

بسم الله الرحمن الرحيم

جامعة صنعاء

كلية العلوم

مستوى 3 رياضيات حاسوب

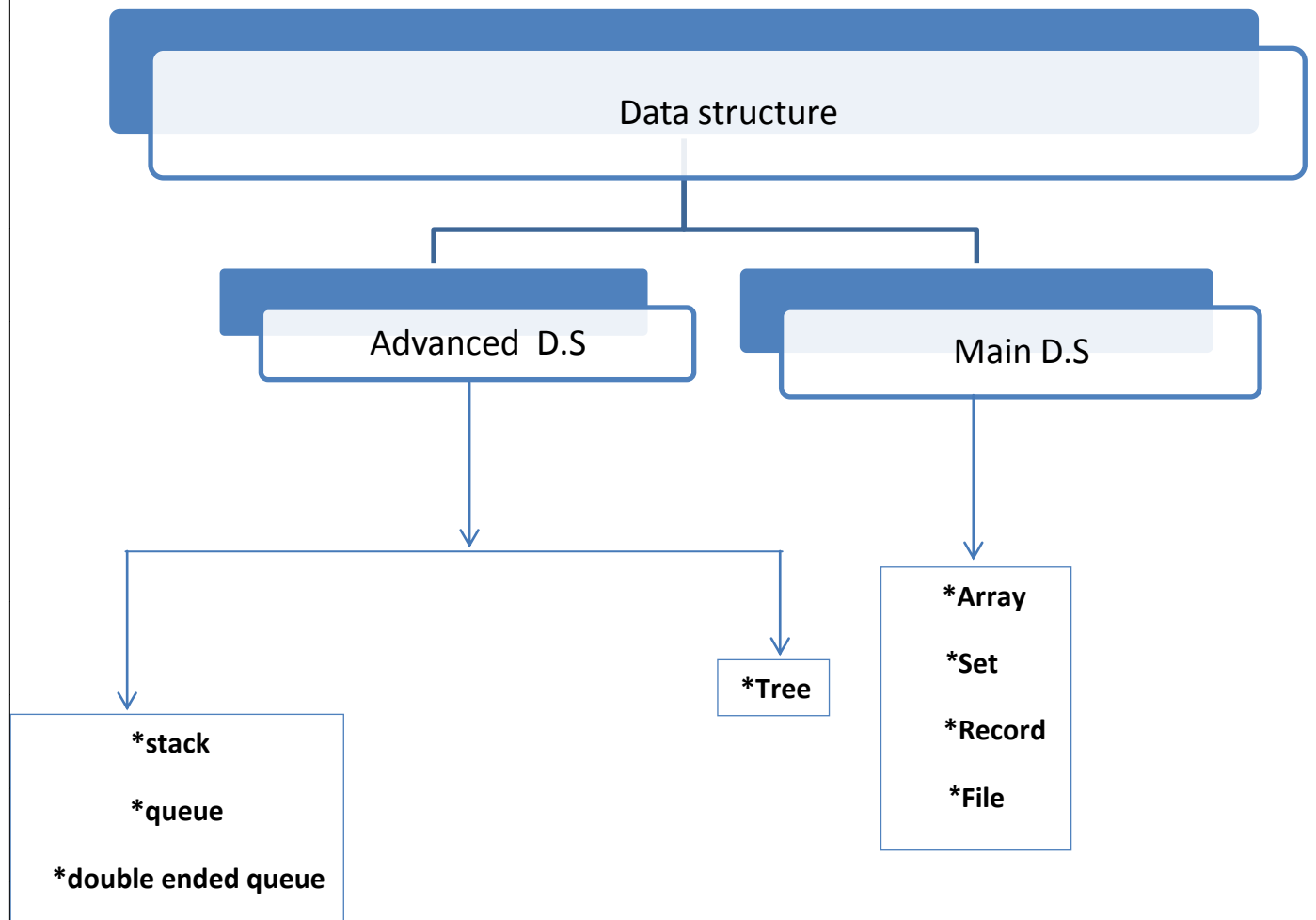
DATA STRUCTUER

(هياكل بيانات)

للاستاذ:

عمر الخالدي

سهام علي عامر



data: بيانات خام

Information: البيانات بعد معالجتها

Knowledge: مجموعه من الـ Information

Experience: مجموعه من الـ Knowledge

Expert system: decision support system & data mining

Set: مجموعه من الـ files

File: مجموعه من الـ records

Record: وهي عباره عن سجل يحتوي مجموعه من البيانات

Data mining:

*ماهي هيكله البيانات؟؟ هي طريقه تم إنشائها لكثرة البيانات (Data) المستخدمه في كل المجالات فكان لزاما علينا هيكلتها وتنظيمها لمعرفة كيفية الوصول للبيانات بالطرق الاسهل والادق، والواضح لدينا ان كل تخصص يتعلق بالحاسوب يبنى عليه _Data structure_

*ماهو الفرق بين البرنامج والخوارزميه؟؟

Program	Algorithm
*Syntax *Machine language *غير منتهي	*Semantic Human language *منتهي

** (Syntax)

وهي الصيغ او اكواد كتابة العمل او التعليمات في لغات البرمجه وتختلف من لغة برمجه الى اخرى

** (semantic)

وهي معنى الصيغه البرمجه بشكل عام في اللوغاريتمات او لغة البشر

*ماهي الخوارزميات؟؟

مفهوم الخوارزميه قديم فقد وضع من قبل البابليون (١٨٠٠ سنة ق.م) في عهد حام ورابي وهو اول توصيف لقواعد بعض انواع المعادلات وجاءت كلمة خوارزميه نسبة للعالم المسلم (محمد بن موسى الخوارزمي)،

والخوارزميات هي مجموعه من الخطوات المنطقيه التي يتم تنفيذها حسب ترتيب محدد والتي تُصف بدقه ووضوح وتكون طريقه عامه لحل مسأله معينه.

ولحل مسائل عن طريق الحاسوب يجب إتباع الخطوات ذات الترتيب التالي:

١. تعريف المسألة، اي فهمها ثم تحديد المدخلات والمخرجات (I/O) لهذه المسألة.
٢. التحليل، اي تحديد العمليات التي تؤدي الى حل المسألة.
٣. التنفيذ، اي ادخال البرنامج وبياناته الى الحاسوب لحل المسألة وايجاد الناتج.

وكل مسألة نريد حلها عن طريق خوارزميه يجب ان تتوفر فيها المواصفات التاليه:

١. المدخلات (In put) واول قيمه للمدخلات هي صفر ($i \setminus p >= 0$) اي اننا نستطيع عمل برنامج بدون ان نقوم بإدخال اي بيانات.
٢. المخرجات (out put) واول قيمه للمخرجات هي ١ اي ان النتائج الخارجه يجب ان تكون معلومه واحده ($o \setminus p >= 1$) على الاقل
٣. المحدد، فكل تعليمه (instruction) يجب ان تكون واضحه ليس فيها غموض .
٤. يجب ان يكون لها حد اي (نهايه) فإذا قمنا بتتبع الإيعازات او الأوامر لأي لوغاريتميه يجب ان نصل الى عدد محدد من الخطوات في النهايه.
٥. الفعاليه (effectiveness)، اي ان كل ايعاز يجب ان يكون اساسي وبسيط وفي حال وصول العمليه اليه يجب ان يكون قابل للتحقيق.

*ما علاقة الخوارزميه بال data structure؟؟

هيكله البيانات هي نتيجة لدمج الكائنات مع الخوارزميات، والكائن (Object) وهو بيان او رمز تحفظ البيانات بداخله وقد نعتبره x ونمرر من خلاله قيمه (بيان) وقد نحتفظ بالقيمه بداخله.

*ماهي خصائص الخوارزميات:

- ١) خطوات الخوارزميه مرتبه.
- ٢) خطواتها منتهيه.
- ٣) خطوات الخوارزميه تنفذ اعمال بسيطه.
- ٤) معرفه جدا.
- ٥) طريقه الحل يجب ان تكون عامه.

- ٦) فعاله وعند تنفيذ الخوارزميه في الحاسوب يجب ان لا تكون التكلفة عاليه سواءً من ناحية زمن المعالجه (processing time) او المساحه المشغوله في الذاكره الرئيسيه.
- ٧) استخدام المخططات الانسيابية.
- ٨) استخدام جمل قصيره وخاليه من الكلمات الغير مفيده.

أشكال المخططات الانسيابية:



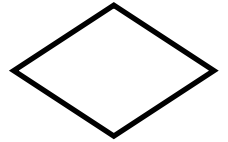
يستخدم للدلالة على بداية ونهاية الخوارزميه.



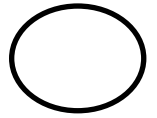
يستخدم للدلالة على عمليات الادخال والايخراج.



يستخدم للدلالة على عمليات المعالجات الحسايبه.



يستخدم لاختبار شرط معين وعمليات المقارنه وللعمليات التي تحتاج الى اتخاذ قرار.



يستخدم لنقطة عودة البيان وتصل جز ء معين المخطط الانسيابي بأخر وغالبا يكتب داخلها رقم او حرف.

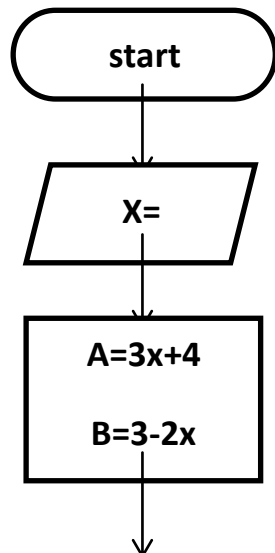
يدل على ترتيب خطوات الخوارزميه.



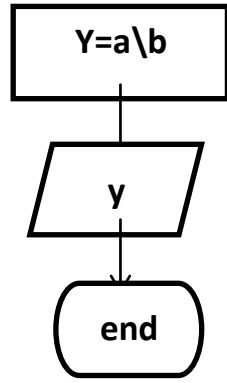
**مثال (١) ضع المخطط الانسيابي لخوارزميه لحل المسأله الرياضيه التاليه:

$$Y=3x+4 \setminus 3-2x$$

الحل:

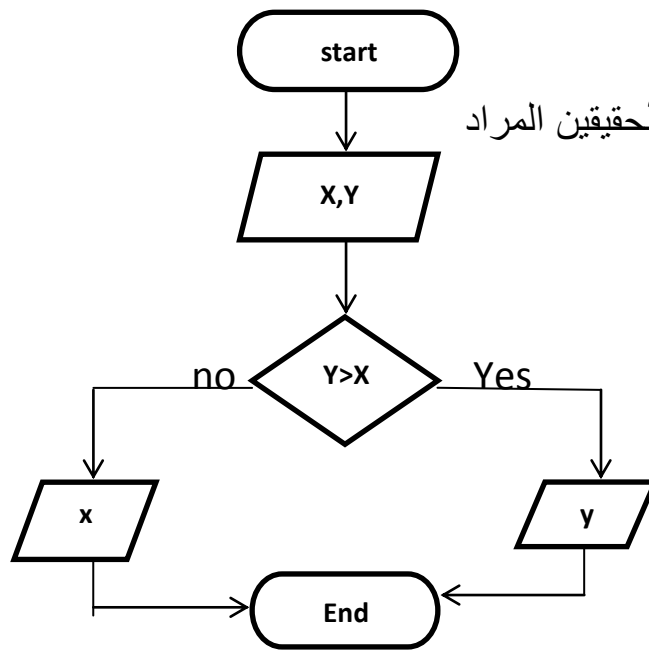


o



مثال (٢) صمم المخطط الانسيابي لخوارزميه للمقارنه بين عددين حقيقيين موجبين وطباعة الاكبر بينهما:

الحل:

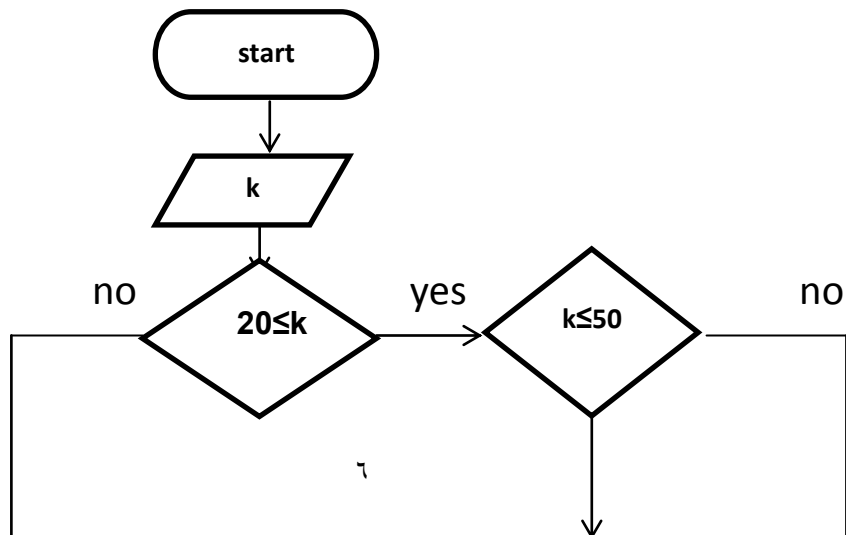


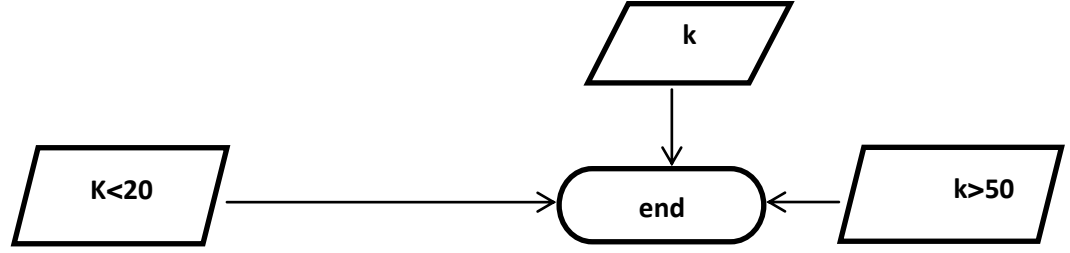
حيث ان X,Y تمثلان العددين الحقيقيين المراد المقارنه بينهما.

مثال (٣) ارسم المخطط الانسيابي للخوارزميه المناسبه للمتراجحه التاليه:

$$20 \leq k \leq 50$$

الحل:

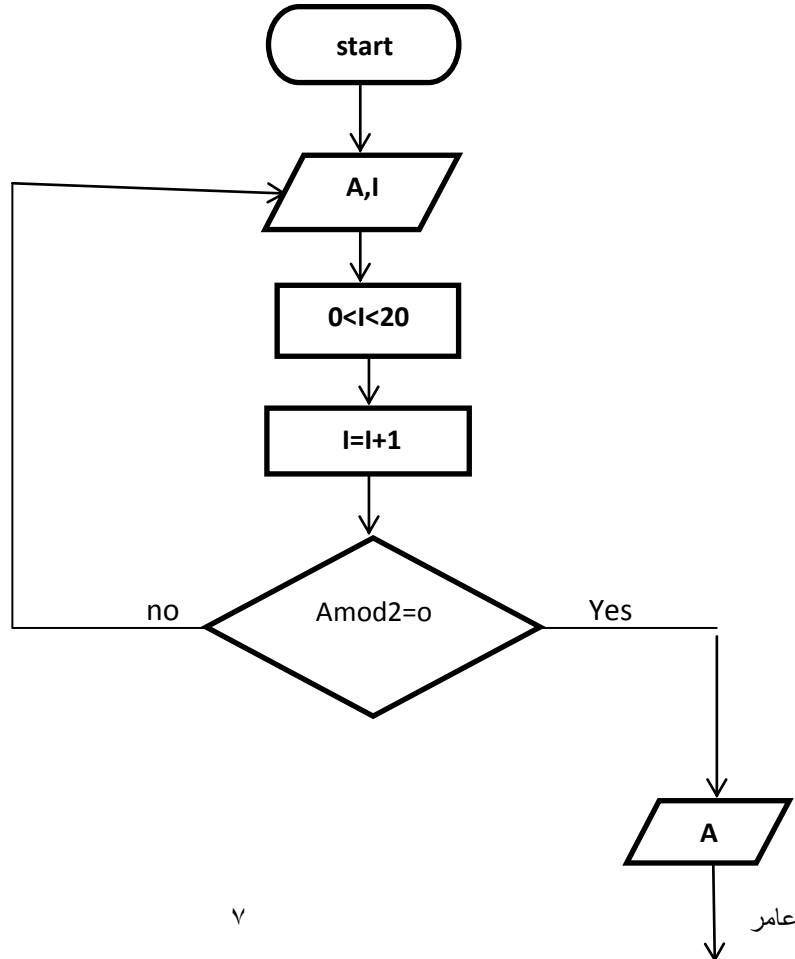




المكونات التكرارية:

يتفوق الحاسوب على الانسان بالقدره على تكرار العمل مهما كان حجم التكرار كبير ولأي فتره زمنيه دون تعب او كلل او ارتكاب اخطاء ويتم في حلقة دوران بطريقة (counter)، فبعض المسائل تتضمن تكرار لمعالجة بعض أجزائها عدداً معين من المرات ويطلق على هذه العمليه (loop) وهي تكرار معالجة مجموعه من الخطوات الى ان يتحقق الشرط ثم انهاء البرنامج

*مثال(٤) لدينا ٢٠ عدد صحيح ونريد فرز الزوجي منها فقط:



*الهيكل العام للخوارزميه:

يتكون الهيكل العام للخوارزميه من رأس وجسم فيتكون الرأس من اسم الخوارزميه ومجموعة (parameters) ويجب ان تنتهي بقوسين ()، اما جسم الخوارزميه فيتكون من مجموعة الخطوات او الاوامر التي تطلبها الخوارزميه.

*العناصر الأساسية للخوارزميه:

- ❖ اسم الخوارزميه يكون متبوع بقوسين.
- ❖ يتم التواصل عن طريق (parameters) بين الخوارزميات وتبادل البيانات.
- ❖ نقطة البداية.
- ❖ جسم الخوارزميه وداخلها الخطوات والاوامر.
- ❖ النهاية (end)

مثلا:

Algorithm add1() }	اسم الخوارزميه
Begin	
Sum=0;	
For I = 1 to 10 do;	جسم الخوارزميه
Sum=sum+I	
end }	النهايه

Main D.S.:

المصفوفات (Arrays): وهي عبارة عن مجموعة من البيانات التي تتشابه في النوع ولها اسم مشترك وتُخزن في الذاكرة بشكل تسلسلي.

أولاً المصفوفه في بعد واحد بمعنى اما مصفوفة صف واحد او عمود واحد تحتوي على (index) يبدأ من ٠ في لغة C ومن ١ في لغة باسكال.

ويتم تمثيله ب(١) العموديه ، بداية ال (index) + عدد الصفوف

(٢)الصفيه ، بداية ال (index) + عدد الاعمده

اما اكثر من بُعدين فيتم تمثيلها بالطريقه الاتيه:

(١) العمود بداية ال (index + رقم الصف × عدد الاعمده)

(٢) الصف بداية ال (index + رقم العمود × عدد الصفوف)

تمثيل المصفوفه في الحاسوب:

اي مصفوفه مه ما كانت ابعادها سواء بُعد واحد او عدة ابعاد فإن الحاسوب يحولها الى بعد واحد دائماً ويعتمد على مواقع العناصر ال (index) كما في الشكل التالي

(index): هي مواقع العناصر في المصفوفه

A{0}	A{1}	A{2}	A{3}	A{4}	A{i}
------	------	------	------	------	------	------	------	------

مواقع العناصر في مصفوفة ذات بعد واحد في الحاسوب

A{0,0}	A{0,1}	A{0,2}	A{0,3}	A{i,j}
--------	--------	--------	--------	------	------	-----	------	--------

تمثيل مواقع العناصر في مصفوفة ذات عدة ابعاد في الحاسوب

وفي المصفوفة ذات البعد الواحد لمعرفة موقع اي عنصر نعلم على موقع العنصر الاول في المصفوفة وذلك لان موقع اول عنصر يتولد عشوائياً (لأنه في الram) اعتماداً على ما قبله من قيم والذي يليه يكون تراكمي، وتوزع العناوين اوتوماتيكياً وحسب نوع البيانات.

فمثلاً بفرض ان نوع البيانات integer والموقع الاول هو 005 فإن الموقع الذي يليه هو 007 وذلك لان هذا النوع من البيانات يحجز تلقائياً 2 بايت في الذاكرة لكل عنصر.

A{1}	A{2}	A{3}
------	------	------	------	------	------

005 007 009

← ذلك بفرض ان البيانات من نوع integer

* ولإيجاد موقع اي عنصر نأخذ الخوارزمية التالية وهي بلغة باسكال:

$$\text{Loc}(a\{i\}) = \text{add}(a\{1\}) + \text{size of type} * i - 1$$

حيث ان $a\{i\}$ هو موقع العنصر المراد ايجاده

$A\{1\}$ عنوان اول عنصر في المصفوفة

size of type حجم نوع البيانات

$i - 1$ وهو موقع العنصر السابق للعنصر المطلوب

* مثال : اوجد قيمة الموقع $a\{3\}$ في الشكل السابق:

$$\text{Loc}(a\{3\}) = \text{add}(a\{1\}) + 2 * 2$$

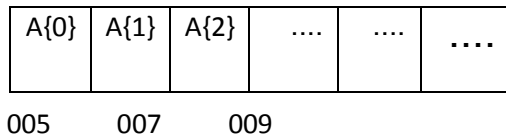
$$= (005 + 2 * 2)$$

$$= 009$$

ولإيجاد موقع العنصر نفسه بلغة ال++C

$$\text{Loc}(a\{i\}) = \text{add}(a\{0\}) + \text{size of type} * i$$

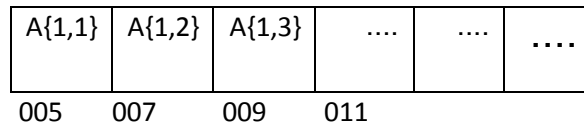
$$\text{Loc}(a\{2\}) = \text{add}(a\{0\} + 2 * 2)$$



$$= (005 + 2 * 2)$$

$$= 009$$

* اما في المصفوفه ذات عدة بعدين نجد مواقع المصفوفه التاليه بلغة الباسكال كالتالي:



$$\text{Loc}(a\{i\},\{j\}) = \text{add}(a\{1\}\{1\} + \text{size of type} * [M * (i-1) + (j-1)])$$

حيث ان m عدد الاعمده الكلي

** وفي لغة ال C++:

$$\text{Loc}(a\{i\},\{j\}) = \text{add}(a\{0\},\{0\}) + \text{size of type} * (M * i + j)$$

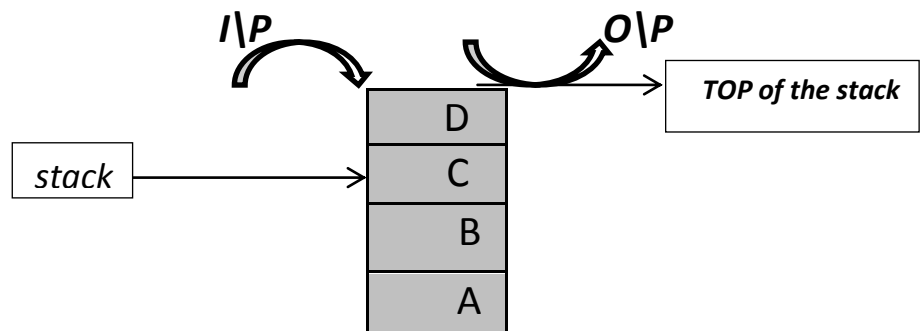
:Advanced D.S.

:Stack

هو عبارته عن مصفوفه تُعبأ البيانات فيها بطريقه تعكس البيانات ، وهو نمط من الـ D.S.

وعملية عكس البيانات تحدث بسبب وضعية الـ Stack وترتب البيانات على حسب (FIFO) First in last out)

ودخول البيانات وخروجها يتم من نهايه طرفيه واحده تسمى (top of the stack) المؤشر



ماهي العمليات (primitive operation) التي يجريها الـ stack:

(1) التهيئه (initialization) وفيه يتم تحديد حجم الـ stack وموقع المؤشر ونوعه

(2) الادخال (insertion) or (push)

(3) الازحاج (deletion) or (pop)

ولدينا top وهو المؤشر الخاص بـ stack .

ففي البدايه يكون المؤشر (top) في الاسفل وكلما تم ادخال بيان على الـ stack يصعد المؤشر الى الاعلى فيعتمد حجم الـ stack على موقع المؤشر، فعندما يكون الـ stack فارغاً تكون قيمة المؤشر

(-1 في C) و (0 في باسكال)

ومن صفات المؤشر (top):

(1) هو مؤشر يصعد ويهبط على حسب العمليات (ادخال، اخراج)

(2) وهو موقع في الذاكره يحمل قيمه واحده

(3) يتناقص عند الحذف ويزداد عند الاضافه او الادخال

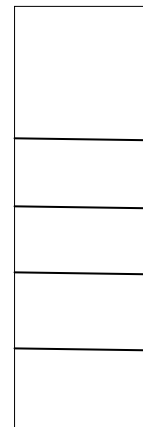
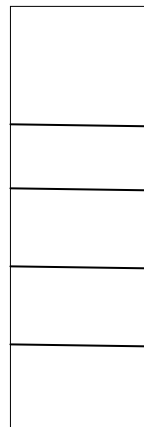
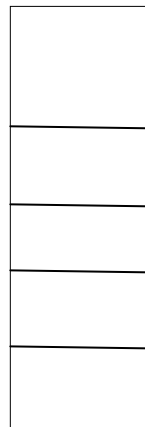
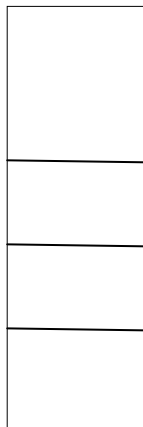
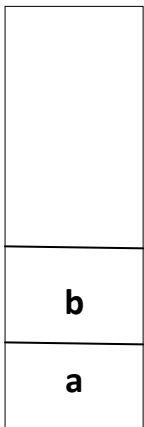
(4) يدل على حجم الـ stack

(5) عند الادخال (push) يجب التأكد ان المؤشر لم يصل الى القيمه العظمى

(6) عند الازحاج (pop) يجب التأكد ان المؤشر ليس فارغاً

وعند تهيئه الـ stack علينا ان نعطي المؤشر قيمه تدل على انه فارغ ونحدد الـ stack بـ s و item بـ i

فيقدموا لنا عملية الادخال $push(i, s)$ و الازحاج $pop(s)$



B

C

D

E

F

Push (s , c) ; (frame (b))

Push (s , d) ; (farm (c))

Push (s , e) ; (farm (d))

Push (s ,f) ; (farm (e))

Pop (s) ; (farm (e))

Pop (s) ; (farm (f))

وسميت بالـ push down list بسبب العمليات المتكرره

اذن ما نلاحظه يشرح لنا عمليات pop & push

و ينتج لنا الاطار ويتم داخله جميع العمليات وهو ايضاً ما يُعرف بالـ array ومعه المؤشر.

*الاطار (frame) في الخوارزميات ولغات البرمجه هو الكود الخاص المختص بإنشاء (stack) والمؤشر والـ items

**تمثيل الـ stack بلغة الـ C :

مثال: انشئ بلغة C (stack) حجمه ١٠٠:

```
Include<isotream.h>
```

```
# dfine st_size 100
```

```
Stract stack
```

```

    {
        Int top;
        Int item [st_size];
    };

```

frame

الـframe يحتوي على عنصرين رئيسيين:

- ❖ مصفوفه لتحمل جميع العناصر لـstack
- ❖ Integer ليشير الى موقع مؤشر الـstack

*صمم إستماره تحتوي على ٢٠ عنصر (اسماء طلاب)

```
# define name_size 20
```

```
Struct stack
```

```
{
```

```
Int top;
```

```
Char item [name_size]
```

```
};
```

S عباره عن كائن نضع فيه البيانات وتوجد عمليات تطبيقيه ستؤخذ لاحقاً تعتمد على الادخال والايخراج.

لو افترضنا ان عناصر الـstack تحتوي العناصر item[0] to item[99] وايضاً لا يوجد لدينا سبب بان نجعلها محدده فقط على نوع واحد من البيانات ولكن نفرض اننا نحتاج عدة انواع من البيانات مثلاً Float , char , int او أيا كانت نوع البيانات التي سنعطيهها للـstack

اذن يمكن للـstack ان تحتل مختلف انواع البيانات

```
# define STACK SIZE 100
```

```
# define INT 1
```

```
# define FLOAT 2
```

```
# define STRING 3
```

```
Struct stack element
```

```
{ int e type
```

Union

```
{ int I val;  
    Float f val;  
    String c val;  
}element;  
}
```

```
Strict stack
```

```
}
```

```
Int top ;
```

```
Strict stack element item[STACKSIZE] ;
```

ولطباعة العنصر الاعلى للـ stack :

```
Strict stack element sp;
```

```
Sp = s.item {s.top}
```

```
Switch (sp-etype)
```

```
{
```

```
case INT :print f (sp.i val);
```

```
Case float :print f (sp.f val);
```

```
Case string :print f (sp.c val);
```

```
}
```

وعندما تكون الـ stack فارغه لا تحتوي على عناصر نجعل الـ top == -1 أي (s.top == -1) كود برمجي

```
If(s.top == -1);
```

```
Return(null/Empty);
```

```
Else
```

```
Return (item);
```

او كخوارزميه عامه

```
If(s.top == -1)
```

```
stack is empty;
```

```
Else
```

```
Stack is not empty;
```

تطبيقات الـ pop:

إحتماليات (under flow) "وهي الوصول الى النهايه" توجب علينا دراسة عملية الـ pop للعناصر داخل الـ stack الذي قد يكون فارغا وتجنب ذلك.

عند عمل مثل هذا الطلب عن طريق المستخدم يجب إظهار رساله للمستخدم تعلمه انه لا يوجد اي قيم

وتخبره عن حالة (under flow) هذا يحتم على الـ pop ان يؤدي الخيارات الثلاثه التاليه:

(1) اذا كان الـ stack فارغا إطبغ رساله تحذير وأوقف التنفيذ (halt execution)

(2) أبعد المؤشر (top) من الـ stack

(3) ارجع هذا العنصر element للبرنامج الذي يقوم بالاستدعاء.

*مثال:

```
Int pop (struct stack .ps)
{
If (empty (sp) );
{ print ("stack under flow" ) ;
Exit();
}
Return (ps_item s [ps_top--]);
}
```

ولو افترضنا ان عملية pop تستدعي

```
ps;
ps_top[88];
فيأشـر الى ps_top[87];
```

تطبيقات لعمليات push:

وهذه تكون أسهل في التطبيق باستخدام مصفوفة pest array وأول محاولة للـpush الإجراءات تكون كالتالي:

```
Void push(struct stack PS, int x )
{
PS →items[ ++(PS → Top)]=x;
Return;
}
```

وهنا الإجراء يجعل item x إلى stack بزيادة S.top بقيمة واحدة.

ومن ثم يدخل قيمة x إلى عناصر array s لكن تدخل معنا مشكلة overflow وأن تطفح الحاوية stack بـ items عن المقدر له هنا يجب إظهار رسالة بذلك.

```
Void push (struct* PS; int x)
{
If(PS    top== stack size -1)
{print f("stack is full");
Exit(1);
}else
PS    items[(PS    top) ];
Return;
};
```

ملاحظة:

- 1/ أي شيء يستخدم مصفوفة يواجه مشاكل تحتاج لحل.
- 2/ لا تستطيع تغيير حجم المصفوفة أثناء التنفيذ بالبرنامج.

****Infix, postfix, prefix***

إذا كانت العملية ضمن المعادله فتعتبر ***Infix***

وإذا كانت العملية تلحق بالمعادله فتعتبر ***postfix***

اما اذا كانت العملية سابقه للمعادله فتعتبر ***prefix***

هذا القسم يعتبر التطبيق الرئيسي الذي يوضح الانواع المختلفه بالstack والعمليات المتنوعه وايضا الدوال المطبقه عليهم.

*مثال للتوضيح:

$$\text{Infix} = A+B$$

$$\text{Postfix} = AB+$$

$$\text{Prefix} = +AB$$

حيث ان operands=A,B

operator تكون +,-,*,%,,\$ (عمليات)

اي ان هذه التسميات تطلق على حسب موقع العمليات .

**مثال:

حول العمليه التاليه الى **Prefix** و **Postfix** :

$$1) A+B*C$$

نقوم بتحريك العمليات فقط وليس العمليات،

$$\text{Prefix} = +A*BC$$

$$\text{Postfix} = A+BC*$$

$$=ABC*+$$

** مثال:

اذا كان لدينا المعادله الاولى ولكن مع وجود اولويه للجمع حولها لل **Prefix** , **Postfix** :

$$(A+B)*C$$

$$\text{Prefix 1}) = +AB*C$$

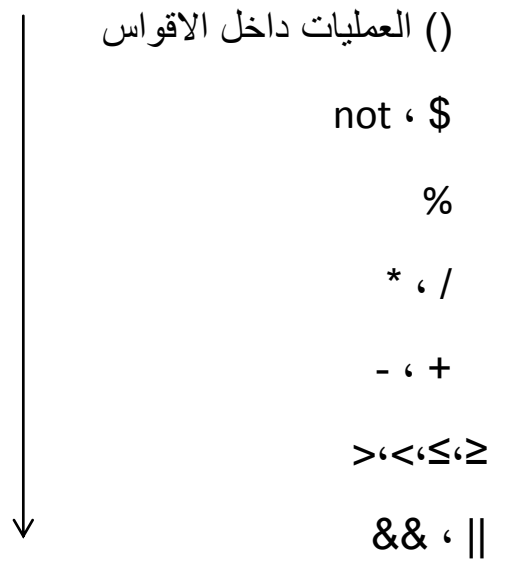
$$2) = *+ABC$$

$$\text{Postfix} = AB+C*$$

ولو اعتبرنا ان لدينا 5 operators التاليه اضافه الى العمليات المنطقيه:

- Add (a
- Multiplication (b
- Subtraction (c
- Division (d
- Exponentiation (e
- Not (f
- (g % باقي القسمة

فيكون ترتيب أولوية العمليات كما يلي :



** مثال: حول المعادله التاليه الى postfix & prefix:

$$=(A+B) * (C-D)$$

$$\text{Prefix =1) (+AB) * (-CD)}$$

$$2)*+AB -CD$$

$$\text{Postfix =1) (AB+) * (CD-)}$$

2)AB+ CD-*

** مثال:

A\$B*C-D+E/F/(G+H)

prefix= 1) \$AB*C -D+E/F/ (G+H)

2)*\$ABC- D+E/F/ (G+H)

3) -*\$ABCD +E/F/ (G+H)

4)+-*\$ABCD//EF +GH

تمرين: اوجد postfix لنفس المعادله??

** مثال:

=((A+B)*C-(D-E)) \$ (F+G)

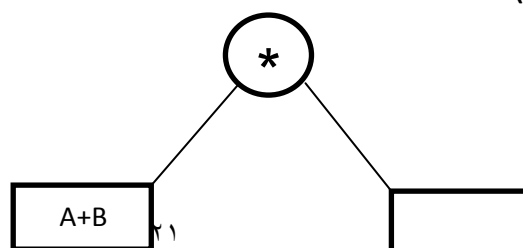
Prefix=1) ((+AB)* C- (-DE) \$ (F+G)

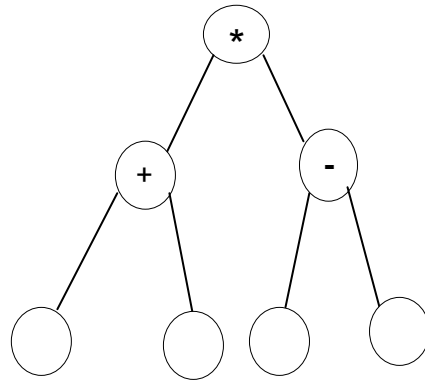
2) -*+ABC-DE\$(+FG)

تمرين: اوجد postfix لنفس المعادله?

طريقة TREE:

1: (A+B) * (C-D)



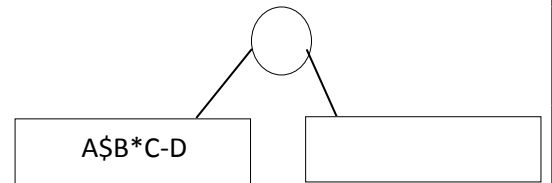
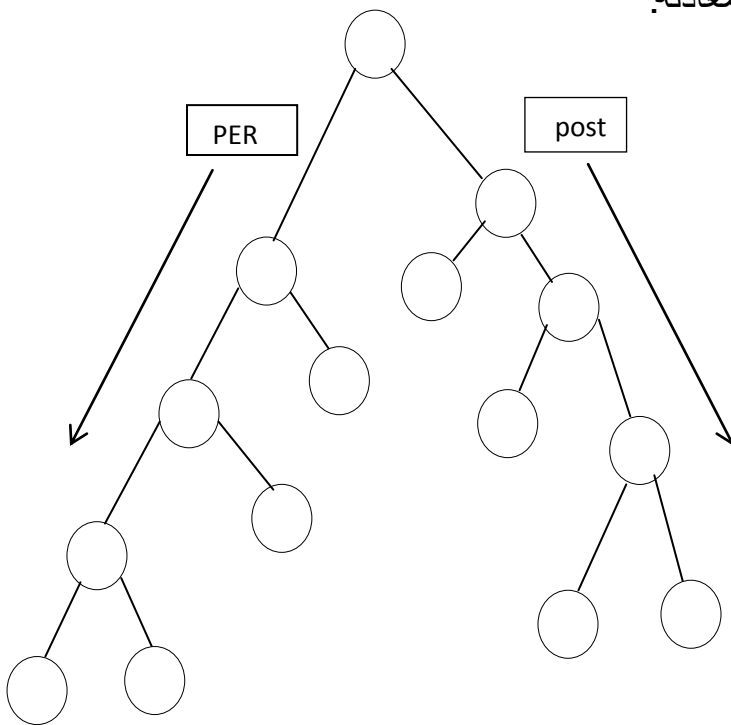


Per = *+AB-CD

Post = AB+CD-*

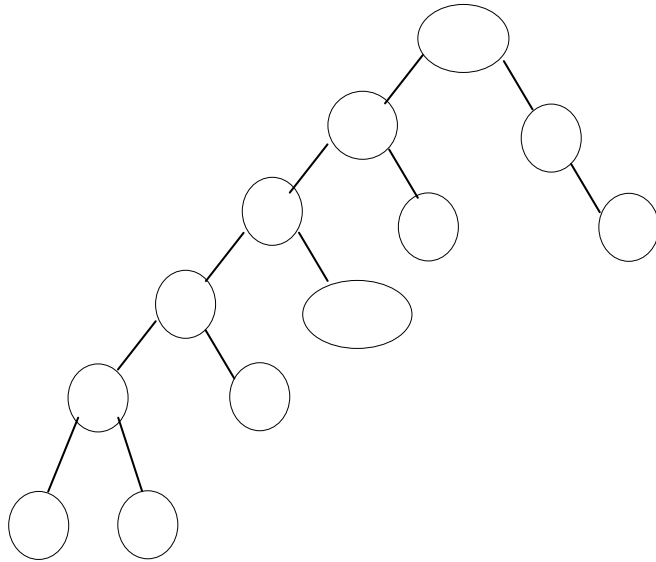
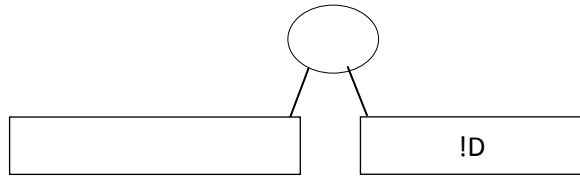
ونختار العملية الاولى بحيث يحصل توازن في المعادله.

2:A\$B*C-D+E/F/(G+H)



PER = +-*\$ABCD//EF+GH

3:(X+Y*Z > 100)&&C|| ! D



pre = ll&&>+X*YZ100C!D+

:Queue

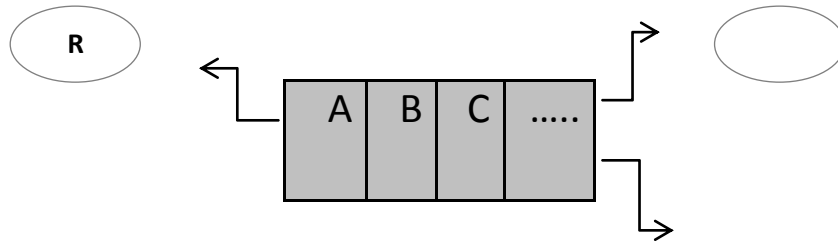
ماهو الفرق بين الـ stack و الـ queue؟؟

*الـ stack: كما ذكرنا إنها مصفوفة يتم التعامل معها بطريقة مميزة بحيث يكون الـ I/O من طرف واحد بواسطة الـ Top

* الـ queue: هي أيضاً مصفوفة لكنها مفتوحة الطرفين ، اذن فهي تحتاج الى مؤشرين يقومان بعمليات الحذف والاضافه .

insert (1) Rear → (مؤشر عملية الادخال)

delete (2) Front → (مؤشر عمليات الاخراج)



(queue)

كيف تتم التهيئه؟

في الـ stack كانت تتم التهيئه بجعل (Top = Null) ،

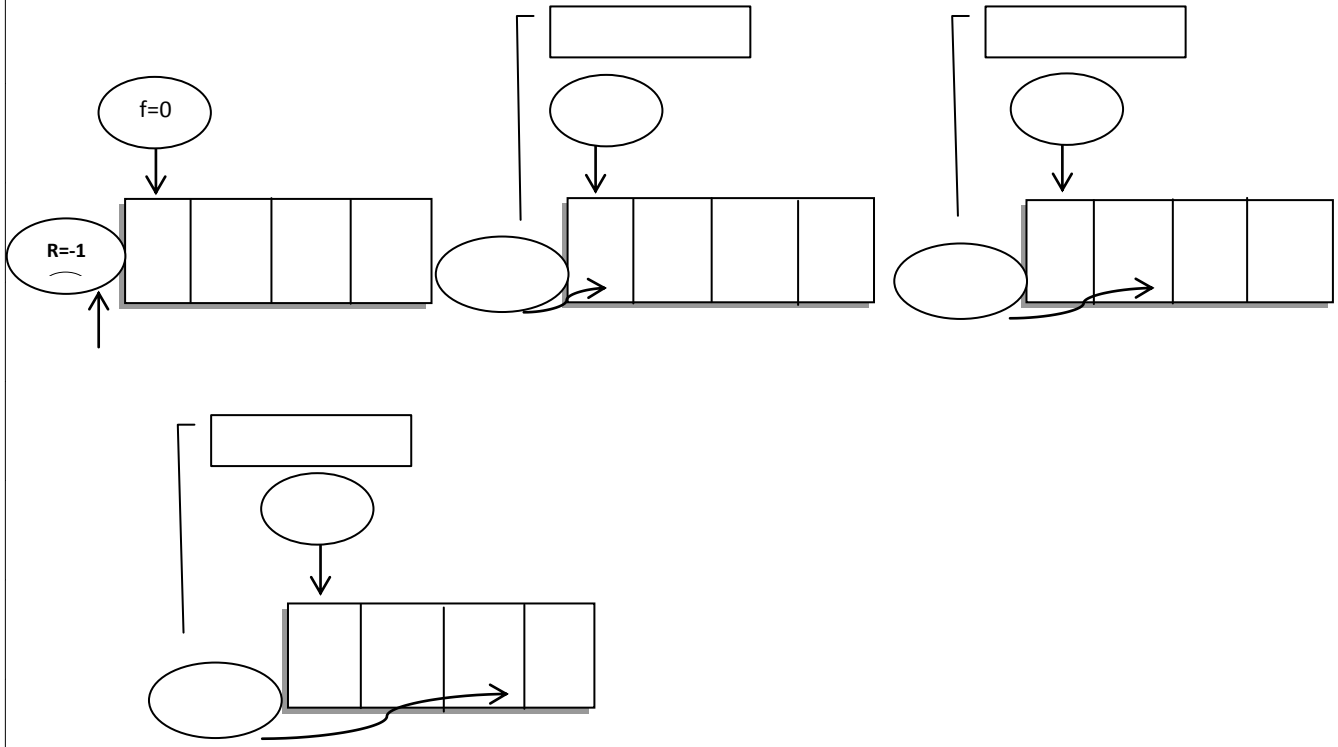
أما في الـ queue يكون من خلال Rear الذي يتحكم بالتهيئه عندما تكون (R = -1 , F = 0) وبذلك يتولد لدينا قانون خاص بالـ queue وهو (R < F) عند التهيئه

ولإيجاد عدد عناصر الـ queue نستخدم القانون:

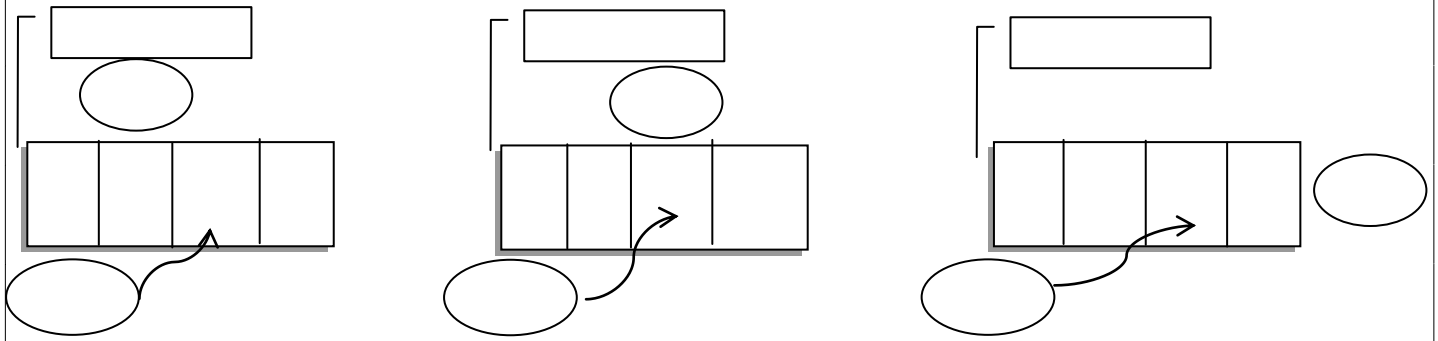
$$Q = R - F + 1$$

** تمثيل عملية الإضافة Insert :

$$-1-0+1=0$$



ولتمثيل عملية الحذف: مثلاً (Delete A)



الفرق بين عمليتي الحذف والاضافه في ال stack وال queue

في stack اذا اردنا الحذف نجعل ال Top عند القيمه المراد حذفها ثم نحذف فنتنقل قيمة ال Top وعند الاضافه تزيد قيمة ال Top

Queue: عند الحذف تتزايد قيمة front وتبقى قيمة rear ثابتة وعند الاضافه تتزايد قيمة rear وتبقى front ثابتة.

وعندما $R < F$ يكون ال Queue مهيبه.

*برنامج من اجل إنشاء Queue يحمل مؤشرين:

Struct Queue

```
{  
    Char element [max];  
    Int r, f;  
}
```

** برنامج لتهيئة ال Queue:

Intial (struct Queue * pq)

```
{  
    Pq → rear= -1;  
    Pq → front= 0;  
}
```

*** خوارزمية برنامج إضــــــــــــــــافه:

Insert (struct queue * pq , char)

```
{  
    If (pq → rear < max - 1)  
        { (pq → rear)++  
    Pq → element [pq → rear]=e;  
        }  
    Else print f ("full");  
}
```

**** خوارزمية برنامج حذف :

```
Char delete (struct queue * pq)
```

```
{
```

```
  If (pq → rear < pq → front);
```

```
  {
```

```
    Print f ("empty")
```

```
    Return ;
```

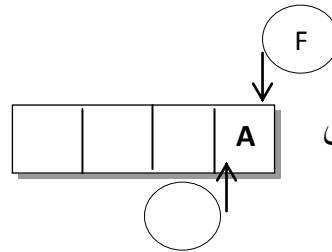
```
  }
```

```
  Else
```

```
    Return (pq → element [pq → front++]);
```

```
  }
```

فلا نستطيع الاضافه لأن المؤشر يشير الى الامتلاء



اما اذا كان لدينا هذا الشكل

بحسب الترتيب ولكنه في الحقيقه فارغ. ولذلك نجد أن الحل فيه عملية shift لنقل العناصر وهذا يأخذ منا وقت كبير وأيضا جهد في عمل code وأيضا يؤثر على مشكلة الأسبقية ولذلك ظهرت لدينا Queue circular .

: Linked list

كما ذكرنا أن stack و Queue عبارة عن مصفوفات لها خصائصها واحتياجاتها من الناحية الخوارزمية ومن الناحية البرمجية وقواعدها في الإدخال والإخراج وعملية الحفظ للعناصر والتي تتم في عناوين ثابتة في الذاكرة Index. وهنا يأتي تمثيل المصفوفات في الذاكرة بطريقة ديناميكية تسمى Dynamic representation memory address

تمكنها من الحفظ والإضافة وتمثل على شكل عُقد

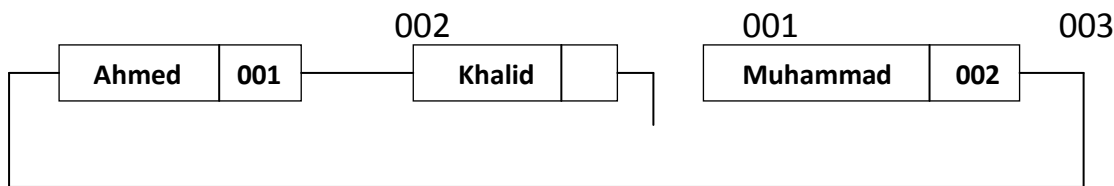
(node) وكنا إذا أردنا تحديد حجم الـ stack و Queue

فلا بد أن نحدده قبالا ويصبح هنا الحجم ثابت.

ولكن هنا في Linked list لا نحدد الحجم مسبقا

فكنا إذا أردنا إضافة عنصر ينشأ عقدة جديدة وهكذا.

وللربط بين العُقد يكون بحسب العناوين. مثال:-



ولإنشاء سجل عن طريق node بلغة الباسكال:

```
Ptr node=^Node
```

```
Node=Record
```

```
Name=string [20]
```

```
ID=Int
```

```
Link: Ptr, Node;
```

```
End
```

^ : عندما يجد الـ compiler هذه العلامة فإنه يقوم بتعريفها لاحقا على حسب لغة البرمجة في C++ language سيتم تعريفها في جميع اللغات على حسب اللغة إلا في لغة الباسكال فهو يعرف هذه العلامة محجوزة.

Link: (الرابط) يُوْشر على node من نفس النوع .

وفي المثال السابق نعرف الرابط link بنفس Ptr Node (بنفس المؤشر) فهو يحمل عنوان Node آخر.
ومن أجل إنشاء أول عُقدة نستخدم مُعرف جديد (New) ومن أجل تعبئة البيانات سنكتب عبارة التعبئة (first
. (^ .Name='A';

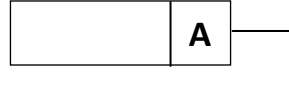
Begin

First (New)

First ^ .Link =null;

First ^ .Name='A';

End;



عملية الحذف نضيف عبارة `dispose first;`

من مميزات **Linked list**:

(١) عند الحذف تُحذف العنصر فعلياً من الذاكرة ولا نستطيع استرجعها .

(٢) لكل سلسلة لا بد أن يكون لها مؤشر لأول السلسلة وإذا فُقد الرابط (link) نفقد البيانات وحينها تظهر لنا رسالة وهي الذاكرة ممتلئة .

(٣) إذا خرجنا من البرنامج تبقى العناصر في الذاكرة حاضرةً مساحةً إلى أن يتم حذفها على عكس المصفوفات عند إغلاق البرنامج فإن البيانات تُحذف.

ولو قمنا بلغة ++C بتعريف رابط وعقدة جديدة يجب أن يحتوي هذا التعريف على ثلاث نقاط أساسية:

أ - نقطة البداية .

ب - المعلومة التي بداخل هذه النقطة .

ت - الرابط (Link)

← جملّة التصريح (١) Struct ptr Node

{

Int info;

Struct ptr Node *link;

*First, *last;

```
}
```

هذا البرنامج يحدد أول عُقْدة وآخر عُقْدة للبرنامج .

← Type def struct ptr Node Node ← جملة تصريح (٢).

وعند هذا التصريح يجب أن أنسب جميع المتغيرات إليه مثل:

```
Type def int x.
```

إذا أردنا الحجز في الذاكرة هذا يعني أننا نريد أن نحجز في مؤشر لو افترضنا

```
P=malloC(2); لو عرفنا Int *p;
```

```
Int *P;
```

```
P=malloC(2);
```

مثال: للحصول على Node يجب أن ننشأ have بدالة الإنشاء وهي get Node.

إنشاء Node جديد ضمن Node السابقة:

```
Node *get Node ( )
```

```
{
```

```
Node *P;
```

```
P= (Node*) malloC (size of (int));
```

```
return;
```

```
}
```

أنشأ عُقْدة جديدة f تكون بالشكل:

```
Node *f
```

```
F=get Node ( )
```

أدخل قيمة لهذه العُقدة ثم احذفها؟

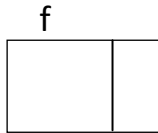
Linked list لها شروط:

- إذا كانت وحيدة لها برنامج وهو:

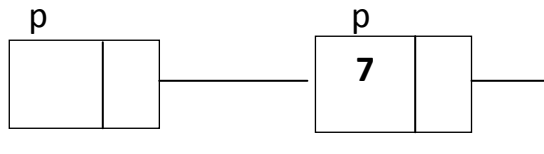
Initial (Node *f)

```
{  
    F=get Node;  
    f ->Link=null;  
    f-> Info=7;  
}
```

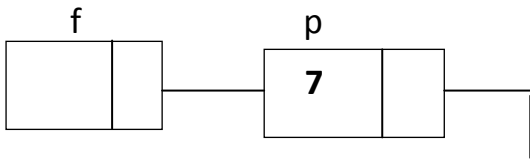
- في حالة أننا نريد إضافة عقدة جديدة للسابقة بحيث أن تكون f هي الأولى حيث أن تكون العقد f هي الأولى وتأخذ دائما قيم الجديدة والقيم السابقة إدخالها تنزاح إلى العقد الجديدة.



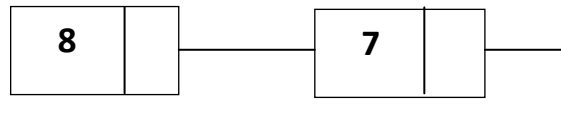
(a)



(b)



(c)



(d)

Insertion (Node *P)

```
{  
    P=get Node;  
    P->link=f;
```

```
P=f;
```

```
f->info=8;
```

```
}
```