هياكل البيانات و الخوارزميات

المرحله الثانيه

علوم الحاسبات

م.د.فائزه عبد الجبار

م.م. هبه عادل

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 فقط بل يجب 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 الهيكل 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 A[4][5] فأنها تُمثل في ذاكرة الحاسوب في (20) موقع متعاقب ، والمبرمج عند استخدامه أو عند تعامله مع بيانات هذه المصفوفة باعتبارها مكونة من أربعة صفوف [0،1،2،3] وخمسة أعمدة [0،1،2،3،4] كما نتعامل معها رياضياً ، فالوصول إلى الموقع A[2][3] لا يعني البحث فيزياوياً في الصف الثالث والعمود الرابع لأن مثل هذه الصورة غير موجودة فيزياوياً بل يجب البحث عن الموقع الثاني عشر (بافتراض استخدام طريقة الصفوف لتمثيل المصفوفة في لغة C++) ابتداءاً من أول موقع حُدد 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 (وخصوصاً 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) فمثلاً في لغة C++ تستخدم التعريفات :

```
int    x
float  y
char   a
long   m
short  k
```

لتمثل في ذاكرة الحاسوب ويتمّ 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

Array	1- المصفوفة
Structure	2- الهيكل
File	3- الملف
Linear Structures	4- الهياكل الخطية
Non – linked structures	+ الهياكل غير الموصولة
Stack	+ المكدس
Queue	+ الطابور
Circular Queue	+ الطابور الدائري
Linked Structures	+ الهياكل الموصولة
Linked Stack	+ المكدس الموصول
Linked queue	+ الطابور الموصول
Non – Linear Structures	5- الهياكل غير الخطية
Graphs	+ المخططات
Directed graph	+ المخطط المتجه
Tree structure	+ هيكل الشجرة
Undirected graph	+ المخطط غير المتجه

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 :

Size of the data	1. حجم البيانات
Speed & Method of Using the data	2. سرعة وطريقة استخدام البيانات
Data dynamics	3. الطبيعة الديناميكية للبيانات كتغييرها وتعديلها دورياً
Storage Space Required	4. السعة الخزنية المطلوبة
Retrieval Time required to Access any data item	5. الزمن اللازم لاسترجاع أية معلومة من الهيكل البياني
Programming Techniques	6. اسلوب البرمجة

7-1 النوع البياني المجرد (Abstract Data Type – ADT)

عند تصميم نوع بياني معين (data type) فمن المهم التمييز بين مهمتين هما :

+ تعريف وظيفته أو عمله (أي غرض أستخدامه).
+ طريقة تمثيله حاسوبيا" (Implementation) .

أن النوع البياني المجرد هو المعني بالمهمة الأولى ويعتبر مجموعة من الأشياء (set of objects) مع مجموعة من العمليات (set of operations) التي سيقوم بها .

أي انه تصور أو تحقيق (أو بناء) النوع البياني (Data Type) كمكون برمجي (software component) يُعرف النوع البياني و العمليات التي يمكن تنفيذها عليه وفق نوع المدخلات (input) ونوع المخرجات (output)

فمثلا" النوع البياني المجرد للمكدس (stack ADT) هو هيكل بياني تنفذ فيه عمليتي الأضافة (push) و الحذف (pop) ، أما طريقة تمثيله حاسوبيا" فانه يمكن أن تتم بعدة طرق (كأستخدام المصفوفة أو أستخدام القائمة الموصولة مثلا") .

في لغة البرمجة الشيئيه (C++) يمكن تمثيل النوع البياني المجرد (ADT) بأستخدام (class) الذي سيتضمن توصيف النوع البياني (Data Specifications) والعمليات التي ستنفذ عليه (Functions) .

فلو أخذنا سيارة ما ، فأن أهم الفعاليات الأساسية فيها هي : القيادة ، التسارع ، التوقف . وتصميم هذه السيارة يمكن ان نصوره كنوع بياني مجرد (ADT) وفعالياته هي (القيادة ، التسارع ، التوقف) ، وهذا التصميم هو نفسه لجميع انواع السيارات حتى لو أختلفت الموديلات أو أنواع المحركات مثلا"، لذا فأن السائق يمكنه استخدام (سياقة) اية سيارة لأنه سيتعامل معها كنموذج لـ (ADT) تتشابه فيها الصفة الغالبة للسيارة كمركبة و فعالياتها الأساسية هي قيادة – تسارع – توقف.

الفصل الثالث
المكدس والطابور

Linear List	القائمة الخطية	1-3
Types of linear lists	أنواع القوائم الخطية	1-1-3
Operations on linear Lists	العمليات التي يمكن أجراؤها على القوائم الخطية	2-1-3
Stack	المكدس	2-3
Array Representation of Stack	تمثيل المكدس باستخدام المصفوفة	1-2-3
Stack's Algorithms	خوارزميات المكدس	2-2-3
Stack's subprograms	البرامج الفرعية لتنفيذ عمليات المكدس	3-2-3
Structure representation of Stack	تمثيل المكدس باستخدام القيد	4-2-3
Stack's Applications	أهم تطبيقات المكدس	5-2-3
Queue	الطابور	3-3
Array Representation of Queue	تمثيل الطابور باستخدام المصفوفة	1-3-3
Queue's Algorithms	خوارزميات الطابور	2-3-3
Queue's Subprograms	البرامج الفرعية لتنفيذ عمليات الطابور	3-3-3
Structure Representation of queue	تمثيل الطابور باستخدام القيد	4-3-3
Queue's Applications	تطبيقات الطابور	5-3-3
Circular Queue (CQ)	الطابور الدائري	4-3
Double Ended Queue	الطابور المزدوج	5-3

3-1 القائمة الخطية Linear List

هي مجموعة من العناصر البيانية (elements , nodes , items) المتسلسلة والمرتبة ، تربط عناصرها علاقة تجاور بحيث يسبق كل عنصر عنصرا عدا العنصر الأول الذي لا يسبقه عنصر أخر 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]

فالعقـدة الأولى هي x[1] والعقـدة 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) فمثلا :

+ 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

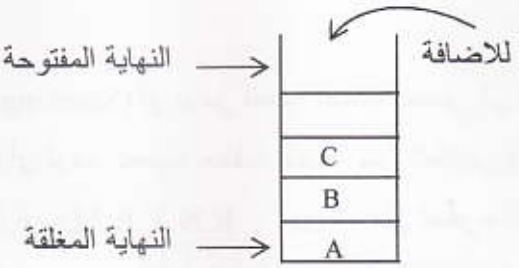
ترتيب عناصر (عقد) الهيكل البياني وفق قيمة حقل (field) أو مجموعة حقول.

9- الوصول Access

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

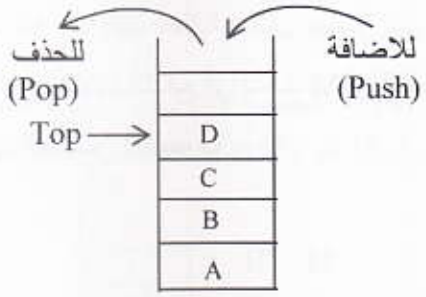
2-3 المكدس Stack

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 :



الشكل (3 – 2)

وعند حذف عنصر من المكدس يجب أن تُستخدم 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 أن 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 إلى المكدس (push) أو (Insertion) وعملية الحذف من المكدس (pop) أو (Deletion) .

مثال :

نفرض (S) تعني (Stacking) أي ترمز لعملية إضافة عنصر إلى المكدس و (U) تعني (Unstacking) أي ترمز لعملية حذف عنصر من المكدس وكانت 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 عند تنفيذ عملية إدخال عنصر إلى المكدس فان 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 التالية أثناء 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

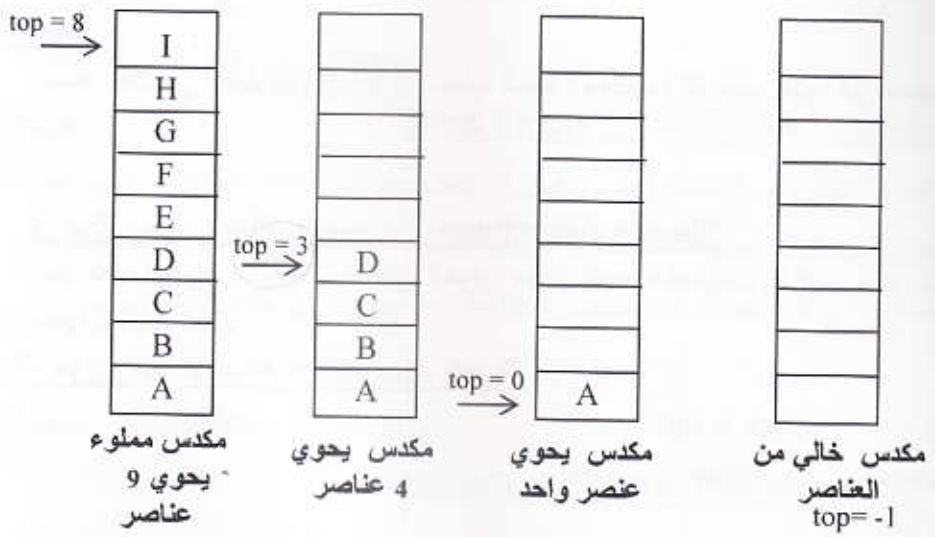
3-2-1 تمثيل المكدس باستخدام المصفوفة

Array Representation of Stack

يمكن تمثيل المكدس حاسوبيا باستخدام مصفوفة أحادية بالسعة المطلوبة (size) وبالنوع المناسب للبيانات (Data Type) التي ستخزن فيه (int , float ... الخ) مع استخدام متغير مستقل يدعى (top) يستعمل كمؤشر يشير إلى موقع أعلى عنصر في المكدس (موقع اقرب عنصر إلى النهاية المفتوحة) ، وابتداءً تكون قيمة المؤشر (top = -1) عندما يكون المكدس خالياً من العناصر ، ويُعرف المكدس برمجياً بلغة C++ كالآتي:-

```
const int size = 9 ;        { or any other value }
int stackelement ;          { or any other type }
stackelement stack[size] ;
int top ;
```

ملاحظة : مواقع المصفوفة في لغة (C++) تبدأ من (0) وليس من (1) ، وبالنسبة لهذا المثال ستكون المواقع التسعة من (0-8)



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<>-1) 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 تم حذفه .

ملاحظة :

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

2-2-3 خوارزميات المكدس Stack's Algorithms

يمكن تصميم (اعداد) 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 أن المكدس مملوء أم لا اعتماداً على قيمة المؤشر (top) قبل عمليات الإضافة، أذ يجب التحقق من هذه الحالة قبل عملية أضافة أي عنصر بياني لأنه سيكون من المتعذر تنفيذ أية عملية أضافة الى مكدس مملوء .

```
If      top = size - 1
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 ، أذ يجب التحقق من هذه الحالة قبل عملية حذف أي عنصر بياني لأنه سيكون من المتعذر تنفيذ أية عملية حذف الى مكدس خال .

```
If      top = - 1 .
Then    stackempty ⟵ true
```

5- خوارزمية إخلاء المكدس ClearStack

هذه الخوارزمية تستخدم لغرض تهيئة المكدس وإخلائه من العناصر بجعل قيمة المؤشر (top = - 1)

```
Top ⟵ ( - 1 )
```

3-2-3 البرامج الفرعية لتنفيذ عمليات المكدس

Stack Subprograms

بما أن المكدس هو هيكل بياني خصائصه (attributes) هي : حجمه size ، ونوع البيانات المخزونة فيه data type ، وأهم عملياته (actions) هي الأضافة (Push) و الحذف (Pop) ، لذا فأن تعريفه برمجيا بلغة C++ يكون بما يسمى (Class) ، وستكون مكوناتها كالأتي :

1 - نفترض وجود التعريفات البرمجية التالية :

```
const int  size  = 20 ;  { or any other integer value }
int  stackelement  ;   { or any other type }
stackelement  stack[size]  ;
int  top ;
```

2- برنامج فرعي (function) لاخلاء المكدس

```
void   initialize( );
{     top  = - 1 ; }
```

لاحظ عدم الحاجة للمرور على جميع مواقع المصفوفة وجعلها مساوية للصفر والاكتفاء فقط بجعل المؤشر (top = - 1) وهذا البرنامج الفرعي (function) يُستدعى في بداية العمل لجعل المكدس خاليا .

ويمكن أيضا استخدام (constructor) في لغة C++ لتحقيق نفس الهدف باستخدام اسم الـ class بالصيغة التالية :

```
Stack ( )
{     top= -1 ; }
```

3- اضافة عنصر بياني واحد الى المكدس (Push)

```
void push( int item )
{
    if ( top == size )
    cout<<"\n The stack is Full " ;
    else
       stack[ ++top ] = item ;
}
```

هذه الخطوات هي لأضافة عنصر واحد فقط ، ويمكن استخدامها داخل أحد ايعازات التكرار مثل (for ... do) او (while ... do) بما يسمح بقراءة (ادخال) قيمة العنصر الجديد (item) مع كل دورة تكرار ومن ثم ادخاله (push) الى المكدس

4 – حذف عنصر بياني واحد من المكدس (Pop)

```
int pop( )
{
    if ( top == -1 )
       cout<<"\n  The stack is Empty " ;
    else
       item = stack[top -- ] ;
    return  item ;
}
```

هنا أيضا هذه الخطوات هي لحذف عنصر واحد فقط ، ويمكن استخدامها داخل أحد ايعازات التكرار بما يسمح بأخذ العنصر الذي يشير اليه (top) من المكدس ونسخه (حفظه) في المتغير (item) لأستخدامه لاحقا و أجراء أية عملية مطلوبة عليه أو له، ومن ثم الأستمرار في ذلك مع كل دورة تكرار.

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

```
Main { 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 ——
}
```

2 - استخدام المكدس في معالجة التعابير الحسابية

Arithmetic expressions

من المعروف أن التعابير الحسابية تكتب بثلاث صيغ هي :

1- صيغة Infix notation

تكون إشارة العملية الحسابية تتوسط العوامل مثل :

3 + 4 , A – B , 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

في هذه الصيغة تكون إشارة العملية الحسابية تلحق العوامل مثل :

3 4 + , 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* -	.....

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

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

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 :

M : = X / 6 + (a – 2 * (b / 3) ^ 5 + f) ^ 2

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	Not
2	Not	A	Not (
3	(	A	Not (
4	B	A B	Not (OR
5	OR	A B	Not (OR
6	Z	A B Z	Not (OR
7	OR	A B Z OR	Not (OR Not
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
14	)	A B Z OR G E < Not OR	.....
15	...؟	A B Z OR G E < Not OR Not	

٤- تطبيقـــات أخرى

يستخدم المكدس كهيكل لخزن المعلومات التي نحتاج استرجاعها بصورة معكوسة (ترتيب 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
Do
{ Read a character
If character <> '.'
Then push the character onto stack
}
While (character == '.')
While stack is not empty
Do
{ Pop the stack
Print the character
}
End

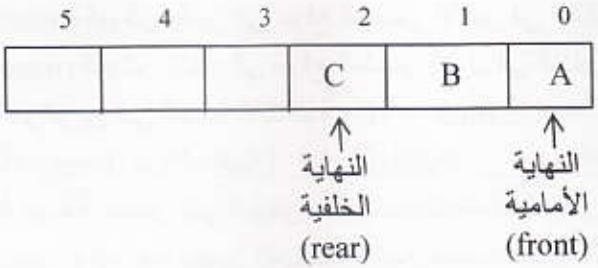
تمرين : اعتمد خوارزمية التمرين السابق واكتب الأيعازات اللازمة لقراءة اية جملة (string) تنتهي بالرمز (.) ، ثم اطبع تلك الجملة بترتيب معكوس وذلك باستخدام المكدس .

Print Reverse

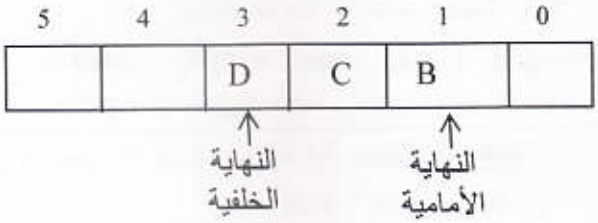
```
{
const char dot ='.';
char  ch ;
int  top = -1  ;
 do
{
      cin>>ch  ;
      if( ch != dot )
          Push( ch ) ;
} while ( ch != dot )

while ( top != -1 )
{
         pop( ) ;
        cout<<item  ;
}
}
```

3-3 الطابـــور Queue

هو هيكل تسلسلي (sequential) تكون فيه عمليات الإضافة في النهاية الخلفية (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) وعند إضافة عنصر جديد يكون موقعه بعد (C) ، أما عند حذف عنصر من الطابور 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 الفعاليات التي تتضمن جدولة الأعمال حسب ترتيب وصولها أو طلبها ويمكن تلخيص هذا بالعبارة الآتية :

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

أي أن من يصل أولا يحصل على الخدمة أولا وتسمى عملية الإضافة إلى الطابور (ENQueue) أو (Insertion) أما عملية الحذف فتسمى (DEQueue) أو (Deletion).

1-3-3 تمثيل الطابور باستخدام المصفوفة

Array Representation of Queue

يمكن تمثيل الطابور حاسوبيا باستخدام مصفوفة أحادية بالسعة المطلوبة (size) وبالنوع المناسب لنوع البيانات (Data Type) التي ستخزن فيه (float, int ... الخ) مع استخدام ما يلي :

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 = -1 , rear = -1) 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 الموقع الذي كان يشير اليه المؤشر (front) ، ولنفترض لدينا الطابور (Q) سعته (6) عناصر وننفذ عليه سلسلة العمليات الآتية :-

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

يعرف الطابور برمجيا بلغة C++ باستخدام العبارات البرمجية التالية :-

```
const  int size = 10 ;   { or any other value }
char  item ;             { or any other type }
char  queue[size ] ;
int  rear , front  ;
```

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-1) 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

تعتمد الخطوات الآتية لحذف عنصر واحد من الطابور :

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

2- اخذ العنصر من الموقع الذي يشير إليه المؤشر (front) وخزنه وقتيا 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 .

2-3-3 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 - 1 )
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 = -1
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 = -1 , rear = -1)

front ⟵ -1

rear ⟵ -1

3-3 البرامج الفرعية لتنفيذ عمليات الطابور Queue's Subprograms

إن الطابور هو هيكل بياني خصائصه (attributes) هي : حجمه (size) ، ونوع البيانات المخزونة فيه (data type) ، وأهم عملياته (actions) هي : الإضافة (Enqueue) ، والحذف (Dequeue) ، ويمكن تعريفه برمجيا بلغة C++ بما يسمى (class) ، وستكون مكوناتها كالأتي :

1- وجود التعريفات البرمجية التالية

```
const int size = 10 ;     { or any other value }
char item ;               { or any other type }
char queue[size] ;
int rear , front ;
```

2- برنامج فرعي(function) لاخلاء الطابور

```
void initialize( )
{
    rear = -1 ;
    front = -1 ;
}
```

لاحظ عدم الحاجة للمرور على جميع مواقع المصفوفة والاكتفاء فقط بجعل قيمة كل من المؤشرين مساوياً (-1) ، وهذا البرنامج الفرعي (function) يُستدعى في بداية البرنامج لجعل الطابور خالياً .

ويمكن أيضا" استخدام (constructor) في لغة C++ لتحقيق نفس الهدف بأستخدام اسم الـ class بالصيغة التالية :

```
Queue( )
{
rear= -1 ;
front= -1 ;
}
```

3- أضافة عنصر واحد الى الطابور (Enqueue)

```
void  enqueue( char item )
{
    if( rear == size-1 )
       cout<<"\n   Queue is Full  " ;
    else
    {
       rear ++ ;
       Queue[rear]= item  ;
    }
     if ( front == -1 )
         front= 0 ;
}
```

هذه الخطوة لمعالجة اول اضافة (اول عنصر) الى الطابور ليصبح المؤشر front يشير اليه

ملاحظة : يمكن استدعاء هذا البرنامج الفرعي داخـــل البرنـامج الرئيسي (main) بأي عدد من المرات باستـــخدام أحد ايعازات التكرار مثل (For – While) بقدر عدد العناصر المطلوب إضافتها .

ب- حذف عنصر واحد من الطابور (Dequeue)

```
void dequeue ( char item ) ;
{
    if ( front == -1 )
        cout<<"\n  The queue is Empty " ;
    else
    {
        item = queue[front] ;
        if (front == rear)
        {
            front = -1 ;
            rear  = -1 ;
        }
        else
        front = front + 1
    }
}
```

هذه الخطوة لمعالجة حذف آخر عنصر في الطابور عندما يكون كلا المتغيرين front, rear يشيران اليه ولكي نعيد المؤشرين الى البداية

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

4-3-3 تمثيل الطابور باستخدام Structure

يستخدم الـ (structure) في تمثيل الطابور والمؤشرين (front , rear) في هيكل بياني 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 C++ كما يأتي :-

أسئلة الفصل

1. Let S be a stack of size (15) characters , write a C++ segment of code to add (push) one element.
2. Write a procedure to (pop) three elements from the stack (TABLE) of size (20) integers.
3. Let S for stacking an element in the stack , and U for unstacking. If the order of the stack input stream is 1 2 3 4 5.
   a - what is the output if we execute the following operations SSUUSSSUUU
   b - which of the following permutations can be obtained as output stream (explain the reason for each case).
   i - 5 1 3 2 4
   ii – 2 3 5 1 4
   iii – 3 2 1 5 4
4. If the stack input stream is A B C D E F what is the sequence of operations to get the output C B D E F A ?
5. Write a function to check if the stack (St) is empty.
6. Explain how we can use the structure for stack representation ?
7. State the main applications of the stack?
8. Explain how we can use stack to achieve the processing of Subprograms calls in computer programming ?

Explain in details how we can use the stack in processing of a program.

10. Convert the following infix expressions into postfix notations using two stacks:
    i) a + b ^ 2/4-(c *5/8-f)^3
    ii) m ^3 OR n-b/2 AND (m + n)
    iii) a + b + c * (-d /3 ^ 4) AND Not F
    iv) a ^ (b/2) OR (x + y/3 – w) AND (c ^ 2) ^ 3
    v) x ^ N + (M – P * 4) ^ F AND (-b) ^ 2 / C + 6
    vi) a + b – c OR (x * 2 ^ n) – m/p AND F ^ 2
    vii) a ^ (b/2) AND (x – y/3 + w) OR (c^3) ^2
    viii) N ^ x – (p * 4 + M)/F OR –c / b ^ 2
    ix) B – a + c AND n ^ x – (p/M OR F ^ 2)
11. Execute the following postfix notation using stack ab*cde^/+, when a=5,b=6,c=8,d=2,e=2.
12. Execute the following infix notation using stack a / (b*c/2)^4+m If a=10, b=8 , c=4, m=20.
13. Write an algorithm that reads in a string and prints its characters in reverse order [Note: the string terminator is a '.' which should not be printed as a part of the reversed string]
14. Write a C++ segment of code that reads in a string and prints its characters in reverse order [Note: the string terminator is a blank, which should not be printed as a part of the reversed string] .
15. Write a C++ segment of code to add an element to the queue called (LINE) of size (30) characters.

16. Write a C++ segment of code to delete an element from the circular queue called (BOB) of size (15) integers .
17. What are the main differences between simple queue and circular queue ? Explain that in detail.
18. What are the main applications of the queue in the computer field?

الفصل الرابع
الهياكل الموصولة
Linked Structures

4-1	التخصيص الخزني	Storage Allocation
4-1-1	الخزن التسلسلي	Sequential Storage Allocation
4-1-2	الخزن الديناميكي	Dynamic Storage Allocation
4-1-3	المقارنة بين الخزن التسلسلي والخزن الديناميكي	
4-2	المؤشرات	Pointers
4-3	القائمة الموصولة	Linked List
4-4	المكدس الموصول	Linked Stack
4-5	الطابور الموصول	Linked queue
4-6	القائمة الموصولة الدائرية	Circular Linked list
4-7	القائمة الموصولة المزدوجة	Double linked list

4-1 التخصيص الخزني Storage Allocation

4-1-1 التخصيص الخزني التسلسلي
Sequential Allocation of Storage

ان ابسط الطرق لخزن القائمة الخطية هو استخدام الخزن التسلسلي في ذاكرة الحاسوب أي يتم الخزن في مواقع متتابعة (متسلسلة) ويمكن ان نعرف موقع أي عنصر اذا عرفنا موقع العنصر الاول الذي هو عنوان البداية (Base Address) ، ومواقع العناصر التالية ستكون منسوبة له . فالعنصر (k) سيكون موقعه تالياً لموقع العنصر (k – 1) وهكذا بقية العناصر .

المزايا : Advantages

1- اكثر سهولة في التمثيل والتطبيق .
2- يكون اقتصادياً اكثر لأنه يستخدم مساحة خزنية اقل .
3- اكثر كفاءة في الوصول العشوائي .
4- مناسب جداً عند التعامل مع المكدس .

المساوئ : Disadvantages

1- صعوبة تنفيذ عمليات الاضافة والحذف .
2- يتطلب التعريف المسبق وتحديد عدد العناصر المطلوب خزنها .

4-1-2 التخصيص الخزني الديناميكي
Dynamic Allocation of Storage

ان الطريقة الاخرى للخزن هي استخدام رابط (link) او مؤشر (pointer) مع كل عنصر يحتوي عنوان موقع العنصر التالي لذلك لا توجد ضرورة لخزن البيانات في مواقع متعاقبة (متسلسلة) بل يمكن خزن أي عنصر بياني في أي موقع ، ولهذا فكل عنصر (عقدة) يتكون من جزأين هما :

الجزء الاول : يحتوي البيانات (Data)
الجزء الثاني : حقل يحتوي على عنوان موقع العنصر التالي (link)

x :

Data (x)	Link (x)

فالعنصر x : يتكون من الجزأين link (x) , Data (x)

المزايا :

- سهولة تنفيذ عمليات الاضافة والحذف لأي عنصر اذ لا يتطلب اكثر من تحديث قيمة حقل المؤشر الذي يعطي عنوان الموقع التالي ..

المساوئ :

-يحتاج الى مساحة خزنية اكبر لتمثيل حقل المؤشر اضافة الى البيانات الاساسية

3-1-4 المقارنة بين الخزن التسلسلي والخزن الديناميكي

يمكن ان تتركز المقارنة في النقاط التالية :

أ- المساحة الخزنية Amount of Storage

ان اسلوب الخزن الديناميكي يحتاج الى مساحة خزنية اكبر لأن كل عنصر في الهيكل البياني يحتاج الى مؤشر (أي موقع خزن اضافي) يحتوي عنوان موقع العنصر التالي ، وهكذا لجميع العناصر .

ب – عمليات الاضافة والحذف

ان هذه العمليات اسهل تنفيذاً في الخزن الديناميكي لأنها لا تتطلب غير تغيير قيمة المؤشر ليحتوي عنوان موقع العنصر بعد الاضافة او الحذف . اما في الخزن التسلسلي فأنه يتطلب عمليات تزحيف (shifting) العناصر .

جـ- الوصول العشوائي للعناصر Random Access

ان اسلوب الخزن التسلسلي يعتبر افضل في الوصول عشوائياً لأي عنصر من عناصر الهيكل البياني مباشرة (k- th element from the start) اما في الخزن الديناميكي فأنه يتطلب البدء من اول عنصر ثم العناصر التالية بالتتابع لحين الوصول الى موقع خزن العنصر المطلوب .

د- الدمج والفصل Merge & Split

في الخزن الديناميكي تكون هذه العمليات اسهل تنفيذاً وذلك بتغيير قيمة المؤشر للعناصر (العقد) في مواقع الدمج او الفصل اما في الخزن التسلسلي فالعمل اكثر تعقيداً اذ قد يتطلب تزحيف العناصر واعادة تنظيم الخزن (Reorganization of Storage) .

Pointers المؤشرات

لكي نستطيع فهم تمثيل وعمل الهياكل البيانية باستخدام الخزن الديناميكي لابد من أولاً يمكن تعريف المؤشرات وطريقة استخدامها برمجيا.

عند التعامل مع المصفوفات (arrays) سبق ان استخدمنا الدليل (index) للوصول الى موقع احد عناصرها ، أي ان الدليل عبارة عن متغير استخدمه البرنامج في الوصول الى الموقع المطلوب ويستخدمه المترجم (compiler) كعنوان لموقع معين في الذاكرة بنفس الاسم (اسم الدليل) .

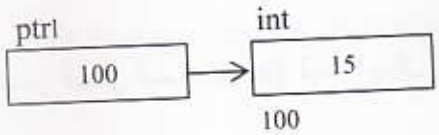
ان هذا المؤشر (الدليل) يعتبر مؤشرا نسبيا (relative) أي يدلنا على موقع العنصر بالنسبة الى موقع بداية (Base Address) خزن المصفوفة في الذاكرة .

هناك حالات تستوجب بناء هياكل بيانية مختلفة السعة الخزنية خلال مرحلة تنفيذ البرنامج (program execution stage) وهي الهياكل البيانية الموصولة linked data structures) اذ يكون لكل عنصر بياني فيها حقل (جزء) اضافي (link) يستعمل كمؤشر يشير الى موقع العنصر التالي ، أي ان هذه الهياكل تكون (تنمو) بإضافة عناصر جديدة لها بصورة ديناميكية خلال مرحلة تنفيذ البرنامج عند الحاجة لمواقع جديدة .

برمجيا بلغة C++ يكون تعريف المؤشرات كالأتي :

```
int  *ptr1 ;
```

يعني تعريف المتغير (ptr1) كمؤشر ، وهو سيشير الى عنوان موقع (address) يحتوي بيانات نوعها (int) ويقرأ هكذا (ptr1 مؤشر لموقع من نوع int) .



وبهذه الطريقة يمكن تعريف مؤشرات أخرى لكل نوع متغير ، فمثلا" الأيعاز التالي:

```
char *ptr2 ;
```

سيعني تعريف المؤشر (ptr2) ليشير الى عنوان موقع (address) يحتوي بيانات نوعها (char) ، وهكذا لكل نوع بياني اخر float ، double ، ... الخ .

ومن الصيغ البرمجية الأخرى للتعامل مع المؤشرات ما يلي :

أ / أذا كان لدينا متغير مثل

```
int var1 ;
```

فلمعرفة عنوان الموقع يكون باستخدام الأيعاز

```
ptr = & var1
```

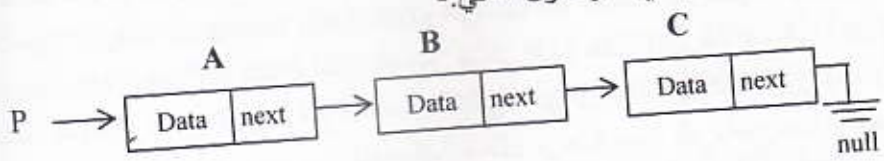
أذ ان أشارة (&) تعني عنوان موقع (address) المتغير (var1) .

ب / لطبع محتويات الموقع الذي يشير اليه المؤشر (ptr1) يستخدم الأيعاز التالي :

```
cout<< *ptr1 ;
```

أذ أن (*ptr1) تعني محتوى الموقع الذي يشير اليه (ptr1) وهو (15) كما في الرسم السابق.

لنأخذ الهيكل البياني الموصول التالي:-



لان كل عنصر يتكون من جزأين فيعرف كـ (Structure) وفق الاتي :

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

هذا التعريف يعني ان العنصر البياني اسمه (node) لهذا الهيكل (structure) يتكون من جزأين :

الجزء الاول : Data ونوعه (int) ، ويمكن تعريفه بأي نوع اخر حسب نوع البيانات .

الجزء الثاني : next وهو نوع مؤشر سيحتوي عنوان موقع (address) العنصر التالي .

مثلاً :
تعريف هيكل بياني موصول (Linked Structure) تتكون عناصره البيانية من حقول هما (اسم الطالب – عمره).
سيكون تعريفه وفق الآتي :

```
structure   student
{
    char name ;
    int  age ;
   student *next ;
} ;
```

تعريف قيد الطالب ومكوناته هي :
الاسم : نوع (string)
العمر : نوع عدد صحيح
المؤشر : نوعه (int) المعرف أعلاه

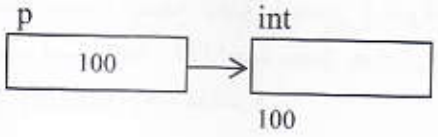
ويظهر الهيكل البياني بالرسم كالاتي:

لاحظ ان مؤشر العنصر الاخير قيمته (null) ويعني لا شئ بعده .

أ- **البرنامج الفرعي (new):**
توفر لغة C++ استخدام (new) للحصول على مواقع خزنيه في الذاكرة ، وتحديد المؤشر (pointer) الذي يشير الى تلك المواقع ، ولنأخذ المثال التالي للتوضيح .

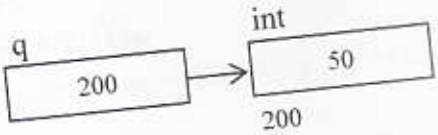
+ تعريف المؤشرين p , q لموقعين ذي صفة int
```
int *p , *q ;
```

+ استخدم new لحجز موقع في الذاكرة لمتغير نوعه int ويكون عنوان الموقع في المؤشر p
```
p = new int ;
```



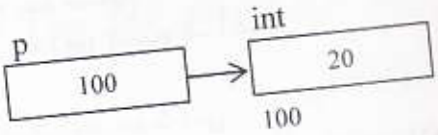
+ استخدم new لحجز موقع في الذاكرة لمتغير نوعه int وقيمته (50) ويكون عنوان الموقع في المؤشر (q)

```
q = new int(50) ;
```



+ ضع القيمة (20) في الموقع الذي يشير اليه p

```
*p = 20;
```



+ أطبع محتويات الموقع الذي يشير اليه المؤشر p وستكون (20) كما في الرسم السابق

```
cout<<*p<< ;
```

+ أطبع محتويات الموقع الذي يشير اليه المؤشر q وستكون (50) كما في الرسم السابق

```
cout<<*q<< ;
```

ب – البرنامج الفرعي (delete)

يستخدم هذا الأيعاز في لغة C++ ، ووظيفته هو تحرير الموقع الذي حجز باستخدام (new) ويكتب بالصيغة التالية :

```
delete  p  ;
```

ليعني تحرير موقع الذاكرة الذي يشير اليه المؤشر (p) أي ان البرنامج لا يحتاج الى استخدامه وهذا يحقق الخاصية المهمة للخزن الديناميكي الذي يقنن استخدام المساحة الخزنية بقدر الحاجة الفعلية فيسمح بخلق (حجز) الموقع عند الحاجة اليه وحذفه او تحريره عند انتهاء استخدامه . لذا فعند تنفيذ عملية حذف أي عنصر من الهيكل الموصول يستخدم (delete) بعده مباشرة .

القائمة الموصولة Linked List

هي مجموعة من العناصر (العقد) التي كل منها يحتوي البيانات (Data) والمؤشر (Next) الذي يشير الى العنصر (العقدة) التالي في القائمة .

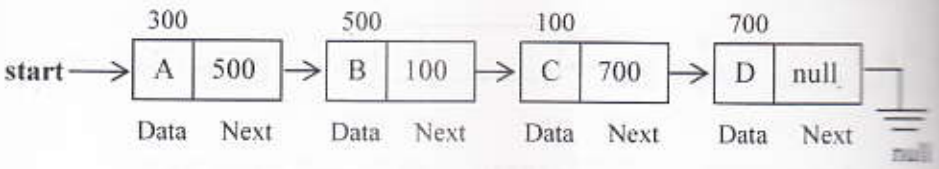
العنصر (x) يتكون من الجزأين Next , Data

x :

Data (x)	Next (x)

مثال :

لدينا قائمة موصولة تتكون من اربعة عناصر كالآتي :-



نلاحظ ما يلي :

start: هو المؤشر الرئيسي الذي يشير الى بداية اول عنصر في القائمة وستكون قيمته في هذا المثال (300).

محتويات حقل المؤشر للعنصر الاول هو (500) ويدل على موقع العنصر الثاني. next (A) = 500

محتويات حقل المؤشر للعنصر الثاني هو (100) ويدل على موقع العنصر الثالث. next (B) = 100

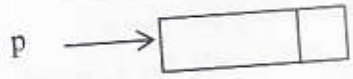
محتويات حقل المؤشر للعنصر الثالث هو (700) ويدل على موقع العنصر الرابع. next (C) = 700

محتويات حقل المؤشر للعنصر الرابع هو (null) ويدل على انه لايوجد عنصر بعد العنصر الرابع. next (D) = null

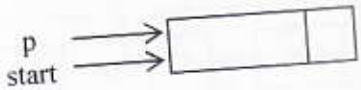
فيما يأتي بعض الصيغ البرمجية (مقطع من برنامج) (segment of code of C++ program) لتوضيح كيفية تكوين (خلق) هيكل موصول (Linked Structure) او تنفيذ بعض العمليات كالاضافة والحذف مع استخدام التعريف الوارد في صفحة (108).

1. تكوين قائمة موصولة من عنصر واحد Create a Linked list (of one node)

p = new node ;



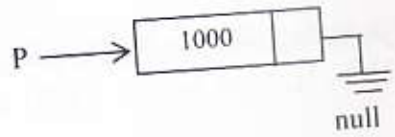
start = p ;



cin>> p→data ;

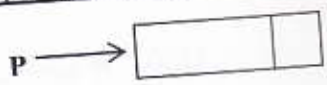
ادخال بيانات العنصر ولتكن (1000)

p →next = null ;

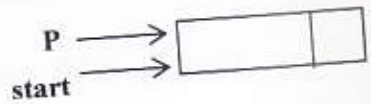


2. تكوين قائمة موصولة من عنصرين Create a linked list (of two nodes)

p =new node ;



start= p ;

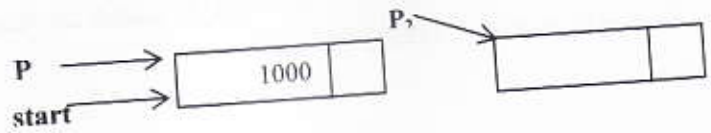


cin>>(p→data) ;

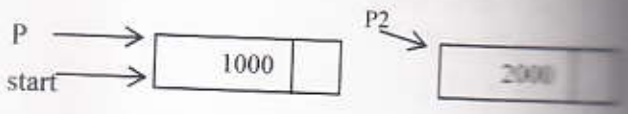
ادخال بيانات العنصر الأول ولتكن (1000)

تكوين موقع لعنصر جديد يشير اليه p2

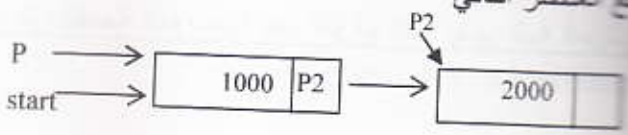
p2=new node ;



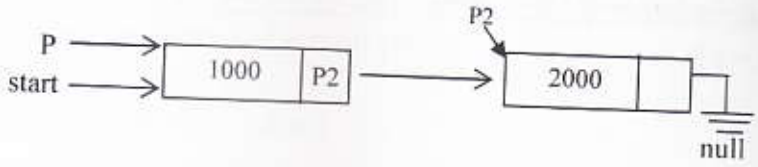
cin<<(p2-data) ;

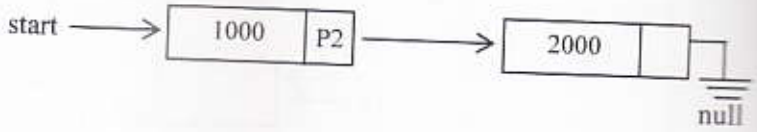


p→next = p2 ;



P2→next = null ;





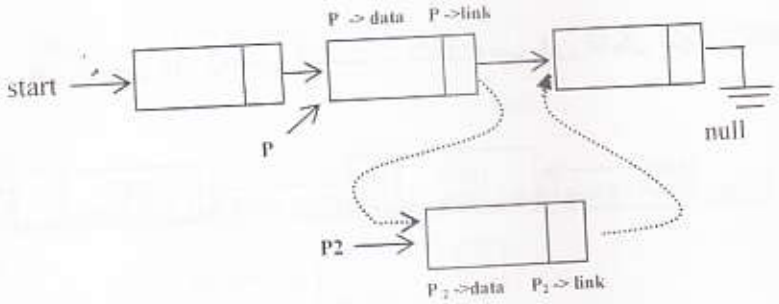
Create a Linked list (of N node)

```
p = new  node  ;
start = p  ;
cout<<"\n  How many elements you like to create : "  ;
cin>>n  ;
for( int  i =1 ; i <= n ;  i++ )
{
    cin>>(p→data)  ;
```

```
if ( i != n )
    p2 = new node ;
else   p2= null ;
p→link = p2 ;
p = p2 ;
```

4.أضافة عنصر واحد الى القائمة الموصولة بعد الموقع الذى يشير اليه المؤشر(P)

```
void addafterp( ):

{
  p2 = new node ;
  cin>>(p2→data) ;
  p2→link = p→link ;          [ * ]
  p→link = p2 ;
}
```



[*] موقع هذه الخطوة مهم جدا" ، أذ انها تجعل قيمة المؤشر للعنصر الجديد هي نفس قيمة مؤشر العنصر السابق (p→link) قبل تغييره في الخطوة التالية .

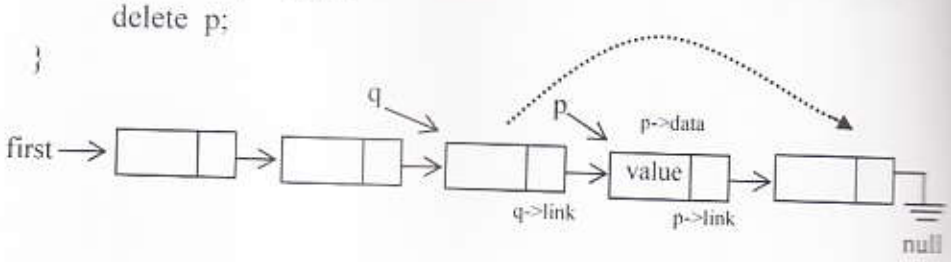
(عرض) جميع عناصر القائمة الموصوله

```
void display( )
{
     node *p ;
     p = first ;
     while ( p != null )
     {
          cout<<(p→data) ;
           p = p→link ;          تحريك المؤشر p الى الموقع التالي
     }
}
```

حذف عنصر (ذو قيمة محددة لتكن value) من القائمة الموصوله

To delete an element of certain value

```
void deleteval( char value )
{
     node *p , *q ;
     p = first ;
     while ( p→data != value )
     {
          q = p ;
          p = p→link ;
     }
     q→link = p→link ;
     delete p;
}
```



لاحظ هنا ما يأتي :

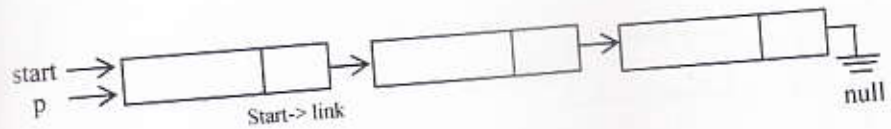
+ استخدام المؤشر (p) في موقع ما ، ثم يتبعه المؤشر (q) في الموقع الذي يسبقه ، ويتحرك المؤشران معا" لحين الوصول الى الموقع المطلوب حذفه وفق الشرط المحدد بالعبارة الشرطية (while) . ويمكن أستخدام هذه الصيغة في معظم عمليات الحذف بعد صياغة الشرط المناسب.

+ تغيير قيمة مؤشر العنصر في الموقع السابق الذي يشير اليه المؤشر (q) ليشير الى موقع العنصر التالي بعد الموقع الذي يشير اليه (p) ، لأن العنصر الذي يشير اليه (p) سيحذف .

+ من المهم أستخدام (delete p) لتحرير الموقع الذي يشير اليه (p) واعادته للذاكرة لأستخدام لاحق .

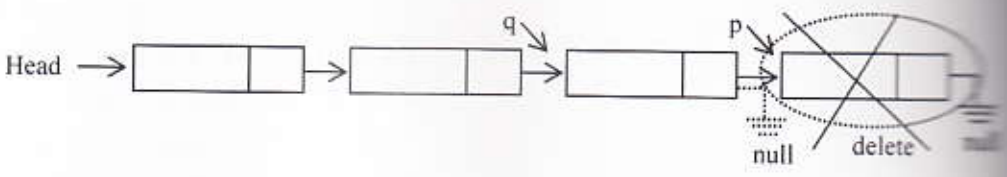
7.حذف العنصر الأول في القائمه الموصوله Delete the first element

```
node *p ;
p = start ;
start = start→link ;
delete p ;
```



Delete the last element حذف العنصر الأخير في القائمه الموصوله

```
node *p , *q ;
p = head ;
if ( p→link == Null )
{
      delete p ;
      head = null ;
}
else
{
      while ( p→link != Null )
      {
            q = p ;
            p = p→link ;
      }
      q→link = Null ;  delete p ;
}
```



إضافة عنصر واحد الى نهاية القائمة الموصوله

Segment of code of C++ program to add one element to the end of the linked list pointed by (start) .

```
p = start ;
while ( p→link != NULL )
        p = p→link ;
q = new  node ;
cin>>(q→data) ;
q→link = NULL ;
p→link = q ;
```

إضافة عنصر بعد الموقع الذى ترتيبه (n) في القائمه الموصوله
عندما يكون عدد عناصر القائمه أكبر أو يساوى (n)

```
cout<<" \n  Enter the position ( n ) : " ;
cin>>n ;
p = head ;
for ( int i = 1 ; i <= n-1 ; i++ )
      p = p→link ;
q = new node ;
cin>>(q→data) ;
q→link = p→link ;
p→link = q ;
```

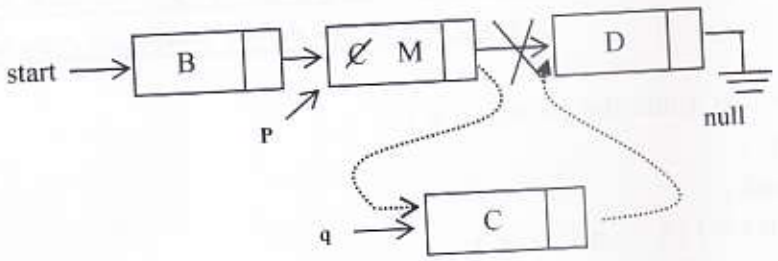
11. أضافة عنصر قبل العنصر في الموقع (p) للقائمه الموصوله

أن هذه الحالة تفترض أن المؤشر الرئيسي للقائمه مجهول وكذلك بيانات العناصر مجهولة ، لذا فأن فكرة الحل هنا هي كالأتي :

+ خلق العنصر الجديد
+ أضافة العنصر الجديد بعد العنصر في الموقع (p)
+ استبدال قيمة العنصر الجديد مع قيمة العنصر في الموقع (p)

```
q = new node ;
q→data = p→data ;
cin>>( p→data ) ;

q→link = p→link ;
p→link = q ;
```

قراءة العنصر الجديد وليكن (M) بدلا" من قيمته السابقة (C)



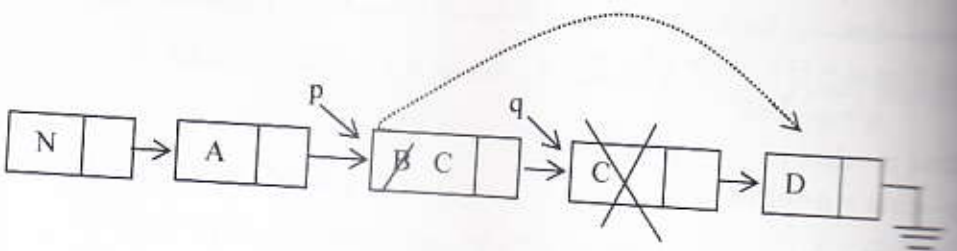
- لاحظ أن عناصر القائمه أصبحت BMCD بعد ان كانت BCD

حذف عنصر في الموقع (p) من القائمه الموصوله

في هذه الحالة نفترض أن كل من المؤشر الرئيسي للقائمه و بيانات العناصر غير معروفين لنا فان فكرة الحل هنا هي كالأتي :

- نسخ قيمة العنصر في الموقع اللاحق الى الموقع (p) .
- حذف العنصر في الموقع اللاحق الذي نسخنا قيمته في الخطوة السابقة.

```
q = p→link ;
p→data = q→data ;
p→link = q→link ;
delete q ;
```



- لاحظ أن عناصر القائمه أصبحت NACD بعد ان كانت NABCD

تمرين : أضافة عنصر واحد الى قائمه موصوله عناصرها مرتبة (ordered linked list) بصورة تصاعديه (Ascending) على ان تبقى عناصر القائمه مرتبة بعد الأظافة .

الحل :

```
cin>>value ;
p = start ;
while ( p→data < value )
{
      q = p ;
      p = p→link ;
}
```

```
t = new node ;
t→data = value ;
t→link = p ;
q→link = t ;
```

تمرين : استبدل (Exchange) قيمة عنصر في موقع معين (i) للقائمه الموصوله (start) مع قيمة العنصر في موقع أخر (j) في نفس القائمه على ان يكون (i < j)

الحل :

```
p =start ;
for ( int  n = 1 ; n <= i-1 ; n++ )
      p = p→link ;
q = p→link ;
for ( int  n = i+1 ; n <= j-1 ;  n++ )
      q = q→link ;
x = ( p→data ) ;
p→data = q→data ;
q→data = x ;
```

تمرين : اكتب برنامج فرعي (function) لتنفيذ عملية دمج (merge) القائمه الموصوله التي مؤشرها الرئيسي (y) في نهاية عناصر القائمه الموصوله التي مؤشرها الرئيسي (x) .

الحل :

```
void  merge( char  *x, *y )
{     node  *z , *p ;
      if ( x==NULL )
          z = y ;
      else
      {    z = x ;
           if ( y != NULL )
           {   p = x ;
               while ( p→link != NULL )
                       p = p→link ;
```

```
            p→link = y ;
        }
    } return z ;
}
```

التعليق : أكتب برنامج فرعي (function) لتجزئة (split) القائمه الموصوله التي مؤشرها الرئيسي (start) الى قائمتين موصولتين أحداهما (first) تحتوي على جميع العناصر في المواقع الفردية للقائمه الأصليه ، والقائمه الثانيه (second) تحتوي على جميع العناصر في المواقع الزوجيه للقائمه الأصليه .

الحل :

```
void split ( char *start )
{   node *first , *second , *p , *n , *m ;  int L ;
    first = NULL ; second = NULL ;
    p = start ;   L = 0 ;
    while ( p != NULL )
    {   L = L + 1 ;
        if ( L%2 != 0 )
        {   if ( L == 1 )
            {   first = p ;
                n = first ;
            }
            else
            {   n→link = p ;
                n = p ;
```

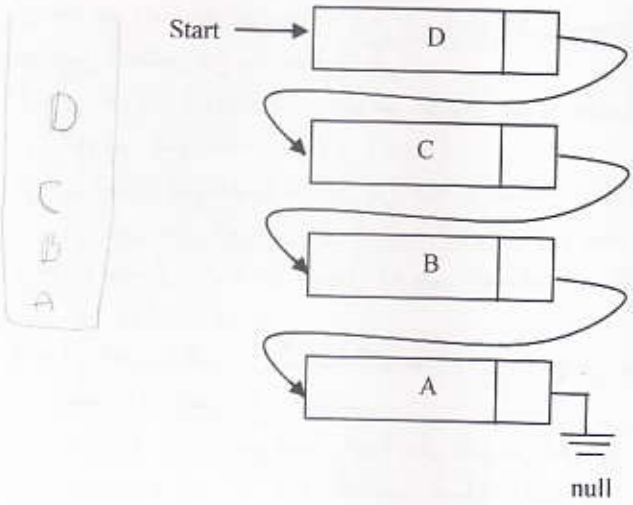
```
            }
        }
        else
        {       if( L == 2 )
                {       second = p ;
                        m = second ;
                }
                else
                {       m→link = p ;
                        m = p ;
                }
            }
        p = p→link ;
        }
n→link = NULL ;  m→link = NULL ;
return first , second ;
```

4-4 المكدس الموصول Linked Stack

يمكن الأستفادة من خصائص الخزن الديناميكي لتمثيل المكدس باعتباره حالة خاصة من القائمة الخطيه التي تكون عمليات الأضافة و الحذف من نهاية واحدة (هي النهاية المفتوحة) .

أن مبدأ عمليات الأضافة و الحذف هي نفسها التي سبق ذكرها ، الا أن الفرق يكون في طريقة التمثيل في الذاكرة .

والشكل التالي يبين مكدس موصول ذو اربعة عناصر



- المؤشر (start) يشير الى قمة المكدس حيث العنصر (D) ، وهو يمثل النهاية المفتوحة حيث تنفذ عمليات الأضافة و الحذف .
- العنصر (A) في قعر المكدس ، وهو يمثل النهاية المغلقة ، مع ملاحظة أن حقل المؤشر (link) لهذا العنصر هو (NULL) اذ لم يسبقه شيء .

برنامج فرعي (function) لأضافة عنصر الى المكدس الموصول

```
void pushls( int item )
{
    node *p ;
    p = new node ;
    p→data = item ;
    if ( start == NULL )
        p→link = NULL ;
    else
        p→link = start ;
    start = p ;
}
```

أن هذا البرنامج الفرعي يعتمد التعريف الوارد في الصفحة (108) فيما يتعلق بعناصر المكدس مع ملاحظة ما يلي :

1– أن المؤشر (start) هو المؤشر الرئيس لبداية عناصر المكدس ، اي المؤشر الذي يناظر المؤشر (top) .

2 – لا نحتاج الى خطوة للتحقق من امتلاء المكدس (stack full) لأننا نستطيع خلق العنصر عند الحاجة اليه ومن ثم ربطه بالعنصر السابق له .

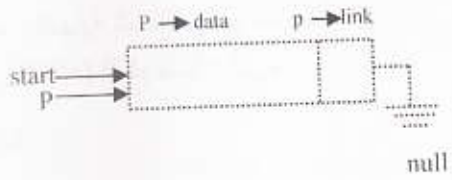
3 – بموجب أول ايعازين سيُخلق العنصر المطلوب أضافته (المؤشر p يشير اليه) وتُدخل بياناته .

4 – أن العبارة التي تلي الجملة الشرطية (.... if) هي لمعالجة الحالة عند خلق اول عنصر في المكدس .

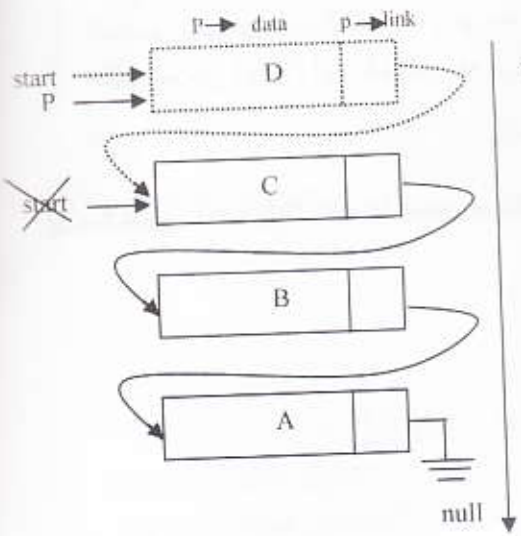
5 – عبارة (else) هي لجعل قيمة حقل المؤشر للعنصر الجديد (المطلوب أضافته) تشير الى موقع العنصر السابق والذي يشير اليه (start) .

6 – الخطوة الأخيرة هي لتحديث المؤشر (start) ليشير الى العنصر الجديد بعد ان اصبح في مقدمة المكدس .

7 – الرسم التوضيحي يبين كيفية تنفيذ الخطوات أعلاه .



حالة خلق أول عنصر في المكدس
(بداية تكوين المكدس)



حالة أضافة اعتيادية لعنصر إلى مكدس
موجود اصلا ويتكون من ثلاثة عناصر

البرنامج الفرعي (function) لحذف عنصر من المكدس الموصول

```
int popls( )
{
    int item ;
    if( start == NULL )
        cout<<"\n  the linked stack is empty " ;
    else
    {
        q = start ;
        item = q→data ;
        start = q→link ;
        delete q ;
    }
return item ;
}
```

ن خطوات هذا البرنامج الفرعي هي :

1 – عندما يكون المؤشر الرئيسي (start = NULL) فأن المكدس خال وعملية الحذف لن تعد ممكنة .

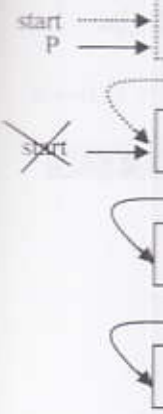
2 – أستخدم المؤشر (q) ليشير الى بداية المكدس (أول عنصر في المكدس).

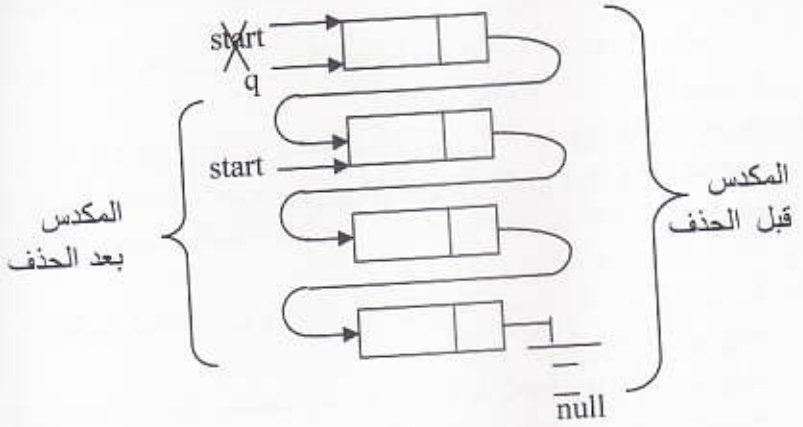
3 – أخذ (سحب) قيمة العنصر الأول الموجودة في الحقل (q→data) وخزنها وقتيا في المتغير (value) .

4 – تحديث قيمة المؤشر (start) ليشير الى موقع العنصر التالي المحددة بالحقل (q→link) .

5 – تحرير الموقع الذي كان يشغله العنصر المحذوف والذي يشير اليه المؤشر (q) باستخدام (delete q) .

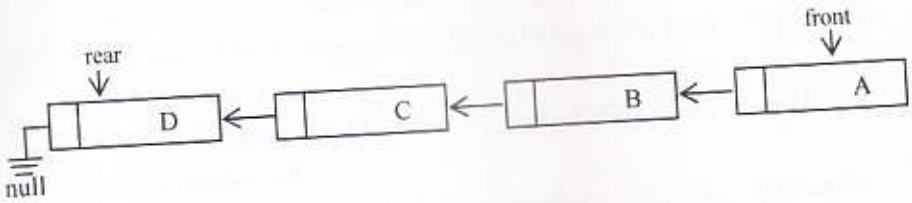
6 – الرسم التوضيحي يبين كيفية تنفيذ الخطوات أعلاه .





5-4 الطابور الموصول Linked Queue

كما سبق ان مثلنا المكدس باستخدام الخزن الديناميكي يمكن تمثيل الطابور بنفس الطريقه مع وجود المؤشرين (front) ، (rear) وسيظهر الطابور كالأتي :



- المؤشر (front) يشير الى أول عنصر في الطابور.
- المؤشر (rear) يشير الى أخر عنصر في الطابور.
- كل عنصر من عناصر الطابور الأربعة (D,C,B,A) فيه حقل يحتوي قيمة المؤشر الى العنصر التالي.
- مؤشر العنصر الأخير قيمته (null) ، أذ لايوجد بعده عناصر.

البرنامج الفرعي (function) لأضافة عنصر الى الطابور الموصول

```
void addlq( )
{
      p = new node ;
      cin>>(p→data) ;
      p→link = NULL ;
      if ( rear == NULL )
            front = p ;
      else
            rear→link = p ;
      rear = p ;
}
```

ان خطوات هذا البرنامج الفرعي هي كالأتي :

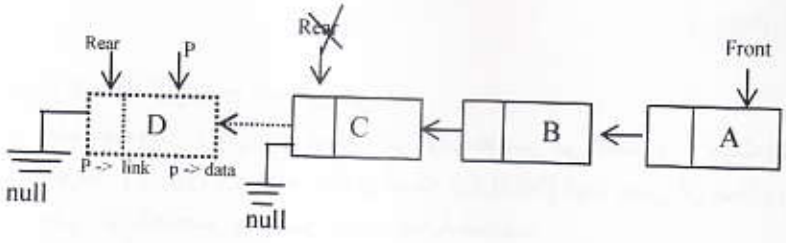
1 – أن الخطوات الثلاث الأولى هي لغرض خلق (تكوين) العنصر وأدخال قيمته وجعل قيمة المؤشر (NULL).

2 – العبارة التي تلي الجملة الشرطيه (... if) هي لمعالجة الحالة عند خلق أول عنصر في الطابور ، وسيشير اليه المؤشر (front).

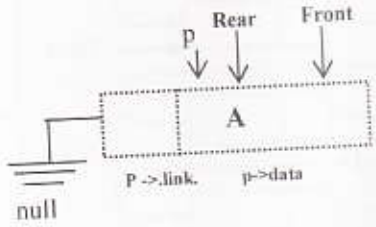
3 – في عبارة (else) يتم تحديث قيمة حقل المؤشر للعنصر الأخير في الطابور جعله يشير الى العنصر الجديد الذي يشير اليه (p).

4 – أن الخطوة الأخيرة هي تحديث قيمة المؤشر (rear) ليشير الى العنصر الجديد (المضاف) بعد أن اصبح هو الأخير.

5 – الرسم التوضيحي التالي يبين أضافة العنصر (D).



6 – والرسم التوضيحي التالي يبين تكوين أول عنصر في الطابور وهو (A).



البرنامج الفرعي (function) لحذف عنصر من الطابور الموصول

```
void deletelq( )
{
    int item ;
    p = front ;
    if( p == NULL )
        cout<<"\n  the linked queue is empty  " ;
    else
    {
        q = p→link ;
        item = p→data ;
        delete p ;
        front = q ;
        if ( front == NULL )
            rear = NULL ;
    }
return item ;
}
```

ان خطوات هذا البرنامج الفرعي هي :

1 – أستخدام مؤشر وقتي (p) ليشير الى أول عنصر في الطابور ، حيث يشير المؤشر (front) ، وعندما تكون قيمته (NULL) فهذا يعني أن الطابور خالي من العناصر ولايمكن تنفيذ عملية الحذف .

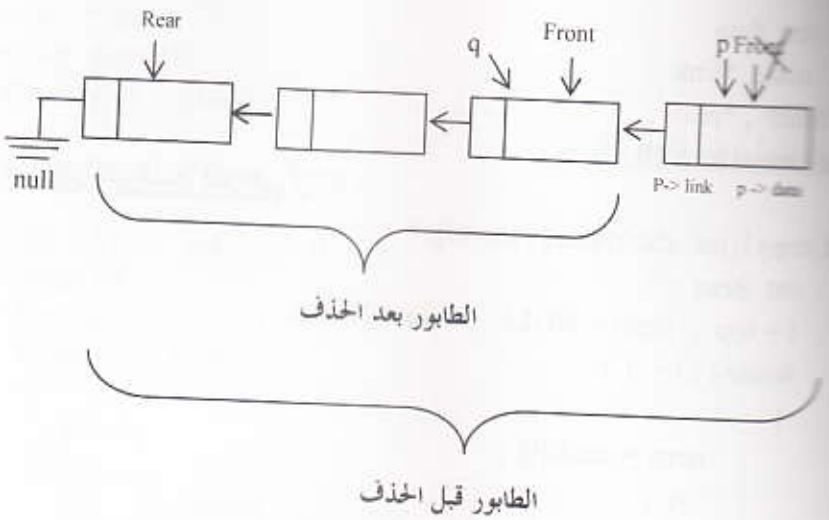
2 – في مقدمة عبارة (else) نستخدم مؤشر ثاني هو (q) يشير الى العنصر التاني في الطابور لكي نستطيع حذف العنصر الأول بعد خزن قيمته وقتيا"

في المتغير (item) .

3 ـ الخطوة الرابعة في عبارة (else) هي لتحديث قيمة المؤشر (front) ليشير الى موقع العنصر الثاني حيث يشير (q).

4 ـ الخطوتان الأخيرتان هي لمعالجة حذف اخر عنصر في الطابور ، مما يتطلب جعل قيمة كل من المؤشرين (front) ، (rear) هي (NULL) .

5 ـ الرسم التوضيحي التالي يبين كيفية تنفيذ الخطوات أعلاه :



تمرين : أكتب برنامج فرعي (function) لنسخ جميع عناصر المكدس المتسلسل (sequential stack) الى طابور موصول (linked queue) خالٍ من العناصر بحيث أعلى عنصر في المكدس يصبح اول عنصر في الطابور.

الحل : تستخدم التعريفات التاليه :

```
struct node
{
    int data
    node *link
} *front , *rear ;
const int size = 10 ;
```

وفيما يلي البرنامج الفرعي :

```
void copy1(int stack[size] , int top )
{    int item ;
     t = top ; front = NULL ; rear = NULL ;
     while ( t != -1 )
      {
          item = stack[t] ;
          --t ;
          p = new node ;
          p→data = item ;
          p→link = NULL ;
          if ( rear == NULL )
              front = p ;
          else
              rear→link = p ;
          rear = p ;
      }
      return *front , *rear ;
}
```

تمرين : اكتب برنامج فرعي لنسخ جميع عناصر الطابور الموصول (linked queue) الى مكدس متسلسل (sequential stack) خالٍ من العناصر بحيث اول عنصر في الطابور يصبح أعلى عنصر في المكدس

الحل : نستخدم التعريفات التالية

```
struct node
{
    int  data
    node *link
} *front , *rear ;
const int size=10 ;
int stack[size] , s[size] ;
```

وفيما يلي البرنامج الفرعي :

```
void  copy2( node *front , *rear ;)
{
      int  t = 0 ;   int  top = 0 ;
      node *f ;
      f = front ;
      while ( f != NULL )
      {
          p = f ;
          ++t ;
          s[t] = (p→data) ;
          f = (f→link) ;
      }
      while ( t != 0 )
      {
          ++top ;
          stack[top] = s[t] ;
          --t ;
      }
} Return top ;
```

- لاحظ هنا أستخدام المكدسين (s) ، (stack) . ماهو السبب ؟