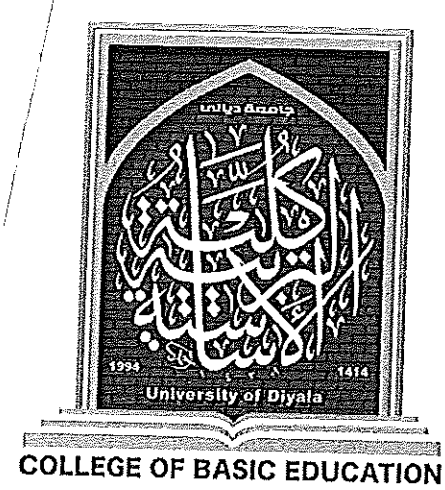


جامعة ديالى

كلية التربية الاساسية

قسم الحاسبات

المرحلة الثانية

٥٣٨

هياكل بيانات

الكورس الاول

م.م. زينب قحطان محمد

١٤٣٩هـ

٢٠١٧م

الفصل الاول
مبادئ أساسية

1 – 1 المقدمة	Introduction
1 – 2 هياكل البيانات	Data Structures
1 – 3 أنواع هياكل البيانات	Types of Data Structures
1 – 4 كيفية اختيار الهيكل البياني	Selection of Data Structures

1 - 1 مقدمة

من العوامل المهمة في معالجة البيانات والحصول على النتائج المطلوبة بطرق كفوءة هو ضرورة معرفة طرق تمثيلها وأساليب التعامل مع هياكلها التمثيلية لذا فأن هياكل البيانات لا تعني تمثيل البيانات في هياكل معينة بل قياس متطلباتها من حيث المساحة الخزنية (space) والوقت (time) إذ أن لكل طريقة مزايا تختلف عن غيرها مما يستوجب اختيار المناسب منها وفق التطبيق المعني .

تنقسم الهياكل إلى نوعين الأول هو الهيكل الفيزياوي ويقصد به المادي أو الحيز الذي تخزن أو تمثل فيه البيانات في ذاكرة الحاسوب (memory) التي نتعامل معها بصورة مصفوفة أحادية من المواقع الخزنية .

أما الهيكل الثاني فهو الهيكل المنطقي وهو الشكل البرمجي أو الأسلوب الذي يتعامل به المبرمج مع تلك البيانات.

فمثلاً عند تعريف مصفوفة ثنائية [1..5 , 1..4] A فأنها تمثل في ذاكرة الحاسوب في (20) موقع متعاقب ، والمبرمج عند استخدامه أو عند تعامله مع بيانات هذه المصفوفة باعتبارها مكونة من أربعة صفوف وخمسة أعمدة كما نتعامل معها رياضياً ، فالوصول إلى الموقع [2,3] A لا يعني البحث فيزياوياً في الصف الثاني والعمود الثالث لأن مثل هذه الصورة غير موجودة فيزياوياً بل يجب البحث عن الموقع الثامن (بافتراض استخدام طريقة الصفوف لتمثيل المصفوفة في لغة باسكال) ابتداءاً من أول موقع حدد لتمثيل المصفوفة أي أن المبرمج لم يكن معنياً بكيفية تمثيل بيانات المصفوفة في ذاكرة الحاسوب (التمثيل الفيزياوي) واستخدم خوارزمية الوصول إلى العناصر البيانية للمصفوفة بصيغ برمجية معينة للتوصل إلى الحل .

ان وجهة نظر المبرمج هنا تمثل الهيكل المنطقي ، والترابط بين وجهة نظر المبرمج مع الهيكل الفيزياوي الفعلي فتعالجه لغة البرمجة .

2-1 هياكل البيانات Data Structures

يمكن تعريف هياكل البيانات بأنها :

دراسة طرق الترابط بين نظرة المبرمجين للبيانات وعلاقة المعلومات بالأجهزة (وخصوصا ذاكرة الحاسوب التي تخزن فيها البيانات) .

فهياكل البيانات تشمل طرق تنظيم المعلومات ، والخوارزميات الكفوءة في الوصول لها وطرق التعامل معها أو تداولها (كالإضافة والحذف والتحديث والترتيب والبحث ... الخ) لذا فأن الاهتمام لا ينحصر فقط بأساليب الخزن وخوارزمياته لأن الأهمية الحيوية هي قياس كلفة كل أسلوب من تلك الأساليب ومدى ملأمة استخدامها في الحالات المختلفة .

3 – 1 أنواع هياكل البيانات

توفر لغات البرمجة الصيغ المناسبة لتعريف واستخدام العناصر البيانية ذات القيمة الواحدة (المنفردة) فمثلا في لغة باسكال تستخدم التعريفات :

X : Integer
Y : Real
A : Char
P : Boolean
S : String

لتمثل في ذاكرة الحاسوب ويتم التعامل معها بصيغ برمجية بسيطة مثل :

X : = X + 100
Y : = Y + 15.6

وتكاد تكون هذه الصيغ متوفرة في جميع لغات البرمجة بشكل قياسي شبه موحد.

أما بالنسبة للعناصر البيانية التي تتكون من عدة قيم بيانية فأنها تحتاج لاستخدام هيكل بياني مختلف وفيما يلي ذكر لأهم تلك الهياكل البيانية

1- المصفوفة Array

2- القيـــــد Record

3- الملف File

4- الهياكل الخطية Linear Structures

+ الهياكل غير الموصولة Non – linked structures

+ المكدس Stack

+ الطابور Queue

+ الطابور الدائري Circular Queue

+ الهياكل الموصولة Linked Structures

+ المكدس الموصول Linked Stack

+ الطابور الموصول Linked queue

5- الهياكل غير الخطية Non – Linear Structures

+ المخططات Graphs

+ المخطط المتجه Directed graph

+ هيكـل الشجرة Tree structure

+ المخطط غير المتجه Undirected graph

4 - 1 كيفية اختيار الهيكل البياني المناسب

لكل مجموعة من البيانات هنالك اكثر من طريقة لتنظيمها ووضعها في هيكل بياني معين ويتحدد ذلك وفق عدد من العوامل والاعتبارات لاختيار الهيكل البياني المناسب وهي :

1- حجم البيانات
2- سرعة وطريقة استخدام البيانات
3- الطبيعة الديناميكية للبيانات كتغييرها وتعديلها دورياً
4- السعة الخزنية المطلوبة
5- الزمن اللازم لاسترجاع أية معلومة من الهيكل البياني
6- أسلوب البرمجة

الفصل الثاني

المصفوفة Array

2 – 1 المصفوفة Array

2 – 2 تمثيل المصفوفة الأحادية في الذاكرة

Representation of One – Dimensional Array

2 - 3 تمثيل المصفوفة الثنائية في الذاكرة

Representation of Two – Dimensional Array

2- 3 – 1 طريقة الصفوف Row - wise

2 - 3 –2 طريقة الأعمدة Column - wise

2 - 4 تمثيل المصفوفات الثلاثية والرباعية الأبعاد

Representation of Three & Four – Dimensional Arrays

2-1 المصفوفة Array

هي عبارة عن مجموعة من المواقع الخزنيـة فـي الذاكـرة تستخدم وتتصف بمـا يأتي:

1- جميع المواقع تكون من نوع بياني واحد ، حسب صيغة التعريف , char , real integer , Boolean , ... الخ .

2- يمكن الوصول عشوائياً (Randomly accessed) إلـى أي موقـع مـن مواقعـها دون الاعتماد على أي موقع في المصفوفة فمقدار الوقت المطلوب للوصول إلى أي موقع هو مقدار ثابت.

3- مواقع عناصر المصفوفة تبقى ثابتة ولا تتغير أثناء التعامل مع أي مـن عنـاصر المصفوفة

4- تمثل المصفوفة في مواقع متعاقبة في الذاكرة

2-2 تمثيل المصفوفة الأحادية فى الذاكرة

في لغة باسكال تعرف هذه المصفوفة كالآتي :-

VAR X : array [1 .. N] of integer { or any other type }

وهذا يعني تعريف هيكل بياني يستوعب مجموعة من العناصر البيانية عددها (N) مثلا باسم بيـاني واحـد هـو (X) ويسـتخدم الدليـل (index) للوصـول إلـى العنصـر البياني المطلوب ، وتتراوح قيمة الدليل $1 <= I <= N$

وبموجـب هـذا التعريـف يحـدد مـترجم اللغـة (compiler) المنطقـة الخزنيـة لأستيعاب مجموعة العناصر البيانية ويكـون الموقـع الأول مخصصـا للعنصـر الأول في المصفوفة وهو ما يطلـق عليـه عنـوان البدايـة Base Address (BA) وليكـن افتراضا هو (500) أما العنصر الثاني للمصفوفة فيكـون عنوانـه بعـد عنـوان البدايـة مباشـرة أي (501) وهكـذا بقيـة العنـاصر بالتتـابع ويسـتخدم الدليـل (I) بقيمتـه التـي تتراوح بين $1 <= I <= N$ نسبة إلى موقع البداية (500) باستخدام العلاقة التالية:

Location (X [I]) = Base Address + (I – 1)

فإذا كان المطلوب تحديد عنوان (موقع) العنصر الرابع في المصفوفة أي $I = 4$ فإن :

n (X [4]) = 500 + (4 – 1)
= 500 + 3
= 503

أي أن موقع (عنوان) العنصر الرابع هو الخلية (503)

لأن العنصر الأول في الموقع 500
العنصر الثاني في الموقع 501
العنصر الثالث في الموقع 502
العنصر الرابع في الموقع 503

فعندما يتضمن البرنامج أية إشارة أو تعامل مع عناصر المصفوفة في أي إيعاز مثل Read (X[I]) , Write (X[I]) ، أو غيرهما فأن المترجم يعتمد العلاقة المشار إليها أعلاه لتحديد الموقع المطلوب

2 – 3 تمثيل المصفوفة الثنائية في الذاكرة

هنالك طريقتان لتمثيل المصفوفة الثنائية هما طريقة الصفوف (Row – wise method) ، وطريقة الأعمدة (Column – wise method) . لنأخذ التعريف التالي للمصفوفة

VAR A : array [1 .. M , 1 .. N] of integer { or any other type }

وهذا يعني تعريف هيكل بياني اسمه (A) يستوعب مجموعة من العناصر البيانية عددها (M * N) ويستخدم دليلين للوصول إلى العنصر البياني المطلوب وهما :

1 <= I <= M لتحديد الصف الذي فيه العنصر
1 <= J <= N لتحديد العمود الذي فيه العنصر

فمثلاً العنصر A [3 , 5] حيث I = 3 , J = 5

سيعني العنصر الذي يقع في السطر الثالث والعمود الخامس .

ويعتمد مترجم اللغة (compiler) إحدى الطريقتين الآتيتين لتمثيل هذه المصفوفة:-

2 – 3 – 1 طريقة الصفوف Row – wise Method

وهذه مستخدمة في لغة باسكال ، كوبول ... الخ

حيث تؤخذ جميع عناصر الصف الأول (I = 1) للمصفوفة وتخزن في الذاكرة ابتداءاً من موقع البداية (Base Address) وليكن 700.

فالعنصر A[1,1] يخزن في الموقع BA أي 700
والعنصر A[1,2] يخزن في الموقع BA + 1 أي 701
والعنصر A[1,3] يخزن في الموقع BA + 2 أي 702

ثم تؤخذ جميع عناصر الصف الثاني (I = 2) للمصفوفة وتخزن في الذاكرة ابتداءاً من الموقع الذي يلي أخر مواقع الصف الأول .

وتخزن جميع عناصر الصف الثالث (I = 3) للمصفوفة في الذاكرة ابتداءاً من الموقع الذي يلي موقع أخر عنصر من عناصر الصف الثاني وهكذا ..

ولهذا فأن احتساب موقع العنصر A [i , j] يكون وفق العلاقة التالية :-

$$\text{Location}(A[i,j]) = \text{Base Address} + N * (i-1) + (j-1)$$

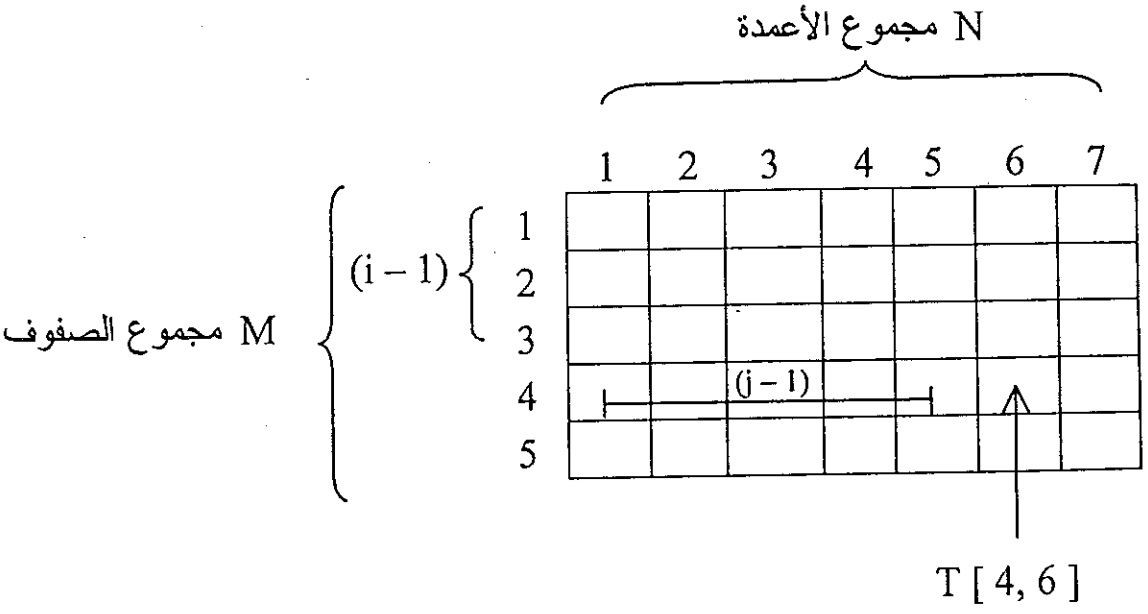
- Base Address: مجموع أعمدة المصفوفة
- $N * (i-1)$: عدد الصفوف السابقة لموقع العنصر المطلوب
- $(j-1)$: عدد الأعمدة السابقة لموقع العنصر المطلوب

وهذه العلاقة هي التي يحتسب المترجم بموجبها موقع العنصر المطلوب معالجته بموجب كل إيعاز من ايعازات البرنامج

مثال : لدينا تعريف المصفوفة

```
VAR   T : array [ 1 .. 5 , 1 .. 7 ] of integer
```

أحسب موقع العنصر T [4 ,6] بافتراض أن عنوان البداية BA = 900



الشكل (2 – 1)

بما أن المطلوب هو العنصر T [4 , 6] فهذا يعني أن العنصر يقع في الصف الرابع (I = 4) والعمود السادس (J = 6) وبما أن مجموع صفوف المصفوفة (M = 5) ومجموع أعمدة المصفوفة (N = 7)

لذا تصبح العلاقة عند التعويض فيها كما يأتي :-

$$\text{Location}(T[4,6]) = BA + 7 * (4-1) + (6-1)$$
$$= 900 + 7 * 3 + 5$$
$$= 900 + 21 + 5$$
$$= 926$$

2 – 3 – 2 طريقة الأعمدة Column – Wise Method

وهي مستخدمة في لغة فورتران ، بيسك ، ... الخ

إذ تؤخذ جميع عناصر العمود الأول (J =1) للمصفوفة وتخزن في الذاكرة ابتداءاً من موقع البداية (Base Address) وليكن 200

فالعنصر A [1 , 1] يخزن في الموقع BA أي 200

والعنصر A [1 , 2] يخزن في الموقع BA+ 1 أي 201

والعنصر A [1 , 3] يخزن في الموقع BA+ 2 أي 202

وهكذا بقية عناصر العمود

ثم تؤخذ جميع عناصر العمود الثاني J = 2 للمصفوفة وتخزن في الذاكرة ابتداءاً من الموقع الذي يلي أخر مواقع العمود الأول

وتخزن جميع عناصر العمود الثالث (J = 3) للمصفوفة في الذاكرة ابتداءاً من الموقع الذي يلي موقع أخر عنصر من عناصر العمود الثاني وهكذا ..

وعليه فأن احتساب موقع العنصر A [i , j] يكون وفق العلاقة التالية :-

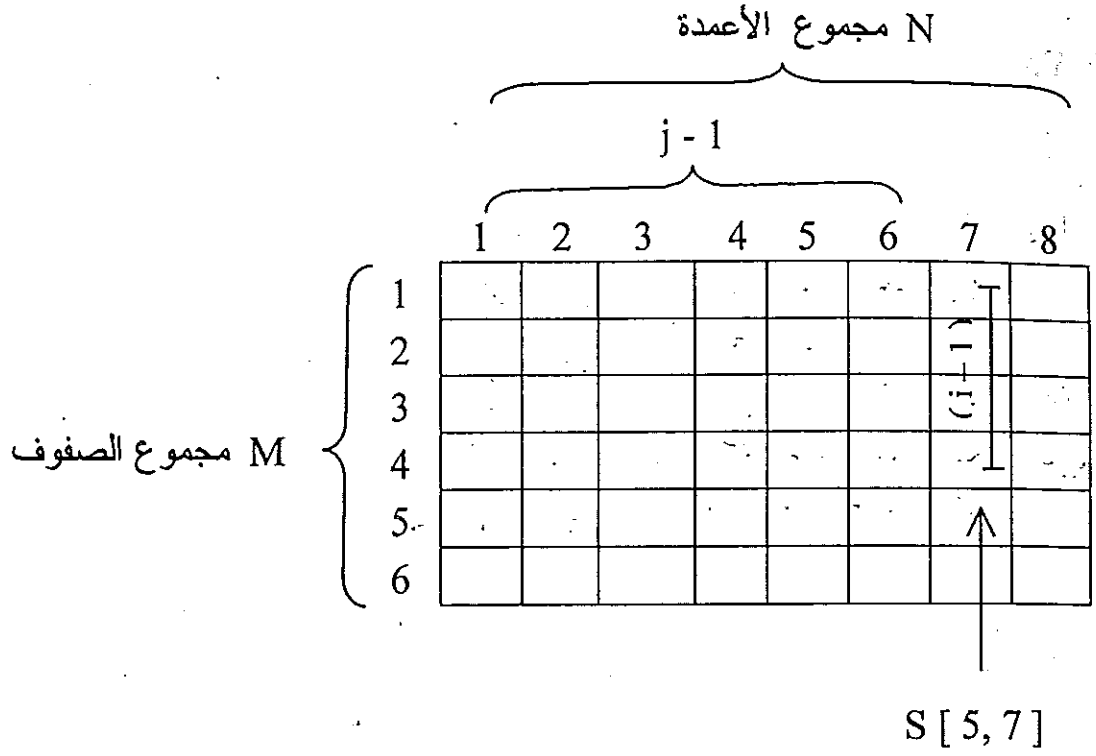
$$\text{Location}(A[i,j]) = \text{Base Address} + \underbrace{M}_{\text{مجموع صفوف المصفوفة}} * \underbrace{(j-1)}_{\substack{\text{عدد الأعمدة السابقة لموقع العنصر المطلوب}}} + \underbrace{(i-1)}_{\substack{\text{عدد الصفوف السابقة لموقع العنصر المطلوب}}}$$

مثال :-

لدينا تعريف المصفوفة

Var S : array [1 .. 6 , 1 .. 8] of integer

فما هو موقع العنصر S [5 , 7] عندما يكون عنوان البداية BA = 300



الشكل (2 – 2)

بما أن المطلوب هو العنصر S [5 , 7] فهذا يعني أن العنصر يقع في الصف الخامس (I = 5) والعمود السابع (J = 7) وبما أن مجموع صفوف المصفوفة (M = 6) ومجموع أعمدة المصفوفة (N = 8) فالعلاقة تصبح :-

Location (S [5 , 7]) = BA + 6 * (7 – 1) + (5 – 1)

= 300 + 6 * 6 + 4

= 300 + 36 + 4

= 340

2 – 4 تمثيل المصفوفات الثلاثية والرباعية الأبعاد

بنفس الطريقة يمكن وضع الصيغة العامة لتحديد موقع العنصر للمصفوفة ذات الأبعاد الثلاثة أو الأربعة

المصفوفة الثلاثية :

Var X : array [1 .. M , 1 .. N , 1 .. R] of integer .

يكون احتساب موقع العنصر X[i , j , k] كالآتي :-

طريقة الصفوف :

$$\text{Location}(X[i,j,k]) = BA + MN(k-1) + N(i-1) + (j-1)$$

طريقة الأعمدة :

$$\text{Location}(X[i,j,k]) = BA + MN(k-1) + M(j-1) + (i-1)$$

المصفوفة الرباعية :

Var Y : array [1 .. M , 1 .. N , 1 .. R , 1 .. P] of integer .

يكون احتساب موقع العنصر X[i , j , k , l] كالآتي :-

طريقة الصفوف :

$$\text{Location}(Y[i,j,k,l]) = BA + MNR(l-1) + MN(k-1) + N(i-1) + (j-1)$$

طريقة الأعمدة :

$$\text{Location}(Y[i,j,k,l]) = BA + MNR(l-1) + MN(k-1) + M(j-1) + (i-1)$$

تمرين : لديك المصفوفة الثلاثية التالية :-

TAB : array [1 .. 8 , 1 .. 5 , 1 ..7] of integer

احسب موقع العنصر TAB [5 , 3 , 6] في كل من طريقة الصفوف وطريقة الأعمدة إذا كان عنوان البداية Base Address = 900

الحل :

The dimensions of TAB are :

M = 8 , N = 5 , R = 7

To compute the location of the element TAB [5 , 3 , 6]

This means the indices are :

i = 5 , j = 3 , k = 6

طريقة الصفوف: **Row – Wise**

Location (TAB [i , j , k]) = BA + MN (k – 1) + N (i – 1) + (j – 1)

Location (TAB[5 , 3 , 6]) = 900 + 8 * 5 * (6 – 1) + 5 * (5 – 1) +(3 –1)

= 900 + 40 * 5 + 5 * 4 + 2

= 900 + 200 + 20 + 2

= 1122

طريقة الأعمدة: **Column – Wise**

Location (TAB [i , j , k]) = BA + MN (k – 1) + M (j – 1) + (i – 1)

Location (TAB[5 , 3 , 6]) = 900 + 8 * 5 * (6 – 1) + 8 * (3 – 1) +(5– 1)

= 900 + 40 * 5 + 8 * 2 + 4

= 900 + 200 + 16 + 4

= 1120

تمرين : لديك المصفوفة الرباعية التالية :-

VAR BOB : array [1 .. 4 , 1 .. 9 , 1 ..6 , 1 .. 8] of integer

احسب موقع العنصر BOB [3 , 7 , 4 , 5] في كل من طريقة الصفوف وطريقة الأعمدة إذا كان عنوان البداية Base Address = 415

الحل :

The dimensions of BOB are :

$M = 4 , N = 9 , R = 6 , P = 8$

To compute the location of the element BOB [3 , 7 , 4 , 5]

This means the indices are :

$i = 3 , j = 7 , k = 6 , L = 5$

Row – Wise

طريقة الصفوف:

$$\text{Location (BOB [i , j , k , l])} = BA + MNR\,(l-1) + MN\,(k-1) + N(i-1) + (j-1)$$

$$\text{Location (BOB [3 , 7 , 4 , 5])} = 415 + 4*9*6*(5-1) + 4*9*(4-1) + 9*(3-1) + (7-1)$$

$$= 415 + 216*4 + 36*3 + 9*2 + 6$$

$$= 415 + 864 + 108 + 18 + 6$$

$$= 1411$$

Column – Wise

طريقة الأعمدة:

$$\text{Location (BOB [i , j , k , l])} = BA + MNR\,(l-1) + MN\,(k-1) + M(j-1) + (i-1)$$

$$\text{Location (BOB [3 , 7 , 4 , 5])} = 415 + 4*9*6*(5-1) + 4*9*(4-1) + 4*(7-1) + (3-1)$$

$$= 415 + 216*4 + 36*3 + 4*6 + 2$$

$$= 415 + 864 + 108 + 24 + 2$$

$$= 1413$$

تمرين : لديك المصفوفة التالية :

VAR X : array [1..6,0..8] of char

أحسب موقع العنصر X[5,4] باستخدام طريقة الصفوف Row - wise method إذا كان عنوان البداية (BA = 100) .

الحل :

ان مجموع الصفوف (M) هو (6) لأنه معرف من (1) إلى (6)

أما مجموع الأعمدة (N) فهو (9) لأنه معرف من (0) إلى (8)

وعليه فأن صيغة احتساب موقع العنصر بطريقة الصفوف هي :

$$Loc(X[i,j]) = BA + M\,(i-1) + (j-1)$$

وسيكون تعويض القيم فيها كالأتي :

$$Loc(X[5,4]) = 100 + 6 * (5 - 1) + (4 - 0)$$

لاحظ هنا صيغة التعويض هي (0 - 4) وليس (1 - 4) لأن تعريف مجموع أعمدة المصفوفة هو[0..8] وليس [1..8] أي أن حساب الفرق (عدد الأعمدة السابقة للموقع) يكون (i - L) وعدد الصفوف السابقة للموقع فهو (j - L) حيث أن (L) هو الحد الأدنى (Lower Bound) الذي ورد في تعريف ذلك البعد .

وعليه فأن الناتج النهائي هو :

$$Loc(X[5,4]) = 100 + 6 * 4 + 4$$
$$= 100 + 24 + 4 = 128$$

ملاحظة مهمة :

عندما يكون تعريف المصفوفة يتضمن أبعاد سالبة بصيغة :

Var A : array [-2 ..4 , -3 .. 5] of char

فأن مجموع الصفوف (M) هو (7) لأنه معرف من (-2) إلى (4)

أما مجموع الأعمدة (N) فهو (9) لأنه معرف من (-3) إلى (5)

وعند احتساب موقع العنصر A[3,4] , أي أن $i = 3$, $j = 4$

فأن عدد الصفوف السابقة للموقع $(i - L) = [3 - (-2)] = 3 + 2 = 5$

لأن (L = -2).

وعدد الأعمدة السابقة للموقع $(j - L) = [4 - (-3)] = 4 + 3 = 7$

لأن (L = -3) .

* يعتمد نفس الأسلوب في المصفوفات ذات الأبعاد الأخرى .

اسئلة الفصل

1- What do we mean by data structures? Explain that in detail
2- What are the classifications of data structures?
3- What are the main factors for selection the required data structure?
4- What are the characteristics of the array?
5- Let X : array [1 .. 50] of integer
What is the address of the element X[33] if the base address (BA = 970) ?
6- Let A : array [1 .. M , 1 .. N] of integer
How you can compute the address of the general element A[i , j] using row – wise method?
7- Let X : array [1 .. A , 1 .. B , 1 .. C] of integer
What is the address of the general element X [i , j , k] using column – wise method ?
8- Let S : array [1 .. 7 , 1 .. 10 , 1 .. 8] of char
Compute the location of the element S[5 , 9 , 3] using row – wise and column – wise methods when the base address is 1200 .
9- Let A : array [1 .. 9 , 1 .. 5 , 1 .. 8 , 1 .. 6] of integer
Compute the location of the element A[8 , 3 , 6 , 4] using row – wise and column – wise methods when the base address is 950 .

الفصل الثالث

المكدس والطابور

Linear List	القائمة الخطية	3 – 1
Types of linear lists	انواع القوائم الخطية	3 - 1-1
Stack	المكدس	3– 2
Array Representation of Stack	تمثيل المكدس باستخدام المصفوفة	3 - 1-2
Stack's Algorithms	خوارزميات المكدس	3 – 2-2
Stack's Subprograms	البرامج الفرعية لتنفيذ عمليات المكدس	3 – 2-3
Record Representation of Stack	تمثيل المكدس باستخدام القيد	3 – 2-4
Stack's Applications	اهم تطبيقات المكدس	3 – 2-5
Queue	الطابور	3 – 3
Array Representation of Queue	تمثيل الطابور باستخدام المصفوفة	3 – 3-1
Queue's Algorithms	خوارزميات الطابور	3 – 3-2
Queue's Subprograms	البرامج الفرعية لتنفيذ عمليات الطابور	3 – 3-3
Record Representation of Queue	تمثيل الطابور باستخدام القيد	3 – 3-4
Queue's Applications	تطبيقات الطابور	3 – 3-5
Circular Queue (CQ)	الطابور الدائري	3 - 4
Double Ended Queue	الطابور المزدوج	3 – 5

3 – 1 القائمة الخطية Linear List

هي مجموعة من العناصر البيانية (elements , nodes , items) المتسلسلة والمرتبة تربط عناصرها علاقة تجاور بحيث يسبق كل عنصر عنصراً أخر عدا العنصر الأول الذي لا يسبقه عنصر والعنصر الأخير الذي لا يليه عنصر فلو مثلنا كل عنصر على شكل عقدة (node) فأن القائمة تصبح مجموعة من العقد (n).

x [1] , x [2] , x [3] , ... x [k – 1] , x [k] , x [k +1] , ..., x [n]

فالعقدة الأولى هي x[1] والعقدة الأخيرة هي x[n] أما العقدة x [k] عندما $1 <= k <= n$ فان العقدة التي تسبقها هي x [k-1] والتي تليها هي x [k+1]

أن كل مجموعة من البيانات والمعلومات يمكن تسميتها قائمة (list) فمثلاً :

+ مجموعة أسماء طلبة كلية ما مرتبة حسب الحروف.
+ مجموعة أسماء المشتركين في دليل الهاتف مرتبة وفق نسق معين .

3 – 1 – 1 أنواع القوائم الخطية

أ- القوائم غير الموصولة Non – linked lists

وهي القوائم التي لا تستخدم المؤشرات وتكون على شكل بيانات متتابعة ومتجاورة (sequential) وتستخدم المصفوفات في تمثيلها كما يستخدم هذا النوع عند معالجة البيانات التي لا تتعرض للتغيير كثيرا لصعوبة عمليات الحذف والإضافة إذ قد تكون المواقع التالية في ذاكرة الحاسوب مشغولة أصلا مما يتعذر استخدامها لأغراض الحذف والإضافة

ب - القوائم الموصولة Linked lists

وهي القوائم التي تستخدم المؤشرات (pointers) لتسهيل عمليات الإضافة والحذف والتعديل إذ يكون لكل عنصر مؤشر يحدد موقع العنصر التالي ، ووجود المؤشرات يلغي الحاجة لخزن بيانات القائمة في مواقع خزنية متجاورة .

3 – 1 – 2 العمليات التي يمكن إجراؤها على القوائم الخطية
The Operations on the linear lists

يمكن تنفيذ عدد من العمليات (الفعاليات) على أي هيكل بياني عند معالجة بياناته وفيما يلي أهم أنواع هذه العمليات التي يمكن تنفيذ بعضها أو كلها حسب التطبيق .

1- البحث Search

هي عملية بحث داخل الهيكل البياني بقصد الوصول إلى عنصر (عقدة) معين فيه بموجب قيمة أحد الحقول يسمى حقل المفتاح (key field) أي أن البحث يتم وفق المحتويات وليس العنوان .

2- إدخال (إضافة) Addition (Insertion)

لإضافة عنصر (عقدة)جديد إلى الهيكل البياني مثل تسجيل طالب جديد في المدرسة .

3- حذف Deletion

حذف عنصر (عقدة) من الهيكل البياني ، مثل نقل طالب إلى مدرسة أخرى .

4- دمج Merge

دمج بيانات هيكلين أو اكثر لتكوين هيكل بياني واحد .

5- فصل Split

تجزئة بيانات هيكل بياني إلى هيكلين أو اكثر .

6-احتساب Counting

احتساب عدد العناصر أو العقد في الهيكل البياني .

7- نسخ Copying

نسخ بيانات الهيكل البياني إلى هيكل بياني أخر .

8- ترتيب Sort

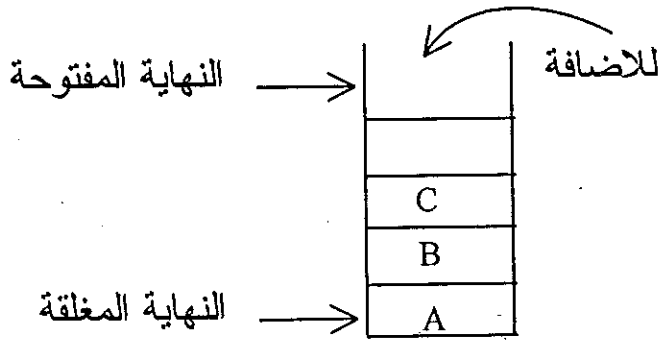
ترتيب عناصر (عقد) الهيكل البياني وفق قيمة حقل (field) أو مجموعة حقول.

9- الوصول Access

تتطلب أحيانا الحاجة للوصول إلى عنصر (عقدة) بياني في الهيكل البياني لعدة أغراض لاختباره مثلاً أو تغييره ... الخ.

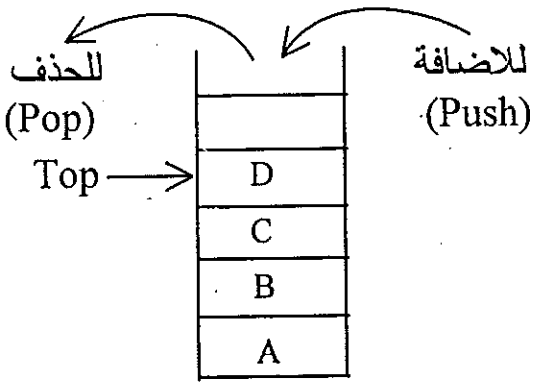
2-3 المكدس Stack

هو عبارة عن قائمة خطية تتم فيها عمليتي الإضافة والحذف من إحدى نهايتي القائمة وتكون النهاية الأخرى مغلقة.



الشكل (3 – 1)

لنأخذ المكدس الموضح في الشكل إذ نجده يحتوي على العناصر A,B,C وعند إضافة عنصر جديد مثل (D) يجب أن تكون الإضافة من الجهة المفتوحة ليصبح الشكل كالآتي :



وعند حذف عنصر من المكدس يجب أن تستخدم نفس الجهة المفتوحة فقط، أي نستطيع أن نأخذ العنصر (D) ثم نأخذ العنصر (C) بالتتابع ولا نستطيع أن نأخذ العنصر (C) قبل أن نأخذ العنصر (D) ، مع ملاحظة آن العنصر (D) دخل أخيرا.

ولهذا نستطيع أن نلخص عمل المكدس بالعبارة الآتية :

(آخر من يدخل أول من يخرج) Last In First Out (LIFO)

كما انه لايمكن اخذ (حذف) عنصر من وسط عناصر المكدس إلا بعد حذف (إخراج) العناصر التي تسبقه من جهة النهاية المفتوحة مع التأكيد على أن النهاية الأخرى مغلقة ولا تستخدم أبدا.

وتسمى عملية الإضافة إلـى المكـدس (push) أو (Insertion) وعمليـة الحـذف من المكدس (pop) أو (Deletion) .

مثال :

نفرض (S) تعني (Stacking) أي ترمز لعملية إضافة عنصـر إلـى المكـدس و (U) تعنـي (Unstacking) أي ترمـز لعمليـة حـذف عنصـر مـن المكـدس وكـانت مجموعة المدخلات للمكدس بالترتيب R,N,Y,B,M, بين ما هي المخرجات بعد تنفيذ كل سلسلة من العمليات التالية :

أ / SSUUSUSUSU

ب / SSSUSUUSUU

الحل :

يقصد بترتيب المدخلات انه عنـد تنفيذ عمليـة إدخـال عنصـر إلـى المكـدس فـان اختيار العنصر يكون من تلك المدخلات بالتتابع أي نأخذ M أولا ثم B ،... وهكذا، ولا نستطيع اخذ العنصر N قبل العناصر السابقة له .

أ -

المدخلات →	M	B			Y		N		R	
سلسلة العمليات →	S	S	U	U	S	U	S	U	S	U
المخرجات →			B	M		Y		N		R

ب -

المدخلات →	M	B	Y		N			R		
سلسلة العمليات →	S	S	S	U	S	U	U	S	U	U
المخرجات →				Y		N	B		R	M

مثال :

إذا كانت مجموعة مدخلات المكدس بترتيب 5,4,3,2,1 بين أياً من المخرجات المبينة أدناه صحيحة وفق أسلوب عمل المكدس ..

أ- 2 4 5 3 1

ب- 4 2 3 1 5

جـ- 4 5 1 2 3

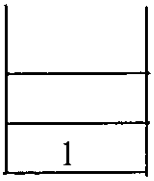
د - 4 3 5 2 1

الحل :

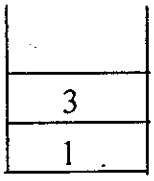
الفرع أ: المخرجات المطلوبة (2,4,5,3,1)

لاخراج العنصر (2) يجب أولا إدخال العنصرين 2,1 أي أن تسلسل تنفيذ العمليات هو SSU

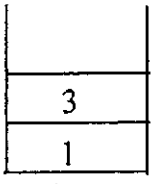
أي أن محتويات المكدس تصبح



ولاخراج العنصر (4) بعد العنصر (2) يجب إدخال العنصرين 4,3 أي أن تسلسل تنفيذ العمليات في هذه الحالة هو SSUSSU وتصبح محتويات المكدس :



ولاخراج العنصر (5) بعد العنصر (4) يجب إدخاله أولا ثم إخراجه أي أن تسلسل تنفيذ العمليات يكون SSUSSUSU وتصبح محتويات المكدس :

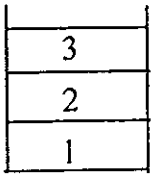


وفق حالة المكدس الحالية يمكن إخراج العنصرين 1,3 بالتتابع أي أن تسلسل تنفيذ العمليات هو SSUSSUSUUU .

إذن يمكن الحصول على مثل هذه المخرجات إذا كان تسلسل العمليات بالصيغة الأخيرة مع الالتزام بترتيب المدخلات .

الفرع ب : المخرجات المطلوبة (4, 2,3,1, 5)

لاخراج العنصر (4) يجب أولا إدخال العناصر 4,3,2,1 وفق سلسلة العمليات SSSSU وتصبح محتويات المكدس :



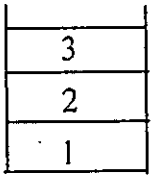
ولاخراج العنصر (2) من المكدس بحالته الحالية يجب إخراج العنصر (3) قبله لذا فان هذا التسلسل من المخرجات (4,2,3,1 ,5) لا يمكن تنفيذه .

الفرع ج: المخرجات المطلوبة (4,5,1,2,3)

يمكن إخراج العنصرين 5,4 بعد تنفيذ سلسلة العمليات الآتية :

المدخلات →	1	2	3	4		5	
سلسلة العمليات →	S	S	S	S	U	S	U
المخرجات →					4		5

وستصبح محتويات المكدس :



وهنا سيتعذر إخراج العنصر (1) قبل العنصرين (2,3) لذا فان تسلسل المخرجات (4,5,1,2,3) غير صحيح .

الفرع د : المخرجات المطلوبة (4,3,5,2,1)

يمكن الحصول على هذه المخرجات عند تنفيذ عمليات الإدخال والإخراج بالتسلسل الآتي :

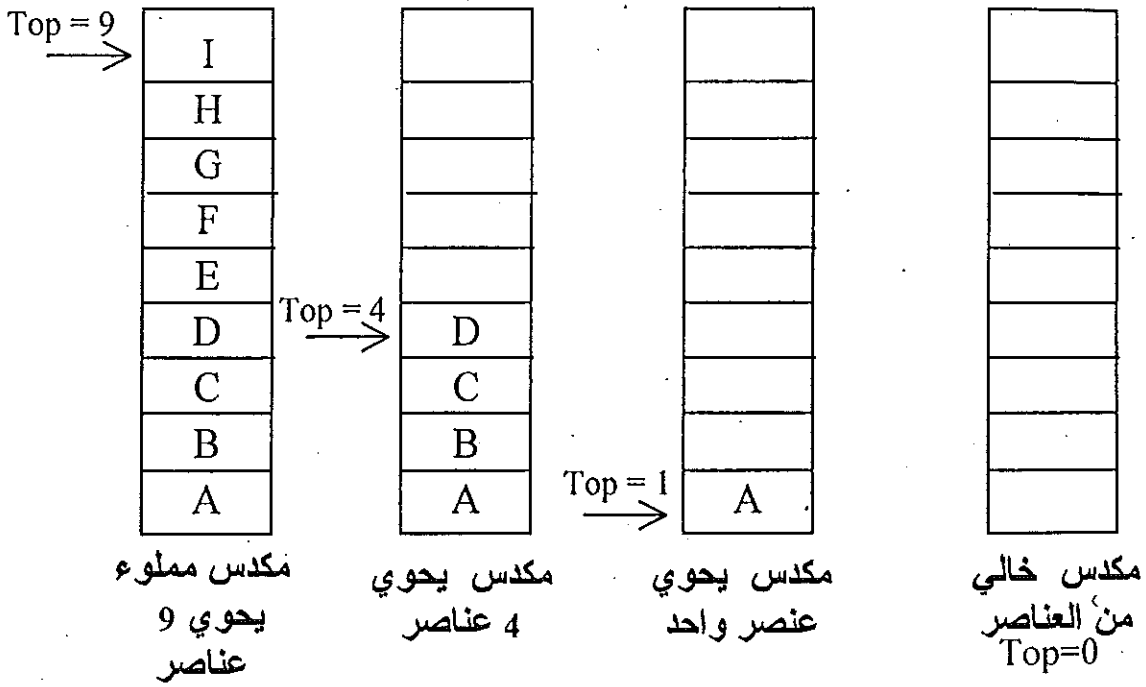
المدخلات →	1	2	3	4			5			
سلسلة العمليات →	S	S	S	S	U	U	S	U	U	U
المخرجات →					4	3		5	2	1

3-2-1 تمثيل المكدس باستخدام المصفوفة

Array Representation of Stack

يمكن تطبيق المكدس باستخدام مصفوفة أحادية بالسعة المطلوبة (size) وبالنوع المناسب للبيانات (Data Type) التي ستخزن فيه (Real , integer ... الخ) مع استخدام متغير مستقل يدعى (Top) يستعمل كمؤشر يشير إلى موقع أعلى عنصر في المكدس (موقع اقرب عنصر إلى النهاية المفتوحة) وابتداء تكون قيمة المؤشر (Top := 0) عندما يكون المكدس خاليا من العناصر ، ويعرف المكدس برمجيا كالآتي:-

```
Const   Size = 9 ;  { or  any other value }
Type    Stackelement = integer  ; {  or any other type }
        St = array [ 1 .. size ]  of  stackelement ;
Var     stack : St ;
        Top : integer ;
```



الشكل (3- 3)

عملية الإضافة للمكدس (Push)

لتنفيذ عملية الإضافة بشكل صحيح نتبع الخطوات الآتية :

1- التحقق من كون المكدس غير مملوء (not full) أي أن المؤشر (Top <> Size) لتجنب حالة الفيض (over flow) وتعذر تنفيذ عملية الإضافة .

2- تحديث قيمة المؤشر Top : = Top + 1 ليشير إلى الموقع التالي .

3- إضافة العنصر الجديد في الموقع الجديد Stack [Top]

عملية الحذف من المكدس (Pop)

أن تنفيذ عملية حذف أي عنصر من المكدس يجب أن تكون وفق الخطوات الآتية:

1- التحقق بأن المكدس غير خالٍ (not Empty) أي أن المؤشر Top<>0 لتجنب حالة الغيض (under flow) وتعذر تنفيذ عملية الحذف .

2- اخذ العنصر من الموقع الذي يشير إليه (Top) وخزنه وقتياً في متغير مستقل

Item : = Stack [Top]

3- تحديث قيمة المؤشر Top : = Top – 1 ليشير إلى موقع العنصر التالي للعنصر الذي حذف .

ملاحظة :

يتضح أعلاه أن الخطوتين 2 ، 3 في عملية الحذف معكوسة الترتيب عنها في عملية الإضافة .

3 – 2 – 2 خوارزميات المكدس Stack's Algorithms

يمكن تصميم مجموعة من الخوارزميات لتغطية فعاليات المكدس ومن ثم برمجتها لتمثيلها عملياً .

1- خوارزمية الإضافة Push Algorithm

```
If      Stack is full
Then    Overflow ← True
Else
        Overflow ← false
        Top ← Top + 1
        Stack [ Top ] ← New element
```

2- خوارزمية الحذف POP Algorithm

```
If      Stack is Empty
Then    under flow ⟵ True
Else
        under flow ⟵ false
        element ⟵ stack [ Top ]
        Top ⟵ Top - 1
```

3- خوارزمية ملء المكدس Stack full

هذه الخوارزمية للتحقق من هل المكدس مملوء أم لا اعتماداً على قيمة المؤشر (Top) قبل عمليات الإضافة

```
If      Top = size
Then    stackfull ⟵ True
```

4- خوارزمية خلو المكدس Stack Empty

هذه الخوارزمية للتحقق من هل أن المكدس خال أم لا اعتماداً على قيمة المؤشر (Top) قبل عمليات الحذف

```
If      Top = 0
Then    stackempty ⟵ True
```

5- خوارزمية إخلاء المكدس ClearStack

هذه الخوارزمية تستخدم لغرض تهيئة المكدس وإخلائه من العناصر بجعل قيمة المؤشر (Top : = 0)

```
Top ⟵ 0
```

3 – 2 – 3 البرامج الفرعية لتنفيذ عمليات المكدس

Stack's Procedures and Functions

أن تصميم برامج فرعية (functions , procedures) لكل فعالية أو عملية من عمليات المكدس تساعد على تبسيط وتوضيح كيفية برمجة تلك العمليات ومن ثم تجميعها في برنامج واحد تتوفر فيه صفات البرمجة المهيكلة ويكون واضحاً للقراءة وسهل الفهم والمتابعة والتحديث والتطوير .

ونفترض وجود التعريف التالي في مقدمة البرنامج لتكون البرامج الفرعية اللاحقة صحيحة

```
Const   Size = 20 ; { or any other integer value }
Type    Stackelement = integer ; { or any other type }
        St = array [ 1 .. Size ] of stackelement ;
Var     Stack : St ;
        Top : integer ;
        Item : stackelement ;
```

1- برنامج فرعي لاخلاء المكدس

```
Procedure clearstack( Var Top : integer );
Begin
    Top : = 0
  End;
```

لاحظ عدم الحاجة للمرور على جميع مواقع المصفوفة وجعلها مساوية للصفر والاكتفاء فقط بجعل المؤشر (Top = 0) وهذا البرنامج الفرعي يستدعى في بداية العمل لجعل المكدس خالياً .

2- برنامج فرعي للتحقق من امتلاء المكدس

```
Function  FullStack( Top : integer ) : Boolean;
Begin
    If      Top = Size
    Then    FullStack : = True
    Else    FullStack : = False
End ;
```

هذه الدالة (Function) مدخلها المؤشر (Top) وبموجب قيمته فأن المخرج هو المتغير المنطقي (FullStack) إذ تكون قيمته أما (True) عندما يكون المكدس مملوء وتكون قيمته (False) عندما يكون المكدس غير مملوء .

والبرنامج الفرعي (FullStack) يستدعى داخل البرنامج الفرعي (procedure push) لينفذ عملية الاضافة .

3- برنامج فرعي للتحقق من خلو المكدس

```
Function   EmptyStack( Top : integer ) : Boolean;
Begin
      If       Top = 0
      Then     EmptyStack : = True
      Else     EmptyStack : = False
End ;
```

هذه الدالة مدخلها المؤشر (Top) وبموجب قيمته فأن المخرج هو المتغير المنطقي (EmptyStack) وقيمته (True) عندما يكون المكدس خالياً و(False) عندما يكون المكدس غير خالٍ وهذا والبرنامج الفرعي (EmptyStack) يستدعى داخل البرنامج الفرعي (procedure pop) الذي ينفذ عملية الحذف .

4- برنامج فرعي لإضافة عنصر واحد إلى المكدس

```
Procedure  Push( Var Stack : St ; Var Top : integer ;
                                  Item :stackelement) ;
Begin
      If   Fullstack ( Top)
      Then  writeln ( ' Error … the Stack is Full ' )
      Else
          Begin
              Top : = Top  + 1 ;
              Stack [ Top] : = Item
          End
End ;
```

هذا البرنامج الفرعي يضيف عنصر واحد (Item) للمكدس ويمكن استدعائه في البرنامج الرئيسي (main Program) بأي عدد من المرات باستخدام أحد ايعازات التكرار مثل (For … Do) الذي يتضمن قراءة العنصر (Item) ثم استدعاء البرنامج الفرعي (push) لإضافته إلى المكدس .

أن هذه الصيغة تسمح باستدعائه في البرنامج الرئيسي في اكثر من موقع ولعدة مرات .

5- برنامج فرعي لحذف عنصر واحد من المكدس

```
Procedure  POP(Stack : St ; Var Top :integer ,
                                Var  Item: Stackelement ) ;
Begin
      If     Emptystack( Top)
      Then  writeln ( ' Error … the Stack is Empty ' )
      Else
          Begin
              Item : = Stack [ Top ] ;
              Top : = Top - 1
          End
End ;
```

هذا البرنامج الفرعي يأخذ العنصر الذي يشير إليه (Top) وينسخه في المتغير (Item) لاستخدامه لاحقاً بمعالجة معينة لتحقيق الغرض الذي من اجله سحب هذا العنصر من المكدس .

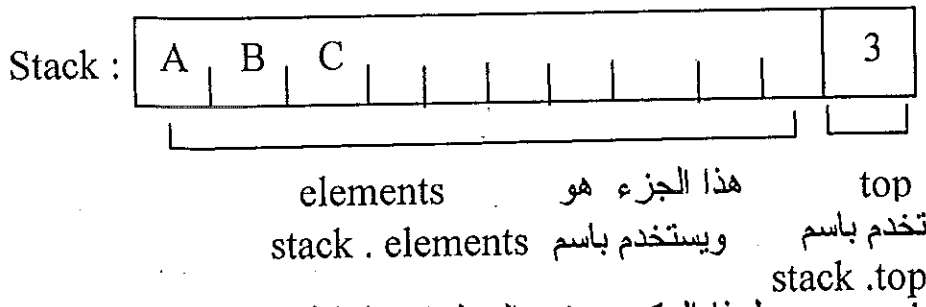
ولغرض حذف أو سحب اكثر من عنصر من المكدس بصورة متتابعة فان هذا البرنامج يستدعى بأي عدد من المرات وفي أي موقع من البرنامج الرئيسي .

4-2-3 تطبيق المكدس باستخدام القيد

Record Implementation of Stack

في التطبيق السابق باستخدام المصفوفة ورد تعريف المؤشر (Top) كمتغير مستقل عن المصفوفة التي تمثل المكدس ، إلا أننا هنا نستخدم القيد (Record) في تمثيلهما معاً كهيكل بياني واحد حيث يتكون القيد من جزأين الأول يمثل المكدس وهو على شكل مصفوفة والجزء الثاني هو حقل يمثل المؤشر (Top) ويعرف في لغة باسكال بالطريقة التالية :

```
Const    Size = 10 ;    { or any other value }
Type     Stelement = integer ; { or any other type }
         St = Record
         elements : array [ 1 .. size ] of  Stelement ;
         top : 0 .. size
         End ;
Var      Stack : St ;
         Item : Stelement ;
```



لإضافة عنصر جديد لهذا المكدس نتبع الخطوات التالية :

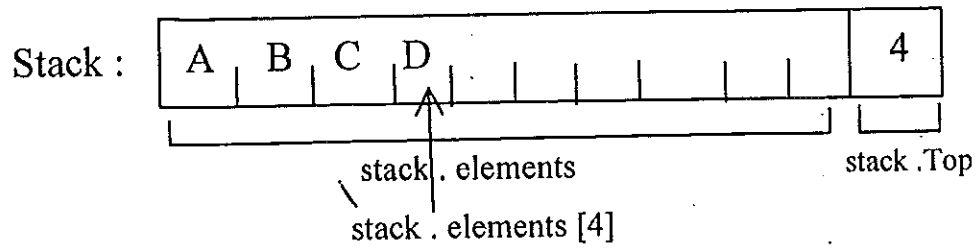
1- نحدث قيمة المؤشر (Top) الذي هو حقل في القيد stack ليصبح (4)

```
stack . top : = stack . top + 1
```

2- نضيف العنصر الجديد (D) في الموقع الجديد (4)

```
stack . elements[ stack . top ] : = D
```

وبهذا يصبح المكدس بالصورة التالية :



تمرين

تمرين : اعد كتابة البرنامج الفرعي(POP) لحذف عنصر من المكدس باستخدام القيد (Record Implementation) .

الحل

الحل :

سنعتمد التعريفات الواردة في الصفحة رقم (45)

```
Procedure  POP(stack :st ; Var Item : stelement ) ;
Begin
      IF      Emptystack(stack)
      THEN  Writeln( ' Error .. the stack is empty ')
      ELSE
            Begin
            Item : = stack . elements[stack . top] ;
            Stack . top : = stack . top - 1
            End ;
End ;
```

تض
داخل إ
أننا لا

تمرين : اكتب برنامجا فرعيا لإضافة ثلاثة عناصر من الأعداد الصحيحة إلى المكدس (SET) الذي سعته (20).

الحل : أن المكدس المطلوب يمثل بالمصفوفة (SET) وسعتها (20) ونوع البيانات (integer)

```
Type  St = array [ 1 .. 20 ] of integer ;
Procedure   Push3(Var  SET : St ; Var Top : integer ) ;
Var     I : integer ;
Begin
For  I := 1   TO  3    Do
        Begin
        If     Top  = 20
        Then   writeln ( ' the tack SET  is full ' )
        Else
                Begin
                Top : = Top + 1;
                Write ( ‘ Enter the element ‘ ) ;
                Readln ( Set [ Top ] )
                End
        End
End;
```

تضمن هذا البرنامج الفرعي (procedure) خطوة التحقق من امتلاء المكدس داخل إيعاز التكرار لينفذ عند كل عملية إضافة مع أن سعة المصفوفة (20) والسبب أننا لا نعرف عدد عناصر المكدس قبل الإضافة .

مثال : المكدس (TABLE) بسعة (30) عنصر يحتوي على أربعة عناصر A , B , C , D ، اكتب برنامجاً فرعياً (procedure) لإضافة (8) عناصر أخرى .

الحل : أن المكدس المطلوب يمثل بالمصفوفة (TABLE) وسعتها (30) ونوع بياناته هو (char)

```
Type     st = array [ 1 .. 30 ] of  char ;
Procedure  Push8(Var  TABLE : st ; Var  Top : integer ) ;
Var  I : integer ;
Begin
        Top : = 4 ;
        For   I : = 1   To   8    Do
        Begin
            Top : = Top + 1 ;
            Write ( ' Enter the new element ' ) ;
            Readln ( TABLE [ Top] )
        End
    End;
```

في هذا البرنامج الفرعي (procedure) لم نضع خطوة التحقق من امتلاء المكدس لكونها غير ضرورية لأن سعة المكدس هي (30) ويحتوي على أربعة عناصر فقط والإضافة المطلوبة هي (8) فقط لذا فإن المكدس لن يصل إلى حالة الامتلاء .

مثال : اكتب برنامج فرعي لحذف (4) أعداد حقيقة من المكدس (BOB) الذي سعته (15) عنصر

الحل : أن المكدس المطلوب يمثل المصفوفة (BOB) بسعة (15) عنصر ونوع البيانات (Real)

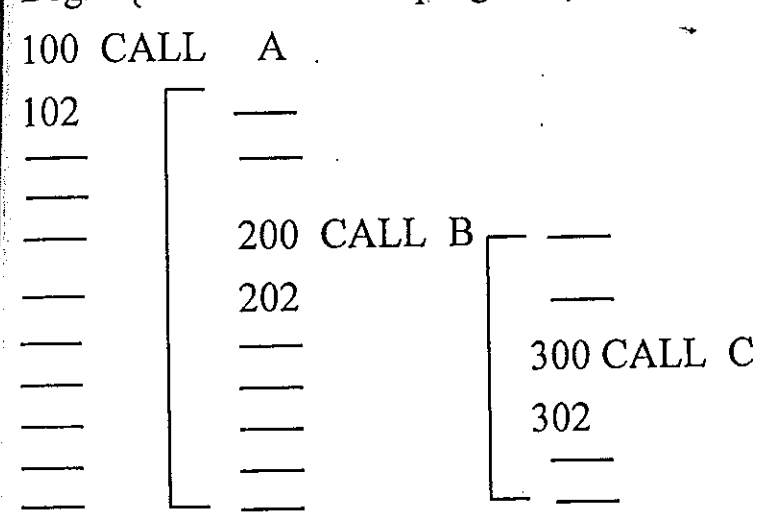
```
Type   St = array [ 1 .. 15 ] of real ;
Procedure   POP4( BOB : St; Var  Top : integer ; Var  Item : real ) ;
Var      I : integer ;
Begin
    For   I : = 1   to   4   Do
    Begin
          If   Top = 0
          Then   writeln ( ' Error … the tack is Empty ' )
          Else
                Begin
                     Item : = BOB [ Top ] ;
                     Top : = Top – 1
                End
     End
End ;
```

3-2-5 أهم تطبيقات المكدس

1- معالجة البرامج التي تحتوي على برامج فرعية

يستخدم المكدس بأهمية كبيرة من قبل المترجمات في معالجة البرامج التي تحتوي على برامج فرعية (functions & procedures) وتنظيم طريقة استدعائها وذلك بخزن عناوين الرجوع (Return Addresses) فعند استدعاء برنامج فرعي (procedure) أو (function) داخل البرنامج الرئيسي فأن ذلك يتطلب خزن عنوان الإيعاز التالي بعد إيعاز الاستدعاء لكي يستطيع البرنامج الرئيسي تنفيذ البرنامج الفرعي والعودة بشكل صحيح إلى موقع الخطوة أو الإيعاز التالي لأن عنوان هذا الموقع (Return – address) يكون مخزوناً في المكدس .

لنفترض أن البرنامج التالي الذي يتضمن استدعاء عدد من البرامج الفرعية هي C , B , A



نلاحظ في هذا المثال :

أ- أن البرنامج الرئيسي يستدعي البرنامج الفرعي (A) الذي بدوره وفي داخله يستدعي البرنامج الفرعي (B) وبداخله استدعاء البرنامج الفرعي (C) .

ب- لغرض التوضيح نفترض أن عناوين الايعازات كما يأتي :

أن عنوان إيعاز استدعاء (A) هو (100) أما عنوان الإيعاز التالي له (Ret. Add) فهو (102) وعنوان إيعاز استدعاء (B) هو (200) أما عنوان الإيعاز التالي له (Ret . Add) فهو (202) وعنوان إيعاز استدعاء (C) هو (300) أما عنوان الإيعاز التالي له (Ret . Add) فهو (302).

أن مترجم اللغة يستخدم المكدس في معالجة مثل هذا النوع من البرامج وبالطريقة التالية :

1- عند الوصول إلى استدعاء البرنامج الفرعي (A) وقبل تنفيذ الاستدعاء يخزن عنوان الرجوع (102) في المكدس بعملية (push).

2- عند تنفيذ ايعازات البرنامج الفرعي (A) نجده يتضمن إيعاز استدعاء البرنامج الفرعي (B) وهذا يتطلب قبل تنفيذ الاستدعاء خزن عنوان الرجوع (202) في المكدس بعملية (push) أخرى .

3- عند تنفيذ ايعازات البرنامج الفرعي (B) نجده يتضمن إيعاز استدعاء البرنامج الفرعي (C) وهذا يتطلب قبل تنفيذ الاستدعاء خزن عنوان الرجوع (302) في المكدس بعملية (push) أخرى .

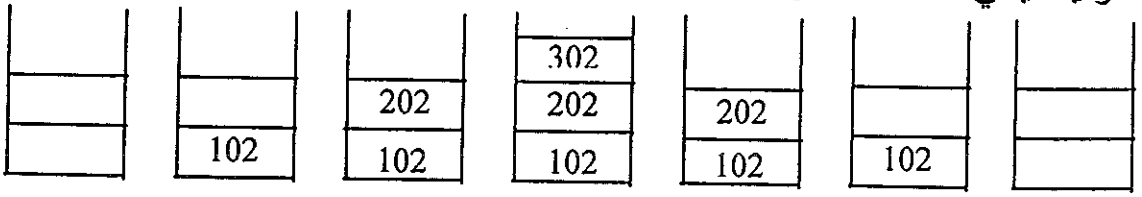
4- عند انتهاء تنفيذ البرنامج الفرعي (C) فأن البرنامج الرئيسي يحتاج معرفة عنوان الرجوع الذي سبق خزنه في المكدس ويتم ذلك من خلال تنفيذ عملية (POP) لاخراجه وتنفيذ الإيعاز الموجود في ذلك العنوان وما بعده داخل البرنامج الفرعي (B) .

5- عند انتهاء تنفيذ إيعازا البرنامج الفرعي (B) فأن البرنامج الرئيسي يحتاج معرفة عنوان الرجوع الذي سبق خزنه في المكدس ويتم ذلك من خلال عملية (POP) لاخراجه وتنفيذ الإيعاز الذي في ذلك العنوان وما بعده داخل البرنامج الفرعي (A) .

6- عند انتهاء تنفيذ ايعازات البرنامج الفرعي (A) فأن البرنامج الرئيسي يحتاج معرفة عنوان الرجوع الذي سبق خزنه في المكدس ويتم ذلك من خلال تنفيذ عملية (POP) لاخراجه وتنفيذ الإيعاز الذي في ذلك العنوان وما بعده داخل البرنامج الرئيسي .

7- يستمر البرنامج الرئيسي في تنفيذ الايعازات التالية بصورة اعتيادية بعد أن انتهت البرامج الفرعية ولم يعد المكدس يحوي شيئاً (أي خالياً) .

وفيما يأتي حالة المكدس بعد كل خطوة من الخطوات المشار إليها آنفاً:



البداية : المكدس خالي | بعد استدعاء A | بعد استدعاء B داخل A | بعد استدعاء C داخل B | بعد انتهاء تنفيذ C | بعد انتهاء تنفيذ B | بعد انتهاء تنفيذ A والعودة الى البرنامج الرئيسي

الشكل (3 – 4)

تمرين : وضح بالرسم جميع حالات المكدس عند تنفيذ البرنامج التالي :-

```
Begin { main program }

    100  CALL  X              |
    102   ——
    200  CALL   Y
    202   ——
    ——        400 CALL   P
    ——        402   ——
    ——        ——     600  CALL  R
    ——        ——     602   ——
    ——        ——     ——    700  CALL  S
    ——        ——     ——    702   ——
    ——        500 CALL  Q
    ——        502   ——
   300  CALL Z
    302   ——
End .
```

2 ـ استخـدام المكـدس في معـالجـة التعـابير الحسابيـة

Arithmetic expressions

من المعروف أن التعابير الحسابية تكتب بثلاث صيغ هي :

1- صيغة Infix notation

تكون إشارة العملية الحسابية تتوسط العوامل مثل :

A – B , 3 + 4 , X/20 وهذه هي الصيغة الاعتيادية .

2- صيغة Prefix Notation

إذ تكون إشارة العملية الحسابية تسبق العوامل مثل :

+ 3 4 , - A B , / X 20 وتسمى (Polish Notation)

3- صيغة Postfix Notation

إذ تكون إشارة العملية الحسابية تلحق العوامل مثل :

+ 34 , A B - , 20 X / وتسمى Reverse Polish Notation (RPN)

لأنها عكس الحالة الثانية (Polish Notation) .

ملاحظة :

لتنفيذ أي تعبير حسابي مكتوب بصيغة (infix) فأن العمليات تنفذ من اليسار إلى اليمين وحسب أعلى أسبقية للعملية الحسابية وهي :

نوع العملية الحسابية	الأسبقية
^ (power) , Unary (-) , Unary (+) , Not	4
* , / , AND , DIV , MOD	3
+ , - , OR	2
= , < , > , <> , <= , >=	1

وتستخدم الأقواس عند الحاجة إلى تغيير أسبقيات التنفيذ وتسلسل الخطوات .

أن البرامج التي تتضمن تعابير حسابية بصيغة (Infix) يقوم المترجم (compiler) بتحويلها إلى صيغة (postfix) باستخدام المكدس وفق الخوارزمية الآتية :

خوارزمية تحويل صيغة (infix) إلى (postfix) باستخدام مكدسين

1- نستخدم مكدسين ، المكدس الأول (ST1) لخزن المتغيرات (العوامل operands) وفي الخطوة الأخيرة ستتجمع فيه الصيغة النهائية (صيغة Postfix) والمكدس الثاني (ST2) يستخدم لخزن إشارات العمليات الحسابية (Operators) .

2- نفحص التعبير الحسابي رمزاً رمزاً من اليسار إلى اليمين .

3- عند كل رمز نقوم بما يأتي :-

إذا كان الرمز :	ينفذ ما يأتي :
+ أحد العوامل (operand)	+ يخزن (push) في المكدس (ST1) .
+ قوس ايسر	+ يخزن (push) في المكدس (ST2) .
+ قوس ايمن	+ إخراج (pop) جميع الرموز من المكدس (ST1) وخزنها (push) بالتتابع في المكدس (ST2) لغاية الوصول إلى القوس الأيسر الذي يجب إخراجه وإهماله مع القوس الأيمن .
+ عملية حسابية (operator)	+ إخراج (pop) جميع العمليات الحسابية (أن وجدت) في المكدس (ST2) التي أسبقيتها أعلى أو تساوي أسبقية العملية الحسابية الحالية وخزنها في المكدس (ST1) (التوقف عن ذلك عند عدم تحقق الشرط) ومن ثم خزن العملية الجديدة في المكدس (ST2).

4- عند انتهاء كل رموز التعبير الحسابي يتم إخراج (pop) جميع الرموز المتبقية في المكدس (ST2) بالتتابع وخزنها (push) في المكدس (ST1) الذي يحوي الصيغة النهائية (postfix).

مثال : حول العبارة الحسابية التالية من صيغة (infix) إلى صيغة (postfix) باستخدام مكدسين .

a – b * (c + d) / (e – f) ^ g * h

الحل :

رقم الخطوة	الرمز المدخل	المكدس الأول ST1	المكدس الثاني ST2
1	A	a	.....
2	-	a	-
3	B	ab	-
4	*	ab	- *
5	(	ab	- * (
6	C	abc	- * (
7	+	abc	- * (+
8	D	abcd	- * (+
9	)	abcd +	- *

نلاحظ هنا عند ورود القوس الأيمن يتم إخراج (نقل) جميع العمليات الحسابية لغاية القوس الأيسر من المكدس (ST2) إلى (ST1) مع إخراج القوس الأيسر ليهمل هو والقوس الأيمن .

10	/	abcd + *	- /
11	(	abcd + *	- / (
12	E	abcd + * e	- / (
13	-	abcd + * e	- / (-
14	F	abcd + * ef	- / (-
15	)	abcd + * ef-	- /
16	^	abcd + * ef-	- / ^
17	g	abcd + * ef-g	- / ^
18	*	abcd + * ef-g ^ /	- *

لأن أسبقية الضرب (*) = < أسبقية الرفع (^) والقسمة (/)

19	H	abcd + * ef-g ^ /h	- *

هنا انتهت جميع المدخلات لذا ينقل المتبقي في المكدس (ST2) إلى المكدس (ST1) بالتتابع ليصبح

20	-	abcd + * ef-g ^ /h* -	.....

تمرين
stfix)
الحل :

رقم الخط

خوارزمية تحويل صيغة (Infix) إلى (postfix) باستخدام مكدس واحد

1- نستخدم مكدس واحد (ST) لخزن إشارات العمليات الحسابيه (operators).
2- نفحص (نقرأ) التعبير الحسابي رمزا رمزا من اليسار إلى اليمين
3- عند كل رمز نقوم بما يأتي :-

إذا كان الرمز :	ينـفذ ما يأتي :
+ أحد العوامل (operand)	+ ينقل إلى جملة المخرجات output string
+ قوس ايسر	+ يخزن (push) في المكدس (ST) .
+ عملية حسابية (operator)	+ إخراج (pop) جميع العمليات الحسابية (إن وجدت) في المكدس (ST) التي أسبقيتها أعلى أو تساوي أسبقية العملية الحسابية الجديدة وأضافتها إلى جملة المخرجات (التوقف عن ذلك عند عدم تحقق الشرط) + بعد ذلك تخزن (push) إشارة العملية الحسابية الجديدة في المكدس (ST).
+ قوس ايمن	+ إخراج (pop) جميع إشارات العمليات الحسابيه من المكدس وإضافتها بالتتابع إلى جملة المخرجات لغاية الوصول إلى القوس الأيسر في المكدس الذي يجب إخراجه وإهماله مع القوس الأيمن المقابل له.

4- عند انتهاء فحص (المرور على) جميع رموز التعبير الحسابي يتم إخراج (pop) جميع الرموز المتبقية في المكدس (ST) بالتتابع وإضافتها إلى جملة المخرجات ليصبح الشكل النهائي لجملة المخرجات هو صيغة ال (postfix) المطلوبة .

تمرين : حول العبارة الحسابية التالية من صيغة (infix) إلى صيغة (postfix) باستخدام مكدس واحد .

y * m + (a ^ 3 / b - n) - d

الحل :

رقم الخطوة	الرمز المدخل	المكدس ST	Output string
1	Y	...	Y
2	*	*	Y
3	M	*	Ym
4	+	+	Ym*
5	(	+(	Ym*
6	A	+(	Ym*a
7	^	+(^	Ym*a
8	3	+(^	Ym*a3
9	/	+(/	Ym*a3^
10	B	+(/	Ym*a3^b
11	-	+(-	Ym*a3^b/
12	n	+(-	Ym*a3^b/n
13	)	+	Ym*a3^b/n-
14	-	-	Ym*a3^b/n-+
15	d	-	Ym*a3^b/n-+d
16	...	...	Ym*a3^b/n-+d-

احتساب قيمة (تنفيذ) التعبير الحسابي المحول إلى صيغة Postfix

بعد أن يحول المترجم العبارة الحسابية من صيغة (infix) إلى صيغة (postfix) فأن احتساب قيمتها في المرحلة التالية يكون بموجب الخوارزمية المبينة أدناه باستخدام مكدس واحد .

الخوارزمية

1- يستخدم مكدس واحد وليكن (ST) .

2- نفحص (نأخذ) التعبير الحسابي رمزا رمزا من اليسار إلى اليمين ويعامل كالآتي :

إذا كان الرمز المدخل هو :	ينفذ ما يأتي :
+ أحد العوامل (operand)	+ يخزن (push) في المكدس(ST) .
+ عملية حسابية (operator)	+ تنفذ هذه العملية على العاملين في أعلى المكدس : أي يتم إخراج (pop) العاملين من المكدس (ST) وتنفذ العملية عليهما وتخزن (push) النتيجة المتحققة في المكدس (ST) .

3- عند انتهاء مدخلات التعبير الحسابي فان القيمة المتبقية في المكدس هي النتيجة النهائية للعبارة الحسابية.

مثال : لنأخذ العبارة الحسابية المكتوبة بصيغة (infix) 7+8-6*3/2

عند تحويلها إلى صيغة (postfix) تصبح -/2*63+78

ولاحتساب قيمة هذه العبارة بصيغتها الأخيرة نطبق خطوات الخوارزمية كالآتي :

رقم الخطوة	المدخلات	محتويات المكدس ST
1	7	7
2	8	7 8
3	+	15
	لاحظ هنا تنفيذ عملية الجمع (+) على العاملين الموجودين في المكدس وهما (7) , (8) وخزن (push) نتيجة الجمع (15) بدلهما في المكدس	
4	6	15 6
5	3	15 6 3
6	*	15 18
	لاحظ هنا تنفيذ عملية الضرب (*) على العاملين (6) , (3) وخزن النتيجة (18) بدلهما في المكدس	
7	2	15 18 2
8	/	15 9
	لاحظ هنا تنفيذ عملية القسمة (/) على العاملين (18) , (2) وخزن النتيجة (9) بدلهما في المكدس	
9	-	6
	لاحظ هنا تنفيذ عملية الطرح (-) على العاملين (15) , (9) وخزن النتيجة (6) بدلهما في المكدس	

أن القيمة المتبقية في المكدس وهي (6) تمثل النتيجة النهائية لعملية احتساب قيمة العبارة الحسابية

خوارزمية احتساب قيمة (تنفيذ) العبارة الحسابية Infix

من التطبيقات الأخرى للمكدس استخدامه في المفسرات (Interpreters) لاحتساب قيمة العبارة الحسابية المكتوبة بصيغة (Infix) بدون تحويلها إلى صيغة (posfix).

خطوات الخوارزمية

1- يستخدم مكدسان هما (ST1) لخزن العوامل الحسابية (operands) و (ST2) لخزن إشارات العمليات الحسابية (operaters) .

2- تؤخذ رموز العبارة الحسابية بالتتابع واحداً بعد الآخر من اليسار إلى اليمين.

3- حسب نوع الرمز نقوم بما يلي :

إذا كان الرمز :	ينفذ ما يأتي :
+ أحد العوامل (operand)	+ يخزن (push) في المكدس (ST1) .
+ عملية حسابية (operator)	+ إخراج (pop) بالتتابع جميع العمليات الحسابية (أن وجدت) في المكدس (ST2) التي أسبقيتها >= أسبقية العملية الحسابية الجديدة وتنفيذ كل منها على العاملين في قمة المكدس (ST1) وخزن (push) النتيجة بدلهما في (ST1) . بعد ذلك تخزن (push) إشارة العملية الحسابية الجديدة في المكدس (ST2) .

4- بعد انتهاء جميع رموز العبارة الحسابية نبدأ بتنفيذ جميع العمليات الحسابية المتبقية في المكدس (ST2) بالتتابع على كل عاملين في قمة المكدس (ST1) واحلال نتيجة تلك العملية محلهما في نفس المكدس (ST1) ونستمر بتكرار هذه الخطوة لحين خلو المكدس (ST2) وتكون آخر قيمة موجودة في المكدس (ST1) هي النتيجة النهائية.

مثال : أوجد قيمة العبارة الحسابية الآتية المكتوبة بصيغة (Infix) باستخدام المكدس : 3 + 7 * 2 – 6

الحل :-

الخطوة	الرمز	ST1	ST2
1	3	3	.....
2	+	3	+
3	7	3 7	+
4	*	3 7	+ *
5	2	3 7 2	+ *
6	-	17	-

لاحظ هنا تنفيذ عملية الضرب (*) على العاملين (7) , (2) والنتيجة هي (14)لأن أسبقيتها >= من العملية الجديدة الطرح (-) ثم الاستمرار في تنفيذ عملية الجمع (+) على النتيجة المتحققة (14) والقيمة (3) لنحصل على (17) ولانتهاء العمليات الحسابية التي أسبقيتها >= أسبقية العملية الجديدة نخزن إشارة هذه العملية (-) في المكدس (ST2)

7	6	17 6	-

عند انتهاء جميع رموز العملية الحسابية المدخلة نبدأ بتنفيذ العمليات الحسابية المتبقية في المكدس (ST2) بالتتابع على محتويات المكدس (ST1) وتصبح الخطوة الأخيرة

8	.....	11	....

فالقيمة (11) المتبقية في المكدس (ST1) هي نتيجة الاحتساب

مثال :-حول التعبير الحسابي التالي من صيغة Infix إلى صيغة Postfix باستخدام مكدسين :2 ^ (a – 2 * (b / 3) ^ 5 + f) + 6 / X = : M

الحل :-

Step No.	Input char	ST1 For operands	ST2 For operators
1	M	M	.....
2	:=	M	:=
3	X	M X	:=
4	/	M X	:= /
5	6	M X 6	:= /
6	+	M X 6 /	:= +
7	(	M X 6 /	:= + (
8	A	M X 6 /a	:= + (
9	-	M X 6 /a	:= + (-
10	2	M X 6 /a2	:= + (-
11	*	M X 6 /a2	:= + (- *
12	(	M X 6 /a2	:= + (- * (
13	B	M X 6 /a2 b	:= + (- * (
14	/	M X 6 /a2 b	:= + (- * (/
15	3	M X 6 /a2 b3	:= + (- * (/
16	)	M X 6 /a2 b3/	:= + (- *
17	^	M X 6 /a2 b3/	:= + (- * ^
18	5	M X 6 /a2 b3/5	:= + (- * ^
19	+	M X 6 /a2 b3/5^ * -	:= + (+
20	F	M X 6 /a2 b3/5 ^ * - f	:= + (+
21	)	M X 6 /a2 b3/5 ^ * - f +	:= +
22	^	M X 6 /a2 b3/5 ^ * - f +	:= + ^
23	2	M X 6 /a2 b3/5 ^ * - f +2	:= + ^
24	.....	M X 6 /a2 b3/5 ^ * - f +2 ^ + : =	.....

مثال :

حول التعبير الحسابي التالي من صيغة (Infix) إلى صيغة (Postfix) باستخدام مكدسين :((A > B) AND ((E – C > A) OR (G < F))

الحل :

Step No.	Input char	ST1 For operands	ST2 For operators
1	(	.....	(
2	A	A	(
3	>	A	(>
4	B	A B	(>
5	)	A B >	.....
6	AND	A B >	AND
7	(	A B >	AND (
8	(	A B >	AND ((
9	E	A B > E	AND ((
10	-	A B > E	AND ((-
11	C	A B > E C	AND ((-
12	>	A B > E C -	AND ((>
13	A	A B > E C – A	AND ((>
14	)	A B > E C – A >	AND (
15	OR	A B > E C – A >	AND (OR
16	(	A B > E C – A >	AND (OR (
17	G	A B > E C – A > G	AND (OR (
18	<	A B > E C – A > G	AND (OR (<
19	F	A B > E C – A > G F	AND (OR (<
20	)	A B > E C – A > G F <	AND (OR
21	)	A B > E C – A > G F < OR	AND
22	.....	A B > E C – A > G F < OR AND	.....

مثال :

حول التعبير الحسابي التالي من صيغة (Infix) الى صيغة (Postfix) باستخدام مكدسين :

A Not (B OR Z OR Not(G < E))

الحل :

Step No.	Input char	ST1 For operands	ST2 For operators
1	A	A	.....
2	Not	A	Not
3	(	A	Not (
4	B	A B	Not (
5	OR	A B	Not (OR
6	Z	A B Z	Not (OR
7	OR	A B Z OR	Not (OR
8	Not	A B Z OR	Not (OR Not
9	(	A B Z OR	Not (OR Not (
10	G	A B Z OR G	Not (OR Not (
11	<	A B Z OR G	Not (OR Not (<
12	E	A B Z OR G E	Not (OR Not (<
13	)	A B Z OR G E <	Not (OR Not
14	)	A B Z OR G E < Not OR	Not
15	.....	A B Z OR G E < Not OR Not	.....

3 - تطبيقـــات أخرى

يستخدم المكدس كهيكل لخزن المعلومات التي نحتاج استرجاعها بصورة معكوسة (بترتيب معكوس) والحالات التي تتطلب العودة إلى موقع الخطوة السابقة (back tracking) وكمثال على ذلك مسائل المتاهة (A mazing problems) فعند المرور بموقع معين وتكون هنالك عدة مسارات يفترض اختيار أحدها للوصول إلى الهدف فأن الأمر يتطلب خزن هذا الموقع قبل تركه وتجربة مسار آخر إذ نحتاج إلى العودة لهذا الموقع في حالة خطأ ذلك المسار .

أن استخدام المكدس في مثل هذه الحالات يسمح بخزن سلسلة المواقع السابقة بحيث يمكن العودة إليها بعكس ترتيب المرور فيها .

تمرين محلول :

اكتب خوارزمية لقراءة جملة (string) تنتهي بالرمز (.) ثم طبعها بترتيب معكوس باستخدام المكدس .

الحل :

```
Algorithm
Begin
clear the stack
Repeat
    Read a character
   If      character <> ' . '
   Then  push the character onto stack
Until   character = ' . '
While  stack is not empty  Do
    Begin
       Pop the stack
       Print the character
    End
End
```

تمرين : اعتمد خوارزمية التمرين السابق واكتب البرنامج الفرعي (procedure) لقراءة الجملة (string) وطبعها بترتيب معكوس باستخدام المكدس .

```
Procedure   PrintReverse ;
Const    Dot = '.';
Var       character : char    ; stack  : st ;
          top : integer ;
Begin
        Clearstack ( top ) ;
        Repeat
            Read ( character ) ;
            If       character < > Dot
            Then    push ( stack , top , character )
        Until        character = Dot
        While        Not Emptystack ( top )   Do
              Begin
                    Pop ( stack , top , character ) ;
                    Write ( character )
              End
 End ;
```

برنامج - 1 : تمثيل المكدس (Stack) وعملياته

```
PROGRAM STACKS;
USES CRT;
CONST SIZE=10;
TYPE ELEMENT=INTEGER;  { OR ANY OTHER TYPE }
   ST=ARRAY [1..SIZE] OF ELEMENT;
VAR  STACK1:ST;
   ITEM1:ELEMENT;
   TOP1,CHOICE,I,L,M:INTEGER;
FUNCTION  FULLSTACK(TOP:INTEGER):BOOLEAN;
BEGIN
   IF  TOP=SIZE
   THEN FULLSTACK:=TRUE
   ELSE FULLSTACK:=FALSE
END;
FUNCTION  EMPTYSTACK(TOP:INTEGER):BOOLEAN;
BEGIN
   IF  TOP=0
   THEN EMPTYSTACK:=TRUE
   ELSE EMPTYSTACK:=FALSE
END;
PROCEDURE  PUSH(Var  STACK:St; Var TOP:INTEGER; ITEM:ELEMENT);
BEGIN
   IF  FULLSTACK(TOP)
   THEN WRITELN('ERROR...THE STACK IS FULL')
   ELSE BEGIN
      TOP:=TOP+1;
      STACK[TOP]:=ITEM
      END
END;
```

```
PROCEDURE POP(STACK:St; Var TOP:INTEGER; Var ITEM:ELEMENT);
BEGIN
   IF  EMPTYSTACK(TOP)
   THEN WRITELN('ERROR...THE STACK IS EMPTY')
   ELSE BEGIN
      ITEM:=STACK[TOP];
      TOP:=TOP-1
      END
END;
BEGIN  { MAIN PROGRAM }
CLRSCR;
TOP1:=0;
REPEAT
   WRITELN('REPRESENTATION OF STACK OPERATIONS ');
   WRITELN('_____');
   WRITELN('1- INSERTION OPERATION (PUSH)    ');
   WRITELN('2- DELETION OPERATION (POP)      ');
   WRITELN('3- DISPLAY THE CONTENT OF THE STACK');
   WRITELN('4- EXIT                          ');
   WRITELN;WRITELN;
   WRITE('SELECT YOUR CHOICE :  ');
   READLN(CHOICE);
   CASE  CHOICE  OF
   1: BEGIN
    WRITE('HOWMANY ELEMENTS YOU LIKE TO ENTER ? ');
    READLN(M);
    FOR  I:=1  TO  M  DO
       BEGIN
       WRITE('ENTER THE NEW ELEMENT : ');
       READLN(ITEM1);
```

```
        PUSH(STACK1,TOP1,ITEM1)
        END
      END;
    2: BEGIN
      WRITE('HOWMANY ELEMENTS YOU LIKE TO DELETE ? ');
      READLN(L);
      FOR I:=1 TO L DO POP(STACK1,TOP1,ITEM1)
      END;
    3: BEGIN
      WRITELN('THE CONTENT OF THE STACK IS : ');
      WRITELN('TOP = ',TOP1,'--->');
      FOR I:=TOP1 DOWNTO 1 DO  WRITELN(STACK1[I]:15);
      WRITELN;
      END;
    4: END;
UNTIL CHOICE=4
END.
```

برنامج - 2 : لقراءة جملة (string) وطبعها بصورة معكوسة باستخدام المكدس (Stack).

```
PROGRAM  Reversst;
USES CRT;
CONST SIZE=10; { OR ANY OTHER VALUE }
TYPE ELEMENT=CHAR;
       ST=ARRAY [1..SIZE] OF ELEMENT;
VAR  STACK1:ST;
   ITEM1:ELEMENT;
   TOP1,I:INTEGER;
FUNCTION   FULLSTACK(TOP:INTEGER):BOOLEAN;
BEGIN
   IF  TOP=SIZE
   THEN FULLSTACK:=TRUE
   ELSE FULLSTACK:=FALSE
END;
FUNCTION  EMPTYSTACK(TOP:INTEGER):BOOLEAN;
BEGIN
   IF  TOP=0
   THEN EMPTYSTACK:=TRUE
   ELSE EMPTYSTACK:=FALSE
END;
PROCEDURE  PUSH(VAR STACK:ST;VAR TOP:INTEGER;ITEM:ELEMENT);
BEGIN
   IF  FULLSTACK(TOP)
   THEN WRITELN('ERROR...THE STACK IS FULL')
   ELSE BEGIN
          TOP:=TOP+1;            STACK[TOP]:=ITEM
          END
END;
```

```
PROCEDURE POP(STACK:ST;VAR TOP:INTEGER;VAR ITEM:ELEMENT);
BEGIN
   IF  EMPTYSTACK(TOP)
   THEN WRITELN('ERROR...THE STACK IS EMPTY')
   ELSE BEGIN
          ITEM:=STACK[TOP];
          TOP:=TOP-1
          END
END;
BEGIN  { MAIN PROGRAM }
CLRSCR;
TOP1:=0;
      WRITELN('THIS PROGRAM READS IN ANY STRING AND PRINTED ');
      WRITELN('IN REVERSE ORDER USING STACK          ');
      WRITELN;
      WRITELN('INPUT YOUR STRING TERMINATED BY (.) ');
      ITEM1:='A';
      WHILE  ITEM1<>'.' DO
         BEGIN
         READ(ITEM1);
         PUSH(STACK1,TOP1,ITEM1)
         END;
      TOP1:=TOP1-1;
      WRITELN('YOUR STRING IN REVERSE ORDER IS ');
      FOR  I:=TOP1  DOWNTO  1  DO
           BEGIN
           POP(STACK1,TOP1,ITEM1);
           WRITE(ITEM1)
           END
 END.
```

برنامج - 3 : لقراءة جملة (string) وفحصها هل يمكن قراءتها من الاتجاهين أي أن نوعها (PALINDROME) وذلك باستخدام المكدس .

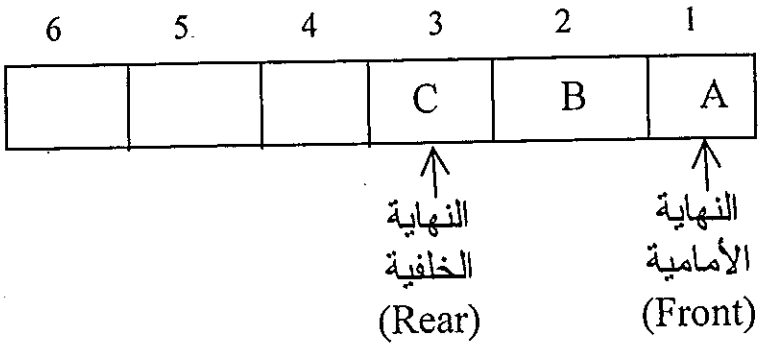
```
PROGRAM PALINDST;
USES CRT;
CONST SIZE=30; { OR ANY OTHER VALUE }
TYPE ELEMENT=CHAR;
   ST=ARRAY [1..SIZE] OF ELEMENT;
VAR  STACK1,STACK2:ST;
   ITEM1,CH1,CH2:ELEMENT;
   TOP1,TOP2,COUNT,I:INTEGER;
   PALINDROME:BOOLEAN;
FUNCTION  FULLSTACK(TOP:INTEGER):BOOLEAN;
BEGIN
   IF  TOP=SIZE
   THEN FULLSTACK:=TRUE
   ELSE FULLSTACK:=FALSE
END;
FUNCTION  EMPTYSTACK(TOP:INTEGER):BOOLEAN;
BEGIN
   IF  TOP=0
   THEN EMPTYSTACK:=TRUE
   ELSE EMPTYSTACK:=FALSE
END;
PROCEDURE  PUSH(VAR STACK:ST;VAR TOP:INTEGER;ITEM:ELEMENT);
BEGIN
   IF  FULLSTACK(TOP)
   THEN WRITELN('ERROR...THE STACK IS FULL')
   ELSE BEGIN
      TOP:=TOP+1;
      STACK[TOP]:=ITEM
```

```
END;
PROCEDURE POP(STACK:ST;VAR TOP:INTEGER;VAR ITEM:ELEMENT);
BEGIN
   IF  EMPTYSTACK(TOP)
   THEN WRITELN('ERROR...THE STACK IS EMPTY')
   ELSE BEGIN
      ITEM:=STACK[TOP];
      TOP:=TOP-1
      END
END;
BEGIN  { MAIN PROGRAM }
CLRSCR;
TOP1:=0;  TOP2:=0;  COUNT:=0  ; PALINDROME:=TRUE ;
    WRITELN('THIS PROGRAM CAN READS IN ANY STRING AND TESTED
IF ITS');
    WRITELN('PALINDROME OR NOT (i.e. IT CAN BE READ FROM BOTH
SIDES)');
    WRITELN;
    WRITELN('INPUT YOUR STRING TERMINATED BY THE CHAR (.) ');
     CH1:='A';
     WHILE  CH1<>'.' DO
        BEGIN
        READ(CH1);
        PUSH(STACK1,TOP1,CH1);
        COUNT:=COUNT+1
        END;
     COUNT:=COUNT-1;
     POP(STACK1,TOP1,CH1);   { GETRID OF THE '.' }
     FOR  I:=1  TO  (COUNT DIV 2)  DO
```

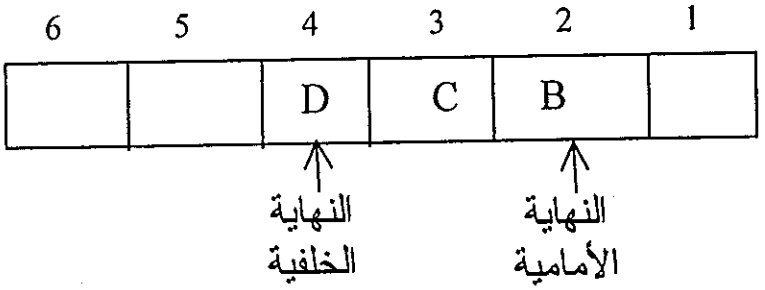
```
BEGIN
   POP(STACK1,TOP1,CH1);
   PUSH(STACK2,TOP2,CH1)
END;
IF  (COUNT MOD 2)=1
THEN POP(STACK1,TOP1,CH1);
WHILE  NOT EMPTYSTACK(TOP1) AND  PALINDROME  DO
BEGIN
   POP(STACK1,TOP1,CH1);
   POP(STACK2,TOP2,CH2);
   IF  CH1<>CH2
   THEN PALINDROME:=FALSE
END;
IF   PALINDROME
THEN  WRITELN('THE STRING IS PALINDROME ')
ELSE  WRITELN('THE STRING IS NOT PALINDROME');
END.
```

3_3 الطابـــور Queue

هو هيكل تسلسلي (sequential) تكون فيه عمليات الإضافـة فـي النهايـة الخلفيـة (Rear) وعمليات الحذف في النهاية الأخرى (الأمامية Front)
كما في الشكل التالي :



حيث نلاحظ أن العنصر (A) في مقدمة الطابور يليه العنصر (B) ثم (C) وعنـد إضافة عنصر جديد يكـون موقعـه بعد (C) ، أمـا عنـد حـذف عنصـر مـن الطـابور تكون عملية الحذف من النهاية الأمامية أي حذف العنصر (A) ويصبح الشكل أعلاه بعد إضافة العنصر (D) وحذف العنصر (A) كالآتي :-



نجد أن الطابور يفيد في الفعاليـات التـي تتضمـن جدولـة الأعمـال حسـب ترتيـب وصولها أو طلبها ويمكن تلخيص هذا بالعبارة الآتية :

أول من يدخل أول من يخرج First IN First Out [FI FO]

أي أن مـن يصـل أولا يحصـل علـى الخدمـة أولا وتسـمى عمليـة الإضافـة إلـــى الطابور (ENQueue) أو (Insertion) أما عملية الحذف فتسمى (DEQueue) أو (Deletion) .

3 – 3 – 1 تمثيل الطابور باستخدام المصفوفة

Array Representation of Queue

يطبق الطابور باستخدام مصفوفة أحادية بالسعة المطلوبة (SIZE) وبالنوع المناسب لنوع البيانات (Data Type) التي ستخزن فيه (Real , Int. ... الخ) مع استخدام :

المتغير (Rear) كمؤشر يشير إلى موقع العنصر الأخير في الطابور

المتغير (Front) كمؤشر يشير إلى موقع العنصر الأول في الطابور

أن قيمة المؤشرين في الحالة الابتدائية (Front = 0 , Rear = 0) عندما يكون الطابور خالياً (Empty) من العناصر

تنفذ عملية إضافة عنصر ألي الطابور بعد تحديث قيمة المؤشر (Rear) ليشير إلى الموقع الجديد بعد موقع أخر عنصر أما تنفيذ عملية حذف عنصر من الطابور فيحدث المؤشر (Front) ليشير إلى موقع العنصر التالي بعد حذف العنصر في المقدمة ولنفترض لدينا الطابور (Q) سعته (6) عناصر وننفذ عليه سلسلة العمليات الآتية :-

الحالة	المؤشر F	المؤشر R	حالة الطابور Q [1]	Q [2]	[3]	[4]	[5]	[6]
- الطابور خالي	0	0	-	-	-	-	-	-
- إضافة العنصر A	1	1	A	-	-	-	-	-
- إضافة العنصر B	1	2	A	B	-	-	-	-
- إضافة العنصر C	1	3	A	B	C	-	-	-
- حذف عنصر	2	3	-	B	C	-	-	-
- إضافة العنصر D	2	4	-	B	C	D	-	-
- إضافة العنصر E	2	5	-	B	C	D	E	-
- حذف عنصر	3	5	-	-	C	D	E	-
- حذف عنصر	4	5	-	-	-	D	E	-

ويعرف الطابور برمجياً باستخدام العبارات البرمجية التالية :-

```
Const    Size = 10 ;  { or any other value }
Type     Queueelement = char ; { or ant other type }
         Q = array [1 .. Size ] of Queueelement ;
Var      Queue : Q ;  Rear , Front : integer ;
```

عملية الإضافة للطابور **Add to Queue**

تعتمد الخطوات الآتية لإضافة عنصر واحد إلى الطابور :

1- التحقق بأن الطابور غير مملوء (Not Full) أي أن المؤشر (Rear <> Size) لتجنب حالة الفيض (Overflow) ولتعذر تنفيذ عملية الإضافة عند ذلك .

2- تحديث قيمة المؤشر (Rear := Rear + 1) ليشير إلى الموقع التالي

3- إضافة العنصر الجديد في الموقع الجديد Queue [Rear]

عملية الحذف من الطابور **Delete from Queue**

نعتمد الخطوات الآتية لحذف عنصر واحد من الطابور :

1- التحقق بأن الطابور غير خالٍ (Not Empty) أي أن المؤشر (Front <> 0) لتجنب حالة الغيض (Underflow) وتعذر تنفيذ عملية الحذف .

2- اخذ العنصر من الموقع الذي يشير إليه المؤشر (Front) وخزنه وقتياً في متغير مستقل وليكن Item := Queue [Front]

3- تحديث قيمة المؤشر (Front := Front + 1) ليشير إلى موقع العنصر الآتي للعنصر الذي حذف .

ملاحظة :

هنا أيضا يتضح أن الخطوتين (2 ، 3) في عملية الحذف معكوسة الترتيب عنها في عملية الإضافة .

3-3-2 خوارزميات الطابور Queue's Algorithms

أدناه مجموعة من الخوارزميات لتغطية العمليات التي تنفذ على الطابور .

1- خوارزمية الإضافة Add Queue

```
If      Queue is Full
Then Overflow ← True
Else
    ⌈ Overflow ← False
    | Rear ← Rear + 1
    ⌊ Queue [ Rear ] ← New element
```

2- خوارزمية الحذف Delete Queue

```
If      Queue is  Empty
Then Underflow ← True
Else
    ⌈ Underflow ← False
    | Element ← Queue [ Front ]
    ⌊ Front ← Front + 1
```

3- خوارزمية ملء الطابور Full Queue

هذه الخوارزمية للتحقق من الطابور أن كان مملوء أم لا اعتماداً على قيمة المؤشر (Rear) قبل عمليات الإضافة .

```
If      Rear = size
Then    FullQueue ← True
Else    FullQueue ← False
```

4- خوارزمية خلو الطابور Empty Queue

هذه الخوارزمية للتحقق من الطابور أن كان خالياً أم لا اعتماداً على قيمة المؤشر (Front) قبل عمليات الحذف .

```
If      Front = 0
Then    EmptyQueue ← True
Else    EmptyQueue ← False
```

5- خوارزمية إخلاء (تفريغ) الطابور Clear Queue

هذه الخوارزمية تستخدم لغرض تهيئة الطابور وإخلائه من العناصر بجعل قيمة كل من المؤشرين (Front = 0 , Rear = 0)

Front ⟵ 0

Rear ⟵ 0

3-3-3 البرامج الفرعية لتنفيذ عمليات الطابور

Queue Procedures and Functions

فيما يأتي مجموعة من البرامج الفرعية (Functions , Procedures) أعدت بنفس أسلوب البرامج الفرعية للمكدس مع افتراض وجود التعريف التالي للمتغيرات في مقدمة البرنامج

```
Const    Size  = 10 ; { or any other value }
Type       Queueelement  = char ;    { or any other type }
           Q  = array [ 1 .. Size ] of  Queueelement ;
Var    Queue : Q ;
       Rear , Front : integer ;
       Item : Queueelement ;
```

1- برنامج فرعي لاخلاء الطابور

```
Procedure  ClearQueue( Var   Front , Rear : integer ) ;
Begin
      Rear  : = 0 ;
      Front : = 0
End ;
```

نلاحظ عدم الحاجة للمرور على جميع مواقع المصفوفة والاكتفاء فقط بجعل قيمة كل من المؤشرين مساوياً للصفر .

يستدعى هذا البرنامج الفرعي في البداية لجعل الطابور خالياً

2- برنامج فرعي للتحقق من امتلاء الطابور

```
Function    FullQueue ( Rear : integer ) : Boolean ;
Begin
      If      Rear = size
      Then  Fullqueue : = True
      Else   Fullqueue : = False
End ;
```

هذه الدالة (function) مدخلها المؤشر (Rear) وبموجب قيمته تكون قيمة المخرج المنطقي (FullQueue) عندما يكون الطابور مملوءاً (True) وتكون قيمته (False) عندما يكون الطابور غير مملوء .

يستدعى هذا البرنامج الفرعي داخل البرنامج الفرعي (Procedure InsertQueue) الذي ينفذ عملية الإضافة .

3- برنامج فرعي للتحقق من خلو الطابور

```
Function    EmptyQueue ( Front : integer ) : Boolean ;
Begin
      If      Front = 0
      Then Emptyqueue : = True
      Else   Emptyqueue : = False
End ;
```

هذه الدالة (function) مدخلها المؤشر (Front) وبموجب قيمته تكون قيمة المخرج المنطقي (EmptyQueue) عندما يكون الطابور خالياً (True) وتكون قيمته (False) عندما يكون الطابور غير خال .

يستدعى هذا البرنامج الفرعي داخل البرنامج الفرعي (Procedure DeleteQueue) الذي ينفذ عملية الحذف .

4- برنامج فرعي لإضافة عنصر واحد إلى الطابور

```
Procedure  AddQueue( Var Queue : Q ; Var  Front , Rear : integer ;
                                          Item : Queueelement ) ;
Begin
        If      Fullqueue ( Rear)
        Then  writeln ( ' Error … the queue is full ' )
        Else   Begin
                    Rear : = Rear + 1 ;
                    Queue [ Rear ] : = Item
               End ;
        If      Front  = 0
        Then  Front : = 1
   End ;
```

هذه الخطوة لمعالجة أول إضافة إلى الطابور ←

ملاحظة : يمكن استدعاء هذا البرنامج الفرعي داخل البرنامج الرئيسي (main Program) بأي عدد من المرات باستخدام أحد ايعازات التكرار مثل (For – Do) بقدر عدد العناصر المطلوب إضافتها .

5- برنامج فرعي لحذف عنصر واحد من الطابور

```
Procedure  DeleteQueue(Queue : Q ;  Var  Front , Rear : integer ;
                                          Var  Item : Queueelement ) ;
Begin
        If      Emptyqueue ( Front)
        Then  writeln ( ' Error … the queue is Empty ' )
        Else   Item : = Queue [ Front ] ;
        If      Front = Rear
   Then       Begin
                    Front  : = 0 ;
                    Rear   : = 0
              End
   Else Front  : = Front  + 1
End ;
```

هذه الخطوة لمعالجة حذف آخر عنصر في الطابور ويشيران إليه كل من Front , Rear

ملاحظة : يمكن استدعاء هذا البرنامج الفرعي من قبل البرنامج الرئيسي بأي عدد من المرات بقدر عدد العناصر المطلوب حذفها باستخدام أحد ايعازات التكرار .

3-3 – 4 تمثيل الطابور باستخدام القيد

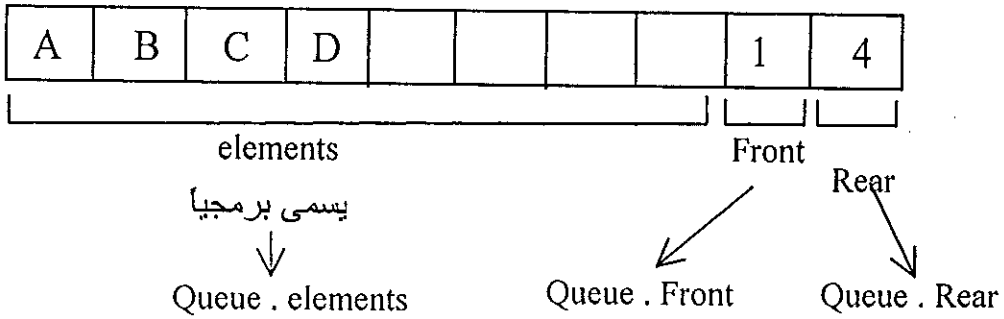
Record Representation of Queue

يستخدم القيد في تمثيل الطابور والمؤشرين (Front , Rear) في هيكل بياني واحد أي أن القيد يتكون من ثلاثة أجزاء ، الجزء الأول يمثل مصفوفة الطابور والجزء الثاني وهو حقل يمثل المؤشر (Front) والجزء الثالث هو حقل أخر يمثل المؤشر (Rear).

ويكون التعريف حسب لغة باسكال كما يأتي :-

```
Const     Size  = 8  ; { or any other value }
Type      Qelement = char ;  { or any other type }
          Q = Record
           elements : = array [ 1 .. size ] of  Qelement  ;
           Front : integer  ;
           Rear : integer
           End  ;
Var       Queue : Q  ;
          Item : Qelement  ;
```

Queue :



لإضافة عنصر جديد لهذا الطابور نتبع الخطوات التالية :-

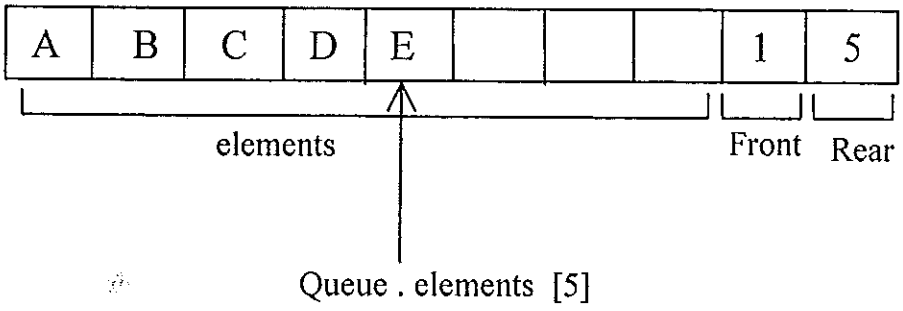
1- نحدث قيمة المؤشر (Rear) الذي هو حقل في القيد Queue ليصبح (5)

```
Queue . Rear : = Queue . Rear + 1
```

2- نضيف العنصر الجديد (E) في الموقع الجديد (5)

```
Queue . elements [ Queue . Rear ] : = E
```

وبهذا يصبح الطابور بالصورة التالية :



أما عند حذف عنصر من الطابور فنتبع الخطوات الآتية :

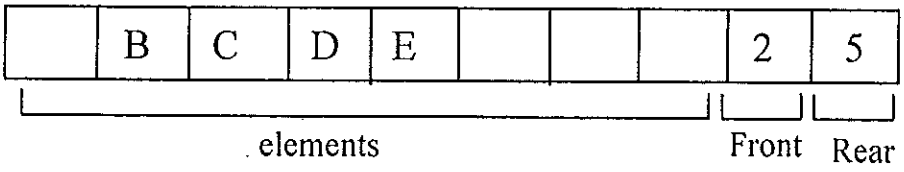
1- نأخذ العنصر من مقدمة الطابور كما يشير إليه المؤشر (Front) بالإيعاز :

Item : = Queue . elements[Queue.Front]

2- تحديث قيمة المؤشر (Front) من (1) ليصبح (2) بالإيعاز:

Queue.front := Queue.front+1

ويصبح الطابور بالصورة الآتية :



تمرين: اعد كتابة البرامج الفرعية لعمليات الطابور باستخدام القيد.

3 - 3 – 5 تطبيقات الطابور

من التطبيقات الشائعة للطابور في مجال الحاسوب :

* جدولة الأعمال (job scheduling) ، ففي نظام معالجة الدفعة (batch processing) تنظم الأعمال المطلوب تنفيذها في طابور حسب وقت وصولها ومن ثم تنفذ بالتتابع واحداً بعد الأخر .

* كما تستخدم أنظمة التشغيل الطابور في جدولة استخدام المصادر المختلفة للحاسوب (Computer Resources) ، فطابور للأعمال التي تحتاج وقت للاخراج على الطابعة أخر للادخال واستخدام القرص، وهكذا باقي الأجهزة.

تمرين : أعـد كتابـة البرنـامج الفرعـي (AddQueue) لاظافـة عنصـر إلـى الطابور باستخدام القيد (Record Implementation) .

الحل :

سنعتمد التعريفات الموجودة في الصفحة رقم (82) .

```
Procedure  AddQueue( Var Queue : Q ; Item : Qelement);
Begin
      IF       Queue . Rear = size
      THEN   Writeln( ' Error .. the queue is full ')
      ELSE
            Begin
            Queue . Rear := Queue.Rear + 1 ;
            Queue . elements[Queue . Rear] := Item
            End ;
      IF       Queue . Front = 0
      THEN   Queue . Rear := 0
End ;
```

إلى

تمرين :

اكتب برنامجا فرعيا لإضافة (5) عناصر إلى الطابور (LINE) الذي سعته (25) عدد صحيح.

الحل:

```
Const  Size  = 25 ;
Type element = integer ;{ or any other type }
Qt = array [1..size] of element;
Procedure   AddQueue( Var line: Qt ;Var Front,Rear : integer);
Var   I: integer ;    Item : element ;
Begin
        For   I:= 1   to   5   Do
        Begin
            If      Rear = size
            Then   writeln (' Error.. the queue is full');
            Else
                Begin
                    Rear := Rear +1  ;
                    Read (Item)  ;
                    Line (Rear):= Item
                End;
            If      Front = 0
            Then  Front : = 1 ;
        End
End;
```

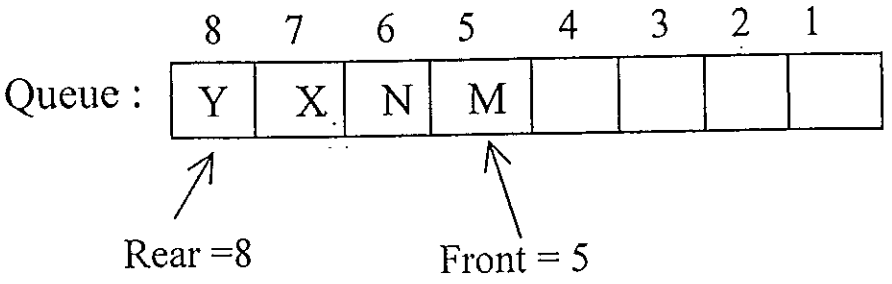
تمرين :

اكتب خوارزمية (Algorithm) لقراءة جملة (string) تتكون من جزأين (two substrings) يفصلهما الرمز '.' ثم التحقق بكونهما متطابقين أم لا باستخدام الطابور .

```
Algorithm
Begin
    Clear the Queue
    Stringmatch ⟵ True
    Repeat
        Read a character and put it in the string
        If      character <> '.'
        Then    EnQ the character
        Until character = '.'
        While (more elements in the Queue) Do
        Dequeue an element
        Read the next character and place in string
        If      Dequeue character <> character
        Then    stringmatch ⟵ false
End .
```

4-3 الطابور الدائري Circular Queue:

لاحظنا عند إضافة عنصر إلى الطابور يتطلب تدقيق (فحص) قيمة المؤشر (Rear = size) إذ أن ذلك يعني أن الطابور مملوء حتى وان كانت هنالك مواقع خالية في مقدمة الطابور كما في الشكل الآتي :



أي أننا سنخسر مساحة خزنية دون استخدام ، ولتجنب هذه الحالة نستطيع استخدام الطابور استخداما دائريا وذلك بان نسمح للمؤشر (Rear) بالدوران إلى النهاية الأمامية للطابور (wrap – around) عندما يكون هنالك مواقع خالية فيها.

أن خصائص هذا الطابور هي كالآتي:

1- أن المؤشر (Front) يشير إلى الموقع الذي أمام أول عنصر في الطابور .

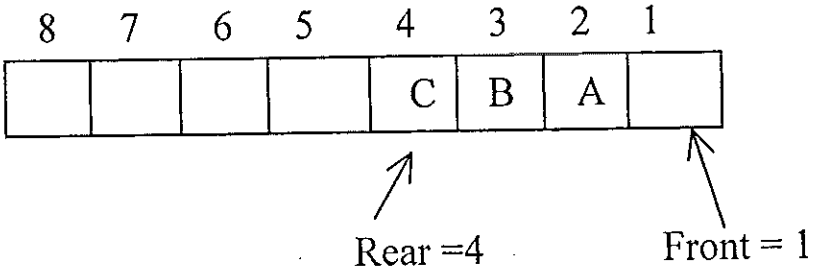
2- أن المؤشر (Rear) يشير إلى موقع العنصر الأخير في الطابور .

3- عندما يصل المؤشر (Rear) إلى الموقع الأخير في الطابور أي (Rear = size) نجعله يدور إلى البداية أي (Rear = 1) وكذلك الحال بالنسبة للمؤشر (Front) .

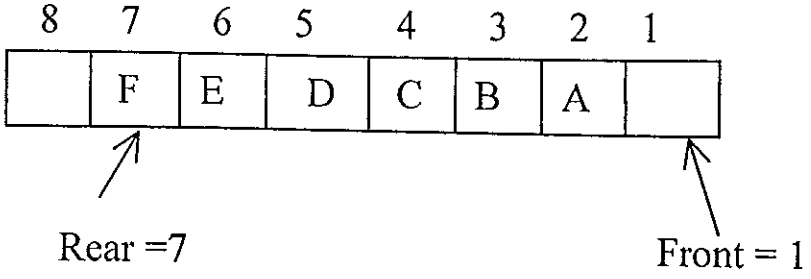
4- أن اكبر عدد من العناصر التي يستوعبها هذا الطابور هو (size-1) لان المؤشر (Front) يشير إلى الموقع الخالي أمام أول عنصر في الطابور.

ولنأخذ هذه الأمثلة التوضيحية الآتية لطابور دائري سعته (8):

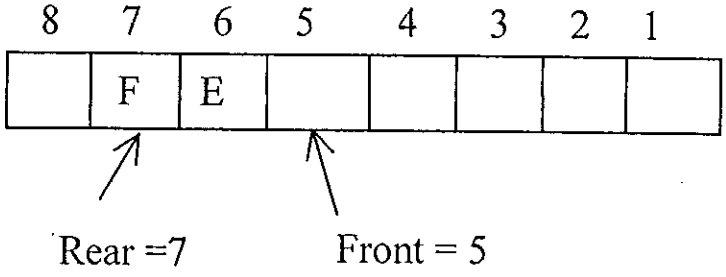
الحالة الأولى: يحتوي الطابور ثلاثة عناصر C,B,A



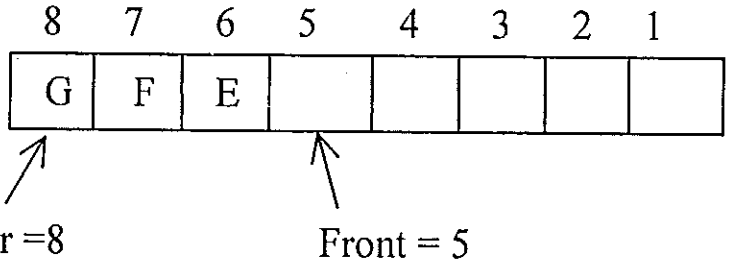
الحالة الثانية : بعد إضافة العناصر F,E,D



الحالة الثالثة : بعد حذف العناصر D,C,B,A

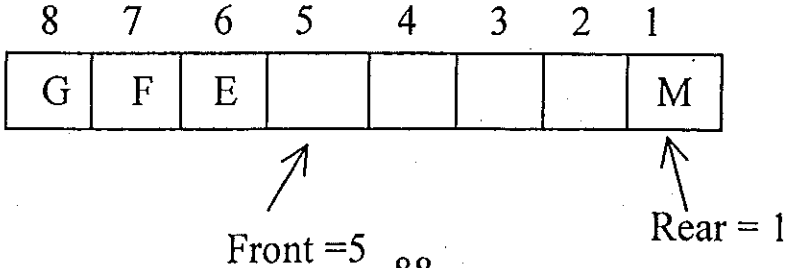


الحالة الرابعة : بعد إضافة العنصر (G)



لاحظ هنا في الطابور البسيط يعتبر الطابور مملوء لان (Rear = Size =8) مع انه يوجد مواقع خالية في المقدمة أما في الطابور الدائري فنستطيع إضافة عنصر كما في الحالة التالية .

الحالة الخامسة: لإضافة العنصر (M) لاحظ دوران المؤشر (Rear) إلى الجهة الأمامية بعد وصوله إلى الموقع الأخير في الطابور .



الحالة السادسة: بعد إضافة العناصر X,P,N لاحظ هنا أن الطابور اصبح مملوءاً لأننا إذا أردنا إضافة عنصر فيجب تحديث قيمة المؤشر (Rear) ليصبح مساويا (5) ويكون مساويا لقيمة المؤشر (Front = 5) الذي يجب أن يبقى موقعا خاليا كما أن سعة هذا الطابور هي (size-1) أي (7=1-8) وهذا ما يحتويه حاليا ، لذا فيتعذر الإضافة في هذه الحالة .