بسم الله الرحمن الرحيم

الوحدة الأولى: مقدمة إلى تراكيب البيانات

***البيانات التي تنطوي على قيمة Simple Data Types :** هي البيانات التي تنطوي على قيمة بيانية واحدة في أدنى مستويات التعبير عن الأشياء والأحداث المحيطة بنا. وهي تضم القيم العددية الصحيحة والحقيقية والرموز على اختلاف أشكالها والقيم المنطقية وما شابه ذلك.

***أنماط بيانية مركبيه Structured Data Types :** هي التراكيب البيانية التي ينطوي بناؤها وتعريفها على أكثر من قيمة بيانية، ولها صورة تنظيمية معينة سواء من الناحية المنطقية أو الناحية الفيزيائية في ذاكرة الحاسوب. ومثال ذلك المصفوفات، والسجلات، والمجموعات، وسلاسل الرموز وغير ذلك م ما يتوفر في لغات الحاسوب أو م ما يوفره المبرمج نفسه.

***تراكيب البيانات Data_Structures:** هو الموضوع الذي يتناول الطرق والأساليب المختلفة التي يمكن من خلالها ترجمة التصور المنطقي ل لبيانات كما براها المبرمج إلى تمثيل مناسب في ذاكرة الحاسوب.

****ميزات التراكيب البيانية :**

١.يمكن التعامل مع كل عنصر منها على حدة ،كما لو كان متغيراً مستقلاً كبقية المتغيرات .

٢.طريقة تنظيم العناصر تؤثر مباشرة على طريقة الوصول إلى العنصر الواحد أول التعامل مع المركب البياني ككل .

****مهما اختلفت طرق التخزين أو تعددت فقد تخزن في هذه العناصر قيماً:**

- عددية: صحيحة (Integer) أو حقيقية Real.

- رمزية: Characterكالحروف وغيرها من الرموز.

-منطقية (Logical).

- تعدادية أو جماعية Enumerated.

– إشارية (Pointer Type)، كعناوين مطلقة لمواضع الذاكرة Absoluteأو عناوين نسبية(Relative).

1.الأعداد الصحيحة(Integers):

***يتم تخزين الأعداد الصحيحة في ذاكرة الحاسوب باستخدام إحدى طريقتين**

١.**التمثيل الثنائي (Binary Representation):** يخزن الأعداد الصحيحة في ذاكرة الحاسوب على شكل ثنائي وذلك لتسهيل العمليات الحسابية على هذه الأرقام. وبصفة عامة فإن العدد الصحيح يخزن في موضع واحد Word من مواضع الذاكرة ،إلا أن هناك بعض لغات الحاسوب مثل PL/I والأسمبلي التي تتيح للمبرمج إمكانية تخزين العدد الصحيح في جزء من الموضع ،والحد الأدنى في هذه الحالة هو بايت واحد ،وفي هذه الحالة يضطر الجهاز إلى نقل القيمة إلى مخزن مؤقت من أجل إعادته إلى وضعه الأصلي لتجري عليه العملية الحسابية والمنطقية المطلوبة.

***ومن أساليب التخزين المتبعة لهذه الغاية طريقة:**

أ. المكمل الثاني '2Complements': **ويحسب كما يلي:**

1- يحول الرقم العشري إلى النظام الثنائي.

2- يحسب المكمل الأول عن طريق تحويل كل 1 إلى صفر وكل صفر إلى 1.

3- نضيف 1 إلى المكمل الأول لنحصل على المكمل الثاني.

*مع العلم بأن الخانة الثنائية الأخيرة على اليسار تدل على إشارة الرقم: فالصفر يدل على أن الرقم موجب والواحد يدل على أن الرقم سالب.

ب. التمثيل العشري:(Decimal Representation): باستخدام الشيفرة العشرية المكونة من أربع خانات، والتي تعرف بالاسم(Binary Coded Decimal: BCD) ،ووفقاً لهذا النوع من التخزين فإن كل جزء من الرقم يمثل بأربع خانات ثنائية، وتمثل الإشارة بأربع خانات أخرى.

2.**الأعداد الحقيقية (Real):**هي تلك الأعداد التي تشمل على فاصلة عشرية ،و تستخدم لأغراض المسافات والأوزان...

*وتخزن معظم الحواسيب الأعداد الحقيقية بطريقة تسمح بإجراء العمليات الحسابية بسرعة عالية ولكن مقابل ذلك عدم الاحتفاظ بالدقة المطلقة .

*وتمثل بطريقة تدعى **طريقة النقطة العائمة(Floating-Point Representation):** هي أكثر أساليب التمثيل شيوعاً، الموجودة بين الأجهزة . ***يمثل الرقم كما يلي:**

1- يحول الرقم العشري إلى نظيره في النظام الثنائي.

2- يعبر عن الرقم الثنائي بالصيغة الأسية بتحريك الفاصلة إلى أقصي اليسار والضرب في الأس المناسب.

3- يمثل الأس والقاعدة في المكان المخصص لهما.

القاعدة Mantissa	الأس Exponent	إشارة الأس Sign E	إشارة القاعدة Sign M

***الفائدة الأساسية لهذا النوع من التمثيل** : أنه يتيح لنا أن نخزن في ذاكرة الحاسوب أعدادا ذات قيمة كبيرة مما لا يسهل تخزينه بالصورة الاعتيادية في موضع واحد من مواضع الذاكرة، وفي معظم الحواسيب يتم تخزين الأرقام الممثلة بهذه الصورة في موضع تخزيني واحد. ولكن في حالة التطبيقات العلمية فإن موضعا واحدا لا يكفي للحصول على درجة كافية من الدقة.

*ولهذا فإن معظم أجهزة الحواسيب لديها ما يسمي بالدقة المضاعفة(Double Precision)،وتستعمل لهذه الغاية موضعين لتخزين الجزء الكسري.

3.**الرموز (Characters) :**

نعني بالرموز الحروف والأعداد من صفر إلى تسعة والعلامات الخاصة ،إن لغة مثل PL/I لا تتضمن هذا النمط في تصميمها، واستعاضت عنه بسلاسل الرموز Strings. أما من حيث التخزين، فيتم تمثيل الرموز كسياق ثابت الطول من الخانات الثنائية Bits التي تستمد من الشيفرة المستخدمة مثل ASCII و EBCDIC.

٤. القيم المنطقية:(Logical):

تكون القيم المنطقية، أو البولينية Boolean كما نسميها في كثير من الأحوال إما: TRUE أو FALSE ،وأهم العمليات التي تجري عليها هي عملية الإسناد والعمليات البولينية الثلاثة المعروفة:NOT, OR, AND

***تخزينها:** إن أسلوب تمثيلها يعتمد على مترجم اللغة والجهاز المستعمل، **هناك طرقا ثلاث لتمثيلها :**

أ. استخدام خانة ثنائية واحدة Bit: وفي هذه الحالة يعبر عن القيمة الإيجابية

بالواحد1 = TRUEوالقيمة السلبية بالصفر 0 = FALSE

ب- استخدام الموضع التخزيني بكامله WORD: وفي هذه الحالة يتم تخزين القيمة صفر في الموضع بكامله للتعبير عن FALSE، وتعبئة الموضع بكامله بالقيمة واحد للتعبير عن TRUE.

ج- استخدام بايت واحد من الموضع Word: لأن الموضع Wordقد يحوي أكثر من بايت وفي هذه الحالة يتم تخزين القيمة المنطقية في أصغر وحدة معنونه.

٥. **مؤشرات الربط Pointers:**

يعد مؤشر الربط بحد ذاته نمط بيانيا بسيطا، ومهمته الإحالة إلى موضع آخر في ذاكرة الحاسوب، حيث تخزن إحدى القيم البيانية أو مجموعة منها. ومن هنا فإن محتوى مؤشر الربط هو عنوان ذلك الموضع. وقد يكون مؤشر الربط مستقلا بذاته وقد يكون جزء من مركب بيائي آخر، كأن يكون حقلاً في سجل.

*وفي أغلب الحواسيب يخزن المؤشر في موضع واحد من مواضع الذاكرة أو في نصف موضع. أما القيمة المخزنة في المؤشر فيمكن أن تكون **العنوان المطلق Absolute Address** للموضع الذي يشير إليه ، أو **العنوان النسبي Relative Address:** لهذا الموضع ضمن منطقة معينة من الذاكرة تم تخصيصها لهذا الغرض .

التمثيل المتصل Linked Representation :هو تخزين العناصر المختلفة لتركيب بياني معين في أماكن غير متجاورة في الذاكرة واستخدام مؤشرات الربط ل لوصل بين العناصر والإشارة إلى عناوينها. فكل عنصر يشير إلى ما يليه على ضوء التصور المنطقي الخاص بالتركيب البياني المعني. وقد يتم هذا التمثيل باستخدام المصفوفات المتوازية.

****التركيب المنطقي والفيزيائي للبيانات Logical and Physical Structure of Data :**

*نظرة المستخدم إلى البيانات تسمى **التصور المنطقي** :وهي هي نظرة نابعة من وظيفة هذه البيانات ،

أما نظرة الجهاز للبيانات تسمى **التصور الفيزيائي** :وهي نظرة نابعة من طبيعة تكوين الجهاز وطريقة تعامله مع البيانات من حيث التخزين.

****الفرق بين التصورين:** إذا أراد المبرمج أن يصل إلى محتويات أحد العناصر، فما عليه إلا أن يستخدم A[I][J] ولا يحتاج إلى معرفة التفاصيل الداخلية لعملية الوصول. أما بالنسبة للجهاز فتتم عملية الوصول بإجراء عملية حسابية خاصة نحدد من خلالها العنوان النسبي ل لعنصر المطلوب حسب التمثيل الافقي بالنظر إلى عنوان البداية Base، وهو عنوان العنصر الأول، كما يلي:

(A[I][J])= base + size (offset)

Offset = (I- LB) n + (J – LB)

حيث: العنوان المطلق ل لبداية :"base" عدد المواضع التي يحتلها العنصر الواحد. size

الحد الأدنى لكل بعد في المصفوفة 'LB' عدد الأعمدة."n"

وعلى هذا الأساس، فإن جهد المبرمج سينصرف إلى حل المشكلة نفسها بدلا من الاهتمام بالتفاصيل الداخلية والتركيب الفيزيائي ل لبيانات في داخل الذاكرة.

***مثال 4 تدريب 1 صفحة 20 *شكل 13 صفحة 23 .**

*** أسس نختار بموجبها من بين التراكيب البيانية المتوفرة لحل مشكلة معينة أو مجموعة من العوامل التي تؤثر على الاختيار:**

1.حجم البيانات المتصلة بالمشكلة.

2. مدى تكرار حدوث المشكلة والطريقة التي ستستعمل بها البيانات.

3. طبيعة البيانات نفسها، من حيث مدى ثباتها أو مدى التغيير الذي تتعرض له.

4. مساحة الذاكرة التي بتطلبها التركيب البياني لأغراض التخزين.

5. الوقت اللازم لاسترجاع أحد عناصر البيانات.

6. مدى سهولة أو صعوبة ترجمة التركيب البياني إلى جمل برمجية بإحدى لغات البرمجة.

***قرار المبرمج في اختيار أحد نماذج التراكيب البيانية يعتمد على عاملين أساسيين:**

أولاً : أن يعكس التركيب البياني العلاقات الحقيقية بين البيانات والأشياء التي تمثلها في عالم الواقع.

ثانياً: أن يكون التركيب البياني بسيطاً بدرجة كافية، وفي الوقت نفسه علي درجة مناسبة من الكفاءة لمعالجة البيانات وقت الحاجة.

****أهم التراكيب البيانية المستعملة بكثرة**

أ- المصفوفات Arrays:

هناك اختلاف بين لغات الحاسوب في تسلسل عناصر المصفوفة :تسير لغة c++ على طريقة الصفوف ،أما لغة الفورتران حسب التسلسل في الأعمدة.

ب- القوائم المتصلة Link Lists :

*إن قيمة المؤشر في العنصر الأخير هي NULL .

***الفرق الأساسي بين بناء القوائم المتصلة باستخدام مؤشرات الربط التي توفرها اللغة وبين طريقة المصفوفات المتوازية :**

١. في نوع المعلومات الإشارية المستخدمة في الدلالة على مكان العنصر التالي.

٢.في طريقة حجز الأماكن في الذاكرة ومدى التقارب بين هذه العناصر من الناحية الفيزيائيـة، فباستخدام مؤشرات الربط الخاصة باللغة لا تدعو الضرورة إلى وجود تقارب مكاني بين العناصر، بل يتم حجز الأماكن ل لعناصر المختلفة حسب توفرها.

****العمليات الأساسية التي تتم على التركيب البياني :**

١. الاستقصاء Searching :هي عملية البحث عن البيانات ،وقد تقتصر على عنصر واحد محدد أو عدداً من العناصر التي تستوفي شرطاً معيناً.

٢. الاستعراض Traversal : الوصول إلى العناصر وزيارة كل عنصر من العناصر التي تحتوي على قيم بيانية مرة واحدة ،مثل. عملية البحث عن أصغر وأكبر قيمة .

٣.الفرز Sorting :ان هذه العملية جزء من عملية الترتيب وليس العملية بأكملها ،تعطي قيمة مميزة لقيمة بيانية أو مجموعة من البيانات المنظمة على شكل سجل كأن تكون شيفرة خاصة أو رمز مميز .

٤.الدمج Merging :دمج مجموعتين من البيانات معاً لنكون تركيب بياني جديد له سمات الأصل ذاتها .تستخدم غالباً في الملفات.

٥.الإضافة والحذف : (Insertion and Deletion): يستلزم القيام بعملية الإضافة أن يتوفر مكان شاغر في ذاكرة الحاسوب لكي نتمكن من تخزين القيمة الجديدة، وإلا وصلنا إلى وضع نطلق عليه اسم الفائض Overflow. أما بالنسبة لعملية الحذف فتستلزم وجود قيمة بيانية واحدة على الأقل، والا سنصل إلى وضع نطلق عليه اسم الخالي Underflow، وهو عكس الفائض.

٦.التحديث :(Updating) : تحديث البيانات المخزنة بمعنى إزالة صفة القدم عنها من خلال إجراء التغيرات المناسبة عليها يستعمل مصطلح التغيير Change للتعبير عن هذه العملية.

7- عمليات أخرى: هناك بعض العمليات الأخرى التي تتم على بعض أنواع التراكيب البيانية وقد لا تتم على غيرها. ومثال ذلك تلك العمليات التي تجرى على سلاسل الرموز كالنسخ Copying، والاستبدال Replacement ،والمطابقة المنطقية (Pattern Matching)،إيجاد الطول (Length) ...تدريب (٢)(٣) صفحة 31

الوحدة الثانية: مبادئ تحليل الخوارزميات

****المقاييس التي نستخدمها في المقارنة بين الخوارزميات هي:**

1. مقدار وقت الحاسوب ال لازم لتنفيذها.
2. مساحة ذاكرة الحاسوب التي تحتاج إليها الخوارزمية.
3. وضوح الخوارزمية وبساطتها.

****إن وقت التنفيذ الحقيقي يعتمد على عدة عوامل:**

1. سرعة الحاسوب إذ أن الوقت الحقيقي يعتمد على سرعة الحاسوب المستخدم لتنفيذ البرنامج.
2. كمية البيئات، إذ أنه من الطبيعي أن يعتمد الوقت الحقيقي على كمية البيانات المراد معالجتها، فكلما ازدادت كمية البيانات ازداد وقت التنفيذ.
3. عوامل أخرى لها علاقة بطريقة تنفيذ الخوارزمية (تحويلها إلى برنامج في لغة برمجة معينة) كلغة البرمجة المختارة والمبرمج الذي كتب البرنامج ومقدار خبرته.

**** تحليل الخوارزميات رياضياً:**

* أننا نتعامل مع الخطوات وكأن كلا منها تحتاج إلى نفس المقدار من الوقت للتنفيذ وذلك لتسهيل المهمة

***طريقة ترميز O الكبيرة Big-O Notation :** طريقة رمزية تختص في استخلاص العوامل المهمة في التعبير عن كفاءة الخوارزمية.

*وقت التنفيذ الازم للخوارزمية ينمو بعلاقة خطية نسبة إلى حجم البيانات

*إن وقت تنفيذ الخوارزمية يعتمد على عدد خطوات الخوارزمية.

*إن طريقة الترميز Oالكبيرة تتجاهل الثوابت، ولذلك فإنها قد لا تكون طريقة دقيقة للمقارنة بين خوارزميات من الدرجة نفسها، ولكنها بالتأكيد طريقة حيدة للمقارنة بين خوارزميات من درجات مختلفة

* ولعدم اهتمامنا بالثوابت فإن نركز على الخطوة أو العملية الأكثر تكرارا في الخوارزمية عند تحديد الاقتران T(n) ونتجاهل بقية العمليات أو الخطوات. **وتسمى هذه العملية بالعملية الحرجة The Critical Operation**

*وغالباً ما يعتمد عدد تنفيذ خطوات خوارزمية معينة على طبيعة البيانات، لذلك

****عادة ما نحلل الخوارزمية آخذين ثلاثة سيناريوهات بعين الاعتبار:**

1- الحالة الأفضل Best case:وهي الحالة التي تحتاج إلى تنفيذ أقل عدد من الخطوات.

2- الحالة الأسوأ Worst case:وهي الحالة التي تحتاج فيها الخوارزمية إلى تنفيذ أكبر عدد من الخطوات.

3- والحالة الوسطى Average case :وهي الحالة التي تتطلب تنفيذ عدد معتدل من الخطوات.

*قد تنفذ عملية مقارنة واحدة في **الحالة الأفضل**، وذلك إذا كانت القيمة التي نبحث عنها مساوية للقيمة المخزنة في العنصر الأول وفي هذه الحالة تكون الخوارزميةO(1). وقد تحتاج الخوارزمية إلى n من عمليات المقارنة إذا كانت القيمة التي نبحث العنصر الأخير في المصفوفة أو لم تكن موجودة في المصفوفة، وفي هذه عنها هي الحالة تكون الخوارزمية O(n).

الحالة الوسطى هي العملية الأعقد، وذلك لكونها تعتمد على معلومات إحصائية عن احتمال وجود القيمة في الموقع الأول في المصفوفة أو في الموقع الثاني أو الثالثة أو الأخير. وإذا افترضنا توزيعا متكافئا Uniform Distributionللبيانات فإن احتمال وجود القيمة المبتغاة هو(=n1p/) وعليه تكون الحالة الوسطى أيضاO(n) وغالبا ما تركز جل اهتمامنا على تحليل الحالة الأفضل والحالة الأسوأ،ونتجاهل الحالة الوسطى لصعوبة تحليلها بطريقة دقيقة. **تدريب 1،2،3،4 صفحة48**

***مقاييس تنفيذ مميزة ومكررة :**

ترتيب هذه المقاييس حسب درجة قوتها من اليسار إلى اليمين:

$$O(1)<O(\log n)<O(n)<O(n \log n)<O(n^2)<O(n^3)<O(2^n)$$

١.الزمن الثابت (Constant Time) :

زمن الحوسبة الثابت يشار إليه بالعلاقة(O(1، أي أن الوقت المطلوب للعملية لا يتغير،مثل: جملة الكتابة أو القراءة.

٢.الزمن الخطي (Linear Time) :

إن الخوارزمية التي يشار إليها بالعلاقة (O(n هي خوارزمية ذات زمن يتناسب

طردياً مع حجم البيانات مثل: القائمة متصلة سيأخذ وقتا خطيا، لأن من المحتمل أن نزور جميع عناصر القائمة.

٣.الزمن اللوغاريتمي(Logarithmic Time) :

إن الخوارزمية ذات الكفاءة اللوغاريتمية، والتي لشير إليها بالعلاقة O(log n) تقوم بعمل أكثر من الخوارزمية ذات الوقت الثابت، وتحتاج بالتالي وقتاً أطول منها ولكنه أقل من الوقت الذي تستغرقه خوارزمية ذات وقت خطي، مثال : الوصول إلى أحد عناصر إحدى الشجيرات الثنائية الاستقصائية يستغرق وقتا لوغاريتميا.

**وهناك بعض الحالات التي تستفذ زمناً خطياً لوغاريتميا، وفي هذه الحالات العلاقة (O(n.nlog ويزيد معدل النمو بشكل يفوق الزمنين الخطي واللوغارتمي ومن بين الحالات التي ينطبق عليها هذا الوضع ترتيب مجموعتين من العناصر في مجموعة واحدة فيما يعرف باسم الفرز الدمجي أو المزجي (Merge Sort) .

٤.الزمن المتعدد الحدود (Polynomial Time) : إذا كانت العلاقة الزمنية محكومة بمتغير متعدد الحدود فإن نمو العلاقة يزداد بمعدل تربيعي(Quadratic)أو تكعيبي(Cubic)،O(n^2(n^c),O(n^c)O(3(. فالخوارزمية ذات الكفاءة التربيعية تقوم بأعمال أكثر م ما تقوم به خوارزمية ذات وقت خطي، ومن هنا تستنفذ وقتا أكثر، وبالمثل فإن الخوارزمية ذات الوقت التكعيبي تستغرق وقتا أكثر من الخوارزمية ذات الزمن التربيعي. مثال :تخزين قيمة ابتدائية في كل عنصر من عناصر مصفوفة ذات ثلاثة أبعاد، فإن ذلك سيستغرق زمنا تكعيبيا.

٥.الزمن الأسي(Exponential Time) : إن الخوارزمية التي تستنفذ مثل هذا الزمن يشار إليها بالعلاقة O(a^n)، وكلما زادت قيمة n ظهر فرق كبير بين الزمن الأسي والزمن الخاص بالحدود المتعددة وبصفة عامة فإن الخوارزميات ذات الزمن الأسي غير عملية، ولا يُنصح باستخدامها إلا في حالات خاصة عندما يكون حجم n صغيراً جداً، ذلك لأن كلفتها كبيرة ويصعب الوفاء بها. وكقاعدة عامة، ينبغي عدم اختيار خوارزمية يزيد معدل كفاءتها الزمنية عن الزمن التكعيبي.

مثال :التركيب الدوراني مثل for, while.

****خوارزميات الفرز :**

*هناك العديد من الخوارزميات لفرز قوائم من البيانات "ترتيبها تصاعديا أو تنازلياً".

١.**خوارزمية الفرز الفقاعي The Bubble Sort Algorithm:**خوارزمية بسيطة تقوم بمقارنة كل عنصرين متجاورين في المصفوفة، واستبدالهما إذا وجد أن العنصر الأول أصغر من الذي يليه. مثال صفحة:53.

2.**خوارزمية الفرز الانتقائي The Selection Sort Algorithm:**تعمل الخوارزمية على ترتيب عناصر المصفوفة بتكرار عمليتين إيجاد أقل عنصر في الجزء غير المرتب من المصفوفة واستبدال ذلك العنصر مع أول عنصر في الجزء غير المرتب. مثال :صفحة 53.

***تقوم الخوارزمية بالفرز الانتقائي بترتيب عناصر المصفوفة بتكرار عمليتين رئيسيتين هما:**

١. إيجاد أقل عنصر في الجزء غير المرتب من المصفوفة.

٢.استبدال ذلك العنصر مع أول عنصر في الجزء غير المرتب.

تدريب 5: لماذا يكفي تكرار الدوران الخارجي -1n من المرات فقط في خوارزمية الفرز الانتقائي ؟

لأنك لو وضعت -1N من العناصر في المكان الصحيح فإن العنصر الأخير سيكون حتماً في المكان الصحيح.

*للمقارنة الدقيقة بين الخوارزميات علينا أخذ عدد تكرار عملية التبديل بعين الاعتبار.

*ولكن الأمر أعقد بالنسبة لعملية التكرار في خوارزمية الفرز الفقاعي لأن عملية التبديل موجودة داخل دورانين متداخلين.

*في حالة كون المصفوفة مرتبة ترتيباً عكسياً تكون خوارزمية الفرز الفقاعي أسوأ بكثير من الخوارزمية الانتقائية من حيث عدد عمليات التبديل الواجب إجراؤها.

*وفي حالة كون المصفوفة مرتبة فإن خوارزمية الفرز الفقاعي لن تجري أي عملية تبديل بينما ستجري خوارزمية الفرز الانتقائية عملية تبديل حيث أن كل عنصر سيستبدل بنفسه

خوارزمية الفرز الإدخالي: The Insertion Sort Algorithm: تعمل على ترتيب العناصر (ابتداء من العنصر الثاني) بإدخال كل عنصر من المكان الصحيح نسبة إلى جميع العناصر التي سبق ترتيبها بإدخالها .

*عيوب خوارزمية الفرز الانتقائي والفرز الفقاعي أن كلا منها سوف تجري نفس العدد من المقارنات بغض النظر عن الحالة الابتدائية للمصفوفة حتى لو كانت المصفوفة مرتبة أصلا. أما خوارزمية الفرز الإدخالي فإنها تستطيع ملاحظة فيما إذا كانت الخوارزمية مرتبة أو شبه مرتبة أصلا وتنفذ في تلك الحالة عددا أقل من عمليات المقارنة.

*ان أفضل حالة لهذه الخوارزمية هي عندما تكون المصفوفة مرتبة اصلا وان عملية المقارنة هي العملية الحرجة في هذه الخوارزمية أيضاً.

*أما الحالة الأسواء عندما تكون المصفوفة مرتبة ترتيباً عكسي تنازلياً إذ تحتاج الخوارزمية عملية مقارنة واحدة لإدخال العنصر الثاني وإلى عمليتي مقارنة لإدخال العنصر الثالث وإلى ثلاث عمليات لإدخال العنصر الرابع وهكذا.

تدريب٦ :لماذا نستخدم في الخوارزميات الدورانية المتغير البوليني ؟

ذلك حتى تتوقف عملية المقارنة (الدوران الداخلي) عندما يصل العنصر إلى موقعه الصحيح في المصفوفة .

***مصطلحات مهمة :**

- تحليل الخوارزميات Algorithm Analysis :دراسة الخوارزمية والتعرف على مدى كفاءتها من حيث وقت التنفيذ اللازم وذاكرة الحاسوب اللازمة.

- خوارزميات الترتيب Sort Algorithm: مجموعة من الخوارزميات لترتيب قوائمLists من البيانات، حيث أنها تعمل على ترتيبها وفق معايير معينة وفي الغالب ترتيبها ترتيبا تصاعديا أو تنازليا.

– خوارزمية Algorithm: مجموعة من الخطوات الممنهجة تتسم بسمات محددة تؤدي إلى تحقيق هدف ما أو مجموعة أهداف محددة.

الوحدة الثالثة:التراكيب التجريدية

****التراكيب التجريديةAbstract Data Type:** هي تراكيب البيانات التي تستطيع استخدامها من خلال عمليات معينة معرفة على شكل دوال على هذه التراكيب، من دون الحاجة إلى معرفة تفاصيل تمثيل هذه التراكيب.

*وهذا ما ينبغي علينا عمله عندما نصمم وننشئ تركيبة بيانات جديدة، أي يجب علينا أن ننشئها كتركيبة بيانات تجريدية بحيث تستطيع ويستطيع آخرون استخدامها فيما بعد من خلال عمليات معرفة عليها دون الحاجة إلى معرفة تفاصيل تمثيلها وتنفيذها. ونهدف بذلك إلى إخفاء الكثير من المعلومات Information Hiding المتعلقة بكيفية تمثيل وتنفيذ تركيبة البيانات.

*وكل ما تحتاج إلى معرفته كي تستطيع استخدام تركيبة البيانات هذه هو العمليات Operationsالمعرفة عليها وكيفية استدعاء هذه العمليات. وعادة ما تنفذ هذه العمليات كدوال Functions، كل ما نحتاج إلى معرفته كي نستطيع استدعاء الدوال هو اسم الدالة ونوع ومعاملاتArguments هذه الدالة ونوع البيانات التي ترجعها الدالة.

***فوائد التراكيب التجريدية**

الفائدة الأولى :أن المستخدم هو قد يكون مبرمجاً آخر، يحاول استخدام تركيبة البيانات هذه في برامج تطبيقية كبير ومعقدة يستطيع أن يركز جهده وتفكيره علي المسألة التي يحاول كتابة برنامج لها بدون الحاجة إلى التفكير في الوقت نفسه في تفاصيل تمثيل تركيبة البيانات، ولذلك فهو يصبح أكثر إنتاجاً وفاعلية، وأيضا تصبح برامجه أقل تعقيدا وأيسر ل لكتابة والفهم والتعديل.

الفائدة الثانية: أنه في حالة تغيير طريقة تمثيل وتنفيذ تركيبة بيانات معينة فإن البرامج التطبيقية الكبرى التي تستخدم تركيبة البيانات هذه لن تتأثر أي أنها لن تحتاج إلى عمل تعديلات عليها طالما أن العمليات المعرفة أصلاً على تركيبة البيانات ما زالت معرفة، وأن طريقة استخدامها (استدعائها) لم تتغير.

الفائدة الثالثة: هي ما يسمى بالإستخدام المتكرر للبرامج Code Reuse بدون الحاجة إلى كتابتها مرة أخرى ،بل يزيد مقدار ثقتنا في برنامجنا الجديد، لأنه يستخدم دوال سبق استخدامها والتأكد منصحتها.

الفائدة الرابعة : هي تسهيل عمل فرق التصميم والبرمجة. إن استخدام تراكيب بيانات تجريدية موحدة في البرامج الفرعية المختلفة التي يكتبونها يساعد على جعلها متوافقة بحيث يستطيع برنامج فرعي معين استدعاء برنامج فرعي آخر، إذ أنهما يستخدمان تركيبة بيانات واحدة مُمثلة ومنفذة بطريقة واحدة.

خطوات لتصميم وتمثيل التراكيب التجريدية :

✓في المرحلة الأولى وهي مرحلة التعريف نعرف بدقة تركيبة البيانات الجديدة ونحدد مواصفاتها قبل أن نفكر بتفاصيل البناء (التمثيل) .

✓وفي المرحلة الثانية، وهي مرحلة التمثيل ، نحدد تفاصيل التمثيل المناسبة كاستخدام المصفوفات أو القوائم المتصلة وما إلى ذلك.

***ولتعريف تركيبة البيانات هنالك ثلاثة عوامل يجب أخذها بعين الاعتبار:**

1. أن نعطي وصفاً للعناصر المكونة لتركيبة البيانات.

2. أن نعطى وصفاً للعلاقة بين هذه العناصر.

3. نعطي وصفا للعمليات التي نرغب بتنفيذها على تركيبة البيانات، وهذا الوصف عادة ما يكون عبارة عن ترويسة الدوال Function Headerوالدوال المنفذة لهذه التعليمات بلغة + +C على سبيل المثال، لا نقصد هنا كتابة الدالة كاملة بل فقط ترويسة تلك الدالة ان هذه العوامل الثلاثة أسميناها بالتصور المنطقي لتركيبة البيانات، وهي نظرة نابعة من الوظيفة التي تؤديها هذه البيانات، ولا بد في نهاية المطاف من تمثيل هذا التصور المنطقي بطريقة مناسبة باستخدام لغة برمجة معينة لتحويلها إلى ما أسميناه بالتصور الفيزيائي لتركيبة البيانات وهي نظرة نابعة من طبيعة تكوين الجهاز وطريقة تعامله مع البيانات من حيث التخزين.

** إن تعريفنا لتركيبة البيانات أي وصفنا لتركيب المنطقي في هذه المرحلة يجب أن يحتوي كل المعلومات اللازمة لاستخدام هذه التركيبة من قبل اشخاص آخرين دون الحاجة إلى معرفة تفاصيل التمثيل، وبعد مرحلة التعريف تاتي

مرحلة التمثيل حيث يتقرر في هذه المرحلة كيفية تمثيل تركيبة البيانات باستخدام مصفوفة ذات بعد واحد على سبيل المثال أو باستخدام مصفوفة ذات بعدين أو باستخدام قائمة متصلة أو بطريقة أخرى مختلفة. والسبب في تأخير هذه المرحلة (مرحلة التمثيل) إلى ما بعد مرحلة التعريف هو كي تصبح في وضع أفضل لتقرير كيفية تمثيل البيانات بعد أن فهمنا مكوناتها وكيفية استخدامها من خلال العمليات المعرفة عليها.

إخفاء المعلومات التفصيلية InformationHiding: المقدرة على تصميم وإنشاء قائمة ما وعند استخدامها لا ضرورة لمعرفة كيفية وتفاصيل تمثيلها.

ترويسة الدالـة Function Heading:جزء من الدالة (أول سطر في الدالة)من أجـزاء الدالـة يخصص لوصف آلية التعامـل مـع هذه الدالة وبالتالي وصف للعمليات التي نرغب بتنفيذها على تركيبة البيانات.

**المجموعة باعتبارها تركيبة بيانات تجريدية :

***المجموعات Sets:** المجموعة هي عدد من الأشياء غير المكر رة المأخوذة من مدى Universe محدد من الأشياء. ولا يشترط أن تكون عناصر المجموعة مرتبة بأي ترتيب معين.

****العمليات التي تنفذ على المجموعات** :

.1انشاء المجموعات :SetCreate: حيث تنشا المجموعة كمجموعة خالية Empty Set ويجب استخدام هنه التعليمة قبل استخدام المجموعة لأول مرة، وتحتاج في هذه المرحلة)مرحلة التعريف(إلى كتابة ترويسة الدالة التي ستنفذ هذه العملية فقط. والترويسة هي: SetCreate(SetType& S);

.2.الانتماء الى المجموعة: IsElementof: حيث تقرر هذه العملية فيما إذا كان عنصر ما E منتميا إلي المجموعة Sأم لا ، أن العنصرE لا بد أن يكون من مدى Iniverse المجموعة، أي من نفس نوع الأشياء المسموح لها الإنتماء إلى المجموعة، لاحظ أيضاً أن نتيجة العملية والتي سننفذها كدالة يعيد لنا القيمة البولينية true إذا كان العنصر منتميا إلى المجموعة ويعيد لنا القيمة البولينية false إذا لم يكن منتميا إلى المجموعة أي أن الدالة هي دالة بولينية Functio Boolean وعليه تكون ترويسة الدالة كما بلي: bool IsElementOf(setType S, int E);

3. **تعيين المجموعات Set Assign :** تعيّن هذه العملية قيمة المجموعة S إلى مجموعة أخرى T أي أنها تجعلهما متساويتين void SetAssign(setType S, setType& T);

4.التأكد من أن المجموعة خالية Set Empty : وهي عملية نستخدمها إذا أردنا فحص مجموعة هل هي خالية أم لا، مرة أخرى دالة بولينية يعيد لنا القيمة Trueإذا كانت المجموعة خالية و False إذا لم تكن خالية، سيكون مناسبا لتنفيذ هذه العملية: bool SetEmpty(setType s);

5.التأكد من مساواة المجموعات : Set Equal

وتستخدم لفحص فيما إذا كانت المجموعتان S,T متساويتين bool SetEqual(setType S, setType T);

6. فحص المجموعة الجزئية SubSetOf: وتستخدم لفحص فيما إذا كانت المجموعة S هي مجموعة جزئية في المجموعة Tbool SubSetOf(setType S,setType T);

7. **اتحاد المجموعات Union:** وتستخدم لإيجاد المجموعة T وهي مساوية لاتحاد المجموعتين R وS

void Union(setType S,setType R, setType& T)؛

.8**تقاطع المجموعات :Intersection:** وتستخدم هذه العملية لإيجاد المجموعة T وهي مكونة من العناصر الموجودة في

كل من المجموعتين R وS.;void Intersection(setType S,setType R, setType& T)

9. **فرق المجموعات Difference:**وتستخدم هذه العملية لإيجاد المجموعة T المكونة من العناصر الموجودة في المجموعة R وليست موجودة في المجموعة S ;void Difference(setType S,setType R, setType& T)

10. **إضافة عنصر إلى المجموعة AddElement:** وتستخدم هذه العملية لإضافة العنصر U من المدى Univyerse إلى المجموعة S void AddElement(setType& S,int U);

.11**إزالة عنصر من المجموعة : RemoveElement:** وتستخدم لإزالة العنصر U من المجموعة S.

void RemoveElement(setType& S,int U);

.12**طباعة عناصر المجموعة : SetDisplay :**تقوم هذه الدالة بعرض العناصر المنتمية إلى المجموعة على شاشة العرض void SetDisplay(setType S);

*كل ما يحتاج أي مستخدم لعمله هو الحصول على نسخة من البرامج الفرعية المنفذة للعمليات واستخدامها بدون الحاجة إلى دراستها ومعرفة تفاصيلها. و سوف يكون هذا المستخدم بحاجة إلى تعريف مدى المجموعةUniverse (الأشياء المكونة للمجموعة) ، بالطبع فإن طبيعة البرنامج المستخدم للمجموعات هي التي تملي ماهية المدى Universe.

****تنفيذ المجموعات باعتبارها تركيبة بيانات تجريدية setType:**

حيث نبحث في كيفية تمثيل المجموعة وكيفية كتابة الدوال والدوال المنفذة للعمليات المعرفة أعلاه. ومن المهم جدا أن تتقيد هنا بترويسة الدوال التي حددناها في مرحلة التعريف، وذلك لأننا نريد أن يستطيع المرء استخدام تركيبة البيانات فقط من خلال معرفة تلك المعلومات التي حددناها في مرحلة التعريف فلا يجوز أن نكتب الدالة الخاصة بعملية اتحاد المجموعات مفترضين أنه يأخذ معاملا جديدا (لم يكن موجودا في ترويسة الدالة) كعدد العناصر في المجموعة على سبيل المثال.

1.**إنشاء المجموعة SetCreate:** كما اتفق في مرحلة التعريف ينشئ هذه الدالة مجموعة خالية، وعليه يجب أن يخزن القيمة false في كل موقع من مواقع المصفوفة.

```
void setType:: SetCreate(setType& S){
for(int i==firstValue;i<== lastValue;i++)
    S.setfi)-false;
        False ; }
```

.2**الانتماء الى المجموعة ISElementOf:**

```
bool setType::IsElementOf(setType S, int E){
        return S.set[E];   }
```

.3تعيين المجموعات : SetAssign:

```
void setType::SetAssign(setType S, setType& T){
    for(int i==firstValue;i<==lastValue;i++)
        T.set[i] =S.set[i]   ;  }
```

.4التأكد من أن المجموعة خالية SetEmpty:

```
bool setType::SetEmpty(setType S){
for(int i==firstValue;i<=lastValue;i++)
 if (S.set[i]==true)
        return false;  }
```

5. **التاكد من مساواة المجموعات SetEquual:**

تدريب 1: اكتب دالة بولينية لفحص فيما إذا كانت المجموعتان S,T متساويتين.

الحل:

```
bool setType::SetEqual(setType Ss, setType T)
{for(int i==firstValue;i<=last Value;i++)
    if (S.set[i]==T.set[i]) return false;   }
```

6.فحص المجموعة الجزئية: SubsetOr

```
bool setType::SubSetOf(setType S,setType T)
```

```
for(int i==firstValue;i<=lastValue;i++)
if (S.set[i] && !T.set[i]) return false;}
```

7.اتحاد المجموعات: Union

```
void setType::Union(setType S,setType R, setType& T){
for(int i=firstValue;i<=lastValue;i++)
T.set[i]=R.set[i] || S.set[i];}
```

8. تقاطع المجموعات :Intersection:

تتريب2 :اكتب دالة لإيجاد حاصل تقاطع مجموعتين، استخدم نفس ترويسة الدالة :

الحل:

```
void setType::Intersection(setType S,setType R, setType& T) {
for(int i=firstValue;i<=lastValue;i++)
T.set[i]=R.set[i]&& S.set[i];}
```

9.فرق المجموعات Differenee:

تدريب 3 :اكتب دالة لإيجاد حاصل طرح مجموعتين، إستخدم تفس ترويمسة الدالية

الحل:

```
void setType::Difference(setType S,setType R, setType& T){
for(int i=firstValue;i<=lastValue;i++)
T.set[i]=R.set[i]&& !S.set[i];}
```

10. إضافة عنصر إلى المجموعة AddElement:

تريب 4: اكتب دالة لإضافة عنصر جديد إلى مجموعة ما، إستخدم تفس ترويسة الدالة المحددة في مرحلة التعريف.

الحل:

```
void setType::AddElement(setType& S,int U){
S.set[U]=true; }
```

11. إزالة عنصر من المجموعة :

تدريب 5: اكتب دالة لحذف عنصر ما من مجموعة ما، إستخدم نفس ترويسة الدالة المحددة في مرحلة التعريف.

الحل:

```
void setType::RemoveElement(setType& S,int U){
S.set[U]=false; }
```

12. طباعة عناصر المجموعة SetDisplay :

```
void setType:SetDisplay(setType S){
for(int i=firstValue;i<=lastValue;i++)
cout<<S.set[i]<<" "; }
```

**ومن المهم أيضاً أن تلاحظ أهمية الالتزام، عند كتابة البرامج الفرعية المنفذة للعمليات المعرفة على setType، بترويسة تلك الدوال التي حددت في مرحلة التعريف بـ setType. وذلك حتى نستطيع استخدام هذه العمليات بدون الحاجة إلى دراسة البـرامج الفرعيـة المنفذة لتلك العمليات. ومن المهم أيضا الالتزام بهذا الأمر عند تغيير طريقة تمثيل وتنفيذ التراكيب التجريدية حتى تبقي طريقة استدعاء هذه العمليات كما كانت.

* **القائمة المرتبة**: هي مجموعة من الأشياء مرتبة بشكل خطي حسب علاقة ترتيب بين عناصرها.

***إن العمليات التي تستخدم القائمة من خلالها هي:**

١. **إنشاء القائمة (List create):** وتعد هذه العملية كدالة تقوم بإنشاء قائمة فارغة ؛(void Listcreate(slisi* list:

٢.**الإضافة (Insertion):** يمكن الإشارة إلى عملية الإضاقة بصفة مجردة **بالصيغة التالية:**

slist* insert(slist* list, int element)؛ وهذه الصيغة تعني إضافة العنصر element في القائمة list.

***أي أننا بحاجة إلى معلومتين حتى تقوم بهذه العملية**، وهما: العنصر الذي ترغب في إضافته ، واسم القائمة التي ترغب في إضافة العنصر إليها.

٣. **الحذف (Deletion):** يمكن الإشارة إليه بالصيغة المجردة التالية:(slist* deletee(slist* list, int element؛

أي حذف العنصر element من القائمة list. ***ومعنى ذلك أننا نحتاج إلى معلومتين حتى نقوم بهذه العملية، وهما:** العنصر الذي ترغب في حذفه، واسم القائمة التي سيتم الحذف منها.

٤. **الاستقصاء وتحديد الموقع (Searching and Finding Location):** * إن عملية الاستقصاء وتحديد موقع أحد العناصر ذات أهمية خاصة بالنسبة لعمليتي الإضافة والحذف، وهي ذات أهمية مماثلة بالنسبة لعملية الاسترجاع وعملية تحديث العناصر. والصيغة العامة الأساسية لعملية الاستقصاء **يمكن أن تتخذ الشكل التالي:**

slist* locateP(slist* list, slist* loc, int x)؛ اي حدد موقع العنصر x في القائمة list . *ومعنى ذلك أننا بحاجة إلى معلومتين أساسيتين حتى نقوم بهذه العملية، وهما: العنصر الذي ترغب في البحث عنه، واسم القائمة التي ترغب في إجراء عملية الاستقصاء فيها، ونتيجة هذه العملية هي قيمة تحدد موضع العنصر loc داخل القائمة أو الإشارة إلى أنه غير موجود.

*في حالة تكرار العنصر في أكثر من موضع، فإن النتيجة المعطاة تتصل بالموضع الذي ورد فيه العنصر للمرة الأولى.

٥. **تحديد العنصر السابق والعنصر اللاحق (Determine the Predecessor and the Successor):**

هي أيضاً ذات أهمية كبيرة في عمليات الحذف والإضافة بشكل خاص، ومن أهم هذه العمليات تحديد السابق واللاحق، وتحديد العنصر الأول وتحديد العنصر الأخير، ونشير هنا فقط إلى عملية تحديد السابق واللاحق. **ويمكن التعبير عنها بالصيغة التالية:**slist* find(slist* pred, slist* succ, int x, slist* list) ؛ أي حدد موقع pred وموقع succ بالنسبة للقيمة x في القائمة list .

وكما هو واضح، فإن العنصر الأول لا سابق له وبذلك تكون نتيجة البحث عن السابق الثابت الخاص NULL، *وقيمته تعتمد على طريقة تنفيذ القائمة فهو NULL إذا استخدمنا المؤشرات وهي صفر إذا استخدمت المصفوفات. وبالمثل فإن العنصر الأخير لا تالي له، وبذلك نخلص إلى نتيجة مماثلة وهي أن قيمة succ هي NULL.

٦.**الاسترجاع (Retrieval):** * الاسترجاع هو خطوة تالية لعملية الاستقصاء وتحديد الموقع، **ويمكن التعبير**

عنها مجردة كما يلي: slist* retrieve(slist* list, int p, int& element)؛ أي استرجاع العنصر الذي يحتل الموضع p في القائمة list.

***ومعنى ذلك أننا بحاجة إلى معلومتين القيام بهذه العملية**، وهما: موقع العنصر الذي نرغب في استرجاعه، وأسم القائمة التي نرغب في استرجاع العنصر منها، وعلى خلاف عمليتي الإضافة والحذف فإن عملية الاسترجاع لا تعدل في وضع القائمة التي تتم عليها عملية الاسترجاع. وإذا كانت قيمة الموضع صفرا أو NULL فإن عملية الاسترجاع ستتؤدي إلى الوصول إلى النتيجة نفسها (أي NULL).

٧. **تعديل قيمة عنصر (Update)** تغيير قيمة أحد العناصر أو استبداله بقيمة أخرى، **وهي تتخذ الصيغة التالية**

slist* replace(int x1, int x2, int p, slist* list)؛

أي بمعنى استبدال العنصر x1 الوارد في الموضع p في القائمة list بالعنصر الجديد أو القيمة الجديدة x2.

٨. **تحطيم القائمة وإزالة عناصرها (Destroy List):** عملية إزالة جميع العناصر وتحويل القائمة إلى قائمة خالية، ويمكن أن تتخذ الصيغة التالية(slist* makeNull(slist* list ؛

٩. طباعة القائمة (Display List): * عملية طباعة جميع عناصر القائمة، **وهي تتخذ الصيغة التالية:**

void printList(slist* list)؛

١٠. **استعراض قائمة (Traversal):** * وتعني المرور على كل عنصر فيها وتنفيذ خطوات معينة على كل منهم كطباعة

قيمهم أو تعديلها أو إحصائها وما إلى ذلك، **وتأخذ الصيغة**:(void traversal(slist* list ؛

***** تمثيل القوائم باستخدام القوائم المتصلة**

لقد أشرنا فيما سبق إلى أن من الممكن تمثيل القائمة على هيئة سلسلة من العناصر، كل منها يشير إلى الآخر بواسطة رابطة خاصة نطلق عليها، بصفة عامة، **اسم المؤشر (Pointer)**، وهو يحتوي على قيمة معينة تحدد موقع تخزين العنصر التالي في القائمة.

*وبناء على ذلك، أصبح كل عنصر يشير إلى الآخر، وشكلنا بذلك قائمة تتصل عناصرها معاً مكونة سياقة خطية منتظمة. ومن هنا، يصطلح عليها باسم القوائم المتصلة.

***إن كل عنصر من عناصر القائمة يتكون من جزئين :** الأول يمثل القيمة المعنية في القائمة ، والآخر يضم عنوان الموضع الذي يتبع هذه القيمة في القائمة وهو ما نسميه المؤشر.

*** قوائم مفردة** :نعني بذلك أبسط أنماط القوائم المتصلة وأكثرها شيوعا، وهي تتخذ شكلا خطية وتسير في اتجاه واحد وقيمة آخر مؤشر فيها هي NULL، **فإنها قد ترد بصيغتين أخريين**: **إحداهما تمتاز** بوجود عنصر إضافي خاص في بدايتها نشير إليه باسم المقدمة أو الرأس (Header) **والصيغة الأخرى تتميز** بوجود عنصر إضافي خاص في نهايتها بالإضافة إلى الرأس، نشير إليه باسم الذيل أو الذنب (Tailer).

***والهدف منهما** هو تخزين بعض المعلومات الخاصة عن القائمة كعدد العناصر، أو عدد مرات التعديل التي جرت على القائمة أو غير ذلك من المعلومات ، وقد يتركها بلا شيء.

***** تنفيذ عمليات القوائم باستخدام القوائم المتصلة المفردة :**

١.**إنشاء القائمة List Create :** كل ما تصنعه هذه الدالة هو إنشاء قائمة فارغة أي NULL.

void listCreate (slist* list{ list-NULL;}

***ولإنشاء قائمة غير فارغة (**تحتوي على عدد من العناصر) فإن الدالة Insert تستخدم العدد المطلوب من المرات كل مرة لإضافة عنصر جديد وذلك بعد استخدام الأمر listCreate لإنشاء قائمة فارغة.

٢. **استعراض القوائم المتصلة (List Traversal):** بحيث نقوم بزيار كل عنصر مرة واحدة، وإجراء أية معالجة مطلوبة على هذا العنصر. والخوارزمية التالية تحدد تفاصيل القيام بهذه العملية: الخوارزمية (1):

```
void slist::traversal(slist* list){

    slist* current;      current=list;

while(current!= NULL){     process(current-> info);

    current= current->link;  } }
```

مثال ٢،تدريب ١ صفحة ١٠١ .

وقد تحدث الإضافة في نهاية القائمة في حالتين، الأولى: هي الطوابير القائمة على التمثيل المتصل، **والثانية:** هي القوائم المرتبة وفق نظام معين بحيث يكون وضع العنصر الجديد في آخر التسلسل القائم للقيم .

الخوارزمية (2)(3) مثال (3) صفحة ١٠٤. تدريب (2) صفحة ١٠٩ .

٢.حذف العناصر من القوائم المتصلة (Deletion): نحذف العناصر التي لم نعد بحاجة إليها، وتؤدي إلى إجراء بعض التعديل على سلسلة المؤشرات القائمة بين العناصر. ***يقتضي هنا أن نحدد أولا مكان وجود هذا العنصر** داخل القائمة ومن ثم نقوم بفك الارتباط بين هذا العنصر وما يليه وما قبله، وإعادة سد الثغرة الحاصلة بواسطة ربط السابق باللاحق.

*** فإذا كان العنصر المرغوب حذفه في بداية القائمة** فإن ذلك يقتضي تعديل المؤشر الخارجي (list) لكي يشير إلى العنصر الثاني في القائمة ثم فك ارتباط العنصر الأول، وبذلك فإن العملية لا تنطوي على مجهود كبير.

أما في حالة وجود العنصر المرغوب حذفه في نهاية القائمة، فإن ذلك يستدعي فك ارتباط العنصر السابق بالعنصر الأخير، وذلك بتغيير قيمة المؤشر الخاص بالعنصر المشار إليه بواسطة العنصر الأخير إلى NULL .

***الخوارزمية (4) (5) مثال ٤ صفحة ١١٢ تدريب ٣ صفحة ١١٦. ***

***المصفوفة الأحادية:** مجموعة متجانسة من العناصر تتخذ شكلا خطية، وتخزن في مواضع متجاورة في ذاكرة الحاسوب بناء على ذلك فإن العنصر **i**في المصفوفة ياتي مسبوقا بالعنصر (i-1) ويأتي متبوعا بالعنصر (i+1).

*ان الغالبية العظمى من اللغات الراقيه توفر نمطاً بيانية متميزاً للمصفوفات، بما في ذلك المصفوفات الأحادية. ولكن ينبغي أن نشير إلى أن هذه اللغات (باستثناء لغة الجول -60) تسير على نهج الحجز الثابت (Fixed Allocation) للمصفوفة.

*وعلى هذا الأساس ينبغي أن نميز بين المصفوفة من الناحية الفيزيائية المتصلة بالذاكرة، ومجموعة القيم التي تحتويها والتي تشكل ما نشير إليه بمصطلح القائمة الأحادية. فقد يكون عدد عناصر القائمة مساوية لعدد الأماكن المحجوزة، وقد يكون أقل من ذلك.

*****طريقتين لتمثيل هذا النوع من القوائم المنفردة : الأولى** هي استخدام المؤشرات التي توفرها لغة البرمجة، **والثانية** هي استخدام الدالات النسبية في المصفوفات المتوازية أو المصفوفات المركبة. وهناك طريقة رئيسة ثالثة مختلفة لا تستخدم المؤشرات من أي نوع. وبدلا عن ذلك فإنها تخزن مجموعة القيم التي تضمها القائمة في مواضع متجاورة ضمن منطقة تم حجزها خصيصا لهذه الغاية.

*وكل عنصر يعرف بعنوانه النسبي (أي نسبة إلى بداية المصفوفة) الذي تتم ترجمته من جانب نظام لغة البرمجة المستعملة إلى عنوان مطلق (Absolute Address) حين البحث عنه في الذاكرة. وتقوم هذه الطريقة على وجود مصفوفة أحادية بعدد محدد من العناصر، ووجود مؤشر إلى العنصر الأخير في القائمة .

مثال مناهج تعريف المؤشرات صفحة ١١٨-١٢٠.

**** مجموعة من المسائل ينبغي أخذها بالحسبان عند اختيار الطريقة الأنسب في تمثيل القوائم**(استخدام المؤشرات ،أم المصفوفات المتوازية أو المركبة ،أم المصفوفات الأحادية) :

١. أن التمثيل بالمصفوفات، على اختلاف تطبيقاتها، يتطلب منا أن نقرر سلفا ، منذ وقت البرمجة الحد الأعلى لعدد العناصر. فإذا لم يكن باستطاعتنا تحديد هذا العدد فإن من الأفضل أن تختار طريقة التمثيل بالمؤشرات إذا كانت اللغة توفر مثل هذه الإمكانية.

٢. تتفاوت الأساليب الثلاثة فيما بينها من حيث الوقت الذي تستغرقه بعض العمليات. فبينما تتطلب عمليتا الإضافة والحذف، في القائمة المتصلة(على اختلاف أساليب تمثيلها) عدداً ثابتاً من الخطوات، فإنها تستغرق وقتا يتناسب مع عدد القيم التي ينبغي إزاحتها في حالة المصفوفات الأحادية، وفي المقابل فإن عملية تحديد العنصر السابق والعنصر الأخير في المصفوفات الأحادية تستغرق وقتا ثابتا، بينما تستغرق العملية وقتا يتناسب مع عدد العناصر في حالة القائمة المتصلة.

٣. أن القوائم الخطية المفردة لا تتمتع بقدر مناسب من المرونة. فلو تم تحديد موضع معين للإضافة أو الحذف بواسطة مؤشر خارجي فإن هذا المؤشر لن يصلح لغير الحالة التي أسند لها. لأن هذا المؤشر، بحكم طبيعة تكوينه، يستطيع أن يتحرك باتجاه واحد فقط. والأسلوب الوحيد لتجاوز هذه المشكلة هو استخدام صيغة القائمة

الثنائية، وبذلك يمكن توفير قدر مناسب من المرونة وإن كان لا يصل لدرجة المرونة نفسها التي تتمتع بها عملية الإشارة إلى العناصر في حالة المصفوفات الخطية.

٤. إن أسلوب التمثيل بالمصفوفات على اختلاف أنواعها، أحادية أو متوازية أو مركبة، تؤدي إلى إهدار في المساحة، وذلك لأن عدد عناصر المصفوفة مستقل عن عدد القيم. أما في حالة التمثيل باستخدام المؤشرات، فإن الحجز في الذاكرة يتم بناء على الطلب، لكن الثمن الذي ندفعه مقابل ذلك هو جزء إضافي من الذاكرة للاحتفاظ بالمؤشرات. وبذلك يمكن القول، بصفة عامة، بان اسلوب المؤشرات أكثر الاستثمار للذاكرة. ولكن ينبغي التنبيه إلى أن هناك بعض الظروف التي يمكن أن تكون فيها المصفوفات أكثر استثماراً للذاكرة، خاصة في حالة الأعداد الثابتة القيم .

أ. تنفيذ عمليات (OPERATIONS) القوائم باستخدام المصفوفات:

*ان جميع العمليات التي أشرنا إليها يمكن أن تتم على القوائم التي تمثل على شكل مصفوفات أحادية ممكن أن تقوم بعمليات الاستعراض، والاستقصاء، والإضافة، والحذف، والفرز وبعض العمليات الأخرى. ونكتفي بشكل خاص بعمليات الاستعراض، والإضافة، والحذف.

***إنشاء القائمة Listcreate:** في حالة استخدام المصفوفات لتمثيل القائمة فإن كل ما تعمله هذه الحالة هو إعطاء المؤشر last القيمة صفر void MtList::Listcreate(MtList& L)

{؛L.last= 0}

***الاستعراض (array list traversal):** لا بد من إسناد مجموعة من القيم إلى هذه المصفوفة باستخدام أحد التراكيب الدورانية المعروفة في لغات البرمجة مثل: (while) أو (do ,while) او (for). ولا يعني ذلك، بالضرورة أن نملأ جميع خلايا المصفوفة بالقيم. فعدد القيم الفعلي يمكن أن يزيد أو يقل عن العدد الكلي للخلايا المحجوزة، وبذلك فإن العمليات التي تتم على المصفوفات ينبغي أن تكون محدودة بعدد القيم المخزنة في المصفوفة، والتي تشير إليها من خلال متغير خاص.

*ومن هنا فإننا عندما نقوم بعملية الاستعراض لا نتعامل مع الحد الأعلى للمصفوفة، بل مع المتغير الذي يشير إلى آخر قيمة مخزنة في المصفوفة (**وهو last في الخوارزمية 6 صفحة ١٢٣). مثال ٥ ،تدريب ٤ صفحة ١٢٤ .**

ب. إضافة قيم جديدة إلى المصفوفة :

تمتاز المصفوفة بأن المكان محجوز منذ البداية، وبذلك فإن عملية الإضافة تقتصر فقط على إدخال القيمة في موضع محدد ضمن الحيز المتوفر في المصفوفة لأغراض الإضافة، وعلى خلاف القوائم المتصلة، فإننا قد نصل في حالة المصفوفات الأحادية إلى وضع لا نستطيع معه إضافة قيم جديدة إذا لم يتوفر مكان شاغر. وهذا الوضع هو ما نسميه الفائض (Overflow).

****عملية الإضافة تنطوي على ثلاث خطوات رئيسة هي:**

أولا : التأكد من وجود مكان شاغر في المصفوفة وتحديد مكان إضافة القيمة الجديدة ضمن مجموعة القيم المخزنة في المصفوفة.

ثانيا: إزاحة القيم التالية لمكان الإضافة إلى مواضع جديدة من أجل إخلاء موقع الإضافة في حالة وجود مثل هذه القيم.

ثالثا: إدخال القيمة الجديدة في الموضع الذي تم إخلاؤه. **خوارزمية ٧،٨،٩ تدريب ٥ صفحة١٢٧.**

** أن عدد القيم التي تحتاج إلى إزاحة باتجاه نهاية المصفوفة يعتمد على الموقع الذي ستحتله القيمة الجديدة. فكلما اقتربنا من بداية المصفوفة زاد عدد العناصر التي تحتاج إلى إزاحة. وعلى العكس من ذلك، كلما اقترب

موضع الإضافة من نهاية المصفوفة، قل عدد القيم التي تحتاج إلى إزاحة. ومعنى ذلك أن عدد القيم المزاحة يتراوح بين صفر (وهي حالة الإضافة إلى نهاية القائمة) والعدد الكلي للقيم المخزنة (n). وبصفة عامة، فإن متوسط عدد القيم المزاحة هو حوالي نصف عدد القيم تقريبا. ويمكن أن يحتسب ذلك على أساس العلاقة التالية :

$$Mi= \sum(n-k+1).Pk \quad k=1 \rightarrow n$$

حيث: Mi: يشير إلى المتوسط. K: موضع الإضافة :n:عدد القيم في القائمة.

P: احتمالية إضافة قيمة إلى الموضع K وتساوي (n/1)

$$Mi= \sum(n-1).Pk = (n(n+1)/n) -1/n.(n(n+1) /2)$$

$$=n+1 - (n + 1)/2$$

$$=(n + 1)/2$$

ج. حذف القيم (Deleting Values) من المصفوفة الأحادية: إن عملية الحذف تنطوي على إلغاء القيمة وليس الموضع. فالحجز للمصفوفة يبقى طيلة مدة تنفيذ البرنامج الذي عرفت فيه، رئيسا كان أو فرعيا. وهذا على خلاف القوائم المتصلة، التي تنطوي عملية الحذف فيها على إلغاء للموضع والقيمة على السواء.

***ان عملية الحذف تنطوي على خطوتين أساسيتين هما:**

أولا: التأكد من وجود عناصر في المصفوفة وتحديد مكان القيمة المراد حذفها.

ثانيا: إزاحة القيم، إذا اقتضى الأمر ذلك، إلى بداية القائمة لسد الثغرة الناتجة عن الحذف.

ويمكن التعبير عن هاتين الخطوتين بالخوارزميات الثلاثة التالية (10,11,12)

****ان تحديد موقع القيمة المراد حذفها يعتمد على ثلاثة أوضاع مختلفة :**

١.أن تكون المصفوفة خالية ،وبالتالي فإن القيمة التي نبحث عنها غير موجودة .

٢.أن تكون القيمة غير موجودة ضمن قائمة القيم المخزنة في المصفوفة .

٣.أن تكون القيمة موجودة في مكان ما من القائمة ،وقد يكون هذا المكان بدايتها أو نهايتها أو خلالها ،**ولتوضيح هذه الحالات مثال ٧ صفحة ١٣١. تدريب ٦ .**

*تحتاج إلى إزاحة باتجاه بداية المصفوفة يعتمد علي الموقع الذي تحتله القيمة المراد حذفها، وكما هو الشأن بالنسبة لعملية الإضافة، فإن عدد القيم المزاحة يزيد كلما اقتربنا من البداية. وهذا العدد يتراوح بين الصفر والقيمة (n-1)حيث الرمز n يشير إلى العدد الكلي للقيم، وفي المتوسط فإن عدد القيم المزاحة يصل إلى حوالي النصف. ويمكن احتساب ذلك على أساس المعادلة التالية:

$Md=\sum (n-k).Pk$ حيث أن: Ma يشير إلى عدد القيم المزاحة. K يشير إلى موضع الحذف.

ىn عدد القيم في القائمة. P احتمالية حذف قيمة من الموضع k وتساوي1/n.

$$Md=sum(n-k).pk = n- (n(n+ 1) /2n)$$

$$=n- ((n+ 1) /2)$$

$$=n - 1/2$$

محاكاة القوائم المتصلة باستخدام المصفوفات: إن هذا الأسلوب الموضح أعلاه في تمثيل القوائم المتصلة هو ما نصطلح عليه باسم التمثيل المتصل باستخدام المؤشرات (Pointer Implementation). وهو يصلح بصفة خاصة في اللغات التي توفر للمبرمج إمكانات برمجية تسمح له بتعريف المؤشرات والإفادة منها في الربط بين العناصر المتتابعة.

وبالإضافة إلى ذلك، **هناك طريقة أخرى لتنفيذ القوائم المتصلة تقوم على استخدام المصفوفات المتوازية وفكرة الدالة النسبية (Cursor Based Implementation) أي نسبة إلى بداية المصفوفة. وهذه الطريقة تصلح بصفة خاصة في اللغات التي لا توفر نمطا إشاريا متميزا مثل فورتران والجول. كما يمكن تطبيقها في اللغات التي توفر مثل هذا النمط، إلا أن استخدام أسلوب المؤشرات يمتاز عليه في كثير من الجوانب.**مثال٨ وتدريب٧ صفحة ١٣٤ لتوضيح هذه الأساليب.**

وبالإضافة إلى الأسلوبين السابقين لتمثيل القوائم، فإن هناك **أيضا أسلوباً ثالثاً** يقوم على استخدام المصفوفات المركبة من سجلات (Composite Arrays). أي أن كل عنصر في المصفوفة هو سجل مكون من عدد من الحقول، وأحد هذه الحقول يستغل لتخزين قيمة الدالة النسبية إلى العنصر التالي. مثال ٩وتدريب٨ لتوضيح هذا الأسلوب.

****ان مسألة الاختيار بين الأساليب الثلاثة السابقة يتوقف على عاملين أساسيين هما:**

أ. مدى توفر إمكانية استخدام هذه الأساليب الثلاثة في لغة البرمجة المستعملة. فقد لاحظنا أن لغة الفورتران، مثلا، ليس لديها إمكانات التعريف المؤشرات ولا لتعريف السجلات. وبذلك فإن الوسيلة الوحيدة المتوفرة هي استخدام المصفوفات المتوازية. أما في لغة C++، فإننا نجد جميع الإمكانات للتعامل مع الأساليب الثلاثة المذكورة.

ب- مدى النمو المتوقع الذي تتسم به القائمة. فإذا تركنا مسألة السهولة في التعامل مع القائمة من حيث الإضافة والحذف جانباً، فإن الإحساس بوجود درجة عالية من التغير على القائمة يوجب اختيار أسلوب يتسم بالديناميكية، وهو ما يتوفر في التمثيل القائم على المؤشرات (في حالة وجود مثل هذه الإمكانية في اللغة).

****المصفوفات الثنائية والمتعددة الأبعاد:**

إن لغات الحاسوب توفر ضمن إمكاناتها البرمجية ما يمكن المبرمج من التعامل مع الجداول على اختلاف أشكالها و ابعادها، مع بعض التفاوت فيما بينهما من حيث عدد الأبعاد التي تسمح بها، فإن هذا النوع من التراكيب البيانية (المصفوفات متعددة الأبعاد)يتيح للمبرمج سهولة التعامل مع هذه البيانات سواء في التخزين أو الوصول والمعالجة.

*ولكن المصفوفات المتعددة الأبعاد ، على خلاف المصفوفات الأحادية، لا تستلزم القيام بالعمليات ذاتها التي أشرنا إليها في الجزء السابق كعمليات العذف والإضافة والفرز، مما يجعل التعامل معها مقصورا على نطاق محدود من التطبيقات، إلا إذا نظر إلى المصفوفة الواحدة على أنها مجموعة مركبة من القوائم والمصفوفات الأحادية، وفي هذه الحالة، فإن العمليات التي سبق أن أشرنا إليها تصبح ضرورية .

****ومن بين المسائل الأساسية التي تبرز لدى التعامل مع المصفوفات الثنائية والمتعددة،** مسألة تحديد طريقة تمثيلها في الذاكرة وبالتالي تحديد العناوين المطلقة للعناصر، إلا أنه ينبغي التنبيه إلى أن هذه ليست مهمة المبرمج . لغات البرمجة هي التي تعالج ذلك. **وتشير هذه اللغات وفقاً لأحد أسلوبين: الأول :** هو التمثيل الطولي (Column-Major Ordering). **الثاني :** هو التمثيل الأفقي (Row- Major Ordering)

ففي التمثيل الأفقي يتم تخزين القيم الواردة في الصف الأول متلوة بالقيم في الصف الثاني... وهكذا حتى الصف الأخير في الجدول . **أما في التمثيل الطولي،** فإن التخزين يسير وفق الأعمدة إذ يتم أولا تخزين القيم الواردة في العمود الأول متلوة بالقيم الواردة في العمود الثاني،... وهكذا حتى آخر عمود في الجدول .

***وايضاً فإن ففكرة التمثيل الأفقي أو الطولي تنطبق على المصفوفات ذات الأبعاد المتعددة(عدة صفحات)،** حيث يتم كالتالي **في التمثيل العامودي** : القيم الواردة في العمود الأول، ثم القيم الواردة في العمود الثاني، فالثالث...، وهكذا حتى انتهينا من الصفحة الأولى، ثم بعد ذلك قيم العمود الأول في الصفحة الثانية، فالعامود الثاني ... وهكذا حتى انتها قيم الصفحة الثانية ، ثم تتلو ذلك قيم الصفحة الثالثة، عموداً بعد عمود . **أما في حالة التمثيل الأفقي،** فإن القيم تتخزن في ذاكرة الحاسوب بتخزين القيمة الأولى في الصف الأول من الصفحة الأولى، وتلتها القيمة الأولى في الصف الأول الصفحة الثانية، تلتها القيمة الأولى في الصف الأول من الصفحة الثالثة. ثم انتقلنا إلى القيمة الثانية في الصف الأول من الصفحة الأولى، وتلتها القيمة الثانية من الصفر الثانية، وتلتها القيمة الثانية في الصف الأول من الصفحة الثالثة... وهكذا حتى انتهت قيم الصف الأول في الصفحات الثلاثة. وانتقلنا بعدها إلى الصف الثاني وقمنا بتخزين قيمه بالأسلوب نفسه، وهكذا حتى يتم تخزين جميع القيم.

س:كيف يمكن تحديد العنوان المطلق لعنصر معين داخل الذاكرة ؟

أننا بحاجة إلى إجراء بعض العمليات الحسابية للوصول إلى ذلك. والمعلومات التي نحتاجها لهذه الغاية هي:

أ. العنوان المطلق للعنصر الأول في الذاكرة وهو ما نسميه الأساس (Base).

ب، عدد المواضع التخزينية التي يحتلها العنصر الواحد.

ج. البعد الطولي للعنصر المعني بالنسبة إلى الأساس. ويمكن التعبير عن ذلك بالمعادلة التالية:

a(A[K1 ,K2 ,...Kn])= base+size(offset)

**** وينعكس الفرق بين التمثيل الأفقي والتمثيل الطولي في طريقة حساب الجزء الأخير من المعادلة، وهو البعد الطولي (offset) ، والذي يتم على النحو التالي:**

أولا: التمثيل الطولي (column-major):((..En L(n-1))+ E(n-1)L(n-2) ...+E2)L1 +E1)..)) = offset

ثانيا: التمثيل الأفقي (row-wajor): offser = (...((E1 L2 + E1)L3 +E3)L4 ...E(n-1))Ln +En

وتحسب كل من E و L على النحو التالي:Ei = Ki - LB Li=UBi - LBi +1

علما بأن: K: تشير إلى دالة نسبية (subscrip1) :ومثال ذلك قولنا A [2][3] أي بمعنى (A[K1][K2])

LB: القيمة الدنيا لكشاف المصفوفة (lower-bound): فإذا قلنا ؛array [10] فإن القيمة الدنيا في هذه الحالة هي "0".

UB: القيمة القصوى لكشاف المصفوفة (upper- bound) : وبذلك فإن القيمة القصوى في ؛array [10] هي "9" .

الوحدة الخامسة: الطوابير Queue:

***الطابور** :تركيب خطي يسير على مبدأ إضافة العناصر الجديدة إلى نهاية القائمة وحذف العناصر من بدايتها.

* يشار للطابور أحيانا باسم الفيفو (FIFO) وذلك اختصارا للتعبير الإنجليزي (First In First Out) من يأتي أولاً يخدم أولاً .

نظرية الطوابير (Queueing Theory) هو دراسة السلوك المتعلق بحركة الطوابير .

**ولكن ينبغي أن نلاحظ بأن هذا التركيب البياني(الطوابير) وما يبنى عليه من عمليات وخوارزميات لا يهتم بالمسائل والتعقيدات التي تنطوي عليها حركة البيانات بداخله على النحو الذي نجده في نظرية الطوابير. وجل اهتمامنا هنا ينصب على كيفية تخزين البيانات في هذا التركيب البياني وكيفية الوصول إليها.

-المقدمة (Front): مؤشر خاص يشير إلى العنصر الأول في الطابور، وهو يحدد اتجاه الحذف من الطابور.

- المؤخرة (Rear): مؤشر خاص يشير إلى العنصر الأخير في الطابور، وهو يحدد اتجاه الإضافة إلى الطابور.

**الطابور ليس ثابت في الطول وفي طبيعة القيم المكونة له .فإن الطابور يمتاز بالمرونة الحيوية وآلية هذه المرونة تقوم على الحذف والإضافة .

****السمات الأساسية التي يتسم بها الطابور وتجعله قادرا على محاكاة الواقع الفعلي الذييحاول ترجمته والتعبير عنه:**

أ - يتم تزويد الطابور، عند البدء بينائه، بالوسيلة المناسبة للإشارة إلى مقدمته والى مؤخرته لتسهيل عمليات الإضافة والحذف.

ب- عندما يتساوى عدد مرات الحذف مع عدد القيم المضافة إلى الطابور، فإن الطابور يصل إلى وضع يصبح معه خاليا من القيم

ج- أن القيمة الوحيدة التي يسمح بالوصول إليها ومعالجتها في الطابور هي القيمة الأولى ولكي يتم الوصول إلى العناصر الأخرى ومعالجتها، فلا بد من استرجاع قيمة العنصر الأول ومعالجته قبل كل شيء.

د- أن حجم الطابور متغير وذلك اعتمادا على ما يضاف إليه وما يحذف منه، ومن هنا فإننا نصفه بالديناميكية.

ه- يمكن وصف عملية الوصول إلى العناصر في الطابور بأنها مباشرة. وبذلك فإنها لا تستغرق وقتا كبيرا. بيد أن هذا الوصول محدود بمقدمة الطابور ومؤخرته. وهذا بالطبع على خلاف الاسترجاع والمعالجة، فهما يتحددان بالبداية وليس بالنهاية.

****وتبرز أهمية الطوابير بشكل خاص في** عمل نظم التشغيل المستخدمة في أجهزة الحاسوب التي تعمل وفق مبدأ اقتسام الوقت (Time-Sharing Systems)وتقدم خدماتها لأكثر من مستفيد، إذ لا بد في هذا الوضع من تنظيم عملية استخدام الموارد المتوفرة والتي تتمثل بوحدة الحساب والمنطق والذاكرة وأجهزة الطباعة والذاكرة المساعدة ، وبذلك فإن من الممكن أن يكون هناك عدد من الطوابير بعدد الموار المطلوبة

هناك أكثر من اسلوب يمكن أن يتبعه نظام التشغيل لتنظيم عمل الطابور . وأحد هذه الأساليب هو حجز جزء مناسب من الذاكرة لتخزين الطابور نفسه. ويتضمن هذا الطابور معلومات أساسية مثل: رقم العمل، ومكان تخزينه في الذاكرة المساعدة، وكمية المعلومات التي يتضمنها العمل الواحد (أي عدد الأسطر التي ستطبع). ويبدأ نظام التشغيل بوضع الأعمال تباعا في الطابور ، وكلما فرغ جهاز الطباعة من عمل معين، فإنه ينتقل إلى العمل التالي له في الطابور . ولكي يتجنب نظام التشغيل المشكلات الناجمة عن الحاجة إلى إزاحة العناصر الموجودة في الطابور باتجاه بداية المادة المخصصة له في الذاكرة من أجل إفساح المجال لإضافة العناصر الجديدة إلى الطابور فإنه يمكن أن يتعامل مع المواضع المحجوزة في الذاكرة على أنها تشكل طابوراً دائرياً.

إن من المشاكل الأساسية التي تبرز: هي تقدير حجم الطابور فكما ذكرنا قد يصل الطابور الى الفائض (Overflow) الأمر الذي يؤدي إلى توقف مؤقت لعمل وحدة المعالجة المركزية . ولتجنب مثل هذا الوضع أو

التقليل من حالات وقوعه فإن من الممكن القيام بتقدير الحجم المتوقع للطابور وبالتالي حجز مساحة في الذاكرة تتناسب مع درجة نمو الطابور. ويدخل في هذا التقدير ثلاثة عوامل رئيسة هي: معدل الإضافة، ومعدل الحذف، والزمن اللازم لإجراء المعالجة أو تقديم الخدمة. الطابور يمكن أن يمثل في الذاكرة باستخدام المؤشرات القوائم المتصلة أيضا. ونظرا لأن هذا النوع من التمثيل يقوم على فكرة الحجز الديناميكي للذاكرة فإن مشكلة الوصول إلى وضع الفائض غير واردة.

**** وفيما يلي نستعرض كيفية تمثيل الطوابير باستخدام المصفوفات أولا ومن ثم باستخدام المؤشرات:**

أولاً :تمثيل الطوابير التتابعي (باستخدام المصفوفات أحادية البعد)Queues as Arrays : فكرة هذا النوع من التمثيل يقوم على وجود مصفوفة أحادية بعدد (MaxSize)من المواقع ووجود متغيرين يشير أحدهما إلى بداية الطابور والآخر يشير إلى نهاية الطابور . ولعلك تلاحظ أيضا بأن قيمة المتغير (REAR) لا تساوي MaxSize في هذه الحالة. كما أن قيمة المتغير (FRONT) لا تساوي واحد (1) أو الدالة الدنيا للمصفوفة في كل الحالات. فبينما تتحدد قيمة الأول في ضوء عمليات الإضافة، فإن قيمة المتغير الثاني تتحدد في ضوء عمليات الحذف.

ثانياً :التمثيل الم تصل(الطوابير باستخدام القوائم المتصلة) (Queues as Linked Lists): يقوم هذا النوع من التمثيل على استخدام المؤشرات التي توفرها لغات البرمجة (مثل لغة C++)، ويسمح للمبرمج بتخزين الطابور على شكل قائمة متصلة مفردة. وعلى خلاف أسلوب التمثيل التابعي، فإن هذا النوع من التمثيل لا يحجز في الذاكرة إلا عددا من المواضع مساوية لعدد القيم التي يشتمل عليها الطابور في لحظة معينة. وهو بذلك انعكاس طبيعي لحقيقة الطابور ونهجه العملي.

*أما من الناحية البرمجية لا يكاد يختلف عن القوائم المتصلة المفردة .فكل عنصر من عناصر الطابور يتكون من سجل يتضمن حقلين :أحدهما للقيمة البيانية والأخرى للمؤشر ،وبالإضافة الى ذلك لدينا مؤشرين خارجيين وهما :المقدمة والمؤخرة.تدريب١صفحة ١٦٦.

****تنفيذ العمليات المستخدمة على الطوابير آخذين بعين الاعتبار الطريقتين المختلفتين لتمثيل الطوابير .**

1. **إنشاء الطابور QueueCreate:** تقوم هذه الدالة بتهيئة الطابور للاستخدام لأول مرة بجعله طابورا خاليا.

*في حالة التمثيل التتابعي (المصفوفات) كل ما يجب عمله هو جعل REAR=-1،،، FRONT=-1 وعليه فإن الإجراء هو: void MyQueue::QueueCreate (MyQueue&Q)

{FRONT=-1;. REAR= -1.10؛}

**أما في حالة التمثيل المتصل فإن ما يقوم به الإجراء هو جعل FRONT=NULL ، و REAR = NULL كما يلي:

```
void MyQueue::QueueCreate(){

FRONT=NULL; REAR=NULL;}
```

2. **فحص فيما إذا كان الطابور ممتلئة QueueFull:**

في حالة التمثيل المتتابع فإن الطابور يكون ممتلئا إذا كانت جميع مواقع المصفوفة مستخدمة. ويكون ذلك في حالة أن FRONT=0 و REAR= Maxsize

وعليه فإننا نستطيع كتابة QueueFull كما يلي: boll MyQueue::Queue FullMjQueue Q){

```
if(Q.FRONT==0&& Q.REAR==Maxsize)

؛return true

else    return false ;}
```

**أما في حالة التمثيل المتصل فإن حجم الطابور يعتمد على حجم الذاكرة. ويعتبرالطابور ممتلئة عندما تفشل العملية new في تعيين موقع جديد، يضمه إلى باقي عناصر الطابور.

3. **التحقق فيما اذا كان الطابور فارغا QueueEmpty:**

*يكون الطابور فارغا في حالة التمثيل التتابعي إذا كانت FRONT=-1 REAR=-1 وعليه فإن الدالة QueueEmpt يمكن كتابتها كما يلي: bool MyQueues::QueueEmpty(MyQueueQ){

if(Q.FRONT==-1&& Q.REAR==-1)

return true؛

else return false }؛

*أما في حالة التمثيل المتصل فيكون الطابور فارغة إذا كانت FRONT=NULL وREAR=NULL وعليه نستطيع كتابة QueueEmpty كما يلي : bool MyQueue::QueueEmpty(){

if(FRONT==NULL && REAR==NULL)

return true؛

else return false؛

4.**<u>عملية الإضافة إلى الطوابير (Enqueue)</u>** تتم عملية الإضافة إلى الطابور في اتجاه واحد هو مؤخرة الطابور . وهي تتأثر إلى حد كبير بأسلوب التمثيل المستخدم في الذاكرة، وبمعدل الحذف الذي يتم على الطابور . **إضافة القيم إلى الطابور شكل ٦و٧ صفحة ١٧٠ .**

**** الخوارزمية (1) صفحة ١٧١ تتعلق بالتمثيل التتابعي :**

void MyQueue::EnQueue(MyQueue& Q, int element){ int i, last

if(Q.FRONT==0 && Q.REAR==Maxsize)

cout<<"queue is full .. overflow";

else {if(O.FRONT= -1){

Q.FRONT=0؛

Q.REAR=1;}

else if (Q.REAR==-1){ last=1

for (i=Q.FRONT;i<=Q.REAR;i++){

data[last]=data[i];

last==last+1 ;}

Q.REAR==Q.REAR - Q.FRONTA+1;

Q.FRONT=1;}

Q.REAR=Q.REAR + 1 ;

data[Q.REAR]=element }؛

*****إن هذه الخوارزمية التي تعالج مسألة إضافة أحد العناصر إلى طابور ممثل في الذاكرة على شكل مصفوفة تنطوي على أربعة أوضاع مختلفة:**

أولا: ليس هناك أي متسع لإضافة اي عناصر جديدة، وبذلك يوصف الطابور بأنه قد وصل إلى وضع الفائض. ويستدل على ذلك من خلال قيمة كل من المتغيرين (FRONT) و (REAR) ، حيث تم التعبير عن ذلك بالاختبار الشرطي :if(Q.FRONT=0&&Q.REAR=aMarsize)

ثانيا: إن الطابور خال من القيم، وبذلك فإن القيمة المضافة تشكل العنصر الأول في الطابور ويستدل على خلوه من القيم من خلال قيمة المتغير (Q.FRONT). وقد تم التعبير عن هذا الوضع في الخوارزمية بالاختبار الشرطي: if (Q.FRONT==-1)

ثالثا: لم يعد هناك مشع لإضافة أي قيم جديدة إلى نهاية المصفوفة إلا أن هناك أماكن شاغرة في بدايتها. وبذلك فإنه لا بد من إعادة تنظيم عملية التخزين من خلال إزاحة القيم الموجودة إلى بداية المصفوفة، وبالتالي تحويل الأماكن الشاغرة إلى نهاية المصفوفة بدلا من أولها. ويستدل على هذا الوضع من خلال قيمة المتغير (REAR)، حيث يتم التعبير عن ذلك في الخوارزمية بالاختبار الشرطي التالية(if (Q.REAR==Maxsize

رابعا: إن هناك مجالا لإضافة قيمة جديدة إلى الطابور في مكانها المناسب. وهذا هوالوضع الاعتيادي الذي نصل إليه في الخوارزمية، إذا لم يتحقق اي من الاختبار الشرطية الثلاثة السابقة. تدريب ٢صفحة١٧٣.

* ينبغي أن نذكر بانه ليس لدينا أي من الأوضاع المختلفة التي ذكرناها أعلاه فيما يتصل باستخدام المصفوفات. فإما أن يكون الطابور خاليا (empty)، و بذلك فإن قيمة كل من المؤشرين (REAR) و (FRONT) هي NULL، أو يكون هناك في الطابور عنصر واحد أو أكثر، وبذلك فإن الإضافة تتر بسهولة كبيرة. الخطوات التي تنطوي عليها عملية الإضافة من **خلال الخوارزمية (2)صفحة١٧٣:**

```
void MyQueue::EnQueue(int element){

QPtr  *newNode;؛

newNode=new QPtr;

newNode->value=element;؛

newNode->next= NULL;؛

if (REAR=NULL) // queue is empty

FRONT= newNode;؛

else REAR->next=newNode;؛

REAR= newNode;}؛
```

تدريب ٣ صفحة ١٧٤ .

ولو حاولنا تبع ما يحدث في الخوارزمية 2 لوجدنا، أن الجمل الثلاثة الأولى ستنفذ على كل حال، وبذلك يتم حجز المكان المناسب للعنصر الجديد ثم اسناد القيمة إليه. والفرق الوحيد يكمن في شقي التركيب الشرطي. **فكما ترى أن الحالة الأولى تعبر عن طابور خال من القيم، وبذلك فإن الإجابة على السؤال: (REAR==NULL) هي بالإثبات. ومن هنا، يتم تنفيذ جواب الشرط مباشرة. وبذلك يصبح العنصر المضاف هو العنصر الوحيد في الطابور. **وخلو الطابور من القيم قد يعني شيئا واحدا من اثنين: الأول** إن الطابور لم يدخل مرحلة الاستعمال والنشاط بعد، وبذلك يكون العنصر المضاف هو العنصر الأول والوحيد. **أما الثاني** فهو أن الطابور في مرحلة النشاط الفعلي وأن عملية الحذف قد تساوت مع عملية الإضافة وبذلك تم حذف جميع العناصر التي دخلت الطابور حتى الآن.

أما الحالة الثانية، فإنها تمثل الوضع الاعتيادي للطابور. وهو وضع يتسم بالحركة والزيادة والنقصان. وطالما أن قيمة المؤشر (REAR) ليست NULL، فإن الشق الثاني للتركيب الشرطي المعبر عنه بكلمة else هو ما سيتم تنفيذه. وهذا الأمر لا يختلف باختلاف عدد العناصر، فسواء تضمن الطابور عنصرا واحدا أو تضمن عشرات العناصر فإن أسلوب تنفيذ الخوارزمية لن يتغير في هذا الشأن.

5.**<u>عملية الحذف من الطوابير (Dequeue):</u>** تتم عملية الحذف في الطوابير في اتجاه واحد هو مقدمة الطابور. وتتأثر عملية الحذف بأسلوب تمثيل الطوابير في الذاكرة،كما أنها تؤثر على عملية الإضافة. ذلك أن حذف أحد العناصر يفسح المجال لإضافة عنصر آخر. ولكن ينبغي أن نذكر بأن تأثير أسلوب التمثيل لا يصل إلى

المستوى نفسه الذي لمسناه في عملية الإضافة. والاختلاف القائم بين التمثيل التتابعي والتمثيل المتصل لا يتجاوز مسألة الصياغة البرمجية. أما أوضاع الحذف بحد ذاتها فلا تعكس أي اختلاف ملموس بين الأسلوبين.

****إن هناك ثلاثة أوضاع عامة لحذف العناصر من الطوابير :**

الحالة الأولى الممثلة بالطابور الخالي: ليس لدينا، أو لم يعد لدينا، في الطابور أي قيم نتعامل معها. وبذلك تتوقف عملية الحذف ريثما تدخل الطابور قيم أخرى، ويتم تنشيط عملية الحذف من جديد. وكما تلاحظ فإننا نستدل على هذا الوضع من خلال قيمة أي من المتغيرين (FRONT) أو (REAR) اللذين يعبران عن هذا الوضع بالقيمة 1- أو NULL.

أما في الحالة الثانية: فإن لدينا عنصرا واحدا في الطابور. وهذا يتضح من تساوي قيمة المتغيرين (FRONT) و (REAR)، مع الأخذ بالاعتبار أن هذه القيمة ليست صفرا أو NULL. فكلا المؤشرين يشيران إلى هذا العنصر الوحيد، ومعنى ذلك أن اجراء للحذف على هذا العنصر سيؤدي إلى إخلاء الطابور من القيم والتحول إلى الوضع الذي تم عرضه في الحالة الأولى.

الحالة الثالثة : فإن لدينا الوضع الاعتيادي والمتمثل بوجود قيمتين أو أكثر في الطابور ومعنى ذلك أن حذف إحدى القيم من الطابور لن يؤدي إلى إخلائه وبالتالي فإن التعديل يقتصر على المتغير (FRONT).

وفيما يلي نعرض لتفاصيل التعامل مع هذه الأوضاع الثلاثة من خلال الخوارزميتين التاليتين. إن الخوارزمية (3) تتعلق بالطوابير باسلوب تمثيل الطوابير باستخدام المصفوفات،

```
void MyQueue::DeQueue(MyQueue& Q, int& element){

if(Q.FRONT==-1 )

cout<<"the queue is empty";

else

if (Q.FRONT==Q.REAR){

element=Q.data[Q.FRONT];

Q.FRONT= -1 ;

Q.REAR=-1;}

else  {element=Q.data[Q.FRONT];

Q.FRONT=Q.FRONT + 1;}
```

**احتفظنا بالقيمة المحذوفة مؤقتة من أجل أي معالجة مطلوبة عليها.

***والخوارزمية (4) تختص بالطوابير الممثلة على شكل قوائم متصلة:**

```
void MyQueue::DeQueue(int element){

if (FRONT==NULL)

cout<<" the queue is empty"

else   if (FRONT=REAR){

element=FRONT->value;

FRONT=NULL؛
```

REAR=NULL;

else {element=FRONT->value

FRONT=FRONT->next; } }

**وبمقارنة هذه الخوارزمية بالخوارزمية السابقة ندرك مدى التشابه بينهما من حيث معالجة مسألة الحذف. فالفرق بين عملية الحذف القائمة على استخدام المصفوفات وعملية الحذف القائمة على استخدام القوائم المتصلة هو فقط في صيغة الجمل، أما منطق العملية وتسلسل الخطوات فهو متطابق تماما.تدريب ٤صفحة ١٧٧.

** **الطابور الدائري (Circular Queue):** تصور منطقي للطابور يقوم على النظر إلى الطابور في حالتي الإضافة والحذف على أنه يشكل حلقة متكاملة يتصل أولها بآخرها،وبناء على هذا المفهوم الجديد فإن اضافة ع نصر جديد إلى الطابور يؤدي إلى تحريك المؤشر REAR خطوة واحدة إلى الأمام باتجاه عقارب الساعة.كما أن حذف عنصر من الطابور يؤدي إلى تحريك المؤشر (FRONT) خطوة مماثلة باتجاه عقارب الساعة إلى الموضع التالي.

في ضوء هذه الطبيعة الدائرية للطابور وحركة كل من المتغيريت (FRINT)و (REAR) باتجاه عقارب الساعة، فإننا سنواجه مشكلة تتعلق بتحديد الحالة التي يكون فيها الطابور خاليا من القيم، ومشكلة أخرى مماثلة تتعلق بتحديد الحالة يكون في الطابور مليئة بالقيم أو ما أشرنا إليه من قبل بوضع الفائض. وسبب الإشكال يعود إلى أن قيمة المتغير (FRONT) قد تكون أكبر من قيمة المتغير (REAR) الأمر الذي لم نصادفه في تعاملنا مع الطوابير الاعتيادية. **ولتوضيح هذه الفكرة مثال١ صفحة ١٨٠.

****الطابور يمكن أن يصل إلى وضع الفائض او الامتلاء (Overflow) في حالتين هما:**

أولا: عندما تصبح قيمة (FRONT = 0) وقيمة (REAR=MaxSize).

ثانيا:عندما تصبح قيمة(REAR=FRONT-1)

أما الوضع الخالي (Empty) فإنه يحدث في حالة واحدة فقط، وهي عن تكون قيمة (FRONT=RNAR =-1) حيث يتم حذف العنصري الوحيد المتبقي في الطابور وتصبح قيمة (FRONT=-1) وقيمة (REAR=-1).

*إن الطبيعة الدائرية اللطابور تنعكس من خلال قيمتي (FRONT) (REAR)، وذلك في الحالتين التاليتين:

أولا: عندما تكون قيمة (REAR=MaxSize) وقيمة (FRONT >0)

ثانيا: عندما تكون قيمة(FRONT=MaxSize) وقيمة (MaxSize>REAR >0) وفي كلتا الحالتين تصبح قيمة المتغير المعني بالحركة واحد بعد أن كانت قيمته MaxSize هذا هو ما يميز الطوابير الدائرية عن الطوابير الاعتيادية

****وفيما يلي نستعرض خوارزميتي الإضافة والحذف لهذه الصيغة في معالجة الطوابير. الخوارزمية (5):**

void MyQueue::EnQueue (MyOuene& Q, int element){

int state;

state=Q.REAR - Q.FRONT + 1;

if (state==0||state==MaxSize)

cout<<"queue is full .. overflow";

else

if (O.FRONT==-1){

Q.FRONT=0;

؛Q.REAR=0}

else

if (Q.REAR-=-1)

Q.REAR=0;

else Q.REAR==Q.REAR + 1

Q.data[Q.REAR]==element;}

هذه الخوارزمية فيها أربعة أوضاع مختلفة للإضافة، وهي:

أولا: الطابور ممتلئ، وبذلك فإنه ليس هناك مجال لإضافة أي قيم جديدة

ثانيا: الطابور فارغ، وبذلك فإن العنصر المضاف هو الوحيد في الطابور .

ثالثا: الطابور لم يعد يسمح بالإضافة إلى نهاية المصفوفة، وبذلك تتم الإضافة إلى بداية المصفوفة نتيجة لتطبيق فكرة الطابور الدائري

رابعا: الطابور في وضعه الاعتيادي، وبذلك تتم الإضافة في المكان المناسب. **والشكل (14) يعرض هذه الأوضاع الأربعة،**

الخوارزمية (6): void MyQuene::DeQueue(MyQueue& Q, int& element){

if (Q.FRONT==-1)

cout<<"queue is empty... undeflow";

else{

element=Q.data[Q.FRONT];

if (Q.FRONT=Q.REAR)

Q.FRONT= -1 ; Q. REAR= -1 ;}

else

if (Q.FRONT==MaxSize)

Q.FRONT=0; else

Q.FRONT=Q.FRONT + 1 }}

****تعالج هذه الخوارزمية أربعة أوضاع مختلفة شانها شأن خوارزمية الإضافة وهذه الأوضاع هي:**

أولا: الطابور فارغ، وبذلك فإنه ليس هناك حذف.

ثانيا: الطابور يتكون من عنصر واحد، وبذلك فإن الطابور يصبح خاليا من القيم بعد إجراء عملية الحذف.

ثالثا: الطابور مكون من مجموعة من القيم، أحدها في نهاية المصفوفة والأخرى موجودة في بداية المصفوفة، وبذلك يتحرك المؤشر باتجاه عقارب الساعة إلى بداية المصفوفة.

رابعا: الطابور في وضعه الاعتيادي، وبذلك تتم عملية الحذف في المكان والأسلوب المناسبين **والشكل (15) يعرض هذه الأوضاع الأربعة. مثال ٥صفحة ١٨٥ .**

*تلجأ نظم التشغيل إلى تبني أسلوب معين لتجاوز مشكلة تفاوت الوقت .ومن بين هذه الأساليب المتبعة صيغة طابور الأولوية :

*****طابور الأولوية (Priority Queue):** نوع من الطوابير يقوم على إضافة العناصر الجديدة إلى الطابور في المكان المناسب وفقا لدرجة الأولوية، وليس في آخر الطابور كما هو معتاد. ***هناك أسلوبين لتمثيل هذه الصيغة :**

الأسلوب الأول: هو تمثيل الطابور على شكل قائمة خطية (أي باستخدام مصفوفة ذات بعد واحد) أو متصلة (أي باستخدام قائمة متصلة)، بالصيغة الدائرية أو غير الدائرية. وفي هذه الحالة لن يحتاج الطابور إلا للمتغير (FRONT)، والسبب في ذلك أن الإضافة ستتم في المكان المناسب ضمن تسلسل الأولويات ولا يوجد حاجة إلى استخدام المتغير (REAR)، وإن تم استخدامه في حالة القوائم الاحادية فما ذلك إلا الغايات تحديد نهاية القائمة وليس لأغراض الإضافة بالتخصيص. أما الحذف فإنه سيتم من البداية بالأسلوب نفسه الذي تم اتباعه في الطوابير بصفة عامة، سواء أكانت دائرية أم غير دائرية.

**ان كل عنصر من العناصر قد تم التعبير عنه بسجل يتكون من ثلاثة حقول في حالة القوائم المتصلة، وهي: القيمة نفسها، ودرجة الأولوية، ومؤشر إلى العنصر التالي . ومن حقلين في حالة القوائم الاحادية وهما: القيمة والأولوية.

وفيما يلي تعبر عن هذه الحقول في لغة C++ بالتعريفات البرمجية التالية **أولاً طابور الأولوية كقائمة متصلة:**

```
class QPtr. {public:  char INFO;   int PRTY[4]      QPtr* NEXT;};

class MyQueue {public:   QPtr*FRONT;    QPtr* REAR;}
```

ثانيا: طابور الأولوية كقائمة احادية

```
class MyQueue{ private:    char info;

int PRTY [4], FRONT, REAR;

int QUEUE[ MaxSize];

public :.....};
```

**إن الاختلاف بين طوابير الأولوية والصيغ الأخرى للطوابير تكمن في أسلوب الإضافة. فبينما تتم عملية الحذف على النحو الذي وصفناه فيما سبق، فإن عملية الإضافة لا تختلف عن الأسلوب المتبع في إضافة عناصر جديدة إلى القوائم المتصلة أو القوائم الاحادية المرتبة تصاعدية. وينبغي التنبيه هنا إلى أن العنصر المضاف ينبغي أن يأتي بعد كل العناصر الموجودة في الطابور التي تحمل درجة الأهمية نفسها في حالة تشابه قيمة الأولوية.

الأسلوب الثاني: استخدام المصفوفات الثنائية. وفي هذه الحالة يخصص لكل فئة من فئات الأولوية صف مستقل. كما يمكن النظر إلى هذا الصف على أنه يشكل قائمة مستقلة : دائرية أو غير دائرية. وبذلك فإنه يتم تخصيص متغيرين لكل صف شأنه في ذلك شأن الطوابير المستقلة، أحدهما للإشارة إلى بداية طابور ذلك الصف وهو (FRONT) والآخر للإشارة إلى مؤخرة طابور ذلك الصف وهو (REAR). بناء على هذا الأسلوب في تمثيل طوابير الاولويه نستطيع أن نقوم بعمليتي الإضافة والحذف على كل طابور من هذه الطوابير الأربعة بالأسلوب نفسه. ولكن ينبغي أن نراعي أن الإضافة يمكن أن تتم في أي صف من الصفوف الأربعة وفقاً لدرجة الأولوية، إلا أن الحذف ينبغي أن يبدأ بالقيم الموجودة في الصف الأول، متلوة بالقيم الموجودة في الصف الثاني... وهكذا على التتابع. **تدريب ٦ صفحة ١٨٩.**