بسم الله الرحمن الرحيم

جامعة السّودان المفتوحة

برنامج علوم الحاسوب

هياكـــل البيانـــات

رمز المقرر ورقمه: حسب 201

تأليف: د. السماني عبد المطلب أحمد

التصميم التعليمـــي: أ. عبد الباسط محمّد شريف محمّد

التدقيق اللغـــوي: أ. السيد عوض الكريم الدوش

تصميم الأشكـــال: أ. عبد الباسط محمّد شريف محمّد

التصميم الفنـــي: نادر محمّد علي الأمين

منشـــورات جامعـــة السّـــودان المفتوحـــة، الطبعة الأولى 2005م.

جميع الحقوق محفوظة لجامعة السّودان المفتوحة، لا يجوز إعادة إنتاج أي جزء من هذا الكتاب، وبأي وجه من الوجـــوه، إلا بعد الموافقة المكتوبـــة من الجامعـــة.

مقدمة المقرر

الحمد لله، الذي خلق الإنسان، علمه البيان، وله الحمد أن رفع الذين آمنوا والذين أوتوا العلم درجات.

والصلاة والسلام على رسول الله المبعوث معلما للناس أجمعين، ورضي الله عن صحابته أجمعين، ومن تبعهم بإحسان إلى يوم الدين.

عزيزي الدارس بين يديك مقرر هياكل البيانات[1] (Data Structure) ويأتيك هذا المقرر محتوياً على بعض البنائيات البيانية البسيطة، أو المتغيرات البسيطة، والبنائيات المركبة مثل المصفوفات ومصفوفات السجلات، ولكن هناك بنائيات أكثر تعقيداً وتستطيع أن تتعامل مع التطبيقات بكفاءة عالية ولكنها غير موجودة في لغات البرمجة، لأنها تحتاج إلى برمجيات خاصة تمكن من التعامل معها.

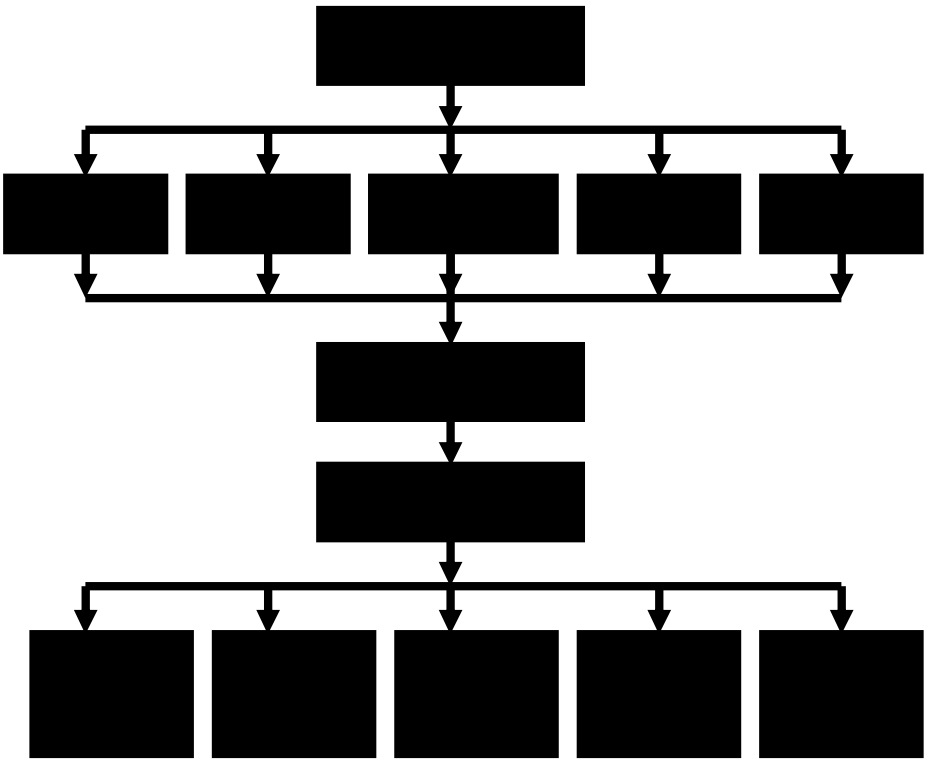
ويعتبر هذا المقرر أحد العلوم المهمة في علم الحاسوب إذ يهتم بموضوع تنظيم البيانات داخل الحاسوب وذلك من خلال اختيار طريقة مناسبة لتنظيم هذه البيانات إذ أن التنظيم يمثل معالجة استخدمها الإنسان علي البيانات التي يستقبلها في محيط حياته العادية فمثلاً عندما نتعامل مع بيانات الطلاب نضعها في تنظيم محدد يسهل علينا عملية الاسترجاع في أقل وقت ممكن ومن خلال جهد بسيط، لهذا فإن المبرمج يحتاج لإيجاد آلية أو تنظيم أو تراكيب وهياكل مناسبة للبيانات التي يتعامل معها بحيث يكون هذا التنظيم علي درجة عالية من الكفاءة خاصة إذا كان حجم البيانات التي يتعامل معها كبيراً هذا فضلاً عن القيود التي يفرضها الحاسوب عندما نتعامل مع وحدة المعالجة وحدة التخزين والقيود الخاصة بلغات البرمجة المختلفة.

مما سبق يمكن أن نقول إن هياكل البيانات هي عبارة عن مجموعة من عناصر البيانات التي لها تنظيم خاص تحكمه طرق وأساليب يمكن من خلالها ترجمة التصور المنطقي للبيانات إلي تمثيل مناسب في الذاكرة وبهذا الفهم يمكن أن نقسمها إلي قسمين أحدهما تطبيقي

---

[1] أفردت عدد من المراجع ترجمات مختلفة لعبارة **Data Structure** ولكن أشهرها هو هياكل أو تراكيب البيانات لهذا عندما يستخدم المؤلف علي صفحات هذا المرجع كلمة تراكيب أو هياكل يقصد نفس المعنى.

يتعلق ببناء البرامج التي تخلق الترابط بين عناصر الهيكل حتى نتمكن من البحث أو الوصول إلي هذه العناصر والشكل رقم (1) يوضح ذلك وسوف يركز هذا المرجع علي هذا الجانب بشي من التفصيل

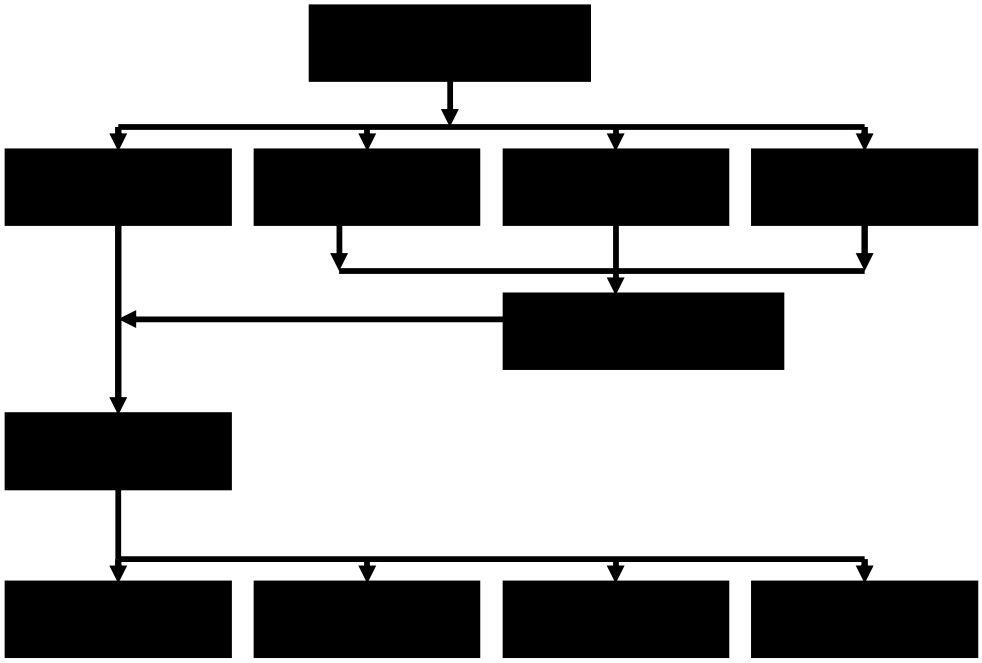


الشكل رقم (1) الجوانب المختلفة لدراسة هياكل البيانات التطبيقية

أما القسم الثاني فهو نظري يتعلق بالأسس المنطقية والرياضية التي تقوم عليها عملية هياكل البيانات وتحليل كفاءة هذه الهياكل من خلال الخوارزميات المستخدمة وهو يقع خارج نطاق هذا المقرر [2]. إلا فيما يتعلق بطرف البحث والترتيب والشكل رقم (2) يوضح ذلك.

---

[2]) صمم هذا المرجع ليغطي تفاصيل منهج هياكل البيانات ويوجد مقرر آخر يسمي تحليل وتصميم الخوارزميات يقوم بدراسة طـــرق وكفاءة الخوارزميات. لهذا سوف لا نتطرق لقياس كفاءة الطرق والخوارزميات المستخدمة.



الشكل رقم (2) الجوانب المختلفة لدراسة هياكل البيانات النظرية

تكمن أهمية هياكل البيانات في إيجاد علاقة منطقية بين الأشياء التي تمثلها باختلاف أنواعها وبالبحث عن أفضل استثمار للموارد الموجودة من حيث المساحة الكافية للتخزين ومن حيث الوقت اللازم للبحث عن البيان ومن خلال ذلك يمكن أن توفر لغات البرمجة عدداً من الهياكل علينا أن نختار الهيكل المناسب لحل المشكلة لهذا لابد من مراعاة الآتي عند اختيار الهيكل المناسب:

- حجم البيانات المتعلقة بالمشكلة وطبيعتها.
- مساحة الذاكرة التي يحتاجها الهيكل.
- الوقت اللازم للبحث عن عنصر محدد.
- مدي سهولة وترجمة الهيكل إلي جمل تسهل برمجتها بأحدى لغات البرمجة.

لقد رأينا أن لغات البرمجة تحتوي على بعض الهياكل البيانية البسيطة أو المتغيرات البسيطة التي تنطوي عن قيمة بيانية واحدة في أدنى مستوي للتعبير عن الأشياء والأحداث

المحيطة مثل القيم العددية الصحيحة والحقيقية والرموز والقيم المنطقية ولقد رأينا أيضاً الهياكل البيانية المركبة التي تنطوي عن أكثر من قيمة بيانية تحكمها صورة تنظيمية واحدة سواءً كان ذلك من الناحية المنطقية أو الناحية الفيزيائية في ذاكرة الحاسوب وخير مثال لذلك هو المصفوفات والسجلات والمجموعات وسلاسل الرموز ولكن هناك هياكل أكثر تعقيداً وتستطيع أن تتعامل مع بعض التطبيقات بكفاءة عالية ولكنها غير موجودة في لغات البرمجة لأنها تحتاج إلى برمجيات خاصة تمكن من التعامل معها وهذه الهياكل تعرف بهياكل البيانات المتجردة. إذن هياكل البيانات المتجردة هي عبارة عن هيكل بياني زائداً البرمجيات التي تمكن من التعامل معها ولما كانت العمليات التي تتم على الهياكل البيانية هي في الغالب إضافة وحدة بيانية أو حذفها لذا تصمم الهياكل البيانية المتجردة عادة لتكون مناسبة للطريقة التي يتم بها الإضافة والحذف.

سوف نتطرق خلال هذا الكتاب إلى طرق عديدة وأساليب متنوعة لهياكل البيانات.

لذا يتضمن هذا المقرر سبع وحدات تتناول الوحدة الأولى، النداء الذاتي والمؤشرات (وهي خلفية برمجية) وفي النداء الذاتي للدالة نوضح كيف أن الدالة تعيد تكرار نفسها أو تستدعى الدالة نفسها مجدداً حيث يتم استدعاءها داخل الدالة بنفس اسم الدالة إلى أن يتحقق شرط التوقف للدالة المستدعاة، أما في المؤشرات فنعتبر عرض عنوان المتغير في الذاكرة ليس مفيداً وإنما المفيد هو القيمة التي يحتلها هذا العنوان لهذا نلجأ إلي تعريف متغيرات تكون قادرة علي تخزين قيم العناوين وهذا المتغير يسمى متغير مؤشر أو مؤشراً.

في الوحدة الثانية نتاول عملية البحث والوصول إلى عنصر من عناصر البيانات وتحديد موضعه بغرض إجراء واحدة أو أكثر من المعالجات عليه (تعديل، حذف، إضافة، استرجاع). أما الوحدة الثالثة وهي بعنوان الترتيب ونعني بترتيب المعلومات وضع السجلات حسب ترتيب معين، ويعتبر ترتيب المعلومات من الاستخدامات المتكررة في معالجة البيانات. وتتناول الوحدة الرابعة أشكال البيانات المجردة وهي تنقسم إلى قسمين، القسم الأول يتحدث عن المكدسات وفيه نستخدم مبدأ الـ LIFO (Last In First Out) الداخل أخيراً هو الخارج أولاً. أما القسم الثاني فيتناول الصفوف وفيه نستخدم مبدأ الـ FIFO (First In First Out) العنصر الأول في الإدخال هو العنصر الأول في الإخراج. ينتقل بك المقرر بعد ذلك إلى

الوحدة الخامسة التي تحمل عنوان القوائم المتصلة وفيها شرح لفكرة إنشاء القوائم المتصلة والأسباب التي أدت إلى قيامها. الوحدة السادسة تتناول الأشجار وأنواعها وتقارن بينها.أما الوحدة الأخيرة في هذا المقرر (الوحدة السابعة) تتحدث عن المخططات وهي صورة موسعة للأشجار وذلك من خلال قلة القيود المفروضة على الأشجار وهي تستخدم لتمثيل بعض العلاقات الهرمية بين البيانات.

هذه المباحث التي حواها المقرر من الأهمية بمكان للدارس لعلوم الحاسوب إذ أنها تمثل الأسس التي ينطلق منها في دراسته وفهمه لهذا العلم..

الأهداف العامة للمقرر



بعد أن تفرع من دراسة هذا المقرر ينبغي أن تكون قادراً على أن:

1. **تُعرِّف** النداء الذاتي والمؤشرات وتحدد استخداماتهما.
2. **تُحدد** أشهر الطرق أو الخوارزميات التي تم تصميمها للبحث عن المعلومة **وتُوازن** بينها.
3. **تُعدد** مع الشرح المفصل خوارزميات الترتيب.
4. **تُعرف** المكدسات والصفوف **وتُوازن** بينهما.
5. **تشرح** فكرة إنشاء القوائم المتصلة والأسباب التي أدت إلى قيامها.
6. **تشرح وتُقارن** مستخدماً الرسم أنواع الأشجار.
7. **تُقارن** بين المخططات والأشجار.

محتويـــات المقـــرر

متسلسل	اسم الوحدة	الصفحة
1	**الوحدة الأولى:** النداء الذاتي والمؤشرات	1
2	**الوحدة الثانية:** البحث	21
3	**الوحدة الثالثة:** الترتيب	47
4	**الوحدة الرابعة:** أشكال البيانات المجردة	83
5	**الوحدة الخامسة:** القوائم المتصلة	127
6	**الوحدة السادسة:** الأشجار	157
7	**الوحدة السابعة:** المخططات	217

يحتوي المقرر على مجموعة من التدريبات في ثنايا الوحدات مع وجود حلول لها فـــي نهاية كل وحدة وهي وسيلة مهمة من وسائل عرض المادة وتطوير التفاعل والنقاش، ولها دور كبير في إقحامك في عملية التعلم، كما ستجد أسئلة للتقويم الذاتي في نهاية كل قسم وهي وسيلة فاعلة لتلخيص محتوى المادة وتقييم أدائك والرجوع إلى أجزاء المادة التي تكتشـــف قصـــوراً فيها.

وأخيراً نأمل أن نكون قد وفقنا في إعانتك على أَنْ تمتلـــك وتكتســـب مهـــارات لغـــة البرمجة، والتقدم بخطى ثابتة وواثقة في سبيل بلوغ الأهداف المنشودة.

الوحدة الأولى

النداء الذاتي والمؤشرات

(خلفية برمجية)



محتويات الوحدة

الموضوع	الصفحة
المقدمة	**3**
تمهيد	3
أهداف الوحدة	3
1. النداء الذاتي للدوال	**4**
2. المؤشرات	**8**
1.2 مقدمة عن المؤشرات	8
2.2 إدارة الذاكرة	13
1.2.2 العامل new	13
2.2.2 العامل delete	14
الخلاصة	**15**
لمحة مسبقة عن الوحدة التالية	**15**
إجابات التدريبات	**16**
مسرد المصطلحات	**19**
المراجع	**20**

المقدمة

تمهيد

عزيزي الدارس،

مرحبا بك إلى هذه الوحدة ، الوحدة الأولى

تحتوي هذه الوحدة على قسمين رئسين القسم الأول منها يتناول مفهوم النداء الذاتي للدوال وهو_كيفية قيام الدالة بتكرار نفسها أو استدعاء الدالة لنفسها مجدداّ، حيث ساعد استخدم النداء الذاتي للدوال في حل كثير من المسائل الرياضية بكثرة وذلك لكونها أنسب طريقة برمجية للتعامل مع جمع وضرب المتسلسلات وترتيبها.

القسم الثاني يتناول مفهوم المؤشرات وفيه نوضّح كيفية الوصول للقيّم مباشرة أو من خلال المؤشر الذي يؤشر بطريقة مباشرة إلي القيمة أي من خلال عنوان المتغّير.

وكذلك يشير القسم الثاني لكيفية إدارة الذاكرة واستخدام مساحة أكبر من تلك التي كانت تستخدمها المصفوفات.

أتمنى عزيزي الدارس، أن تقوم بحل التدريبات الموجودة في هذه الوحدة وأن تكثر من التدريب العملي لتكون الفائدة أكبر.

أهداف الوحدة

عزيزي الدارس،

بعد فراغك من هذه الوحدة ينبغي أن تكون قادرا على أن:

1. **تَسْتَخدم** النداء الذاتي للدالة.
2. **تَتَعرّف** على المؤشرات وكيفية استخدامها.
3. **تُدِير** الذاكرة والتحكم في حجم المصفوفة.

1. النداء الذاتي للدوال Recursion function

النداء الذاتي للدالة هو أن تعيد الدالة تكرار نفسها أو تستدعي الدالة نفسها مجدداً حيث يتم استدعاءها داخل الدالة بنفس اسم الدالة إلى أن يتحقق شرط التوقف للدالة المستدعاة وهنالك بعض الملاحظات التي يجب أن تذكر وهي:

- تستدعى الدالة نفسها داخل الدالة بواسطة معاملات مختلفة في كل مرة يتم الاستدعاء.
- في النداء الذاتي للدوال إذا لم يتم وضع شرط التوقف فستستدعي الدالة نفسها إلى ما لا نهاية **function will call itself forever**
- عند اكتمال آخر استدعاء للدالة يقوم المترجم بإرجاع كل القيم التي نتجت عند الاستدعاء الذاتي وتبدأ بآخر قيمة نتجت عند الاستدعاء حتى أول قيمة.

استخدم النداء الذاتي للدوال في حل كثير من المسائل الرياضية بكثرة وذلك لكونها أنسب طريقة برمجياً للتعامل مع جمع وضرب المتسلسلات وترتيبها.

مثال (1)

اكتب برنامجاً لإيجاد مضروب أي عدد صحيح **N** وذلك من خلال

1. استخدام الدوال العادية
2. استخدام النداء الذاتي

من خلال استخدام الدوال العادية

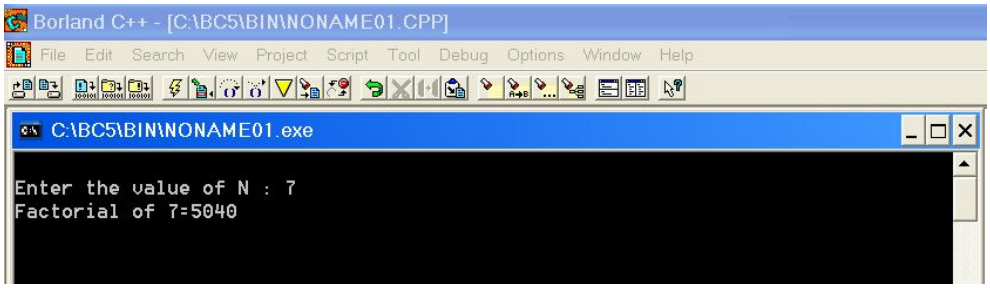
```
#include<iostream.h>
#include<conio.h>
int fact(int n)
{
     int F=1;
     for (int i=n;i>=1;)
         F*=i--;
```

```
    return F;
}
void main()
{
    int N;
    cout<<"\nEnter the value of N : ";
    cin>>N;
    cout<<"Factorial of "<<N<<"="<<fact(N);
    getch();
}
```

ثانياً : من خلال استخدام النداء الذاتي للدوال

```
#include<iostream.h>
#include<conio.h>
int fact(int n)
{
    if (n <=1)
        return 1;
    else
        return (n*fact(n-1));
}
void main()
{
    int N;
    cout<<"\nEnter the value of N : ";
    cin>>N;
    cout<<"Factorial of "<<N<<"="<<fact(N);
    getch();
}
```

وعليه يكون إخراج البرنامج على النحو التالي:



يتم إرسال العدد المراد إيجاد المضروب له إلى الدالة والتي بدورها تستقبله في متغير من النوع **(intn)** داخل الدالة، إذا كان هذا العدد أصغر أو يساوي واحداً فإن الدالة ستعيد قيمة المضروب **(1)** إما إذا كان أكبر من الواحد فإنها ستعيد حاصل ضرب **(n*fact(n-1))**

وبما إننا ذكرنا اسم الدالة مرة ثانية داخل الدالة نفسها لكن مع ملاحظة أن القيمة المرسلة إلى الدالة هذه المرة أقل من القيمة المرسلة إلى الدالة من البرنامج الرئيس فسيتم استدعاء الدالة مرة أخرى وهكذا إلي أن ترسل الدالة قيمة واحد إلي الدالة نفسها وعندئذ يتحقق شرط التوقف.

مثال (2)

اكتب برنامجاً لحساب فابونانسي **Fibonacci number** وذلك من خلال العلاقة التالية

F(0) = 0

F(1) = 1

F(n) = F(n-1) + F(n-2)

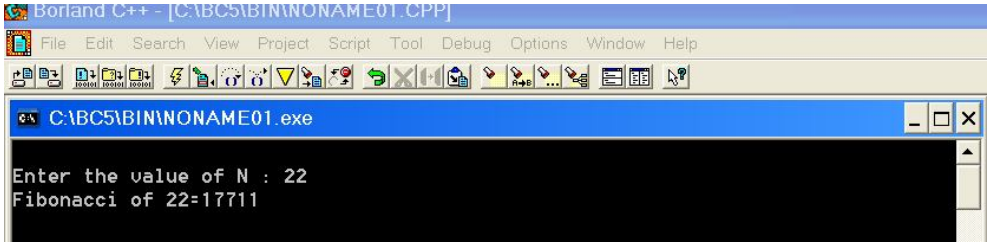
```
#include<conio.h>
#include<iostream.h>
int fib(int n)
{
     if (n==0||n==1)
          return n;
     else
          return fib(n-1)+fib(n-2);
}
void main()
```

```
{
     int N;
     cout<<"\nEnter the value of N : ";
     cin>>N;
     cout<<"Fibonacci of "<<N<<"="<<fib(N);
     getch();
}
```

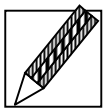
نلاحظ عند استدعاء الدالة في البرنامج الرئيس من خلال عبـــارة **Fib(N)** نرســل للدالة العدد **N** وتستقبله في المتغير **n** ثم بعد ذلك يتم اختبار الشرط هل هذا العدد يساوى **0** أو **1** فإذا كانت الإجابة بنعم فان الدالة سوف تعيد هذه القيمة إلى البرنامج الرئيس أما إذا كانـــت اكبر منهم فإنها سوف تستدعي الدالة مرة أخرى من خلال التعبير التالي:

```
return fib(n-1)+fib(n-2);
```

هنا الاستدعاء الذاتي للدالة قد تم أكثر من مرة وفى كل مرة بمعامـــل مختلـــف عـــن الأخرى. وعلية يكون إخراج البرنامج علي النحو التالي:



تدريب (1)



اكتب برنامجاً لإعطاء التركيبات المختلفة لحروف الكلمة الواحدة **(Anagram)** باستخدام النداء الذاتي.

أسئلة تقويم ذاتي

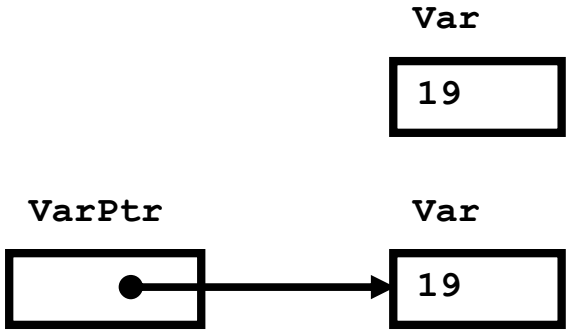


1. عرّف النداء الذاتي للدالة.
2. متى تستدعي الدالة نفسها إلى ما لانهاية؟
3. لماذا يستخدم النداء الذاتي للدوال في حل كثير من المسائل الرياضية؟

2. المؤشرات Pointers

1.2 مقدمة عن المؤشرات

المؤشرات هي عبارة عن متغيرات تحتوي علي عناوين في الـذاكرة كقـيم مخزنـة ضمنها فالمتغير العادي عادة يحتوي علي قيمة مخزنة مباشرة ضمنه بينما المؤشـر يتضـمن عنواناً في الذاكرة لمتغير يحتوي علي قيمة وعلية يمكن الوصول إلي هذه القيمة مـن خـلال المتغير ويسمي بالوصول المباشر للقيمة أو من خلال المؤشر الذي يؤشر بطريقة مباشرة إلـي القيمة أي من خلال عنوان المتغير والشكل رقم (1) يبين طريقة الوصـول المباشـر وغيـر المباشر لقيم المتغيرات.



الشكل رقم (1) طريقة الوصول المباشر والغير مباشر لقيمة المتغير`Var`.

كما هو معروف لدينا أن لكل بايت في الذاكرة عنواناً هو عبارة عن أرقـام وأن كـل برنامج عند وضعه في الذاكرة يحتل نطاقاً معيناً من تلك العناوين ولهذا لكل متغير ودالة فـي

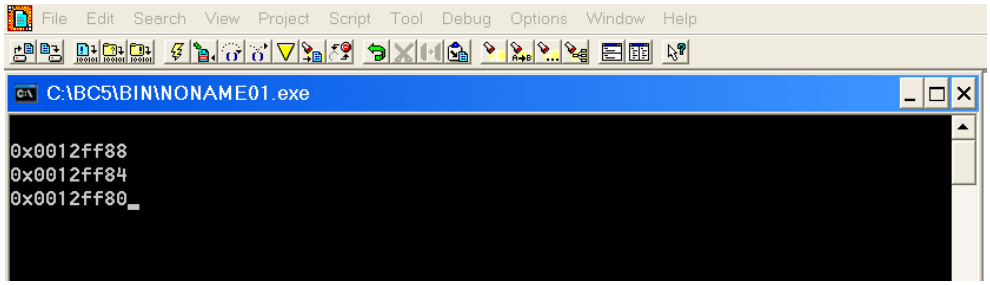
البرنامج عنوان[1]. يمكن معرفة العنوان الذي يحتله متغير ما في لغة **C++** باستعمال عامـــل العنوان **&**.

مثال (3)

اكتب برنامجاً يعرف 3 متغيرات ويعطيها القيم 11، 22، 33 على التوالي ثم يعـــرض عناوينها.

```
#include <iostream.h>
#include <conio.h>
void main()
{
     int var1 =11;
     int var2 =22;
     int var3 =33;
     cout<<endl<<&var1
          <<endl<<&var2
          <<endl<<&var3;
     getch();
}
```

إخراج البرنامج:



1) العناوين في الأساس مرقمة من أعلى إلي أسفل أي تظهر بترتيب تنازلي لأن المتغيرات التلقائية يتم تخزينها في المكدس الذي ينمو إلى الأسفل في الذاكرة. أما لو كانت المتغيرات خارجية (معرفة خارج أي دالة) لحصلت المتغيرات على عناوين تصاعدية لأنه يتم تخزين المتغيرات الخارجية في الذاكرة التكويمية.

تعتمد العناوين التي تحتلها متغيرات البرنامج على عدة أمور منها:

- الجزء من الذاكرة الذي يحتله البرنامج.
- حجم نظام التشغيل.
- وجود برامج أخرى تعمل في الذاكرة أم لا.

كل هذه الأسباب قد تؤدي للاختلاف في نتائج البرنامج عند التنفيذ من شخص لآخر ونلاحظ أيضاً أن العناوين تكتب بالتدوين الست عشري والذي يشار إليه ب **0x** الظاهرة قبل كل رقم كل عنوان في إخراج البرنامج أعلاه.

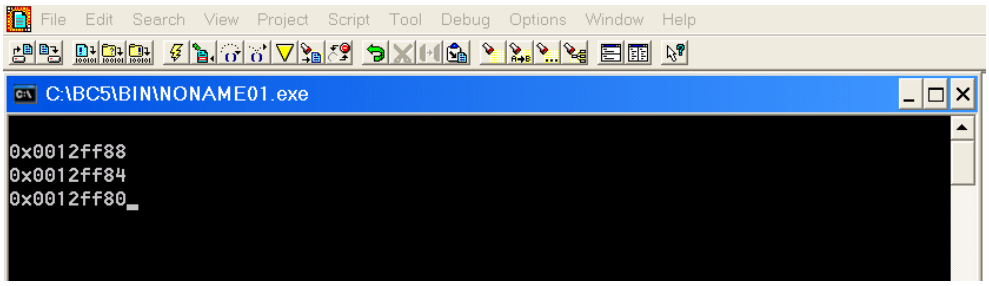
يعتبر عرض عنوان المتغير في الذاكرة ليس مفيداً وإنما المفيد هو القيمة التي يحتلها هذا العنوان لهذا نلجأ إلي تعريف متغيرات تكون قادرة علي تخزين قيم العناوين وهذا المتغير يسمى متغير مؤشر أو مؤشراً. إن المؤشر ليس كالمتغير الذي يتم تخزين عنوانه انه مؤشر إلى النوع **int** وهو في الأصل ليس من النوع **int.**

مثال (4)

أعد كتابة البرنامج الموجود بالمثال رقم (3) ليعطي عنوانين المتغيرات من خلال المؤشرات

```
#include <iostream.h>
#include <conio.h>
void main()
{
     int var1 =11;
     int var2 =22;
     int var3 =33;
     int* ptr;
     ptr = &var1;
     cout<<endl<<ptr;
     ptr = &var2;
     cout<<endl<<ptr;
     ptr = &var3;
     cout<<endl<<ptr;
     getch();
}
```

إخراج البرنامج يكون علي النحو التالي:



وهو يشبه إخراج البرنامج الموجودة بالمثال رقم (3) ونلاحظ في البرنامج أنه تم الإعلان عن متغير مؤشر من خلال التعبير

```
int* ptr ;
```

تعرف تلك العبارة المتغير **ptr** كمؤشر إلى النوع **int** وبعبارة أخرى نقول إن هذا المتغير يمكنه تخزين عناوين متغيرات عددية صحيحة. وبالتالي يمكن أن نعلن عن مؤشرات من أنواع أخري مثل التعابير التالية:

```
char*  cptr ;          //pointer to char
int*  iptr;            //pointer to int
float*  fptr;          //pointer to float
```

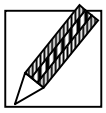
أن كتابة النجمة قريبة من النوع أو من المؤشر يعد أمر غير مهم بالنسبة للمترجم لكن وضعها بالقرب من النوع يساعد على التشديد أنها جزء من النوع وليست جزء من الاسم نفسه. العبارة التالية توضح تعريف أكثر من مؤشر في نفس السطر.

```
int *ptr1,  *ptr2, *ptr3;
```

يمكن أن يخزن المؤشر عنوان أي متغير من أي نوع لكن يجب إعطائه قيمة ما وإلا سيشير المؤشر إلى عنوان لا يرغب المستخدم في الإشارة إليه، كالإشارة إلى شفرة البرنامج أو إلى نظام التشغيل ويمكن أن تؤدي قيم المتغيرات المتشردة إلى تعطل النظام كما أن اكتشافها أمر صعب لأن المترجم لا يعطي تنبيه حولها.

في حالة عدم معرفة اسم المتغير ولكن معرفة عنوانه فيمكن الوصول إلى محتوياتـــه. والتركيب النحوي الخاص بمعرفة قيمة المتغير من خلال معرفة عنوانه بدلاً من اسمه تتضـــح في البرنامج التالي:

تدريب (2)



أعد كتابة البرنامج الموجود بالمثال رقم (3) ليعطي قيم المتغيرات من خلال استخدام المؤشرات.

مثال (5)

اكتب برنامجاً يستعمل مؤشر لتعيين قيمة إلى متغير ثم تعيين تلك القيمة إلـــى متغيـــر آخر:

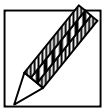
```
#include<iostream.h>
#include<conio.h>
void main()
{
     int var1, var2;
     int* ptr ;
     ptr  =&var1;
     *ptr = 37;
     var2 = *ptr;
     cout <<endl <<var2;
     getch();
}
```

تقويم ذاتي



1. عرّف المؤشرات
2. ما الفرق بين المتغيّر والمؤشر؟

تدريب (3)



اكتب برنامجاً يستقبل 4 متغيرات ثم يقوم بطباعة عناوينها.

2.2 إدارة الذاكرة بواسطة delete و new

قد رأينا أن أكثر الآليات التي تستخدم لتخزين عدد كبير من المتغيرات أو الكائنات هي المصفوفة وقد صاحب استعمال المصفوفات بعض من القصور يتمثل في حجم المصفوفة ثابت في معظم الأحيان وقد لا يعرف المستخدم كمية الذاكرة التي يحتاج إليها أثناء تشغيل البرنـــامج لهذا وفرت لغة **C++** أسلوباً لتوفّر مساحة من الذاكرة وذلك من خلال العامل **new**.

1.2.2 العامل new

تزود لغة C++ أسلوباً آخر للحصول على مساحة من الذاكرة يخصص هذا العامـــل ذاكرة ذات حجم معين ويعيد مؤشراً. البرنامج التالي يبين استعمال العامـــل **new** للحصـــول على ذاكرة لسلسلة :

```
#include<iostream.h>
#include<conio.h>
void main()
{

     char* str="computer science";
     int len=strlen(str) ;
     char*  ptr;
     ptr=new char [len +1];
     strcpy (ptr,str) ;
     cout<<"ptr ="<<ptr;
     delete[] ptr;
     getch();
}
```

الكلمة الأساسية **new** يليها نوع المتغيرات التي سيتم تخصيصها وعدد تلك المتغيرات ونقوم هنا بتخصيص المتغيرات من نوع **char** ونحتاج **len+1** حيث تساوي طول السلسلة str وهو أمر معروف من خلال الدالة المكتبية **strcpy()**. يعيد العامل **new** مؤشراً يشـــير إلى بداية قطعة الذاكرة ويحصل على الذاكرة ديناميكياً أي أثناء تنفيذ البرنامج

2.2.2 العامل delete

يرافق العامل **new** العامل **delete** يعيد تحرير الذاكرة إلى نظام التشغيل. العبارة التالية تعيد إلى النظام كمية الذاكرة التي يشير إليها المؤشر **ptr**.

```
Delete [] ptr;
```

الخلاصة

عزيزي الدارس، نلخّص وحدتنا في أننا قمنا باستعراض ما سبق وتم دراسته في مقرر أساليب البرمجة، و هي بمثابة خلفية برمجية تشمل النداء الذاتي للدوال وهو أن تعيد الدالة نفسها مجدداً حيث يتم إستدعائها بنفس اسمها.

كذلك تحدثنا في الجزء الثاني عن المؤشرات وذكرنا بأنها متغيرات تحتوي على عناوين في الذاكرة كقيم مخزنة ضمنها. وذكرنا بأن المتغير العادي يحتوي على قيمة بينما المؤشر يتضمن عنواناً في الذاكرة لمتغير يحتوي على قيمة.

وأخيراً تحدثنا عن إدارة الذاكرة بواسطة new , delete لتوفير مساحة أكبر من الذاكرة.

أتمنى عزيزي الدارس أن تكون قد فهمت هذه الوحدة، والتي تعتبر دراستها مدخلاً مهماً لهذه المادة ، مع أمنياتنا بالتوفيق.

لمحة مسبقة عن الوحدة التالية

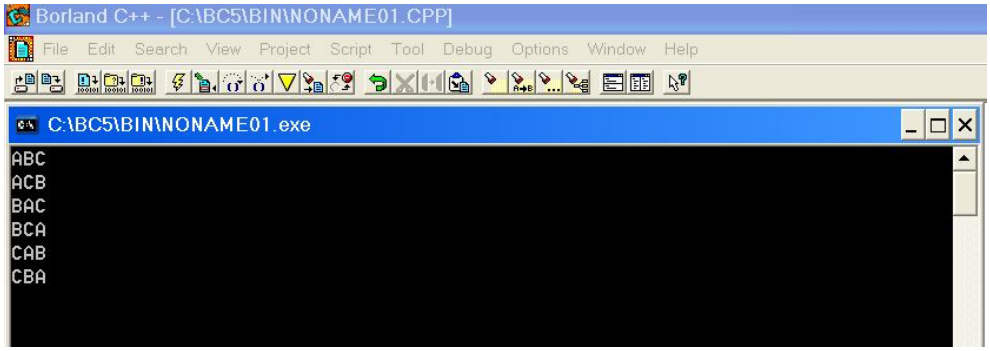
سوف نتطرّق بإذن الله في الوحدة القادمة إلى البحث والذي يشير إلي الوصول إلى عنصر من عناصر البيانات و وتحديد موقعه بغرض إجراء واحدة أو أكثر من المعالجات عليه (تعديل، حذف، إضافة، استرجاع).

إجابات التدريبات

تدريب (1)

```
#include<iostream.h>
#include<conio.h>
const size=4;
void permute(char data[],int pos)
{
   char new_data[size];
   char hold;
    for(int i=0;i<size;i++)
    new_data[i]=data[i];
    if (pos >= size-1)
    {
        for(int i=0;i<size;i++)
           cout<<data[i];
        cout<<endl;
    }
    else
    {
         for(int n=pos ;n<size;n++)
         {
              hold=new_data[n];
              new_data[n]=new_data[pos];
              new_data[pos]=hold;
              permute(new_data,pos+1);
         }
    }
 }
void main()
{
     char array[size]={'A','B','C','D'};
     int pos =0;
     permute(array,pos);
     getch();
}
```

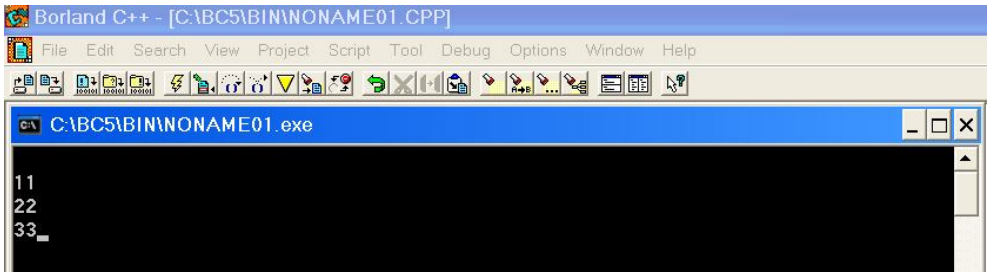
وبالتالي يكون الإخراج علي النحو التالي



تدريب (2)

```
#include <iostream.h>
#include <conio.h>
void main()
{
     int var1 =11;
     int var2 =22;
     int var3 =33;
     int* ptr;
     ptr = &var1;
     cout<<endl<<*ptr;
     ptr = &var2;
     cout<<endl<<*ptr;
     ptr = &var3;
     cout<<endl<<*ptr;
     getch();
}
```

ويكون الإخراج على النحو التالي:



تدريب (3)

```
# include <iostream.h>
# include < conio.h>
Void main ( )
{
  int var1, var2, var3, var4 ;
  cout<<" \n Enter first number" ;
  cin>> var1 ;
  cout<<" \n Enter second number" ;
  cin>> var2 ;
  cout<<" \n Enter third number" ;
  cin>> var3 ;
  cout<<" \n Enter fourth number" ;
  cin>> var4 ;
  cout<<endl<<&var1 ;
  cout<<endl<<&var2 ;
  cout<<endl<<&var3;
  cout<<endl<<&var4 ;
     getch( ) ;
}
```

مسرد المصطلحات

- **النداء الذاتي للدوال Recursion Function**

هو أن تعيد الدالة تكرار نفسها أو تستدعي الدالة نفسها مجدداً حيث يتم استدعائها داخل الدالة بنفس اسم الدالة.

- **المؤشرات Pointers**

هي عبارة عن متغيرات تحتوي على عناوين في الذاكرة كقيم مخزنة ضمنها.

- **مصفوفة Array**

عبارة عن متغيّر مركّب يتكوّن من عدة عناصر كلها من جنس واحد تشترك في الاسم وتختلف في الموقع داخل المصفوفة.

المراجع

1. William Ford and William Topp, **Data Structure with C++**, Prentice-Hall Inc 1996.
2. Sara Baase ad Allen Van Gelder, **Computer algorithms**, Addison Wesley Longman 2000.
3. Deitel and Deital, **C++ How to program**, Prentice-Hall Inc 2001.

الوحدة الثانية

البحث



محتويات الوحدة

الموضوع	الصفحة
المقدمة	**23**
تمهيد	23
أهداف الوحدة	24
1. مفهوم البحث	**25**
2. طرق البحث	**26**
1.2. البحث المتتالي	26
2.2. البحث الثنائي	30
الخلاصة	**38**
لمحة مسبقة عن الوحدة التالية	**38**
إجابات التدريبات	**39**
مسرد المصطلحات	**44**
المراجع	**45**

المقدمـــة

تمهيـــد

عزيزي الدارس،

مرحبا بك إلى هذه الوحدة

تحتوي هذه الوحدة على أقسام رئسين القسم الأول منها يتناول مفهوم البحث وفيه نشير لأهمية البحث للوصول إلى عنصر من عناصر البيانات لتحديد موقعه بغرض إجراء واحدة أو أكثر من المعالجات عليه (تعديل، حذف، إضافة، استرجاع).

القسم الثاني يتناول بعض طرق البحث التي تم تصميمها للبحث عن السـجل وأشـهر هذه الطرق:

البحث المتتالي أو الخطي: وهي طريقة ذات كفاءة متدنية ولكنها شديدة السـهولة فـي عملية البحث عن السجل، وفيها نستخدم الخوارزميات للبحث عن عنصر من قائمـة، وأيضـاً سنتعرف على كيفية كتابة برنامجاً يقوم بالبحث عن عنصر في قائمـة اختيـرت عناصـرها بصورة عشوائية.

ثم ينتقل بك القسم إلى الطريقة الثانية وهي:

طريقة البحث الثنائي: وتستخدم هذه الطريقة لزيادة كفاءة البحث مـن خـلال زيـادة سرعة الوصول إلى السجل، ونستخدم فيها خوارزمية البحث الثنـائي للبحـث عـن عنصـر عشوائي محصور بين عددين في قائمة مرتبة عناصرها، وكذلك سنتعرف على كيفيـة كتابـة برنامجاً يقوم بالبحث عن عنصر عشوائي محصور بين عددين في قائمة مرتبة عناصرها.

أرجو أن تفيد من التدريبات التي سترد بين ثنايا المادة وأن تتعامل معها بجدية، لما لها من دور كبير في تثبيت المادة وتعزيز ما ستدرسه من مفاهيم.

أرجو أن تجدها وحدة دراسية نافعة جداً.

أهداف الوحدة



عزيزي الدارس،،

بعد فراغك من هذه الوحدة ينبغي أن تكون قادراً على أن:

1. **تُعدد** أشهر الخوارزميات التي تم تصميمها للبحث عن السجلات **وتُوازن** بينها.
2. **تذكر** القانون الذي تقاس به فعالية طريقة البحث المتتالي.
3. **تستخدم** خوارزمية البحث المتتالي للبحث عن عنصر من قائمة بها عناصر متعددة.
4. **تكتب** برنامجاً يقوم بالبحث عن عنصر في قائمة اختيرت عناصرها بصورة عشوائية ومحصورة بين عددين.
5. **تشرح** بالتفصيل خوارزمية البحث الثنائي.
6. **تستخدم** خوارزمية البحث الثنائي للبحث عن عنصر عشوائي محصور بين عددين في قائمة مرتبة عناصرها.
7. **تُقارن** بين طريقة البحث المتتالي والبحث الثنائي.

1. مفهوم البحث Searching Concept

أعتقد أنك تدرك قيمة البحث للوصول إلى عنصر من عناصر البيانات لتحديد موقعه بغرض إجراء واحدة أو أكثر من المعالجات عليه (تعديل، حذف، إضافة، استرجاع). وسوف نختصر في هذه الوحدة للتحدث عن خوارزميات البحث عن سجل في مصفوفة السجلات ثم البحث عن المعلومة في أحد حقول ذلك السجل ويتم ذلك عادة بالبحث عبر حقل محدد من حقول السجل والذي يسمي بالمفتاح **(Key)** : هو حقل مفرد للتعبير عن السجل مثال لذلك حقل رقم الطالب هو الحقل الذي يتم به البحث عن سجل الطالب وبعد الوصول إلى سجل الطالب يمكن النظر إلى الحقول الأخرى للطالب. يسمي حقل رقم الطالب بالمفتاح.

ثم بعد ذلك يتم التعامل مع أي حقل آخر من حقول السجل أو مع كل حقول ذلك السجل.

وعندما نتحدث عن البحث في المصفوفة فإننا نتحدث عن المصفوفة المرتبة تصاعدياً أو تنازلياً ، وبذلك نستفيد من كونها مرتبة ، حيث ينتهي عملنا عندما تكون قيمة عنصر من عناصر المصفوفة أكبر من أو تساوي القيمة قيد البحث (الهدف) ، وذلك عندما تكون المصفوفة مرتبة تصاعدياً ، وليس هناك داع لفحص باقي المصفوفة ، لأن باقي عناصرها ستكون بالتأكيد أكبر من القيمة المطلوبة .

سنتعرض في هذه الوحدة إلى نوعين من أنواع البحث الخاصة بالمصفوفات وهما:

- البحث المتتالي أو الخطي
- البحث الثنائي

أسئلة تقويم ذاتي



1.في ضوء ما درست في هذا القسم كيف تعرف البحث؟

2.عرف المفتاح مع ذكر مثال له؟

3. ماذا نعني عندما نتحدث عن البحث في المصفوفة ؟

2. طرق البحث

من المهم هنا أن نذكر الطرق أو الخوارزميات التي تم تصميمها للبحث عن المعلومة وهي شتى وكثيرة ولكن دعنا نأخذ أشهر هذه الطرق على سبيل المثال لا الحصر:

- البحث المتتالي أو الخطي.
- البحث الثنائي.
- شجرة البحث الثنائية.

1.2 البحث المتتالي Sequential Search Technique

لعلك تتسأل ــ عزيزي الدارس ــ ما المقصود بالبحث المتتالي وما الفائدة منه؟

نهدف من البحث المتتالي إلى إحداث مطابقة بين قيمة العنصر المراد البحث عنه (**SearchKey**) وبين بقية العناصر الأخرى بالمصفوفة (**Array**)، ولكي يتم البحث عن هذا العنصر لابد من المرور علي كل العناصر في المصفوفة، وإذا تم الوصول إلي نهاية المصفوفة دون الوصول إلي مطابقة، فهذا يعني عدم وجود العنصر داخل المصفوفة.

الخوارزمية:

1. أدخل عدد عناصر المصفوفة **N** .
2. أدخل عناصر المصفوفة **Array[N]** .
3. أدخل العنصر المطلوب البحث عنه **SearchKey**.
4. أجعل **pos=0** وهو موقع العنصر المطابق للعنصر المطلوب .
5. **if Array[pos]=SearchKey then goto 8**
6. **pos=pos+1**.
7. **if pos>N then goto 10 else goto 5**
8. طباعة رسالة تفيد بأن **SearchKey** موجود في الموقع **pos+1**
9. **goto 11**
10. طباعة رسالة تفيد بأن **SearchKey** غير موجود
11. **End**.

أسئلة تقويم ذاتي



1. عدد الطرق أو الخوارزميات التي تم تصميمها للبحث عن المعلومة؟
2. إلى ماذا يهدف البحث المتتالي؟
3. متى نعتبر العنصر غير موجود داخل المصفوفة؟
4. عدد خطوات خوارزمية البحث المتتالي؟

عزيزي الدارس هل ترى أن هذه الطريقة من طرق البحث ذات سرعة وكفاءة عالية في الوصول للسجل المطلوب؟

دعنا نجيب على هذا التساؤل من خلال الأمثلة القادمة، وقبل كتابة الأمثلة يجب أن تعلم أن فعالية (Efficiency) هذه الطريقة تقاس بالقانون:

```
Search Length(SL) = N/2.
```

ولمعرفة زمن الاستجابة نستخدم القانون:

```
Response time= SL*time of record search.
```

مثال (1)

احسب زمن الاستجابة إذا كان لدينا ملف به **1000** سجل وبزمن قدره **0.0005** ثانية

```
Efficiency = 1000/2 = 500.
Response time =500*0.0005=0.25 second.
```

مثال (2)

استخدم خوارزمية البحث المتتالي ابحث عن العنصر **43** إذا كان لدينا قائمة بالعناصر التالية :

24 31 61 16 55 17 43 54 47 11 38 49

(الحـــل)

0	24
1	31
2	61
3	16
4	55
5	17
6	43
7	54
8	47
9	11
10	38

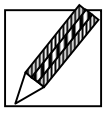
حسب الخوارزمية في البداية نجعل **pos=0** ثم بعد ذلك تتم مقارنة **SearchKey** وهي 43 بأول عنصر في القائمة وهو 24 فنجد أنه لا يساويه فننتقل إلي العناصر

SearchKey=43

N=10

الثاني في القائمة وهو 31 ونجد أنه لا يساويه وتتابع المقارنة مع بقية العناصر حتى نصل إلي العنصر الذي يطابق **SearchKey** وهو العنصر رقم **7 (pos+1)** لو قمنا بتغير قيمة **SearchKey** لكي تصبح 50 فنلاحظ أن عملية البحث سوف تستمر حتى نهاية القائمة دون أن نجد العنصر المطلوب.

تدريب (1)

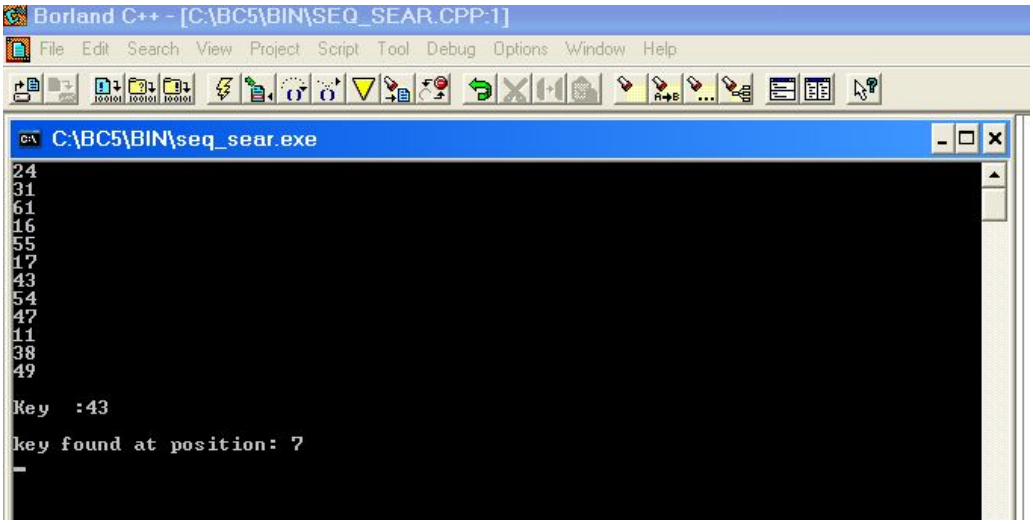


اكتب برنامجاً يقوم بالبحث عن العنصر34 في قائمة التي عناصرها **24,31,61,16,55,17,43,54,47,11,38,49** مستخدماً خوارزمية البحث المتتالي.

أعتقد أنك تدرك الآن أن هذه الطريقة من طرق البحث ذات كفاءة متدنية ولكنها شديدة السهولة وتكون هذه الطريقة عملية عندما يكون حجم البيانات صغيراً .

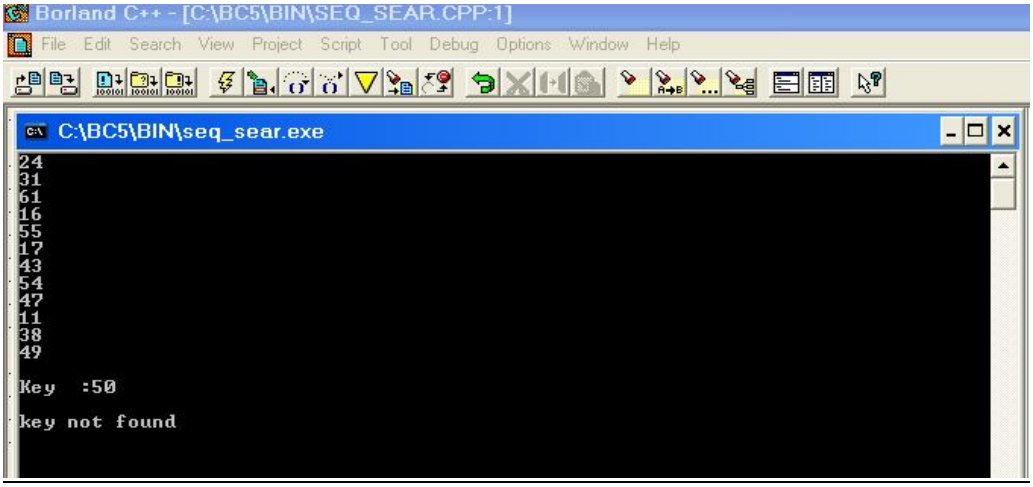
إنناً عندما نتحدث عن كفاءة أو سرعة هذه الخوارزمية، فإننا نجد أن متوسط عدد المقارنات يساوي **N/2** (تعتبر أحسن حالة ممكنة لهذه الخوارزمية عندما يكون العنصر المطلوب هو العنصر الأول وبالتالي فإن الخوارزمية تكون قد اكتفت بمقارنة واحدة فقط وأما أسوأ حالة هي أن يكون العنصر الأخير هو العنصر المطلوب وعليه يكون في هذه الحالة عدد المقارنات يساوي N). إذاً تدني سرعة الوصول إلى السجل هو السمة الملازمة لهذه الطريقة علما بأن هذا البطء يزداد بزيادة حجم البيانات

الشكل رقم (1) يوضح إخراج البرنامج أعلاه الذي يبين أن العنصر43 موجود بالقائمة في الموقع رقم 7.



الشكل رقم (1):إخراج البرنامج بالمثال رقم (2)

في حالة عدم وجود المفتاح ضمن القائمة وذلك خلال تغير قيمة المفتاح بالقيمة يكون إخراج البرنامج بالشكل رقم (2)

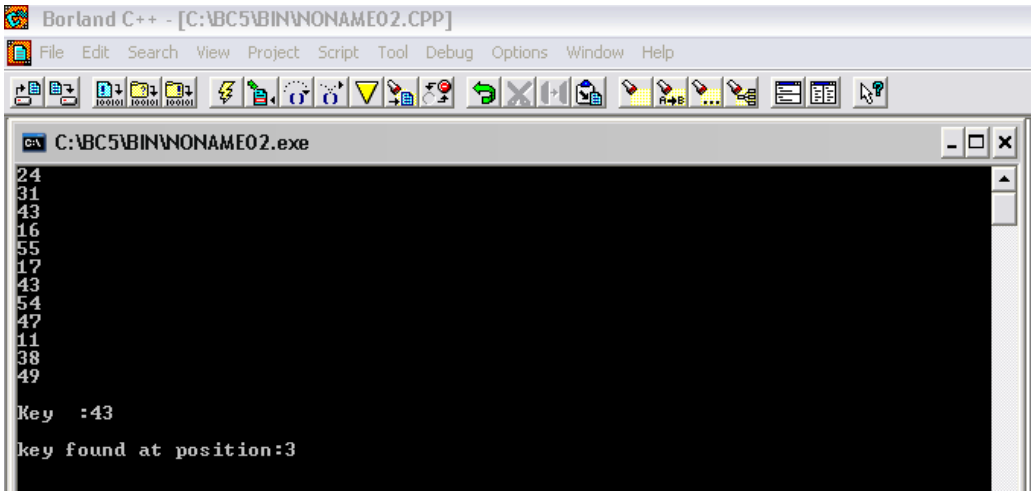


الشكل رقم (2):إخراج البرنامج بالمثال رقم (2)

تدريب(2)

1. أجري التعديلات المطلوبة لكي يقوم البرنامج أعلاه بالبحث عن عنصر في قائمة اختيرت عناصرها بصورة عشوائية ومحصورة بين 0 و100 مستخدما خوارزمية البحث المتتالي؟
2. احسب زمن الاستجابة إذا كان لدينا ملف به 1990 سجل وبزمن قدره 0.095 ثانية؟

ماذا يحدث عندما تحتوي القائمة علي عناصر غير متفردة (وجود عناصر متكررة) سوف نجرب ذلك عندما تحتوي القائمة في البرنامج أعلاه علي عنصرين قيمتهما 43 أي عندما تكون القائمة عناصرها هي **24,31,43,16,55,17,43,54,47,11,38,49** وسوف يكون الإخراج كما هو بالشكل (3)



الشكل (3): إخراج البرنامج أعلاه

2.2 البحث الثنائي Binary Search

تستخدم هذه الطريقة لزيادة كفاءة عملية البحث من خلال زيادة سرعة الوصول إلى السجل وتتطلب هذه الطريقة ترتيب البيانات تصاعديا أو تنازليا حسب قيمة المفتاح.

الخوارزمية :

1.حدد العدد الكلي للسجلات **N**.

2.تأكد أن البيانات مرتبة تصاعدياً أو تنازلياً.

3.أدخل المفتاح المطلوب **SearchKey**

4.اجعل **first=1** و**last=N**.

5.حدد منتصف القائمة **mid=(first+ last)/2**

6.إذا كان **SearchKey=Array[mid]** اذهب إلي 11.

7.إذا كان **SearchKey>Array[mid]** أجعل **first=mid+1** و إلا اجعل **last=mid-1** .

8.إذا كانت **last>=first** أذهب إلي 5.

9.اكتب **key not found**.

10. اذهب إلى 12.

11. اكتب **key found** في الموقع **mid+1**.

12. End.

مثال (3)

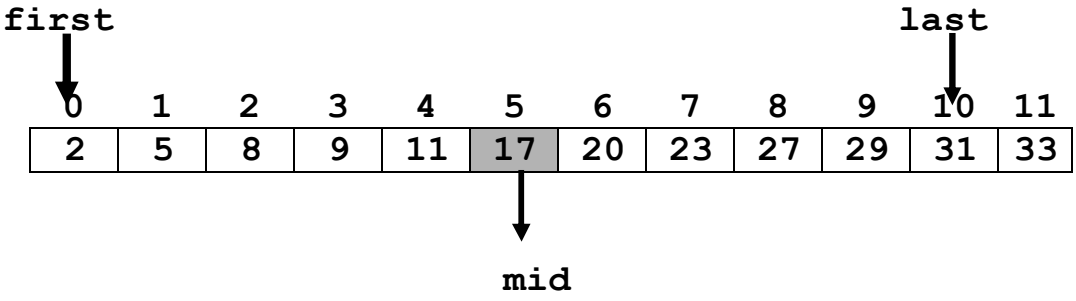
أستخدم خوارزمية البحث الثنائي ابحث عن عنصر عشوائي محصـــوراً بـــين **2** و**33** في قائمة مرتبة عناصرها

2	**5**	**8**	**9**	**11**	**17**	**20**	**23**	**27**	**29**	**31**	**33**

(الحـــل)

حسب الخوارزمية يتم تحديد العـــدد الكلـــي للعناصـــر **N=12** وعليـــه تكـــون قـــيم **first=0** و**last=11** ونفترض أن العنصر العشوائي المطلوب البحث عنه في القائمـــة أعلاه هو **SearchKey=29** (نلاحظ أن العناصر في القائمة مرتبة ترتيبـــاً تصـــاعدياً لأن ذلك هو الشرط الأساسي للخوارزمية) ومن ثم نحدد منتصف القائمة الذي يحسب وفقاً للأتي:

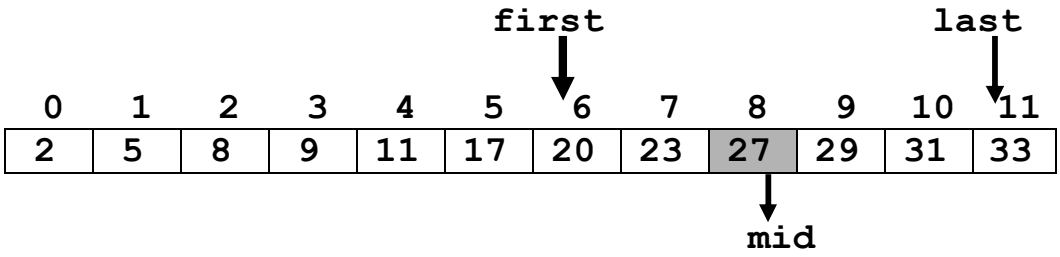
mid=(first+ last)/2=(0+11)/2=5 [1]



الشكل رقم (3): تحديد العنصر الوسط بالقائمة

نلاحظ في الشكل رقم (3) أن العنصر الذي يقابل الوسط **mid** يساوي **17** وهو أقل من العنصر المطلوب البحث عنه الذي يساوي **SearchKey=29** وعليه فإن النصف الأول من القائمة لا يحتوي علي العنصر المطلوب وسوف نبحثه عنه في النصف الثاني وحسب الخوارزمية سوف نجعل قيمة **first=mid+1=6** وستظل قيمة **last=11** كما هي ومن ثم نحدد منتصف القائمة من جديد :

mid=(first+ last)/2=(6+11)/2=8



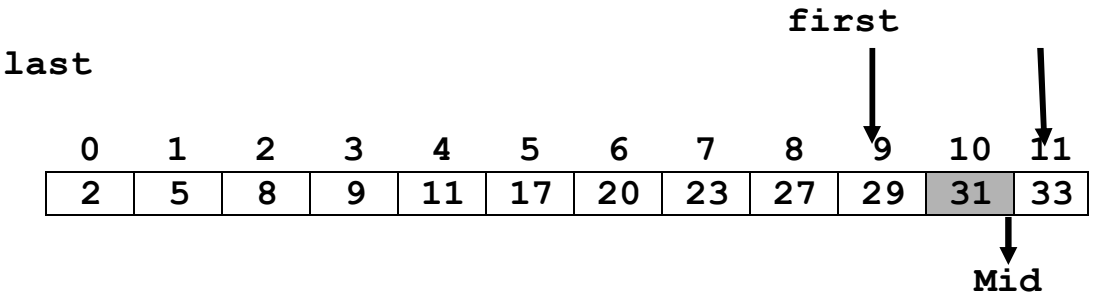
الشكل رقم (4): تحديد العنصر الوسط بالقائمة

ومره أخري نلاحظ في الشكل رقم (4) أن العنصر الذي يقابل الوسط **mid** يساوي **27** وهو أقل من **SearchKey=29** وعليه فإن النصف الأول من القائمة لا يحتوي علي العنصر المطلوب وسوف نبحثه عنه في النصف الثاني وحسب الخوارزمية سوف نجعل قيمة **first=mid+1=9** وستظل قيمة **last=11** كما هي ومن ثم نحدد منتصف القائمة من جديد :

mid=(first+ last)/2=(9+11)/2=10

---

[1] نقوم بإسقاط الكسر



الشكل رقم (5):تحديد العنصر الوسط بالقائمة

و نلاحظ في الشكل رقم (5) أن العنصر الذي يقابل الوسط **mid** يساوي **31** وهو أكبر من **SearchKey=29** وعليه فإن النصف الثاني من القائمة لا يحتوي علي العنصر المطلوب وسوف نبحثه عنه في النصف الأول وحسب الخوارزمية سوف نجعل قيمة **last=mid-1=9** وستظل قيمة **first=9** كما هي ومن ثم نحدد منتصف القائمة من جديد :

mid=(first+ last)/2=(9+9)/2=9

ونجد أن العنصر الذي يقابل الوسط **mid** يساوي **29** وهو يساوي العنصر المطلوب **SearchKey=29**

تعتبر هذه الطريقة أسرع من طريقة البحث المتتالي إذ بلغ عدد المقارنات في المثال السابق 4 مقارنات فقط بينما لو استخدمنا البحث المتتالي سوف تكون عدد المقارنات 12 مقارنة (كأسوأ حاله) ويتضح الفرق الكبير بين عدد المقارنات في الطريقتين كلما زاد عدد العناصر وعموما يتم حساب عدد المقارنات أو كفاءة الخوارزمية بالقانون التالي :

$$\text{Efficiency} = \log_2 N$$

مثال (4)

أكتب برنامجاً يقوم بالبحث عن عنصر عشوائي محصوراً بين **2** و**33** في قائمة مرتبة عناصرها

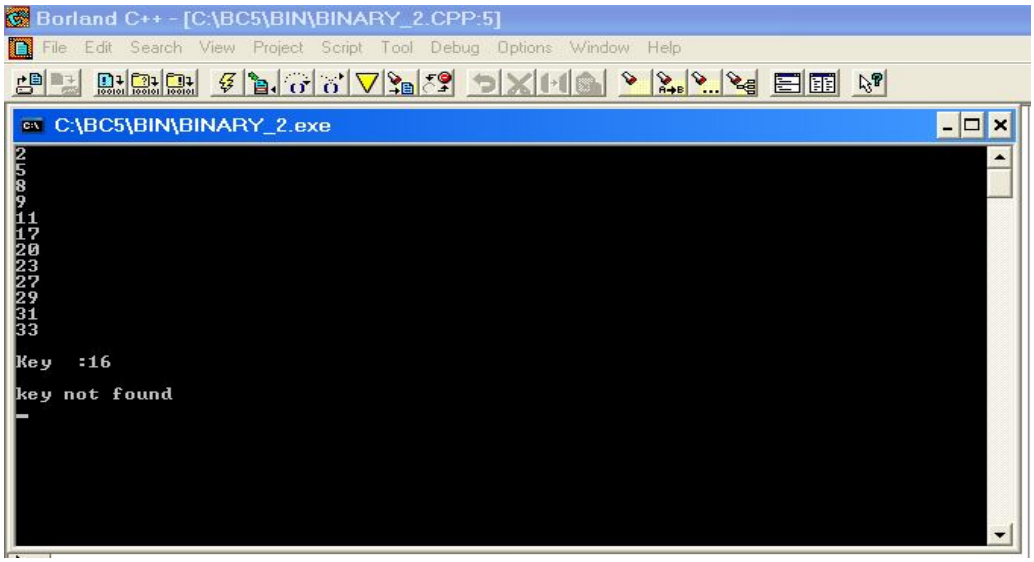
2	5	8	9	11	17	20	23	27	29	31	33

مستخدماً خوارزمية البحث الثنائي.

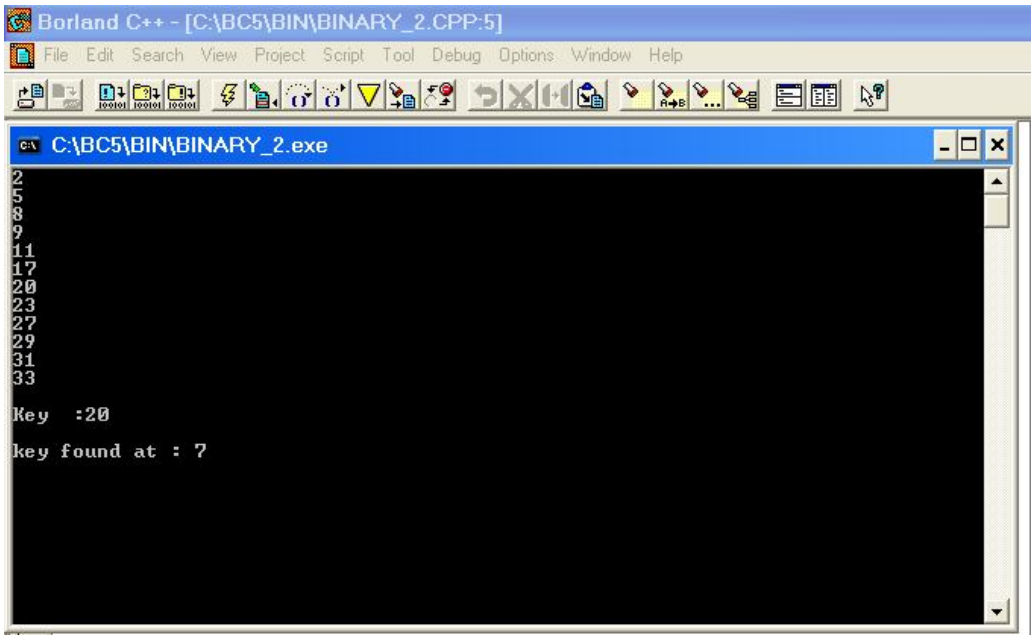
```
#include <iostream.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
int bin_search(int,int);
```

```
int arr[]={2,5,8,9,11,17,20,23,27,29,31,33};
enum boolean {false,true};
int found = false;
int main()
{
    int NN=16;
    for (int j=0;j<NN;j++)
         cout<<arr[j]<<endl;
    randomize();
    int skey=random(31)+2;
    int middle=bin_search(NN,skey);
    cout<<"\nKey  :" <<skey<<"\n";
    if (Found)
        cout<<"\nkey found at : "
            <<(middle+1)<<endl;
    else
        cout<<"\nkey not found"<<endl;
     getch() ;
}
int bin_search(int N,int searchkey)
 {
    int first=1,last= N , mid;
    do
     {
       mid= (first + last)/2;
       if (arr[mid]==searchkey)
          Found=true;
       else
          {
               if (arr[mid]>searchkey)
             last=mid-1 ;
          else
             first=mid+1;
          }
     }
    while (!Found &&last >= first)  ;
    return mid;
}
```

الشكل رقم (6) يوضح إخراج البرنامج أعلاه في حالة عدم وجود المفتاح ضمن القائمة والشكل رقم (7) يوضح إخراج البرنامج في حالة وجود عنصر مطابق للمفتاح .

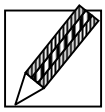


الشكل رقم (6):إخراج البرنامج بالمثال رقم (4)



الشكل رقم (7):إخراج البرنامج بالمثال رقم (4)

تدريب(3)



1. استخدم خوارزمية البحث الثنائي وابحث عن العصر 35 في القائمة الآتية:

20	35	37	40	45	50	51	55	67

2. اكتب برنامجاً يقوم بالبحث عن عنصر عشوائي محصـــورٍ بـــين 5 و95 في قائمة مرتبة عناصرها

5	7	13	22	28	43	57	65	78	87	90	95

مستخدماً خوارزمية البحث الثنائي.

نشاط



قارن بين طريقة البحث المتتالي والبحث الثنائي، ناقش ذلك مع زملائك وأعرض إجابتك على مشرفك الأكاديمي.

أسئلة تقويم ذاتي



1. لماذا نستخدم طريقة البحث الثنائي؟ وماذا تتطلب هذه الطريقة؟
2. اذكر خطوات خوارزمية البحث الثنائي؟

يمكن تلخيص بعض مزايا وعيوب طرق البحث المتتالي والثنائي في الجدول التالي:

طريقة البحث المتتالي	طريقة البحث الثنائي
⁕أقل كفاءة وأكثر سهولة.	⁕أكثر كفاءة وأقل سهولة.
⁕تدني سرعة الوصول إلى السجل المطلوب، يزداد البط بزيادة حجم البيانات.	⁕أسرع من طريقة البحث المتتالي للوصول إلى السجلات.
⁕تكون عملية عندما يكون حجم البيانات صغيراً	⁕ تتطلب ترتيب البيانات تصاعدياً أو تنازلياً حسب قيمة المفتاح

جدول(1) يوضح مزايا وعيوب طرق البحث المتتالي والثنائي

الخلاصة

في هذه الوحدة تعرفنا على عملية البحث وكيفية الوصول إلى عنصر من عناصر البيانات وتحديد موقعه بغرض إجراء واحدة أو أكثر من المعالجات عليه (تعديل، حذف، إضافة، استرجاع)، وتناولنا في الوحدة خوارزميات البحث عن سجل في مصفوفة السجلات ثم البحث عن المعلومة في أحد حقول ذلك السجل.

ومن الطرق التي تم تصميمها للبحث عن المعلومة والتي عرفناها:-

✯البحث المتتالي أو الخطي:

والذي يهدف إلى إحداث مطابقة بين قيمة العنصر المراد البحث عنه وبين بقية العناصر الأخرى بالمصفوفة،وهذه الطريقة من الطرق الأقل كفاءة وأكثر سهولة وتكون عملية عند جمع البيانات الصغيرة،وتعرفنا أيضاً على مقياس الفعالية لهذه الطريقة وهو القانون:

Search Length (SL) = N/2.

وحتى نعرف زمن الاستجابة استخدمنا القانون:

Response time= SL*time of record search.

وطبقنا ذلك على الأمثلة.

✯البحث الثنائي:

طريقة تستخدم لزيادة الكفاءة في عملية البحث من خلال سرعة الوصول إلى السجل وتتطلب هذه الطريقة ترتيب البيانات تصاعدياً أو تنازلياً حسب قيمة المفتاح.

ثم تناولنا خوارزمية هذه الطريقة وطبقنا عليها مجموعة من الأمثلة.

تأكد عزيزي الدارس من تطبيق الأمثلة التي وردت في هذه الوحدة على جهاز الحاسب الآلي.

نأمل أن يكون هذا الملخص قد حقق الأهداف التي جاءت بها هذه الوحدة.

لمحة مسبقة عن الوحدة التالية

الوحدة التالية بعنوان الترتيب، ويعتبر ترتيب المعلومات من الاستخدامات المتكررة في معالجة البيانات، ونعني بترتيب المعلومات وضع سجلات حسب ترتيب معين، وتتناول الوحدة أهم استخدامات الترتيب من حيث تسهيل أو تسريع عملية البحث عن المعلومة، وترتيب

المستويات للأفراد مثل نتائج الامتحانات ... الخ ، وتتطرق الوحدة إلى أشهر الخوارزميـــات في ترتيب المعلومات، مع كتابة برنامجاً لكل طريقة من طرق الترتيب.

إجابات التدريبات

تدريب (1)

```
# include <iostream.h>
        #include <conio.h>
        int seq_search (int ,int);
        int arr[ ] = {24,31,61,16,55,17,43,54,47,11,38,49};
        enum boolean {false,true};
        int found = false;
        void main()
        {
         int nn=12;
         for (int j=0; j<<nn; j++)
         cout << arr[j] <<endl;
         int skey=43;
         int position = seq_search(nn,skey);
         cout<<" \n key :" <<skey<<"\n";
         if (found)
            cout<<"\n key found at :" <<(position+1)<<endl;
                else
                  cout<< "\n key not found"<<endl;
         getch();
         }
                int seq_search (int n,int searchkey)
                 {
                  int pos=0;
                  do
                  {
                    if (arr[pos]==searchkey)
                          found=true;
                    else
                         pos++;
```

```
        }
            while (! found && pos <= n) ;
            return pos;
 }
```

تدريب (2)

-1

```
# include <iostream.h>
# include <conio.h>
int seq_search (int ,int);
int arr[100] ;
enum boolean {false,true};
int found = false;
void main()
{
 int nn=7;
 cout <<"\n enter any seven numbers"<<endl;
 for (int j=0; j<nn; j++)
  {
 cin>>arr[j];
   }
 int skey;
 cout<<"\n enter the search key"<<"\t";
 cin>> skey;
 int position = seq_search(nn,skey);
 cout<<" \n key :" <<skey<<"\n";
 if (found)
    cout<<"\n key found at :" <<(position+1)<<endl;
        else
          cout<< "\n key not found"<<endl;
 getch();
 }
        int seq_search (int n,int searchkey)
         {
           int pos=0;
           do
```

```
{
  if (arr[pos]==searchkey)
       found=true;
  else
      pos++;
          }
        while (! found && pos <= n) ;
        return pos;
}
```

– 2

```
Efficiency = 1990/2 = 995.
Response time = 995*0.095 =94.525 second
```

تدريب (3)

.1

العدد الكلي للعناصر N= 9 وعليه تكون قيم Last= 8, First= 0

وعليه منتصف القائمة= mid= (first + last)/ 2= (0 + 8)/2= 4

العنصر الأخير العنصر الأول

0	1	2	3	4	5	6	7	8
20	35	37	40	45	50	51	55	67

العنصر الأوسط

لاحظ أن العنصر المطلوب أصغر من العنصر الأوسط 45 > 35 فلذلك فإننا نهتم بالنصف السفلي للقائمة.

حسب الخوارزمية سوف نجعل قيمة last = mid – 1= 3 وستظل قيمة First= 0. ومن ثم نحدد منتصف جديد mid= (first + last)/ 2= (0+3)/2=1

نقوم كما هو معلوم بإسقاط الكسر، ونجد أن العنصر الذي يقابل الوسط = 35 وهو يساوي العصر المطلوب Search key= 35.

20	35	37	40

العنصر الأول العنصر الأوسط العنصر الأخير

```
#include <iostream.h>
#include <conio.h>
int bin_search(int,int);
int arr[]={5,7,13,22,28,43,57,65,78,87,90,95};
enum boolean {false,true};
int found = false;
void main()
{
   int NN=16;
   int skey;
   for (int j=0;j<NN;j++)
  cout<<arr[j]<<endl;

   cout<<"\n enter the search key";
   cin>> skey;

   int middle=bin_search(NN,skey);
   cout<<"\nKey  :" <<skey<<"\n";
   if (found)
      {
        cout<<"\nkey found at : "
            <<(middle+1)<<endl;
     }
   else
        cout<<"\nkey not found"<<endl;
         getch() ;
}
int bin_search(int N,int searchkey)
 {
   int first=1,last= N , mid;
   do
    {
      mid= (first + last)/2;
      if (arr[mid]==searchkey)
```

```
            found=true;
        else
            {
                        if (arr[mid]>searchkey)
              last=mid-1 ;
            else
              first=mid+1;
            }
     }
    while (!found &&last >= first)  ;
    return mid;
}
```

مسرد المصطلحات

- **البحث Searching**

الوصول إلى عنصر من عناصر البيانات لتحديد موقعه بغرض إجراء واحدة أو أكثر من المعالجات عليه (تعديل، حذف، إضافة، استرجاع).

- **البحث المتتالي Sequential Search Technique**

الهدف من البحث المتتالي إحداث مطابقة بين قيمة العنصر المراد البحث عنه (**SearchKey**) وبين بقية العناصر الأخرى بالمصفوفة (**Array**)، ولكي يتم البحث عن هذا العنصر لابد من المرور علي كل العناصر في المصفوفة، وإذا تم الوصول إلي نهاية المصفوفة دون الوصول إلي مطابقة، فهذا يعني عدم وجود العنصر داخل المصفوفة.

- **البحث الثنائي Binary Search**

طريقة تستخدم لزيادة كفاءة عملية البحث من خلال زيادة سرعة الوصول إلى السجل وتتطلب هذه الطريقة ترتيب البيانات تصاعديا أو تنازليا حسب قيمة المفتاح.

المراجع

1. William Ford and William Topp, **Data Structure with C++**, Prentice-Hall Inc 1996.
2. Sara Baase ad Allen Van Gelder, **Computer algorithms**, Addison Wesley Longman 2000.
3. Deitel and Deital, **C++ How to program**, Prentice-Hall Inc 2001.

الوحدة الثالثة

الترتيب Sorting



محتويات الوحدة

الموضوع	الصفحة
المقدمة	**49**
تمهيد	49
أهداف الوحدة	50
1. مفهوم الترتيب Sorting Concept	**51**
2. خوارزميات الترتيب	**52**
1.2 الترتيب الفقاعي Bubble sorting	52
2.2 الترتيب بالاختيار Selection sorting	58
3.2 الترتيب بالإدخال Insertion sorting	62
4.2 الترتيب السريـــع Quick sorting	67
الخلاصة	**74**
لمحة مسبقة عن الوحدة التالية	**75**
إجابات التدريبات	**76**
مسرد المصطلحات	**80**
المراجع	**82**

مقدمة

تمهيد

عزيزي الدارس،

مرحباً بك إلى هذه الوحدة

بين يديك الوحدة الثالثة من مقرر هياكل البيانات وهي وحدة تناول بالشرح والتوضيح أهم استخدامات الترتيب، تبدأ الوحدة بمفهوم الترتيب وفيه توضيح لمعنى الترتيب وكيف يتم ترتيب مفاتيح السجلات حسب نوع المفتاح (رقمي، اسمي).

ثم ينتقل بك هذا القسم من الوحدة إلى أهم استخدامات الترتيب.

أما القسم الثاني من هذه الوحدة فيتناول بعض من أنواع خوارزميات الترتيب، وهناك عدة خوارزميات لترتيب المعلومات بعضها معقدة جداً وعالية الكفاءة ونعني بالكفاءة أن يتم الترتيب في أقل وقت ممكن وفي أقل سعة تخزينية، ومن أشهر هذه الخوارزميات التي يتناولها هذه القسم:

- خوارزمية الترتيب الفقاعي.
- خوارزمية الترتيب بالاختيار.
- خوارزمية الترتيب بالإدخال.
- خوارزمية الترتيب السريع.

أتمنى عزيزي الدارس أن تقوم بحل التدريبات الموجودة في هذه الوحدة وأن تكثر من التدريب العملي لتكون الفائدة أكبر.

أهداف الوحدة



عزيزي الدارس،،

بعد فراغك من هذه الوحدة ينبغي أن تكون قادرا على أن:

1. **تذكّر** فوائد الترتيب.
2. **تتعرّف** على محاسن وعيوب الخوارزميات المختلفة.
3. **تبيّن** محاسن وعيوب الخوارزميات المختلفة.
4. **تكتب** برامج مستخدماً جميع أنواع الخوارزميات.

1. مفهوم الترتيب Sorting Concept

يعتبر ترتيب المعلومات من الاستخدامات المتكررة في معالجة البيانات، ونعنى بترتيب المعلومات وضع السجلات حسب ترتيب معين ويتم ذلك بالطبع عن طريق ترتيب مفاتيح السجلات التي ترتب ترتيباً تصاعدياً أو تنازلياً. فإذا كان المفتاح مفتاحاً رقمياً مثل رقم الطالب أو رقم الموظف أو رقم البطاقة الشخصية فإن ترتيب المفاتيح يتم من أصغر رقم أي أصغر قيمة مفتاح تصاعدياً إلى أكبر رقم، أي أكبر قيمة مفتاح، وبهذا تكون سجلات الطلاب أو الموظفين أو المواطنين تم ترتيبها تصاعدياً، والعكس، إذا بدأنا الترتيب بأكبر رقم ثم نزلنا

لأصغر يكون الترتيب تنازلياً، وبالمثل إذا كان المفتاح بالأسماء فإن الترتيب يكون أبجدياً مبتدئاً بالحرف الأول للاسم بالألف ومنتهياً بالياء وعندما يتساوى مفتاحان في الحرف الأول ينظر إلى الحرف الثاني وإذا تساوى في الحرف الثاني ينظر إلى الحرف الثالث وهكذا، من الضروري ألا يتطابق اسمان في كل الحروف و إلا يكون حقل الاسم غير صالحٍ ليكون مفتاحاً، لأن هنالك شرطاً أساسياً لمفتاح السجل وهو أن يكون مفرداً في التعبير عن السجل (لهذه المشكلة لا يفضل استخدام الأسماء في المفاتيح).

إن أهم استخدامات الترتيب هو تسهيل أو تسريع عملية البحث عن السجل هذا إضافة إلى استخدامات أخرى في ترتيب المستويات للأفراد مثل نتائج الامتحانات والمنافسات بين الطلاب والموظفين وعامة المتنافسين ومثل أولويات الحجوزات والطلبات والخدمات عموماً والتي تعطي أولوية للأول فالتالي وهكذا، إذن ليس كل الترتيب يتم فقط لحقل المفاتيح وإنما يمكن أن يتم على أي حقل حسب الحاجة الاستفسارية أو التحليلية.

أسئلة تقويم ذاتي



1. ماذا نعني بترتيب المعلومات.
2. في حالة أن يكون المفتاح اسماً، هل يمكن أن يكون هنالك تطابق في الأسماء؟
3. ما هي استخدامات الترتيب؟

2. خوارزميات الترتيب

هناك خوارزميات عدة في ترتيب المعلومات بعضها معقدة جداً وعالية الكفاءة ونعني بالكفاءة أن يتم الترتيب في أقل وقت ممكن وفي أقل سعة تخزين ممكنة ومن أشهر وأهم هذه الخوارزميات:

- **الترتيب الفقاعي (Bubble sorting).**
- **الترتيب بالإدخال (Insertion sorting).**
- **الترتيب بالاختيار (Selection sorting).**

- **الترتيب بالتجزئة (Shell sorting).**
- **الترتيب السريع (Quick sorting).**
- **الترتيب المقيد (Heap sorting).**
- **الترتيب بالإبدال (Exchange sorting).**
- **الترتيب بالدمج (Merge sorting).**

ولكن لأغراض المقرر سوف نكتفي بشرح بعض هذه الخوارزميات

1.2 الترتيب الفقاعي (Bubble sorting)

يعتبر الأكثر و الأبسط من حيث الاستخدام ولكن يعاب عليه إنه بطئ جداً وتقوم فكرته علي مقارنه كل عنصر في القائمة مع العنصر الذي يليه ثم يقوم بإبدالها إذا لزم الأمر، وتكرر خوارزمية هذه العملية حتى تنتهي الحاجة إلي الإبدال أي حتى يتم التبديل. بمعنى آخـــر يقفـــز العنصر الأكبر إلي الطرف الأعلى في حين يبقى العنصر الآخر في العمق.

خوارزمية الترتيب بالفقاعة:

1. ادخل عدد المفاتيح أو العناصر **(N)** وعناصر القائمة.
2. قارن كل عنصر و العنصر الذي يليه مبتدئاً من العنصر الأول مع الثاني، الثاني مع الثالث ... حتى العنصر **(N-1)**
3. إن كانا غير مرتبين الترتيب المطلوب بدل موقعيهما (نلاحظ هنا أن المقارنة التالية ستكون بين العنصر السابق والعنصر التالي إذا تمت عملية تبديل الموقعين، مثلاً إذا تم تبديل العنصر الثالث مع الرابع فإن العنصر الرابع الجديد سيكون في الواقع هو العنصر الثالث؛ لذا من الناحية العملية ستتم المقارنة هذه المرة بين العنصر الثالث والعنصر الخامس، وهذا سبب تسمية الفقاعة حيث يقفز العنصر ليعيد الترتيب بسرعة إلى وضعه الطبيعي مثل الفقاعة الصغيرة الحقيقية تقفز بسرعة إلى سطح الماء أو الزيت تاركة الفقاعات الأكبر داخل السائل.
4. قف، إذا لم يعد هنالك أي تبديل مواقع و إلا عد إلى **2.**

مثال (1)

إذا كانت القائمة تحتوي علي العناصر التالية:

33 86 92 12 37 48 57 25

المطلوب ترتيب العناصر ترتيباً تصاعدياً حسب خوارزمية الترتيب الفقاعي.

الحل:-

قائمة العناصر الأساسية:

	25	**57**	**48**	**37**	**12**	**92**	**86**	**33**
لا تبديل	**25**	**57**						
تبديل	**25**	**48**	**57**					
تبديل	**25**	**48**	**37**	**57**				
تبديل	**25**	**48**	**37**	**12**	**57**			
لا تبديل	**25**	**48**	**37**	**12**	**57**	**92**		
تبديل	**25**	**48**	**37**	**12**	**57**	**86**	**92**	
تبديل	**25**	**48**	**37**	**12**	**57**	**86**	**33**	**92**

مصفوفة العناصر بعد الدورة الأولى:

	25	**48**	**37**	**12**	**57**	**86**	**33**	**92**
لا تبديل	**25**	**48**						
تبديل	**25**	**37**	**48**					
تبديل	**25**	**37**	**12**	**48**				
لا تبديل	**25**	**37**	**12**	**48**	**57**			
لا تبديل	**25**	**37**	**12**	**48**	**57**	**86**		

تبديل	25	37	12	48	57	33	86	
لا تبديل	25	37	12	48	57	33	86	92

مصفوفة العناصر بعد الدورة الثانية:

	25	37	12	48	57	33	86	92
لا تبديل	25	37						
تبديل	25	12	37					
لا تبديل	25	12	37	48				
لا تبديل	25	12	37	48	57			
تبديل	25	12	37	48	33	57		
لا تبديل	25	12	37	48	33	57	86	
لا تبديل	25	12	37	48	33	57	86	92

مصفوفة العناصر بعد الدورة الثالثة:

	25	12	37	48	33	57	86	92
تبديل	25	12						
لا تبديل	25	12	37					
لا تبديل	25	12	37	48				
تبديل	25	12	37	48	33			
لا تبديل	25	12	37	48	33	57		
لا تبديل	25	12	37	48	33	57	86	
لا تبديل	25	12	37	48	33	57	86	92

مصفوفة العناصر بعد الدورة الرابعة:

	25	12	37	48	33	57	86	92
لا تبديل	25	12						

لا تبديل	12	25	37					
تبديل	12	25	33	37				
لا تبديل	12	25	33	37	48			
لا تبديل	12	25	33	37	48	57		
لا تبديل	12	25	33	37	48	57	86	
لا تبديل	12	25	33	37	48	57	86	92

مصفوفة العناصر بعد الدورة الخامسة:

	12	25	33	37	48	57	86	92
لا تبديل	12	25						
لا تبديل	12	25	33					
لا تبديل	12	25	33	37				
لا تبديل	12	25	33	37	48			
لا تبديل	12	25	33	37	48	57		
لا تبديل	12	25	33	37	48	57	86	
لا تبديل	12	25	33	37	48	57	86	92

إذن تم الترتيب النهائي

أسئلة تقويم ذاتي

1. ماذا نعني بكفاءة الخوارزميات؟
2. عدّد عيوب ومميزات الترتيب الفقاعي.
3. اذكر خطوات خوارزمية الترتيب الفقاعي.

مثال (2)

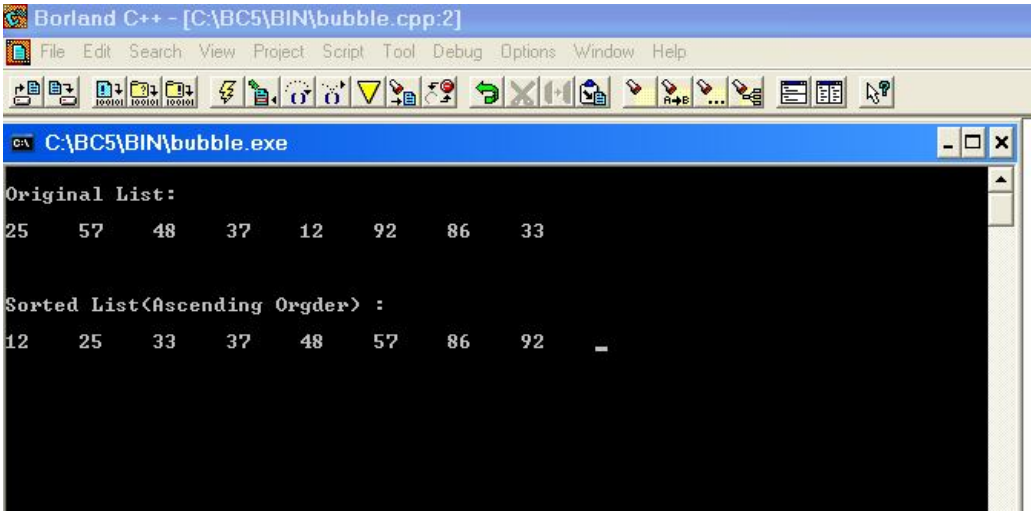
اكتب برنامجاً يقوم بترتيب العناصر في قائمة التالية:

33 86 92 12 37 48 57 25

ترتيباً تصاعدياً مستخدماً خوارزمية الفقاعة.

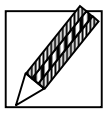
```
#include <iostream.h>
#include <conio.h>
void bubble_sort();
int arr[]={25,57,48,37,12,92,86,33},N=8;
int main()
{
    cout<<"\nOriginal List: \n\n" ;
    for (int j=0;j<N;j++)
        cout<<arr[j]<<"    ";
    bubble_sort();
    cout<<"\n\n\n\nSorted List(Ascending
          Order) : \n\n" ;
    for (int i=0;i<N;i++)
        cout<<arr[i]<<"    ";
     getch() ;
}
void bubble_sort()
{
  for(int out=N-1;out>1;out--)
    for (int in=0;in<out;in++)
       if (arr[in]>arr[in+1])
       {   int temp=arr[in] ;
           arr[in]=arr[in+1];
           arr[in+1]=temp;
       }
}
```

وعند تنفيذ البرنامج يظهر الشكل رقم (1) الذي يوضح إخراجه (كتمرين اجعل العناصر عشوائياً).



الشكل رقم (1):إخراج البرنامج لـــ مثال(2)

تدريب (1)



اكتب برنامجاً لترتيب خمســة أعــداد تصــاعدياً ، حيــث يقــوم المستخدم للبرنامج بإدخال خمسة أعداد بطريقة عشوائية ليتم ترتيبها ترتيباً فقاعياً.

2.2 الترتيب بالاختيار (Selection sorting)

تقوم فكرة الترتيب بالاختيار بالبحث عن أصغر عنصر في القائمة، ونقوم بتبديله مع العنصر الأول، ثم نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الثاني إلي العنصر الأخير، ونقوم بتبديله مع العنصر الثاني، ثم نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الثالث إلي العنصر الأخير، ونقوم بتبديله مع العنصر الثالث،

وهكذا إلي أن تنتهي كل عناصر المصفوفة وعندها نحصل علي مصفوفة مرتبة ترتيباً تصاعدياً.

خوارزمية الترتيب بالاختيار:

1. أدخل عدد العناصر (**N**) وعناصر القائمة.
2. ضع **i=0**
3. ابحث عن اصغر عنصر من عناصر القائمة من العنصر **Array[i]** إلي العنصر **Array[N]**.
4. قم بتبديل العنصر الأصغر مع **Array[i]**
5. أجعل **i=i+1**.
6. إذا كانت **i=N** فان القائمة تكون قد ترتبت و إلا اذهب إلى**3**.

مثال (3)

إذا كانت القائمة تحتوي علي العناصر التالية:

33 86 92 12 37 48 57 25

المطلوب ترتيب العناصر ترتيباً تصاعدياً حسب خوارزمية الترتيب بالاختيار.

الحل

الخطوة (مَحَرَّة)

نبحث عن أصغر عنصر من عناصر المصفوفة وهو العنصر رقم [5] الذي يساوي "**12**" ونقوم بتبديله مع العنصر رقم [1]:

رقم العنصر	1	2	3	4	5	6	7	8

33	86	92	25	37	48	57	12

الخطوة (صفر)

نبحث عن أصغر عنصر من بين عناصر المصفوفة من العنصر الثاني إلي العنصر الأخير وهو العنصر رقم [5] الذي يساوي "**25**" ونقوم بتبديله مع العنصر رقم [2]:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	48	37	57	92	86	33

الخطوة (أول)

نبحث عن أصغر عنصر من بين عناصر المصفوفة من العنصر الثالث إلي العنصر الأخير وهو العنصر رقم [8] الذي يساوي "**33**" ونقوم بتبديله مع العنصر رقم [3]:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	33	37	57	92	86	48

الخطوة (ثان)

نبحث عن أصغر عنصر من بين عناصر المصفوفة من العنصر الرابع الي العنصر الأخير وهو العنصر رقم [4] الذي يساوي "**37**" ونقوم بتبديله مع العنصر رقم [4] وهو نفس العنصر:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	33	37	57	92	86	48

الخطوة (ثالث)

نبحث عن أصغر عنصر من بين عناصر المصفوفة من العنصر الخامس إلي العنصر الأخير وهو العنصر رقم [8] الذي يساوي "**48**" ونقوم بتبديله مع العنصر رقم [5]:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	33	37	48	92	86	57

الخطوة (رابع)

نبحث عن أصغر عنصر من بين عناصر المصفوفة من العنصر السادس إلي العنصر الأخير وهو العنصر رقم [8] الذي يساوي "**57**" ونقوم بتبديله مع العنصر رقم [6]:

رقم العنصر	1	2	3	4	5	6	7	8
	12	**25**	**33**	**37**	**48**	**57**	**86**	**92**

الخطوة (ﷺ)

نبحث عن أصغر عنصر من بين عناصر المصفوفة من العنصر السابع إلي العنصر الأخير وهو العنصر رقم [7] الذي يساوي "**86**" ونقوم بتبديله مع العنصر رقم [7] وهو نفس العنصر:

رقم العنصر	1	2	3	4	5	6	7	8
	12	**25**	**33**	**37**	**48**	**57**	**86**	**92**

ونجد العنصر الأخير يكون في موضعه الصحيح.

إذن تم الترتيب النهائي .

مثال (4)

اكتب برنامجاً يقوم بترتيب العناصر في القائمة التالية:

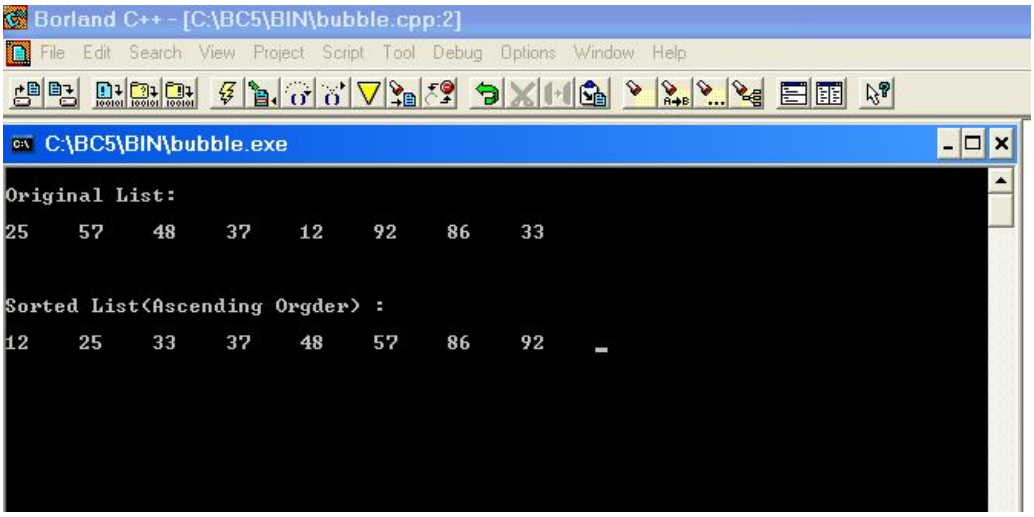
33 86 92 12 37 48 57 25

ترتيباً تصاعدياً مستخدماً خوارزمية الترتيب بالاختيار.

```
#include <iostream.h>
#include <conio.h>
void selection_sort();
int arr[]={25,57,48,37,12,92,86,33},N=8;
int main()
{   cout<<"\nOriginal List: \n\n" ;
    for (int j=0;j<N;j++)
        cout<<arr[j]<<"    ";
    selection_sort();
```

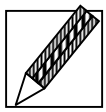
```
    cout<<"\n\n\n\nSorted List : \n\n" ;
    for (int i=0;i<N;i++)
        cout<<arr[i]<<"    ";
     getch() ;
}
void selection_sort()
{
  int temp,min;
  for(int h=0;h<N-1;h++)
  {
 min=h;
    for (int m=h+1;m<N;m++)
       if (arr[m]<arr[min])
          min=m;
    temp=arr[h] ;
    arr[h]=arr[min];
    arr[min]=temp;
  }
}
```

وعند تنفيذ البرنامج يظهر الشكل رقم (2) الذي يوضح إخراجه وفيه نجد عناصر القائمة قبل عملية الترتيب ومن ثم عناصر القائمة بعد تطبيق خوارزمية الترتيب بالاختيار.



الشكل رقم (2):إخراج البرنامج لـــ مثال(4)

تدريب (2)



باستخدام طريقة الترتيب بالاختيار اكتب برنامج لترتيب الحـــروف التالية تصاعدياً:

N A D I R

3.2 الترتيب بالإدخال (Insertion sorting)

تقوم فكرة هذه الطريقة علي البحث عن أصغر عنصر في القائمة ويحفظ فـــي متغيـــر آخر ثم تزاح العناصر من **Array[0]** وحتى العنصر الذي يقع قبل مكان العنصر الأصغر الذي وجدناه فيتقدم كل عنصر خطوة إلي الإمام ونضع العنصر الأصغر الموجود في المتغيـــر المؤقت في موقع العنصر **Array[0]** ثم نكرر العملية السابقة ولكن في هـــذه المـــرة مـــع العناصر من **Array[1]** وحتى آخر عنصر في القائمة وهكذا نكرر هذه العملية حتى آخر عنصر في القائمة.

تختلف فكرة هذه الخوارزمية في الترتيب عن الترتيب الفقاعي والترتيب بالاختيـــار إذ لا تستخدم فكرة دالة التبديل **(swap)** يستعيض عنها بإزاحة العناصر وذلك لخلق مســـاحة أو موقع فارغ للعنصر الجديد(الأصغر).

خوارزمية الترتيب بالإدخال:

1. أدخل عدد العناصر **(N)** وعناصر القائمة.
2. ضع **i=0**
3. ابحث عن اصغر عنصر من عناصر القائمة من العنصر **Array[i]** إلي العنصر **Array[N].**
4. اجعل **temp** يساوي العنصر الأصغر.
5. اعمل إزاحة لجميع العناصر من العنصر **Array[i]** وحتى العنصر الذي يقع قبل مكان العنصر الأصغر بتقديم كل منهم خطوة إلى الأمام.
6. أجعل **i=i+1.**

7. إذا كانت **i=N** فإن القائمة تكون قد ترتبت و إلا اذهب إلى**3**.

مثال (5)

إذا كانت القائمة تحتوي علي العناصر التالية:

33 86 92 12 37 48 57 25

المطلوب ترتيب العناصر ترتيباً تصاعدياً حسب خوارزمية الترتيب بالإدخال.

الحل:-

الخطوة (مَحَرَّمْ)

نبحث عن أصغر عنصر من عناصر القائمة وهو العنصر رقم [5] الذي يساوي "**12**" و نجعل قيمة المتغير **temp=12** ثم نعمل إزاحة لجميع العناصر من العنصر الأول وحتى العنصر الرابع (هو العنصر الذي يقع قبل مكان العنصر الأصغر) بتقديم كل منهم خطوة إلى الأمام وأخيراً نقوم بجعل قيمة العنصر رقم [1] تساوي **temp**:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	57	48	37	92	86	33

الخطوة (صَفَرْ)

نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الثاني إلي العنصر الأخير وهو العنصر رقم [2] الذي يساوي "**25**" ونجعل قيمة المتغير **temp=25** ولكن لأن العنصر رقم [2] يقع في موقعه الصحيح فليس هنالك عملية إزاحة ستتم للعناصر:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	57	48	37	92	86	33

الخطوة (نَبْدَأْ)

نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الثالث إلي العنصر الأخير وهو العنصر رقم [8] الذي يساوي "**33**" و نجعل قيمة المتغير **temp=33** ثم نعمل

إزاحة لجميع العناصر من العنصر رقم [3] وحتى العنصر رقم [7] (هو العنصر الذي يقع قبل مكان العنصر الأصغر) بتقديم كل منهم خطوة إلى الأمام وأخيراً نقوم بجعل قيمة العنصر رقم [3] تساوي **temp**:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	33	57	48	37	92	86

الخطوة (ثمانية)

نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الرابع إلي العنصر الأخير وهو العنصر رقم [6] الذي يساوي "**37**" ونجعل قيمة المتغير **temp=37** ثم نعمل إزاحة لجميع العناصر من العنصر رقم [**4**] وحتى العنصر رقم [5] بتقديم كل منهم خطوة إلى الأمام وأخيراً نقوم بجعل قيمة العنصر رقم [**4**] تساوي **temp**:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	33	37	57	48	92	86

الخطوة (تسعة)

نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الخامس إلي العنصر الأخير وهو العنصر رقم [6] الذي يساوي "**48**" ونجعل قيمة المتغير **temp=48** ثم نعمل إزاحة لجميع العناصر من العنصر رقم [5] وحتى العنصر رقم [5] بتقديم كل منهم خطوة إلى الأمام وأخيراً نقوم بجعل قيمة العنصر رقم [5] تساوي **temp**:

رقم العنصر	1	2	3	4	5	6	7	8
	12	25	33	37	48	57	92	86

الخطوة (عشرة)

نبحث عن أصغر عنصر من بين عناصر المصفوفة من العنصر السادس إلي العنصر الأخير وهو العنصر رقم [6] الذي يساوي "**57**" ونجعل قيمة المتغير **temp=57** ولكن لأن العنصر رقم [6] يقع في موقعه الصحيح فليس هنالك عملية إزاحة ستتم للعناصر:

رقم العنصر	1	2	3	4	5	6	7	8

86	92	57	48	37	33	25	12

الخطوة (ج)

نبحث عن أصغر عنصر من بين عناصر المصفوفة من العنصر السابع إلي العنصر الأخير وهو العنصر رقم [8] الذي يساوي "**86**" ونجعل قيمة المتغير **temp=86** ثم نعمل إزاحة لجميع العناصر من العنصر رقم [7] وحتى العنصر رقم [7] بتقديم كل منهم خطوة إلى الأمام وأخيراً نقوم بجعل قيمة العنصر رقم [7] تساوي **temp**:

رقم العنصر	8	7	6	5	4	3	2	1
	92	86	57	48	37	33	25	12

ونجد العنصر الأخير يكون في موضعه الصحيح.

إذن تم الترتيب النهائي.

أسئلة تقويم ذاتي



ما الفرق بين خوارزمية الترتيب بالإدخال والترتيب الفقاعي والترتيب بالاختيار؟

مثال (6)

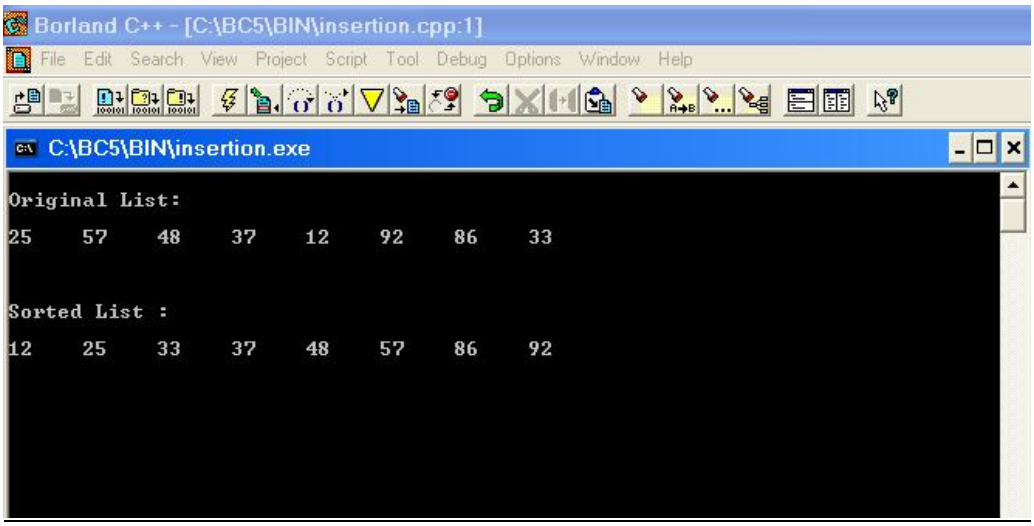
اكتب برنامجاً يقوم بترتيب العناصر في قائمة التالية:

33	86	92	12	37	48	57	25

ترتيباً تصاعدياً مستخدماً خوارزمية الترتيب الاختيار.

```
#include <iostream.h>
#include <conio.h>
void insertion_sort();
int arr[]={25,57,48,37,12,92,86,33},N=8;
int main()
{
    cout<<"\nOriginal List: \n\n" ;
    for (int j=0;j<N;j++)
```

```
        cout<<arr[j]<<"    ";
    insertion_sort();
    cout<<"\n\n\n\nSorted List : \n\n" ;
    for (int m=0;m<N;m++)
        cout<<arr[m]<<"    ";
     getch() ;
}
void insertion_sort()
{
  for(int i=1;i<N;i++)
  {
    int temp=arr[i];
    int j=i;
    while (j>0 && arr[j-1]>=temp)
    {
       arr[j]=arr[j-1];
       --j;
    }
    arr[j]=temp;
  }
}
```



الشكل رقم (3): إخراج البرنامج لـــ مثال(6)

4.2 الترتيب السريع (Quick sorting)

يعتبر هدا النوع هو الأسرع من حيث السرعة عند تطبيقه على القوائم الكبيرة. وتقوم فكرة الترتيب السريع باختيار عنصر معين من العناصر المراد ترتيبها ويسمي هذا العنصر بالقطب أو المحور**Pivot** ومن ثم يتم تقسيم العناصر المتبقية إلى مجموعتين تعتبر كل مجموعة قائمة جزئية جديدة حيث تشمل المجموعة الأولى جميع العناصر الأقل من العنصر المحور **Pivot** بينما تشمل المجموعة الثانية جميع العناصر الأكبر من العنصر المحور **Pivot** ويتوسط العنصر المحور **Pivot** هاتين المجموعتين ونلاحظ أن القائمة ترتبت ترتيباً جزئياً. وبتطبيق الترتيب السريع على عناصر المجموعتين الأولى والثانية مرة أخرى كل على حدة تترتب القائمة أكثر فأكثر وهكذا نقوم بتطبيق فكرة الترتيب السريع علي مجموعة جديدة إلى أن تترتب القائمة بشكل كامل. ويلاحظ أن في كل خطوة من خطوات الترتيب السريع تكون القائمة التي تطبق عليها عملية الترتيب السريع أصغر من القائمة الأصلية .

تأتى التسمية من طبيعته حيث انه يتميز بالسرعة وتعتبر هذه الطريقة من الطرق المبسطة نظرياً ولكنها معقدة في البرمجة حتى إنها جعلت المتخصصين يجثون على ركبهم لعدة سنوات ليكتبوا خوارزمية مبسطة عملياً إلا إنهم لم يتمكنوا من إزالة هذا التعقيد في مستوى الطلاب الجامعيين وقد أطلق عليها خوارزميته . **Recursive Algorithm**

خوارزمية الترتيب السريع Quick Sort:

1. يتم اختيار العنصر **Pivot** ويفضل أن يكون العنصر الأوسط في القائمة، وذلك عن طريق المعادلة التالية:

((خانة العنصر الأول + خانة العنصر الأخير−1) / 2) +1

2. يوضع العنصر المحور**Pivot** في مكانه الصحيح بحيث تكون كل العناصر الأقل منه يساراً وتلك التي أكبر يميناً
3. إذا احتوت القائمة اليسرى على أكثر من عنصر أعد تطبيق الخوارزمية عليها.
4. إذا احتوت القائمة اليمنى على أكثر من عنصر أعد تطبيق الخوارزمية عليها.

مثال (7)

إذا كانت القائمة تحتوي علي العناصر التالية:

33 86 92 12 37 48 57 25

المطلوب ترتيب العناصر ترتيباً تصاعدياً حسب خوارزمية الترتيب السريع.

الحل:-

الخطوة الأولي

يتم اختيار العنصر **Pivot** ويفضل أن يكون العنصر الأوسط في القائمة وفى هذا المثال نجد أن العنصر الأوسط في القائمة هو العنصر A[5] وقيمته تساوي **37.**

A[1]	A[2]	A[3]	A[4]	A[5]	[6]	[7]	[8]
33	86	92	12	37	48	57	25

الخطوة الثانية

يتم تعيين مؤشرين **L** و **R** بحيث يشير المؤشر**L** إلي بداية القائمة (أي القيمة المبدئية للمؤشر**L** تساوى**1**) و يشير المؤشر**R** إلي نهاية القائمة (أي القيمة المبدئية للمؤشر**R** تساوى**8**) ويمكن توضيح ذلك على النحو التالي:

A[1]	A[2]	A[3]	A[4]	A[5]	[6]	[7]	[8]
33	86	92	12	37	48	57	25
L=1							R=8

الخطوة الثالثة

تبدأ عملية مقارنة قيمة عنصر المصفوفة المشار إليه بالمؤشر**L=1** الذي يساوي **33** مع العنصر **Pivot=37** والهدف هو إيجاد العناصر الأكبر من المحور وذلك لنقلها إلى يمينه وتزداد قيمة **L** بمقدار **1** وفى كل مقارنة إذا كانت قيمة العنصر أقل من قيمة **Pivot=37** ليشير إلى العنصر التالي ويترك العنصر في مكانه وتتوقف عملية المقارنة عند الوصول إلى عنصر قيمته أكبر من قيمة **Pivot=37** وهذا يعنى أن مثل هذه القيمة يجب أن تكون إلى يمين المحور (القطب) وعليه عند المقارنة مع العنصر الذي يشير إليه المؤشر**L** وقيمته تساوي **33** نجده أقل من المحور وهذا يعني زيادة قيمة المؤشر**L** بمقدار **1**

(أي أن **L=2**) ليشير إلي العنصر **A[2]** وقيمته تساوي **86** وعند مقارنتها بالمحور نجـــدها أكبر منه وعندها تتوقف عملية المقارنة ويمكن توضيح ذلك على النحو التالي:

A[1]	A[2]	A[3]	A[4]	A[5]	[6]	[7]	[8]
33	86	92	12	37	48	57	25
	L=2						R=8

الخطوة الرابعة

تبدأ عملية مقارنة قيمة عنصر المصفوفة المشار إليه بالمؤشر**R=8** الذي يســـاوي **25** مع العنصر **Pivot=37** والهدف هو إيجاد العناصر الأصغر من المحور وذلك لنقلها إلـــى يساره وتتناقص قيمة **R** بمقدار **1** في كل مقارنة إذا كانت قيمة العنصـــر أكبـــر مـــن قيمـــة **Pivot=37** ليشير إلى العنصر التالي ويترك العنصر في مكانه وتتوقف عمليـــة المقارنـــة عند الوصول إلى عنصر قيمته أقل من قيمة **Pivot=37** وهذا يعني أن مثل هـــذه القيمـــة يجب أن تكون إلى يسار المحور (القطب) وعليه عند المقارنة مع العنصر الـــذي يشـــير إليـــه المؤشر**R** وقيمته تساوي **25** نجده أصغر من المحور وعندها تتوقف عملية المقارنـــة ويمكـــن توضيح ذلك على النحو التالي:

A[1]	A[2]	A[3]	A[4]	A[5]	[6]	[7]	[8]
33	86	92	12	37	48	57	25
	L=2						R=8

الخطوة الخامسة

نجد في الخطوات السابقة تم إيجاد عنصرين يجب تبديليهما ليصبح كل منهما في مكانه الصحيح بالنسبة إلى القيمة المحور وفى كل مرة يتم فيها تبديل عنصرين مع بعضيهما البعض يجب أن تزيد قيمة المؤشر **L** بمقدار**1** وتنقص قيمة المؤشر**R** بمقدار**1** ويمكـــن توضـــيح ذلك على النحو التالي:

A[1]	A[2]	A[3]	A[4]	A[5]	[6]	[7]	[8]
33	25	92	12	37	48	57	86

L=3 **R=7**

ومن ثم بتكرار الخطوة السابقة نجد أن المؤشر**L** يشير إلي العنصر **A[3]** وقيمته تساوي **92** وعند مقارنتها بالمحور نجدها أكبر منه وعندها تتوقف عملية المقارنة وننتقل إلي المؤشر**R** الذي يشير إلي العنصر **A[7]** وقيمته تساوي **57** وعند مقارنتها بالمحور نجدها أكبر منه لذا تنقص قيمة المؤشر**R** لتشير إلي العنصر **A[6]** وقيمته تساوي **48** وعند مقارنتها بالمحور نجدها أكبر منه لذا تنقص قيمة المؤشر**R** مرة أخري لتشير إلي العنصر **A[5]** وقيمته تساوي **37** وعند مقارنتها بالمحور نجدها أقل منه او تساويه وعندها تتوقف عملية المقارنة.

إذن تم إيجاد عنصرين يجب تبديليهما ليصبح كل منهما في مكانه الصحيح بالنسبة إلى القيمة المحور وعندها يجب أن تزيد قيمة المؤشر **L** بمقدار**1** وتنقص قيمة المؤشر **R** بمقدار **1** ويمكن توضيح ذلك على النحو التالي:

A[1]	**A[2]**	**A[3]**	**A[4]**	**A[5]**	**[6]**	**[7]**	**[8]**
33	**25**	**37**	**12**	**92**	**48**	**57**	**86**

R=L=4

أصبحت الآن قيمة **R=L=4** وهذا يعنى انتهاء الجولة الأولى من هذه الخوارزمية حيث تصبح المصفوفة الكلية مرتبة جزئياً ومقسمة إلى مجموعتين كالأتي:

المجموعة الأولى **12** **37** **25** **33**

المجموعة الثانية **86** **57** **48** **92**

نكرر جميع الخطوات السابقة على القائمتين الجزئيتين الجديدتين ونبدأ بالقائمة الأولى ونحدد **كخطوة أولي** العنصر **Pivot** ويفضل أن يكون العنصر الأوسط في القائمة وفى هذا المثال نجد أن العنصر الأوسط في القائمة هو العنصر **A[3]** وقيمته تساوي **37.**

A[1]	A[2]	A[3]	A[4]	A[5]	[6]	[7]	[8]
33	25	37	12	92	48	57	86

وفي **الخطوة الثانية** يتم تعيين مؤشرين **L** و **R** بحيث يشير المؤشر**L** إلي بداية القائمة (أي القيمة المبدئية للمؤشر**L** تساوى**1**) و يشير المؤشر**R** إلي نهاية القائمة الأولي (أي القيمة المبدئية للمؤشر**R** تساوى**4**) ويمكن توضيح ذلك على النحو التالي:

A[1]	A[2]	A[3]	A[4]	A[5]	[6]	[7]	[8]
33	25	37	12	92	48	57	86
L=1			R=4				

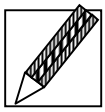
نجد أن **الخطوة الثالثة** تبدأ بمقارنة قيمة عنصر المصفوفة المشار إليه بالمؤشر **L=1** الذي يساوي **33** مع العنصر **Pivot=37** نجدها أقل منه وهذا يعني زيادة قيمة المؤشر**L** بمقدار **1** (أي أن **L=2**) ليشير إلي العنصر **A[2]** وقيمته تساوي **25** وعند مقارنتها بالمحور نجدها أقل منه وهذا يعني زيادة قيمة المؤشر**L** بمقدار **1** (أي أن **L=3**) ليشير إلي العنصر **A[3]** وقيمته تساوي **37** وعند مقارنتها بالمحور نجدها أكبر، أو تساوي قيمة المحور وعندها تتوقف عملية المقارنة من هذا الاتجاه لتبدأ من الاتجاه الآخر **كخطوة رابعة** وفيها تبدأ عملية مقارنة قيمة عنصر المصفوفة المشار إليه بالمؤشر **R=4** الذي يساوي **12** مع العنصر **Pivot=37** نجده أصغر من المحور وعندها تتوقف عملية المقارنة و**الخطوة الخامسة** تم إيجاد عنصرين يجب تبديليهما ليصبح كل منهما في مكانه الصحيح بالنسبة إلى القيمة المحور وفى كل مرة يتم فيها تبديل عنصرين مع بعضيهما البعض يجب أن تزيد قيمة المؤشر **L** بمقدار **1** وتنقص قيمة المؤشر**R** بمقدار **1** ويمكن توضيح ذلك على النحو التالي:

A[1]	A[2]	A[3]	A[4]	A[5]	[6]	[7]	[8]
33	25	12	37	92	48	57	86
		R=3	L=4				

أصبحت الآن قيمة **R=3<L=4** وهذا يعنى انتهاء الجولة الثانية من هذه الخوارزمية حيث تصبح المصفوفة الكلية مرتبة جزئياً ومقسمة إلى مجموعتين المجموعة الأولى تحتوي

علي كل العناصر الأقل من أو تساوي العنصر **Pivot=37** (تضم كل عناصـــر القائمـــة) والمجموعة الثانية التي تحتوي علي كل العناصر الأكبر من العنصر **Pivot=37** (لا تضم أي عنصر وهذا يحدث دائماً عندما يكون للمحور أكبر أو أقل قيمة بالقائمة)

تدريب (3)



اكتب برنامجاً يقوم بترتيب العناصر في قائمة التالية:

33 86 92 12 37 48 57 25

ترتيباً تصاعدياً مستخدماً خوارزمية الترتيب السريع.

نشاط



ناقش الآتي مع زملائك وأعرض إجابتك على مشرفك الميداني:

الترتيب السريع من حيث:

- ❖ طبيعة التسمية.
- ❖ الفكرة التي تقوم عليه.
- ❖ إسهامات المختصين في تبسيط خوارزميته.

تقويم ذاتي



1. بما تمتاز خوارزمية الترتيب السريع عن باقي الخوارزميات؟
2. ماذا يعيب طريقة الترتيب السريع؟

الخلاصة

عزيزي الدارس،

تخلص بك هذه الوحدة للآتي:

- ترتيب المعلومات هو وضع السجلات حسب ترتيب معين ويتم ذلك عن طريق ترتيب مفاتيح السجلات التي ترتب ترتيباً تصاعدياً أو تنازلياً. حسب نوع المفتاح (رقمي أو اسمي).
- أهم استخدامات الترتيب هو تسهيل أو تسريع عملية البحث عن السجل هذا إضافةً إلى استخدامات أخرى.
- وليس كل الترتيب يتم فقط لحقل المفاتيح وإنما يمكن أن يتم على حقل حسب الحاجة الاستفسارية أو التحليلية.

- هناك خوارزميات عدة في ترتيب المعلومات بعضها معقد جداً وعالية الكفاءة، ونعني بالكفاءة أن يتم الترتيب بأقل وقت ممكن وفي أقل سعة تخزينية ممكنة.
- الترتيب الفقاعي يعتبر الأكثر والأبسط من حيث الاستخدام ولكن يعاب عليه أنه بطيء جداً.
- تقوم فكرة الترتيب الفقاعي علي مقارنه كل عنصر في القائمة مع العنصر الذي يليه ثم يقوم بإبدالها إذا لزم الأمر، وتكرر خوارزمية هذه العملية حتى تنتهي الحاجة إلي الإبدال أي حتى يتم التبديل. بمعنى آخر يقفز العنصر الأكبر إلي الطرف الأعلى في حين يبقى العنصر الآخر في العمق.
- تقوم فكرة الترتيب بالاختيار بالبحث عن أصغر عنصر في القائمة، ونقوم بتبديله مع العنصر الأول، ثم نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الثاني إلي العنصر الأخير، ونقوم بتبديله مع العنصر الثاني، ثم نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الثالث إلي العنصر الأخير، ونقوم بتبديله مع العنصر الثالث، وهكذا إلي أن تنتهي كل عناصر المصفوفة وعندها نحصل علي مصفوفة مرتبة ترتيباً تصاعدياً.
- تقوم فكرة هذه الطريقة علي البحث عن أصغر عنصر في القائمة ويحفظ في متغير آخر ثم تزاح العناصر من **Array[0]** وحتى العنصر الذي يقع قبل مكان العنصر الأصغر الذي وجدناه فيتقدم كل عنصر خطوة إلي الإمام ونضع العنصر الأصغر الموجود في المتغير المؤقت في موقع العنصر **Array[0]** ثم نكرر العملية السابقة ولكن في هذه المرة مع العناصر من **Array[1]** وحتى آخر عنصر في القائمة وهكذا نكرر هذه العملية حتى آخر عنصر في القائمة.
- تختلف فكرة هذه الخوارزمية في الترتيب عن الترتيب الفقاعي والترتيب بالاختيار إذ لا تستخدم فكرة دالة التبديل **(swap)** يستعيض عنها بإزاحة العناصر وذلك لخلق مساحة أو موقع فارغ للعنصر الجديد(الأصغر).
- تقوم فكرة الترتيب السريع باختيار عنصر معين من العناصر المراد ترتيبها ويسمي هذا العنصر بالقطب أو المحور**Pivot** ومن ثم يتم تقسيم العناصر المتبقية إلى مجموعتين

تعتبر كل مجموعة قائمة جزئية جديدة حيث تشمل المجموعة الأولى جميع العناصر الأقل من العنصر المحور **Pivot** بينما تشمل المجموعة الثانية جميع العناصر الأكبر من العنصر المحور **Pivot** ويتوسط العنصر المحور **Pivot** هاتين المجموعتين ونلاحظ أن القائمة ترتبت ترتيباً جزئياً. وبتطبيق الترتيب السريع على عناصر المجموعتين الأولى والثانية مرة أخرى كل على حدة تترتب القائمة أكثر فأكثر وهكذا نقوم بتطبيق فكرة الترتيب السريع علي مجموعة جديدة إلى أن تترتب القائمة بشكل كامل.

أتتمى عزيزي الدارس أن تكون قد استوعبت هذه الوحدة، التي تعتبر دراستها تُنَمَّي لزيادة مهارات البرمجة وتساعد في اختيار الطريقة المناسبة للترتيب، وفقك الله.

لمحة مسبقة عن الوحدة التالية

سوف نتطرّق بإذن الله في الفصل القادم إلى المكدسات والصفوف، المكدسات هي هيكل من هياكل البيانات الخطية المتغيرة وتتكون من مجموعة من العناصر أو القيم، مخزنة بطريقة معينة وخاصة، (تسمي المكدسة في بعض الأحيان بالبيانات المتراصة)، أما القسم الثاني (الصفوف) فتستخدم في عمليات المحاكاة.

إجابات التدريبات

تدريب (1)

```
# include <iostream.h>
# include <conio.h>
void bubble_sort ( );
int arr [5] , N = 5;
int main ( )
{
   for (int j = 0 ; j < N ;  j ++)
     {
      cout << "\n Enter number : \t" ;
      cin >> arr [ j ] ;
       }
     bubble_sort ( );
```

```
      cout << "\n sorted list : \n\n" ;
      for (int i = 0 ; i < N ; i ++)
      cout << arr [ i ] << "  " ;
      getch ( ) ;
}
void bubble_sort ( )
{
   for (int out = N-1 ; out > 1 ; out --)
     for (int in = 0 ; in < out; in ++)
       if (arr [in] > arr [in + 1])
       {
         int temp = arr [in] ;
         arr [in] = arr [in+1];
         arr [in+1] = temp;
        }
}
```

تدريب (2)

```
# include <iostream.h>
# include <conio.h>
void selection_sort ( );
char arr[ ] = { 'N' , 'A', 'D' , 'I', 'R'};
int N = 5;
int main ( )
{
  cout << "\n original list : \n\n";
  for ( int j = 0 ; j < N; j ++)
  cout << arr [ j ] << "      ";
  selection_sort (  ) ;
  cout << "\n sorted list : \n ";
  for (int h = 0 ; h < N ; h++)
  cout << arr [ h ] << "  ";
```

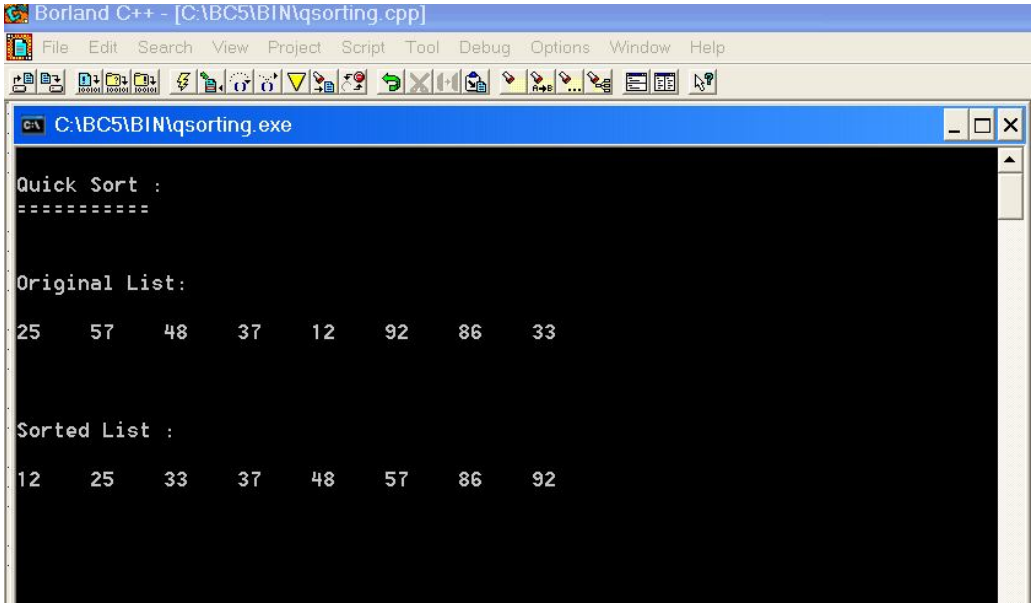
```
   getch ( );
}
void selection_sort ( )
   {
    int  temp , min ;
     for (int i = 0 ; i < N-1 ; i++)
      {
       min = i ;
          for (int j = i+1; j < N ; j ++)
            if (arr[ j ] < arr [min])
            min = j ;
            temp = arr [ i ] ;
            arr [ i ] = arr [min] ;
            arr [min] = temp ;
         }
 }
```

تدريب (3)

```
#include <iostream.h>
#include <conio.h>
void  QSort(int , int);
int arr[]={33,86,92,12,37,48,57,25},N=8;
int main()
{
     cout<<"\nQuick Sort :" ;
     cout<<"\n===========\n\n" ;
     cout<<"\nOriginal List: \n\n" ;
     for (int j=0;j<N;j++)
          cout<<arr[j]<<"    ";
     QSort(0,N-1);
     cout<<"\n\n\n\nSorted List : \n\n" ;
     for (int i=0;i<N;i++)
```

```
         cout<<arr[i]<<"    ";
      getch() ;
}
void QSort(int First,int Last)
{
   int P,L,R,T;
   L = First;
   R = Last;
   int M=(First+Last)/2+1;
   P = arr[M];
   while (L<=R)
   {
      while(arr[L]<P)
         L++;
      while (arr[R]>P)
         R--;
      if (L<=R)
      {
         T=arr[L];
         arr[L]=arr[R];
         arr[R]=T;
         L=L+1;
         R=R-1;
      }
   }
   if (First<R)
      QSort(First,R);
   if (L<Last)
      QSort(L,Last);
}
```

والشكل التالي يوضح إخراج البرنامج أعلاه.



مسرد المصطلحات

- **مفتاح** Key:

هو حقل مفرد في التعبير عن السجل

- **خوارزمية Algorithm:**

هي خطوات البرنامج مكتوبة بطريقة متسلسلة ومترابطة منطقياً ويحكمها المنطق الرياضي

- **مصفوفة** Array:

عبارة عن متغيّر مركّب يتكوّن من عدة عناصر كلها من جنس واحد تشترك في الاسم وتختلف في الموقع داخل المصفوفة.

- **الترتيب الفقاعي** Bubble sorting:

تقوم فكرة الترتيب الفقاعي علي مقارنه كل عنصر في القائمة مع العنصر الذي يليه ثم يقوم بإبدالها إذا لزم الأمر، وتكرر خوارزمية هذه العملية حتى تنتهي الحاجة إلي الإبدال أي حتى يتم التبديل. بمعنى آخر يقفز العنصر الأكبر إلي الطرف الأعلى في حين يبقى العنصر الآخر في العمق.

- **الترتيب بالاختيار** Selection sorting:

تقوم فكرة الترتيب بالاختيار بالبحث عن أصغر عنصر في القائمة، ونقوم بتبديله مع العنصر الأول، ثم نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الثاني إلي العنصر الأخير، ونقوم بتبديله مع العنصر الثاني، ثم نبحث عن أصغر عنصر من بين عناصر القائمة من العنصر الثالث إلي العنصر الأخير، ونقوم بتبديله مع العنصر الثالث، وهكذا إلي أن تنتهي كل عناصر المصفوفة وعندها نحصل علي مصفوفة مرتبة ترتيباً تصاعدياً.

- **الترتيب بالإدخال** Insertion sorting:

تقوم فكرة هذه الطريقة علي البحث عن أصغر عنصر في القائمة ويحفظ في متغير آخر ثم تزاح العناصر من **`Array[0]`** وحتى العنصر الذي يقع قبل مكان العنصر الأصغر الذي وجدناه فيتقدم كل عنصر خطوة إلي الإمام ونضع العنصر الأصغر الموجود في المتغير المؤقت في موقع العنصر **`Array[0]`** ثم نكرر العملية السابقة ولكن في هذه المرة مع العناصر من **`Array[1]`** وحتى آخر عنصر في القائمة وهكذا نكرر هذه العملية حتى آخر عنصر في القائمة.

- **الترتيب السريع** Quick sorting:

تقوم فكرة الترتيب السريع باختيار عنصر معين من العناصر المراد ترتيبها ويسمي هذا العنصر بالقطب أو المحور**`Pivot`** ومن ثم يتم تقسيم العناصر المتبقية إلى مجموعتين تعتبر كل مجموعة قائمة جزئية جديدة حيث تشمل المجموعة الأولى جميع العناصر الأقل من

العنصر المحور **Pivot** بينما تشمل المجموعة الثانية جميع العناصر الأكبر من العنصر المحور **Pivot** ويتوسط العنصر المحور **Pivot** هاتين المجموعتين ونلاحظ أن القائمة ترتبت ترتيباً جزئياً. وبتطبيق الترتيب السريع على عناصر المجموعتين الأولى والثانية مرة أخرى كل على حدة تترتب القائمة أكثر فأكثر وهكذا نقوم بتطبيق فكرة الترتيب السريع على مجموعة جديدة إلى أن تترتب القائمة بشكل كامل.

المراجع

1. William Ford and William Topp, **Data Structure with C++**, Prentice-Hall Inc 1996.
2. Sara Baase ad Allen Van Gelder, **Computer algorithms**, Addison Wesley Longman 2000.
3. Deitel and Deital, **C++ How to program**, Prentice-Hall Inc 2001.

الوحدة الرابعة

أشكال البيانات المجردة



محتويات الوحدة

الموضوع	الصفحة
المقدمة	**86**
تمهيد	86
أهداف الوحدة	87
1.المكدسات	**88**
1.1مفهوم المكدسات	88
2.1 إنشاء فصيلة للمكدسة	91
3.1 تعريف الدوال الأعضاء	92
1.3.1 إنشاء المكدس	92
2.3.1 التأكد من أن المكدس خالٍ	93
3.3.1 حذف عنصر من المكدس	93
4.3.1 إضافة عنصر إلى المكدس	94
4.1 تطبيقات على المكدس	95
2.الصفوف	**99**
1.2 مفهوم الصف	99
2.2 أنواع الصفوف	101
1.2.2 الصف الخطي	101
2.2.2 الصف الدائري	103
3.2 إنشاء فصيلة للصفوف	105
4.2 تعريف الدوال الأعضاء	106
1.4.2 إنشاء الصف	106
2.4.2 التأكد من أن الصف خالٍ	106
3.4.2 حذف عنصر من الصف	107

الموضوع	الصفحة
4.4.2 إضافة عنصر إلى الصف	108
5.4.2 التأكد من أن الصف ممتلئ	109
5.2 الصفوف ذات الأولوية	111
6.2 تطبيقات على الصفوف	117
الخلاصة	**119**
لمحة مسبقة عن الوحدة التالية	**120**
إجابات التدريبات	**121**
مسرد المصطلحات	**124**
المراجع	**125**

المقدمة

تمهيد

عزيزي الدارس،

مرحبا بك إلى هذه الوحدة

من دراستك للوحدة السابقة عرفت أن تصنيف المعلومات من الاستخدامات المتكررة في معالجة البيانات، وأن التصنيف هو تسهيل وتسريع عملية البحث عن السجلات بالإضافة لاستخدامات أخرى.

تنقسم هذه الوحدة إلى قسمين رئسين، نتناول في قسمها الأول:

المكدسات التي تستخدم في نظم التشغيل المختلفة، سنقوم بتعريف المكدسة وسنبين أهم الخصائص التي تتسم بها، وكيف أن كثيراً من المواقف الحياتية اليومية تشابه المكدسات وذلك لاستخدام مبدأ الـ LIFO الداخل أخيراً هو الخارج أولاً، ثم ننتقل بك إلى عمليات المكدسات وفيها ندرس عدداً من الدوال وفيها نقوم بتعريف الدوال الأعضاء.

أما القسم الثاني (الصفوف) والتي تستخدم في عمليات المحاكاة، هنا نجد قائمة من العناصر تحفظ بطريقة خطية لا تختلف عن المكدسات إلا في العنصر الأول في الإدخال ه والعنصر الأول في الإخراج (الحذف) وهذا ما يعرف بأسلوب FIFO، وفي هذا القسم سنوضح نوعي الصفوف:

- ❖ الصف الخطي.
- ❖ الصف الدائري.

عزيزي الدارس، كثير من الخدمات في حياتنا اليومية تقوم في نظام الصفوف إلا انه أحياناً تكون هناك بعض الأولويات في تقديم الخدمات لذا ظهرت فكرة الصفوف ذات الأولوية التي قادت إلى التفكير في عمل الصف باستخدام خصائص القوائم المتصلة ، وهذا ما سندرسه في الوحدة القادمة.

هناك الكثير من الأمثلة وحلولها لتسهيل توصيل المادة إليك، كما أن هناك عـــدد مـــن التدريبات وأسئلة التقويم الذاتي الواردة في ثنايا الوحدة بغرض تدعيم وتعزيز عمليـــة الـــتعلم أرجو الاهتمام بها.

نرحب بك مرة ثانية في هذه الوحدة، متمنين أن تفيد منها وتستمتع بدراستها.

فهلا تفضلت إلى هذه الوحدة العامرة.

أهداف الوحدة



عزيزي الدارس، بعد فراغك من هذه الوحدة ينبغي أن تكون قادرا على أن:

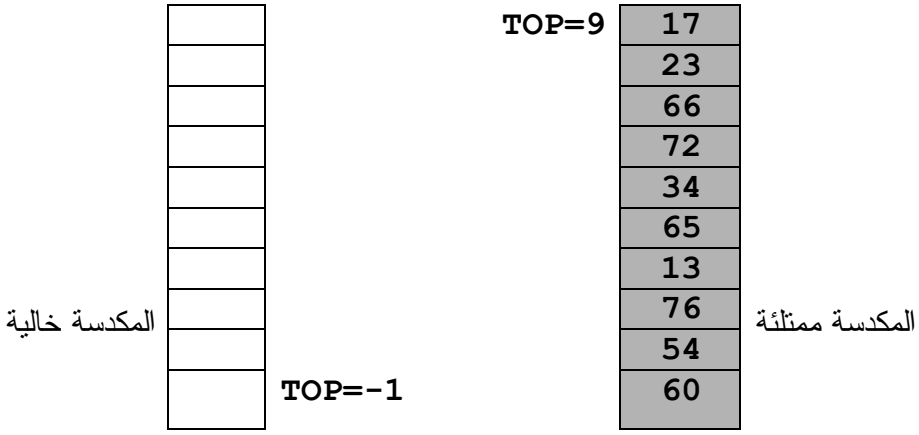
1. **تُعرف** المكدسة.
2. **تُعدد** الخواص التي تتسم بها المكدسة.
3. **تذكر** الدوال التي تتعامل مع المكدسات كعمليات.
4. **تشرح** كيف نضيف ونحذف العنصر من وإلى المكدسة.
5. **تشرح** فكرة نقل محتويات مكدسة إلى مكدسة أخرى ، ثم تكتب دالة بلغة C++ توضح ذلك.
6. **تُعرف** الصف وتحدد أنواعه.
7. **تُقارن** بين المكدسات والصفوف.
8. **تُعدد** الدوال التي تتعامل مع الصفوف كعمليات.
9. **تُقارن** بين الصف الخطي والدائري.

1.المكدسات Stacks

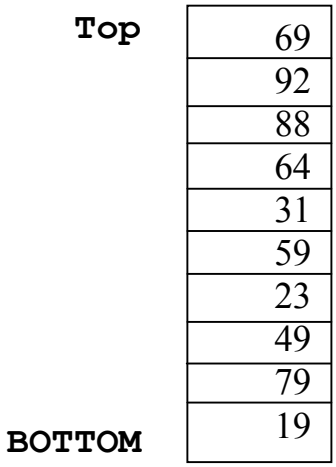
1.1 مفهوم المكدسة Stack Concept

المكدسات هي هيكل من هياكل البيانات الخطية المتغيرة وتتكون من مجموعة من العناصر أو القيم مخزنة بطريقة معينة وخاصة، (تسمى المكدسة في بعض الأحيان بالبيانات المتراصة)، العنصر الأول في المكدسات يشكل قاع المكدس (**BOTTOM**) والعنصر الأخير(**Top**) يشكل وجه المكدس أو قمته، لذلك نجد أن الإضافة والحذف يجريان من طرف واحد هو قمة المكدسة أي من الأعلى.

ويمكن تمثيل المكدس بالشكل التالي:-



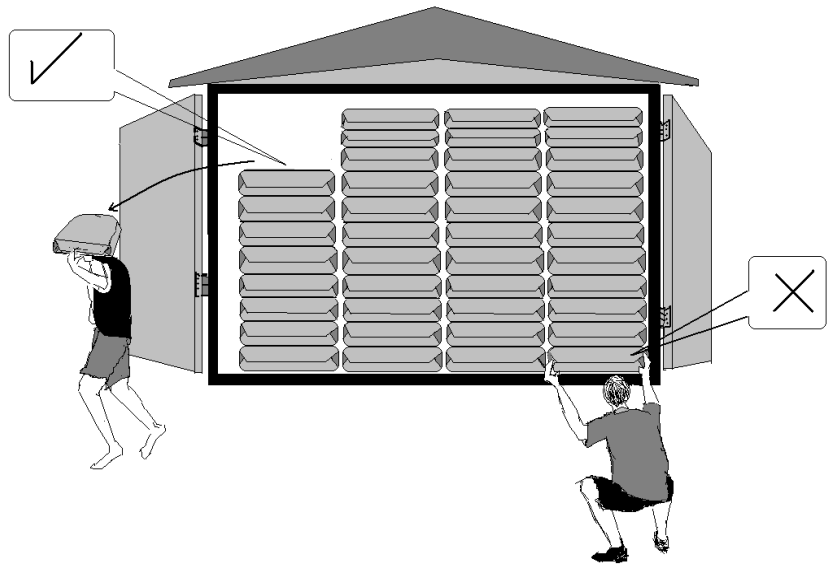
ليكن لدينا المكدسة التالية :



من تعريف المكدسة أن العنصر الممكن حذفه هو العنصر 96 أي أن آخر عنصر يتم إدخاله في المكدسة هو الذي يحذف أولاً، ولهذا السبب يطلق على هذا النوع من أشكال البيانات المجردة اسم LIFO اختصاراً لـ (Last In First Out)

ويستخدم مبدأ ألـ **LIFO** (**Last In First Out**) الداخل أخيرا هو الخارج أولا ً، لذلك فالإضافة والحذف يجريان من طرف واحد هو قمة المكدسة أي من الأعلى.

تخيل أن تاجراً ما أراد أن يعطيك جولاً من سكر من أول دفعة تم تخزينها مع العلم أن المخزن فيه المئات من الجولات. كم من الوقت ستستغرق هذه العملية؟ وكم من الجهد سيبذل الحمال لرفع كمية كبيرة جداً من الجولات ليعطيك مبتغاك ؟ وهل ترى أن هذه الطريقة صحيحة ؟ (أي أن ما دخل أولاً يخرج أولاً)؟

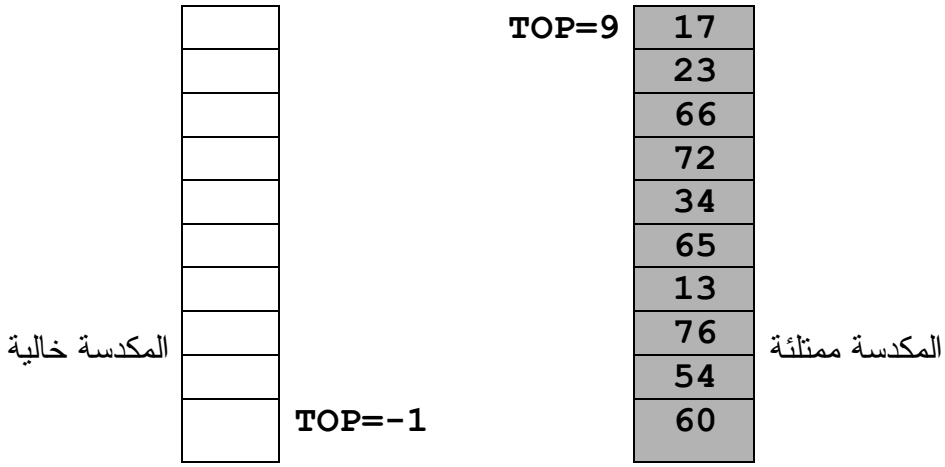


شكل (1) يوضح مبدأ الـ LIFO في المكدسات

لذا نجد في كثير من المواقف التي تواجهنا في حياتنا اليومية استخدام للمكدسات أي تستخدم مبدأ أل **LIFO** الداخل أخيرا هو الخارج أولاً مثلاً في المخازن (مخزن أسمنت أو مواد بناء) تتم عملية صرف البضاعة من المخزن من آخر بضاعة تم إدخالها إلى المخزن (إلا بضاعةً كادت مدة صلاحيتها تنتهي) وعلية فإن معظم المخازن تعمل علي مبدأ ألـ **LIFO.**

ويتسم المكدس بالخصائص التالية:

1. ديناميكي، يزداد ويتقلص حسب الرغبة.
2. الوصول إلى أي عنصر يتم عبر القمة أي لا يمكن الوصول إلى أي عنصر إلا بعد حذف جميع العناصر الأعلى منه.
3. تتطلب المكدسات تعريف مؤشر واحد يشير إلى قمة المكدس نطلق عليه الـــ Top .
4. تزداد قيمة المؤشر بمقدار واحد عند كل عملية إضافة وينقص بمقدار واحد كلما كانت هنالك عملية حذف.
5. يصبح المكدس خالياً إذا كان المؤشر يساوي صفراً أي Top=0 .



الشكل يوضح أن المكدس يمثل مجموعة مواقع فيزيائية في الذاكرة الرئيسة للحاســـوب وهي متتالية وعملية الوصول إلي هذه المواقع تتم من الموقع الأعلى الذي يشير إليه المؤشـــر **Top** و عندما يكون المكدس فارغاً (خالياً)فان الـــ **TOP =-1** وعند إضافة عنصـــر إلـــى المكدس يصبح المؤشر **TOP = TOP + 1** أما إذا تم حذف عنصر من المكـــدس فـــان **TOP = TOP-1** ويستمر في النقصان عند كل عملية حذف إلي أن يصبح **TOP =-1** وهو الوضع الذي يجعل المكدسة خالية .

أسئلة تقويم ذاتي



1. عرف المكدسة.
2. اذكر الخصائص التي تتسم بها المكدسة.
3. اشرح مستخدماً أشكال المكدس كيف نضيف ونحذف العناصر من المكدس.
4. اذكر حالة المكدسة في كل مما يأتي:

Top=-1
Top=Top+1
Top=Top-1

2.1 إنشاء فصيلة للمكدسة

هنالك عدد من الدوال يمكن أن تتعامل مع المكدسات كعمليات ، نذكر من هذه الدوال الأتي :

1. **إنشاء المكدس:-**
2. **التأكد من أن المكدس خالٍ:-**
3. **حذف عنصر من المكدس (POP)**
4. **إضافة عنصر إلى المكدس (PUSH)**
5. **التأكد من أن المكدس ممتلئ أو غير ممتلئ FULL_STACK**

وبالتالي يمكن وضع جميع الدوال في صنف واحد أو فصيلة **class** يتسق مع مفهوم البرمجة الكائنية يسمي **stack** [1].

```
#include <iostream.h>
#include <conio.h>
const  int  max_stack_size=4;
class stack
{
   private:
     int stack_list[max_stack_size-1];
     int top  ;
   public:
     stack();
     int empty_stack();
     int pop();
     void push(int);
      int full_stack();
};
```

تقويم ذاتي



اذكر الدوال التي تتعامل مع المكدسة كعمليات وكيف توضع الدوال حتى تتسق مع مفهوم البرمجة؟

3.1 تعريف الدوال الأعضاء

1.3.1 إنشاء المكدس

الدالة **stack()** التالية هي دالة بناء تنفذ بمجرد الإعلان عن هدف يتبع إلى الفصيلة وتقوم بوضع قيمة أولية لمؤشر القمة **top=-1**.

---

[1] قد تحتوي الفصيلة علي عدد آخر من الدوال ولكن نسبة لمتطلبات المقرر سنكتفي بهذا القدر.

معلومة

- ✓ دالة البناء يجب أن تكون بنفس اسم الفصيلة (الصنف)، حتى يتم للمترجم معرفة دالة البناء ليقوم باستدعائها تلقائياً.
- ✓ للدالة البناء ليس لها قيمة راجعة لان تنفيذها يتم تلقائياً كما ذكرنا سابقاً.

ويمكن تعريف الدالة علي النحو التالي

```
stack::stack ()
{
     Top=-1;
}
```

ويمكن أيضاً تعريفها علي النحو التالي

```
stack::stack (): top (-1)
{}
```

2.3.1 التأكد من أن المكدس خالٍ

الدالة **empty_stack()** التالية تقوم بالتأكد من المكدسة خالية أو غير خالية وذلك من خلال قيمة **top** فإذا كانت تساوي **-1** تقوم بإرجاع **1 (True)** أي أن المكدسة خالية فعلاً وإلا ستقوم بإرجاع **0 (False)** وهي أن المكدسة غير خالية

```
int stack::empty_stack()
{
     return top==-1;
}
```

3.3.1 حذف عنصر من المكدس (POP)

الدالة **pop()** التالية تقوم بالتأكد من المكدسة خالية أو غير خالية وذلك من خلال قيمة **top** فإذا كانت تساوي **-1** تقوم بطباعة رسالة خطأ هي "محاولة حذف عنصر من مكدسة خالية أصلاً" وإلا تقوم بـــ :

1. قراءة العنصر الموجود في قمة المكدسة
2. نقصان قيمة القمة بمقدار واحد
3. إرجاع العنصر المحذوف.

```
int stack::pop()
{ if (top==-1)
 {
   cout<<"\nattempt to pop an empty stack";
   exit(1);
 }
 else
 {    int item =stack_list[top];
    top--;
    return item;
 }
}
```

4.3.1 إضافة عنصر إلى المكدس (PUSH)

الدالة **push(int item)** التالية تقوم بالتأكد من المكدسة ممتلئة أو خالية وذلك من خلال قيمة **top** فإذا كانت تساوي **max_stack_size** تقوم بطباعة رسالة خطأ هي "المكدسة ممتلئة أصلاً" وإلا تقوم بزيادة قيمة قمة المكدسة بمقدار واحد وجعل العنصر الجديد قي قمة المكدسة .

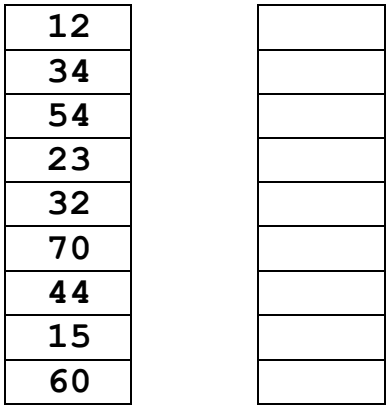
void stack::push(int item)

```
{
  if (top==max_stack_size -1)
  {
     cout<<"\n Stack Is Full Overflow \n";
     exit(1);
  }
  ++top;
  stack_list[top]=item;
}
```

4.1 تطبيقات على المكدسات

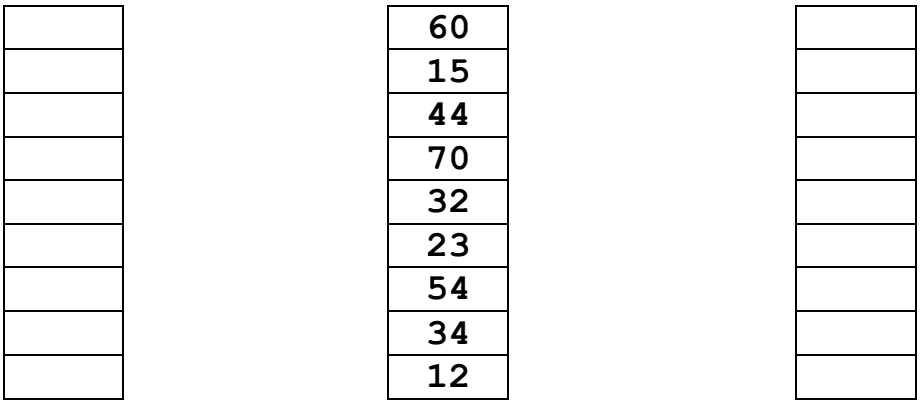
أمثلة

1ــ اشرح فكرة نقل محتويات مكدسة إلى مكدسة الأخرى، ثم اكتب دالة بلغة **C++** توضح ذلك .

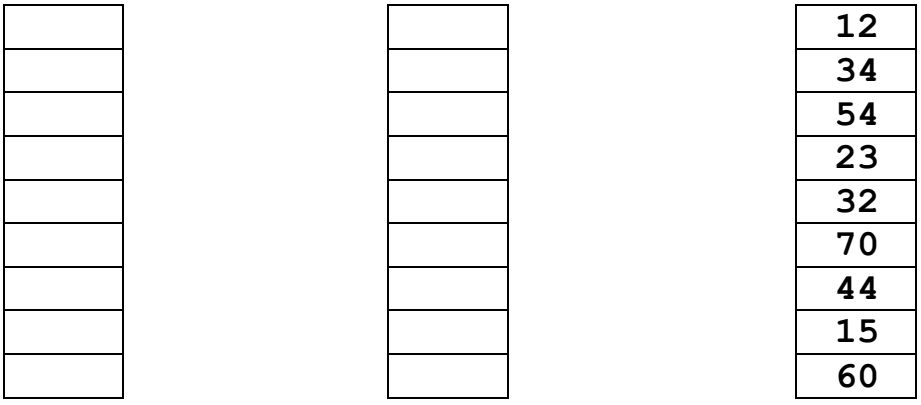


نلاحظ عند أخذ أول عنصر في قمة المكدسة **S** وهو **12** وإضافة هذا العنصر إلى المكدسة الثانية **S1** ،سوف يصبح العنصر **34**في قمة المكدسة **S** وعندما نستمر علي هذا المنوال نلاحظ أن كل عناصر المكدسة **S** انتقلت إلى المكدسة الثانية **S1** ولكن بصورة معكوسة عليه فان عملية النقل لم تتم بصورة صحيحة .

إذن لابد من وجود مكدسة مؤقتة **temp** نقل إليها عناصر المكدسة **S** (سوف تكون بصورة معكوسة كما هو مبين أدناه)



ونقوم بنقل عناصر المكدسة المؤقتة **temp** إلى المكدسة **S1** وستكون معكوسة مـــع عناصر المكدسة المؤقتة**temp** ومرتبة مع عناصر المكدسة **S** . وتصبح عندئـــذ المكدســـة منقولة من المكدسة **S** إلى المكدسة **S1** وهو يوضح المطلوب




```
stack s,s1,temp;
while (!s.empty_stack())
   temp.push(s.pop());
while (!temp.empty_stack())
   s1.push(temp.pop());
```

2. اكتب إخراج الشفرة البرمجية التالية

```
stack s;
int i=15,j=25;
s.push(j);
s.push(10);
s.push(20);
s.push(s.pop());
s.push(30);
s.push(i);
s.push(50);
j=s.pop();
i=s.pop();
s.push(60);
s.push(s.pop());
cout<<j<<endl;
while (!s.empty_stack())
{
   i=s.pop();
   cout<<i<<endl;
}
```

الإخراج هو:

```
50
60
30
20
10
25
```

نشاط



✦ أكتب إخراج الشفرة البرمجية التالية

```
2. Stack S;
S.push(25);
S.push (10);
S.push (40);
S.push (43);
S.push (50);
S.push (33);
S.push (55);
int j= S.pop();
int i= S.pop();
S.push (j);
S.push (S.pop());
while (!S.empty_queue())
{
   cout<<i<<endl;
   i=S.pop();
}
cout<<i<<endl;
```

✦ يمكن استخدام المكدسات في عملية تحويل الأعداد من نظام العشري إلى أي نظام آخر ، والخوارزمية التالية تبين ذلك:

<1>.أنشي مكدسة **S**

<2>.ادخل العدد العشري **n** وأساس النظام المحول إليه **b**.

<3>.أدخل قيمة **n%b** إلى المكدسة **S**.

<4>.أجعل **n=n/b**.

<5>.كرر الخطوة <3>و<4> حتى تصبح **n=0**.

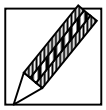
<6>.اخرج كل محتويات المكدسة **S** (الناتج هي العدد الجديد في نظام الأعداد الذي أساسه يساوي **b**).

<7>.النهاية.

أكتب برنامجاً بلغة **C++** يقوم بعمل الخوارزمية أعلاه.

ناقش ما ورد مع زملائك وأعرض إجابتك على مشرفك الميداني.

تدريب (1)



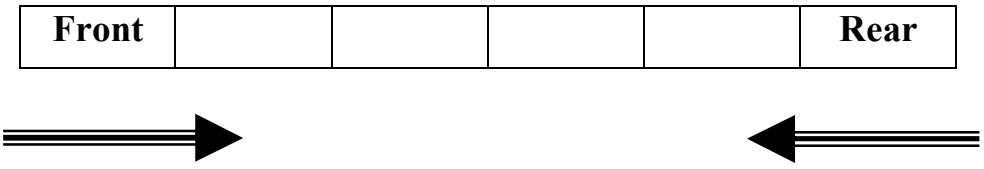
1. اكتب دالة بلغة **C++** توضح نسخ محتويات مكدسة إلى مكدسة أخرى.
2. ما الغرض من الشفرة البرمجية التالية

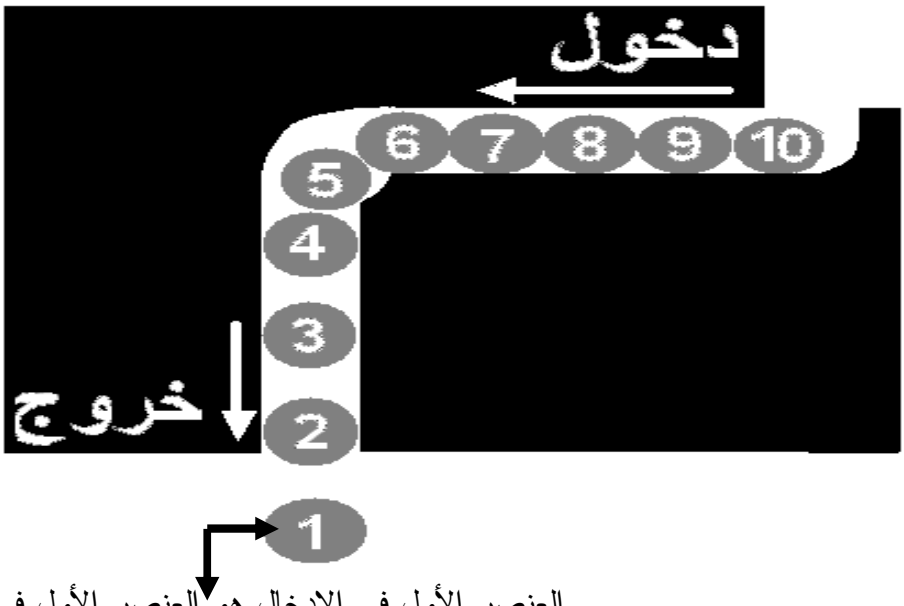
```
int size_stack(stack m)
{
  int sz=0,y;
  while(!m.empty_stack())
  {
    y=m.pop();
    sz++;
  }
```

2. الصفوف Queues

1.2 مفهوم الصف Queue Concept

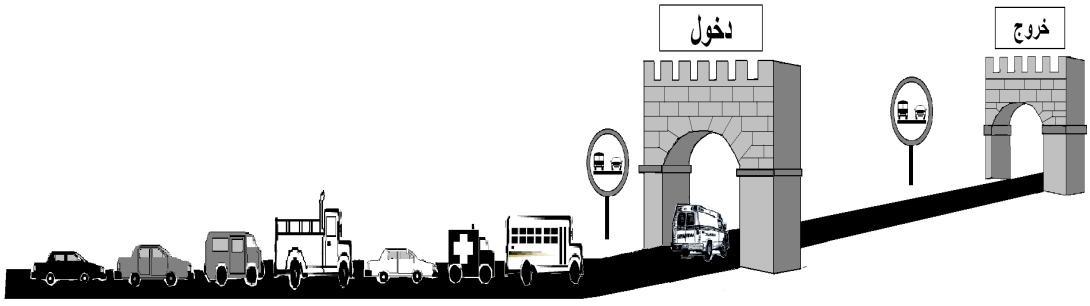
الصف هو قائمة من العناصر تحفظ بطريقه خطية ولا يختلف عن المكدسات إلا في العنصر الأول في الإدخال هو العنصر الأول في الإخراج (الحذف)أيضاً وهذا الاسلوب يعرف بأسلوب**FIFO** (**First In First Out**) في الوصول إلى البيانات، وهذا يعنى أن حذف العناصر يتم من طرف واحد ويسمى المقدمة أو الرأس **Front**، وإضافة العناصر أيضا تتم من الطرف الآخر الذي يسمى الذيل أو المؤخرة **Rear**.





العنصر الأول في الإدخال هو العنصر الأول في الإخراج

شكل (2)



شكل (3)

شكل (2 – 3) يوضحان مبدأ الـــ (FIFO)

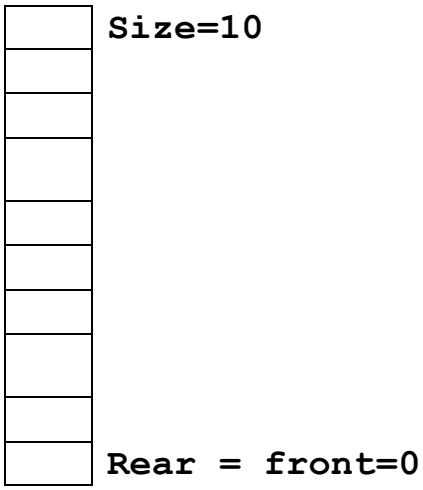
تقويم ذاتي

1. عرف الصف.
2. ما الفرق بين المكدسات والصفوف؟

2.2 أنواع الصفوف

1.2.2 الصف الخطي

محدود الحجم أي أن عدد العناصر محدود أو معروف مسبقاً ولـــه ســـعة محـــددة و شرط امتلائه هو أن **Rear = Size** أي أن قيمة الذيل = عدد العناصر(الحجم).



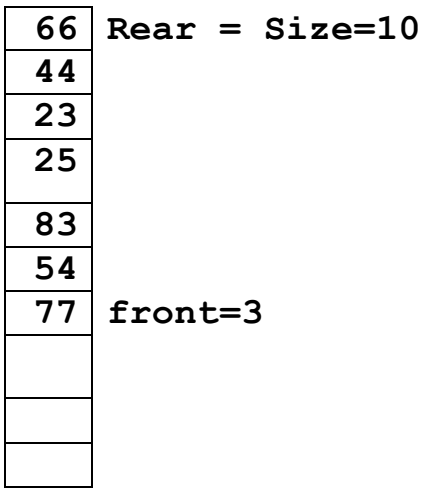
صف خطي فارغ — صف خطي غير ممتلئ

الشكل (ب) — الشكل (أ)

نلاحظ أن الشكل (أ) أعلاه يبين صفاً خطياً غير ممتلئ حيث تكون قيمـــة الـــذيل > الحجم وأن قيمة الرأس =**0** ، قد تكون قيمة الرأس في بعض الحالات أكبر مـــن **0** و قيمـــة الذيل أصغر من قيمة عدد العناصر(الحجم) ففي هذه الحالة يكون الصف غير ممتلئ أيضاً .

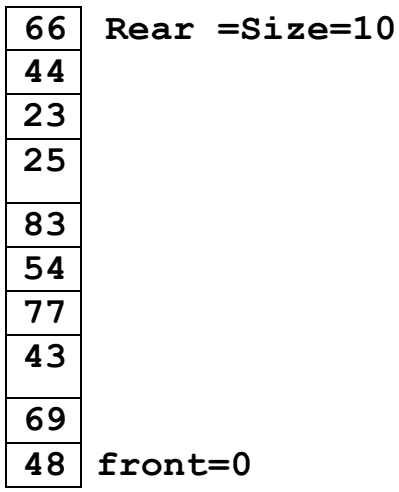
أما الشكل (ب) أعلاه يبين صفاً خطياً فارغاً فيه قيمة الذيل = قيمة الـــرأس = صـــفر أي أن **Rear = front=0** .

و إذا نظرت إلى الشكل (أ) أدناه تجده صفاً خطياً ُممتلئاً، أي عندما كانت قيمة الـــذيل = قيمة الحجم=**10** و كانت قيم المقدمة = **0** و في الشكل (ب) نجد أن قيمـــة المقدمـــة = **3** أي أنها اكبر من **0**، مما يعني وجود خانات فارغة، ورغم ذلك، لا نستطيع إضـــافة أي عنصر للصف لأن الخانات فارغة من جهة المقدمة.



صف خطي ممتلئ

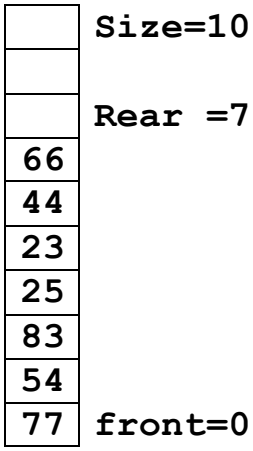
الشكل (ب)



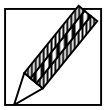
صف خطي ممتلئ

الشكل (أ)

ولكي تتمكن من إضافة أي عنصر للصف نقوم بعملية إزاحة جميع العناصر بحيث يصبح قيمة المقدمة (الرأس) =0 وقيمة الذيل=7 وهي أقل من الحجم =10 ولكن أسلوب عملية الإزاحة لا يتسم بالكفاءة رغم أنه يتم علي أرض الواقع في الحياة العملية.



تدريب (2)



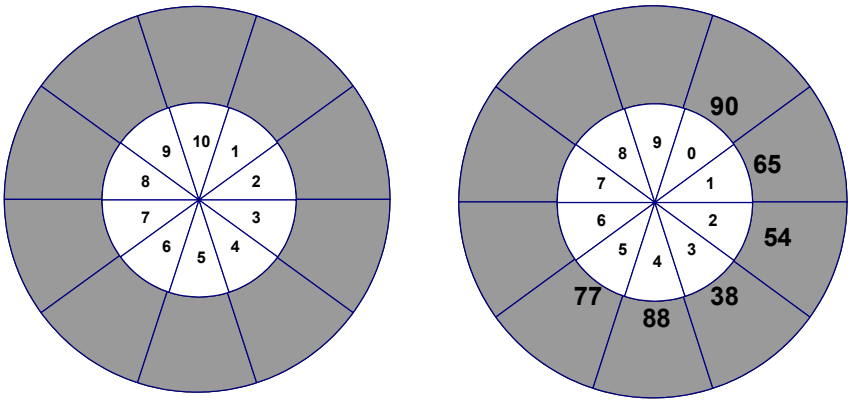
1. كون صفاً خطياً غير ممتلئ حيث تكون قيمة الذيل = الحجم وقيمة الرأس > 0 ؟
2. كون صفاً خطياً غير ممتلئ حيث تكون قيمة الذيل < الحجم وقيمة الرأس > 0 ؟

تقويم ذاتي

1. ما هو الصف الخطي وما هي شروط امتلائه؟

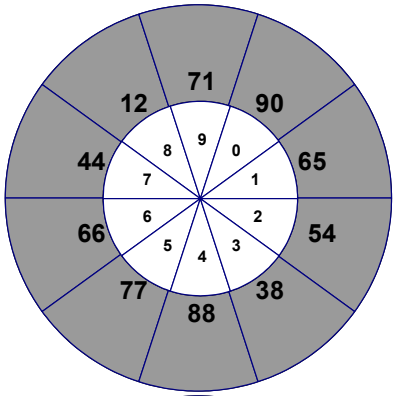
2.2.2 الصف الدائري

هو أسلوب آخر يتجنب فكرة الإزاحة مطلقاً وتقوم فكرته علي اتصال آخر الصف بأولـــه، نلاحظ أن الشكل (أ) أدناه يبين صفاً دائرياً غير ممتلئ حيث تكون قيمة الذيل =**6** وهي أقـــل من الحجم =**10** وأن قيمة الرأس =**0** ، قد تكون قيمة الرأس في بعض الحالات أكبر من **1** و قيمة الذيل أصغر من قيمة عدد العناصر(الحجم) ففي هذه الحالة يكون الصف غير ممتلئ أيضاً ، بينما أن الشكل (ب) أدناه يبين صفاً دائرياً فارغاً فيه قيمة الذيل = قيمـــة الـــرأس = صفر وقيمة الحجم تساوي **10** أي أن **Size=10** وان **Rear = front=0.**



الشكل (أ) الشكل (ب)

نلاحظ في الشكل (أ) أعلاه أن قيمة الذيل =**6** وهي أقل من الحجم =**10** أي أنه يمكـــن إضافة **4** عناصر إضافية لهذا الصف حتى تصل قيمة **rear** إلي قيمـــة الحجـــم = **10.** فمثلاً عند إضافة العنصر **66** إلى الصف المبـــين فـــي الشـــكل (أ) أعـــلاه نجـــد أن قيمـــة **rear=7**، وعند إضافة العنصر **44** نجد أن قيمة **rear =8**،و عند إضافة العنصر **12** تصبح قيمة **rear=9**وعند إضافة العنصر **71** تصـــير قيمـــة **rear=10** وهـــي تساوي الحجم ،



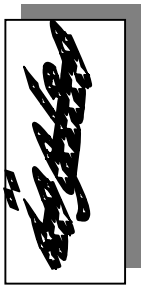
وهنا تبين لنا أن الصف قد امتلأ ،إذن فشرط امتلاء الصف – إذن – هو:

Front = 0

Rear = Size

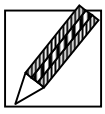
عند حذف عنصر من الصف وهو العنصر **90** تصبح قيمة **front=1** ليصبح العنصر **65**في مقدمة الصف

إذن الصف غير ممتلئ، وعند حذف عنصر ثانٍ وثالث من الصف وهما العنصر **65** والعنصر **54** تصبح قيمة **front=3** ليصبح العنصر **38** في مقدمة الصف وفي هذه الحالة يمكن إضافة **3** عناصر جديدة في المواقع **0,1,2** وذلك بعد تغيير قيمة **rear** من **10** إلي **0,1,2** علي الترتيب عند كل إضافة ، ويصبح الصف ممتلئاً ، وهنا يظهر شرط آخر يبين أن الصف ممتلئ وهو **Front=Rear + 1**



إن الشكلين (أ، ب) للصف الدائري هما شكلان افتراضيان ليس أكثر، ويمكن اعتبار الشكلين السابقين أداة لتسهيل عملية تصور الصفوف الدائرية. أما التمثل الحقيقي للصفوف الدائرية فهي نفس المتجهات العادية المستخدمة في الصفوف الخطية.

تدريب(3)



كون صفاً دائرياً غير ممتلئ حيث تكون قيمة الذيل< الحجم وقيمة الرأس > صفر.

أسئلة تقويم ذاتي

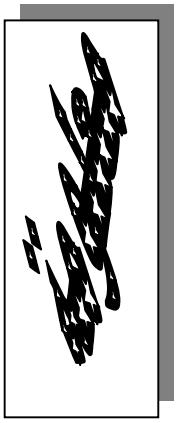


1. عرف الصف الدائري وما هي فكرته الأساسية؟
2. ما هي شروط امتلاء الصف الدائري؟
3. كيف نحذف ونضيف عنصراً في الصف الدائري؟

3.2 إنشاء فصيلة للصفوف

هنالك عدد من الدوال يمكن أن تتعامل مع الصفوف كعمليات، نذكر من هذه الدوال الأتي :

1. **إنشاء الصف :-**
2. **التأكد من أن الصف خالٍ (empty_queue)**
3. **حذف عنصر من الصف (QDelete)**
4. **إضافة عنصر إلى الصف (QInsert)**
5. **التأكد من أنَّ الصف ممتلئ (full_queue)**



ليس بالضرورة أن تكون عناصر الصف أو المكدسة من الأعداد الصحيحة فقط، بل يمكن استخدام جميع أنواع البيانات وسوف نستخدم في فصيلة **Queue** الافتراض **DataType** وإذا رغبت في أن يكون **DataType** من النوع **int** فما عليك إلا أن تخبر المترجم بذلك من خلال العبارة التالية

```
Typedef int DataType;
```

وبالتالي يمكن وضع جميع الدوال في صنف واحد أو فصيلة **class** يتسق مع مفهوم البرمجة الكائنية يسمي **Queue** .

```
#include <iostream.h>
#include <stdlib.h>
const int  max_queue_size=10;
class Queue
{
   private:
     DataType queue_list[max_queue_size];
     DataType rear,front,nItems  ;
   public:
     Queue();
     int empty_queue();
     DataType QDelete();
     void QInsert(DataType);
      int full_queue();
};
```

4.2 تعريف الدوال الأعضاء

1.4.2 إنشاء الصف

الدالة **Queue()** التالية هي دالة بناء تنفذ بمجرد الإعلان عن هدف يتبع إلى الفصيلة وتقوم بوضع قيمة أولية للمؤشرات المستخدمة في الصف .

ويمكن تعريف الدالة علي النحو التالي

```
Queue::Queue(): rear(0),front(0),nItems(0)
{
}
```

2.4.2 التأكد من أن الصف خالٍ

الدالة **empty_queue()** التالية تقوم بالتأكد من خلو الصف أو عدم خلوه وذلك من خلال قيمة **nItems** فإذا كانت تساوي **0** تقوم بإرجاع **1 (True)** أي أن الصف فعلاً خالٍ وإلا ستقوم بإرجاع **0 (False)** وهي أن الصف غير خالٍ.

```
int Queue::empty_queue()
{
     return nItems==0;
}
```

3.4.2 حذف عنصر من الصف (QDelete)

الدالة **QDelete()** التالية تقوم بالتأكد من الصف خالٍ أو لا وذلك من خلال قيمة **nItems** (مؤشر يستخدم كعداد لعد عناصر الصف) فإذا كانت تساوي **0** تقوم بإظهار رسالة خطأ هي "محاولة حذف عنصر من صف خالٍ أصلاً" وإلا تقوم بـــ:

- ✦ قراءة العنصر الموجود في مقدمة الصف
- ✦ نقصان قيمة **nItems** بمقدار واحد
- ✦ حساب قيمة **front=(front+1)% max_queue_size**
- ✦ إرجاع العنصر المحذوف

```
DataType Queue::QDelete()
{
  DataType item ;
  if (nItems==0)
  {
    cout<<"\nDeleting from an empty queue";
    exit(1);
  }
  else
  {
    item =queue_list[front];
    nItems--;
    front=(front+1)% max_queue_size ;
    return item;
  }
}
```

4.4.2 إضافة عنصر إلى الصف (QInsert)

الدالة **QInsert(DataType item)** التالية تقوم بالتأكد من أن الصف ممتلئ أو غير ممتلئ وذلك من خلال قيمة **nItems** فإذا كانت تساوي **max_queue_size** تقوم بطباعة رسالة خطأ هي "الصف ممتلئ أصلاً" وإلا تقوم بزيادة قيمة **nItems** بمقدار واحد وتعديل قيمة ذيل الصف من خلال **rear=(rear+1)% max_queue_siz** وجعل العنصر الجديد في ذيل الصف .

```
void Queue::QInsert(DataType item)
{
  if (nItems==max_queue_size)
  {
     cout<<"\n Queue Is Full Overflow \n";
     exit(1);
  }
  nItems++;
  queue_list[rear]=item;
  rear=(rear+1)% max_queue_size ;
}
```

5.4.2 التأكد من الصف ممتلئ (full_queue)

الدالة **full_queue()** التالية تقوم بالتأكد من أن الصف ممتلئ أو غير ممتلئ وذلك من خلال قيمة **nItems** فإذا كانت تساوي **max_queue_size** تقوم بإرجاع **1(True)** وهي تعني أن الصف ممتلئ وإلا ستقوم بإرجاع **0 (False)** التي تعني أن الصف غير ممتلئ .

```
int Queue::full_queue()
{
 return nItems== max_queue_size;
}
DataType Queue::length_queue()
{
 return nItems ;
}
```

نشاط



- اكتب إخراج الشفرة البرمجية التالية

```
queue Q ;
Q.QInsert(25);
Q.QInsert(10);
Q.QInsert(40);
Q.QInsert(43);
```

```
Q.QInsert(50);
Q.QInsert(33);
Q.QInsert(55);
int j=Q.QDelete();
int i=Q.QDelete();
Q.QInsert(j);
Q.QInsert(Q.QDelete());
while (!Q.empty_queue())
{
   cout<<i<<endl;
   i=Q.QDelete();
}
cout<<i<<endl;
```

- تتم عملية دمج عناصر صفين في صف واحد (يجب أن يكون الصف الجديد قادراً علي استيعاب جميع عناصر الصفين و يكون من نفس النوع) وفقا للأتي

الإجراء الأول	التأكد من حجم الصف الأول
الإدخال	لا يوجد
المعالجة	دمج الصفين
الإخراج	لا يوجد
الإجراء النهائي	صف جديد يحتوى علي عناصر الصفين

أكتب دالة بلغة **C++** تقوم بعملية دمج عناصر صفين في صف واحد.

ناقش ما ورد مع زملائك وأعرض إجابتك على مشرفك الميداني.

5.2 الصفوف ذات الأولوية

من الملاحظ في الواقع العملي أنه و علي الرغم من إن الخدمات تقدم في نظام الصفوف بذات الطريقة إلا انه أحياناً تكون هناك بعض الأولويات في تقديم الخدمات ولنأخذ مثلاً :

دخول طلاب الشهادة السودانية إلى كلية معينة لا يتم بمن تقدموا أولاً (تخيل عملية الصف) بل يتم وفقاً للنسبة المئوية للطلاب، وعلية يتم ترتيب الصف وفقاً للنسبة المئوية وهي مؤشر الأولوية في الصف.

نلاحظ في المثال السابق أن الصف تنازلي ليس بالضرورة أن يكون هنا الصف دائماً تنازلياً بل يمكن أن يكون تصاعدياً.

كمثال آخر في طرح عطاء لتنفيذ مشروع لجامعة السودان المفتوحة فان الجامعة غير ملزمة بقبول أول عطاء قُدم لها بل يتم وفق المواصفات والمعايير التي تضعها الجامعة لقبول هذا العطاء.



شكل (4) يوضح فكرة الأولوية في الصفوف

ولنأخذ مثالاً آخر، وهو ترتيب الأرقام تصاعدياً في الصف التالي:

Front	3	6	7	12	23	34	37		Rear

إذا تم إدخال عنصر جديد فلنفترض انه **15** فهذا يعنى ضرورة وضعه بين الـــ **12** وال**23** وهو غير قابل في ظل عمل الصف العادي ولكننا يمكن أن نتجاوز ذلك من خـــلال الآتي :-

1. يتم الإدخال بذات الطريقة من مؤخرة الصف **Rear** .
2. يتم إرسال عناصر الصف إلي مصفوفة **Array** يتم فيها ترتيب العناصر بالطريقـــة المفترضة وهي الترتيب التصاعدي.
3. يتم إرسال العناصر إلي الصف من جديد بعد إعادة ترتيب العناصر.

Front	3	6	7	12	15	23	34	37	Rear

وهو ما يتطلب إعادة ترتيب الصف من كل مرة في حالة ما إذا كان العنصر الجديد أصغر من العنصر الذي في آخر الصف . ويلاحظ الآتي في تلك العملية :-

1. كثرة العمليات عليها وهو ما يتسبب في إطالة زمن التشغيل .
2. إهدار مساحة مضافة من الذاكرة (الصف + المصفوفة) .
3. عدم القابلية للتوسع باعتبار أن الصف يفترض الإعلان عن طوله مسبقاً وهو ما يتعارض مع الواقع الذي يتحول بكامله للزيادة أو النقصان .
4. يعمل في ظل أولوية واحدة فقط بينما إذا كانت الأولوية أكثر من واحدة فتعجز عن العمل.

فكان لابد من التفكير في طريقة حل امثل وهو ما قاد إلي التفكير في عمـــل الصـــف باستخدام خصائص القوائم المتصلة مع المحافظة على خاصية أن الخروج للعنصر يـــتم مـــن مقدمة القائمة. وهذا ما سوف نتناوله في الفصل القادم بإذن الله.

في المثال التالي نقوم بإدخال عناصر عشوائياً للصف وفي كل مرة يتم فيها الإدخـــال تتم عملية إعادة ترتيب الصف من جديد

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
const  int  max_pqueue_size=6;
typedef int DataType;
class PQueue
{
   private:
     DataType PQueue_list[max_pqueue_size];
     DataType nItems  ;
   public:
     PQueue();
     int empty_PQueue();
     DataType PQDelete();
     void PQInsert(DataType() ;
      int full_PQueue();
};
//*****************************************
int PQueue::empty_PQueue()
{
    return nItems==0;
{
//*****************************************
PQueue::PQueue(): nItems(0)
{}
//*****************************************
int PQueue::full_PQueue()
}
    return nItems==max_pqueue_size;
{
//*****************************************
DataType PQueue::PQDelete()
{
```

```
  if (nItems==0)
  {
    cout<<"\nDeleting from an empty queue";
    getch();
    exit(1);
  }
  return PQueue_list[--nItems];
}
//*****************************************
void PQueue::PQInsert(DataType item)
{
  if (nItems==max_pqueue_size)
  {
     cout<<"\n Queue Is Full Overflow \n";
     getch();
     exit(1);
{
  int j;
  if (nItems==0)
      PQueue_list[nItems++]=item;
  else
  {
    for (j=nItems-1;j>=0;j--)
    {
      if (item>PQueue_list[j])
          PQueue_list[j+1]=PQueue_list[j];
      else
          break;
    }
    PQueue_list[j+1]=item;
    nItems++;
  }
}
//*****************************************
void main()
{
   PQueue PQ1;
```

```
    randomize();
    cout<<"\n\nOriginal Queue";
    int rndd;
    for(int i=0; i<max_pqueue_size; i++)
    {
        rndd=rand() % 100;
        PQ1.PQInsert(rndd);
        cout<<"\nQ["<<(i+1)<<"]  : "<<rndd;
    }
    cout<<"\n\nPriority Queue";
    int j=1;
    while (!PQ1.empty_PQueue())
    {
       int i=PQ1.PQDelete();
       cout<<"\nPQ["<<(j++)<<"]  : "<<i;
    }
    getch();
}
```

محاور المقارنة	المكدسات	الصفوف
التعريف	هي قائمة من البيانات تحفظ بطريقة خطية يتم فيها حذف وإضافة البيانات من طرف واحد ويسمى القمة Top	قائمة من العناصر تحفظ بطريقة خطية بحث يمكن حذف العناصر من طرف واحد يسمى المقدمة أو الرأس Front بينما يتم إضافة العناصر من الطرف الآخر ويسمى المؤخرة أو الذيل Rear
الأنواع	ــــــــــــــــ	أنواع مختلفة أهمها الخطي والدائري
الإضافة والحذف	يستخدم مبدأ الـ LIFO (Last In First Out) الداخل أخيراً هو الخارج أولاً ، لذلك فالإضافة والحذف يجريان من طرف واحد هو قمة المكدسة أي من الأعلى	يستخدم مبدأ الـ FIFO (First In First Out) وهذا يعني أن حذف العنصر يتم من طرف واحد ويسمى المقدمة أو الرأس Front، وإضافة العناصر تتم من الطرف الآخر الذي يسمى الذيل أو المؤخرة Rear
وضع الدوال	يمكن وضع جميع الدوال في صنف واحد أو فصيلة Class يتسق مع مفهوم البرمجة الكائنية يسمى Stack	يمكن وضع جميع الدوال في صنف واحد أو فصيلة Class يتسق مع مفهوم البرمجة الكائنية يسمى Queue
دالة الإضافة	الدالة(Push (Int item تقوم بالتأكد من أن المكدسة ممتلئة أم لا وذلك من خلال قيمة Top	الدالة(Daa Type item) QDelete تقوم بالتأكد من أن الصف ممتلئ أم لا وذلك من خلال قيمة nItems
دالة الحذف	الدالة( ) Pop تقوم بالتأكد من أن المكدس خالٍ أو لا وذلك من خلال قيمة Top	الدالة( ) QDelete تقوم بالتأكد من الصف خالٍ أو غير خالٍ وذلك من خلال قيمة nItems

جدول (1) يوضح محاور المقارنة بين المكدسات والصفوف

6.2 تطبيقات على الصفوف

أمثلة:

1.اكتب دالة بلغة **C++** تسمي **Queue_copy** تقوم بعملية نسخ صف في صف آخر.

يتم ذلك بوجود صف آخر وسيط يتم فيه نقل عناصر الصف المرسل ومن ثم يتم إضافة جميع العناصر الموجود في الصف الوسيط إلى الصف الجديد المراد النسخ إليه .

```
void Queue_copy(Queue &Qu1,Queue &Qu2)
{
  if(Qu1.empty_queue())
  {
   err<<"attempt to copy an empty queue\n";
   exit(1);
  }
  DataType item;
  Queue Qtemp;
  while (!Qu1.empty_queue())
  {
    item=Qu1.QDelete();
    Qtemp.QInsert(item);
  }
  while (!Qtemp.empty_queue())
  {
    item=Qtemp.QDelete();
    Qu1.QInsert(item);
    Qu2.QInsert(item);
  }
}
```

2.اكتب إخراج الشفرة البرمجية التالية

```
queue Q
int i=15,j=25;
Q.QInsert(j);
```

```
Q.QInsert(10);
Q.QInsert(20);
Q.QInsert(Q.QDelete());
Q.QInsert(30);
Q.QInsert(i);
Q.QInsert(50);
j=Q.QDelete();
i=Q.QDelete();
Q.QInsert(60);
Q.QInsert(Q.QDelete());
cout<<j<<endl;
while (!Q.empty_queue())
{
   i=Q.QDelete();
   cout<<i<<endl;
}
```

الإخراج هو:

```
30
15
50
60
25
```

الخلاصة

اشتملت هذه الوحدة على تعريف بعض من أشكال البيانات المجردة وهي المكدسات والصفوف، ويمكننا اعتبار الأنماط المجردة بأنها البداية الحقيقية لطريقة البرمجة الموجهة جهة الموضوع Object Oriented Programming Language (OOPL).

ولأشكال البيانات المجردة كثير من التطبيقات في مجالات علوم الحاسوب المختلفة، فالمكدسات تزداد وتتقلص حسب الرغبة، والوصول إلى أي عنصر يتم عبر القمة أي بعد حذف جميع العناصر التي أعلى منه، وتستخدم المكدسات في نظم التشغيل المختلفة، أما الصفوف فهي تستخدم في عملية المحاكاة.

وفي هذه الوحدة قمنا بتوظيف بعض أنواع الدوال وعرفنا طرق استخدامها داخل البرنامج.

ومن ضمن ما عرفناه في هذه الوحدة وفي قسمها الأول (المكدسات) مبدأ LIFO أي الداخل أخيراً هو الخارج أولاً، وفي نفس القسم وقفنا على أهم سمات وخصائص المكدس.
وفي القسم الثاني (الصفوف) عرفنا مبدأ FIFO العنصر الأول في الإدخال هو العنصر الأول في الإخراج (الحذف).

وفي نفس القسم أيضاً تعرفنا على نوعي الصف وهما:

✯ الصف الخطي وهو محدود الحجم وله سعة محدودة وشرط امتلائه هو
قيمة الذيل = عدد العناصر (الحجم) Rear=Size.

✯ الصف الدائري أسلوب يتجنب فكرة الإزاحة وتقوم فكرته على أن يتصل آخر الصف بأوله.

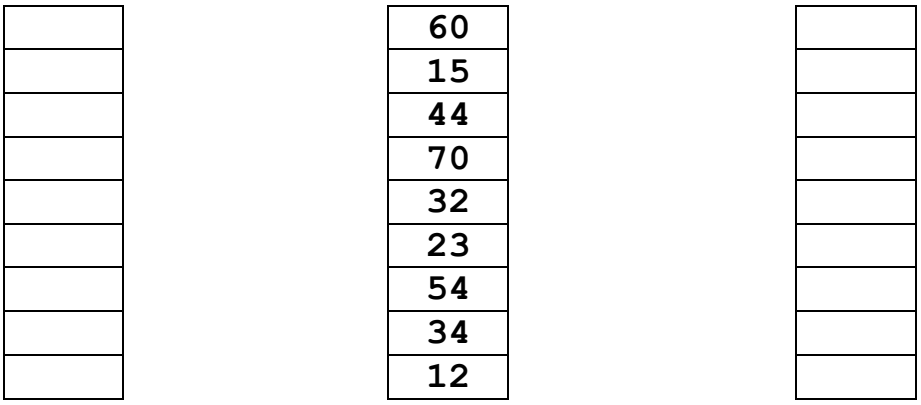
وحتى تستطيع عزيزي الدارس إتقان مهارات البرمجة نرجو منك تطبيق جميع الأمثلة التي وردت في هذه الوحدة على جهاز الحاسوب والتأكد من حلولها وحل جميع التدريبات.

لمحة مسبقة عن الوحدة التالية

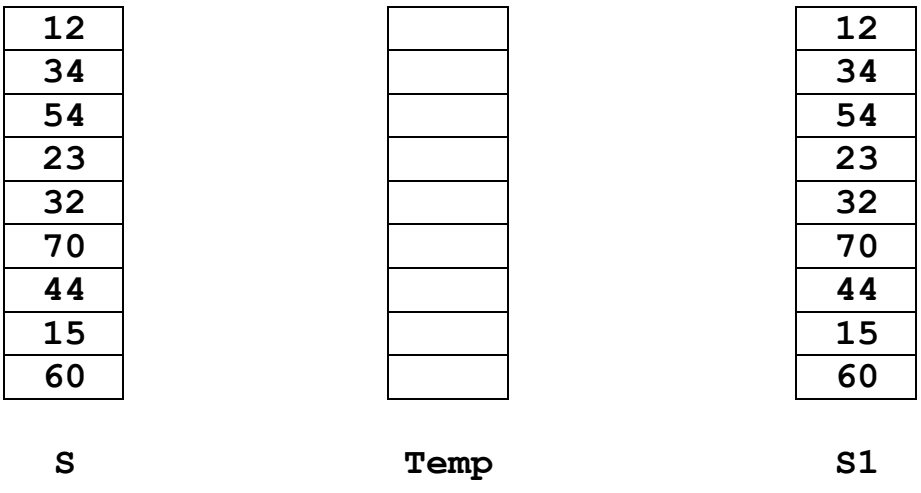
لاحظنا في هذه الوحدة أن عملية الصفوف ذات الأولوية تتطلب إعادة ترتيب الصف في كل مرة في حالة إذا ما كان العنصر الجديد أصغر من العنصر الذي يوجد في آخر الصف، فكان لابد من التفكير في طريقة حل أمثل وهو ما قاد إلي التفكير في عمل الصف باستخدام خصائص القوائم المتصلة مع المحافظة على خاصية أن الخروج للعنصر يتم من مقدمة القائمة. وهذا ما سوف نتناوله في الوحدة القادمة والتي هي بعنوان القوائم المتصلة.

إجابات التدريبات

لابد من وجود مكدسة مؤقتة **temp** نقل إليها عناصر المكدسة **S**



ونقوم بنقل عناصر المكدسة المؤقتة **temp** إلى المكدسة **S1** والي المكدسة .**S**
وتصبح عندئذ المكدسة **S1**منسوخة من المكدسة **S** أي لها نفس المحتويات

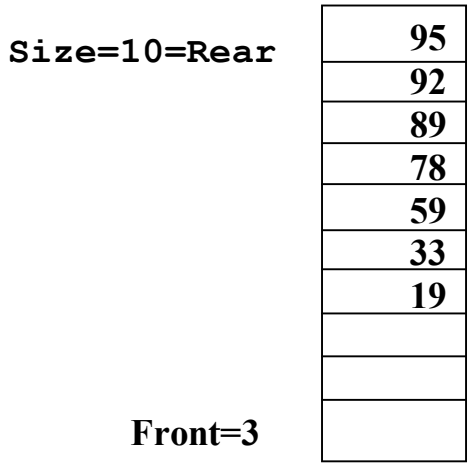


```
stack s,s1,temp;
int i
while (!s.empty_stack())
{
i=s.pop();
temp.push(i);
}
while (!temp.empty_stack())
{
i=temp.pop();
s1.push(i);
s.push(i);
}
```

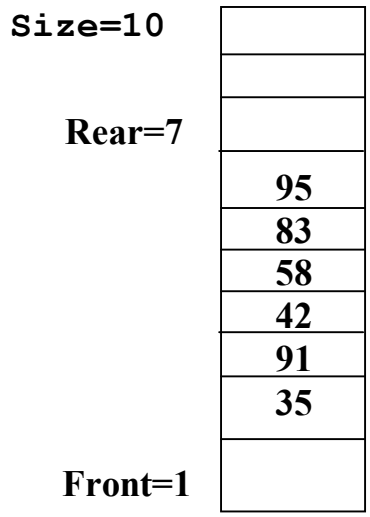
2. الغرض هو حساب عدد العناصر في المكدسة.

تدريب(2)

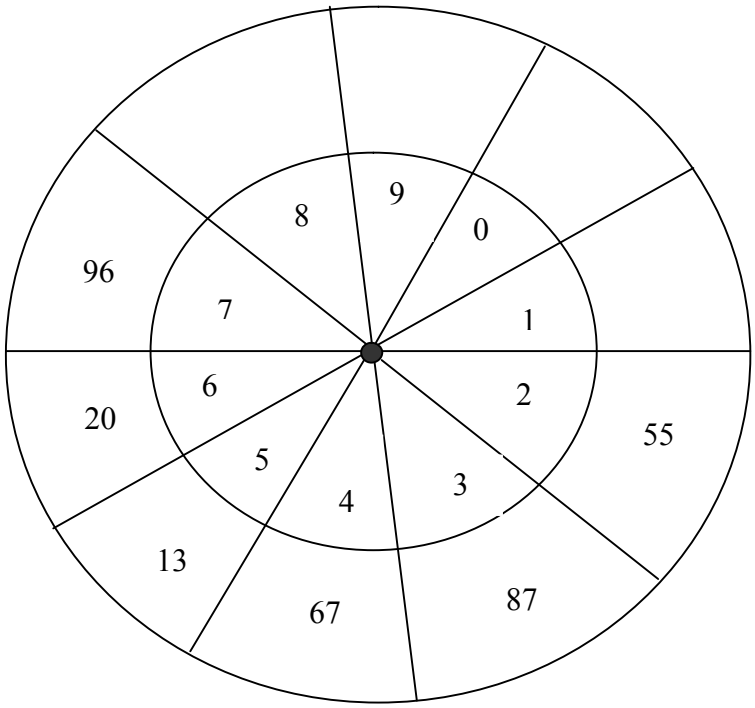
.1



.2



تدريب(3)



الذيل= 8 ، الرأس = 2 ، الحجم = 10

مسرد المصطلحات

- **المكدسة Stack:**

المكدسة هي قائمة من البيانات تحفظ بطريقة خطية، يتم فيها حذف وإضافة البيانات من طرف واحد يسمى القمة Top.

- **الصف Queue:**

الصف هو قائمة من العناصر تحفظ خطيةً، بحيث يمكن حذف العناصر من طرف واحد يسمى المقدمة أو الرأس Front، بينما يتم إضافة العناصر من الطرف الآخر ويسمى المؤخرة أو الذيل Rear.

- **الصف الخطي:**

صف محدود الحجم أي أن عدد العناصر محدود أو معروف مسبقاً وله سعة محددة وشرط امتلائه هو أن Rear = Sizeأي أن:

قيمة الذيل = عدد العناصر(الحجم).

- **الصف الدائري:**

هو أسلوب آخر يتجنب فكرة الإزاحة مطلقاً وتقوم فكرته علي أن يتصل آخر الصف بأوله.

المراجع

1. William Ford and William Topp, **Data Structure with C++**, Prentice-Hall Inc 1996.
2. Sara Baase ad Allen Van Gelder, **Computer algorithms**, Addison Wesley Longman 2000.
3. Deitel and Deital, **C++ How to program**, Prentice-Hall Inc 2001.

الوحدة الخامسة

القوائم المتصلة



محتويات الوحدة

الموضوع	الصفحة
المقدمة	**130**
تمهيد	130
أهداف الوحدة	131
1.مفهوم القوائم المتصلة	**132**
2.تعريف القوائم المتصلة	**133**
3.تمثيل البيانات	**134**
1.3 القائمة المتسلسلة	134
2.3 القوائم المتصلة	135
1.2.3 العقدة	135
2.2.3 حقل بيان	135
3.2.3 حقل ربط	135
4.2.3 المؤشر	135
4.أنواع القوائم المتصلة	**136**
1.4 القائمة أحادية الاتجاه	136
2.4 القائمة الدائرية	137
3.4 القائمة ذات الاتجاهين	138
4.4 القائمة ذات الرأس	138
5.4 القائمة المعممة	139
5.إنشاء فصيلة للقائمة المتصلة	**140**
6.تعريف الدوال الأعضاء	**141**
1.6 المشيد	141
2.6 الإضافة	142

الموضوع	الصفحة
1.2.6 إضافة عقدة جديدة في نهاية القائمة	143
2.2.6 إضافة عقدة جديدة في بداية القائمة	144
3.2.6 إضافة عقدة جديدة بين عقدتين في القائمة	145
3.6 البحث عن عقدة	145
4.6 الحذف	146
5.6 إيجاد طول القائمة	148
6.6 عرض عناصر القائمة	148
الخلاصة	**150**
لمحة مسبقة عن الوحدة التالية	**151**
إجابات التدريبات	**152**
مسرد المصطلحات	**153**
المراجع	**155**

المقدمـــة

تمهيـــد

عزيزي الدارس،

مرحبا بك إلى هذه الوحدة

مازلنا حتى الآن نتعامل مع الصفوف فمن خلال دراستك للوحدة الســابقة عرفــت أن عملية الصفوف ذات الأولوية تتطلب إعادة ترتيب الصف في كل مرة في حالــة مــا إذا كــان العنصر الجديد أصغر من العنصر الذي يوجد في آخر الصف.وعرفنا أن هذه العملية تتســبب في إطالة زمن التشغيل وإهدار مساحة مضافة من الذاكرة لنخزن عليها عناصــر المصــفوفة والصف وهذه الأسباب وغيرها قادت إلى التفكير في طريقة حل أمثل وهو ما يعــرف باســم وحدتنا هذه (القوائم المتصلة) وفيها سنعرف أشكال تمثيل البيانات، القائمة المتسلسلة والقــوائم المتصلة بأنواعها المختلفة.

وستنتقل بك الوحدة بعد ذلك إلى الدوال التي يمكن أن تتعامل مــع القــوائم المتصــلة كعمليات.

ثم تنتقل بك الوحدة إلى تعريف الدوال الأعضاء بدءاً بدالة المشيد ثم الإضافة وتتمثــل في ثلاثة اتجاهات وهي إضافة عقدة جديدة في نهاية القائمة وفي بداية القائمة وبين عقدتين في القائمة، ثم دالة البحث عن عقدة، ثم الحذف مثل حذف عقدة من نهاية القائمة ومن بداية القائمة وبين عقدتين في القائمة، ثم دالة إيجاد طول القائمة، وأخيراً دالة عرض عناصر القائمة.

نرحب بك مرة أخرى في هذه الوحدة ونرجو أن تجد فيها الفائدة جُلهــا، وأن تســتمتع بدراستها ونرحب بأرائك التي ستكون عوناً لنا في التطوير والتجويد والارتقاء.

أهداف الوحدة



عزيزي الدارس،بعد فراغك من هذه الوحدة ينبغي أن تكون قادرا على أن:

1. **تُعرف** القوائم المتصلة.
2. **تشرح** فكرة إنشاء القوائم المتصلة والأسباب التي أدت إلى قيامها.
3. **تُعدد** أشكال تمثيل البيانات.
4. **تُعرف** القائمة المتسلسلة.
5. **تُعرف** القوائم المتصلة.
6. **تُعدد** أنواع القوائم المتصلة.
7. **تُعدد** الدوال التي يمكن أن تتعامل مع القوائم المتصلة كعمليات.
8. **تُعدد** أنواع تعريف الدوال الأعضاء.
9. **تذكر** اتجاهات إضافة عقدة جديدة إلى القائمة.
10. **تشرح** الآتي: الإجراء الأول والإدخال والمعالجة والإخراج والإجراء النهائي عند عقدة جديدة في نهاية القائمة.
11. **توضح** كيف تتم عملية حذف عقدة من القائمة.

1.مفهوم القوائم المتصلة Linked Lists Concept

تخيل أن دخول كلية معينه يتم وفق من قَدِمَ أولاً (أي الأول في الصف) أي أن التقديم لا يتم وفق النسبة المئوية للطلاب.

ولكننا نجد في الواقع أن دخول الكلية يتم وفق النسبة المئوية للطلاب، وعليه يتم ترتيب الصف وفقاً للنسبة المئوية وهي مؤشر الأولوية في الصف.

وقد لاحظنا في الفصل السابق أن عملية الصفوف ذات الأولوية يتطلب إعادة ترتيب الصف في كل مرة في حالة إذا ما كان العنصر الجديد أصغر من العنصر الذي يوجد في آخر الصف، ويلاحظ الآتي في تلك العملية:

1. كثرة العمليات عليها وهو ما يتسبب في إطالة زمن التشغيل.
2. إهدار مساحة مضافة من الذاكرة لنخزن عليها عناصر المصفوفة والصف.
3. تحتاج المصفوفات للإعلان عن طولها أولاً وهذا يقود إلى عدم القابلية للتوسع وهو ما يتعارض مع الواقع الذي يتحول بكامله للزيادة أو النقصان أثناء عملية المعالجة.
4. يعمل في ظل أولوية واحدة فقط بينما إذا كانت الأولوية أكثر من واحدة فتعجز عن العمل.

هذه الملاحظات قادت إلى التفكير في طريقة حل أمثل وهو ما يعرف بالقوائم المتصلة مع المحافظة على خاصية أن خروج العنصر يتم من مقدمة القائمة (أي أسلوب **FIFO**) هذا مع إمكانية أن القائمة قابلة للتوسع والتقليص وفق عدد العناصر مما يعني استغلال مساحة الذاكرة حسب الحاجة مع قابلية إدراج عنصر وسط العناصر الموجودة بالقائمة إلى جانب إمكانية وضع أولويات غير محددة (أكثر من أولوية).

نشاط



اشرح مبدأ الأولوية في الصفوف وعلاقته مع القوائم المتصلة وتأثير كل منهم على الأخر.
ناقش ذلك مع زملائك وأعرض إجابتك على مشرفك الميداني.

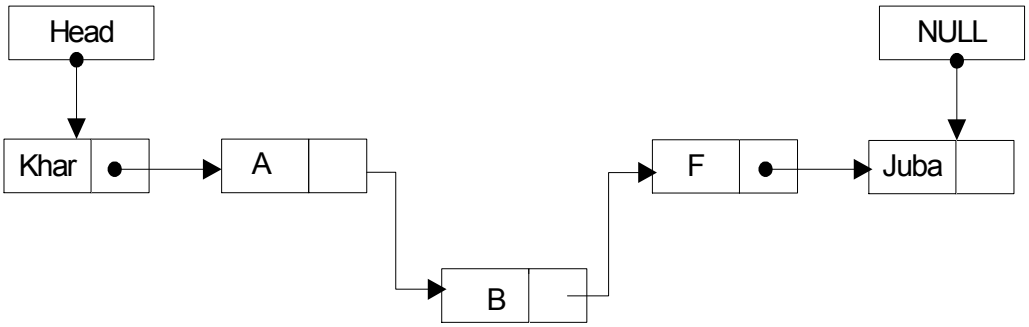
أسئلة تقويم ذاتي



1. ما هي الملاحظات التي يمكن أن تذكر في عملية إدخال عنصر جديد إلى الصف؟
2. اشرح فكرة القوائم المتصلة؟

2.تعريف القوائم المتصلة

القوائم المتصلة هي سلسلة من العقد بحيث أن كل عقدة مرتبطة بالعقدة التي تليها.
الشكل التالي يوضح قائمة متصلة مكونة من عدة عقد



الشكل (1) يوضح قائمة متصلة مكونة من عدة عقد

كل عقدة في الشكل السابق لها حقلان، الأول حقل يحتوي على بيانات، والثاني عبارة عن مؤشر (ممثل في الشكل أعلاه بسهم) إلى العقدة التالية هناك مؤشر يؤشر إلى العقدة الأولى رأس القائمة ويسمى الرأس Head والرمز الآخر Null دائماً يوجد في حقل المؤشر للعقدة الأخيرة في القائمة ويعني أن هذا المؤشر يشير إلى: اللا شيء:

تقويم ذاتي

؟	عرّف القوائم المتصلة مستخدماً الرسم.

3.تمثيل البيانات

يتم تمثيل البيانات في شكلين، شكل يعرف بالتمثيل المتسلسل للبيانات والآخر يعرف بالتمثيل المتصل للبيانات.

1.3 القائمة المتسلسلة

وهي قائمة من العناصر المتتابعة الواحدة تلو الأخرى بدون مسافة بين الواحدة و التي تليها، وخير مثال لذلك هو المصفوفات أو سلسلة الحروف كما في الرسم أدناه:

L:	X1	X2	X3		Xn

ونجد أن القائمة المتسلسلة تتميز بالخصائص التالية:

<1>. إمكانية إضافة أو حذف عنصر عشوائياً.

<2>. إمكانية تحديد حجم المصفوفة مسبقاً أو الفهرس المساوي لآخر عنصر في المصفوفة.

<3>. سرعة الوصول إلى أي عنصر (في عمليتي الاختيار أو الإحلال بالفهرس)

<4>. في عمليتي إضافة عنصر أو حذف عنصر تنشأ مشكلة إعادة ترتيب (فرز) المصفوفة، ونجد أن عملية الترتيب تحتاج إلى عدة عمليات.

<5>. الموقع يكون محدداً مسبقاً.

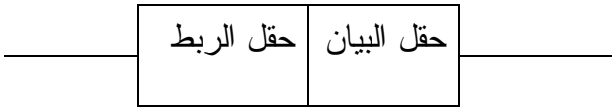
<6>. تنشأ مشكلة تحديد الحجم المناسب للمصفوفة (مثال في حالة أن العناصر التي يراد إدراجها أكبر من حجم المصفوفة) أو عندما نحجز من البداية حجماً أكبر للمصفوفة تنشأ مشكلة المساحة الخالية في الذاكرة.

2.3 القوائم المتصلة

القوائم المتصلة هي سلسلة من العقد بحيث كل عقدة ترتبط بعقدة لاحقة لهـــا، ماعـــدا العقدة الأخيرة التي ترتبط بالقيمة صفر **(Nil)** والتي تشير إلي نهاية القائمة. هنا نحتاج إلي توضيح بعض المصطلحات التي تستخدم في القوائم المتصلة وهي:

1.2.3العقدة Node

وهي عبارة عن سجل أو كائن في القائمة يمثل وحدات مختلفة من الـــذاكرة المعنونـــة ويحتوي على حقول/حقل للبيانات وحقل آخر يحمل عنواناً يؤشر للعقدة التالية فـــي القائمـــة، وليست بالضرورة أن تكون العقد متلاصقة فيزيائياً في الذاكرة، كما أن محتويات العقدة لـــيس بالضرورة أن تكون متجانسة، أي من نفس النوع كما هو الحال في المصفوفات والشكل الآتي يوضح ذلك:



شكل (2) يوضح عقدة تحتوي على حقلين

العقدة أعلاه تحتوي على حقلين:

2.2.3حقل بيان

هو الحقل الذي يحمل العنصر الحقيقي في القائمة وبيانات هذا العنصر قد تكـــون غير متجانسة مثل : رقم موظف ، اسم طالب أو سجلاً كاملاً.

3.2.3حقل ربط

يحمل عنوان العقدة التالية في القائمة ويعرف باسم المؤشر **(Pointer)**.

4.2.3 المؤشر Pointer

يحتوي المؤشر علي قيمة بيانية معينة تحدد موقع تخزين العنصر التالي في القائمة وغالبا ما تكون مكتوبة بالنظام الثنائي أو السادس عشري.

ويمكن أن نعرف العقدة **Node** من خلال السجل التالي:

```
struct node
{
int data;
node* next;
};
```

أسئلة تقويم ذاتي

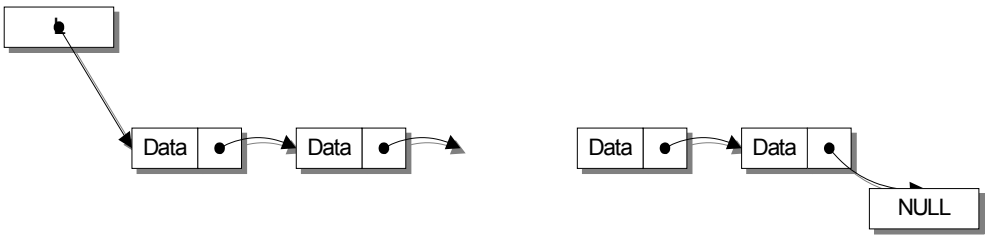


1. عدد أشكال تمثيل البيانات؟
2. عرف القائمة المتسلسلة؟
3. اذكر خواص القائمة المتسلسلة؟
4. عرف القوائم المتصلة؟
5. عرف العقدة وما هي حقولها؟

4.أنواع القوائم المتصلة

1.4 القائمة أحادية الاتجاه One Way linked List

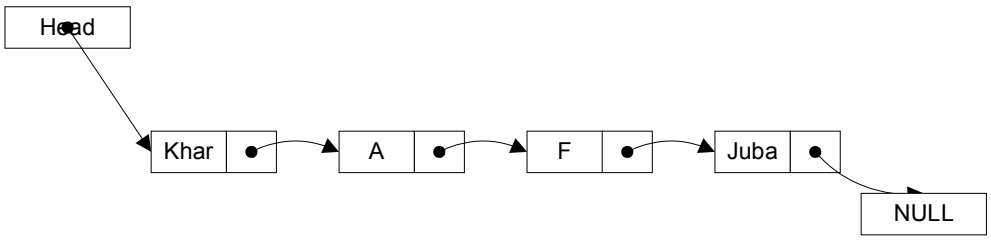
هي قائمة تشير كل عقدة فيها إلي العقدة التي تليها وبالتالي تكون العناصر مرتبطة مع بعضها البعض على شكل سلاسل ويوجد مؤشر لأول عقدة في القائمة والعقدة الأخيرة تؤشـــر إلي (لا شيء).



شكل (3) قائمة اتحادية الاتجاه

مثال:إذا كان لدى شركة الخطوط الجوية السودانية خط رحلات يبدأ من مدينة **Khar** وينتهي بمدينة **Juba** لاتجاه واحد بدون عودة مروراً بالمدينة **A** والمدينة **F**.

ويمكن تمثيل ذلك في شكل قائمة متصلة أحادية الاتجاه كما هو مبين أدناه:



شكل (4) قائمة متصلة اتحادية الاتجاه

الوصول إلي عنصر يتم التحرك له من القيمة الابتدائية Head وجعلها تؤشر إلى العقدة الأولى في القائمة ثم الانتقال إلي العقدة التالية التي يؤشر إليها المؤشر وهكذا إلى أن نصل إلي العقدة المحددة، مثلاً إذا أردنا الوصول إلى المدينة **F** يجب أن نتحرك من مدينة **Khar** مروراً بالمدينة **A** حتى نصل إلى المدينة **F**.

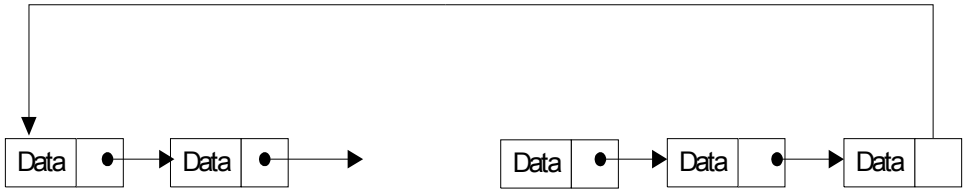
تدريب (1)



إذا كان لدى شركة الخطوط الجوية السودانية خط رحلات يبدأ من مدينة Dungla وينتهي بمدينة Khar لاتجاه واحد بدون عودة مروراً بالمدينة C والمدينة D، مثل ذلك في شكل قائمة متصلة أُحادية الاتجاه.

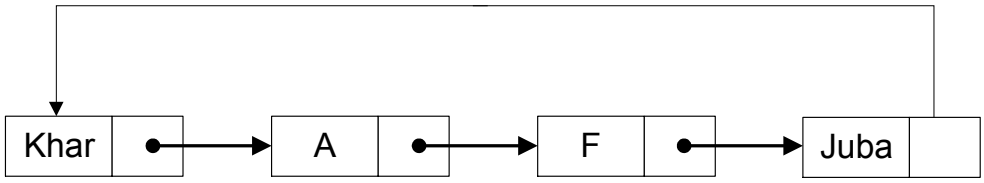
2.4 القائمة الدائرية Circular Linked List

هي قائمة بسيطة أو أحادية الاتجاه فيها العقدة الأخيرة تؤشر إلى العقدة الأولى بدلاً عن أن تؤشر إلى لا شيء.



شكل (5) قائمة دائرية

ففي المثال السابق: تكون العودة مباشرةً إلي مدينة **Khar** بدلاً من العودة بالمسار المعكوس كما في الرسم أدناه:

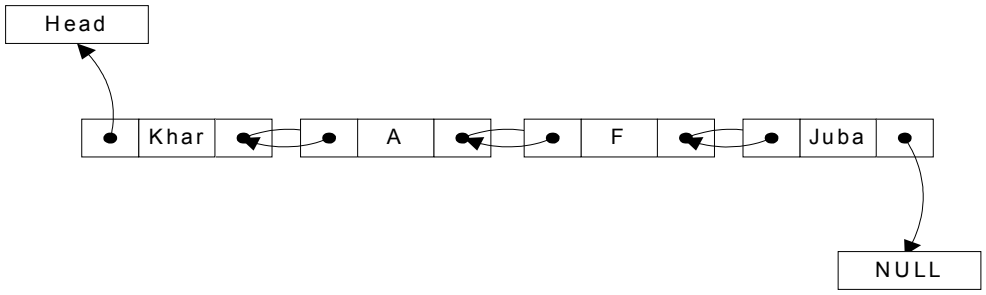


شكل (6) قائمة دائرية

نستطيع المواصلة في البحث بعد أن تكون إحدى العقد قد تم تخطيها بدون الحاجة إلي عملية جديدة وذلك بالعودة إليها من العقدة الأولى بعكس ما في القائمة البسيطة العادية أحادية الاتجاه.

3.4 القائمة ذات الاتجاهين Two Way Linked List

هي قائمة ذات مؤشرين يؤشر إحداهما إلي العقدة السابقة في حين يؤشر الثاني إلى العقدة التالية في القائمة، وأحد مؤشري العقدة الأخيرة يؤشر إلي لاشيء والشكل أدناه يوضح ذلك:



شكل (7) قائمة ذات اتجاهين

ففي المثال السابق يمكن للرحلة أن ترجع إلي نقطة سابقة أو أن تواصل مسيرها.

4.4 القائمة ذات الرأس Linked List with Header

بما أن القائمة المتصلة الدائرية لا تحتوي على قيمة **Null** في العقدة الأخيرة لها فإن أحد الحلول المطروحة لمشكلة البحث عن عقدة محددة وبمعلومات معروفة في القائمة المتصلة الدائرية هو وضع قيمة في العقدة الأولى لتمييزها ومعرفتها تسمى عقدة الرأس **Header**

Node وتحتوي هذه العقدة على معلومات عن القائمة مثل عدد العناصر التـــي بهـــا ونقطـــة التوقف في القائمة الدائرية... الخ. وعادة تستعمل مع القوائم الدائرية والقوائم ثنائيـــة الاتجـــاه، ويراعى فيها الآتي:

1. عند الوصول إلي أي عقدة في القائمة يجب تحديث الرأس.
2. يجب تحديث الرأس بعد كل عملية حذف أو إضافة لإحدى العقد.

5.4 القائمة المعممة Generalized Linked List

هي قائمة فيها كل عنصر مستقل له الفرصة ليكوّن قائمة جزئية **Sublist** إذا كان أحد عناصر القائمة لا يحتوي على قائمة جزئية فإنه يسمى عنصر ذري **Atomic**.

أسئلة تقويم ذاتي



1. ما هي أنواع القوائم المتصلة؟
2. 2عرف كل من:

(أ)القائمة الاتحادية؟ (ب)القائمة الدائرية؟

(جـــ)القائمة ذات الاتجاهين؟ (د)القائمة ذات الرأس؟

(هـــ)القائمة المعممة؟

5. إنشاء فصيلة للقائمة المتصلة

هنالك عدد من الدوال يمكن أن تتعامل مع القوائم المتصلة كعمليات، نذكر مـــن هـــذه الدوال الأتي:

<1>. المشيد : **Constructor**

<2>. الإضافة : **Insertion**

- ✓ إدراج عقدة جديدة في نهاية القائمة.
- ✓ إدراج عقدة جديدة في بداية القائمة.
- ✓ إدراج عقدة جديدة بين عقدتين.

<3>. البحث.

<4>. الحذف :

- ✓ حذف العقدة الأولى.
- ✓ حذف العقدة الأخيرة.
- ✓ حذف عقدة اختيارية.

<5>. إيجاد الطول.

<6>. عرض عناصر القائمة.

<7>. نسخ قائمة.

وبالتالي يمكن وضع جميع الدوال في فصيلة واحدة **class** ويتسق ذلك مع مفهـــوم البرمجة الكائنية تسمي **Linked_List**

```
class link_list
{
  private:
    node* first;
  public:
    link_list();
```

```
    void insert_first(int d);
    void insert_last(int d);
    void displaylist() ;
    bool isempty();
    bool remove(int key);
    int length()  ;
};
```

ويمكن تعديل سجل العقدة ليصبح فصيلة كالأتي

```
class node
{
  public:
    int data;
    node* next;
    node(int d1):data(d1),next(NULL)
    {}
    void display()
    {
       cout<<data<<" ---> ";
    }
};
```

6.تعريف الدوال الأعضاء

1.6 المشيد Constructor

الإجراء الأول	لا يوجد
الإدخال	لا يوجد
المعالجة	أجعل **first** يؤشر إلى **Nil**
الإخراج	لا يوجد
الإجراء النهائي	لا يوجد

ففي مثال شركة الخطوط الجوية السودانية فإننا ننشئ خط الطيران الأساسي ونجعل أول تحرك من الخرطوم ثم نتركه فارغاً.

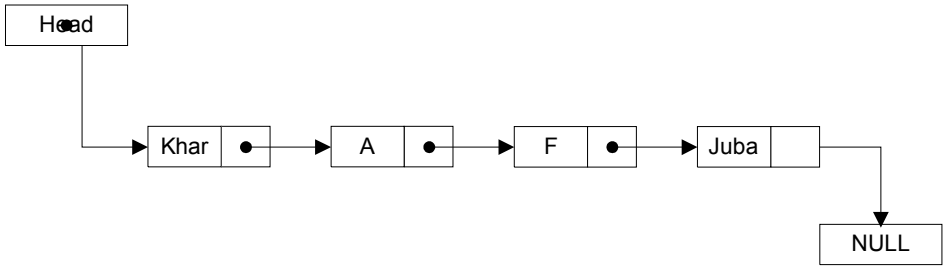
الدالة link_list() التالية هي دالة بناء تنفذ بمجرد الإعلان عن هدف يتبع إلى الفصيلة وتقوم بوضع قيمة أولية لمؤشر**first**هي NULL ويمكن تعريف الدالة (خارج الفصيلة) علي النحو التالي

```
link_list::link_list():first(NULL)
{
   cout<<"|-->";
}
```

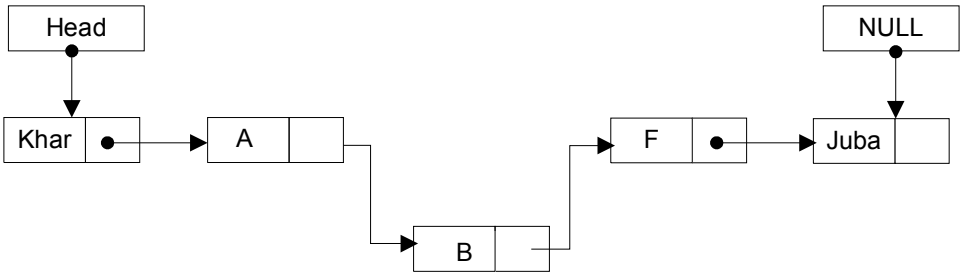
2.6 الإضافة Insertion

وفيه يتم إضافة إحدى العقد إلي سلسلة قائمة فارغة أو بها عقد، في أول القائمة أو بين عقدتين أو في نهاية القائمة، وبشكل عام يتم البحث عن العقدة المراد الإضافة بعدها وتلك المراد الإضافة قبلها ثم يتم تغيير المؤشرات بجعل السابقة تؤشر إلي الجديدة والجديدة تؤشر إلي اللاحقة.

ففي مثال شركة الخطوط الجوية السودانية وبافتراض أن الشركة قررت إدخال محطة جديدة بين محطة المدينة **A** ومحطة المدينة **F** ولتكن محطة المدينة **B** في هذا الخط، ففي هذه الحالة سوف نبحث أولا عن محطة المدينة **A** ثم نجعل محطة المدينة **A** تؤشر إلى محطة المدينة **B** وهي المحطة الجديدة، وأن محطة المدينة **B** تؤشر إلي محطة المدينة **F** ونكون بذلك قد أضفنا عقدة (محطة) جديدة.



الشكل (8) يوضح حالة الخط قبل إضافة المحطة الجديدة

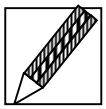


الشكل (9) يوضح حالة الخط عند إضافة محطة المدينة B (وهي المحطة الجديدة)

وملخص القول يمكن أن نأخذ مسالة إضافة عقدة جديدة إلى القائمة من ثلاثة اتجاهات وهي:

- ✓ إضافة عقدة جديدة في نهاية القائمة.
- ✓ إضافة عقدة جديدة في بداية القائمة.
- ✓ إضافة عقدة جديدة بين عقدتين في القائمة.

تدريب (2)



قررت شركة الخطوط الجوية السودانية إضافة محطة مدينة B بين مدينة Khar ومدينة Juba أضف المحطة مستخدماً الرسم.

1.2.6 إضافة عقدة جديدة في نهاية القائمة

الإجراء الأول	القائمة يجب أن تكون موجودة
الإدخال	عقدة جديدة تشير إلى **Nil**
	إذا كان **first** يؤشر إلى **Nil.**
	أجعل **first** يؤشر إلي العقدة الجديدة.
المعالجة	و إلا
	ابحث عن آخر عقدة.
	اجعل مؤشر آخر عقدة تؤشر إلي العقدة الجديدة.
الإخراج	لا يوجد
الإجراء النهائي	تم إضافة عقدة جديدة وأصبح شكل القائمة متغيراً.

```
void link_list::insert_last(int d)
{
  cout<<"Insert new Node with key "
      <<d<<" at rear"<<endl;
  node* current=first;
  node* newnode=new node(d);
  if(isempty())
     first=newnode;
  else
  {
     while(current->next!=NULL)
        current=current->next;
     current->next=newnode;
  }
}
```

2.2.6 إضافة عقدة جديدة في بداية القائمة

الإجراء الأول	القائمة يجب أن تكون موجودة
الإدخال	عقدة جديدة تشير إلى **Nil**
	إذا كان **first** يؤشر إلى **Nil**
	اجعل **first** يؤشر إلي العقدة الجديدة .
المعالجة	وإلا
	اجعل **first** يؤشر إلي العقدة الجديدة .
	اجعل مؤشر العقدة الجديدة يؤشر إلى العقدة التالية.
الإخراج	لا يوجد
الإجراء النهائي	تم إضافة عقدة جديدة وأصبح شكل القائمة متغيراً.

```
void link_list::insert_first(int d)
{
  cout<<"Insert new Node with key "
      <<d<<" at front"<<endl;
  node* newnode=new node(d);
  if(isempty())
```

```
      first=newnode;
  else
  {
      newnode->next=first;
      first=newnode;
  }
}
```

3.2.6 إضافة عقدة جديدة بين عقدتين في القائمة

الإجراء الأول	القائمة يجب أن تكون موجودة وتحتوي علي أكثر من عقدة
الإدخال	عقدة جديدة تشير إلى **Nil**
	إذا كان **first** يؤشر إلى **Nil**
	أطبع **Error** .
المعالجة	وإلا
	ابحث عن العقدة السابقة
	اجعل **first** يؤشر إلي العقدة الجديدة .
	اجعل مؤشر العقدة الجديدة يؤشر إلى العقدة اللاحقة.
الإخراج	لا يوجد
الإجراء النهائي	تم إضافة عقدة جديدة وأصبح شكل القائمة متغيراً.

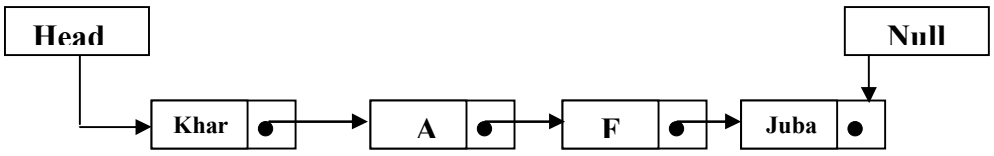
3.6 البحث عن عقدة Search of the Node

ففي مثال شركة الخطوط الجوية السودانية وبافتراض أن الشركة تريد البحث عن سجلات محطة المدينة **F**. سوف نستخدم محطة مؤقتة ولتكن محطة المدينة **R** ونودع بها السجلات التي نود البحث عنها ثم نجعل مؤشر الرأس يؤشر إليها ثم نقارن قيمة سجلاتها بسجلات محطة مدينة **Khar** فإذا لم تتطابق معها نجعل المؤشر يؤشر إلى المحطة التالية لها وهي محطة المدينة **A** ثم نستمر في التحريك والمقارنة حتى نهاية الخط فإذا تطابقت بيانات محطة ما مع المحطة المؤقتة فإن دالة البحث ترجع قيمة الملفات بتلك المحطة وإلا ترجع رسالة بعدم وجود تلك المحطة.

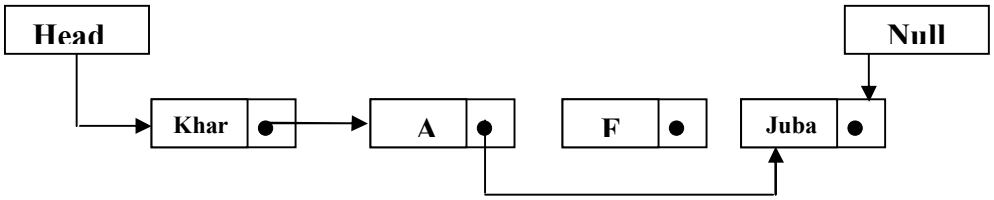
4.6 الحذف Deletion

وفيه يتم حذف إحدى العقد من سلسلة قائمة بها عقد من أول القائمة أو من بين عقدتين أو من نهاية القائمة، وبشكل عام يتم البحث عن العقدة المراد الحذف بعدها وتلك المراد الحذف قبلها ثم يتم تغيير المؤشرات بجعل السابقة تؤشر إلى اللاحقة و تحرير المساحة التخزينية للعقدة المحذوفة.

ففي مثال شركة الخطوط الجوية السودانية وبافتراض أن الشركة قررت إلغاء أو حذف خدمة محطة ولتكن هي محطة المدينة **F** في هذا الخط، ففي هذه الحالة سوف نبحث أولا عن محطة المدينة **F** ثم نجعل مؤشر محطة المدينة السابقة وهي المدينة **A** يؤشر إلى محطة المدينة اللاحقة وهي المدينة **Juba** ونكون بذلك قد حذفنا محطة المدينة**F**.

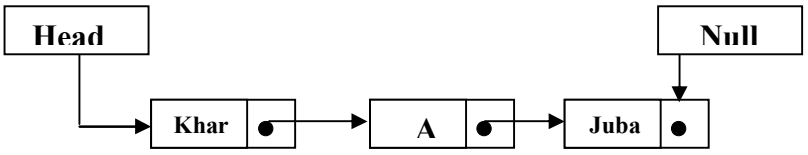


شكل (**10**)



شكل (**11**)

الشكل (10 – 11) يوضح حالة الخط قبل حذف محطة المدينة F



الشكل (12) يوضح حالة الخط بعد حذف محطة المدينة F.

وملخص القول يمكن أن نأخذ مسالة حذف عقدة من القائمة بثلاثة اتجاهات وهي:

- ✓ حذف عقدة من نهاية القائمة.
- ✓ حذف عقدة من بداية القائمة.
- ✓ حذف عقدة من بين عقدتين في القائمة.

تتم عملية الحذف وفقاً للخوارزمية التالية:

- **البحث عن العقدة السابقة للعقدة المراد حذفها.**
- **البحث عن العقدة المراد حذفها.**
- **الاحتفاظ بالعقدة المراد حذفها في مؤشر مؤقت.**
- **جعل مؤشر العقدة السابقة يؤشر للعقدة اللاحقة.**
- **تحرير العقدة المراد حذفها.**

```
bool link_list::remove(int key)
{
   node* current=first;
   node* previous=first;
   while(current->data!=key)
   {
      if(current->next==NULL)
         return false;
      else
      {
         previous=current;
         current=current->next;
      }
   }
```

```
    if(current==first)
       first=first->next;
    else
       previous->next=current->next;
    delete current;
    return true;
}
```

5.6 إيجاد طول القائمة Find Length

وهي دالة تقوم بعمل عداد يتزايد مع حركة المؤشر وانتقاله من أول عقدة إلـــى آخـــر عقدة في القائمة وترجع قيمة العداد والذي يمثل طول القائمة.

```
int link_list::length()
{
  node* current=first;
  int count=0;
  while(current!=NULL)
  {
     count++;
     current=current->next;
  }
  return count;
}
```

6.6 عرض عناصر القائمة Display List

وهي دالة تقوم بعرض جميع العناصر الموجودة بالقائمة من أول عنصر إلي آخر عنصر.

```
void link_list::displaylist()
{
    node* current=first;
    while (current!=NULL)
    {
       current->display();
       current=current->next;
    }
    cout<<"Nil"<<endl;
}
```

أسئلة تقويم ذاتي



1. عدد اتجاهات إضافة عقدة جديدة إلى القائمة؟
2. وضح الإجراء الأول والإدخال والمعالجة والإخراج والإجراء النهائي عند إضافة عقدة جديدة في نهاية القائمة؟
3. ما هي اتجاهات حذف العقدة من القائمة؟
4. كيف تتم عملية حذف العقدة من القائمة؟
5. اذكر مهمة دالة إيجاد طول القائمة؟
6. ما هو الدور الذي تقوم به دالة عرض عناصر القائمة؟

الخلاصة

هناك عدة ملاحظات مهمة قادت إلى التفكير في حل أمثل لمشكلة أولوية الصفوف، وذلك عند إدخال عنصر جديد في الصف وتمثل الحل الأمثل في ما يعرف باسم القوائم المتصلة.

وقد عرفنا في هذه الوحدة أشكَال تمثيل البيانات مثل القائمة المتسلسلة والقوائم المتصلة، وعرفنا فيها مصطلحات مهمة العقدة والمؤشر، وتطرقنا إلى أنواع القوائم المتصلة مثل القائمة أحادية الاتجاه والقائمة الدائرية والقائمة ذات الاتجاهين وأخيراً القائمة المعممة، ثم عرفنا الدوال التي يمكن أن تتعامل مع القوائم المتصلة كعمليات.

ثم تتطرقت الوحدة إلى تعريف الدوال الأعضاء:

- ✯المشيد وذلك من حيث الإجراء الأول والإدخال والمعالجة والإخراج والإجراء النهائي.
- ✯الإضافة وهي تتمثل في ثلاثة اتجاهات وهي: إضافة عقدة جديدة في نهاية القائمة، وفي بداية القائمة، وإضافة عقدة جديدة بين عقدتين في القائمة.
- ✯ثم تناولت الوحدة دالة البحث عن عقدة.
- ✯ودالة الحذف وهي أيضاً مأخوذة من ثلاث اتجاهات:

حذف عقدة من نهاية القائمة، ومن بداية القائمة، أخيراً حذف عقدة بين عقدتين في القائمة. ثم عرفنا أن عملية الحذف تتم وفق خوارزمية محددة

- ✯دالة إيجاد طول القائمة.
- ✯دالة عرض عناصر القائمة وهي تقوم بعرض جميع العناصر بالقائمة من أول عنصر إلى آخر عنصر.

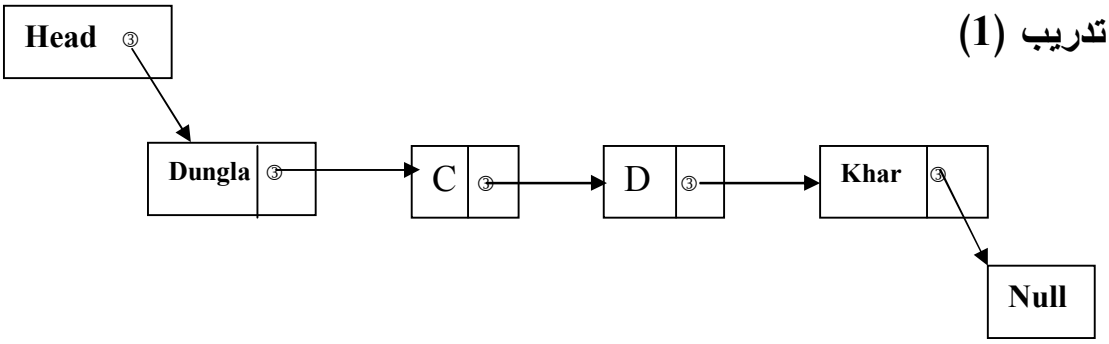
نأمل أن يفيدك هذا الملخص في استعادة المفاهيم الرئيسة التي جاءت بها الوحدة ونرجو أن يكون شاملاً غطى كل الأهداف.

لمحة مسبقة عن الوحدة التالية

في الوحدة القادمة سوف نتعرض إلى أحد أنواع بنيات البيانات، والتي لها درجة عالية من الأهمية، ألا وهي الأشجار Trees والتي تنتمي إلى مجموعة أكثر شمولا تسمى المخطـــط GRAPH، ولا تقف هياكل الأشجار في أنها مفيدة فقط بل يمكن القول بأنها أفضل من جميـــع هياكل البيانات الأخرى من حيث المزايا التي يُستفاد من استخدامها فنجد سرعة تنفيذ العمليـــات الخاصة بالبحث والحذف و الإضافة عاليةً جداً مقارنة مع هياكل البيانات الأخرى.

إجابات التدريبات

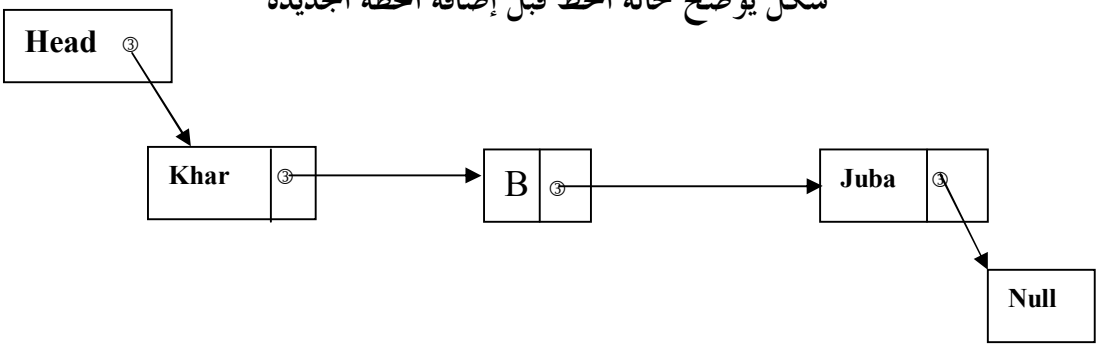
تدريب (1)



تدريب (2)



شكل يوضح حالة الخط قبل إضافة المحطة الجديدة



شكل يوضح حالة الخط بعد إضافة محطة المدينة **B** (المحطة الجديدة)

مسرد المصطلحات

- **القوائم المتصلةLinked Lists:**

هي سلسلة من العقد بحيث كل عقدة ترتبط بعقدة لاحقة لها ما عدا العقدة الأخيرة التي ترتبط بالقيمة صفر (Nil) والتي تشير إلى نهاية القائمة.

- **العقدة Node:**

هي عبارة عن سجل أو كائن في القائمة يمثل وحدات مختلفة من الذاكرة المعنونة ويحتوي على حقول / حقل للبيانات وحقل آخر يحمل عنواناً يؤشر للعقدة التالية في القائمة، وليست بالضرورة أن تكون العقد متلاصقة فيزيائياً في الذاكرة، كما أن محتويات العقدة ليس بالضرورة أن تكون متجانسة أي من نفس النوع كما هو الحال في المصفوفات.

- **حقل بيان:**

هو الحقل الذي يحمل العنصر الحقيقي في القائمة وبيانات هذا العنصر قد تكون غير متجانسة.

- **حقل ربط:**

يحمل عنوان العقدة التالية في القائمة ويعرف باسم المؤشر (Pointer).

- **المؤشر Pointer:**

يحتوي المؤشر على قيمة بيانية معينة تحدد موقع تخزين العنصر التالي في القائمة وغالباً ما تكون مكتوبة بالنظام الثنائي أو السادس عشري.

- **القائمة أحادية الاتجاه One Way linked List:**

هي قائمة تشير كل عقدة فيها إلي العقدة التي تليها وبالتالي تكون العناصر مرتبطة مع بعضها البعض على شكل سلاسل ويوجد مؤشر لأول عقدة في القائمة والعقدة الأخيرة تؤشر إلي لاشيء.

- **القائمة الدائرية** Circular Linked List:

هي قائمة بسيطة أو أحادية الاتجاه فيها العقدة الأخيرة تؤشر إلى العقدة الأولى بدلاً عن أن تؤشر إلى لا شيء.

- **القائمة ذات الاتجاهين** Two Way Linked List:

هي قائمة ذات مؤشرين يؤشر إحداهما إلي العقدة السابقة في حين يؤشر الثاني إلي العقدة التالية في القائمة، وأحد مؤشري العقدة الأخيرة يؤشر إلي لاشيء.

- **القائمة ذات الرأس** Linked List with Header:

بما أن القائمة المتصلة الدائرية لا تحتوي على قيمة **Null** في العقدة الأخيرة لها فأن أحد الحلول المطروحة لمشكلة البحث عن عقدة محدد وبمعلومات معروفة في القائمة المتصلة الدائرية هو وضع قيمة في العقدة الأولى لتمييزها ومعرفتها تسمى عقدة الرأس، **Header Node** وتحتوي هذه العقدة على معلومات عن القائمة مثل عدد العناصر التي بها، ونقطة التوقف في القائمة الدائرية... الخ. وعادة تستعمل مع القوائم الدائرية والقوائم ثنائية الاتجاه، ويراعى فيها الآتي:

1. عند الوصول إلي أي عقدة في القائمة يجب تحديث الرأس.
2. يجب تحديث الرأس بعد كل عملية حذف أو إضافة لإحدى العقد

- **القائمة المعممة** Generalized Linked List:

هي قائمة، لكلٍّ عنصر مستقل فيها الفرصة الفرصة ليكوّن قائمة جزئية **Sublist** وإذا كان أحد عناصر القائمة لا يحتوي على قائمة جزئية فإنه يسمى عنصراً ذرياً **Atomic**.

المراجع

1. William Ford and William Topp, **Data Structure with C++**, Prentice-Hall Inc 1996.
2. Sara Baase ad Allen Van Gelder, **Computer algorithms**, Addison Wesley Longman 2000.
3. Deitel and Deital, **C++ How to program**, Prentice-Hall Inc 2001.


الوحدة السادسة
الأشجار Trees

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**160**
تمهيد	160
أهداف الوحدة	160
1. الأشجار	**161**
2. تعريفات	**162**
3. الأشجار الثنائية	**165**
1.3 الاتزان في الأشجار الثنائية	167
2.3 التنقل عبر الأشجار الثنائية	172
1.2.3 أسلوب التنقل الوسطي	173
2.2.3 أسلوب التنقل البعدي	177
3.2.3 أسلوب التنقل القبلي	178
4. شجرة البحث الثنائي	**182**
1.4 البحث عن عقدة داخل الشجرة	184
2.4 البحث عن قيمة الحد الأدنى والأعلى في الشجرة	185
3.4 إضافة عقدة إلى الشجرة	186
4.4 حذف عقدة محددة بالشجرة	188
5. طرق ترتيب عناصر الشجرة الثنائية	**189**
6. تطبيقات الأشجار	**191**
7. تكرار مداخل القيم وتطابقها داخل شجرة واحدة	**199**
8. Read Black Tree	**199**
1.8 خصائص الـ Read Black Tree	199
2.8 تعريفات خاصة بقواعد الألوان	200

الموضوع	الصفحة
3.8 التدوير	201
4.8 أسلوب الإضافة في الـــ Read Black Tree	204
9. تركيب وتطبيقات 2-3-4 trees	**207**
1.9 دلالة الاسم	208
2.9 أسلوب تنظيم البيانات	208
10. كفاءة 2-3-4 T	**209**
الخلاصة	**210**
لمحة مسبقة عن الوحدة التالية	**210**
إجابات التدريبات	**211**
مسرد المصطلحات	**213**
المراجع	**215**

المقدمة

تمهيد

عزيزي الدارس،

مرحبا بك إلى هذه الوحدة ، الوحدة السادسة

تعتبر هذه الوحدة من الوحدات المهمة لأنها تناقش مواضيع مهمة في هياكل البيانات وهي: تعريف الأشجار، التمييز بين الأشجار المنتظمة وغير المنتظمة، التمييز بين الأشجار الثنائية والثلاثية، أسلوب تنظيم البيانات، وتطبيقات الأشجار

وسوف نتطرق لترتيب العناصر في الأشجار حيث ستجد أنّ الترتيب يكون إما أفقياً أو عمودياً.

وستتعرّف على أساليب التنقّل في الأشجار وهي ثلاثة أنواع: تنقل وسطي، تنقل بعدي، تنقل قبلي.

أتمنى لك عزيزي الدارس كل التوفيق ،،،

أهداف الوحدة

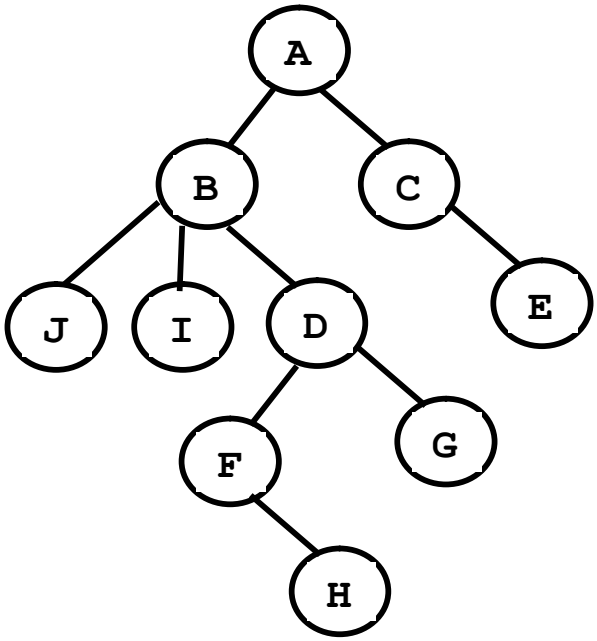


عزيزي الدارس،بعد فراغك من هذه الوحدة ينبغي أن تكون قادرا على أن:

1. **تعرّف** الأشجار ومكوناتها.
2. **تحسب** معامل الاتزان في الأشجار الثنائية.
3. **تفرّق** بين الشجرة الثنائية التامة والشجرة الثنائية الكاملة.
4. **تنتقل** عبر الأشجار الثنائية.
5. **ترتّب** عناصر الشجرة الثنائية.
6. **تستخدم** دوال البحث الثنائي.
7. **تبحث** عن عقد داخل الشجرة.
8. **تضيف** عقدة إلى الشجرة.

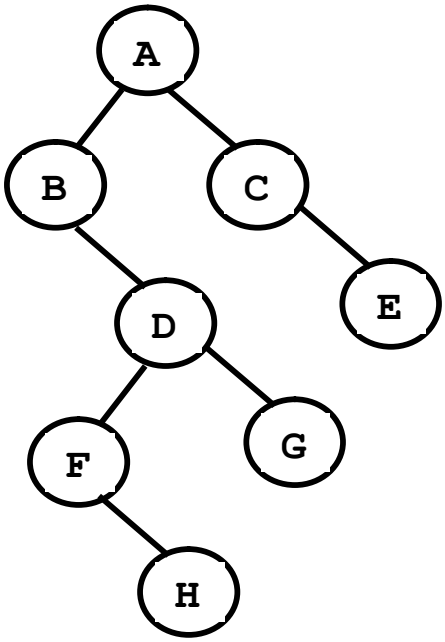
1. الأشجار Trees

تعتبر الأشجار واحدة من نماذج هياكل البيانات التي تنتمي لمجموعة أكثر شمولا تسمى المخطط **GRAPH** وتتكون الشجرة من مجموعة عقد **Nodes** تربط بينها ممرات **Edges** وهنالك عقدة وحيدة في المستوى الأول للشجرة نطلق عليها العقدة الرئيسية أو الجزر **Root** بحيث تكون الممرات منها للعقد التي تليها فقط وهذه الممرات لا تمثل دورة، والشكل رقم (1) يبين نموذجاً للشجرة



الشكل رقم (1) نموذج للشجرة

أما إذا احتوى الهيكل على دورة فإنه يصبح مخططاً وليس شجره والشكل رقم (2) يبين أن الهيكل احتوي علي الدورة **(A B D E C)** لهذا فإن الهيكل لا يمثل شجرة وإنما يسمى مخططاً أو بياناً.



الشكل رقم (2) مخطط احتوى على دورة

لا تقف هياكل الأشجار في أنها مفيدة فقط بل يمكن القول بأنها تعتبر أفضل من جميع هياكل البيانات الأخرى من حيث المزايا التي تعود من استخدامها فنجد سرعة تنفيذ العمليات الخاصة بالبحث والحذف و الإضافة عالية جداً مقارنة مع هياكل البيانات الأخرى.

نشاط



لا تقف هياكل الأشجار في أنها مفيدة فقط بل يمكن القول بأنها تعتبر أفضل من جميع هياكل البيانات الأخرى.

ناقش ذلك مع زملائك وأعرض إجابتك على مشرفك الميداني.

2. تعريفات

لتحديد العلاقة بين العناصر المختلفة التي تكون الهيكل الشجري نلجأ إلى استخدام مصطلحات محددة لابد لنا من تعريفها.

- **الأبناء** Children

هي العقد التي تنحدر مباشرة من العقدة العلوية وترتبط بها بواسطة مؤشرات الربط يطلق علي العقدة العلوية اسم الأب **Parent** والعقد إلي تنحدر منها أبناء **Children**وفي الشكل رقم (1) تسمي العقدة **B**والعقدة **C أبناء الجذر A** وتسمي العقدة **F** والعقدة **G أبناء للعقدة D**التي تسمي الأب **Parent** وإذا كان عدد الأبناء اثنين يسمى الابن من جهة اليمين بالابن الأيمن **Right Child** والابن من جهة اليسار بالابن الأيسر **Left Child وفي** الشكل رقم (5-1) تسمي العقدة **G** الابن الأيمن**Right Child** والعقدة **F** بالابن الأيسر**Left Child للعقدةD**.

- **الممر** Edge

هو عبارة عن مؤشر الربط الذي يربط عقدتين مثل مؤشر الربط الذي يربط بين الجذر والابن.

- **الورقة** Leaf

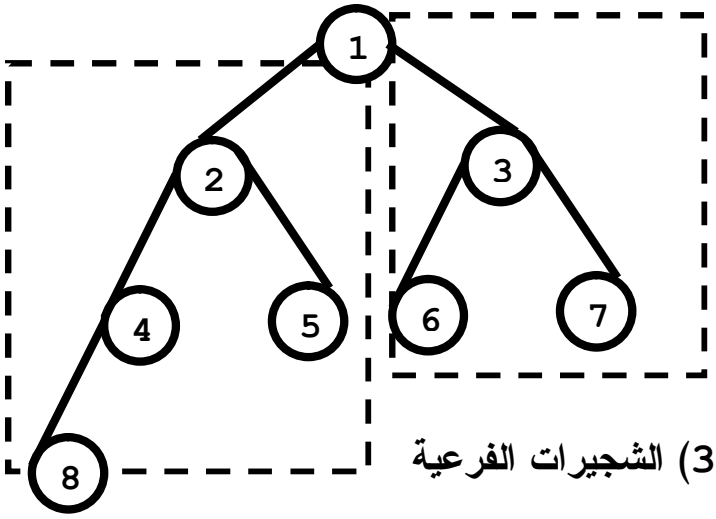
هي عبارة عن عقدة لا تحتوي علي أبناء وفي الشكل رقم (1) تسمى العقد **J** و **I** و **H** و **G** و **E** أوراق.

- **المسار** Path

هو عبارة عن مجموعة متتالية من الممرات وإذا وصل المسار إلي ورقة يسمى المسار غصناً **Branch** وفي الشكل رقم (1) يسمي **المسار A->B->d->F->H** غصن.

- **الشجيرة الفرعية** Sub Tree

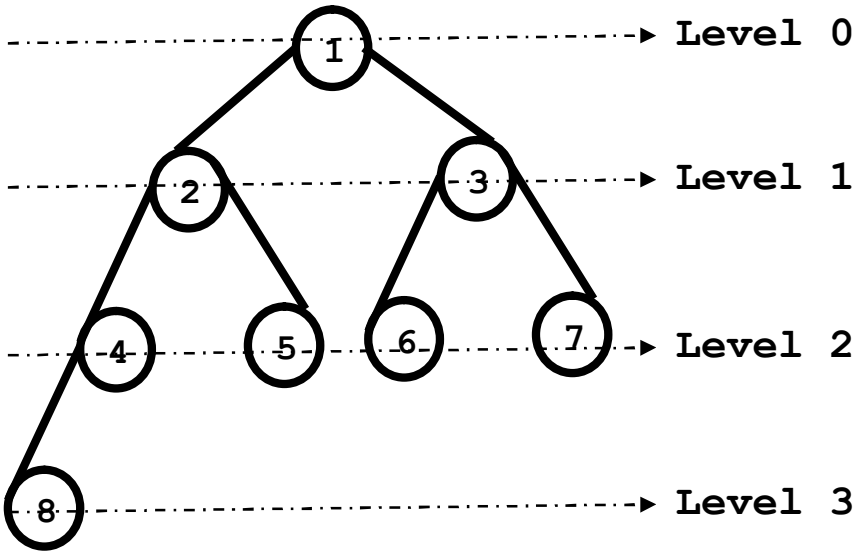
يطلق اسم الشجيرة الفرعية علي أي جزء من الهيكل الشجري والشكل رقم (3) يوضح شجيرات فرعية منبثقة من الشجرة الأساسية.



الشكل رقم (3) الشجيرات الفرعية

- **المستوي** Levels

يستخدم مستوى الشجرة أو عمقها للدلالة علي أكبر عدد من العقد الموجودة في أغصان الشجرة أو أكبر مستوي لأي عقدة بالشجرة والشكل رقم (4) يبين المستويات المختلفة للشجرة وعمقها **(depth)** الذي يساوي **3.**

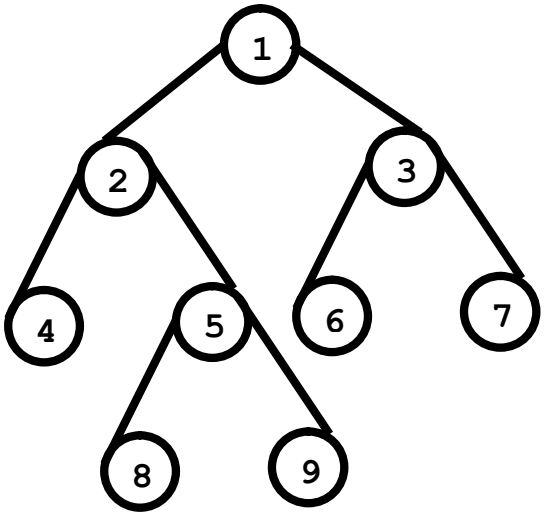


الشكل رقم (4) عمق ومستوي الشجرة

مثال (1)

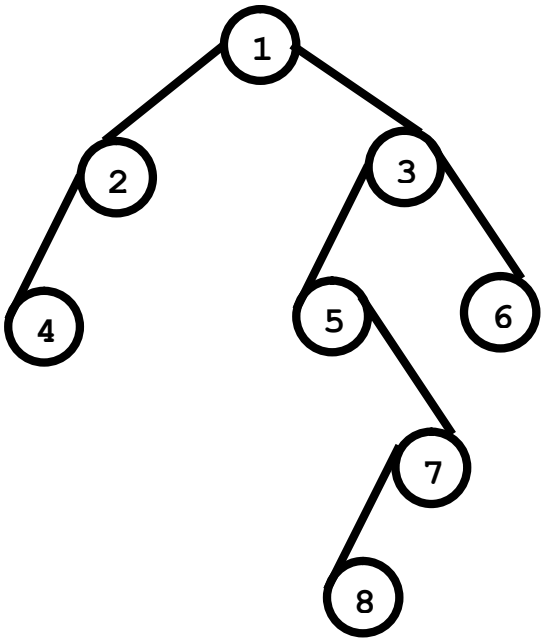
احسب عمق الأشجار التالية

(أ)



العمق (depth) =3

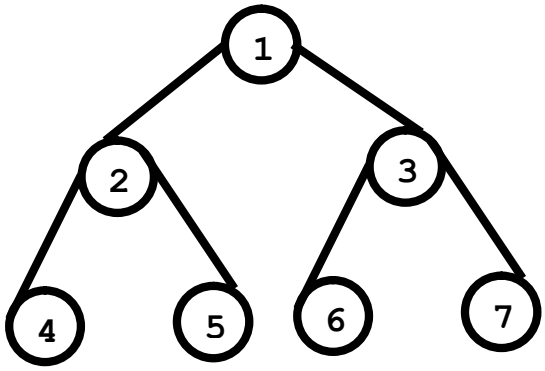
(ب)



العمق (depth) =4

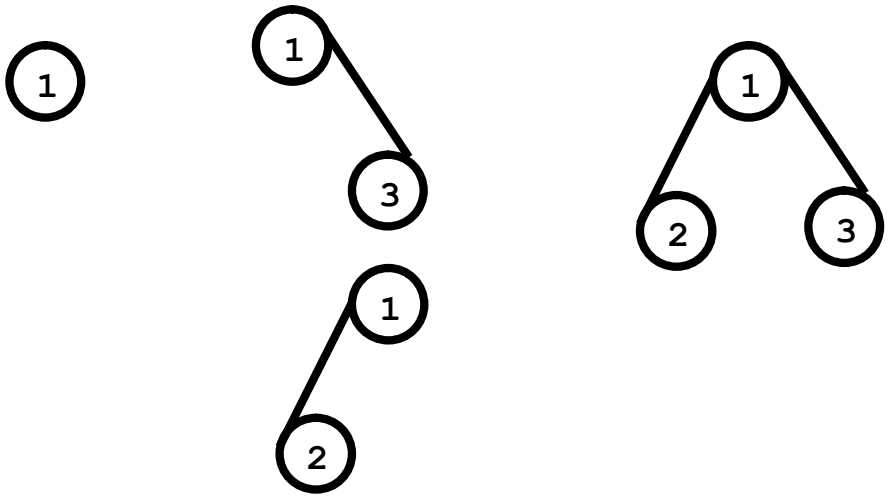
3. الأشجار الثنائية Binary Tree

هي الشجرة التي تحتوى كل عقدة فيها على (**two children**) على الأكثر والشكل رقم (5) يوضحها.



الشكل رقم(5) الشجرة الثنائية

يجب أن يكون لكل عقدة في الشجرة الثنائية عدد **0,1 or 2** من **الأبناء والشكل** رقم (6) يوضح ذلك



عدد الأبناء=**2** عدد الأبناء=**1** عدد الأبناء=**0**

الشكل رقم (6) عدد الأبناء بعقدة في الشجرة الثنائية

كما يمكن أن يكون الابن شجيرة وبالتالي يمكن أن تكون هنالك شجيرتين منفصلتين عن بعضيهما يطلق عليهما الشجيرة اليسرى والشجيرة اليمنى وهما بحد ذاتهما هياكل شجرية ثنائية وبالتالي يكون التعريف متكرراً **Recursive وهذه** ميزة مهمة تسهل عملية البرمجة.

يتميز هيكل الشجرة الثنائية بوجود مؤشري ربط علي الأكثر في أي عقدة أو بمعنى آخر أنه يمكن لعقدة أن تحتوي على ممر واحد أو أثنين أو لا تحتوي على ممر وهي في هذه الحالة تسمى ورقة كما ذكرنا ذلك سابقاً.

1.3 الاتزان في الأشجار الثنائية

نحتاج في كثير من تطبيقات الأشجار لمعرفة حالة الشجرة من حيث الاتزان وذلك لإنجاز عمليات البحث والحذف والإضافة في أقل ممكن ولمعرفة حالة اتزان الشجرة الثنائية من حيث العمق لابد من حساب معامل الاتزان لكل عقدة (**Balance Factor**) إذ يعرف بما يلي:

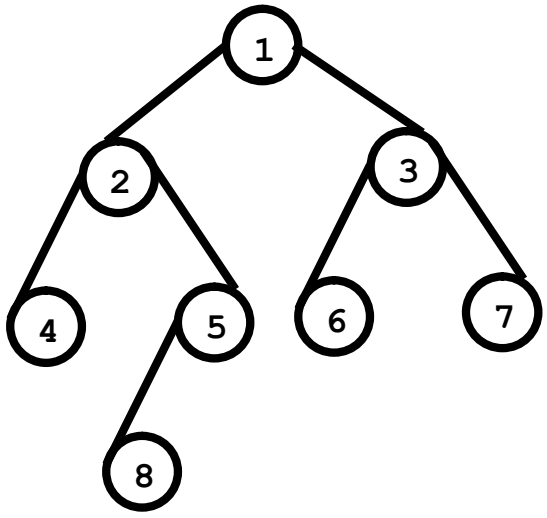
معامل اتزان العقدة= عدد العقد في الشجيرة اليمنى- عددها في الشجيرة اليسرى

ويحسب معامل الاتزان لكل عقدة في الشجرة، نجد أن الشجرة تتزن عندما يكون معامل الاتزان محصوراً بين **-1و1** لكل العقد، أي أن القيمة المطلقة لمعامل اتزان كل عقدة تساوي صفراً أو واحد صحيح، وبناءً على هذا نجد أن معامل اتزان عقدة الورقة دائماً يساوي صفراً.

مثال (2)

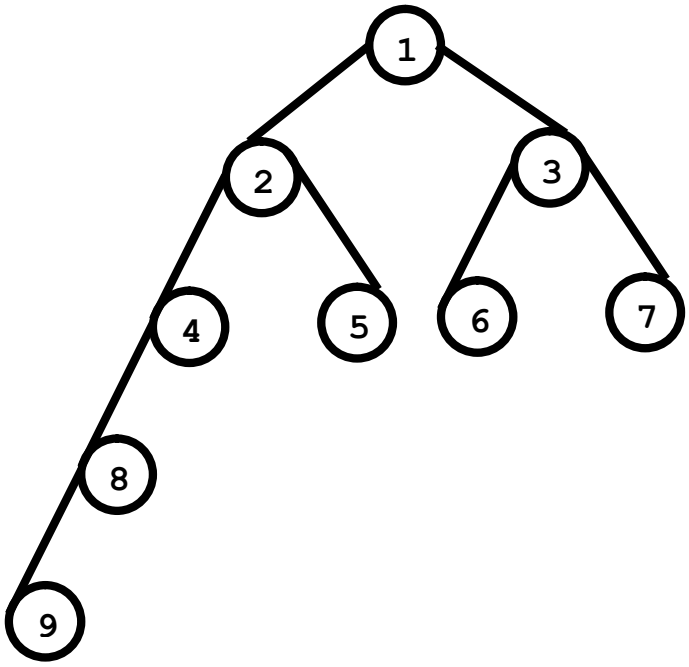
أحسب معامل الاتزان **Balance Factor (BF)** الأشجار التالية:

(أ)



وبناءً علي التعريف السابق نجد أن الشجرة أعلاه (الشكل أ) شجرة متزنة وذلك لأن معامل الاتزان **Balance Factor (BF)** محصوراً بين **-1و1** عندما نقوم بحسابه لكل العقد

Node	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
BF	**-1**	**1**	**0**	**0**	**-1**	**0**	**0**	**0**



(ب)

وبناءً علي التعريف السابق نجد أن الشجرة أعلاه (الشكل ب) شجرة غير متزنة وذلك لأن معامل الاتزان **Balance Factor (BF)** غير محصوراً بين **-1و1** عندما نقوم بحسابه لكل العقد

Node	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
BF	**-2**	**-2**	**0**	**-2**	**0**	**0**	**0**	**-1**	**0**

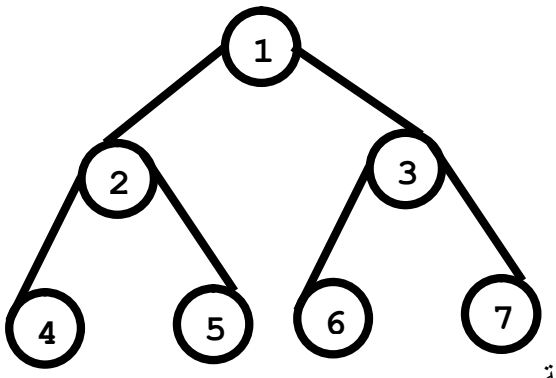
تعرف الأشجار الثنائية المتزنة من حيث العمق باسم **AVL-Tree**[1].

يقال للهياكل الشجرية الثنائية والتي عمقها يساوي **N** بأنها شجرة ثنائية كاملة **Complete Binary Tree** إذا كانت في جميع المستويات من المستوى **0** إلى المستوى **N-1** تحتوي علي أكبر عدد ممكن من العقدة أي أن كل عقدة ترتبط بعقدتين في المستويات من المستوى **0** إلى المستوى **N-1** (عدد العقد في المستوى رقم ***k*** يساوي 2^k) وأن العقد في المستوى**N** تظهر من أقصى اليسار إلي اليمين ويقال بأنها شجرة ثنائية تامة **Full Tree** إذا كانت في جميع المستويات من المستوى **0** إلي المستوى **N** تحتوي على أكبر عدد ممكن من العقدة.

مثال (3)

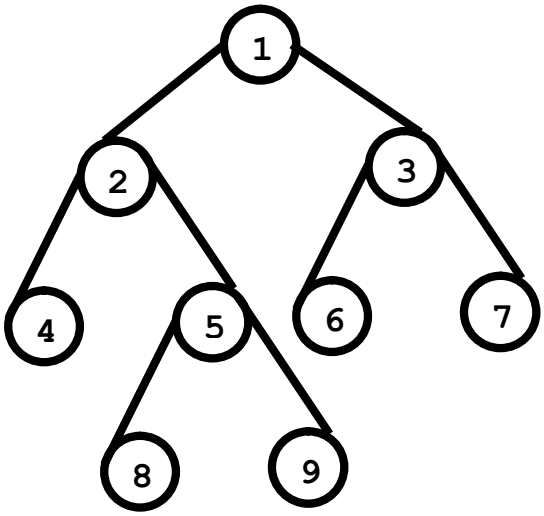
حدد نوع الأشجار التالية:

(أ)



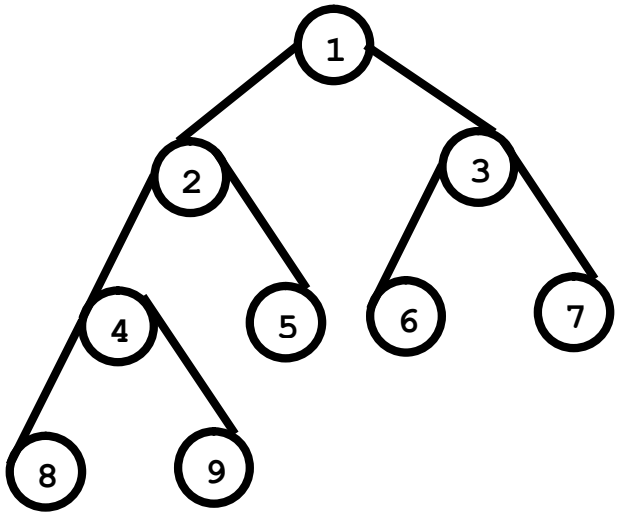
شجرة ثنائية كاملة وتامة

[1] **AVL** هي اختصار لاسم العالمين الروسيين **Adel'son-Vel'skii, Landis** الذي تم وصف هذا الهيكل عام **1962**.



(ب)

شجرة ثنائية غير كاملة وبالتالي غير تامة وذلك لأن العناصر في المستوى الأخير لا تسير من اليسار إلي اليمين دون ترك فراغات.



(ج)

شجرة ثنائية كاملة وغير تامة وذلك لأن العناصر في المستوى الأخير لا تحتوي علي أكبر عدد ممكن من العقدة.

نلاحظ أن عدد العقد في الهياكل الشجرية الثنائية الكاملة والتامة متساوي في جميع المستويات من المستوى **0** إلي المستوى **K-1** وأن عدد العقد في المستوى الأخير رقم **K**

يساوي 2^k في الشجرة الثنائية التامة بينما في الشجرة الثنائية الكاملة يكون أكبر من أو يساوي **1** وأقل من 2^k والجدول رقم (1) يبين عدد العقد في كل مستوى

المستوي	العقد في الشجرة الثنائية الكاملة	العقد في الشجرة الثانية التامة
0	**1**	**1**
1	**2**	**2**
2	**4**	**4**
3	**8**	**8**
4	**16**	**16**
.	.	.
.	.	.
.	.	.
k-1	2^{k-1}	2^{k-1}
k	أكبر من **1** وأقل من 2^k	2^k

جدول رقم (1) عدد العقد في الشجرة الثنائية الكاملة والتامة في كل مستوي

من الجدول رقم (1) نجد أن العدد الكلي للعقد **N** في الشجرة الثنائية التامة هو:

$$N = 1+2+4+8+16+...+2^k$$
$$= 2^0+2^1+2^2+2^3+2^4+...+2^k$$
$$= 2^{k+1}-1$$

وأن عدد الكلي للعقد **N** في الشجرة الثنائية الكاملة يحقق العلاقة التالية:

$$2^k<N<2^{k+1}-1<2^{k+1}$$

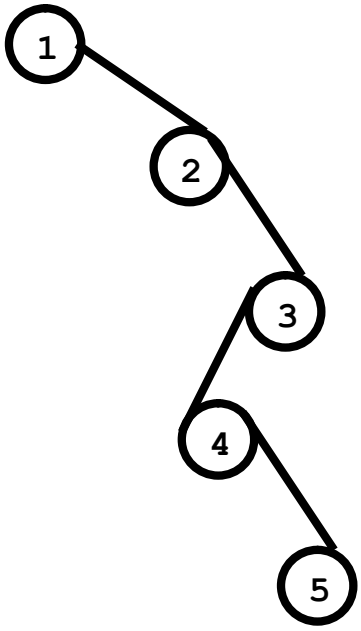
أي أن

$$K<\log_2(N)<k+1$$

مثال (4)

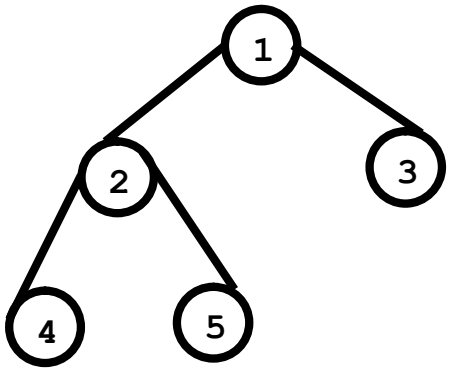
حدد أكبر وأقل عمق لشجرة ثنائية عدد العقد فيها **5**.

(أ) أكبر عمق عندما تكون هنالك عقدة واحدة في كل مستوى والشكل التالي يبن ذلك:



عمق الشجرة يساوي **4**.

(ب) أقل عمق يحقق العلاقة التالية



K<log(5)<k+1
Log(5)=2.32 and k=2

اقل مستوى يساوي **2**.

2.3 التنقل عبر الأشجار الثنائية Binary Tree Traversal

أسلوب التنقل **traversing** عبر الأشجار يقصد به زيارة كل عقدة في الشجرة مرة واحدة وبترتيب معين وذلك بغرض إجراء عملية محددة على البيانات وهو بالتالي غير

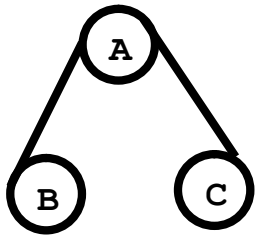
قاصر على الأشجار الثنائية إذ يمكن تنفيذه على أي هيكل بيانات آخر وعادة ما يتم التنقل عبر الأشجار من خلال ثلاثة أساليب هي:

- أسلوب التنقل الوسطي **Inorder Traversal**.
- أسلوب التنقل البعدي **Postorder Traversal**.
- أسلوب التنقل القبلي **Preorder Traversal**.

ويعتبر أسلوب التنقل الوسطي **Inorder Traversal** هو الشائع والأكثر استخداماً.

1.2.3 أسلوب التنقل الوسطي Inorder Traversal

يبدأ التنقل في هذا الأسلوب بزيارة الابن الأيسر **L**ثم العقدة المراد بيانها **N** ثم الابن الأيمن **R** ويمكن أن نختصر هذا الأسلوب بـــ**LNR** وإذا اعتبرنا أن الزيارة تعني طباعة قيمة العقدة وعند تطبيق أسلوب التنقل الوسطي على الشكل التالي



تكون النتيجة هي **BAC**

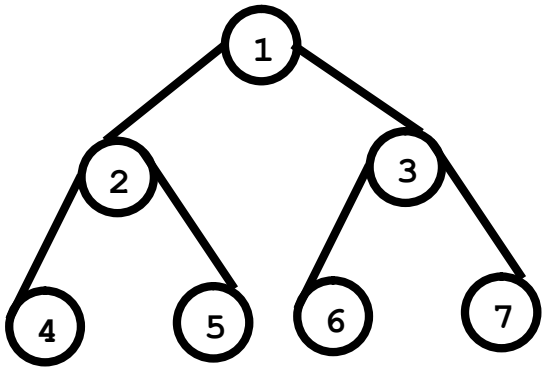
قد يكون الابن الأيسر أو الأيمن أو كليهما عبارة عن شجيرة فرعية فيتم زيارة الشجيرة الفرعية اليسرى أولاَ وفيها يتم زيارة الابن الأيسر وهكذا ونرى في هذا الأسلوب تكراراً يستوجب استخدام استدعاء النداء الذاتي للدالة **Recursive Function** وستؤدي الدالة ثلاث مهام:

- التنقل عبر الشجيرة الفرعية اليسرى للعقدة المدخلة.
- **زيارة العقدة المدخلة.**
- التنقل عبر الشجيرة الفرعية اليمنى للعقدة المدخلة.

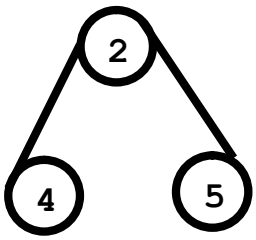
مثال (5)

ما نتيجة الطباعة عند تنفيذ أسلوب التنقل الوسطي علي الهياكل الشجرية التالية

(أ)

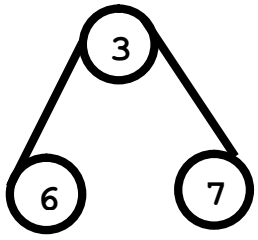


نلاحظ أن للجذر **1** ابن أيسر وابن أيمين وهما عبارة عن شجيرة فرعية نبدأ بالشجيرة الفرعية من الجانب الأيسر وفيها نجد أن العقد **2**لها ابن أيسر **4** وابن أيمن **5** وعليه سوف يتم الزيارة الشجيرة في الجانب الأيسر حسب التسلسل التالي



4->2->5

ومن ثم نقوم بزيارة الجذر **1** وبعدها نقوم بزيارة الشجيرة الفرعية من الجانب الأيمن وفيها نجد أن العقد **3**لها ابن أيسر **6** وابن أيمن **7** وعليه سوف يتم الزيارة الشجيرة حسب التسلسل التالي

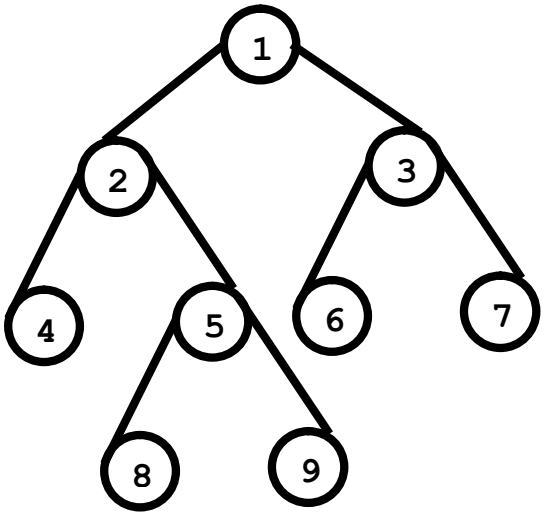


6->3->7

وعليه تكون النتيجة هي

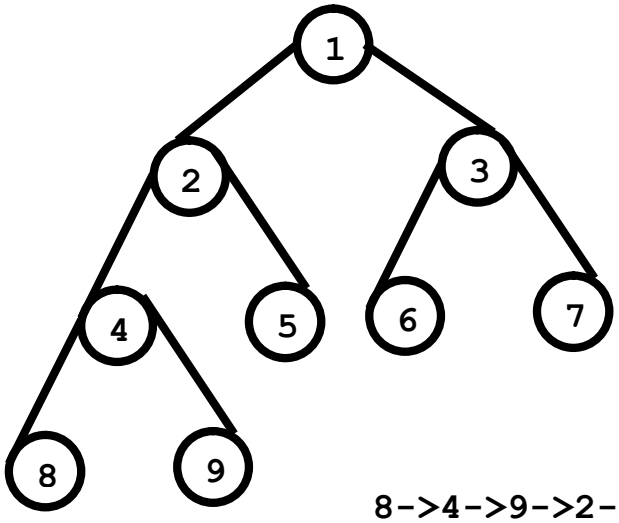
4->2->5->1->6->3->7

(ب)



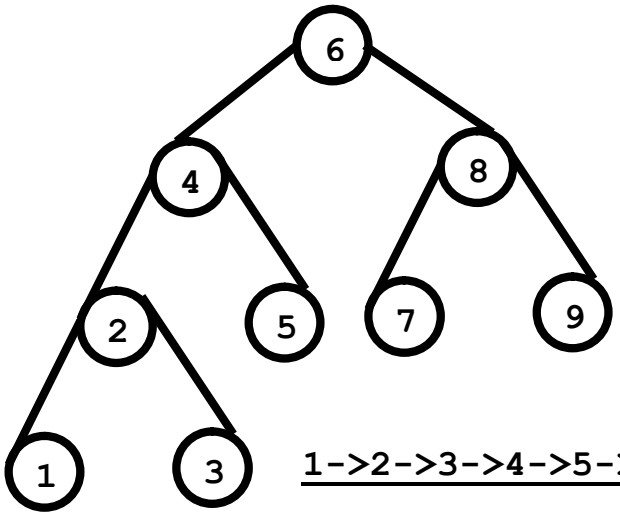
النتيجة هي **4->2->8->5->9->1->6->3->7**

(ج)



النتيجة هي
8->4->9->2->5->1->6->3->7

(د)



النتيجة هي

1->2->3->4->5->6->7->8->9

والدالة التالية تقوم بطباعة قيم عقد الهيكل الشجري الثنائي وفقاً لأسلوب التنقل الوسطي وتستخدم الدالة مفهوم النداء الذاتي:

```
void InOrder (node* t)
{
   if(t !=Null)
   {
      InOrder(t->Left);
      t->display() ;
      InOrder(t->Right) ;
   }
}
```

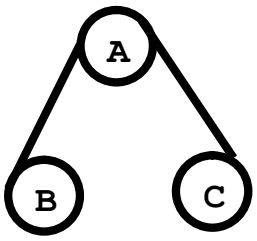
حيث **node** عبارة عن فصيلة يمكن تعريفها كالأتي:

```
class node
{
  public:
    int data;
    node* Left;
    node* Right;
    node(int d1):data(d1),Left(NULL)
                          ,Right(NULL)
    {}
```

```
    void display()
    {
       cout<<data<<" ---> ";
    }
};
```

2.2.3 اسلوب التنقل البعدي Postorder Traversal

يبدأ التنقل في هذا الأسلوب بزيارة الابن الأيسر **Lثم** الابن الأيمن **R** من ثم العقدة المراد بيانها **N** ويمكن أن نختصر هذا الأسلوب بـــ**LRN** وإذا اعتبرنا أن الزيارة تعني طباعة قيمة العقدة وعند تطبيق أسلوب التنقل البعدي علي الشكل التالي



تكون النتيجة هي **BCA**

قد يكون الابن الأيسر أو الأيمن أو كليهما عبارة عن شجيرة فرعية فيتم زيارة الشجيرة الفرعية من الجانب الأيسر أولاَ وفيها يتم زيارة الابن الأيسر وهكذا ونري في هذا الأسلوب تكرار يستوجب استخدام استدعاء النداء الذاتي للدالة **Recursive Function** وستؤدى الدالة ثلاث مهام:

- التنقل عبر الشجيرة الفرعية من الجانب الأيسر للعقدة المدخلة.
- التنقل عبر الشجيرة الفرعية من الجانب الأيمن للعقدة المدخلة.
- **زيارة العقدة المدخلة.**

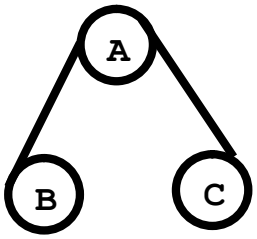
الدالة التالية تقوم بطباعة قيم عقد الهيكل الشجري الثنائي وفقاً لأسلوب التنقل البعدي وتستخدم الدالة مفهوم النداء الذاتي:

```
void PostOrder(node* t)
{
   if(t !=Null)
   {
```

```
        PostOrder(t->Left);
        PostOrder(t->Right) ;
        t->display() ;
    }
}
```

3.2.3 أسلوب التنقل القبلي Preorder Traversal

يبدأ التنقل في هذا الأسلوب بزيارة العقدة المراد بيانها **N** ثم الابن الأيسر **L ومن ثم** الابن الأيمن **R** ويمكن أن نختصر هذا الأسلوب بـــ**NLR** وإذا اعتبرنا أن الزيارة تعني طباعة قيمة العقدة وعند تطبيق أسلوب التنقل القبلي علي الشكل التالي



تكون النتيجة هي **ABC**

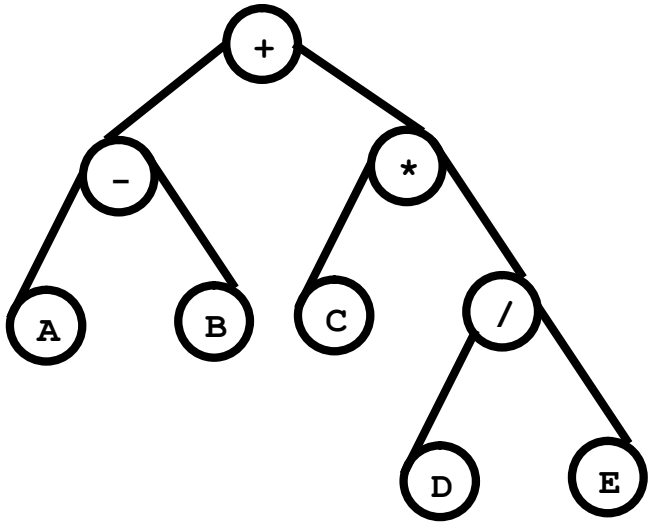
الدالة التالية تقوم بطباعة قيم عقد الهيكل الشجري الثنائي وفقاً لأسلوب التنقل القبلي وتستخدم الدالة مفهوم النداء الذاتي:

```
void PreOrder (node* t)
{
    if(t !=Null)
    {
        t->display() ;
        PreOrder(t->Left);
        PreOrder(t->Right) ;
    }
}
```

مثال (6)

ما نتيجة الطباعة عند تنفيذ أسلوب التنقل الوسطي والبعدي والقبلي علي الاشجار الثنائية التالية

(أ)

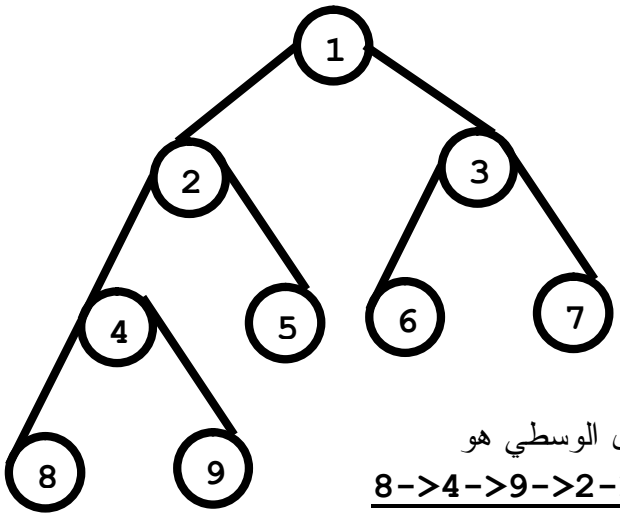


☒ نتيجة أسلوب التنقل الوسطي هو **A-B+C*D/E**

☒ نتيجة أسلوب التنقل البعدي هو **AB-CDE/*+**

☒ نتيجة أسلوب التنقل القبلي هو **+-AB*C/DE**

(ب)



☒ نتيجة أسلوب التنقل الوسطي هو

8->4->9->2->5->1->6->3->7

☒ نتيجة أسلوب التنقل البعدي هو

8->9->4->5->2->6->7->3->1

☒ نتيجة أسلوب التنقل القبلي هو

1->2->4->8->9->5->3->6->7

مثال (7)

إذا طبقنا أسلوب التنقل الوسطي علي شجرة ثنائية ينتج الترتيب التالي

1->2->3->4->5->6->7->8->9

وإذا طبقنا أسلوب التنقل القبلي علي ذات الشجرة ينتج الترتيب التالي

6->4->2->1->3->5->8->7->9

أرسم الشجرة.

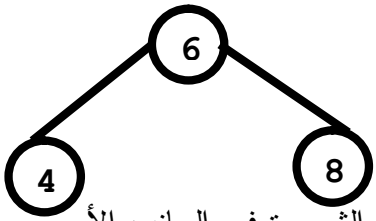
الحل

1- نجد في أسلوب التنقل القبلي أن الجذر هو أول عقدة يتم زيارته وبالتالي يعتبر العقدة **6** هي جذر الشجرة الثنائية.

2- نجد في أسلوب التنقل الوسطي أن الجذر(وهو العقدة **6**) يفصل بين الجانب الأيسر للشجرة والجانب الأيمن وعليه فإن العقد **1,2,3,4,5** تمثل الجانب الأيسر للشجرة بينما تمثل العقد **7,8,9** الجانب الأيمن للشجرة.

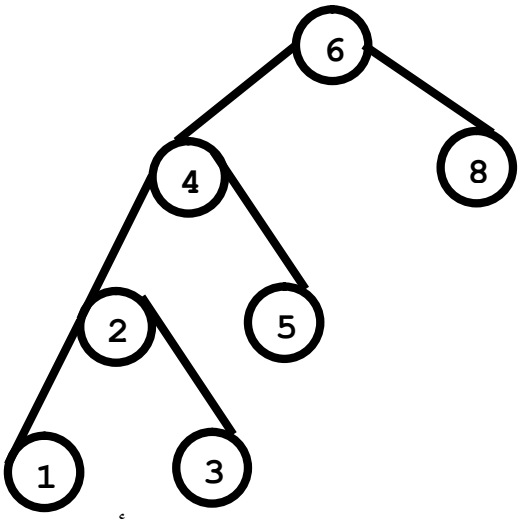
3- تعتبر أول عقدة من العقد التي يتألف منها الجانب الأيسر للشجرة وهي العقد **{1,2,3,4,5}** في أسلوب التنقل القبلي هي الابن الأيسر للجذر **6** وهي العقدة **4** وأن أول عقدة من العقد التي يتألف منها الجانب الأيمن للشجرة وهي العقد **{7,8,9}** في أسلوب التنقل القبلي هي الابن الأيمن للجذر **6** وهي العقدة **8**.

وحتى الآن تم تحديد الجذر والابن الأيمن والأيسر كما يوضحه الشكل التالي:



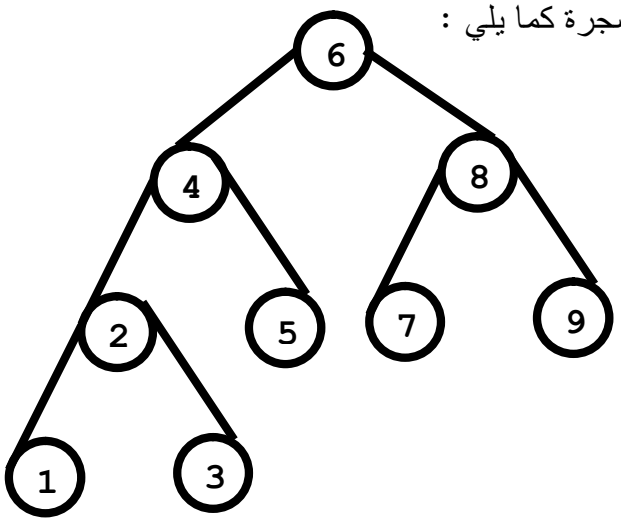
4- يعتبر الابن الأيسر للجذر هو جذر الشجيرة في الجانب الأيسر وعليه فإن العقدة **4** هي جذر الشجيرة في الجانب الأيسر ونجد في أسلوب التنقل الوسطي أن الجذر(وهو العقدة **4**) يفصل بين الجانب الأيسر للشجيرة والجانب الأيمن وعليه فإن العقد **1,2,3** تمثل الجانب الأيسر للشجيرة بينما تمثل العقدة **5** الجانب الأيمن للشجيرة الفرعية.

5- تعتبر أول عقدة من العقد التي يتألف منها الجانب الأيسر للشجيرة وهي العقد **{1,2,3}** في أسلوب التنقل القبلي هي الابن الأيسر للجذر **4** وهي العقدة **2** وهي في نفس الوقت تكون جذراً للعقدتين المتبقيتين وهما **{1,3}** وحسب أسلوب التنقل الوسطي أن الجذر(وهو العقدة **2**) يفصل بين الجانب الأيسر للشجيرة والجانب الأيمن وعليه فإن العقدة **1** تمثل الجانب الأيسر للشجيرة بينما تمثل العقدة **3**الجانب الأيمن



6- يعتبر الابن الأيمن للجذر هو جذر الشجيرة في الجانب الأيمن وعليه فإن العقدة **8** هي جذر الشجيرة في الجانب الأيمن ونجد في أسلوب التنقل الوسطي أن الجذر (وهو العقدة **8**) يفصل بين الجانب الأيسر للشجيرة والجانب الأيمن وعليه فإن العقد **7** تمثل الجانب الأيسر للشجيرة بينما تمثل العقدة **9** الجانب الأيمن للشجيرة الفرعية.

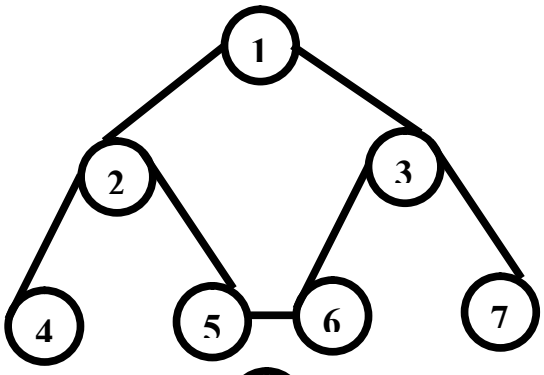
وعليه يصبح الشكل النهائي للشجرة كما يلي :



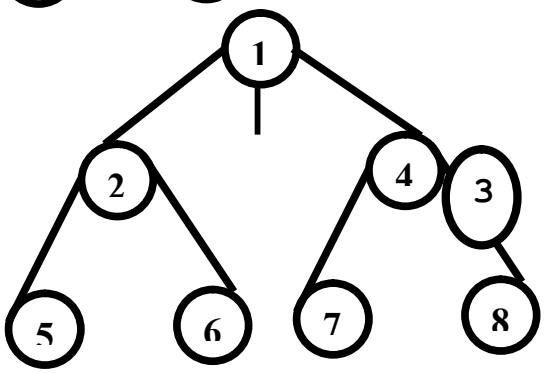
تدريب (1)

وضّح أسباب عدم صحة الهياكل الشجيرة التالية:

أ-

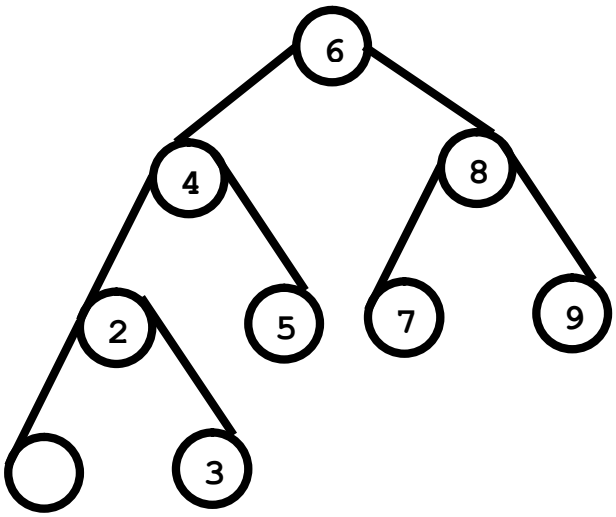


ب -



4. شجرة البحث الثنائي Binary Search Tree

تعتبر شجرة البحث الثنائي نوعاً معيناً من الأشجار الثنائية تحققق فيه علاقة الابن الأيسر لأي عقدة أقل من أو يساوي العقدة والابن الأيمن أكبر من العقدة والشكل رقم (7) يوضح نموذجاً لذلك



الشكل رقم (7) شجرة البحث الثنائي

هنالك عدد من الدوال يمكن أن تتعامل مع شجرة البحث الثنائي كعمليات نذكر من هذه الدوال الآتي:

- البحث **عن عقدة داخل الشجرة.**
- إضافة عقدة إلي الشجرة.
- حذف عقدة محددة بالشجرة.

والفصيلة التالية توضح ذلك.

```
class Tree
{
  private:
   node* root;
  public:
     Tree();
     node* find(int);
     int findMin();
     int findMax();
    void insert(int);
    void InOrder(node*);
    void PostOrder(node*);
    void PreOrder(node*);
     void print()
```

```
    {
          cout<<"\nInOrder    : ";
          InOrder(root);
          cout<<"\npostOrder : ";
          PostOrder(root);
          cout<<"\nPreOrder   : ";
          PreOrder(root);
    }
    bool isempty();
};
bool Tree::isempty()
{
   return (root==NULL);
}
Tree::Tree():root(NULL)
{
}
```

1.4 البحث عن عقدة داخل الشجرة

خوارزمية البحث عن عقدة

نضع العنصر المراد البحث عنه في عقدة ونعلم أن الجذر هو العقدة الوحيدة التي يمكن الوصول إليها مباشرة لمقارنة قيمته مع القيمة الموجودة في الجذر فإذا لم تساويها تتم المقارنة لتحديد المسار إما في اتجاه الابن الأيسر أو الابن الأيمن ثم نقارن قيمة العنصر المطلوب البحث عنه مع العقدة بالجانب الأيسر أو الأيمن حسب المقارنة التي تمت سابقاً ويتم تكرار هذه العملية حتى الوصول للعنصر المطلوب أو الوصول لورقة في آخر مستوى للشجرة و هذا يعني أن العنصر غير موجود.

الدالة **find** هي داله عضو في الفصيلة **Tree** تقوم بالبحث عن العقدة التي قيمتها **d**.

```
node* Tree::find(int d)
{
   node* current=root;
   while (current->data!=d)
   {
```

```
                                    if(d<current->data)
                                  current=current->left;
        else
                                 current=current->right;
                                      if (current==NULL)
            return NULL;
    }
    return current;
}
```

ويمكن استدعاء الدالة من خلال الشفرة التالية:

```
int key=35;
node* found=a.find(key);
if (found==NULL)
  cout<<"\nnode not found";
else
  cout<<"\nfound node with data="<<key;
```

وفيها يتم البحث عن العقدة التي قيمتها تساوي مثلاً 35.

2.4 البحث عن قيمة الحد الأدنى والأعلى في الشجرة

للبحث عن قيمة الحد الأدنى نذهب للابن الموجود في الجانب الأيسر للجذر ومن ثم نذهب إلي الابن الموجود على يساره وهكذا نكرر هذه الخطوات إلي أن نصل إلى عقدة لا تحتوي على child في جانبها الأيسر وتكون هي عقدة الحد الأدنى في الشجرة.

الدالة findMin تقوم بإيجاد أقل قيمة للعقد

```
int Tree::findMin()
{
    node* current=root;
    node* par=root;
    while (current!=NULL)
    {
       par=current;
       current=current->left;
    }
    return par->data;
}
```

للبحث عن قيمة الحد الأعلى نذهب للابن الموجود في الجانب الأيمن للجذر ومن ثم نذهب إلي الابن الموجود على يمينه وهكذا نكرر هذه الخطوات إلي أن نصل لعقدة لا تحتوى على ابن في الجانب الأيمن **right child** وتكون هي عقدة الحد الأعلى في الشجرة.

الدالة **findMax** تقوم بإيجاد أكبر قيمة للعقد

```
int Tree::findMax()
{
   node* current=root;
   node* par=root;
   while (current!=NULL)
   {
      par=current;
          current=current->right;
   }
   return par->data;
}
```

3.4 إضافة عقدة إلي الشجرة

خوارزمية إضافة عقدة

-- Precon وجود شجرة أو على الأقل جذر

-----Input النص المراد إضافته

--Process زيارة العقد و مقارنة العنصر لتحديد موقعه

ثم تحديد هل العقدة تحتوي على موقع خالٍ أم لا تحتوي

Out put --

Postc تتغير الشجرة

عند إدخال أي عقدة جديدة للشجرة لابد من تحديد الموقع المناسب لها من خلال تحديد الأب لهذه العقدة وبعد إجراء عملية المقارنة سيتم ربطها كابن أيسر أو أيمن حسب نتيجة المقارنة والشفرة التالية تبين عملية إضافة عقدة جديدة.

```
void Tree::insert(int d)
{
```

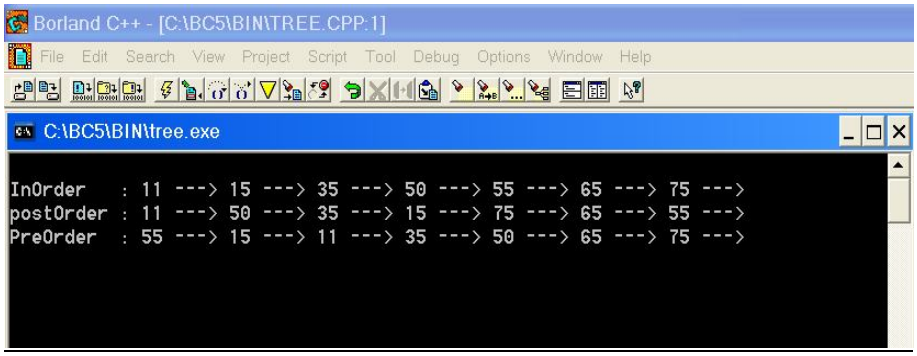
```
  node* newnode=new node(d);
  if(isempty())
     root=newnode;
  else
  {
      node* current=root;
      node* parent;
      while(true)
      {
        parent=current;
         if (d<current->data)
         {
             current=current->left;
             if (current==NULL)
              {
             parent->left=newnode;
              return;
             }
         }
         else
         {
             current=current->right;
             if (current==NULL)
             {
             parent->right=newnode;
                return;
             }
         }
     }
  }
}
```

ويمكن استدعاء الدالة من خلال الشفرة التالية:

```
Tree a;
a.insert(55);
a.insert(15);
a.insert(65);
a.insert(35);
```

```
a.insert(11);
a.insert(50);
a.insert(75);
a.print();
```

ويمكن استخدام الدالة **print** لطباعة قيم العقد حسب طرق التنقل المختلفة (وسطي، قبلي، بعدي) الشكل رقم (7) يوضح ذلك.



الشكل رقم (7) استعراض قيم العقدة حسب طرق التنقل المختلفة

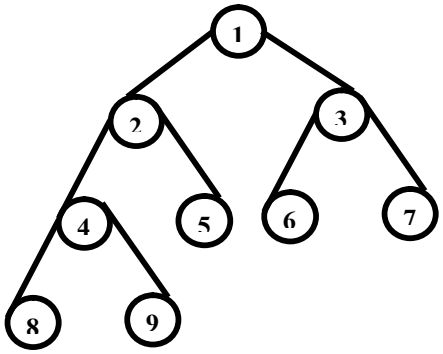
4.4 حذف عقدة محددة بالشجرة

عند تنفيذ عملية الحذف نقوم أولاً بعملية البحث عن العقدة المراد حذفها ويكون وضع هذه العقدة في الحالات التالية:

- ✓ العقدة غير موجودة
- ✓ العقدة موجودة وهي عبارة عن ورقة **leaf**.
- ✓ العقدة موجودة ولها ابن واحد.
- ✓ العقدة موجودة ولها ابنان.

تدريب (2)

ادرس الشجرة التالية جيداً ثم اجب على الأسئلة التالية:

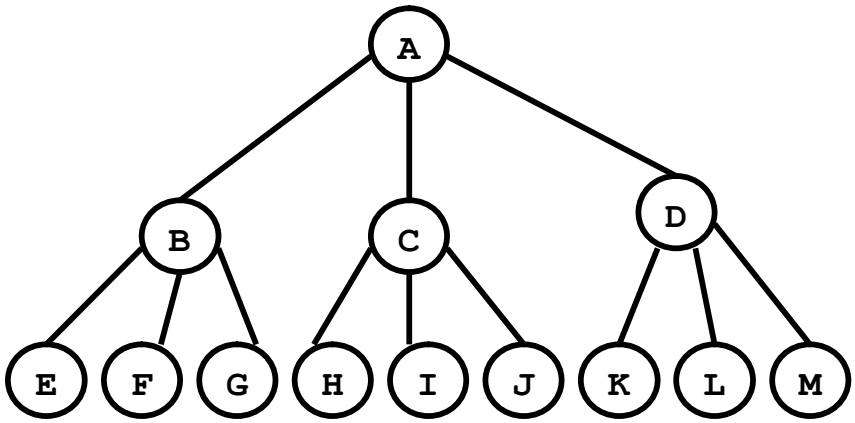


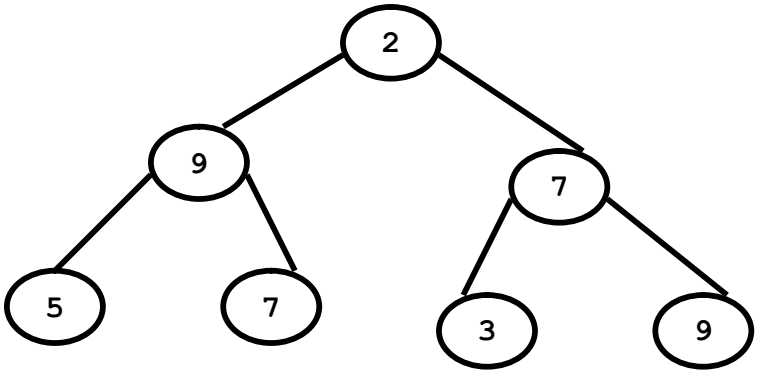
1. ما جذر هذه الشجرة ؟
2. ما هو الابن الأيمن للعقدة 2 ؟ وما والدها ؟
3. ما هي العناصر الورقية في هذه الشجرة ؟
4. ما هو ارتفاع هذه الشجرة ؟
5. ما هي نتيجة استعراض عقد الشجرة حسب أسلوب التنقل الوسطي؟
6. ما هي نتيجة استعراض عقد الشجرة حسب أسلوب التنقل البعدي ؟
7. ما هي نتيجة استعراض عقد الشجرة حسب أسلوب التنقل القبلي ؟

5. طرق ترتيب عناصر الشجرة الثنائية

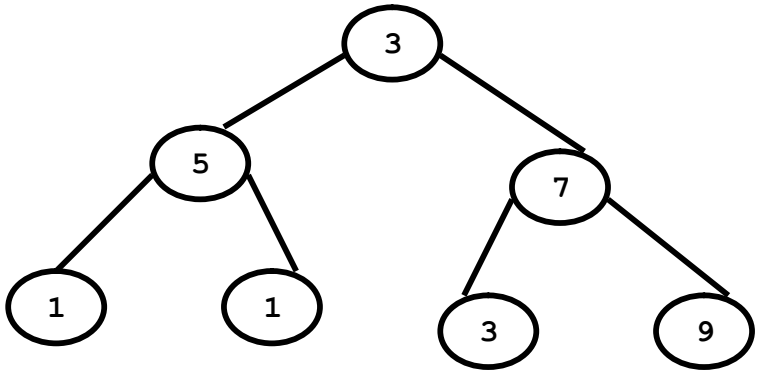
الترتيب الأفقي (من اليسار إلى اليمين): تكون الشجرة مرتبه ترتيباً أفقيا إذا كانت عقدة الفرع الأيسر دائماً اقل أو يساوي الجزر وعقدة الفرع اليمين دائماً أكبر من الجزر.

كالمثال التالي:-

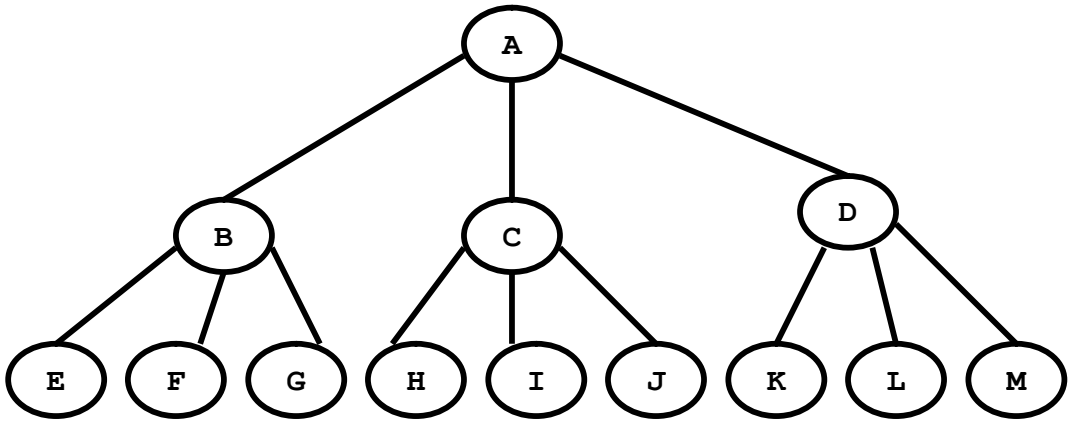




2/ الترتيب العمودي (من الأعلى إلى الأسفل): تكون الشجرة مرتبه ترتيباً عمودياً إذا كانت عقدة كل جزر اكبر (أو اصغر) من الجذور أو الأوراق التي تليها في المستوى

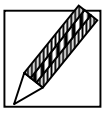


كل جز أصغر من الجذور في المستويات العليا.



كل جزر أكبر من الجذور أو الأوراق في المستويات العليا.

تدريب (3)



- ارسم شجرة البحث الثنائية، إذا كانت نتيجة استعراض قيم العقد حسب أسلوب التنقل الوسطي هو:

11,13,15,35,40,45,50,55,61

6. تطبيقات الأشجار

أ/ التعبير الرياضي

ويمكن قراءة التعبير الرياضي بإحدى الطرق التالية:-

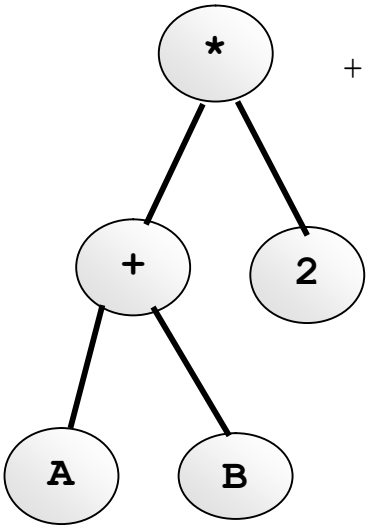
1/ يسار جزر يمين (left root right) وهذه الطريقة تعطينا صيغة التعبير على صورة (infix).

```
A + B
```

توضح الشجرة الثنائية المعادلة
Infix Notation

A * (B+C)
Left Root
Right

2/ جزر يسار يمين (root left right) وهذه الطريقة تعطينا صيغة (prefix).

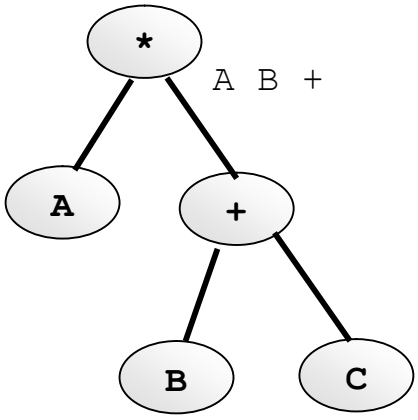


+ A B

توضح الشجرة الثنائية المعادلة
Prefix Notation

(A+B)*2
Root Left
Right

3 / يسار يمين جزر (left root right) وهذه الطريقة تعطينا صيغة (postfix) .



A B +

Post fix Notation

A * (B+C)
Left Right

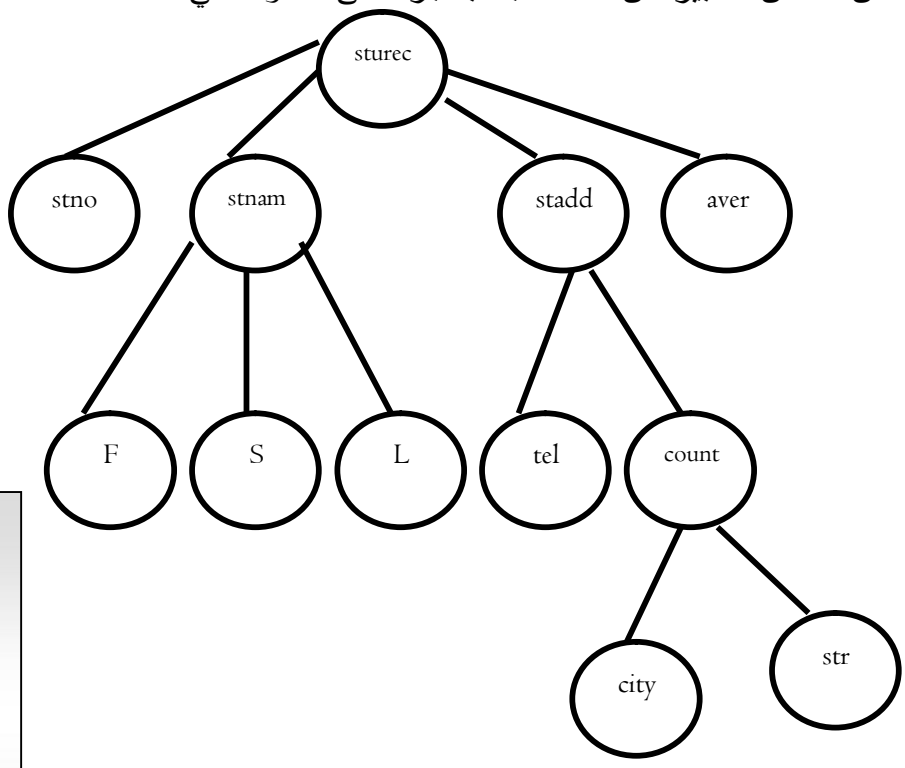
ب / السجلات

من المعلوم أن السجل الموضح في الشكل التالي يمكن وصفة بلغة كوبول:

Srurec

stno	stname			stadd			Aver
	F	S	L	country		Tel	
				city	str		

من الممكن التعبير عن هذا السجل بشجرة على النحو التالي:



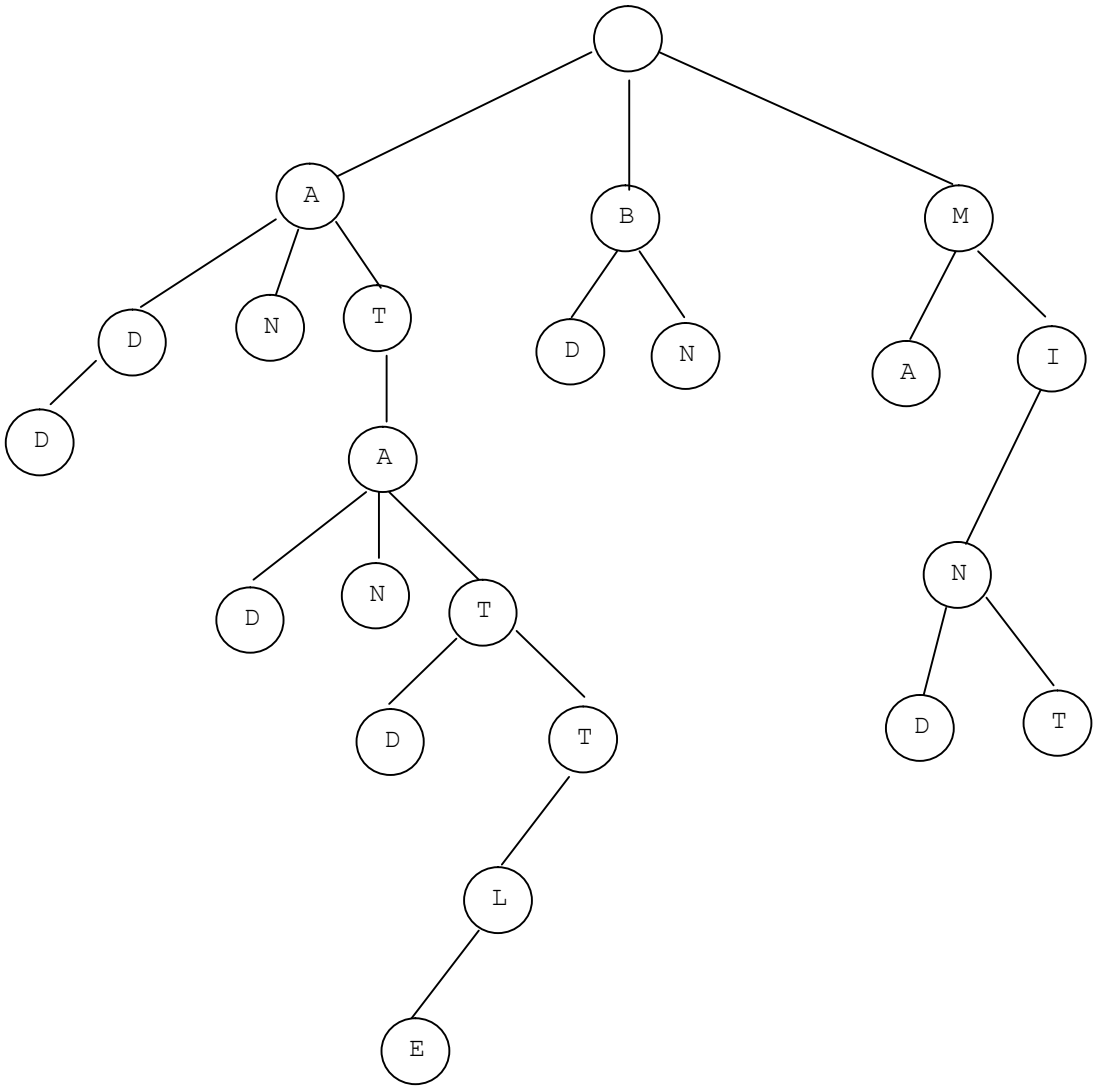

```
01
Sturec
  02
Stno
  02
Stname
   03 F
   03 S
   03 L
```

ج / القواميس:-

لتكن لدينا مجموعة الكلمات التالية :

AND,AT,ADD,ADD,BAT,BATTEL,MAN,MAD,MINT,MIND.

فانه يمكن تمثيل هذه الكلمات بشجرة على النحو التالي:-



لاحظ أن عدد أحرف الكلمات الأساسية يساوى 35 حرفاً بينما كان عدد العقد في الشجرة التي مثلت الكلمات يساوى 24 عقدة ممّا يعنى أن هنالك مجالاً لتوفير المواقع التخزينية التي تحتاجها البيانات.

```
#include <iostream.h>
#include <conio.h>
class node
{
  public:
    int data;
    node* left;
    node* right;
    node(int d1):data(d1),left(NULL),right(NULL)
    {}
    void display()
    {
       cout<<data<<" ---> ";
    }
};
class Tree
{
  private:
   node* root;
  public:
     Tree();
     node* find(int);
     void insert(int);
    void InOrder(node*);
    void PostOrder(node*);
    void PreOrder(node*);
    void print()
    {
         cout<<"\nInOrder   : ";
         InOrder(root);
         cout<<"\npostOrder : ";
         PostOrder(root);
         cout<<"\nPreOrder  : ";
         PreOrder(root);
    }
    bool isempty();
};
```

```
bool Tree::isempty()
{
   return (root==NULL);
}
Tree::Tree():root(NULL)
{
   cout<<"|-->";
}
void Tree::InOrder(node* t)
{
   if(t !=NULL)
   {
      InOrder(t->left);
      t->display() ;
      InOrder(t->right) ;
   }
}
void Tree::PostOrder(node* t)
{
   if(t !=NULL)
   {
      PostOrder(t->left);
      PostOrder(t->right) ;
      t->display() ;
   }
}
void Tree::PreOrder(node* t)
{
   if(t !=NULL)
   {
      t->display() ;
      PreOrder(t->left);
      PreOrder(t->right) ;
   }
}

node* Tree::find(int d)
```

```
{
     node* current=root;
    while (current->data!=d)
    {
     if(d<current->data)
         current=current->left;
       else
         current=current->right;
         if (current==NULL)
          return NULL;
     }
    return current;
}
void Tree::insert(int d)
{
  //cout<<"Insert new Node with key "<<d<<endl;
  node* newnode=new node(d);
  if(isempty())
     root=newnode;
  else
  {
      node* current=root;
      node* parent;
      while(true)
      {
         parent=current;
          if (d<current->data)
          {
              current=current->left;
              if (current==NULL)
             {
              parent->left=newnode;
                 return;
             }
          }
          else
          {
```

```
                current=current->right;
                if (current==NULL)
               {
                parent->right=newnode;
                  return;
               }
            }
           }
  }
}
void main()
{
   Tree a;
    a.insert(55);
    a.insert(15);
    a.insert(65);
    a.insert(35);
    a.insert(11);
    a.insert(50);
    a.insert(75);
   int key=35;
   node* found=a.find(key);
   if (found==NULL)
    cout<<"\n   node   with   data="<<key<<"   not
found\n\n";
   else
    cout<<"\nfound node with data="<<key<<"\n\n";
      a.insert(key);
    a.print();
   getch();
}
```

7. تكرار مداخل القيم وتطابقها داخل شجرة واحدة

لا يمثل تكرار مداخل العقد في عملية الإدخال أي مشكلة فيمكن أن نضيف أكثر من عقدة تتماثل القيم داخلها مع قيم عقد أخرى في التركيب، المشكلة تظهر مع عملية البحث.
عند استدعاء ()find للبحث عن عقده معينة سيكون الوصول قاصراً على العقدة الأولى فقط ولن يكون هناك إمكانية الوصول لباقي الــNode المشتملة على نفس العنصر أو المدخل.
يمكن تعديل ()find للبحث عن عنصر ثالث للتمييز بين عناصر البيانات في حالة تطابق القيم داخلها لكن الأمر يستغرق وقت. إذن الحل هو تجنب إدخال العقد ذات المداخل المتطابقة.

إذا لم نحذف المداخل لابد من تعديل روتين ()insert للتأكد من أي ازدواجية في عملية للإضافة وبالتالي التوقف عند إضافة المزيد من العناصر في حالة وجود مداخل متكررة.

Red Black Tree .8

هي شكل معدل من تراكيب الــ binary search tree وهو يعالج الضعف في هياكل الأشجار الثنائية الذي يتمثل في الإخلال بتوازن الــtree في حالة إضافة العناصر بشكل عشوائي إليها وتعد من أفضل السبل لتحقيق توازن الــ tree حيث تضمن توازنها دائماً .

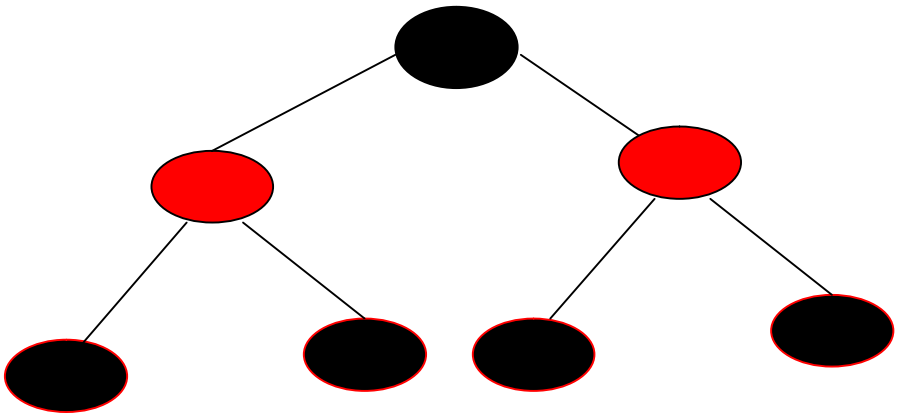
1.8 خصائص الــ Red Black Tree

1. تأخذ العقد داخلها ألوان معينة حمراء أو سوداء.
2. عند الإدخال أو الحذف نطبق قواعد معينة من شأنها الحفاظ على تنظيمات هذه الألوان.

قواعد الألوان

1. لون العقدة أحمر أو اسود.
2. لون الــ Root أسود.
3. لون الــ Parent أحمر وال Child أسود.

4. لابد أن يتساوى عدد العقد السوداء في كل مسار خارج من الـــ Root إلى الـــ leaf أو الـــ null child.



س- ما هي أهم خاصية للـــread black tree؟

ج- تقدم نموذج لشجرة متوازنة وتحافظ على توازن الشجرة دائماً.

س- كيف نستفيد من قواعد الألوان؟

ج- في عمليتي الحذف والإضافة وفى التحقق من توازن الشجرة واكتمالها.

س- ماذا يحدث عند الإخلال بقاعدة من قواعد الألوان؟

ج- 1- تغيير ألوان العقد

2- إجراء عمليات التدوير

2.8 تعريفات خاصة بقواعد الألوان

Black height

عبارة عن عدد العقد السوداء من الـــroot للـــ leaf.

Red to Red conflict

هي حالة تظهر عند إدخال عقدة جديدة باللون الأحمر عندما يكون الأب باللون الأحمر أي وجود عقدتين أحمر في صف واحد وهي تمثل انتهاك للقاعدة الثالثة من قواعد الألوان ونتفادى هذه المشكلة باستخدام تقنية التدوير.

Flip colour

هي عملية تتم عند إدخال أو حذف عقدة من الـــtree فإذا نتج عن ذلك انتهاك للقاعدة الثالثة يتم تعديل ألوان الـــ parent للون الأسود لتفادي انتهاك القاعدة.

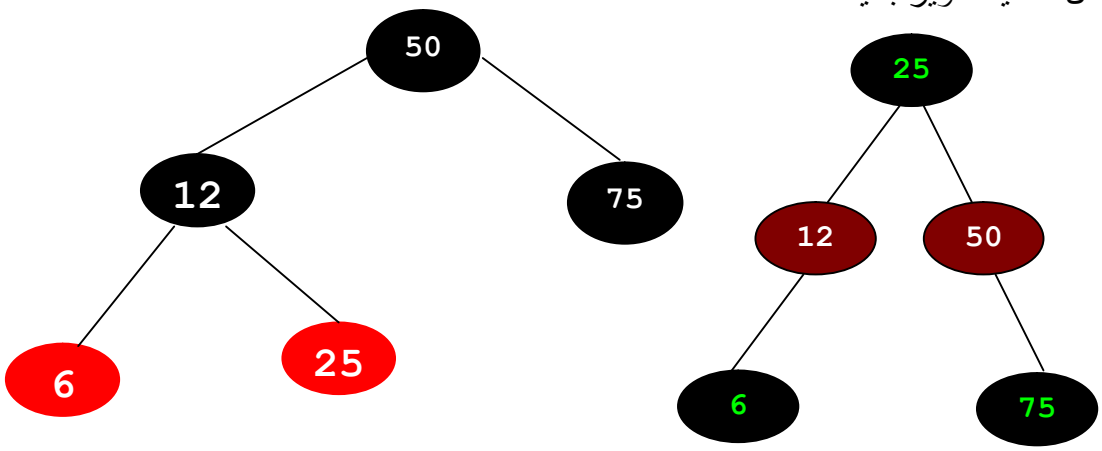
يلاحظ هنا أنه عند إجراء عملية تعديل الألوان إذا حدث انتهاك لقاعدة أخرى تكون الشجرة غير موزونة أصلاً وفى هذه الحالة يتم تفادي مشكلة الـــ read to read بواسطة **التدوير**.

3.8 التدوير

أسلوب من أساليب إعادة تنظيم العقد ويتم التدوير في خطوتين:

- رفع بعض العقد وخفض البعض الآخر للمساعدة في تحقيق التوازن.
- التأكد من عدم الخروج عن أسس وخصائص شجرة البحث الثنائية.

مثال لعملية تدوير بسيطة:



شروط التدوير

- إذا كان التدوير ناحية اليمين تأكد من أن العقدة العلوية لها Child أيسر.
- إذا كان التدوير ناحية اليسار تأكد من أن العقدة العلوية لها Child أيمن.

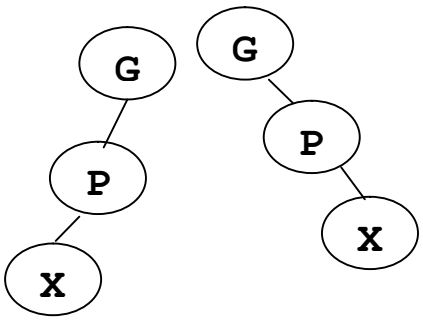
حالات خاصة في التدوير

عند إجراء عملية التدوير لوزن الشجرة نحتاج إلى عملية تدوير واحدة إذا كانت العقدة الجديدة تمثل outside grandchild للـــ Grant parent .

أما إذا كانت العقدة الجديدة Inside grandchild نحتاج للتدوير مرتين مره لجعل العقدة الجديدة عقده خارجية والثانية لوزن الشجرة ويقودنا هذا الحديث إلى تعريف الـــ inside out side.

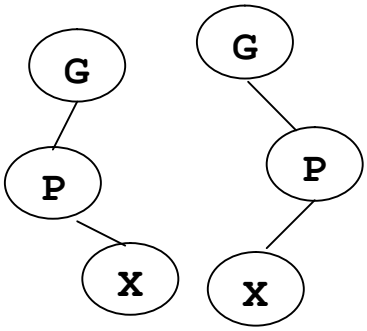
تعريف الـــ Out side

عندما تكون العقدة الجديدة هي عبارة عن ابن الشمال ل p و الـــp ابن شمال للـــG، أو تكون X ابن يمين للـــ P وال P ابن يمين للـــ G.



تعريف الـــ inside

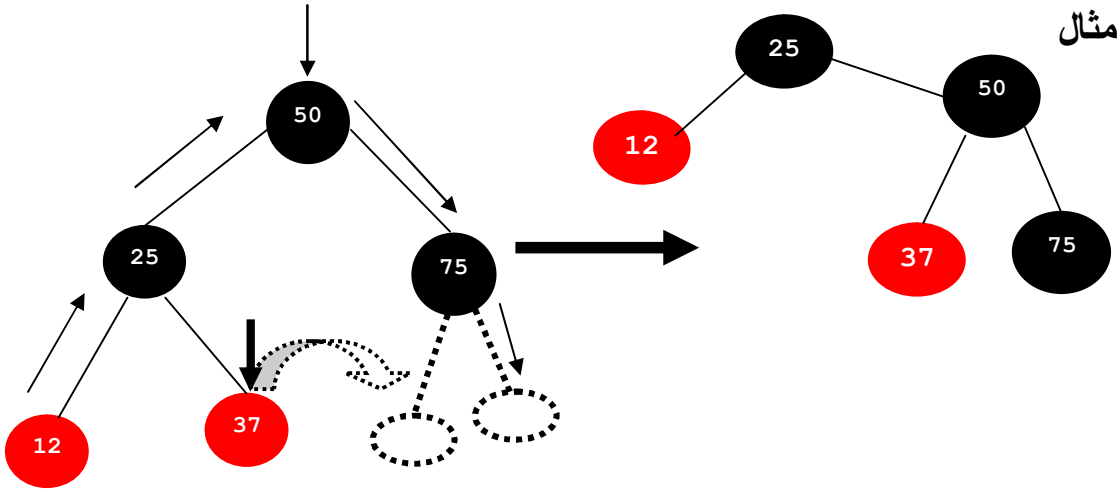
عندما تكون الـــ X مخالفة في موقعها للـــ p و الـــp مخالفة للـــ Gمثال



P ابن يمين للـــ G و X ابن شمال للـــ p

P ابن شمال للـــ G و X ابن يمين للـــ p

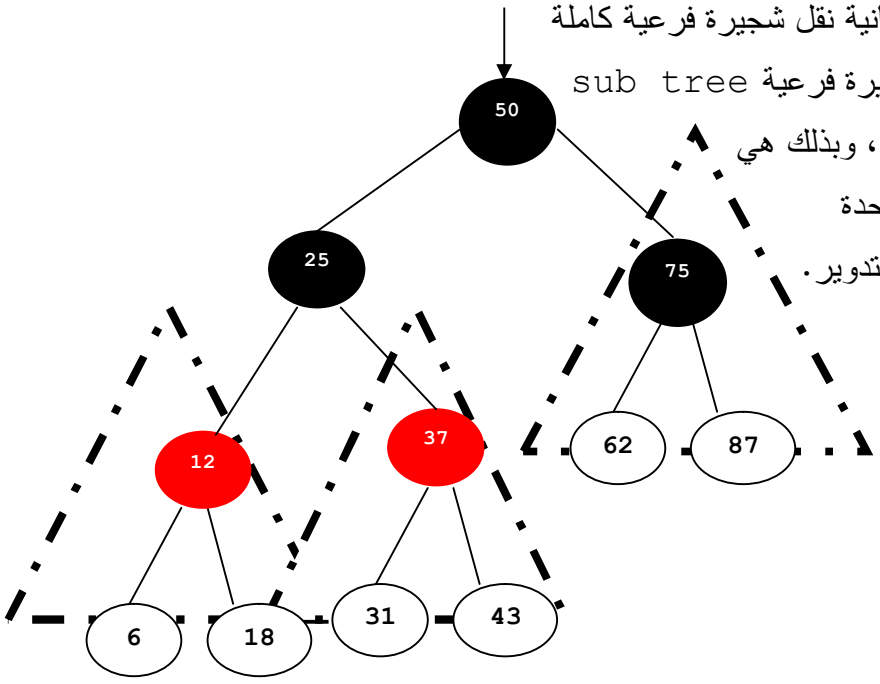
مثال

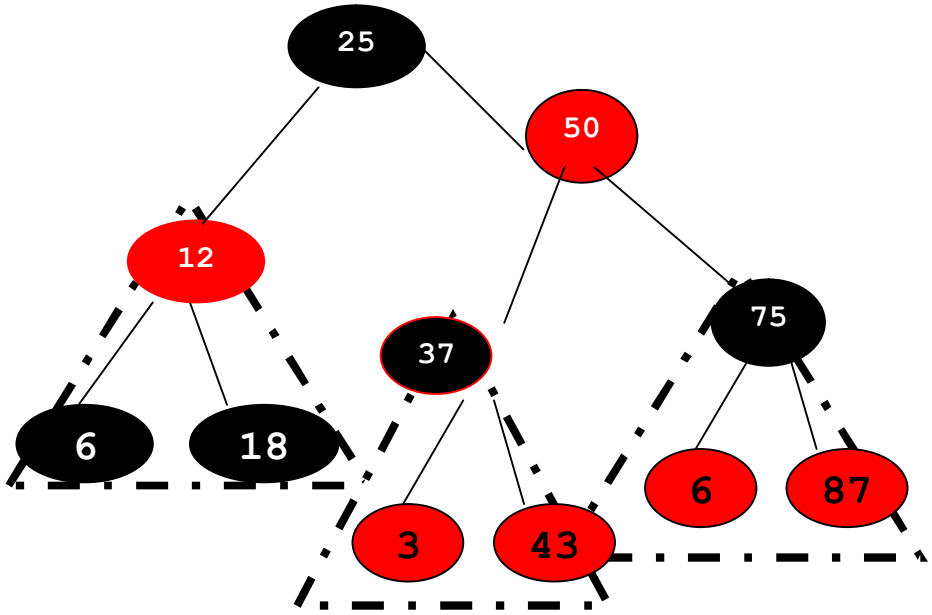


العقدة (37) هي inside grandchild الخاص بالعقدة في أعلى التركيب والمشتملة على المدخل (50). إما العقدة (12) هي outside grandchild وبالنسبة إلى inside grandchild فسيظل دائماً منفصل عن الـــparent الخاص به. وسيعاود ربطه مع الـــ grandparent السابق الخاص به إذا كانت العقدة التي ستتحرك لأعلى والذي هو إبن شمالٍ للعقدة العلوية في التدوير الأيمن.

نقل شجيرة فرعية في عملية التدوير

من الشكل نلاحظ إمكانية نقل شجيرة فرعية كاملة وذلك بمعاملة كل شجيرة فرعية sub tree على أنها عقدة واحدة ، وبذلك هي تتحرك جميعاً معاً كوحدة واحدة خلال عملية التدوير.





4.8 أسلوب الإضافة في الـــ Red Black Tree

يتم إدخال عقد جديدة في الـــRead Black Tree بدون الإخلال بتوازن الشجرة لأن الشجرة توزن نفسها تلقائياً لثناء إدخال العقدة فإذا حدث أي نوع من الانتهاك لقواعد الألوان كما سبق يتم حله بالتدوير وتبديل الألوان.

توجد طريقتان للإدخال:

1- Top Down Insertion

وفيها نحتاج إلى إجراء تغيرات خاصة بالتركيب أثناء البحث عن المكان الذي سندخل فيه عقدة جديدة.

2- Bottom_up insertion

نبحث عن مكان لإدخال العقدة ثم نرجع للشجرة لعمل تغيرات تركيبية . هذا النوع كفائته أقل لأننا نمر على الشجرة مرتين لعمل التغييرات.

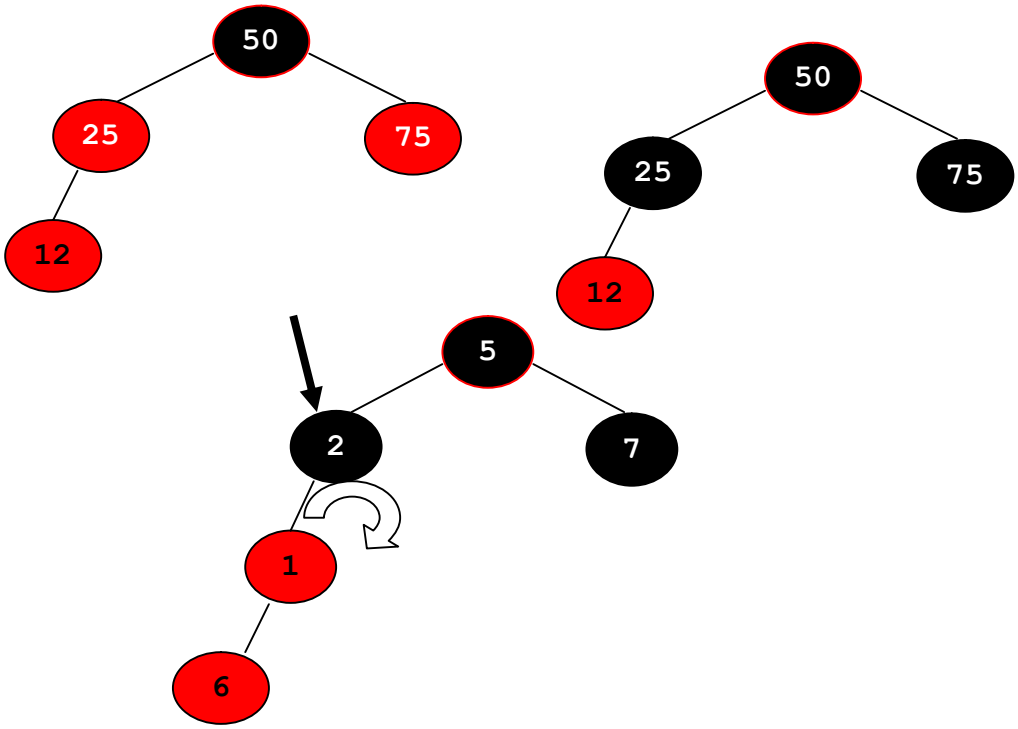
عند إجراء عملية إدخال للعقد سيكون من الضروري **تبديل الألوان للعقد فيما يعرف بعملية الـــ** color filip **وأحياناً نقع في مشكلة الـــ** **read read conflict** **والتي يكون فيها الأب والابن بلون أحمر معا لذا سنعطى أمثلة لتوضيح هذه الحالات**

ملحوظة : أي عقدة عند إدخالها تدخل باللون الأحمر

مثال (1)

قيمة الـــRoot 50 ثم أدخل القيم التالية 25 75 12 6

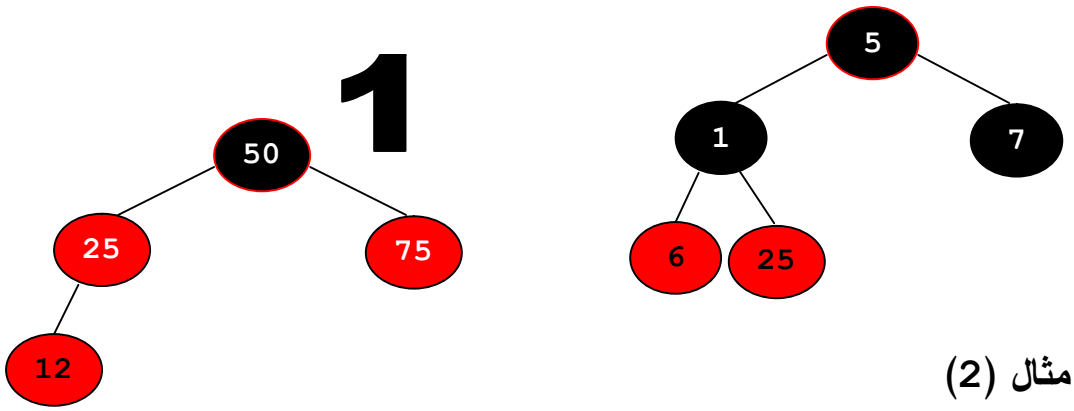
عند إدخال العقدة التي تحمل الرقم 12 صار لدينا مشكلة **read read conflict** **والحل هو تغيير ألوان كل من 25 و75 إلى اللون الأسود ولا تمثل مشكلة.**



ندخل العقدة التي تحمل الرقم 6

تربط أولاً مع العقدة رقم 12 ك outside child ونظراً لأن أي عقدة جديدة تدخل باللون الأحمر تحدث مرة أخرى مشكلة الـــ red red conflict.

الحل نقوم بعملية تدوير نرفع العقده12 لأعلى وتلقائياً تنخفض 25 لتصبح ابناً يميناً للـ12 والـ6 ابن شمال للـــ 12 ثم نغير لون الـ12 من الأحمر إلى الأسود وبذلك تكون الشجرة موزونة كما يلي:



مثال (2)

قيمة الـــRoot 50 ثم ادخل القيم التالية 25 75 12 18

1- عند إدخال العقدة التي تحمل الرقم 12 صار لدينا مشكلة red red conflict

2- والحل هو تغيير ألوان كل من 25 و75 إلى اللون الأسود ولا تمثل مشكلة.

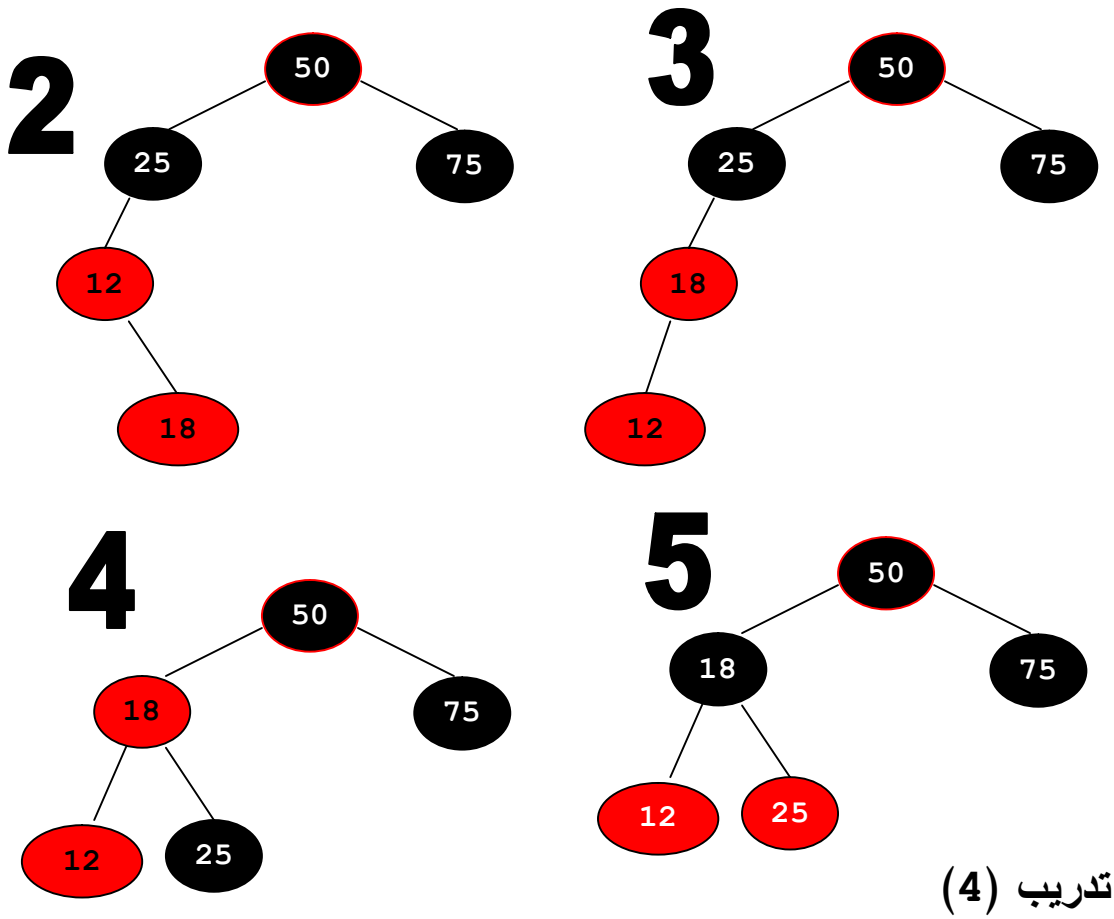
عند إدخال العقدة التي تحمل الرقم 18 تكون ابناً يميناً للعقدة 12وهذه هي الحالة التي تحتاج فيها لإجراء التدوير مرتين inside grand child

بجانب ظهور مشكلة الـــ read read conflict مره أخرى

3- إذن الحل هو :أولا نرفع العقدة التي تحمل الرقم 18 لأعلى لتكون ابناً شمالاً للـ25 والـ12 ابن شمال للـ18 أي outside grandchild

4- ندور مرة ثانيه وذلك بجعل 25 ابن يمين للـ18 و12 ابناً شمالاً للـ18 كما موضح.

5-نغير لون الـ18 إلى أسود و الـ25 إلى لون احمر وبذلك نجد أن الشجرة استعادت توازنها وصحتها مره أخرى.



تدريب (4)

1. ما هي أهم خاصية للـــ read black tree ؟
2. كيف نستفيد من قواعد الألوان ؟
3. ماذا يحدث عند الإخلال بقاعدة من قواعد الألوان؟

9. تركيب وتطبيقات 2-3-4 trees

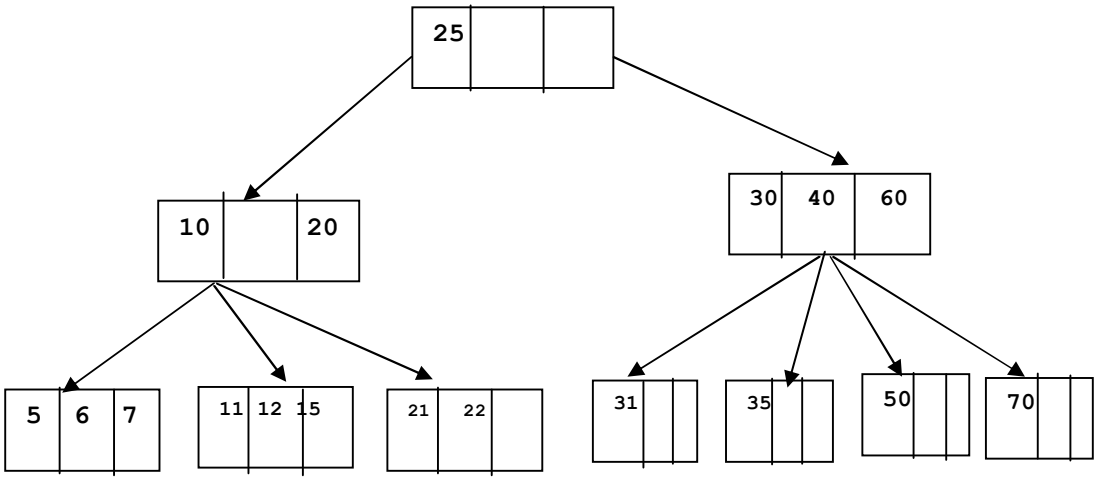
في الأشجار الثنائية تحتوي node على عنصر واحد من عناصر البيانات وعدد من children يصل إلى اثنين كأقصى حد ،أما لو أشتمل node على أكثر من عنصر بيانات واحد وأكثر من اثنين children فإن ذلك يمثل تركيباً جديداً اسمه multilayer tree وبذا تعتبر 2-3-4 tree نموذج من multiway tree حيث تحتوى node فيها على ثلاثة عناصر وعدد أربعة children .

1.9 دلالة الاسم

تشير الأرقام هنا إلى عدد الروابط التي قد توجد بين child node node عقدة ابن وعقدة أخرى تتخذ الأشكال الآتية:

* التي تحتوي node على عنصر بيانات واحد لها أطفلان child .
* التي تحتوي node على عنصرين بيانات لها ثلاثة أطفال child .
* التي تحتوي node على ثلاثة عناصر لها أربعة أطفال child .

وهذه هي العلاقة المميزة للتركيب ففي الوقت الذي لا تحتوي فيه leaf node على أية children. نجدها من الممكن أن تشتمل على عنصر واحد أو اثنين أو ثلاثة من البيانات، ومن غير المسموح بموجود أية node خالية في هذا التركيب.



2.9 أسلوب تنظيم البيانات

الإيضاح سنطلق إلى عناصر البيانات أرقام من 0—9 والروابط من 0—3 كما في الشكل تم تنظيم البيانات حسب الترتيب التصاعدي لمداخل كل من اليسار إلى اليمين وذلك كما في binary tree حيث left child اقل منroot، right child اكبر من root مع وجود **الإضافات الآتية في 2-3-4tree:**

* تحتوى جميع children في sub tree المشتملة على R (0) على مداخل أقل من 0

* تحتوى جميع children في sub tree المشتملة على R(1) على مداخل أكبر0 وأقل1
*تحتوى جميع children في sub tree المشتملة على R(2) على مداخل أكبر من1 وأقل من 2.
* تحتوى جميع children في sub tree المشتملة على R(3) على مداخل أكبر من 2.

10. كفاءة 2-3-4 T

1- السرعة

عرفنا في R.B.T لا بد من زيادة NODE واحدة في كل مستوى أثناء عملية البحث سواء كان لإيجاد أو إضافة عدد المستويات الموجودة $\log_2 (N+1)$ نفس الشيء في 2-3-4 T الفرق أن عدد المستويات أقل من R.B.T ولكن العقدة تحتوى على أربعة Child مكتملة فسيكون ارتفاعها $\log_4 (N)$ يختلف لوغريثم الأساس 2 على لوغريثم الأساس 4 بمضروب ثابت في العدد 2 يبلغ ارتفاعها ½ ارتفاع R.B.T بشرط جميع العقد مكتملة وإلا سيتراوح بين $\log_2 (N+1)$ $\log_2 (N+1)_2$ يترتب علي التناقص تقليل زمن البحث العيوب وجود العديد من العناصر داخل العقدة الواحدة يزيد زمن البحث لا سيما إذا كان linear search الذي يزيد بالبحث بنسبة معينة تتوقف علي M متوسط عدد العناصر $M*\log_4 N$ إذا أخذنا المتوسط $2*\log_4 N$ وهو رقم ثابت بسيط يمكن التقاضي عنه في معادلة Big O notation وعلي ذلك يحول تزايد عدد العناصر يجعلها مماثلة R.B.T في البحث.

2- متطلبات التخزين

يوجد داخل كل node عدد ثلاثة مؤشرات pointer لعناصر البيانات وأربعة Children تابعين لها ويتخذ تخزين هذا العدد من Pointer شكل منظومات ونعلم أنه ليس شرطاً استخدام جميع الـــPointer. ففي حالة العقدة المشتملة علي عنصر بيانات 1 سيهدر نحو ثلثي مساحة البيانات ونصف مساحة children أما إذا كانت تحتوى على عنصرين فسيهدر ثلث مساحة البيانات وربع مساحة children أما في R.B.T فتحتوى علي عدد قليل من العناصر.

الخلاصة

- عزيزي الدارس، نلخّص وحدتنا في أننا قمنا بتعريف الأشجار بأنها واحدة من نماذج هياكل البيانات التي تنتمي لمجموعة أكثر شمولا تسمى المخطط **Graph** وتتكون الشجرة من مجموعة عقد **Nodes** تربط بينها ممرات **Edges**.

- ثم قمنا بتعريف بعض العناصر المستخدمة في الأشجار مثل الممر، المسار، الورقة، الشجيرة الفرعية، والأبناء.

- ثم تحدثنا عن الأشجار الثنائية وعن عملية اتزان الأشجار وأوضحنا أن معامل الاتزان يحسب بالمعادلة التالية: **معامل اتزان العقدة=عدد العقد في الشجيرة اليمنى- عددها في الشجيرة اليسرى.**

- أوضحنا في جزء آخر اسلوب التدوير وهو اسلوب من أساليب إعادة تنظيم العقد ويتم التدوير في خطوتين:

- رفع بعض العقد وخفض البعض الآخر للمساعدة في تحقيق التوازن.

- التأكد من عدم الخروج عن أسس وخصائص شجرة البحث الثنائية.

- كذلك استعرضنا أساليب التنقل عبر الأشجار: أسلوب التنقل الوسطي Inorder Traversal

- أسلوب التنقل البعدي Postorder Traversal - أسلوب التنقل القبلي Preorder Traversal.

أتمنى عزيزي الدارس أن نكون قد وفقنا في تقديم هذه الوحدة،

أمنياتنا بالتوفيق.

لمحة مسبقة عن الوحدة التالية

سوف نتطرق بإذن الله في الوحدة القادمة إلى المخططات وكيفية استخدامها لتمثيل العلاقات الهرمية بين البيانات، وسنرى كيف أنها تتألف من ممرات تنقسم إلى ممرات موجهّة وممرات غير موجّهة، وكذلك سنتعرّف على طرق تمثيل المخططات.

إجابات التدريبات

تدريب (1)

أ- لأنه يحتوي على دورة 3 – 6 – 5 – 2 – 1

ب- العقدة 1 تحتوي على أكثر من اثنين (ابنين)

تدريب (2)

1- 1

2- الابن **5** والدها 1

3- العناصر الورقية 9 8 7 6 5

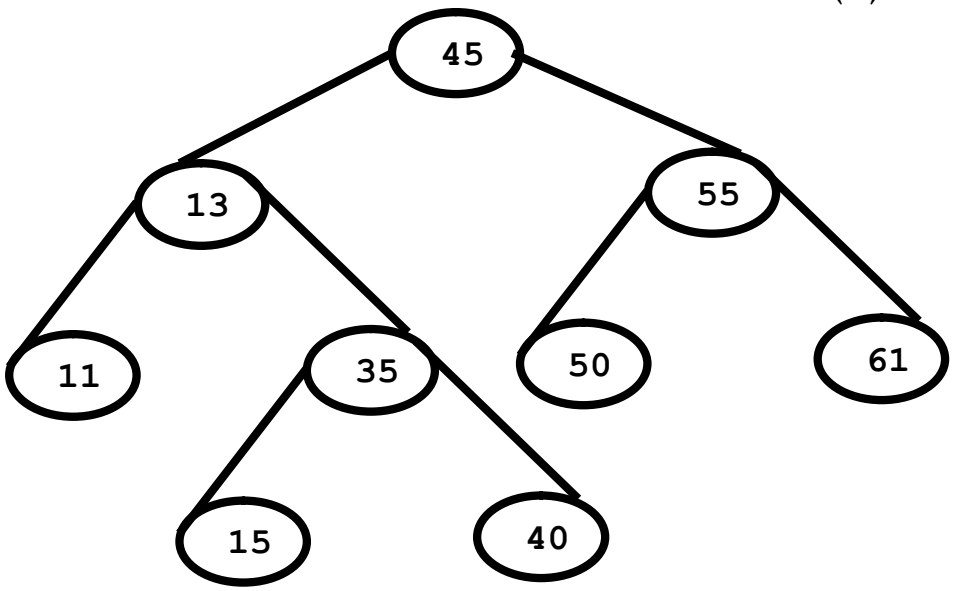
4- ارتفاع هذه الشجرة 3

5- حسب اسلوب التنقل الوسطي 7 3 6 1 5 2 9 4 8

6- حسب اسلوب التنقل البعدي 3 7 6 2 5 4 9 8

7- حسب اسلوب التنقل القبلي 7 6 3 5 9 8 4 2 1

تدريب (3)



تدريب (4)

1. تقدم نموذج لشجرة متوازنة وتحافظ على توازن الشجرة دائماً.
2. في عمليتي الحذف والإضافة وفي التحقق من توازن الشجرة واكتمالها.
3.
   * تغيير ألوان العقد.
   * إجراء عمليات التدوير.

مسرد المصطلحات

الأشجار Trees

تتكون الشجرة من مجموعة عقد Nodes تربط بينها ممرات Edges.

الأبناء Children

هي العقد التي تنحدر مباشرة من العقدة العلوية وترتبط بها بواسطة مؤشرات الربط يطلق علي العقدة العلوية اسم الأب Parent والعقد التي تنحدر منها أبناء Children.

لممر Edge

هو عبارة عن مؤشر الربط الذي يربط عقدتين مثل مؤشر الربط الذي يربط بين الجذر والابن.

الورقة Leaf

هي عبارة عن عقدة لا تحتوي علي أبناء وفي الشكل رقم (5-1) تسمى العقد J و I و H و G و E أوراقاً.

المسار Path

هو عبارة عن مجموعة متتالية من الممرات وإذا وصل المسار إلي ورقة يسمى المسار غصناً Branch.

الشجيرة الفرعية Sub Tree

يطلق اسم الشجيرة الفرعية علي أي جزء من الهيكل الشجري يوضح شجيرات فرعية منبثقة من الشجرة الأساسية.

المستوى Levels

يستخدم مستوى الشجرة أو عمقها للدلالة علي أكبر عدد من العقد الموجودة في أغصان الشجرة أو أكبر مستوي لأي عقدة بالشجرة.

الأشجار الثنائية Binary Tree

هي الشجرة التي تحتوى كل عقدة فيها على (two children).

شجرة البحث الثنائي Binary Search Tree

تعتبر شجرة البحث الثنائي نوعاً معيناً من الأشجار الثنائية تتحقق فيه علاقة الابن الأيسر لأي عقدة أقل من أو يساوي العقدة والابن الأيمن أكبر من العقدة.

Red Black Tree

هي شكل معدل من تراكيب ال binary search tree وهو يعالج الضعف في هياكل الأشجار الثنائية الذي يتمثل في الإخلال بتوازن الtree

التدوير

أسلوب من أساليب إعادة تنظيم العقد ويتم التدوير في خطوتين:

- رفع بعض العقد وخفض البعض الآخر للمساعدة في تحقيق التوازن.
- التأكد من عدم الخروج عن أسس وخصائص شجرة البحث الثنائية.

المراجع

1. William Ford and William Topp, **Data Structure with C++**, Prentice-Hall Inc 1996.
2. Sara Baase ad Allen Van Gelder, **Computer algorithms**, Addison Wesley Longman 2000.
3. Deitel and Deital, **C++ How to program**, Prentice-Hall Inc 2001.

الوحدة السابعة

المخططات Graphs



محتويات الوحدة

الموضوع	الصفحة
المقدمة	**219**
تمهيد	219
أهداف الوحدة	219
1. المخططات	**220**
1.1 تعريفات	221
2.1 المقارنة بين المخططات والأشجار	226
2. طرق تمثيل المخططات	**227**
1.2 استخدام مصفوفة التجاور	228
2.2 استخدام القوائم المتصلة	230
الخلاصة	**234**
إجابات التدريبات	**235**
مسرد المصطلحات	**237**
المراجع	**238**

المقدمة

تمهيد

عزيزي الدارس،

مرحبا بك إلى هذه الوحدة، الوحدة السابعة

تتحدث هذه الوحدة عن المخططات وكيفية استخدامها لتمثيل العلاقات الهرمية بين البيانات، وهي تتألف من ممرات تنقسم إلى ممرات موجهّة وممرات غير موجّهة، وسنوضّح طرق تمثيل المخططات وذلك باستخدام مصفوفة التجاور أو القوائم المتصلة.

أتمنى عزيزي الدارس أن تتعامل مع هذه الوحدة بجدية، لما لها من دور كبير في تعزيز ما ستدرسه من مفاهيم.

أهداف الوحدة



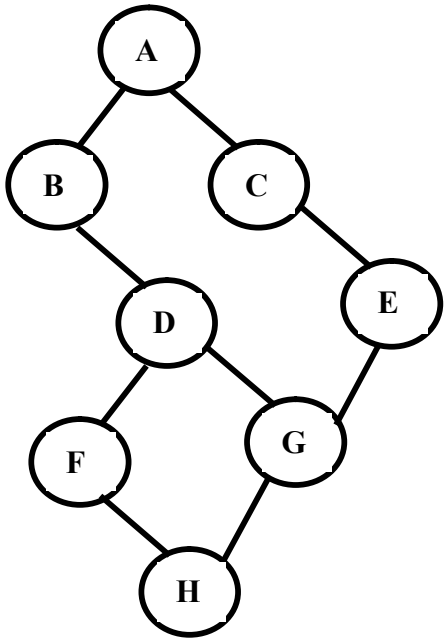
عزيزي الدارس، بعد فراغك من هذه الوحدة ينبغي أن تكون قادرا على أن:

1. **تعرّف** المخططات.
2. **تُميز** بين المخططات والأشجار.
3. **تفرّق** بين المخطط الموجّه والمخطط الغير موجّه.
4. **تُعمم** المخططات بطريقتي مصفوفة التجاور والقوائم المتصلة.
5. **تفرّق** بين المخطط المتصل والمخطط غير المتصل.

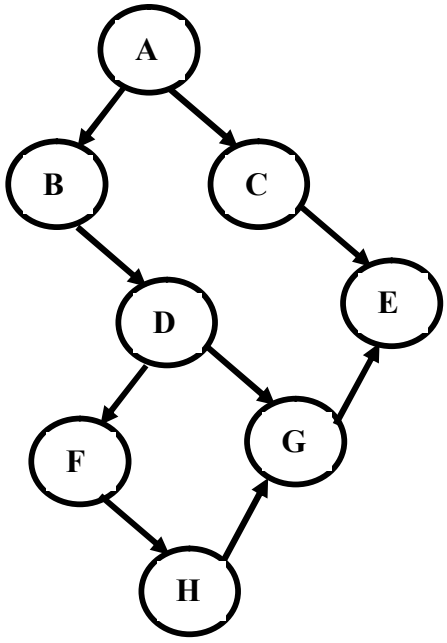
1. المخططات

المخططات هي صورة موسّعة للأشجار وذلك من خلال قلة القيود المفروضة على الأشجار وهي تستخدم لتمثيل بعض العلاقات الهرمية بين البيانات وتتألف المخططات من مجموعة عقد تربطها ممرات أو بمعني آخر هي مجموعة من العقد في شكل زوجي متصلة فيما بينها بواسطة(قوس).

يمكن أن تكون هذه الخطوط (الممرات) موجهّة أو غير ذلك كما يوضح ذلك الشكل رقم **(1)** والشكل رقم **(2)**.



شكل رقم (1): مخطط غير موجّه



الشكل رقم (2): مخطط موجّه

في الشكل رقم **(2)** نجد أن العقدة **B** يتم الوصول إليها عن طريق العقدة **A** ولكن لا يمكن أن نصل إلي العقدة **A** عن طريق العقدة **B.**

كثير من التطبيقات في حياتنا اليومية تتعامل مع المخططات الموجهّة منها شبكة المياه والتلفونات والبترول.

1.1 تعريفات

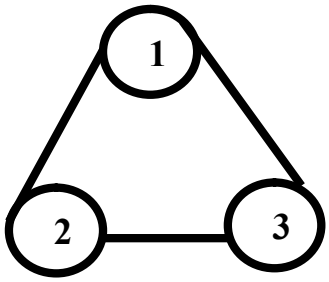
لتحديد العلاقة بين العناصر المختلفة التي تكون المخططات نلجأ إلي استخدام مصطلحات محددة لابد لنا من تعريفها وهي:

أولاً: المخطط Graph

هو عبارة عن زوج مرتب مكون من **(V,E)** حيث **V** تمثل مجموعة من العقد أو العناصر وتسمي أيضاً الرؤوس **(Vertices)** و**E** تمثل مجموعة الممرات أو الأقواس التي تصل بين الرؤوس وتسمى أيضاً الحواف **(Edges)** والقوس الواحد يمثل زوجاً من الرؤوس وعلي هذا الأساس يشار إلي المخطط **(G)** كما يلي:

G =(V,E)

فمثلاً الشكل التالي يمثل مخطط

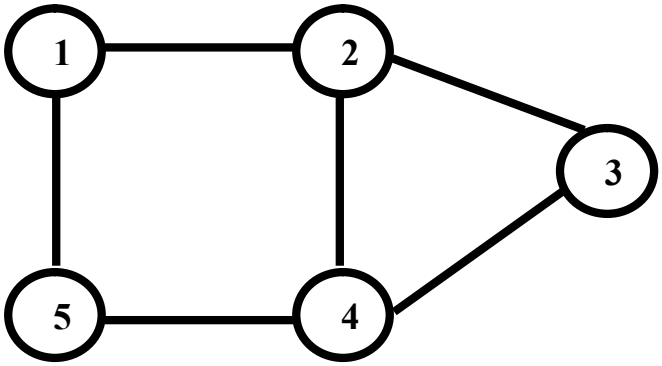


حيث

$$V=\{1,2,3\}$$
$$E=\{(1,2),(1,3),(2,3)\}$$
$$G=(V,E)=(\{1,2,3\},\{(1,2),(1,3),(2,3)\})$$

ثانياً: المسارات

هو متوالية من الأقواس بحيث يكون المكون الثاني من القوس السابق مساوياً للمكون الأول من القوس اللاحق فمثلاً في الشكل التالي:



حيث

$$V=\{1,2,3,4,5\}$$
$$E=\{(1,2),(2,3),(1,5),(3,4),(5,4),(2,4)\}$$
$$G=(V,\{(1,2),(2,3),(1,5),(3,4),(5,4),(2,4)\})$$

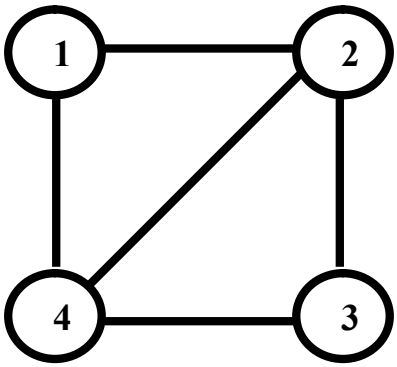
نجد أن المتوالية التالية من الأقواس

(1,2) , (2,3) , (3,4) , (4,5)

تمثل مساراً.

ثالثاً: الـــدورة

هي مسار يبدأ برأس وينتهي بنفس الرأس بشرط عدم تكرار أي قوس فمثلاً في الشكل التالي:



حيث

V={1,2,3,4}
E={(1,2) , (2,3) , (2,4) , (3,4) , (4,2)}
G=({1,2,3,4},{(1,2) , (2,3) , (2,4) , (3,4) , (4,2)})

نجد

1. المسار **(4,1) , (2,4) , (1,2)** يمثل دورة .
2. المسار **(4,2) , (3,4) , (2,3)** يمثل دورة .
3. المسار **(2,1) , (3,2) , (4,3) , (2,4) , (1,2)** لا يمثل دورة لان القوس **(2,1)** هو نفس القوس **(1,2)** وذلك لان المخطط غير موجه.

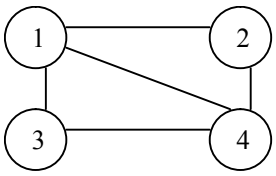
رابعاً:المسار البسيط

هو المسار الذي لا يحتوى على أي دورة جزئية فمثلاً المسار **(3,2) , (4,3) , (2,4) , (1,2)** مسار غير بسيط لأن هذا المسار يحتوي على دورة جزئية وهى **(3,2) , (4,3) , (2,4)** بينما المسار

(4,1) , (3,4) , (2,3) , (1,2) هو مسار بسيط وهو يشكل دورة ولكنها ليست جزئية.

تدريب (1)

استخرج من الشكل التالي ما يلي:

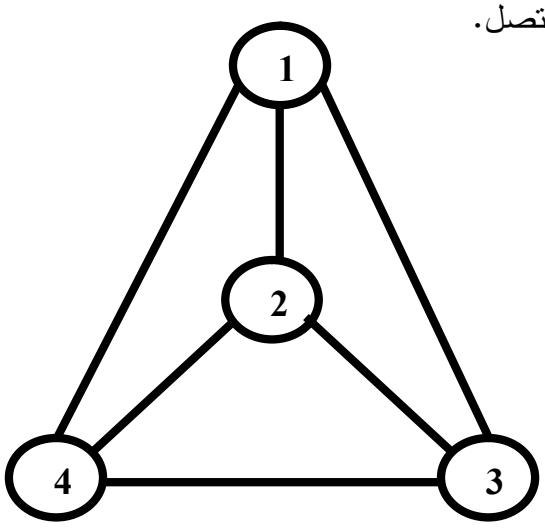


1. مساراً لا يشكّل دورة.
2. مساراً غير بسيط
3. حدّد العقد ، والممرات ثم عوّض في المعادلة التالية:

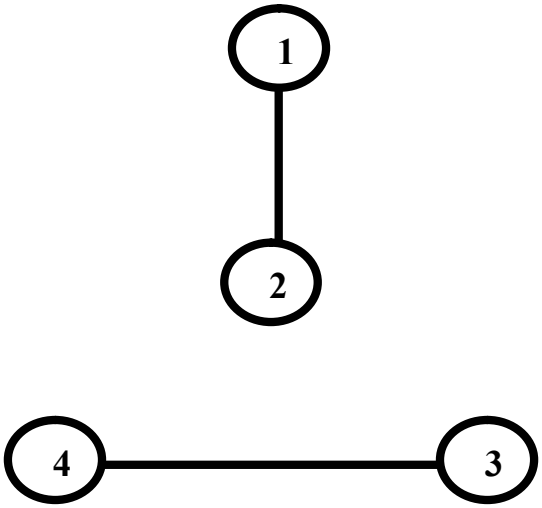
$$G = (V, E)$$

خامساً: المخطط المتصل

إذا وجد لكل رأسين **في** المخطط قوس **يربطهما** فإن المخطط يسمى مخططاً قوي الترابط، **أو** مخططاً متصلاً كما يوضح الشكل رقم **(3)** مخططاً متصلاً بينما يوضح الشكل رقم **(4)** مخططاً غير متصل.



الشكل رقم (3) مخطط قوي الاتصال

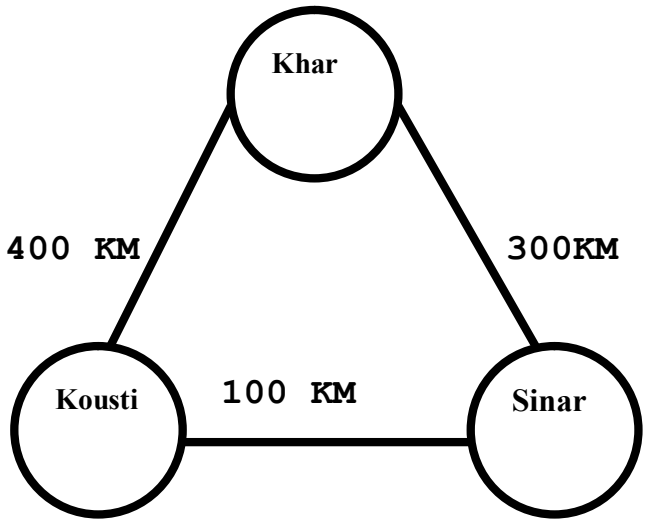


الشكل رقم (4) مخطط غير متصل

سادساً: الشبكة

هي مخطط بحيث يعطي لكل قوس قيمة عددية تمثل وزن القوس فمثلاً الشكل رقم (5) يمثل شبكة طرق برية والقيمة العددية للقوس تمثل طول المسافة بين المدينتين (العقدتين) مقدرة بالكيلومترات:

إن أهمية المخططات تكمن في أن تطبيقاتها الميدانية متعددة وعلي سبيل المثال وليس الحصر فإذا نظرنا إلي العلاقات الاجتماعية بين مجموعة من الأشخاص على إنها شبكة أو مخططاً (رؤوس المخطط هي الأفراد والأقواس درجة القرابة أو الصلة بين الأفراد وهنالك أمثلة عديدة مثل مخطط رحلات الطيران وشبكات الدوائر الكهربية وشبكات الاتصال الهاتفي وغيرها من الشبكات.



الشكل رقم (5):شبكة طرق برية

2.1 المقارنة بين المخططات والأشجار

هنالك ثم اختلافات بين الأشجار والمخططات تصلح لعمل المقارنات بينها ومن هذه الاختلافات الأتي:

أولاً:

تتم زيارة العقد في الأشجار من أعلى إلى أسفل بينما نجد في المخططات أن الزيارة تتم من أي عقدة.

ثانياً:

مؤشرات الربط بين العقد في الأشجار تكون راسية فقط أي من أعلى إلى أسفل بمعني تتجه من الأب إلي الابن بينما مؤشرات الربط بين العقد في المخططات تكون راسية أو أفقية أو الاثنين معاً.

ثالثاً:

يجب أن تحتوي الأشجار علي عقدة تسمى الجذر أما المخططات فلا يجب أن تحتوي علي هذه العقدة.

رابعاً:

يمكن أن نتحرك حركة دائرية في المخططات ولكن لا نستطيع القيام بذلك في حالة الأشجار

خامساً:

العقد قد تحمل قيمة رقمية أو حرفية وأن الخط الرابط بينها لا يحمل أي قيمة أما في المخططات فإن العقد قد تحمل قيمة رقمية أو حرفية أو علامة وأن الخط الرابط بينها يحمل أيضا قيمة.

مما سبق يمكن أن نقول إن جميع الأشجار سواءً كانت ثنائية أم غير ذلك هي عبارة عن مخططات (ليس بالضرورة أن يكون العكس صحيحاً) أو حالة خاصة من المخططات.

أسئلة تقويم ذاتي



1. عرّف المخططات.
2. تنقسم الخطوط (الممرات) بين العقد إلى قسمين، ما هما؟
3. قارن بين المخططات والأشجار لبيان الاختلافات.
4. عرّف الأتي:

أ- المخطط. ب- المسارات. ج- الدورة. د- المسار البسيط.

هـ- المخطط المتصل. و- الشبكة.

2. طرق تمثيل المخططات

هنالك طريقتان لتمثيل المخططات هما:

- استخدام مصفوفة التجاور.
- استخدام القوائم المتصلة.

1.2 استخدام مصفوفة التجاور

1	2	3	4	5	
0	1	0	0	1	1
1	0	1	1	0	2
0	1	0	1	0	3
0	1	1	0	1	4
1	0	0	1	0	5

(ن × ن)حيث ن عدد صحيح يساوي عدد العقد في المخطط علي النحو التالي:

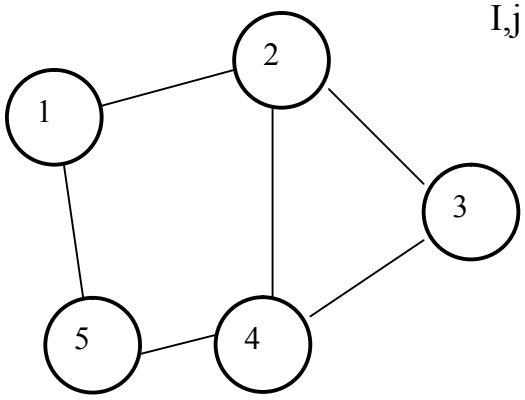
يمثل المخطط بمصفوفة من الرتبة ن، رقم رؤوس المخطط من 1 إلى ن

القوس من الرأس I إلى الرأس j يمثل بالعنصر Aij حيث

Aij=1 إذا وجد قوس بين I,j

Aij=0 إذا لم يوجد قوس بين I,j

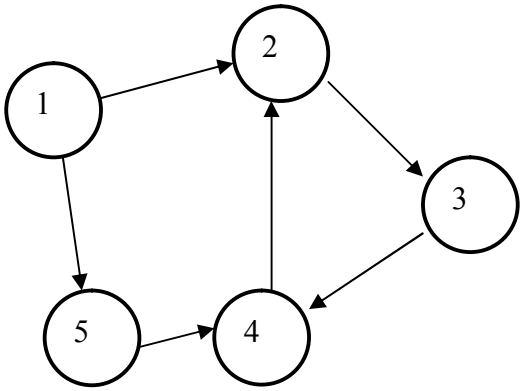
فمثلا المخطط التالي --



يمكن تمثيله بمصفوفة التجاور A1علي النحو التالي:

5 X 5 مصفوفة من المرتبة 5

أما إذا أخذنا المخطط الموجه التالي



1	2	3	4	5	
0	1	0	0	1	1
1	0	1	1	0	2
0	1	0	1	0	3
0	1	1	0	1	4
1	0	0	1	0	5

فإن مصفوفة التجاور هي:

1	2	3	4	5	
0	1	0	0	1	1
0	0	1	0	0	2
0	0	0	1	0	3
0	1	0	0	1	4
0	0	0	1	0	5

لا حظ أن A1 لا تساوي A2 لأن المخطط الأول الذي تمثله المصفوفة A1 هو مخطط غير موجه بينما المخطط الثاني الذي تمثله المصفوفة A2هو مخطط موجه

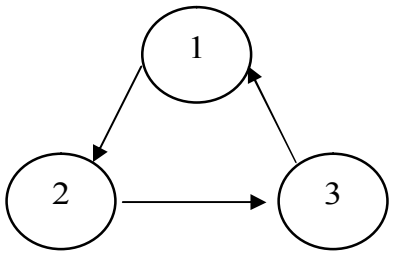
وظيفة مصفوفة التجاور:

إن مصفوفة التجاور تحدد وجود مسارات بين رؤوس المخطط وكذلك تحدد أطوال هذه المسارات، فطول المسار من الرأس I إلى الرأس J يساوي عدد الرؤوس التي مر فيها هذا المسار ناقصا واحداً صحيحاً.

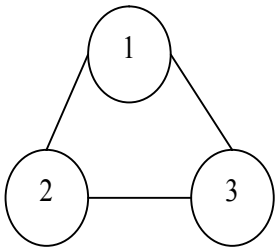
تدريب (2)

استخدم مصفوفة التجاور لتمثيل الأشكال التالية:

أ-



ب-



2.2 استخدام القوائم المتصلة

يمثل المخطط الموجه بواسطة القوائم المتصلة من خلال قائمتين:

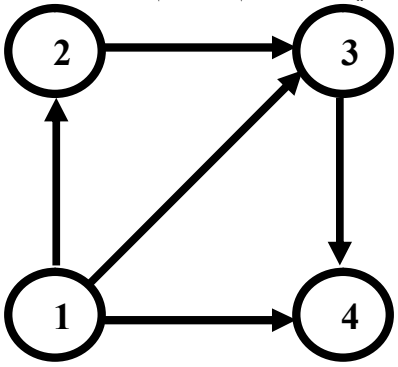
القائمة الأولي:

هي قائمة العقد حيث يكون لكل عقدة ثلاثة حقول هي الحقل الأول للمعلومات **Info** والحقل الثاني لربط هذه العقدة بالعقدة التالية**Next** أما الحقل الثالث فهو عبارة عن مؤشر يؤشر إلي اتجاه القوس الخارج من هذا العقدة**Adj**.

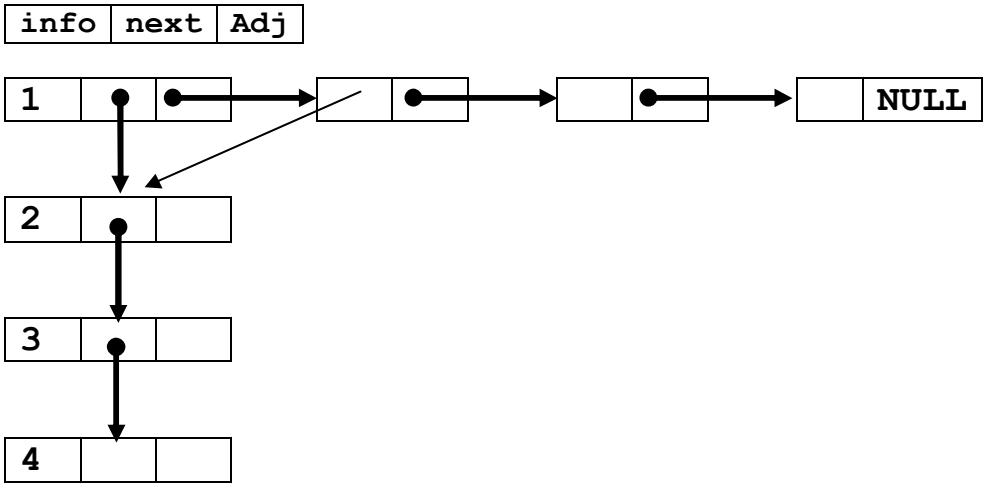
Info	Next	Adj

مثال (1)

مثل المخطط التالي باستخدام القوائم المتصلة



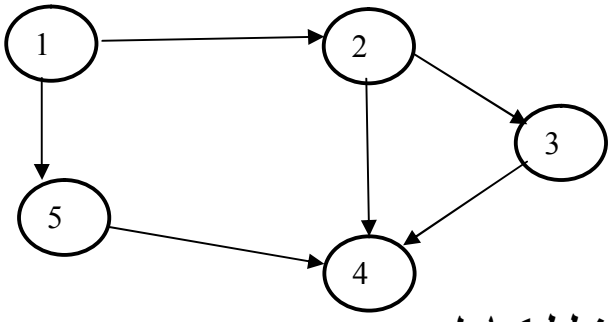
يمكن وصف هذا المخطط كما يلي:



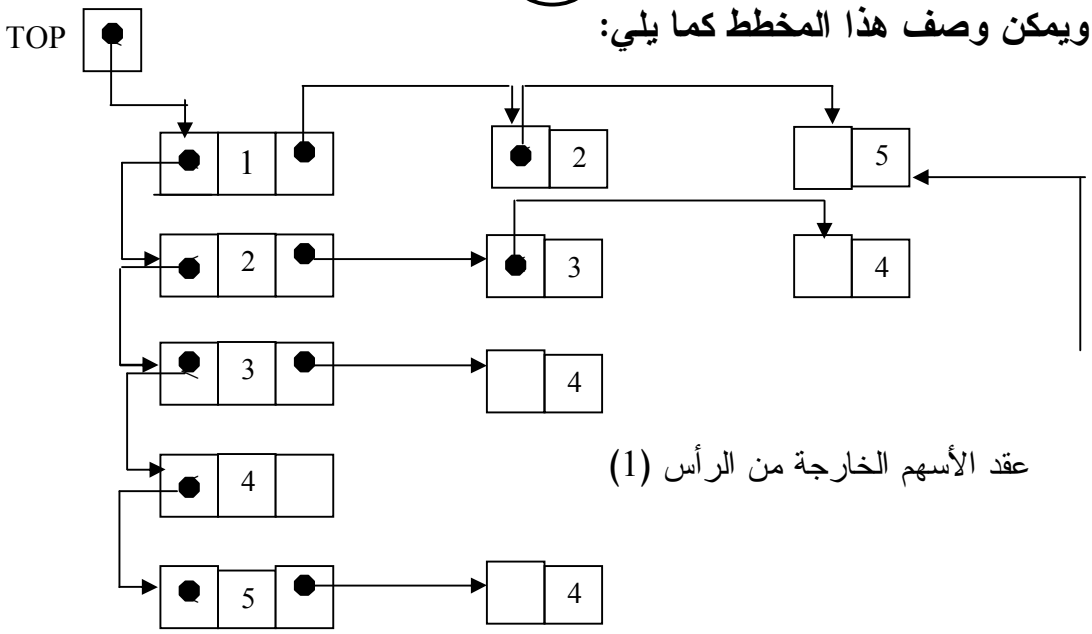
القائمة الثانية:

هي قائمة الأقواس حيث يكون لكل قوس حقلان هما: الحقل الأول عبارة عن حقل المعلومات، وحقل القوس التالي.

والمثال التالي يوضح ذلك :

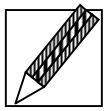


ويمكن وصف هذا المخطط كما يلي:

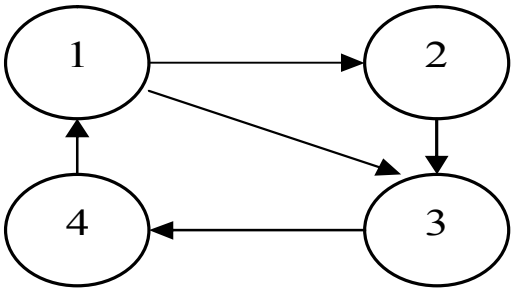


عقد الأسهم الخارجة من الرأس (1)

تدريب (3)



مثّل المخطط التالي باستخدام القوائم المتصلة:



محاور المقارنة	الأشجار	المخططات
الزيادة في العقد	تتم من أعلى إلى أسفل	تتم من أي عقدة
مؤشرات الربط بين العقد	تكون رأسية فقط، أي من أعلى إلى أسفل بمعنى أنها تتجه من الأب إلى الابن	تكون رأسية أو أفقية أو الاثنين معاً
المحتويات	يجب أن تحتوي على عقدة تسمى الجذر	لا تحتوي على عقدة الجذر
الحركة	لا تستطيع القيام بالحركة الدائرية	تتحرك حركة دائرية
العقد والخط الرابط بينها	العقد قد تحمل قيمة رقمية أو حرفية والخط الرابط بينها لا يحمل أي قيمة	العقد قد تحمل قيمة رقمية أو حرفية أو علامة والخط الرابط بينها يحمل أيضاً قيمة

جدول (1) يوضح محاور المقارنة بين الأشجار والمخططات

الخلاصة

عزيزي الدارس، نلخّص وحدتنا السابعة في أننا تعرّفنا على ملامح المخططات الأساسية، واستخدامها لتمثيل العلاقات الهرمية بين البيانات، وقلنا أنها تتألف من مجموعة عقد تربطها ممرات. ويمكن أن تكون هذه الممرات موجهّة أو غير موجّهة.

في الجزء الثاني وضّحنا بأن هنالك اختلافات بين الأشجار والمخططات وذلك من خلال المقارنة بينها. وقلنا إن الأشجار هي حالة خاصة من المخططات.

وفي الجزء الثالث عرّفنا العناصر التي تكوّن المخططات وذلك لتحديد العلاقة بينها. وقلنا ان المخطط عبارة عن زوج مرتّب مكوّن من مجموعة من العقد ومجموعة الممرات.

والمسارات عبارة عن متوالية من الأقواس، والدورة عبارة عن مسار يبدأ برأس وينتهي بنفس الرأس بشرط عدم تكرار أي قوس، وعرّفنا المسار البسيط بأنه لا يحتوي على أي دورة جزئية.

في الجزء الرابع وضّحنا طرق تمثيل المخططات وذلك بإستخدام مصفوفة التجاور أو القوائم المتصلة وقلنا إن مصفوفة التجاور تحدد وجود مسارات بين رؤوس المخطط وكذلك تحدد أطوال هذه المسارات. وتحدثنا عن استخدام القوائم المتصلة وقلنا بأنه يمكن تمثيل المخطط من خلال قائمتين هما قائمة العقد ولها ثلاثة حقول وهي حقل المعلومات، وحقل الربط بالعقدة التالية، وحقل المؤشر حيث يؤشر للقوس أو الممر الخارج من هذه العقدة.

والقائمة الثانية هي قائمة الأقواس حيث يكون لكل قوس حقلان هما: حقل المعلومات وحقل القوس التالي.

أتمنى عزيزي الدارس أن نكون قد وفقنا في تقديم هذه الوحدة.

إجابات التدريبات

تدريب (1)

أ- مسار لا يشكل دورة هو المسار:

) هو نفس القوس (1. 4 (1,4) لان القوس (1,4),(2,1),(3,2),(1,3),(4,1)

ب- مسار غير بسيط:

(4,1),(1,3),(3,2),(2,1) لان هذا المسار يحتوي على دورة جزئية وهي:
(1,3),(3,2),(2,1)

ج- V = {1,2,3,4}

E = { (1,2),(1,4),(1,3),(2,3),(3,4) }

G = ({1,2,3,4},{(1,2),(1,4),(1,3),(2,3),(2,4)})

تدريب (2)

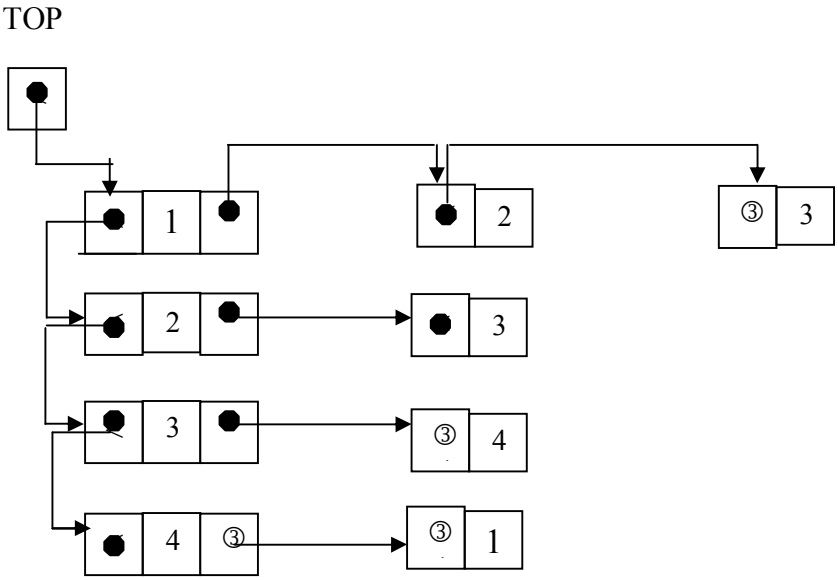
أ-

	1	2	3
1	0	1	1
2	1	0	1
3	1	1	0

ب-

	1	2	3
1	0	1	0
2	0	0	1
3	1	0	0

تدريب (3)



مسرد المصطلحات

المخطط Graph

هو عبارة زوج مرتّب مكوّن من (V,E) حيث V تمثل مجموعة من العقد أو العناصر وتسمى أيضاً بالرؤوس، E تمثّل مجموعة الممرات أو الأقواس التي تصل بين الرؤوس، والقوس الواحد يمثّل زوجاً من الرؤوس.

المسارات Paths

هي عبارة عن متوالية من الأقواس بحيث يكون المكوّن الثاني من القوس السابق مساوياً للمكوّن الأول من القوس اللاحق.

الدورة Cycle

هو عبارة عن مسار يبدأ برأس وينتهي بنفس الرأس بشرط عدم تكرار أي قوس.

المسار البسيط Simple Path

هو المسار الذي لا يحتوي على أي دورة جزئية.

المخطط المتصّل

إذا وجد لكل رأسين في المخطط قوس يربطهما يكون المخطط مخططاً متّصلاً.

الشبكة Network

هي مخطط يعطي لكل قوس قيمة عددية تمثل وزن القوس.

المراجع

1. William Ford and William Topp, **Data Structure with C++**, Prentice-Hall Inc 1996.
2. Sara Baase ad Allen Van Gelder, **Computer algorithms**, Addison Wesley Longman 2000.
3. Deitel and Deital, **C++ How to program**, Prentice-Hall Inc 2001.