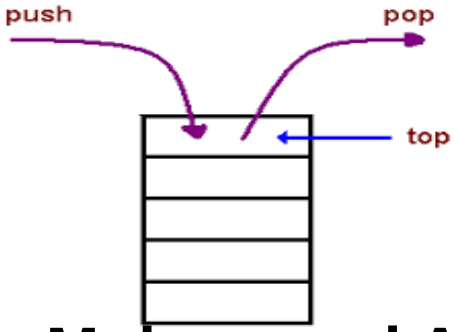
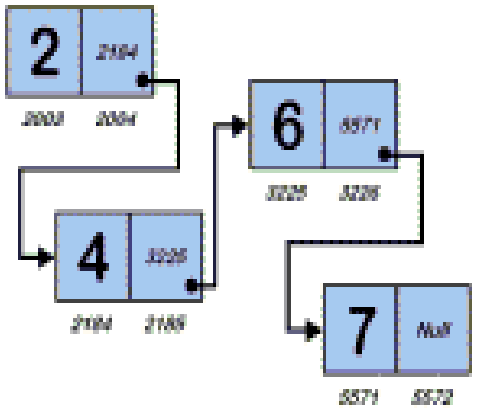
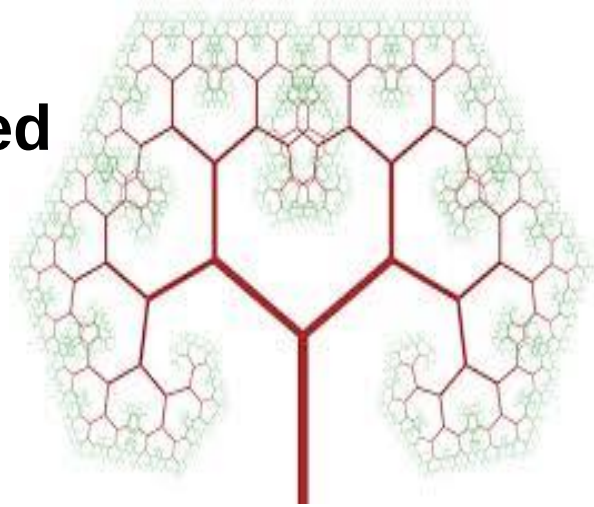
جامعة حماة
الكلية التطبيقية
قسم تقنيات الحاسوب
الفصل الثاني



- المقرر: الخوارزميات وبنى المعطيات
- المحاضرة :السادسة

Dr. Mohammed AL-Mohammed





Chapter 6

بنى المعطيات

Data structure

بنى المعطيات (البيانات) Data structure

بيانات: هي توصيف للبيئة الأساسية للمشكلة ومعالجتها (صور، فيديو، ...)

سنعرف ما يسمى بـــ:

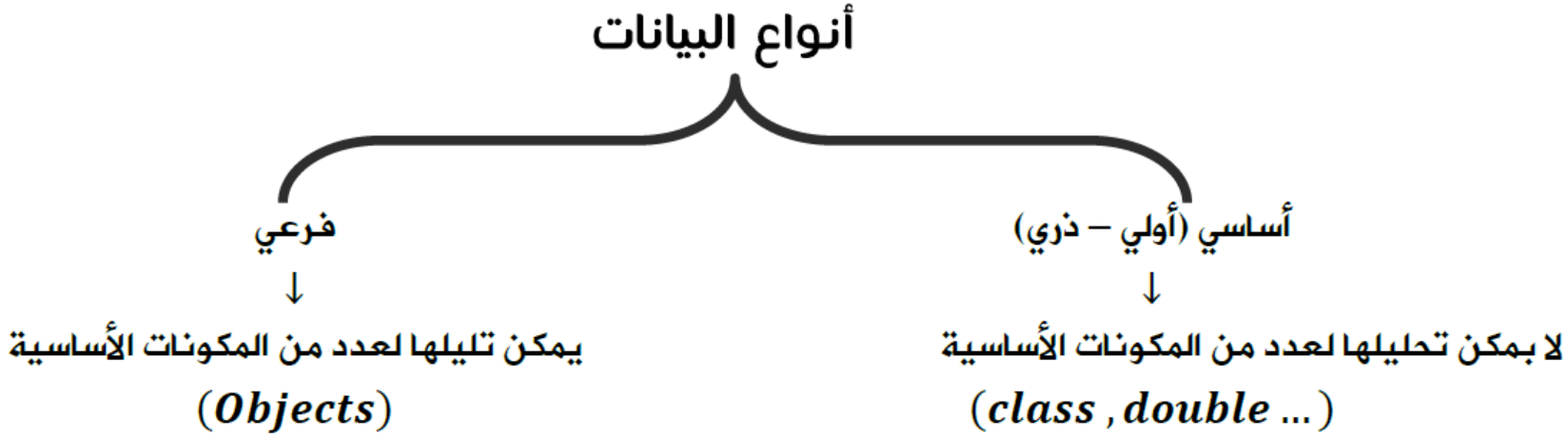
نوع البيانات المجردة (*Abstract Data type*).

التجريد: هو إخفاء التفاصيل ذات المستوى الأدنى والاهتمام بالتفاصيل ذات المستوى الأعلى.

في البرمجة:

من أشكال التفاصيل ذات المستوى الأدنى: مكان تخزين عنصر ما في الذاكرة وكيفية تخزينه.

أما التفاصيل ذات المستوى الأعلى فمن أشكالها: حجز مكان في الذاكرة لمصفوفة ما وما هو حجمها (عدد العناصر المحجوزة لها).



- ADT: مجموعة من القيم values ومجموعة من العمليات operations المعرفة على هذه القيم.

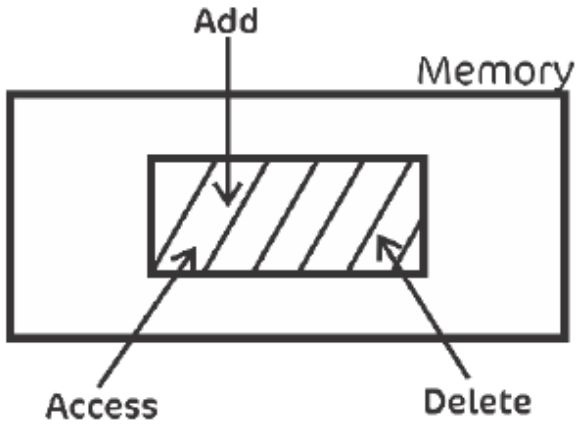
مثال : نوع المعطيات Integer مع العمليات +, -, *, /, mod, abs, max,...

- لاحظ أنه يمكن اعتبار الأغراضobject مثل اللوائح والمجموعات، والبيانات أنواع معطيات مجردة لها مجموعة من العمليات المعرفة لكل منها.
- الفكرة الأساسية هي أنها تنجَّز مرة واحدة في برنامج وتستخدم ضمن أي برنامج آخر
- في c++ أو Java يمكن استخدام الصف class لتعريف نوع معطيات مجرد جديد.
- لا يعرف مستخدم نوعِ معطياتٍ مجرد ما كيف نُجِّزت مجموعة العمليات الخاصة بنوع المعطيات المجرد هذا.

بإمكاننا أن نوصّف أي ***object*** في الطبيعة على أنّه مجموعة من البيانات المتشابهة.

بنى البيانات: طريقة لتنظيم البيانات في الذاكرة ***RAM*** وبشكل مجرّد عن طريق التنفيذ وطريقة بنائها وترابطها مع بعضها البعض.

ADT: هو توظيف لنوع البيانات بشكل مجرّد (مستقل عن التنفيذ والمصادر المستخدمة – أي ماذا تعمل هذه الخوارزمية وليس كيف تعمل) وأساسي وغير أساسي، وتوصيف للعمليات الأساسيّة على هذه البيانات بشكل مجرّد عن التنفيذ أيضاً.



ما هي هذه العمليات الأساسيّة؟

1) ***Access*** طريقة الوصول.
2) ***Add*** الإضافة.
3) ***Delete*** الحذف.

سوف نربط البيانات مع بعضها عبر بنية بيانات لكي نستطيع التّحكم بالعمليات الأساسيّة عليها.

طريقة تشييد (بناء) بنية بيانات

a) طريقة هيكلة البيانات أو الأغراض وارتباطها (المنطقي) فيما بينها.

مثال: نريد أن نربط بين بنية أوليّة int وبين مصفوفة ما عدد عناصر هذه المصفوفة، كما نريد تخزينها في الذاكرة

$$\Rightarrow int\ A[15];$$

لذلك بمعرفة عنصر واحد فقط $A[i]$ نستطيع أن نعرف عدة عناصر مثل $A[i+1]$ و $A[i-1]$ لأنها مربوطة منطقياً.

b) تحديد (توصيف) مجموعة متكاملة من العمليات الأساسية التي ننجزها على هذه الهيكلية.

التعريف: هي طريقة التنفيذ الفيزيائي لنوع البيانات المجردة (طريقة تنظيم البيانات في الذاكرة).

بنى البيانات المشهورة

1) المصفوفات $Arrays$.
2) القوائم المرتبطة (اللوائح) $Linked\ Lists$.
3) المكدسات $stacks$.
4) الطوابير $Queues$. (الأرتال)
5) الجداول $Tables$.
6) الشجرات $Trees$.
7) البيان (المخططات) $Graphes$.

بنى المعطيات الخطية

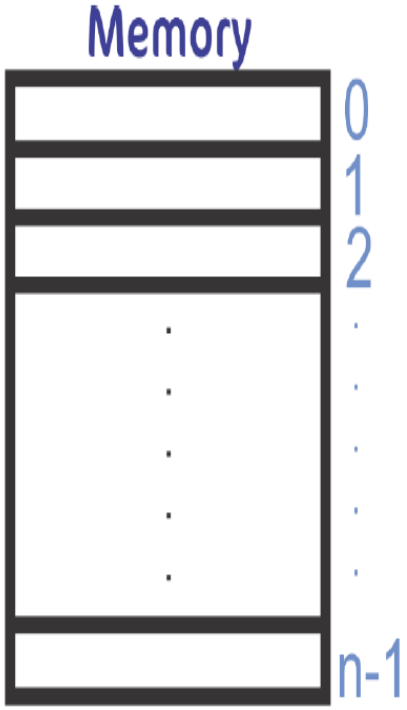
بنى المعطيات غير الخطية

ملاحظات:

1. إن اختيار بنية البيانات المناسبة يزيد من فعاليّة البرنامج وهذه هي النقطة الأساسيّة في دراسة الخوارزميّة.
2. العمليات الأساسية على البنية هي:

إضافة عنصر، حذف عنصر، الوصول لعنصر بالإضافة إلى بعض العمليات الأخرى الخاصة بكل بنية.

مثال: يجب أن نقوم بعملية فحص للـ ***linked list*** لمعرفة فيما إذا كانت ***empty*** أساساً بينما نحن نريد القيام بعملية حذف.



3. التخصيص الديناميكي للذاكرة ***Dynamic allocation Memory***:

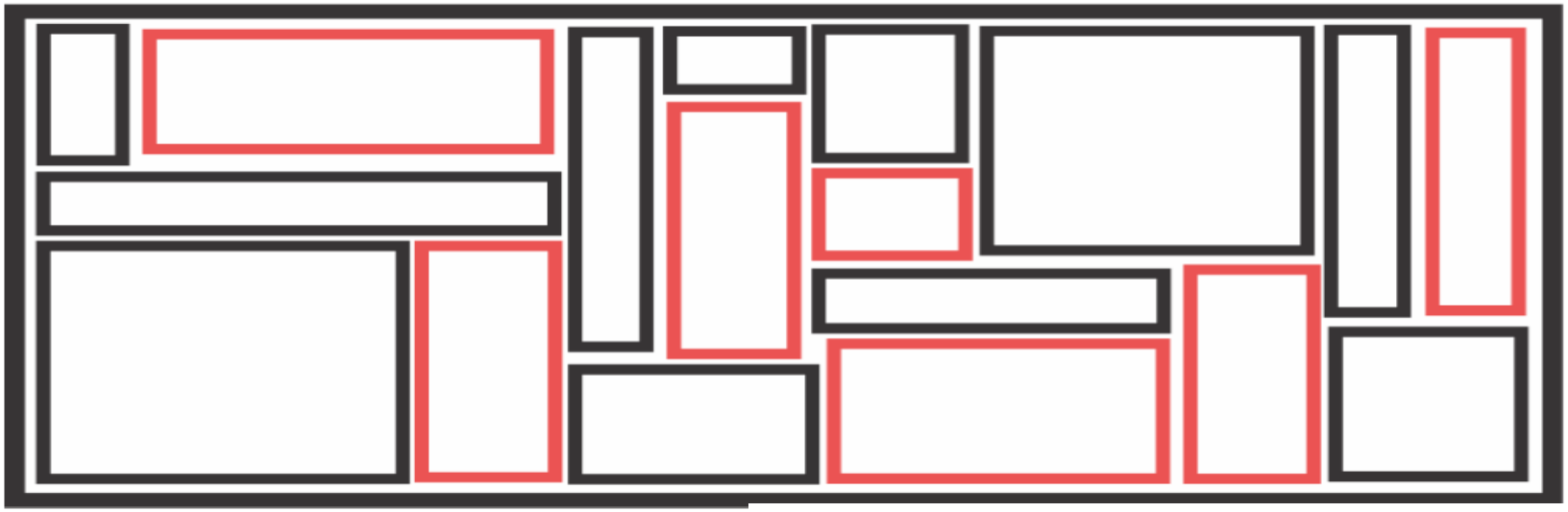
هو الحصول على الذاكرة وتحريرها أثناء التنفيذ.

حيث n هو حجم الذاكرة بواحدة ***Byte*** حيث افترضنا أن الذاكرة مقسّمة إلى 8 بايت لكل جزء وكل بايت سنسميها ***word*** (كلمة).

ملاحظة: تتميز المؤشرات بأنها تصل فوراً إلى قيم هذه العناوين وتسهل البرنامج فيصبح ممتازاً في الأداء، ولكن مشكلتها أن الخطأ فيها مهما كان صغيراً فهو خطأ قاتل ... أي أن بإمكان هذا الخطأ أن يؤدي إلى تدمير ملفات النظام وحذفها أو تدمير ملفات مهمّة جداً (في حواسيب بنك على سبيل المثال).

ما الذي يتم في الذاكرة؟

مثال: أنا الآن أنفذ برنامجاً بلغة $c++$ على حاسوبي وأستمع إلى أغنية ما عبر مشغل الموسيقى وأتصفح الفيسبوك وأقوم بتنزيل صورة ما وألعب لعبة في وقت متزامن ⇐ هنالك بيانات تدخل إلى الذاكرة وأخرى تخرج منها بشكل متعاقب شكلها مثل الفسيفساء وتتألف من قطع ($Blooks$) على الشكل الآتي:



حيث:

المناطق الفارغة.

المناطق المليئة (المشغولة) في الذاكرة.

- إن الذاكرة الكليّة هي $2\ GB$ والفارغ منها قرابة $1\ GB$ وعندما يطلب برنامج ما من هذه الذاكرة $32\ byte$ فقط قد لا يجد كتلة كاملة فارغة بمقدار $32\ byte$ وهذه هي أكبر مشكلة تتعرض لها الذاكرة.

- إن قوة الـ $Operating\ System$ هي قوة الخوارزمية التي تستطيع أن تنظم وترتب المعلومات في الذاكرة مع استغلال لأكبر قدر من المساحة فيها.

مثال:

لدينا ملف $word$ موجود في عدة أجزاء من الذاكرة ولكننا لانشعر بهذا البعد بين الأجزاء فهي تتجمع عبر بنية معطيات ما ترتب أجزاءها وكلما كانت هذه البنية أفضل $\Leftarrow$ التنفيذ يكون أسرع.

إن المشكلة السابقة تسمى بـــالتشكيل الحالي وحلّها مستحيل ولكن يمكن التقليل منه قدر الإمكان.

بنية معطيات المصفوفة

إن المصفوفة مقيّدة بعدة قيود أهمها:

1- الحجم: يجب أن يكون الحجم معروفاً أثناء التجريب والاختبار ***Testing***.

2- أن تكون البيانات المخزّنة من نفس النوع.

3- تخزين البيانات بشكل متتالي ممّا ينشئ لدينا مشكلات عديدة أهمها:

- التشكيل الحالي
- الإضافة والحذف: لأن مكان المصفوفة محدد ومحجوز

وبالتالي لايمكن إضافة عناصر جديدة ولا حذف عناصر حاليّة.

ملاحظات:

- ✓ إن القيم التي تخزّن في الذاكرة غالباً ما تبقى مالم يقم برنامج آخر بمسحها (الكتابة فوقها)
  ⇐ يوجد العديد من البرامج لاستعادة الذاكرة و المحذوفات عبر هذه الطريقة.
- ✓ إن العامل ***new*** يمكن أن يتحكم بالقيد رقم **1** ويخفف منه لأنه يعبر عن حجز مصفوفة ديناميكية الحجم (يمكن تغيير حجمها أثناء التنضيد ولمرة واحدة هي بداية (التنضيد).
- ✓ إن الرابط المنطقي في المصفوفة هو ***index*** الذي نستطيع من خلاله الوصول إلى عناصر المصفوفة.
- ✓ لا نستطيع الإضافة ***Add*** إلا بتبديل عنصر بعنصر أو عبر العامل ***new*** أما الحذف فهو تحرير فضاء الذاكرة
  ⇐ لا نستطيع فعله إلا بجعل قيمة العنصر افتراضيّة أو خياليّة.

اللوائح Lists

Data Struct

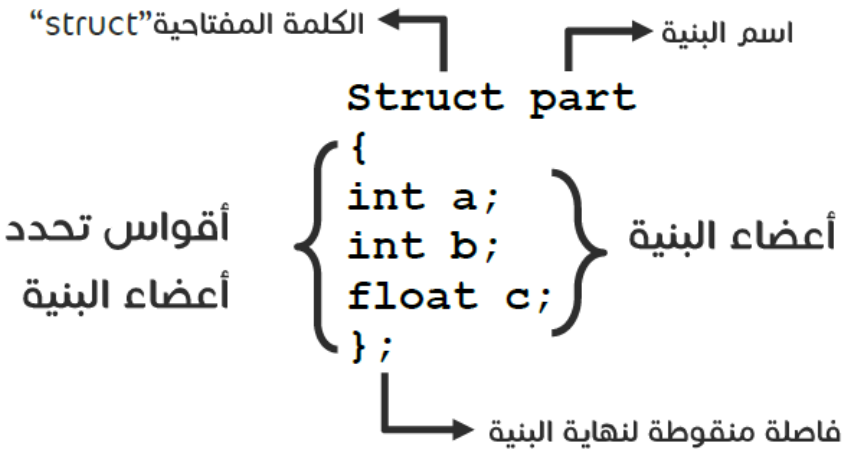
لنفرض أنه لدينا موظفين في دائرة ونريد القيام بقاعدة معطيات تخزن فيها المعلومات (عناوين،أسماء،وأرقام هواتف)

الطريقة الأولى: على شكل ثلاث مصفوفات.

الطريقة الثانية: مصفوفة واحدة من نوع ***Struct*** مؤلفة من ثلاثة حقول.

البنية *Struct*:

هي مصفوفة يمكن أن تحتوي على عناصر من أنواع مختلفة. يتم تعريفها كتعريف الصفوف:



تعليمة *typedef*

الدالة المحجوزة ***typedef*** هي دالة تقوم بالإعلان عن متغير فالمتغير يقوم بالإعلان عن متغير آخر.

```
#include<iostream>
 Typedef int think;
 Int main(){
 think a,b,c; }
```

قمنا بالإعلان عن 3 متغيرات بالمتغير ***think*** وهو من نوع ***int*** الذي قمنا بالإعلان عنه بواسطة ***typedef*** ونفس الطريقة في التعامل مع بنى المعطيات.

```
#include <iostream>
 typedef struct ADDR {
int x, y; }
int main() {
struct ADDR Str1 S1_1, S1_2;
struct ADDR Str2 S2_1, S2_2;}
```

ملاحظة: تعتمد اللوائح على البرمجة غرضية التوجه وذلك باستخدام الصفوف.

اللائحة ADT

نوع المعطيات المجرد الذي يوصف اللائحة.

اللائحة L تخزن نوع ما من المعطيات يسمى $ListElementType$.

العمليات التي تتم على اللائحة

تابع لإدخال حد أو للحصول على قيمة الحد الأول في اللائحة أو للحصول على الحد التالي في اللائحة

ويمكننا كتابة العديد من التوابع وتوصيفها ثم كتابتها بلغة $C++$.

```
void L.insert (ListElementType elem)
Precondition:None.
Postcondition:L_post= L_pre with an instance of elem added to L_post.
```

$L.insert$: الـ L هي اسم $object$ في الصف $List$ أي هو عبارة عن لائحة أما الـ $insert$ هو اسم التابع ويأخذ مدخلات هي $elem$ من النوع $ListElementType$.

$Precondition$: تعني شرط استدعاء التابع في الـ $main$ ولا يوجد أي شرط لذا كتبنا $None$.

$Postcondition$: تعني ما ينتج عن تنفيذ هذا التابع:

```
L_post = L_pre with an instance of elem added to Lpost.
```

تعني أن اللائحة L بعد تنفيذ التابع = اللائحة قبل تنفيذ الحدود مضافاً حد جديد.

فنستنتج أن وظيفة التابع السابق هو إضافة حد لحدود اللائحة.

نمط المعطيات المجرد "اللائحة List"

- اللائحة هي مجموعة منتهية مرتبة من عناصر لها نفس النوع مثل التتالي

a1, a2,... ,an

- لائحة ليس فيها عناصر هي لائحة خالية (معدومة) null.
- يمكن أن تكون مجموعة العمليات التي تجري على اللائحة append، Print_List، Find، Insert، Delete أو أية عملية أخرى متعلقة بالتطبيق.
- كيفية تنجيز اللائحة؟

- Static: arrays
- Dynamic: Linked lists

القضايا المتعلقة باللائحة

- استخدام الذاكرة Memory وحجم size اللائحة
- زمن استحضار retrieval عنصر ما في اللائحة.
- زمن إدراج insertion أو حذف Delete عنصر ما.
- الوصول Access إلى العناصر بترتيب محدد فقط.
- كم نحتاج إلى إضافة add أو حذف remove عناصر من اللائحة.

مثال عن المؤشرات

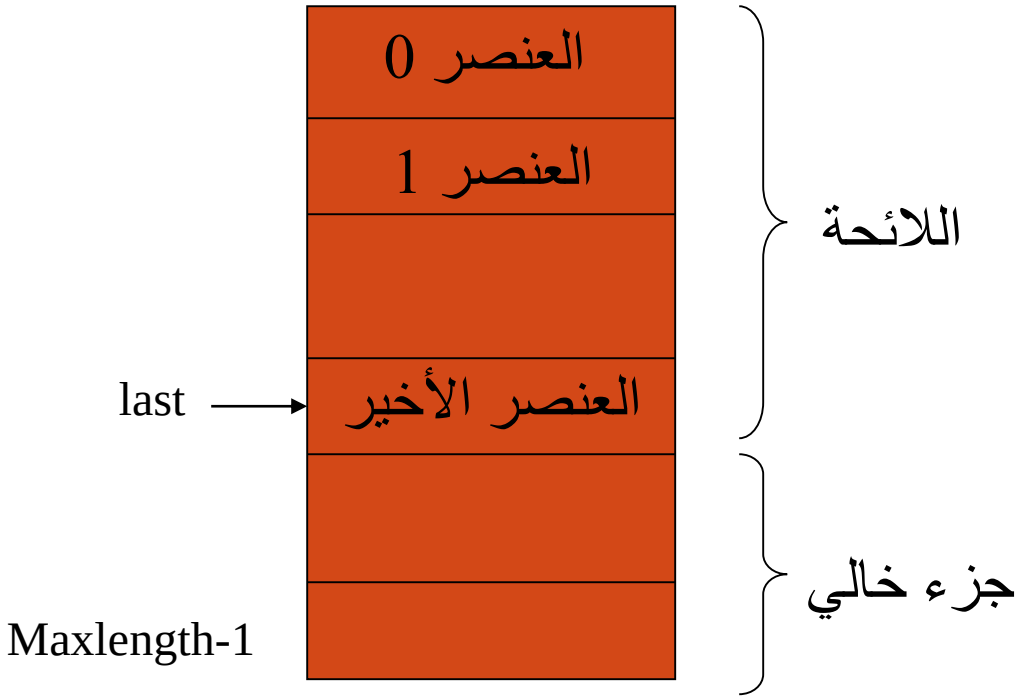
```
#include <iostream>
int main () {
    int * p;

     p = new  int;
      *p =3;
     int * q;
       q=p;
     cout << *p << endl;
      cout << *q << endl;
       *p = 2 * *p;
      cout << *p << endl;
     cout << *q << endl;
        return 0;}
```

Variables & Addresses	cout	p	1234	q
Types	ostream&	int*	int	int*
Values	3	1234	3	1234
	3		6	
	6			
	6			

التنجيز الساكن-المصفوفة

- أسهل طرق تنجيز لائحة هي باستخدام المصفوفة array :

```
Const int Maxlength=100;
Struct List
  {
   Type table [Maxlength];
   Int last;
   }
```



التنجيز الساكن-المصفوفة (تابع)

- لهذه الطريقة السمات التالية:
  - حد علوي لعدد العناصر المسموح بها في اللائحة.
  - بحث search سريع عن عنصر معين أو print-list (O(n) زمن خطي).
  - وصول سريع إلى العنصر رقم k (زمن ثابت O(1)).
  - الإدراج والحذف عمليتان مكلفتان (O(n) زمن خطي).
- نظراً لهذه القيود فلا تستخدم الصفيفة عادة لتنجيز اللائحة، وتستخدم فقط من أجل تطبيقات محددة.

```
bool L.first(ListElementType &elem)
Precondition: None
Postcondition: If the list is empty, none.  Otherwise, the
variable elem contains the first item in L; the "next" item to
be returned is the second in L.
Return:  true if and only if there is at least one element in L.
```

التابع السابق اسمه ***first*** يأخذ قيمة تمرر بالمرجع هي ***elem***.

ماينتج عن تنفيذ هذا التابع: إذا كانت اللائحة فارغة لا ينتج شيء ويعيد ***false*** "نلاحظ أن التابع من نمط ***bool*** أي يعيد ***true*** أو ***false***"

وإذا كانت اللائحة تحتوي على حدود يعيد ***true*** عندها يقوم بإعادة الحد الأول في اللائحة.

أما التابع ***next*** المبين تالياً الذي نستدعيه مباشرةً بعد ***first*** لذلك فإن شروط استدعائه في الـ ***main*** أي الـ ***Precondition*** هو أن يُستدعى تابع الـ ***first*** على الأقل مرة واحدة وينفذ هذا الحد طالما هناك حد تالي لذلك يُستدعى الـ ***next*** طالما يوجد هناك حدود في اللائحة والتابع ***next*** هو:

```
bool L.next(ListElementType & elem)
Precondition: The "first" operation has been called at least once.
Postcondition: Variable elem contains the next item in L, if
there is one, and the next
 counter advances by one; if there is no next element, none.
Return: true if and only if there is a next item.
```

مثلاً الطباعة والبحث:

```
Void L.print( )
Precondition: None.
Postcondition: print the items of the list, if the list L
condition items ,else none
```

أي يطبع حدود اللائحة إذا كانت تحتوي حدود وإلا فلا يطبع شيء.

```
int L.search( )
Precondition:none
Postcondition: Tests if list L contains items,searches for item T.
  if it in the list return its index,else return-1
Return: an integer value which is the index of the item T if it found
  else returns-1;
```

كيفية كتابة اللائحة بلغة $c++$ بشكل عام

نقوم أولاً بتعريف نوع من المعطيات عن طريق الكلمة المحجوزة في لغة $C++$: $typedef$
ونحدد نوعه سواء int أو $float$ أو $double$ إلخ

ثم نعرف صف $List$ وحددنا التوابع الأعضاء فيه أما في الـ $private$ فنضع المتغيرات التي تمكننا من إدارة اللائحة.

```
// the type of the individual elements in the list is defined
here
typedef int ListElementType;
// implementation specific stuff here
class List {
public:
   List();
   void insert(const ListElementType & elem);
   bool first(ListElementType & elem);
   bool next(ListElementType & elem);
private:
   // implementation specific stuff here
};
```

```
typedef    int    ListElementType;
const int maxListSize = 1000;
class List {
public:
   List();
   void insert(const ListElementType & elem);
   bool first(ListElementType & elem);
   bool next(ListElementType & elem);
private:
   ListElementType listArray[maxListSize];
   int numberOfElements;
   int currentPosition;
};

  List::List(){
     // initialize to an empty list
     numberOfElements = 0;
     currentPosition = -1;
  }
  void List::insert(const ListElementType & elem)
  {
     If(numberOfElements < maxListSize){
     listArray[numberOfElements] = elem;
     numberOfElements++;}else  cout<<"List is full";
  }
```

```
bool List::first(ListElementType & elem)  السطر 3
{
if (numberOfElements == 0)
return false;
else {
currentPosition = 0;
elem = listArray[currentPosition];
return true;
}
} bool List::next(ListElementType & elem)  السطر 4
{
if(currentPosition >= 0);
if (currentPosition >= numberOfElements - 1)
return false;
else {
currentPosition++;
elem = listArray[currentPosition];
return true;
} }
```

```
int main()  السطر 5
{
List L1,L2,L3;
ListElementType elem1; bool avail; cin>>elem1;
L1.insert(elem1); L2.insert(elem1); cin>>elem1;
L3.insert(elem1); L2.insert(elem1); L2.insert(elem1);
avail=L2.first(elem1); cout<<elem1; avail=
L2.next(elem1); cout<<elem1; avail= L2.next(elem1);
cout<<elem1; avail= L2.next(elem1); return 0;
}
```

نعرف أولاً متغير من النوع int هو $maxListSize$ وهو حجم المصفوفة.

السطر الأول:

نبدأ بالأعضاء الخاصة المعرفة ضمن الصف.

عرفنا أولاً مصفوفة هي $listArray$ لتخزين حدود اللائحة ضمنها وهي من نوع $ListElementType$ وحجمها $maxListSize$.

عرفنا المتغير $numberOfElements$ وهو عدد حدود اللائحة التي خزنت في المصفوفة فمثلاً بإمكاننا تخزين 50 حد فقط لكن المصفوفة تتتسع لـ 1000 حد.

أخيراً عرفنا المتغير $currentPosition$ الذي يدل على الحد الذي أتعامل معه حالياً وسيتضح دور هذه المتغيرات في تكملة البرنامج.

السطر الثاني:

بعد تعريف الباني الافتراضي وإعطاء قيم ابتدائية للأعضاء الخاصة قمنا بتعريف التابع العضو $insert$ "علماً أن طريقة التعريف المتبعة خارج الصف"

تابع $insert$ نستخدمه لتسجيل المتغيرات التي تدخلها ضمن المصفوفة أي تسجيل حدود اللائحة.

شرط تنفيذ هذا التابع أن يكون عدد الحدود المسجلة $numberOfElements$ أصغر من $maxListSize$ أي الشرط ألا يتجاوز عدد عناصر المصفوفة فعند تحقق الشرط تبدأ بإسناد قيم الـ $elem$ للمصفوفة.

السطر الثالث:

التابع $first$ كما ذكرنا سابقاً يستخدم لتحديد أول عنصر في المصفوفة وهو نفسه التابع المشروح في أول المحاضرة وهنا مبرمج بلغة $c++$ فيقوم بإسناد أول عنصر في المصفوفة للمتغير $elem$.

السطر الرابع:

أيضاً التابع $next$ شرطه أن يكون تابع الـ $first$ مستدعى مرة واحدة لذلك يجب أن يكون $currentPosition >= 0$ وذلك لأننا أسندنا قيمة ابتدائية للـ $currentPosition$ هي -1 لذلك عندما يُستدعى $first$ يصبح $currentPosition = 0$ من خلال التعليمة $currentPosition++$

الشرط الثاني للتابع هو ان يكون $index$ العنصر الحالي الذي أتعامل معه أصغر تماماً من $numberOfElements - 1$

فإذا اختل الشرط تعيد $false$ "علماً أن نمط التابع $bool$"

وسنوضح فكرة هذا الشرط من خلال المثال في آخر الشرح.

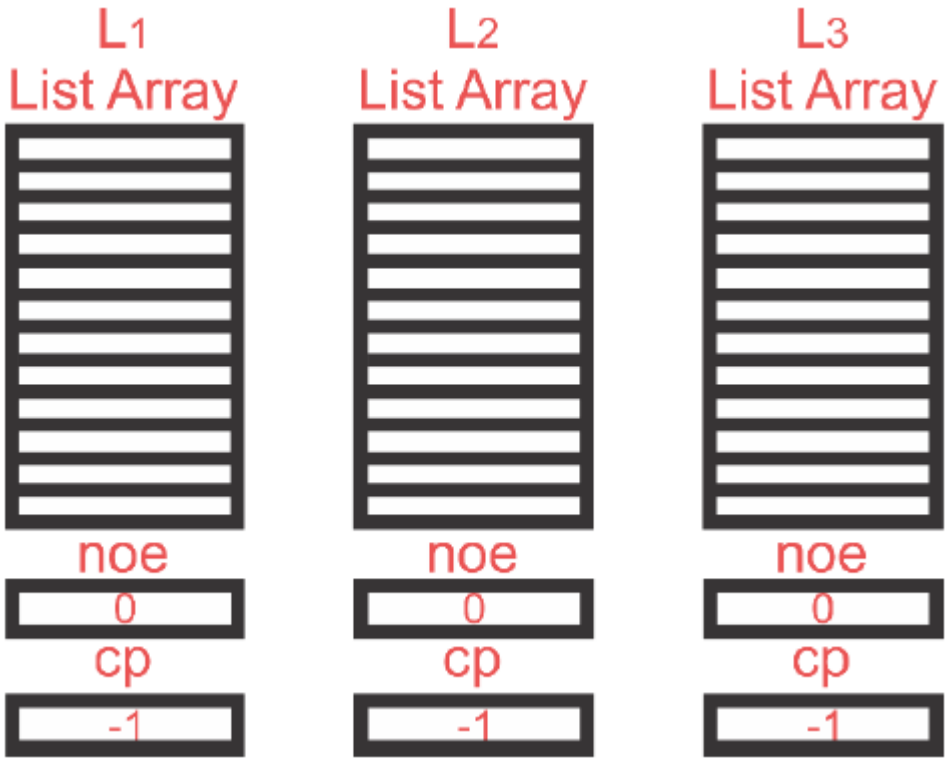
ويقوم التابع $next$ بإسناد عنصر المصفوفة التالي إلى المتغير $elem$.

السطر الخامس:

سنوضح التابع الرئيسي خطوة خطوة من خلال المثال:

أولاً عرفنا ثلاث أغراض من نمط الصف $List$ وهي ثلاث لوائح $L1$ و$L2$ و$L3$.

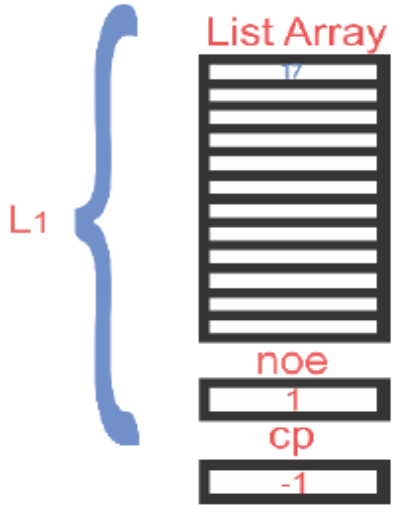
ومن خلال هذه التعليمة نكون قد حجزنا لكل لائحة مصفوفة $listArray$ والمتغيرين $numberOfElements$ و $currentPosition$ كالتالي:

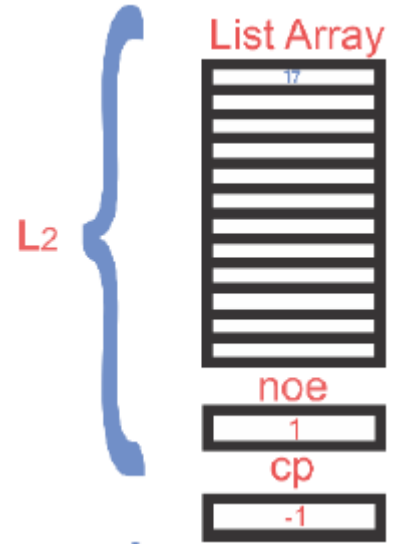


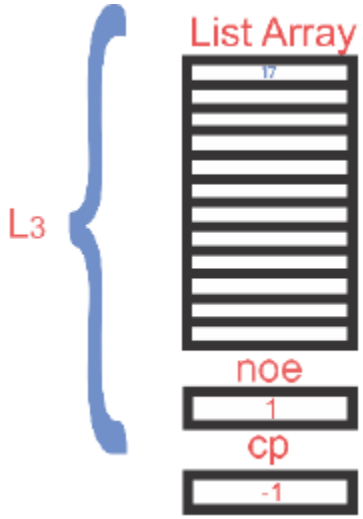
ثم نعرف المتغير $elem1$ من النمط $ListElementType$ أي من النمط $integer$

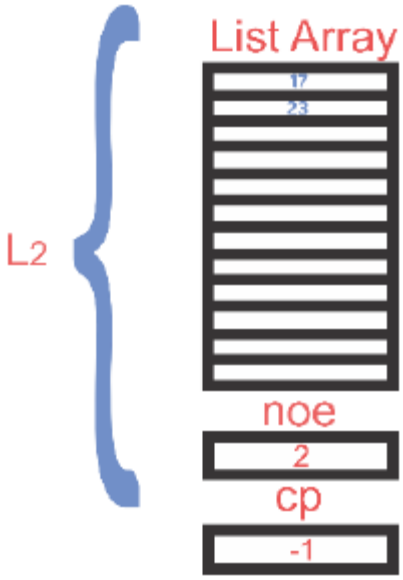
بعدها قمنا بإدخال $elem$ ولنفترض أنه 17 مثلاً من خلال التعليمة $L1.insert(elem1)$

نقوم بإدخال العنصر 17 للمصفوفة الاولى ومن خلال تعليمات التابع $insert$ يزيد المتغير $numberOfElements$ ويصبح $= 1$ تصبح اللائحة الأولى كالتالي:









ثم التعليمة $L2.insert(elem1)$

قمنا بتسجيل العنصر 17 ضمن المصفوفة الثانية وأيضاً يتغير الـ $numberOfElements$ ويصبح 1 كالتالي:

قمنا بإدخال ثاني للـ $elem1$

ولنفترض أنه 23 مثلاً

للتعليمة $L3.insert(elem1)$

نسجل العنصر 23 في المصفوفة الثالثة

ثم التعليمة $L2.insert(elem1)$

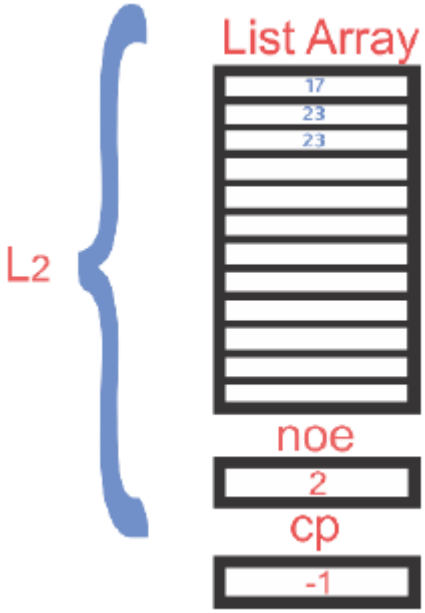
هنا الـ $numberOfElements = 1$ وبالتالي

فإن $ListArray[1] = elem$ أي يخزن الـ 23

في الخانة الثانية من المصفوفة الثانية ويصبح

$numberOfElements = 2$

التعليمة $L2.insert(elem1)$



عرفنا المتغير $avail$ من النوع $bool$ لاستدعاء التابع $first$ والتابع $next$.

التعليمة $avail = L2.first(elem1)$ لاستدعاء التابع $first$ للائحة $L2$

نجعل $currentPosition = 0$ وبذلك نسند أول عنصر في المصفوفة الثانية

للمتغير $elem$ فأصبح $elem = 17$ ويتبين ذلك من خلال عملية الـ $cout$

التعليمة $avail = L2.next(elem1)$ لاستدعاء الـ $next$

للائحة الثانية نختبر أولاً إذا كان $currentPosition >= 0$

وهذا الشرط محقق ثم نختبر إذا كان

$$currentPosition >= noe - 1$$

فإذا كان محقق فلا نستطيع الإكمال نعيد $false$

لكن هنا $3 > 0$ لذلك نكمل ويصبح $currentPosition = 1$

ونسند للمتغير $elem$ العنصر $ListArray[1]$ فيصبح $elem = 23$

التعليمة $avail = L2.next(elem1)$ يصبح $currentPosition = 2$ و$elem = ListArray[2]$

أخيرا التعليمة $avail = L2.next(elem1)$

أولاً نلاحظ تجريبياً أنه لا يوجد أي حد تالي لنسنده للـ $elem$

وعند اختبار الشرط $currentPosition >= noe - 1$ فإن هذا الشرط محقق لأن $cp = 2$ و$noe = 3$ لذلك نعيد $false$.