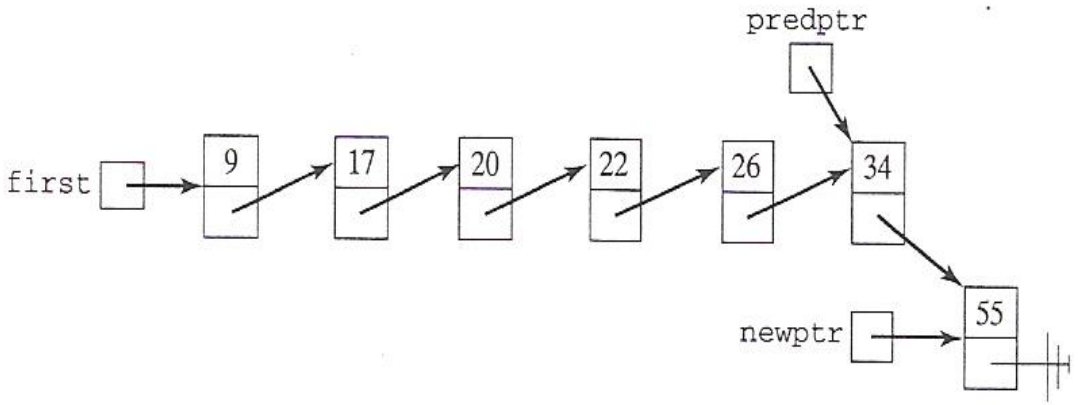
3- حالياً نجعل الجزء next للعقدة المشار إليها بالمؤشر predptr يشير إلى العقدة الجديدة.




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

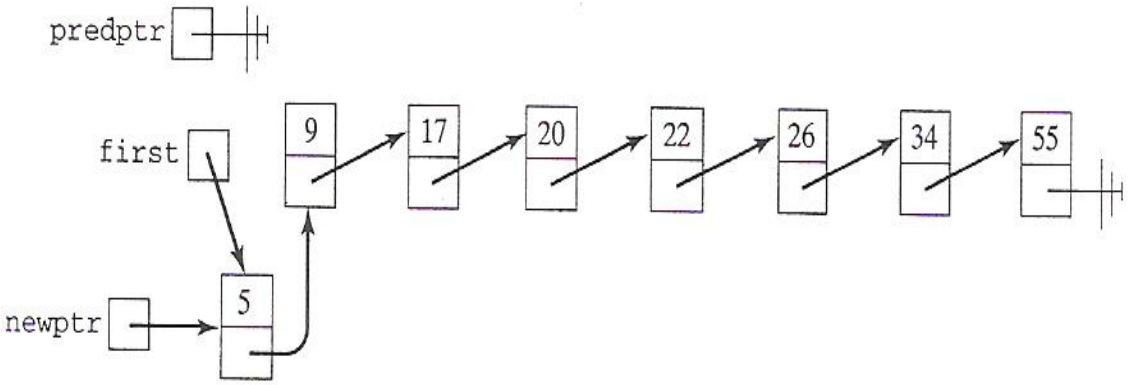
لاحظ أن عملية الحشر هذه تعمل في حالة الحشر إلى نهاية اللائحة ايضاً.

3- مدخل إلى اللوائح المترابطة 8



الشكل التالي يوضح عملية إضافة القيمة 55 إلى نهاية اللائحة فقط نغير القيمة أي n=55.

```
predptr>next=newptr ;   // insert at last
```



لحشر عقدة تحوي القيمة 5 إلى بداية اللائحة.

<u>لتوضيح الحالة الثانية الحشر في بداية الائحة:</u>

الخطوتان الأولى والثانية هما نفسهما كما في الحالة الأولى أما الخطوة الثالثة فتتمثل بجعل المؤشر first يشير إلى العقدة الجديدة أي n=5.

```
newptr->next=first;          first=newptr; // insert at first
```

```
void insert(int n)
{ Node *INN=new Node ;        INN->data=n;   Node *before ,*after=first;
if(INN->data < first->data)                                  // search if insert pos. in first
        {INN->next=first;       first=INN; }                 //insert in first list
else     {while(after!=NULL)
                    {if(INN->data < after->data)break;       // fined the position
                    { before =after; after=after->next;  }   // move to next node
                    }
if(after!=NULL) {INN->next=after;       before ->next=INN;     }
else                { before ->next=INN; }                   //insert in last list
        }
}
```

3- مدخل إلى اللوائح المترابطة 10

العملية delete: للحذف هناك أيضاً حالتان يمكن تمييزهما:

الأولى حذف عنصر له عنصر سابق O(n)

الثانية حذف العنصر الأول في اللائحة O(1).

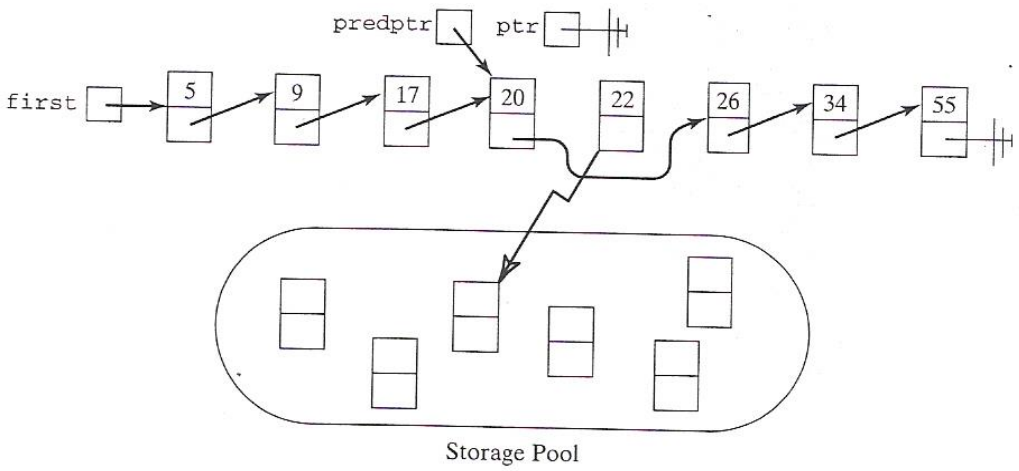
للحفاظ على الموارد تستخدام العملية delete لإعادة عقدة مشار إليها بمؤشر محدد إلى مجمع التخزين storage pool.

الحالة الأولى عنصر له عنصر سابق:

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

نجعل المؤشر ptr يشير إلى العقدة المراد حذفها، والمؤشر predptr يشير إلى العقدة السابقة لها (العقدة الحاوية على ptr->data =20).

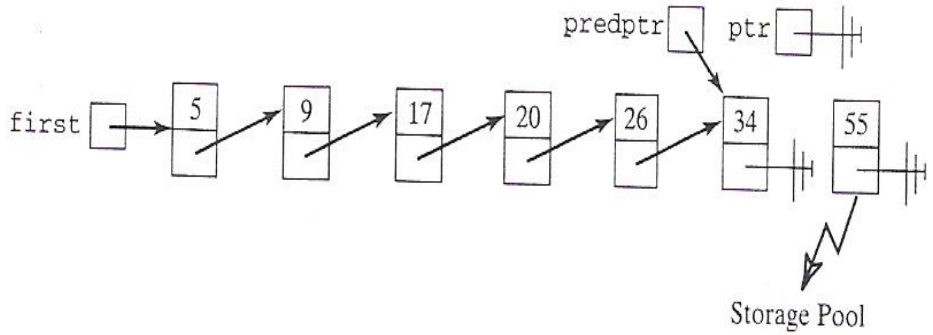
يمكن القيام بعملية الحذف من خلال عملية تمرير تجعل الرابط في العنصر السابق يشير إلى العنصر اللاحق للعقدة المراد حذفها.



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

predptr = ptr->next; delete ptr;

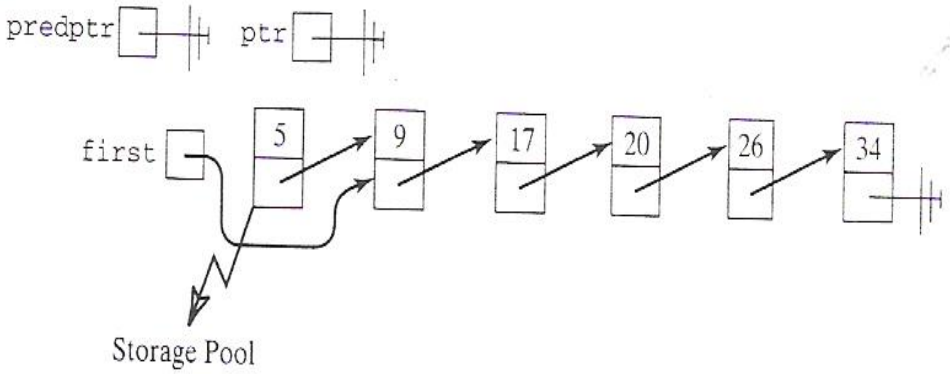
3- مدخل إلى اللوائح المترابطة 11



لاحظ أن هذه الخوارزمية تعمل في حال الحذف من نهاية اللائحة ايضاً وتكون عندما ptr->next =NULL.

وهي نفس الحالة السابقة.

predptr = ptr->next; delete ptr;



الحالة الثانية حذف العقدة الأولى:

سهلة لإن كل من first & ptr يشيران للعقدة الأولى وتقتصر على جعل المؤشر first يشير إلى العقدة الثانية في اللائحة .

first = first->next;

ومن ثم إعادة العقدة المحذوفة من اللائحة إلى مجمع التخزين.

delete ptr;

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.

انتهت المحاضرة الخامسة

applications of queues 1

3- تطبيقات على الأرتال1

```
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;}
        }
}
```

applications of queues 1

```
void sort( ) {Node *current=first,*tempptr;
  while(current!=NULL) {tempptr=current->next;
      while(tempptr!=NULL) {if(current->data>tempptr->data)
      {int temp=current->data;current->data=tempptr->data; tempptr->data=temp;} tempptr=tempptr->next; }
  current=current->next;}
          }          // end sort function
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; } // ins. firest
          else  {while(after!=NULL)  {if(insertNewNode->data < after->data)break;            // fined the position
                    {befor=after;        after=after->next; }                  }             // move to next node
          if(after!=NULL) {insertNewNode->next=after; befor->next=insertNewNode;}        //insert after node
          else{befor->next=insertNewNode;                                                 //insert in last list
                    }}
          }          // end insert function
```

```
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.sort();
        while(1) {cout<<"enter item to insert :";      cin>>t;   if(t==-1)break;    // end input data
        li.insert(t);                                                              // insert node
        cout<<"items are : ";          li.display();                               // print data
        cout<<"press n to finsh and ather key to contiue: ";
        cin>>ch;          if(ch=='n')break;                                        // end while loop
        }
} // end main link list and order the list then insert in list and display
```

3- تطبيقات على الأرتال1

applications of queues 1

مسألة التمارين الحسابية: بفرض أننا نريد تصميم برنامج لإنجاز تمارين على العمليات الحسابية الأساسية, وبشكل أكثر تحديداً, تمارين على إنجاز عملية الجمع لأعداد صحيحة مولدة بشكل عشوائي.
إذا قام الطالب بالإجابة بشكل صحيح يتم توليد مسألة جديدة, ولكن إذا أجاب بشكل خاطئ فإن المسألة يتم تخزينها بحيث يتم إعادة طرحها لاحقاً في نهاية الجلسة.

الأغراض والعمليات في المسألة: إن أحد الأغراض في هذه المسألة هو مسألة الجمع وبالتالي يمكن توصيفها على شكل صنف يدعى AdditionProblem له الصفات التالية:
البيانات الأعضاء: الأعداد المراد جمعها (addends) أعداد صحيحة عشوائية .
الناتج answer عدد صحيح.
العمليات: الحصول على مسألة عشوائية بعددين ضمن مجال معين.
عرض المسألة.
إيجاد جواب المسألة.
اختبار الجواب المعطى فيما إذا كان صحيحاً.
أما باقي الأغراض في المسألة فهي:

3- تطبيقات على الأرتال1

applications of queues 1

numProblem (int): عدد المسائل المراد طرحها.
maxAddend (int): أكبر عدد ممكن استخدامه ضمن مسألة الجمع.
userAnswer (int): إجابة المستخدم عن المسألة.
count (int): عدد المسائل المطروحة.
wrong (int): عدد المسائل التي تمت الإجابة عليها بشكل خاطئ للمرة الثانية.
cout (ostream): مجرى لإخراج النتائج, المسائل, الإحصائيات.
cin (istream): مجرى إدخال المسائل, الإجابات, وباقي المعلومات.

لإنجاز حل هذه المسألة سنستخدم غرضاً من الصنف queue لتخزين المسائل التي لم تتم الإجابة عليها بشكل صحيح يدعى wrongQueue بحيث يتم تخزين المسائل التي تتلقى إجابات خاطئة واحدة إثر الأخرى ومن ثم يتم إخراجها من هذا الرتل في نهاية الجلسة والإجابة عليها واحدة تلو الأخرى.
كما أننا نحتاج إلى آلية لتوليد الأعداد العشوائية, ويتم ذلك من خلال التابع ( )rand الموجود ضمن المكتبة القياسية.
من أجل الحصول على نمذجة أفضل للحل سنقوم بتعريف نمط بيانات يدعى RandomInt كصنف من خلال الملف الرأسي RandomInt.h التالي:

```
#ifndef RANDOMINT
#define RANDOMINT
#include <iostream>
#include <cstdlib>
#include <cassert>
#include <ctime>
using namespace std;
class RandomInt
{  public:       RandomInt();   RandomInt(int low, int high);
  RandomInt(int seedValue);     RandomInt(int seedValue, int low, int high);
  void Print(ostream & out) const;    RandomInt Generate();
  RandomInt Generate(int low, int high);    operator int();
 private:  void Initialize(int low, int high);
  void SeededInitialize(int seedValue, int low, int high);
  int NextRandomInt();
  int         myLowerBound,              myUpperBound,              myRandomValue;
  static bool generatorInitialized;
};
```

```
inline RandomInt::RandomInt(){  Initialize(0, RAND_MAX);}
inline RandomInt::RandomInt(int low, int high)
{  assert(0 <= low);  assert(low < high);  Initialize(low, high);}
inline RandomInt::RandomInt(int seedValue)
{  assert(seedValue >= 0);  SeededInitialize(seedValue, 0, RAND_MAX);}
inline RandomInt::RandomInt(int seedValue, int low, int high)
{  assert(seedValue >= 0);  assert(0 <= low);  assert(low < high);
  SeededInitialize(seedValue, low, high);}
inline void RandomInt::Print(ostream & out) const
{  out << myRandomValue;}
inline ostream & operator<< (ostream & out, const RandomInt & randInt)
{  randInt.Print(out);  return out;}
inline int RandomInt::NextRandomInt()
{  return myLowerBound + (rand() % (myUpperBound - myLowerBound + 1));}
inline RandomInt RandomInt::Generate()
{  myRandomValue = NextRandomInt();  return *this;}
inline RandomInt::operator int(){  return myRandomValue;}
#endif
```

يمكن تعريف التوابع الأعضاء في الصنف من خلال الملف RandomInt.cpp كما يلي:

```
 #include "stdafx.h"
#include "RandomInt.h"
using namespace std;
bool RandomInt::generatorInitialized = false;
void RandomInt::Initialize(int low, int high)
{  myLowerBound = low;  myUpperBound = high;
  if (!generatorInitialized)
    {  srand(long(time(0)));  generatorInitialized = true;    }
  myRandomValue = NextRandomInt();
}
void RandomInt::SeededInitialize(int seedValue, int low, int high)
{ myLowerBound = low;  myUpperBound = high;  srand(seedValue);
  generatorInitialized = true;  myRandomValue = NextRandomInt();
}
RandomInt RandomInt::Generate(int low, int high)
{  assert(0 <= low);  assert(low < high);  myLowerBound = low;  myUpperBound = high;
  myRandomValue = NextRandomInt();  return *this;}
```

3- تطبيقات على الأرتال1

applications of queues 1

يمكن أخيراً وضع خوارزمية حل مسألة التمارين الحسابية كما يلي:

1. قراءة numProblems و maxAddend.
2. من أجل count ضمن المجال 1 إلى numProblems قم بما يلي:
   a. قم بتوليد وعرض مسألة.
   b. اقرأ userAnswer.
   c. إذا كان userAnswer صحيح قم بعرض رسالة تفيد بأن الإجابة صحيحة وإلا قم بعرض رسالة تفيد بأن الإجابة غير صحيحة وأضف المسألة إلى wrongQueue.
3. قم بعرض التعليمات واجعل wrong=0.
4. طالما أن wrongQueue غير فارغ, قم بما يلي:
   a. اجلب واحذف مسألة من بداية الرتل.
   b. اعرض المسألة.
   c. اقرأ userAnswer.
   d. إذا كان userAnswer صحيح قم بعرض رسالة تفيد بأن الإجابة صحيحة وإلا قم بعرض رسالة تفيد بأن الإجابة غير صحيحة وقم بزيادة wrong بمقدار 1.
5. اعرض قيمة wrong.

بالتالي يصبح حل مسألة التمارين الحسابية كما يلي:

الملف AdditionProblem.h:

```
#include "RandomInt.h"
#include <iostream>
using namespace std;
#ifndef ADDITION_PROBLEM
#define ADDITION_PROBLEM
using namespace std;
class AdditionProblem
{/***** Function Members *****/
public: void Get(int maxAddend)
{  myAddend1.Generate(0, maxAddend);  myAddend2.Generate(0, maxAddend);}
void Display(ostream & out)
{  out << myAddend1 << " + " << myAddend2 << " = ? ";}
int Answer(){  return (myAddend1 + myAddend2);}
bool Correct(int userAnswer) {  return (userAnswer == Answer());}
/***** Data Members *****/
 private:    RandomInt myAddend1, myAddend2;
};
#endif
```

الملف driver.cpp للاختبار:

```
// queue4.cpp : main project file.
 #include "stdafx.h"
#include "AdditionProblem.h"
#include <queue>
#include <iostream>
using namespace std;
void main()
{  int numProblems,          maxAddend;
  cout << "How many problems would you like? ";          cin >> numProblems;
  cout << "What's the largest addend you would like? ";   cin >> maxAddend;

  queue<AdditionProblem>     wrongQueue;
  AdditionProblem problem;   int userAnswer;
  for (int count = 1; count <= numProblems; count++)
  {    problem.Get(maxAddend);     problem.Display(cout);    cin >> userAnswer;
    if (problem.Correct(userAnswer))        cout << "Correct!\n\n";
    else  {  cout << "Sorry -- Try again later\n\n";   wrongQueue.push(problem);    }
  }
```

```
cout << "\nIf you got any problems wrong, you will now be given"
          "\na second chance to answer them correctly.\n";

  int wrong = 0;
  while (!wrongQueue.empty())
  {    problem = wrongQueue.front();    wrongQueue.pop();    problem.Display(cout);
    cin >> userAnswer;
    if (problem.Correct(userAnswer))        cout << "Correct!\n\n";
    else
    {
      cout << "Sorry -- correct answer is " << problem.Answer() << "\n\n";
      wrong++;
    }
  }
  cout << "\nYou answered " << wrong << " problem"
       << (wrong > 1 ? "s ": " " ) << "incorrectly\n";
system("pause");
}
```

خرج التمرين

```
How many problems would you like? 4
What's the largest addend you would like? 199
175 + 183 = ? 200
Sorry -- Try again later

76 + 108 = ? 184
Correct!

117 + 13 = ? 130
Correct!

142 + 107 = ? 300
Sorry -- Try again later


If you got any problems wrong, you will now be given
a second chance to answer them correctly.
175 + 183 = ? 358
Correct!

142 + 107 = ? 200
Sorry -- correct answer is 249


You answered 1 problem incorrectly
Press any key to continue . . .
```