الجامعة الافتراضية السورية
SYRIAN VIRTUAL UNIVERSITY

مقدمة في البرمجة

الدكتور زهير دحروج

ISSN: 2617-989X

Books & Refrences

مقدمة في البرمجة

زُهير دحروج

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

Introduction to Programming

Zoher Dahrouj

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

ISSN: 2617-989X

الفهرس

ISSN: 2617-989X

الفهرس

ISSN: 2617-989X

الفهرس

ISSN: 2617-989X

الفهرس

ISSN: 2617-989X

الفهرس

ISSN: 2617-989X

الفصل الأول:
البرنامج الحاسوبي

الكلمات المفتاحية:

ترميز، مُترجم، لغة برمجة، خوارزمية، لغة الآلة، لغة برمجة إجرائية، لغة برمجة غرضية التوجه، تصميم من القمة إلى القاعدة، لغة الخوارزميات.

ملخص:

يستعرض هذا القسم مفهوم البرنامج الحاسوبي من خلال عرضه لمراحل تطور العتاد الحاسوبي الصلب، وأنظمة تشغيله، وأساليب الترميز، ولغات البرمجة المستخدمة لتطوير الأنظمة البرمجية، بالإضافة إلى مراحل تطور منهجيات وأدوات تصميم وبناء هذه الأنظمة من خلال التركيز على "لغة الخوارزميات" Pseudo–Code كأداة مساعدة على تصميم البرامج الصغيرة الحجم، أو التعبير عنها على نحوٍ مستقل عن لغة البرمجة. كما يستعرض أنماط لغات البرمجة وأنواعها كلغات البرمجة الإجرائية ولغات البرمجة الغرضية التوجه.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- نظام التشغيل، البرنامج الحاسوبي
- المترجم
- أنماط الترميز
- أنماط لغات البرمجة
- مفهوم الخوارزمية وأمثلة عنها
- منهجية تطوير البرمجيات
- لغة الخوارزميات
- المخططات التدفقية

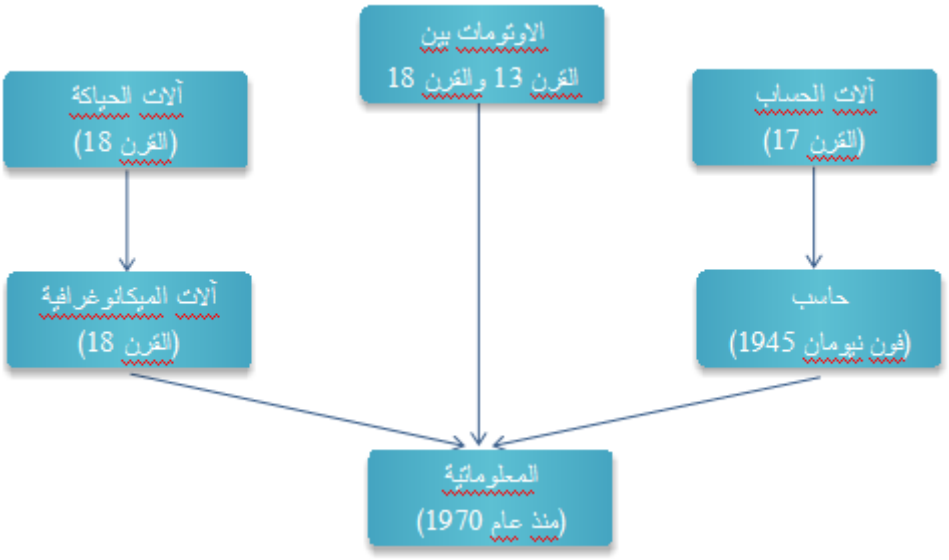
المخطط:

1. الحاسوب – الآلة
2. تطور العتاد الحاسوبي والديمقراطية المعرفية من معلوماتية "النُخبة" إلى معلوماتية الجميع
3. نظم التشغيل
4. الحاسوب ونظام التشغيل
5. التصنيفات الرئيسة لأنواع نظم التشغيل وتطورها
6. ترميز المعلومات
7. البرامج الحاسوبية
8. لغات البرمجة
9. اللغات البرمجية عالية المستوى: لمحة تاريخية
10. اللغات البرمجية عالية المستوى: اللغات الإجرائية (1)
11. اللغات البرمجية عالية المستوى: اللغات الإجرائية (2)
12. اللغات البرمجية عالية المستوى: اللغات الوظيفية
13. اللغات البرمجية عالية المستوى: اللغات المنطقية
14. اللغات البرمجية عالية المستوى: اللغات الغرضية التوجه
15. المترجمات
16. أسئلة
17. التطوير المنهجي للبرمجيات
18. استراتيجيات وضع الحلول البرمجية
19. المخططات التدفقية
20. الخوارزميات
21. لغة الخوارزميات (pseudo code)
22. التعليمات الأساسية للغة الخوارزميات (pseudo code)
23. تعليمة القراءة read
24. تعليمة الكتابة write
25. تعليمة الإسناد
26. التعليمة الشرطية
27. التعليمة التكرارية: while
28. منهجية كتابة نظام برمجي
29. أمثلة كلاسيكية عامة وهامة
30. أسئلة
31. نشاط

1. الحاسوب – الآلة

يبدأ تاريخ المعلوماتية وعلوم الحاسوب مع اختراع أدوات الأتمتة والحساب التي ارتبط تطورها بثلاثة خطوط فكرية أساسية جرى التعبير عنها بثلاثة أنماط من الآلات:

- الآلة الحاسبة
- الأوتومات
- الآلة القابلة للبرمجة

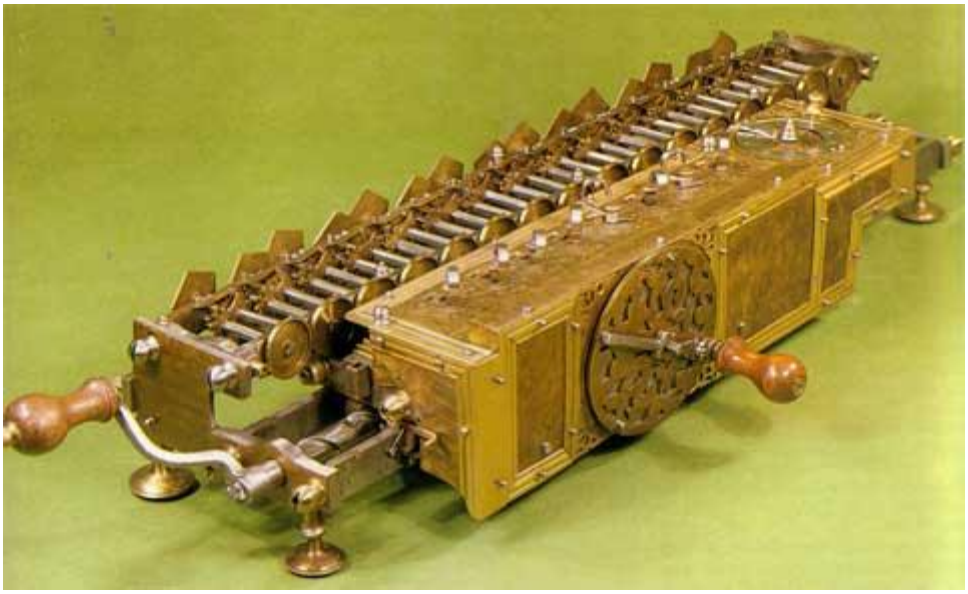


يبدأ تاريخ المعلوماتية وعلوم الحاسوب مع اختراع أدوات الأتمتة والحساب التي ارتبط تطورها بثلاثة خطوط فكرية أساسية مثلت ما ينتظره الإنسان من الآلة التي يخترعها ويطورها، وجرى التعبير عنها بثلاثة أنماط من الآلات: الآلة الحاسبة، الأوتومات، الآلة القابلة للبرمجة.

الآلة الحاسبة:

اخترع Pascal في القرن السابع عشر آلة حساب دعاها La Pascaline لتنفيذ عمليتي الجمع والطرح، وقد اعتمد في بنائها على المحسب الصيني القديم والذي يرجع تاريخه إلى مئات الأعوام قبل الميلاد. ومع نهاية القرن السابع عشر حسَّن Leibniz آلة باسكال بإضافة عمليتي الضرب والقسمة عليها.

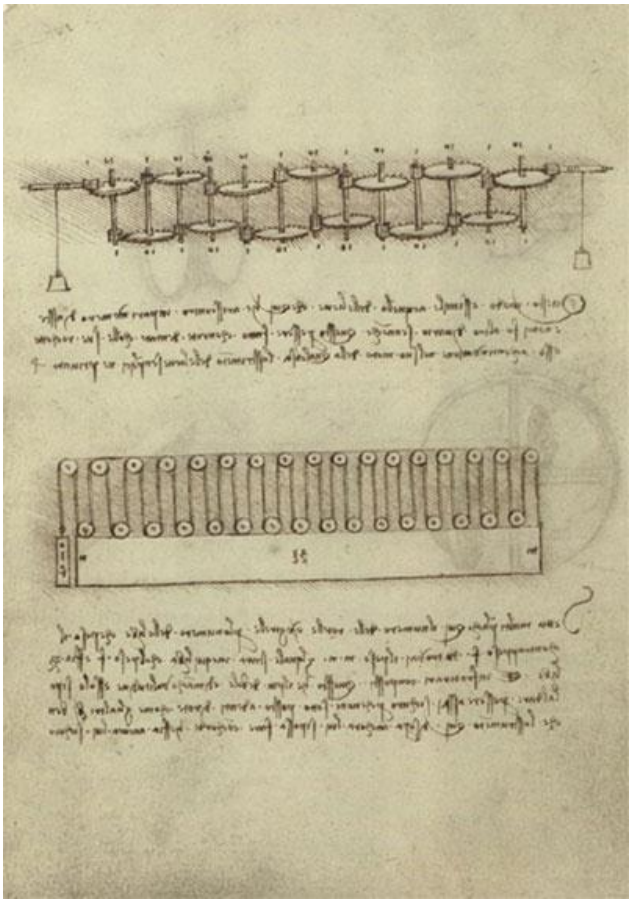
لاباسكالين



آلة ليبنيتز

الأوتومات:

بدأ تطوير الآلات الميكانيكية التي كانت تُستخدم في العمليات العسكرية وفي الساعات الفلكية منذ القرن الثاني عشر الميلادي واستمرت هذه الآلات الميكانيكية بالتطور حتى القرن الثامن عشر. وتظهر نماذج هذه الآلات وأساليب عملها في التصاميم التي تركها Leonardo De Vinci للكثير من الآلات العسكرية والمدنية.



تصميم دافنشي

الآلات القابلة للبرمجة:

بدأ مفهوم الآلات القابلة للبرمجة بالظهور مع اختراع آلات حياكة النسيج. وقد شهد هذا النوع من الآلات قفزة على يد Jaquard الذي عاش بين عامي 1752 و1834 وصمم أول آلة حياكة قابلة للبرمجة (ميكانيكياً)، حيث استُعملت نفس التقنية بعدها لبناء العديد من الآلات الحربية.

وقد نتجت العلوم المعلوماتية عن اندماج الأفكار والمعارف التي جرى تحصيلها من تطوير الآلات الآنفة الذكر.

نول جاكارد القابل للبرمجة

2. تطور العتاد الحاسوبي والديمقراطية المعرفية من معلوماتية "النُخبة" إلى معلوماتية الجميع

	الجيل الأول 1954-1945	الجيل الثاني 1965-1955	الجيل الثالث 1973-1966	الجيل الرابع منذ عام 1974
المكونات	صمامات مُفرّغة (Tubes)	ترانزيستورات (Transistors)	دارات مُدمَجة متكاملة (Integrated Circuits)	VLSI
الذواكر	Ferrite Core Memory	Ferrite Core memory	دارات مُدمَجة متكاملة (Integrated Circuits)	VLSI
زمن المعالجة	10-2 ثانية	10-3 ثانية	10-6 ثانية	10-9 ثانية
نظام التشغيل	بدائي	برنامج وحيد مع معالجة برنامج واحد	عدة برامج مع معالجة برنامج واحد	الاشتراك بالمعالج مع معالجة عدة برامج بآن واحد

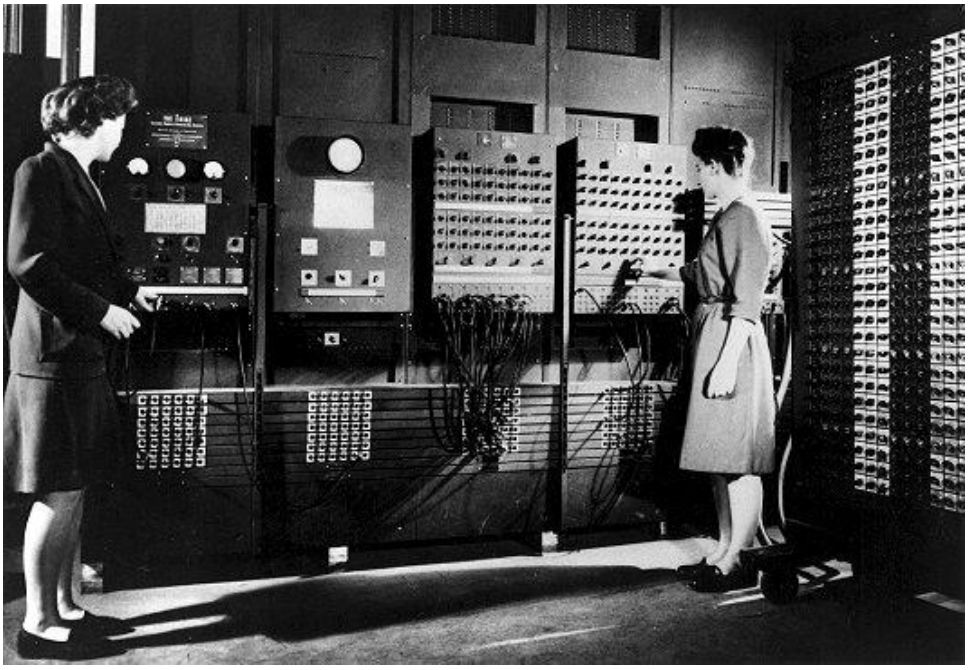
مرّ تطور الحاسوب بعدة مراحل يجري عادةً تصنيفها تحت إسم أجيال الحاسوب، وتُقسَم هذه الأجيال إلى:

الجيل الأول (1945-1954):

استخدمت دارات هذه الحواسيب الصمامات المُفرَّغة، وكانت ضخمة بحيث يصعب تحريكها. كما كانت تعليمات نظام التشغيل تُخزَّن داخلياً، وكان لا بد من إضافة صمامات وأسلاك حديدية جديدة عند بروز الحاجة لإضافة تطبيقات جديدة.

قامت شركة IBM بتصنيع أول حاسوب ضخم وإسمه IBM701. كما شهد عام 1951 تصنيع أول حاسوب أميريكي تجاري، وهو UNIVAC-1 والذي كان الهدف منه تجميع المعلومات السكانية الإحصائية. كان حاسوب UNIVAC-1 يحتاج إلى طابق بناء ضخم، وكان وزنه ثمانية أطنان، ويحتوي على أكثر من ثمانية آلاف صمام مُفرَّغ.

في ذلك الوقت كان استخدام الحاسوب محصوراً في بعض المراكز العسكرية الكبرى في بعض الدول العظمى.



حاسب من الجيل الأول

الجيل الثاني (1955-1965):

استخدمت هذه الحواسيب الترانزيستورات في تنفيذ العمليات الحسابية، واحتوت على ذاكرة مغناطيسية، واستخدمت أقراصاً وأشرطة مجدولة ممغنطة لتخزين المعطيات. وقد سمح هذا التصميم بتخزين البرامج، وأصبح بإمكان مدير النظام إدخال تعليمات التنفيذ، اعتماداً على لوحة مفاتيح. وقد ظهرت في هذه الفترة أولى لغات البرمجة كالـ Fortran التي كانت تُستخدم في تنفيذ الأعمال الحسابية، والـ Cobol والتي كانت تُستَخدَم في أتمتة بعض الأعمال الإدارية والمكتبية. حينها، كانت الدول الكبيرة والغنية فقط قادرة على اقتناء واستخدام الأدوات الحاسوبية.

حاسب من الجيل الثاني

الجيل الثالث (1966–1973):

استخدمت هذه الحواسيب الدارات المُدمَجة والمُتكاملة. لكن الحواسيب لم تكن متوافقة فيما بينها، بحيث كانت الطرفيات مصممة للاستخدام على حاسوب وحيد ولا تعمل مع أي حاسوب آخر، وكان يتوجب إعادة كتابة وترجمة نظم التشغيل الخاصة بأحد الحواسيب لكي تعمل على حاسوب آخر.

أنتجت شركة DEC (Digital Equipment Corporation) الأميريكية حاسوب PDP–8. كان هذا أول حاسوب بحجم صغير نسبياً، وكان هدفه التحكم في عمليات المعالجة الصناعية والعلمية، إلا أن التطبيقات الأخرى ذات الأغراض المختلفة بدأت بعد ذلك بالتوافر في الأسواق تدريجياً.

كما طورت شركة AT&T الأميريكية بالتعاون مع مختبرات Bell نظام التشغيل UNIX، والذي يعتمد على تعدد المستخدمين. وهو نظام التشغيل "الأب" لأنظمة تشغيل منتشرة ومشهورة في وقتنا الحالي: Linux, Android.

منذ ذلك الوقت، صار الحاسوب في متناول عدد أكبر من الدول والبلدان وصار بالإمكان اقتناؤه من قبل الجامعات والمؤسسات الحكومية والخاصة الكبيرة لاستخدامه في الأعمال العلمية.

حاسب من الجيل الثالث

الجيل الرابع (منذ عام 1974):

تميزت هذه الحواسيب بالدارات المتكاملة المُدمَجة ذات الأحجام الصغيرة جداً، وكانت سرعاتها عالية، وكانت الأجهزة التي تحويها موثوقة، ولها شاشات مرئية، ومساحات تخزين واسعة.

حازت شركة مايكروسوفت على رخصة لاستخدام نظام UNIX، وبدأت بتطوير نسخة من نظام Xenix للحواسيب الشخصية.

في عام 1980، اعتمدت شركة IBM على مهندسين هما Paul Allen و Bill Gates لابتكار نظام تشغيل حاسوب شخصي جديد حيث قاموا بشراء حقوق نظام تشغيل بسيط استخدموه كنموذج لنظام تشغيل مبدئي يدعى DOS. وقد سمحت IBM لكل من Paul Allen و Bill Gates بالإحتفاظ بحقوق تسويق نظام التشغيل MS-DOS، إضافة إلى حق استخدام الإسم التجاري DOS. كان نظام MS-DOS أو Microsoft Disk Operating System في البداية نظام تشغيل بسيط، مصمماً لتشغيل برنامج واحد، في آن واحد، ولمستخدم وحيد. في عام 1984، سوقت شركة Apple حاسوب Macintoshعلى نطاق واسع. وقد استخدمت حواسيب Apple Macintosh واجهات بيانية رسومية تعمل بالمؤشر، بدلاً من لوحة المفاتيح، كما كان الأمر عليه في نظام DOS. في نفس الوقت أصدرت Microsoft النسخة الأولى من نظام Windows (windows 3) وطورته عبر عقدين لتحوله من نظام خاص بحاسوب شخصي إلى نظام تشغيل يمكن استخدامه ضمن شبكات حاسوبية في المؤسسات.

وفي عام 1991 طور Linus Torvald نظام التشغيل LINUX المجاني ذو النص البرمجي المفتوح الذي يعمل على الحواسيب الشخصية والمشابه لنظام UNIX من حيث المكونات، بهدف محاربة احتكار Microsoft لأنظمة الحواسيب الشخصية.

في عصرنا هذا، أصبح الحاسوب أداة متوفرة للجميع ولم يعد مقتصراً على مجموعة الأخصائيين فقط!!!

حاسب IBM من الجيل الرابع

3. نظم التشغيل

تعريف:

يُعرَّف نظام التشغيل بأنه برنامج يدير/يشغِّل عتاد الحاسوب بحيث يوفر البرمجيات والتطبيقات الضرورية لتشغيل هذه العتاديات، كما يعمل كوسيط بين المستخدم والحاسوب بحيث يسمح للمستخدم باستثمار موارد الحاسوب (الذاكرة ووحدة المعالجة بشكل رئيسي) وتطبيقاته.

أسلوب تصميم نظام التشغيل:

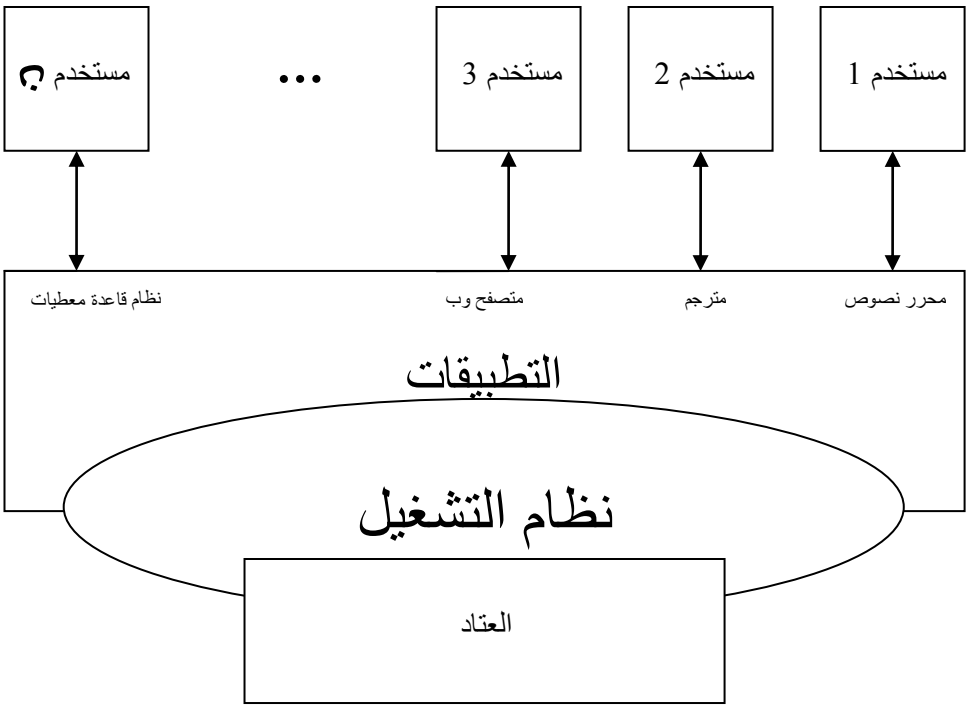
يختلف تصميم نظام التشغيل حسب البيئة التي يُفترض أن يعمل عليها، إذ يُصمم نظام التشغيل الذي يعمل على المخدِّمات على نحو يستطيع فيه استثمار العتاديات بالشكل الأمثل، في حين يصمم نظام التشغيل المُعدّ للعمل على الحاسبات الشخصية ليدعم تطبيقات متنوعة. بالتالي نلاحظ اختلاف وجهة التصميم لتكون إما ملائمة للمُستخدم النهائي في حالة الحواسب الشخصية، أو فعالة في استثمارها للعتاديات في حالة المخدِّمات.

4. الحاسوب ونظام التشغيل

مكونات النظام الحاسوبي:

- العتاديات
- نظام التشغيل
- التطبيقات
- المستخدمين

يمثل الشكل التالي بنية توضيحية للنظام الحاسوبي، ويُبين توضع نظام التشغيل ضمن تلك البنية:



يتولى نظام التشغيل مهمة الإشراف والمراقبة وتوفير البيئة الملائمة للتطبيقات والمستخدمين لكي يُنفذوا أعمالهم ويستثمروا موارد الحاسوب وتطبيقاته. إذ تشكل العتاديات في النظام الحاسوبي الموارد التي يجري الاعتماد عليها عند استثمار الحاسوب، وهي تشمل وحدة المعالجة المركزية، والذاكرة، وتجهيزات الدخل/خرج وغيرها، في حين تُعبِّر التطبيقات عن الأدوات التي يستخدمها المستثمرون لاستثمار الموارد.

يمكن النظر إلى نظام التشغيل كمدير للموارد، وكنظام تحكُم، وكنواة لتشغيل التطبيقات الحاسوبية:

- **نظام التشغيل كمدير للموارد:**

يتكون النظام الحاسوبي من العديد من الموارد العتادية والبرمجية (وحدة معالجة مركزية، وحدات خزن معطيات، ذاكرة رئيسية ...الخ)، حيث يتولى نظام التشغيل مهمة إدارة تلك الموارد وتوزيعها على المستخدمين بشكل مُنصِف يضمن فعالية أداء النظام الحاسوبي. وتبرز أهمية وقدرة نظام التشغيل على الإدارة في أسلوب معالجته للطلبات التي يمكن أن تؤدي إلى تعارض في استخدام الموارد.

- **نظام التشغيل كبرنامج تحكم:**

يمكن النظر إلى نظام التشغيل كبرنامج يتحكم بكيفية تنفيذ برامج المستخدمين بهدف منع حدوث الأخطاء، ومنع الاستخدام غير السليم للحاسب وخاصة فيما يتعلق باستخدام تجهيزات الدخل/خرج والتحكم فيها.

- **نظام التشغيل كنواة:**

إن المفهوم الذي يعتبر نظام التشغيل أداة إدارة للموارد الحاسوبية أو أداة تحكم يُولِّد بالضرورة تصوراً حول مكونات نظام التشغيل من البرمجيات، لذا يجدر بنا التنويه إلى التعريف الأكثر شيوعاً لنظام التشغيل –الذي يُطلق عليه اسم النواة– والذي يشير لنظام التشغيل على أنه البرنامج الذي يكون بحالة تنفيذ دائمة والذي تعمل تحت إشرافه التطبيقات البرمجية الأخرى.

5. التصنيفات الرئيسة لأنواع نظم التشغيل وتطورها

- نظم المهمة الوحيدة
- نظم المهمات المتعددة ونظم المشاركة بزمن المعالج
- نظم الحواسيب الشخصية
- النظم الموزعة

تطورت نظم إدارة الحواسيب تطوراً كبيرا منذ أن نشأت وحتى الآن، سواء كان ذلك التطور يؤثر على طبيعة نظام التشغيل بحد ذاته، أو كان يعبر عن جيل آخر من الأنظمة يقدم خدمات مغايرة أكثر تطوراً وتنوعاً من حيث دعمها للتطبيقات المختلفة وما تقدمه من مهمات، تجارية كانت أم علمية؛
لقد مرت دورة حياة نظم التشغيل بالعديد من المراحل فبدأت من خلال النظم ذات المهمة الوحيدة، وتطورت بعد ذلك لتصبح نظماً تدعم عدة مهمات في آن واحد، ثم بدأت تتشارك بالموارد كالمعالج أو الذاكرة، وترافق ذلك مع تطور أجيال الحواسيب الشخصية التي انتشرت انتشاراً واسعاً بين المستخدمين؛

تُعبِّر نظم المهمة الوحيدة عن نظم التشغيل البسيطة التي كان الحاسوب فيها يقوم بتنفيذ تطبيق واحد فقط، وتُمثِّل هذه النظم الشكل الأول لنظم التشغيل عند بداية ظهورها، حيث كانت الحاسبات في ذلك الوقت ذات حجوم ضخمة جداً وكانت تُدار من خلال واجهات تعليمات خاصة، أما أدوات الدخل / خرج فقد كانت تتمثل بقارئات البطاقات المثقبة وسواقات الأشرطة الممغنطة، كما كانت وسائط التخزين تتمثل عموماً بالبطاقات المثقبة والأشرطة الممغنطة؛

وتُعبِّر نظم المهمات المتعددة عن نظم التشغيل التي تستثمر الموارد على نحو يزيد من معدل استخدام وحدة المعالجة المركزية وبحيث يتم تنفيذ إجرائية في كل لحظة؛ يجري تخزين البرامج في قرص تخزين، كما يجري انتقاء مجموعة من تلك البرامج ونقلها إلى الذاكرة الرئيسية لكي يجري تنفيذها معاً، ولا يجري نقل كافة البرامج المخزنة لأنه غالباً ما تكون المعطيات المخزنة على القرص أكبر من سعة التخزين في الذاكرة الرئيسية ؛ تسمى عملية انتقاء البرامج التي ينبغي اختيارها أولاً بجدولة الأعمال.

ومع الانخفاض الكبير في تكلفة المعالجات أصبح بالإمكان امتلاك المستخدم لنظامه الحاسوبي الخاص به. أُطلق على هذا النوع من النظم اسم نظم الحواسيب الشخصية؛ وتزامن ظهور هذا النوع من النظم مع تطور التجهيزات الحاسوبية تطوراً كبيراً على صعيد الشكل والأداء، فعلى سبيل المثال تغيرت معظم أساليب الدخل التي كانت سائدة لتتحول إلى طرائق استخدام للوحة المفاتيح والفأرة، كما تغيرت معظم أساليب الخرج لتصبح من خلال شاشات عرض أو طابعات صغيرة الحجم عالية الأداء

يعتمد الاتجاه الحالي في تصميم نظم الحواسيب على مفهوم توزيع الحسابات بين عدة معالجات، يختلف هنا المفهوم المطروح عن مفهوم النظم التفرعية من مبدأ أن المعالجات لا تشترك بالذاكرة أو بالميقاتية إذ يمتلك كل معالج منها ذاكرته المحلية الخاصة، كما يتم التخاطب بين المعالجات من خلال أسلوب اتصال مناسب كشبكة محلية أو خطوط هاتف أو أية وسيلة أخرى. يُطلق على هذا النوع من النظم اسم النظم الموزعة. يمكن أن تختلف المعالجات المكونة للنظام الموزع حجماً أو أداءً، فيمكن أن تكون عبارة عن معالجات أو محطات عمل أو حواسب شخصية أو حتى منصّات، كما يمكن الإشارة إليها بأسماء مختلفة كمواقع وب أو كعقد شبكية، حيث تختلف التسمية بحسب السياق الذي يتم فيه الإشارة إلى تلك المعالجات.

6. ترميز المعلومات

تعريف:

تكون المعلومات المُخزَّنة في الحاسوب على شكل سلسلة من 0 و1 (اللغة الثنائية)، وبما أن الإنسان لا يتكلم اللغة الثنائية، نحتاج لترجمة تعليمات المستثمر المكتوبة بلغة برمجية خاصة، إلى هذه اللغة الثنائية. لذا نُعرِّف الترميز على أنه تابع تقابل بين معلومة، وبين سلسلة من 0 و1 تمثل هذه المعلومة وتكون قابلة للتخزين ضمن الآلة.

ترميز المحارف: الترميز ASCII (American Standard Code for Information Interchange)، والترميز UNICODE (Universal Code)

يعتبر الترميز ASCII أحد أهم أساليب الترميز المُتَّبعة في الأنظمة الحاسوبية. يسمح الترميز ASCII بشكله المُعدّل بترميز أي محرف على 8 بت. لذا يمكننا اعتماداً على مثل هذا الترميز، تمثيل 2^8 محرف (أي 256 محرف) مما يسمح بتمثيل الأبجديات الأوروبية كالإنكليزية، والفرنسية، والإسبانية،... الخ، بالإضافة إلى المحارف الخاصة كالأرقام وأحرف التنقيط وغيرها.

جرى إدخال تعديلات حديثة على أنظمة الترميز ضمن الأنظمة الحاسوبية بحيث سمحت بتمثيل المحارف على 16 بت، ودُعيت بالترميز العمومي (Unicode :Universal Code)، مما ساعد على توفير إمكانية تمثيل 65536 حرف، وأدى لفتح المجال أمام تمثيل الأحرف العربية، والصينية، والكورية وغيرها.

جدول الترميز ASCII

0...31		42	*	53	5	64	@	75	K	86	V	97	a	108	l	119	w
32		43	+	54	6	65	A	76	L	87	W	98	b	109	m	120	x
33	!	44	,	55	7	66	B	77	M	88	X	99	c	110	n	121	y
34	”	45	-	56	8	67	C	78	N	89	Y	100	d	111	o	122	z
35	#	46	.	57	9	68	D	79	O	90	Z	101	e	112	p	123	{
36	$	47	/	58	:	69	E	80	P	91	[	102	f	113	q	124	|
37	%	48	0	59	;	70	F	81	Q	92	\	103	g	114	r	125	}
38	&	49	1	60	<	71	G	82	R	93	]	104	h	115	s	126	~
39	‘	50	2	61	=	72	H	83	S	94	^	105	i	116	t	127	□
40	(	51	3	62	>	73	I	84	T	95	_	106	j	117	u		
41	)	52	4	63	?	74	G	85	U	96	`	107	k	118	v		

إن كل لون ضمن الجدول يشمل مجموعة من الرموز بينها رابط مشترك.

0...31	تسمى أحرف تحكم وليس لها شكل إظهار حرف، منها مثلاً رمز نهاية السطر (الرقم 12)
32	الحرف فراغ "space"
33..47	أدوات تنقيط
48..57	الأرقام (digits)
58..64	أدوات تنقيط
65..90	الأحرف الأبجدية الكبيرة
91..96	أدوات تنقيط
97..122	الأحرف الأبجدية الصغيرة
123..127	أدوات تنقيط

ترميز الأعداد الصحيحة: الترقيم

يمكن ترميز الأعداد الصحيحة كمحارف، إلا أن مثل هذا الترميز سيُعقِّد تنفيذ العمليات الحسابية على هذه الأعداد ضمن الأنظمة الحاسوبية. بالنتيجة، يمكن للحاسوب التعامل مع القيم الرقمية على نحوٍ أسهل إذا جرى وضع ترميز خاص لها. ندعو هذا الترميز بالترقيم.

عادةً، يجري التعامل مع القيم الرقمية الصحيحة كقيم عِشرية: فالرقم 5، والرقم 8، والرقم 90 هي أرقام صحيحة ممثلة على قاعدة الترقيم العِشّري بحيث تكون الأرقام محصورة بين 0 و9 وتكون قيم الأعداد محسوبة وفق القاعدة العشرية. فعندما نكتب العدد 5769 وفق القاعدة العِشّرية، يشير ترتيب الأرقام إلى قوة الرقم 10 المرتبطة بالرقم وهي في حالتنا:

$$5 \rightarrow 10^3$$
$$7 \rightarrow 10^2$$
$$6 \rightarrow 10^1$$
$$9 \rightarrow 10^0$$

وتكون قاعدة احتساب القيمة العشرية الموافقة لهذه الارتباطات من الأعلى إلى الأسفل:

$$(5 * 10^3) + (7 * 10^2) + (6 * 2^1) + (9 * 10^0) = 5769$$

يمكننا تعميم هذه القاعدة على أي قاعدة b مهما تكن b سواء كانت b=2 أو b=10 أو b=16. فعندما تكون b=10 ندعو قاعدة الترقيم، قاعدة عِشرية، وتكون الأرقام التي تؤلف الأعداد محصورة بين 0 و9، وعندما تكون b=2 ندعو قاعدة الترقيم، قاعدة ثُنائيّة، وتكون الأرقام التي تؤلف الأعداد محصورة بين 0 و1، وعندما تكون b=8 ندعو قاعدة الترقيم قاعدة ثُمانيّة وتكون الأرقام التي تؤلف الأعداد محصورة بين 0 و7.

بالنتيجة، تكون ارتباطات الأرقام التي تؤلف العدد 10010 الممثل ثنائياً كما يلي من اليسار إلى اليمين:

$$1 \rightarrow 2^4$$
$$0 \rightarrow 2^3$$
$$0 \rightarrow 2^2$$
$$1 \rightarrow 2^1$$
$$0 \rightarrow 2^0$$

وتكون قاعدة احتساب القيمة العشرية الموافقة لهذه الارتباطات من الأعلى إلى الأسفل:

$$(1 * 2^4) + (0 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0) = 18$$

تمارين:

احسب القيم العشرة الموافقة للقيم الثنائية التالية:

10000000 (→ 128)

10000010 (→ 130)

10000011 (→ 131)

11111111 (→ 255)

11111110 (→ 254)

7. البرامج الحاسوبية

تجري كتابة البرامج الحاسوبية على شكل تعليمات وتراكيب حسابية ومنطقية، وذلك باستخدام إحدى لغات البرمجة، مثل C#. وتجري ترجمة هذه التعليمات والتراكيب إلى سلاسل من الرموز الرقمية الثنائية 0,1 التي تعبر عن ترميز يفهمه الحاسوب وتدعى لغة الآلة.

تتضمن عملية البرمجة كتابة مجموعة من التعليمات على نحو متسلسل، بحيث يجري الحصول على النتيجة المطلوبة عند تنفيذ التعليمات المتسلسلة في الحاسوب. ويجري تخزين البرامج على القرص الصلب وتحميلها في الذاكرة الرئيسية عند بدء تنفيذ البرنامج وذلك للبدء بعملية التنفيذ.
تكون وحدة المعالجة المركزية مسؤولة عن معالجة التعليمات باللغة الثنائية 0,1 الناتجة عن عملية ترجمة البرنامج، هذه التعليمات هي ما نسميه لغة الآلة. يجري التنفيذ من خلال:

- نقل المعطيات ضمن وحدة المعالجة: أي من وحدة المعالجة إلى الذاكرة، أو من الذاكرة إلى وحدة المعالجة، أو من وحدة المعالجة إلى الطرفيات
- تنفيذ العمليات الحسابية
- تنفيذ العمليات المنطقية

كيف يعمل الحاسوب؟

إليك مثالاً في غاية البساطة:
(سميَّ حاسوب لأنه عمله الأساسي الحساب: أريد حساب مجموع قيم متحولين، يحويان أعداد، ووضع الناتج في متحول ثالث)

C= A+B

ستكرس خانات ذاكرة للمتحولات A, B, C على التوالي.
طريقة تنفيذ العملية الحسابية التفصيلية (معبراً عنها بلغة رمزية بالإنكليزية) هي كالتالي:

LOAD A, R1
LOAD B, R2
ADD R1,R2,R3
STORE R3,C

إنها فعلياً مجموعة تعليمات (instruction) تحدد للحاسوب مايجب فعله للقيام بالمطلوب (جمع متحولين)، وهذا شرحها:

1. اشحن (انقل) قيمة المتحول A من الذاكرة إلى السجل R1
2. اشحن (انقل) قيمة المتحول B من الذاكرة إلى السجل R2
3. اجمع قيمة السجل R1 و R2 وضع النتيجة في R3
4. خزّن قيمة السجل R3 في خانة الذاكرة C

نعم كل هذا لجمع عددين (هذه هي طريقة الحاسوب "أسطورة الذكاء في بعض الأذهان" لجمع عددين) ؟

ما نراه فعلياً في ذاكرة الحاسوب (قمنا بأخذ لقطة قبل التنفيذ) هو ما يلي:
المتحولات A, B, C كرست لها خانات الذاكرة بالعناوين 10, 11, 12 على التوالي.
وتحتوي على القيم 0, 64, 128 على التوالي.
وتمثيلها بالنظام الثنائي (على التوالي):

0000	0000	0000	0000	0000	0000	0100	0000
0000	0000	0000	0000	0000	0000	1000	0000
0000	0000	0000	0000	0000	0000	0000	0000

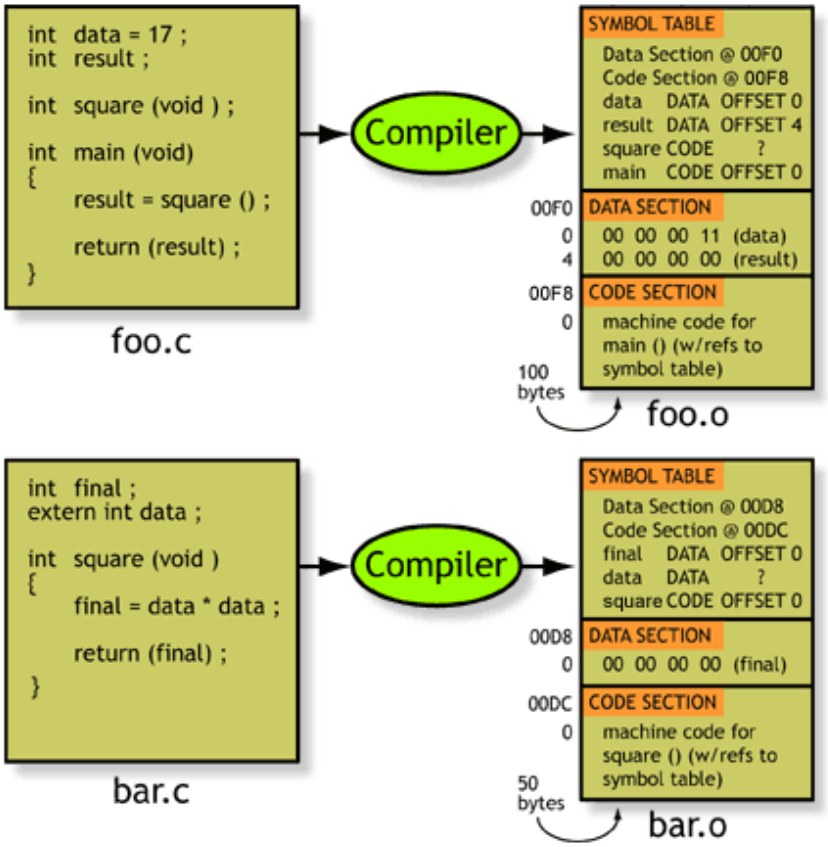
أما التعليمات الأربع فأخذت الشكل الثنائي التالي:

1010	0000	0000	0000	0000	0000	0001	0100
1011	0000	0000	0000	0000	0000	0010	0100
0000	0000	0000	0000	0011	0010	0001	0010
1100	0000	0000	0000	0000	0000	0011	0101

8. لغات البرمجة

هناك نوعان من اللغات البرمجية المستخدمة في الحواسب: اللغات منخفضة المستوى واللغات عالية المستوى.
ترتبط اللغات البرمجية منخفضة المستوى بالعتاد الصلب (نمط وحدة المعالجة، نمط النواقل وسعتها، ... الخ) وتدعى عادةً بلغة المُجمِّع وتَستخدِم رموزاً تمثل عمليات الحاسوب، ويتوجب ترجمة كافة الرموز المكتوبة بلغة المُجمِّع إلى لغة الآلة الممثلة بترميز خاص وسلاسل ثنائية (مؤلفة من 0 و 1). تجري عملية الترجمة باستخدام برامج خاصة تُدعى المُجَمِعات.

أما اللغات البرمجية عالية المستوى فتكون مستقلة عن العتاد الصلب، بحيث تجري كتابة البرامج بتعليمات وعبارات مشابهة للغة الإنكليزية. ولهذه اللغات عدة أصناف: اللغات الإجرائية، واللغات الوظيفية، واللغات غرضية التوجه. ويتوجب ترجمة كافة النصوص المكتوبة بلغة برمجة عالية المستوى إلى لغة الآلة الممثلة بسلاسل ثنائية (مؤلفة من 0 و 1). تجري عملية الترجمة باستخدام برامج خاصة تُدعى المُترجمات.



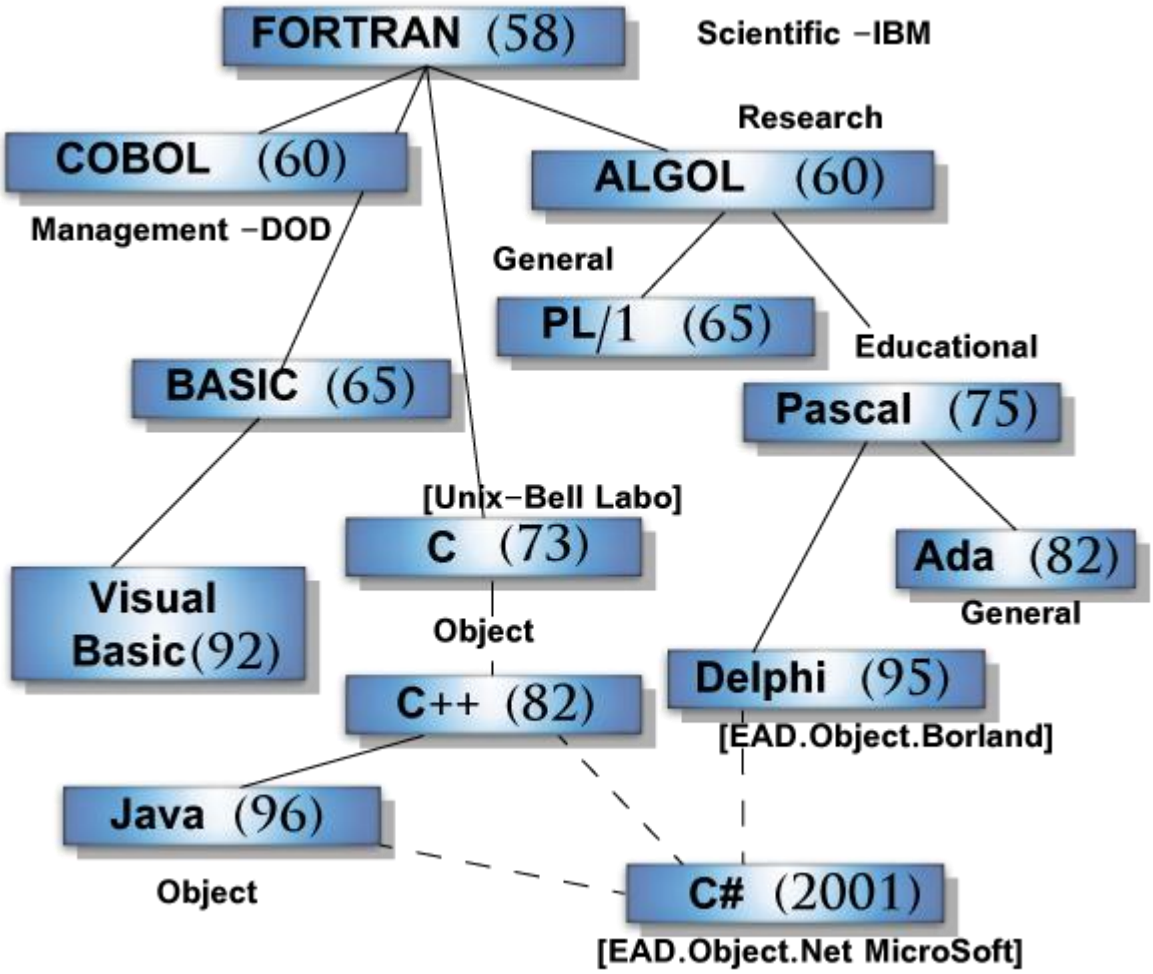
9. اللغات البرمجية عالية المستوى: لمحة تاريخية

تضمن اللغات البرمجية عالية المستوى تحقيق مجالٍ واسعٍ من المهام البرمجية المختلفة.
لقد جرى تطوير العديد من لغات البرمجة المختلفة على مر السنين بهدف تلبية الاحتياجات المتغيرة في تقنيات المعلومات:

- عام 1958: لغة Fortran
- عام 1964: لغة BASIC
- عام 1980: لغة ADA
- عام 1971: لغة Pascal
- عام 1972: لغة C
- عام 1986: لغة C++
- عام 1991: لغة Visual Basic
- عام 1995: لغة Java
- عام 2001: لغة C#.

تضمن اللغات البرمجية عالية المستوى تحقيق مجالٍ واسعٍ من المهام البرمجية المختلفة. فبالرغم من أن معظم اللغات البرمجية عالية المستوى قد صممت خصيصا لمجالات تطبيقية عامة، كلغة Fortran التي صُممت كلغة عامة، إلا أنها استُخدِمت في تطبيقات محددة التي استُخدِمت في حل المشاكل الرياضية والحسابية.
لقد جرى تطوير العديد من لغات البرمجة المختلفة على مر السنين بهدف تلبية الاحتياجات المتغيرة في تقنيات المعلومات. ففي عام 1964 قام كل من John Kemeny وThomas Kurtz من جامعة Dartmouth بتطوير لغة BASIC الموجهة لكافة الأغراض. وفي عام 1980 طورت وزارة الدفاع الأميريكية لغة ADA وهي لغة خاصة ببرمجة الحواسب، وتضمنت هذه اللغة إمكانيات خاصة بتصميم أنظمة دفاعية لتوجيه القذائف العسكرية. وفي عام 1971 ابتكر Niklaus Wirth لغة البرمجة Pascal، وابتكر Dennis Ritchie عام 1972 لغة البرمجة C في مختبرات شركة Bell الأميركية. وجرى تطوير لغة C++ اعتماداً على لغة C في مختبرات Bell التابعة لشركة AT&T الأميريكية عام 1986 وباتت تعتبر واحدة من أكثر اللغات البرمجية ذات التوجه الغرضي استخداماً. وطورت Microsoft عام 1991 لغة Visual Basic التي تعد قوية في تطوير واجهات برمجية تعمل في بيئة نظم Windows. وتبعتها في عام 1995 شركة Sun Microsystems الأميريكية بتطويرها للغة Java التي تدعم برمجيات الانترنيت، بما في ذلك مايدعى Java Applets.
وفي عام 2001 أصدرت Microsoft لغة البرمجة C# لتكون اللغة الأساسية لها في بناء التطبيقات في القرن الحادي والعشرين. وستكون هذه اللغة هي أداتنا في تطبيق مبادىء البرمجة.

10. اللغات البرمجية عالية المستوى: اللغات الإجرائية (1)



مثال عن برنامج بلغة إجرائية هي لغة FORTRAN

```
C  *** FORTRAN ***
   integer I,J,K
   write(*,*) 'Good Day'
  I =1
  K=0
   if (I.EQ.10) goto 100
10  read(*,*) J
   K=K+J
   I=I+1
   goto 10
100 continue

 write(*,*) K
 end
```

يعود أصل لغات البرمجة الإجرائية جميعها إلى لغة FORTRAN والتي جرى تسويقها كلغة برمجة لأول مرة عام 1958. وقد تخصصت اللغة في الحساب العلمي.

تعتمد لغات البرمجة الإجرائية على تقسيم العمل الذي ينفذه البرنامج إلى مجموعة من الأعمال/التعليمات المترابطة ببعضها البعض التي ندعوها إجرائيات. أما التعليمات على الأساسية فهي عملية الإسناد التي تعبر عن نقل قيمة إلى الذاكرة، وتعليمات التحكم بمسار البرنامج (التعليمة الشرطية if، والحلقة while أو goto).

إذ يمكن على سبيل المثال تقسيم برنامج يعبر عن آلة حاسبة بسيطة، إلى مجموعة إجرائيات تمثل كل منها عملية حسابية محددة (جمع، طرح، ضرب، قسمة). وخلال تنفيذ العملية الحسابية، يجري إسناد القيم المطلوب تنفيذ العملية عليها إلى رموز ندعوها متحولات وتُعبِّر عن أماكن في الذاكرة يجري تخزين القيم فيها وتكون لها أنماط محددة تُعبِّر عن حجم الذاكرة المحجوزة لهذه القيم (نمط عدد صحيح، أو نمط عدد حقيقي، أو سلسلة محارف، ... الخ)، ولتكرار تنفيذ مجموعة من التعليمات يمكن استخدام الحلقة معتمدين على تعليمة if و goto كما في المثال السابق. يقوم المثال بحساب مجموع عشرة أعداد يدخلها المستخدم.

11. اللغات البرمجية عالية المستوى: اللغات الإجرائية (2)

تتلخص المكونات الأساسية لمعظم اللغات الإجرائية بما يلي:

- التعليمات
- أنماط المعطيات والمعرّفات
- العمليات
- الدخل والخرج
- التوابع والإجرائيات

التعليمات:

للغات البرمجية معجم محدد من الكلمات والتعليمات الخاصة، كلمات مثل Main أو Program وتعليمات مثل if, while , for….

أنماط المعطيات:

تعبر أنماط المعطيات عن أنواع المعطيات التي نتعامل معها (عددية، حرفية، منطقية...) ويرتبط النمط بحجم محدد من خانات الذاكرة، فنمط/أنماط الأعداد الصحيحة Integer على سبيل المثال، تأخذ 16 بت أو 32 بت. أي هذه هي مساحة الذاكرة المخصصة لتخزين عدد صحيح. ندعو الأنماط الأساسية (أعداد صحيحة، أعداد حقيقية، محارف، قيم منطقية) بالأنماط البسيطة، في حين ندعو المصفوفات/الجداول وسلاسل المحارف بالأنماط المُركبة.

العمليات:

توجد، بالإضافة إلى التعليمات، رموز أخرى تدعى بالعمليات يتم استخدامها للإشارة إما إلى عملية حسابية +,−,*,/، أو إلى علاقة منطقية and, or, not.

الدخل والخرج:

يتم تنفيذ عملية الدخل باستخدام تعليمات محددة مثل INPUT , READ، كما يجري تنفيذ عملية الخرج باستخدام تعليمات محددة مثل WRITE أو PRINT.

يكون الوسيط الافتراضي المُستَخدَم في إدخال المعطيات هو لوحة المفاتيح، إلا إذا قام المُبرمج بتعريف وسيط آخر. أما الوسيط الافتراضي المُستَخدَم في إخراج المعطيات، فغالباً ما يكون شاشة الحاسوب ما لم يقم المبرمج بتعريف وسيط آخر.

الإجرائيات والمكتبات:

تتكون الإجرائية من سلسلة من التعليمات التي تُعدّ جزءاً من البرنامج، لكنها تكون مستقلة عن السلسلة الرئيسية لتعليمات البرنامج التي يجري تنفيذها. لا تشكل الإجرائية بحد ذاتها برنامجاً مستقلاً، ويجري استدعاؤها بوساطة البرنامج الرئيسي حين الحاجة لها فقط. تمتلك كل لغة برمجة مجموعة من الإجرائيات المُعرَّفة مُسبقاً ضمن لوائح جاهزة ندعوها مكتبات برمجية.

12. اللغات البرمجية عالية المستوى: اللغات الوظيفية

تعتمد العمليات في اللغات الوظيفية على توابع رياضية ومنطقية وعلى وجود قاموس من التوابع المُعرَّفة مُسبقاً وعلى آلية لبناء توابع جديدة من قبل المُبرمج.

إذ تقوم لغة LISP (List Processing) مثلاً وهي إحدى اللغات الوظيفية، بالتعامل مع كافة عناصر البرنامج على أنها جزء من سلسلة وتوفر التوابع اللازمة لمعالجة هذه السلاسل ومسحها.

فعلى سبيل المثال يجري التعبير عن عملية على عددين صحيحين بالشكل (op 2 3) حيث يجري التعامل مع التعبير السابق على أنه سلسلة من 3 عناصر، ويجري تنميط الرقمين 2 و3 واعتبارهما عددين صحيحين مباشرةً، ويجري التعامل مع أسم التابع op على أنه رمز خاص يمكن تعريف نتيجة تطبيقه على عددين صحيحين في مكان آخر.

كما يمكن التعبير عن عملية مُعرّفة مُسبقاً مثل عملية الجمع على عددين صحيحين مثل 2 و3 بالشكل (+ 23).

13. اللغات البرمجية عالية المستوى: اللغات المنطقية

جرى اشتقاق اللغات المنطقية وعلى رأسها لغة PROLOG (PROgraming in LOGic) من مفاهيم الذكاء الصنعي وتقنياته.
تجبر اللغة المنطقية المُبرمج على التفكير بأسلوب المنطق الذي يعتمد على الانطلاق من مجموعة من المقدمات للوصول إلى مجموعة من النتائج التي يمكن أن تُصبح بدورها مقدمات تُغني المُقدمات المُعرَّفة مُسبقاً.

اعتماداً على هذا المبدأ يمكن باستخدام لغة منطقية تعريف المُقدمتين: (كل إنسان فان)، و(سُقراط إنسان)، ويمكن اعتماداً على برنامج معتمد على الاستدلال المنطقي خاص بالتنفيذ، استخلاص النتيجة وهي (سقراط فان)، بحيث يمكن إضافة النتيجة إلى المقدمات لإغنائها.

ندعو البرنامج الذي يستخلص النتائج من المقدمات بمحرك استدلال.

14. اللغات البرمجية عالية المستوى: اللغات الغرضية التوجه

تعتمد البرمجة الغرضية التوجه على أساس بناء النظام البرمجي على شكل مجموعة من الأغراض التي تتواصل فيما بينها من خلال رسائل اعتماداً على توابع وإجرائيات مرتبطة بالأغراض ندعوها الطرائق.

يكافئ مفهوم الغرض في التصميم الغرضي التوجه مفهوم المتحول في اللغة الإجرائية العادية، في حين يلعب مفهوم الصف في اللغة الغرضية التوجه، دور النمط في اللغة الإجرائية (غالباً يكافئ نمط مُركَّب مثل التسجيلة).

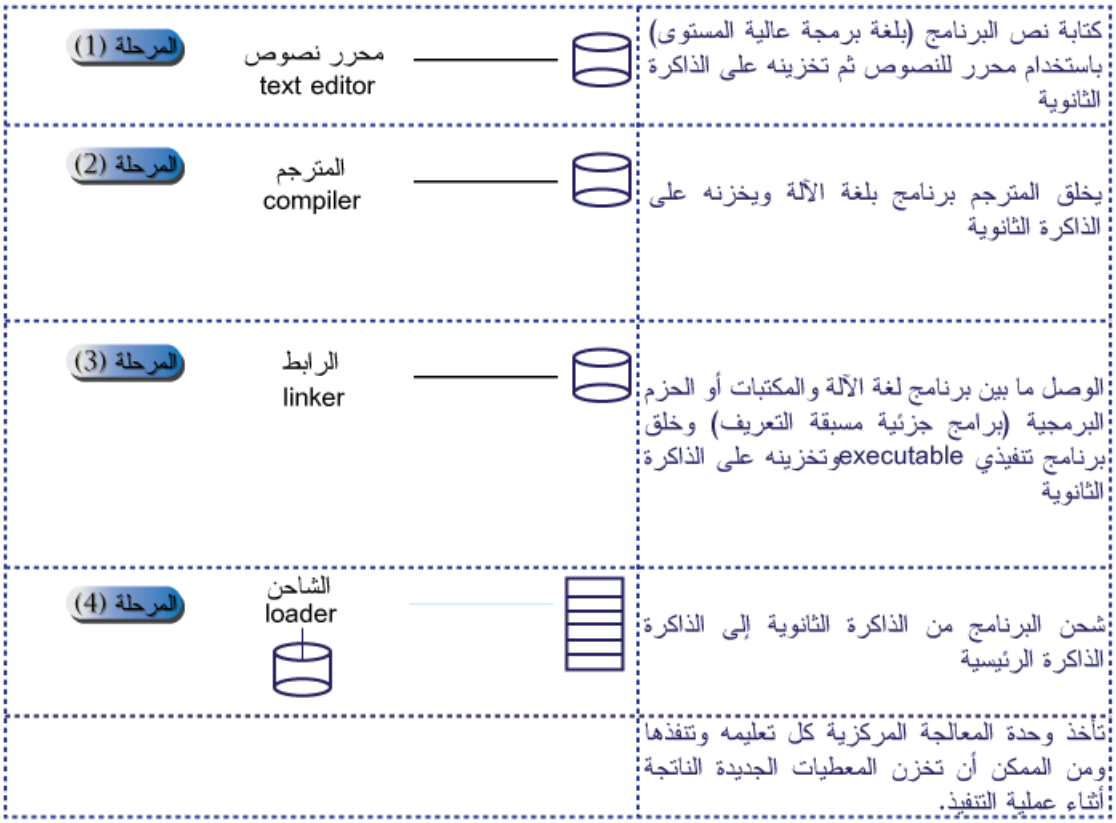
تُعتبر لغات مثل C++، Java، C# من أشهر اللغات الغرضية التوجه.

15. المترجمات

يدعى البرنامج الذي يقوم المبرمج بكتابته بإحدى اللغات البرمجية، بإسم البرنامج المصدري أو البرنامج الأصلي. ولكي يتمكن الحاسوب من تنفيذ البرنامج، ينبغي على المبرمج أن يقوم بترجمة البرنامج إلى لغة الآلة وبناء برنامج تنفيذي مكافئ للبرنامج المصدري.

تجري عملية الترجمة بوساطة مُترجِم خاص بلغة البرمجة المُستخدمة لكتابة البرنامج وخاص بنظام التشغيل الذي يعمل عليه المُبرمج، حيث يقوم المترجم بتحويل البرنامج الأصلي إلى برنامج تنفيذي.

يجري الإعلان عن الأخطاء التي يرتكبها المُبرمج عند كتابته لبرنامجه أثناء الترجمة. كما ينبغي على المترجم أن يتمكن من الدخول إلى مكتبة الإجرائيات الجاهزة التي تتضمن العديد من البرامج والإجرائيات اللازمة لتنفيذ العمليات الحسابية، وعمليات الدخل والخرج، وغيرها. وحيثما أشار البرنامج المصدري لإحدى هذه الإجرائيات، أو احتاج لتنفيذ عملية محددة، يقوم المترجم بالتأكد من إضافة الإجرائية المكتوبة بلغة الآلة إلى البرنامج التنفيذي.



16. أسئلة

أولاً– حدد اللغات الإجرائية من بين اللغات التالية:

- C++
- Pascal
- COBOL
- Java
- C#

ثانياً– حدد اللغات الغرضية التوجه من بين اللغات التالية:

- C++
- Pascal
- PROLOG
- Java
- LISP

ثالثاً– حدد العنصر الغريب من بين العناصر التالية:

- متحول
- غرض
- نمط
- إجرائية
- مكتبة إجرائيات

رابعاً– عرّف المُترجم، وحاول باستخدام الإنترنت البحث عن تعريف لمعنى كلمة مُفسِّر (Interpreter)، وحدد الفرق بين المُترجم والمُفسر وأعط مثالاً عن بعض المُفسرات.

خامساً– حدد التسلسل الزمني التاريخي لظهور اللغات التالية من الأقدم إلى الأحدث:

- PASCAL
- FORTRAN
- C
- JAVA
- COBOL
- C#

17. التطوير المنهجي للبرمجيات

يمكننا النظر إلى فعل البرمجة على أنه محاولة تحويل الفعل الإنساني إلى فعل آلي تنفذه الآلة.

جرى تطوير مجال واسع من تقنيات التحليل والتصميم البرمجية التي سعت إلى تمكين المبرمجين من التخطيط للكيفية التي ستعمل بها برامجهم قبل البدء بالبرمجة الفعلية.

تَتَّبع عملية تطوير نظامٍ برمجي في عصرنا الحالي منهجية واضحة تعتمد بدايةً على المعرفة والخبرة التي يمتلكها الإنسان عن المشكلة التي سنحلها باستخدام النظام البرمجي، وتسعى في غايتها إلى تحديد الأفعال الدقيقة التي يتوجب على هذا النظام تنفيذها حتى يقدم لنا حلاً للمشكلة المطروحة. لذا يمكننا النظر إلى فعل البرمجة على أنه محاولة تحويل الفعل الإنساني إلى فعل آلي تنفذه الآلة.

تاريخياً، كانت عملية تطوير البرمجيات تعتمد في بداياتها على فعالية المُبرمج وحَدسه التنظيمي الذي كان يساعده في وضع تصور واضح للمشكلة، وفي وضع الخطوات الدقيقة لبناء حلٍّ منهجيٍّ لها. إلا أن هذه الأفعال التي كانت تعتمد على الحَدس والتنظيم الشخصي مالبثت أن تحولت إلى آليات منهجية محددة وجرى تطوير مجال واسع من تقنيات التحليل والتصميم البرمجية التي سعت إلى تمكين المبرمجين من التخطيط للكيفية التي ستعمل بها برامجهم قبل البدء بالبرمجة الفعلية.

عموماً، تمر عملية تطوير نظام برمجي بعدة مراحل أهمها:

1. فهم احتياجات المستخدم بدقة ووضوح؛
2. كتابة وصف النظام البرمجي المطلوب (أي توصيفه). حيث يجري وضع التوصيف في مرحلة تحليل النظام ويتطلب تعاوناً وثيقاً بين محللي النظام من جهة ومستخدميه من جهة أخرى. يتضمن التوصيف شرحاً لكافة عمليات المعالجة التي ينفذها النظام بما فيها:
   - تعريف الدخل
   - تعريف الخرج
   - الوصف التفصيلي للملفات التي يحتاجها النظام، وبنيتها، والأدوات، والوسائط التي ستُستخدم لتخزينها
   - الوصف التفصيلي للتقارير، والجداول، والمخططات التي سيجري وضعها
3. استخدام أدوات التصميم والتحليل، بما في ذلك المخططات التدفقية وغيرها والتي تبين تدفق المعطيات والعلاقات المتبادلة في البرنامج. يجب التنبه خلال هذه المرحلة إلى بعض الأمور الهامة التي ينبغي أخذها بعين الاعتبار:
   - شكل واجهة المستخدم.
   - نسق ملفات المعطيات.

- إمكانية تقسيم البرنامج إلى إجرائيات وواحدات، وإمكانية توزيع العمل على أعضاء فريق البرمجة؛

4. كتابة تعليمات وإجرائيات النظام.
5. اختبار جميع مكونات وواجهات النظام، قبل وضعه في حيز التنفيذ الكامل بهدف التحقق من نجاحه وعمله بالشكل المطلوب.
6. إنشاء تطبيق خاص بإعداد وتنصيب النظام، يتضمن الملفات التنفيذية وملحقاتها.

18. استراتيجيات وضع الحلول البرمجية

تتضمن استراتيجية وضع حلّ برمجي:

- التعرف على المشكلة وتأطيرها؛
- وضع تصميم للحلّ المُقتَرح؛
- تنفيذ مجموعة من الاختبارات على الحلّ؛
- التوثيق.

التعرف على المشكلة وتأطيرها:

تتطلب هذه المرحلة فهماً كاملاً للمشكلة، بحيث يمكن تعريف الافتراضات التي يمكن استخدامها والافتراضات غير الممكنة، وذلك بهدف اختبار الحل على النحو الملائم. فعلى سبيل المثال، عند كتابتنا لبرنامج حساب المتوسط الحسابي لمجموعة من الأرقام تبدو المشكلة واضحة نسبياً حين وضع بعض الافتراضات على الأرقام التي ينبغي إدخالها مثل:

- تحديد نمط الأرقام (صحيحة، حقيقية)
- ضرورة إظهار رسالة خطأ في حال أدخل المُستخدم محرفاً آخر (حرف ما)
- تحديد نوع الأرقام الصحيحة (سالبة، موجبة أم مختلطة)

لذا، لا بد من القيام بتحليل شامل لفهم المشكلة تماماً بحيث يمكن صياغة تعريف كامل بالفرضيات عند انتهاء التحليل. وتتضمن الفرضيات، بالإضافة للحالات التي ذكرناها في مثالنا السابق:

- اللغة البرمجية التي ينبغي استخدامها؛
- كمية المعطيات التي يتوجب معالجتها.

وضع تصميم للحلّ المُقترح:

نحتاج بعد تحديد متطلبات البرنامج وتأطير المشكلة التي يعالجها، إلى تحديد خطوطه الرئيسية اللازمة لتنفيذ وتحقيق المتطلبات. لذا يجري استخدام أدوات ومنهجيات متعددة كالمخططات التدفقية ولغة الخوارزميات، وأساليب التحليل من القمة إلى القاعدة لتوصيف خطوط الحلّ.

ويعتبر أسلوب التصميم من القمة إلى القاعدة أسلوباً منهجياً لتحليل المشكلات باستخدام مبدأ الوحدات. ومن المتعارف عليه هو أن وضع تصميم باستخدام أسلوب التصميم من القمة إلى القاعدة، ومبادئ البرمجة المهيكلة يعطي برامجاً واضحة، وصحيحة، وسهلة الاختبار، والصيانة.

تنفيذ مجموعة من الاختبارات على الحلّ:

ويشمل استخدام معطيات الاختبار للتحقق يدوياً من تعليمات البرنامج، بحيث يمكن مقارنة النتائج المتوقعة منها مع النتائج الفعلية التي يقوم البرنامج بتنفيذها.

التوثيق:

يجب أن يتضمن توثيق البرنامج:

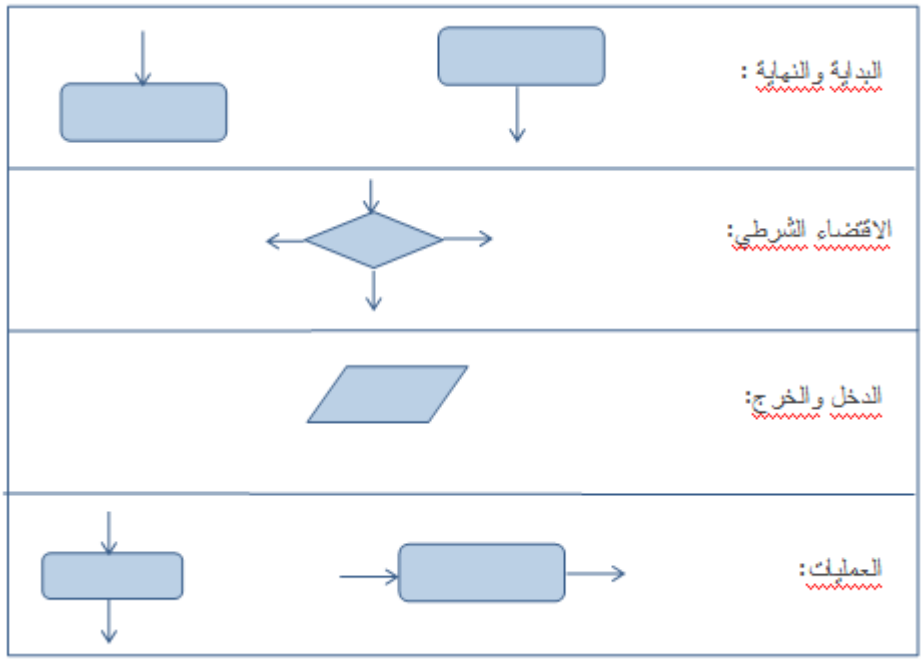
- توصيف النظام وتوصيف الدخل والخرج، والخطوط العريضة لأقسامه
- تصميم البرنامج، بما في ذلك بنى المعطيات والخوارزميات معبراً عنها بمخططات أو بلغة قريبة من لغات البرمجة
- البرنامج النهائي بالتفصيل، ويتضمن خطة الاختبارات، وسجلات الاختبار

ويعد التوثيق أمراً أساسياً للأسباب التالية:

- لإعطاء المستخدم فكرة عن كيفية إعداد معطيات البرنامج وطريقة تفسير الخرج
- لتسهيل عملية صيانة البرنامج
- لتسهيل التعديلات المستقبلية التي تهدف إلى تطوير البرنامج

19. المخططات التدفقية

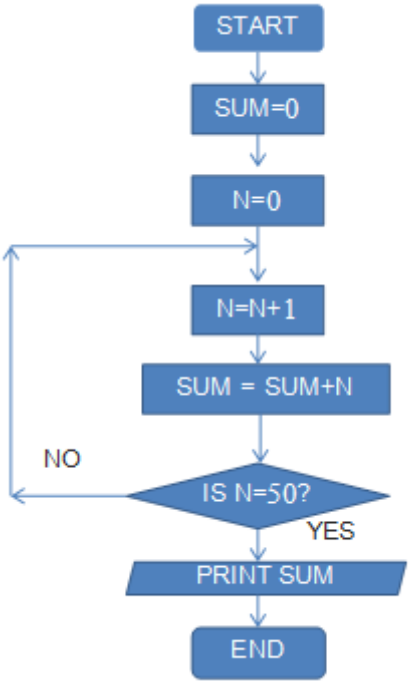
تُعتبَر المخططات التدفقية إحدى أدوات التصميم المرئي للأنظمة البرمجية وخصوصاً الصغير منها. وتُستَخدِم المخططات التدفقية رموزاً خاصة بها نستعرضها فيما يلي:



مثال1:

صمم المخطط التدفقي الذي يمثل برنامجاً يساعد في حساب مجموع الأعداد من 1 إلى 50.

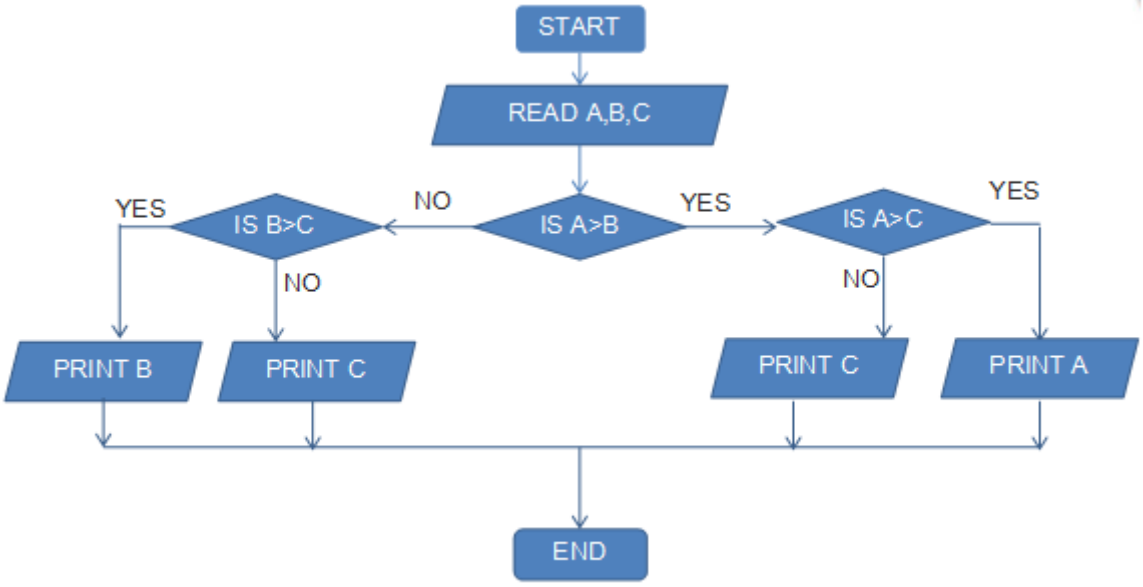
الحلّ:



مثال2:

صمم المخطط التدفقي الذي يمثل برنامجاً يساعد في إيجاد العدد الأكبر من بين ثلاثة أعداد A, B, C يُدخلها المُستخدِم.

الحلّ:

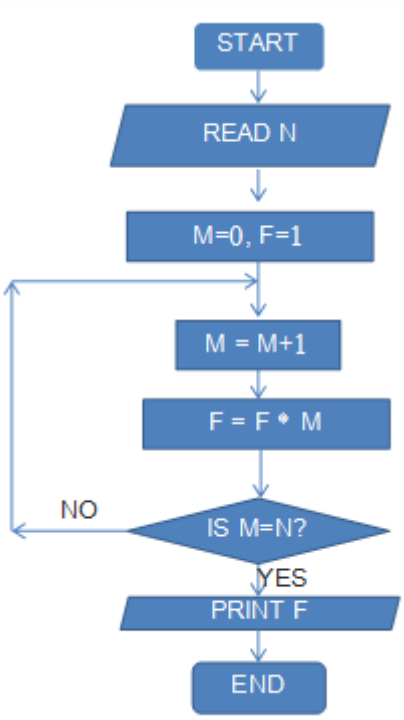


مثال3:

صمم المخطط التدفقي الذي يمثل برنامجاً يساعد في حساب N! (N عاملي) حيث N هو عدد يُدخله المُستخدِم.

(N! = N * (N−1) * (N−2) * (N−3) * … * 3 * 2 * 1)

الحلّ:



20. الخوارزميات

قبل كتابة أي برنامج لحل مسألة ما، يجب أن يتوفر لدينا فهمٌ شاملٌ للمسألة المطروحة. وهذا يتضمن تحديد وتوصيف المعطيات التي نعتمد عليها أو ننطلق منها، والنتائج التي نريد الوصول إليها.
قبل كتابة البرنامج نتبع أسلوباً منهجياً للحل، ونعبر عن هذا الحل بطريقة مؤطرة مهيكلة لا لبس فيها كافية لتنقلنا فيما بعد إلى لغة برمجة معينة دون عناء كبير.

الخوارزمية:

يتم حل أي مسألة برمجية من خلال تنفيذ سلسلة من الأفعال وفق ترتيب معين, إنها خطة الحل، ونطلق عليها تسمية الخوارزمية Algorithm، وتتضمن خطوات حل مسألة عبر تحديد:

1. الأفعال الواجب تنفيذها.
2. الترتيب الواجب إتباعه من أجل تنفيذ الأفعال السابقة.

يجب أن تكون الأفعال وترتيب التنفيذ موصفاً على وجهٍ لا يدعو إلى اللبس أو التأويل.

خوارزميات عامة (إنجاز عمل ما)

- مثال 1: خوارزمية تشغيل برنامج حاسوبي:

1. اضغط على زر التشغيل.
2. انتظر ظهور شاشة الاستقبال.
3. إذا كان من الضروري أن تُعرِّف عن نفسك: أدخل إسم حسابك وكلمة مرورك.
4. ابحث عن أيقونة البرنامج الذي تريد تشغيله وانقر عليها نقرتين بالفأرة.

(لاحظ هنا أن الأفعال قد تكون ملتبسة حسب الحالات: شاشة الاستقبال قد تكون الشاشة البيانية : أيقونات ونوافذ....، وقد تكون الشاشة السوداء : نصية فقط، وبالتالي خوارزميتنا لا تغطي جميع الحالات!! ولكنها تنفذ المطلوب أي تشغيل برنامج حاسوبي، في الحالة الأكثر شيوعاً)

- مثال 2: خوارزمية تحضير بيضة مقلية:

1. ضع الوعاء على النار.
2. أضف مقدار نصف ملعقة صغيرة من الزبدة.
3. انتظر ذوبان الزبدة.
4. اكسر البيضة وصع محتواها ضمن الوعاء.
5. انتظر حتى تنضج البيضة.

(لاحظ أن ترتيب الخطوات يغير في إنجاز العمل المطلوب من الخوارزمية وقد يعطي نتائج خاطئة، فلو وضعنا الخطوة 4 قبل الخطوة 2؛ أي وضع البيض قبل الزبدة، لما حصلنا على "العجة" اللذيذة!!)

خوارزميات حاسوبية

هنا يكون التعبير عن الخوارزمية بخطوات/أفعال محددة لا لبس فيه وقابلة للانتقال لبرنامج للتنفيذ من قبل الحاسوب.

مثال: خوارزمية تحديد العدد الأكبر من مجموعة أعداد:

التوصيف الدقيق: مجموعة الأعداد يدخلها المستخدم وتُحدَّد نهاية المجموعة بقيمة خاصة ولتكن 999.

تحديد متحولات المسألة: (كما في المسألة الحسابية الرياضية حتى البسيطة)

K للقيمة التي يدخله المستخم KMAX لقيمة العدد الأكبر

خطوات (تعليمات) الخوارزمية:

1. أدخل العدد الأول إلى المتحول KMax
2. أدخل العدد التالي إلى المتحول K
3. إذا (K تساوي 999) اذهب إلى الخطوة **6**
4. إذا K أكبر من KMax أسند K إلى KMAX
5. اذهب إلى الخطوة **2**
6. اكتب "قيمة العدد الأكبر"، KMax

فكرة الحل هي كالتالي، تدخل العدد الأول، تعتبره الأكبر (1) إلى أن يثبت العكس: أي تكرر إدخال أعداد المجموعة (2)، تقارن العنصر المُدخَل K مع من اعتبرته الأكبر KMAX فإن كان المُدخَل أكبر تجعله يحل مكان من اعتبرته الأكبر (4)، وهكذا حتى نهاية المجموعة.

21. لغة الخوارزميات (pseudo code)

لغة الخوارزميات هي عبارة عن لغة مصطنعة بمعنى أنها لغة للتعبير عن البرامج ولكنها ليست لغة برمجة فعلية. تساعد المبرمجين على التعبير عن الخوارزميات: خطوات الحل، ومسار أو تسلسل تنفيذها.

سنقوم في هذا الفصل بعرض لغة خوارزميات تفيدنا في التعبير بشكل "طبيعي" عن خطواتنا ومسارنا لحل مسألة بواسطة الحاسوب.

تشبه لغة الخوارزميات لغة تفكيرنا في لغتنا المتداولة، فهي لغة مبسطة ومفهومة، ولكن مؤطرة ومهيكلة، ستسهل لنا فيما بعد الانتقال إلى لغة برمجية فعلية عالية المستوى.

معنى كلمة pseudo code

تستخدم كلمة code في الإنكليزية في سياق علوم الحاسوب للدلالة على الترميز للحاسوب أو لكل ما هو تعليمات لغات برمجة، ومن هنا استخدام المصطلح pseudo code (شبه أو قريب الترميز) للدلالة أنها لغة قريبة أو شبيهة بلغات البرمجة.

لا يمكن تنفيذ البرامج المكتوبة بلغة الخوارزميات على الحاسب، لكنها تساعد المبرمج كثيراً على التفكير ببرنامجه قبل محاولة كتابته بأية لغة برمجة.

22. التعليمات الأساسية للغة الخوارزميات (pseudo code)

يمكن التعبير عن خطوات ومسار حل مسألة بواسطة الحاسوب أو توصيف خوارزمية بواسطة التعليمات الخمسة الأساسية التالية:

1. تعليمة القراءة أو إدخال المعطيات.
2. تعليمة الكتابة أو إظهار النتائج.
3. تعليمة الإسناد أو وضع قيم في متحول .
4. التعليمة الشرطية.
5. التعليمة التكرارية.

لاحظ أن هذه التعليمات هي تجريد ونظرة عالية المستوى لما يمكن أن يقوم به الحاسوب، فالتعليمات 1- و2- هي للدخل والخرج. والتعليمة 3- للتعامل مع المتحولات أي خانات الذاكرة. أما التعليمات 4- و5- فهي للتحكم بمسار التنفيذ، وكما عرفنا من مبادئ عمل الحاسوب أن تنفيذ التعليمات هو ضمنياً تسلسلي، فإذا أردنا تغيير تسلسل التنفيذ فهاتان التعليميتان هما الوسيلة.

سنشرح كل من هذه التعليمات الأساسية، لنستوعب جيداً عما تعبر فعلياً بشكل دقيق.
وسنستخدم كلمة متحول وخانة ذاكرة لنفس الدلالة. وفي أمثلتنا الأولى سيكون معظم استخدامنا لمتحولات عددية صحيحة (integer).

(يعرَّف المتحول variable بثلاثية: اسم تعريف identifier، نمط type، خانة ذاكرة يشغلها memory allocation).

ولنركز معاً، أليس رائعاً أن ندرك أنه يكفينا 5 تعليمات أساسية للتعبير عن أي حل لمسألة بواسطة الحاسوب.

23. تعليمة القراءة read

هي إعطاء قيمة من الدخل (من لوحة المفاتيح) لوضعها في خانة ذاكرة.

نعطي شكلاً نظامياً لهذه التعليمة:

```
اقرأ متحول أو متحولات
```

وهي عبارة عن أمر يقول خذ القيمة (القيم) الموجودة في الدخل وضعها في خانة (خانات) ذاكرة المتحول (المتحولات).

أمثلة:

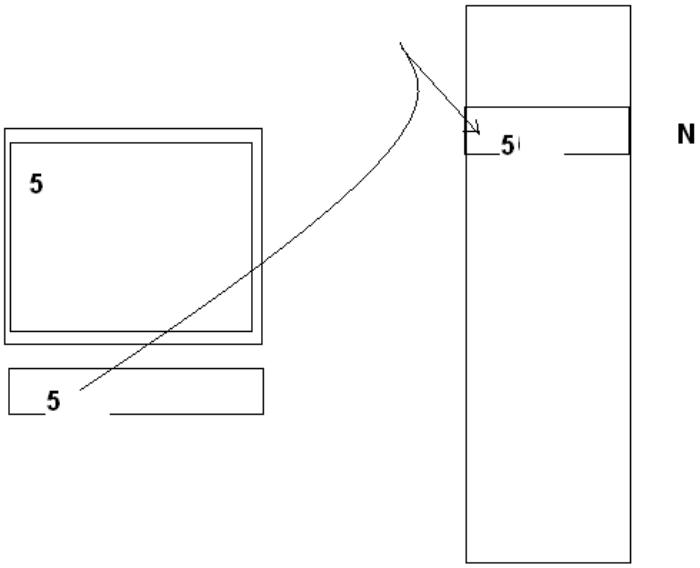
```
اقرأ N
```

ضع القيمة التي تُعطى من الدخل في خانة الذاكرة N

```
اقرأ a, b, c
```

ضع القيم التي تُعطى من الدخل في خانات الذاكرة a, b, c

يجب التنويه أن تنفيذ الحاسوب لتعليمة القراءة يكون بانتظار المستخدم حتى يدخل قيم ويضغط مفتاح الإدخال Enter.



رسم توضيحي لعملية قراءة القيمة 5.

24. تعليمة الكتابة write

و هي إظهار قيم معينة على وحدة الخرج (الشاشة)
نعطي شكلاً نظامياً لهذه التعليمة:

اكتب صيغة أو تعبير

والصيغة أو التعبير (formula , expression): هي تركيب من متحولات وثوابت وتوابع وعمليات، وفي الحالة البسيطة يكون التعبير متحولاً فقط أو ثابتاً فقط.
إذن تعليمة الكتابة هي أمر يقول أظهر قيمة التعبير على وحدة الخرج. وكما في تعليمة الدخل يمكن أن نطلب من تعليمة الكتابة اظهار أكثر من قيمة (عدة صيغ).

أمثلة:

- متحولات

اكتب N

أظهر القيمة الموجودة في خانة الذاكرة N على الخرج.

- ثوابت

اكتب 20

ستظهر القيمة 20 على الخرج.

- توابع

اكتب Cos(0)

ستظهر القيمة 1 على الخرج.

- عمليات

اكتب N*2

احسب قيمة التعبير N*2 وأظهرها على الخرج.
فإن كانت N تساوي 6 فسيظهر على الخرج القيمة 12.
نضيف إمكانية هامة لتعليمة الكتابة هي كتابة نص أو سلسلة حرفية.

اكتب "Programming is going well"

يظهر النص كما هو تماماً على الخرج Programming is going well
(الحروف الكبيرة والصغيرة والفراغات تظهر كما هي محددة بين علامتي التنصيص)

وكي نختم عن حالات تعليمة الكتابة: يمكن أن ندع تعليمة الكتابة تظهر حميع الحالات السابقة بنفس التعليمة، فقط أفصل بين الأشياء التي أريد كتابتها بفاصلة،

```
اكتب a, b, c
```

أظهر القيم الموجودة في خانات الذاكرة a, b, c

```
اكتب 2*N , " The result is "
```

وفي حالة N تملك القيمة 6.
سُتظهر تعليمة الكتابة ما يلي:
The result is 12

25. تعليمة الإسناد

وهي إسناد قيمة صيغة لمتحول (لخانة ذاكرة)

نعطي شكلاً نظامياً لهذه التعليمة:

صيغة أو تعبير ← متحول

أمثلة:

مثال 1:

لدينا في خانتي الذاكرة t, i القيم الخزنة (t = 10, i=25).

عند تنفيذ تعليمة الإسناد التالية:

```
i ← t
```

تصبح في الخانة i القيمة 10أيضاً. أو نقول أن المتحول t يساوي 10.

وبتنفيذ الإسناد:

```
i ← i + 4*t
```

تصبح قيمة i تساوي 50.

مثال 2:

حساب قيمة التابع: $y = x^2 - 16$ في حالة x=4، نكتب:

```
x ← 4
y ← x^2 - 16
```

ما هي قيمة المتحول y بهذا الإسناد؟

مثال 3:

اكتب مجموعة العبارات أو التعليمات التي تسمح بحساب التابع $y=x^3-9$ وإظهار النتيجة من أجل قيم x يجري إدخالها.

```
إقرأ x
y ← x^2-9
اكتب y
```

وإذا أردنا إظهار النتيجة على نحوٍ معبر أكثر، نستبدل تعليمة الكتابة، بالتعليمة التالية:

```
اكتب "x=", x ," y=", y
```

ما هو شكل الإظهار من أجل قيمة الدخل 3؟

26. التعليمة الشرطية

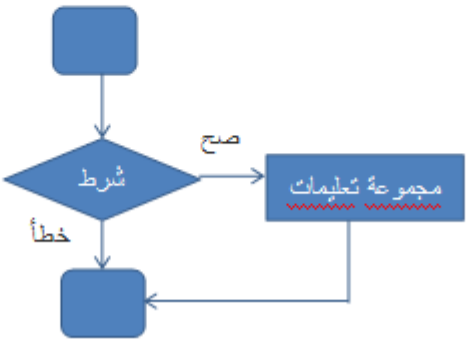
وفيها تتحدد التعليمات التالية في التنفيذ بناءً على اختبار شرط.

وترد بأحد الشكلين التاليين:

1. التعليمة الشرطية البسيطة أو التنفيذ بشرط if
2. التعليمة الشرطية الاختيارية أو التنفيذ باختيار بين مسارين وفقاً للشرط if else

التعليمة الشرطية البسيطة if

نعطي شكلاً نظامياً لهذه التعليمة:



وفي هذه التعليمة يجري اختبار الشرط فإن محققاً (صحيحاً) تُنفَّذ مجموعة التعليمات. (بالطبع يمكن أن تقتصر مجموعة التعليمات على تعليمة واحدة كما في أمثلتنا المبسطة).

وإن لم يكن الشرط محققاً ما الذي يجري برأيك؟

الجواب: بالطبع لن تنفَّذ مجموعة التعليمات، وسينتقل التنفيذ إلى التعليمة التالية، بعد كامل التعليمة الشرطية. ألم نقل أن تنفيذ التعليمات هو ضمنياً تسلسلي.

ما معنى شرط؟

الشرط هو المعنى المألوف لديك، أي أمر يمكن البت بشأنه: محقق أو غير محقق (نعم أو لا) (صح true أو خطأ false)

ويمكن أن نعطي الشرط condition التعريف التالي:

أي صيغة تُقيَّم بصح أو خطأ، وهي غالباً تتضمن المقارنة (التراجح أو المساواة) : أكبر، أصغر، يساوي.

كما يمكن أن تتضمن أدوات التركيب (العطف) المنطقي: (و: and، أو : or، لا: not).

أمثلة:

مثال 1:

```
إذا ( N > 0 ) نفِّذ
بداية
    a ← s/N
نهاية
```

نريد من هذه التعليمة هنا أن نقسم على قيمة المتحول N بشرط أن تكون N أكبر من الصفر.

وبتعبير آخر: إذا كانت N أكبر من الصفر، عندها نفِّذ تعليمة الإسناد للتعبير s/N إلى المتحول a.

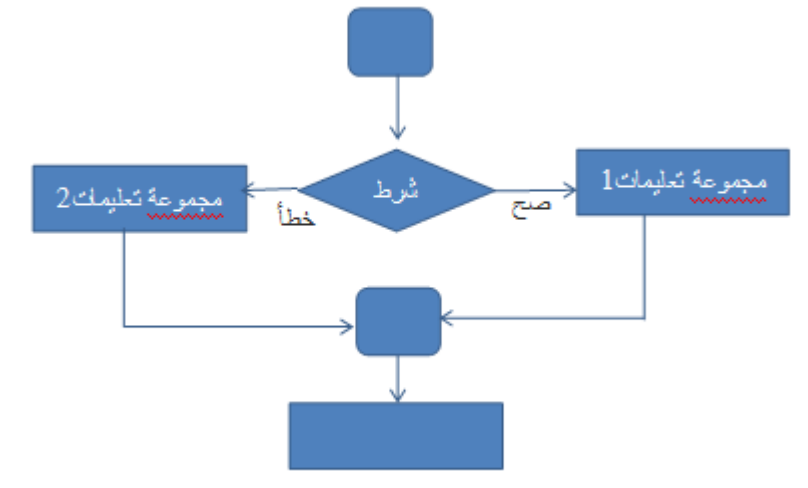
مثال 2:

نريد إدخال (قراءة) علامة مادة ونريد كتابتها فقط إذا كانت في المجال من 0 إلى 100.

ليكن x المتحول الذي نكرسه للعلامة.

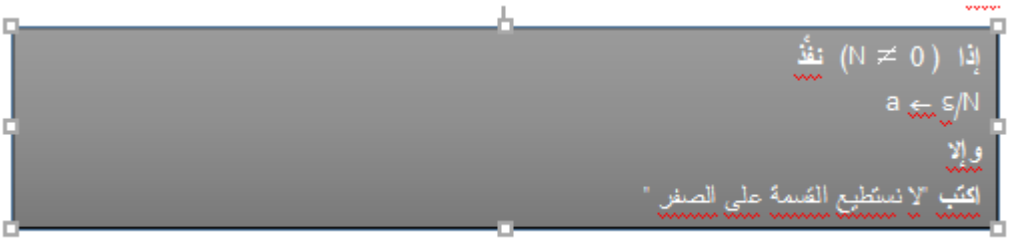
```
اقرأ x
إذا ( x ≥ 0 و x ≤100 ) نفِّذ
بداية
    اكتب x
نهاية
```

التعليمة الشرطية الاختيارية if else



في هذه التعليمة يجري الاختيار بين مساري تنفيذ: إما تنفيذ مجموعة تعليمات–1 في حال كان الشرط محققاً (صحيحاً) أو تنفيذ مجموعة تعليمات–2 في الحالة المعاكسة.

مثال 1:



نريد من هذه التعليمة هنا أن تقسم على قيمة المتحول N بشرط أن تكون N لا تساوي الصفر، وإلا نظهر عبارة تدل على الخطأ.

مثال 2:

إليك هذا المثال المنتهي البساطة، ولكنه سيعلمنا، أمور عديدة: نريد قراءة عددين، وكتابة المقارنة بينهما (أكبر، أصغر، يساوي).

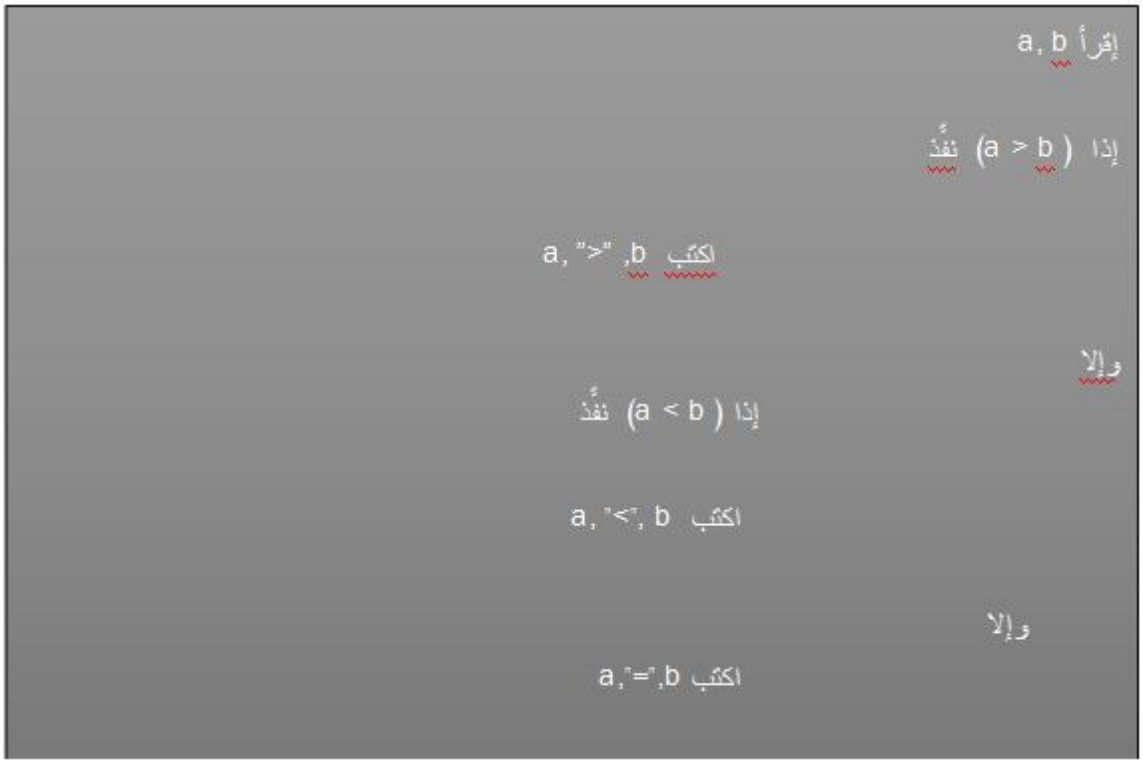
أمثلة عن التنفيذ:

دخل 3 و10 خرج 3 < 10

دخل 15 و10 خرج 15 > 10

دخل 10 و10 خرج 10=10

الخوارزمية:



إن ما كتبناه هنا هو شكل مهيكل ومؤطر لما تفكر به بشـكل طبيعي: فإما العدد الأول (a) أكبر من الثاني (b) وإلا فهو أصغر منه أو يساويه، فإذا كان أصغر كتبنا ذلك وإلا فهو حتماً يساويه.

لا حظ هنا تداخل التعليمات في مستويات أو هياكل. فالتعليمات التي تنفذ داخل التعليمة الشرطية الاختيارية هي بدورها تعليمة شرطية، وهذه بدورها يمكن أن تأخذ أي تعليمة بما فيها التعليمة الشرطية. بهذا يمكننا أن نبد الإدراك أن هذه البنى أو التعليمات الخوارزمية الأساسية يمكنها أن تجعلنا نعبر عن حلولنا (مهما كانت متشعبة) بتمييز الحالات المختلفة على نحو مهيكل ومؤطر.

تنويه هام:

تسمى التعليمـة الشـرطية أيضـاً، بتعليمـة التفريـع. لأن مسـار التحكم فـي تنفيذ البرنـامج يتفرع عندها إلـى الاختيـار بين مسارين وفقاً لشرط.

27. التعليمة التكرارية: while

تستعمل لتكرار تنفيذ مجموعة من التعليمات، يرتبط هذا التكرار بتحقق شرط، إذ يختبر الشرط فإن كان الشرط محققاً (صحيحاً) تنفذ مجموعة التعليمات ويكرر تنفيذ هذه التعليمات مادام الشرط محققاً.

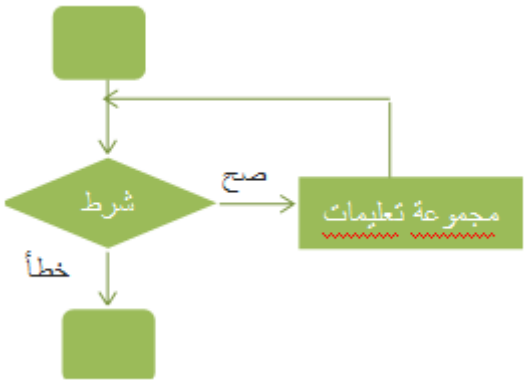
نعطي شكلاً نظامياً لهذه التعليمة:

مادام (شرط) **كرِّر**

بداية

مجموعة تعليمات

نهاية



تسمى هذه التعليمة أيضاً، من بين بنى التحكم control structure، بالحلقة التكرارية. إذ أنه في حال تحقق الشرط تنفذ مجموعة التعليمات، ليعود التنفيذ إلى اختبار الشرط من جديد، لتنفذ التعليمات من جديد مادام الشرط محققاً. وتبقى هذه الحلقة أو التكرار مستمراً في تنفيذ مجموعة التعليمات واختبار الشرط، حتى يصبح الشرط غير محقق، عندها ينتقل التنفيذ، إلى التعليمة التالية لتعليمة التكرار.

مثال 1:

كتابة جدول الضرب للعدد 7.

الفكرة الخوارزمية

هي ما تقوم به فعلياً بيدك. تكرار كتابة أسطر من الشكل:

7 x 1 = 7

7 x 2 = 14

.....

7 x 10 = 70

أي تكرار كتابة: 7 x i = 7*i حيث i تبدأ بالقيمة 1 وتتزايد لتتوقف عند القيمة 10.

خوارزمية الحل:

```
i ← 1
مادام ( i ≤ 10 ) كرر
بداية
    اكتب 7, " x", i ,"= ", 7*i
    i ← i +1
نهاية
```

في كثير من حالات التكرار نستخدم متحولاً يتزايد مع تنفيذ التكرار، مثل حالة i في مثالنا، نسمي مثل هذا المتحول عداد (يعدّ مرات التكرار)، وبما أن التكرار هو حلقة نسميه أيضاً عداد الحلقة التكرارية.

مثال 2:

المطلوب حساب مجموع الأعداد 1, 2, 3,...M
حيث M تُعطى كدخل.

الآن وقد اكتملت لدينا التعليمات الخوارزمية الأساسية، وقد ادعينا أنها كافية لنعبر عن حل أي مسألة، لنبدأ التفكير بخوارزمية الحل بهذه التعليمات.

المهم أن نبدأ بتوصيف المسألة جيداً، ويكون ذلك بتحديد المعطيات والنتائج بشكل دقيق، ويكون الحل (الخوارزمية) بمحاولة بناء الخطوات لمعالجة المعطيات وصولاً إلى النتائج.

المعطيات: M عدد صحيح موجب.
النتائج: مجموع الأعداد 1, 2, 3,...M ولنجعل لهذا المجموع متحولاً S .

الخوارزمية:
العلاقة بين S و M هي: $S=1+2+...+M$
آه، إن الحل بديهي هنا، ما هي إلا تعليمة إسناد وأحصل على المطلوب، أي:

اقرأ M
$S \leftarrow 1+2+3+...M$
اكتب S

ما رأيك!!!
بالطبع لن يكون هذا الحل مقبول خوارزمياً لأنه لن يكون مقبول برمجياً ولا حاسوبياً.
لا يمكن أن تكون تعليمة الإسناد مقبولة بهذا الشكل: أية صيغة رياضية هذه المكتوبة بـ M+...+1! هذا يمكن أن يعبر عن فكرة مفهومة لنا، ولكنه غير قابل للحساب.

ولكن نتابع نفس الفكرة بأسلوب مختلف قليلاً، نجمع إلى المتحول القيمة 1 ثم 2 ثم 3 حتى M.
وجدنا بداية الفكرة: $S \leftarrow S + i$ حيث سيكون i هو المتحول الذي يأخذ 1، 2،...حتى M. ولكن لم نتخلص بعد من هذه النقاط اللعينة ...,...,M.
لا ليست بهذه الصعوبة، إنها قيم متحول عداد يتزايد بشكل تكراري، نكرر زيادته بقيمة 1 إلى أن يصل إلى القيمة M. أي شرط التكرار ($i \leq M$).

خوارزمية الحل:

```
اقرأ M
S ← 0
i ← 1
مادام ( i ≤ M ) كرر
بداية
    S ← S + i
    i ← i +1
نهاية
اكتب S
```

لنتأكد من صحة الخوارزمية وذلك باختبارها يدوياً من أجل قيم مختلفة للمتحول M:

الدخل: M=1

شرط i ≤ M	تكرار	i	S	M
صح	0	1	0	1
خطأ	1	2	1	1

الخرج أو النتيجة: S = 1

الدخل: M=3

شرط i ≤ M	تكرار	i	S	M
صح	0	1	0	3
صح	1	2	1	3
صح	2	3	3	3
خطأ	3	4	6	3

الخرج أو النتيجة: S = 6

إن خوارزمية الحل ليست وحيدة، حتى إن كانت في فكرتها واحدة، فالدقة في القيم وترتيب التعليمات يمكن أن يعطينا حلولاً مختلفة.

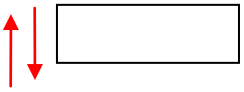
اقرأ M

S ← 1

i ← 1

مادام (i < M) **كرر**

بداية



i ← i +1

S ← S + i

نهاية

اكتب S

وقد أشرنا إلى التغييرات عن الخوارزمية السابقة.

تدريب اختبر الخوارزمية على قيم الدخل السابقة.

الجواب

الدخل: M=3

شرط i < M	تكرار	i	S	M
صح	0	1	1	3
صح	1	2	3	3
خطأ	2	3	6	3

الخرج أو النتيجة: S = 6

ملاحظة هامة جداً:

إن كان الشرط محققاً دائماً, أي كان صحيحاً ولا يتغير في متحولاته شيء، فماذا يحصل؟؟؟

الجواب: إنها أكبر الأخطاء البرمجية، ندخل بحلقة لانهائية، أي يستمر تكرار تنفيذ مجموعة التعليمات بلا توقف.

كيف ننتبه إلى مثل هذا الخطأ؟

إن لم تكن مجموعة التعليمات لا تحمل في طياتها أي تعديل على أي من المتحولات المضمنة بالشرط فهناك حتماً حلقة لا نهائية.

مثال:

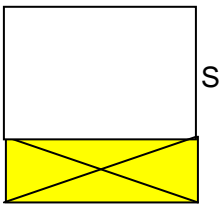
اقرأ M

$S \leftarrow 0$

$i \leftarrow 1$

مادام ($i \leq M$) **كرر**

بداية



نهاية

اكتب S

لاحظ ما الذي يحصل لو نسينا تعليمة تغيير المتحول i:

الدخل: M=1

شرط $i \leq M$	تكرار	i	S	M
صح	0	1	0	1
صح	1	1	1	1
صح	2	1	2	1
صح	3	1	3	1
صح		1	…	1
صح		1		1
صح		1		1
صح		1	لانهاية (أعلى قيمة ممكنة حاسوبياً)	1

هذا ما نسميه حلقة تكرار لانهائية.

28. منهجية كتابة نظام برمجي

بالإمكان التوصل إلى حل لمعظم المشاكل بعد معرفتها وتفهمها. ومن المناهج الشائعة في ذلك منهج التصميم من القمة إلى القاعدة الذي يُعرِّف عملية معالجة عامة تساعد في الإنطلاق من مشكلة كبيرة معقدة إلى مجموعة من المشكلات الأصغر، بحيث تكون قابلية حل هذه الأجزاء أسهل من حل المشكلة الأصلية دفعة واحدة. هذا وتتضمن عملية التصميم بمجملها كل مما يلي:

- وضع توصيف مفصل يحدد دخل وخرج النظام البرمجي والفرضيات الأساسية الخاصة به
- التعبير عن خوارزمية الحل لكل قسم من أقسام النظام البرمجي بلغة الخوارزميات pseudo-codeأو المخططات التدفقية flowcharts
- نقل التعبير عن الخوارزمية إلى لغة برمجية

ملاحظة هامة:

من السائد منذ بداية السبيعنيات التعبير بلغة الخوارزميات pseudo-code وتفضيلها على المخططات التدفقية flowcharts لسببين:

1. المخططات التدفقية توحي باستخدام تعليمة القفز goto وقد أثبتت الدراسات تأثيرها السلبي على سهولة قراءة البرامج وفهمها واختبارها وصيانتها.
2. انتشار ما نسميه البرمجة المهيكلة structured programming الموجودة في معظم لغات البرمجة المنتشرة: Pascal, Basic, C, C++, Java, C#

إن التعبير بلغة الخوارزميات، هو تعبيرنا عن الحل بكلمات مفهومة الدلالة، وهيكلية واضحة، لا تدعو للالتباس. وهي كما ذكرنا pseudo-code شبيهة بالـ code أي بالنص البرمجي في لغات البرمجة. ولكن نكتب فيها كما نفكر بلا قيود قواعدية صارمة تفرضها لغات البرمجة. هي مسودة البرمجة (بالورقة والقلم)، ولكنها مسودة مفهومة قابلة "للتبييض" بأية "لغة برمجة". لذلك اخترنا مثلاً، وخصوصي في بداية البرمجة أن نعبِّر بمفردات باللغة العربية، لغة تفكيرنا.

بعد أن تتأقلم مع لغة برمجة وتتعرف دلالة التعليمات الأساسية، يمكنك استخدام المفردات الإنكليزية إذا كان ذلك أكثر راحةً لك، ولكن ضمن نفس الشكل المؤطر والمهيكل الذي ذكرناه.

مثال: Pseudo-Code (تعبير عن خوارزمية باللغة الإنكليزية)

حل معادلة من الدرجة الثانية

read a,b,c;

$delta \leftarrow b^2 - 4 \times a \times c$

if delta<0
begin
    write “No Real Solution”
end

if delta = 0
begin
    write “One Solution”
    $x \leftarrow \frac{-b}{2 \times a}$
    write x
end

if delta >0
begin
    write “Two Solutions”
    $x1 \leftarrow \frac{-b + \sqrt{delta}}{2 \times a}$
    $x2 \leftarrow \frac{-b - \sqrt{delta}}{2 \times a}$
    write x1,x2
end

29. أمثلة كلاسيكية عامة وهامة

مثال 1:

حساب القاسم المشترك الأعظم لعددين صحيحين موجبين.

القاسم المشترك الأعظم لعددين صحيحين هو أكبر عدد صحيح يمكن نقسم عليه العددين (دون باقي طبعاً).

للعددين: 12، 16 هو 4. للعددين: 75، 45 هو 15. للعددين: 10، 3 هو 1. .....

خوارزمية إقليدس :

خوارزمية رائعة لا تحتاج إلى أية عمليات قسمة أو ضرب.

وهي كالتالي، نطرح الأصغر من الأكبر ونضعه مكانه، نكرر ذلك حتى يتساوى العددان.

لنطبق ذلك على العددين 75, 45

A	B
75	45
30	45
30	15
15	15

لنعبر عن هذه الخوارزمية بلغة الخوارزميات:

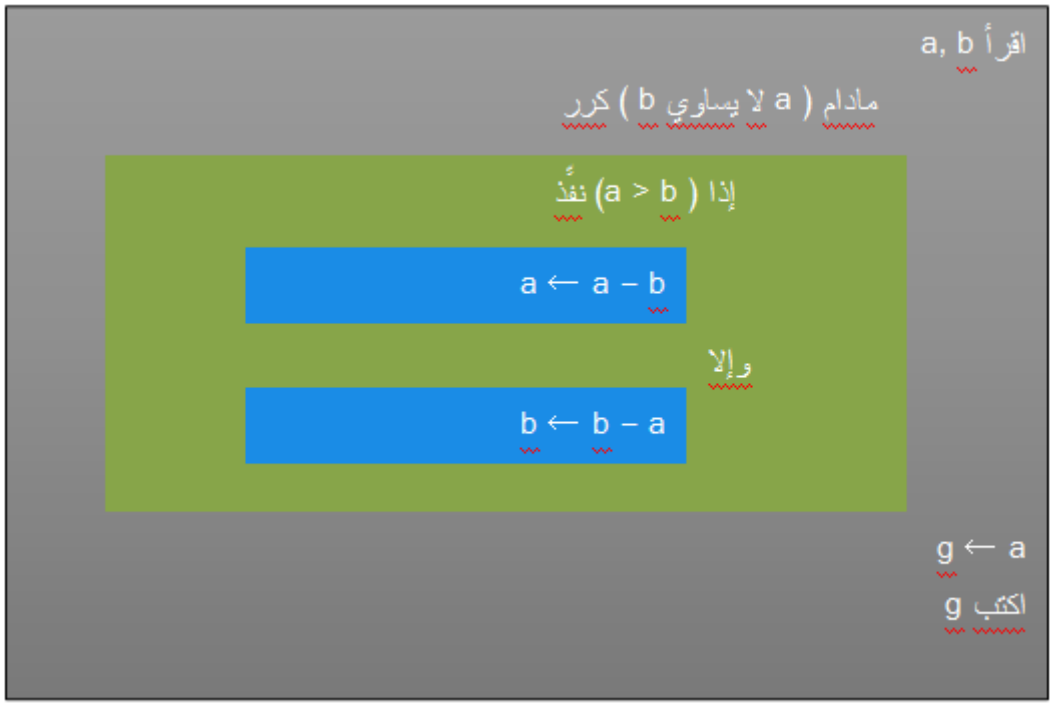
المعطيات (الدخل):

a, b عددان صحيحان موجبان.

الخرج (النتيجة):

g القاسم المشترك الأعظم.

الخوارزمية:



إنها خوارزمية رائعة وبسيطة، واستخدمنا فيها جميع التعليمات الخوارزمية الأساسية (دخل، خرج، إسناد، تعليمة شرطية، تعليمة تكرارية).

سؤال:

هل تتغير النتيجة إذا غيرنا تعليمة الإسناد g ← a بالتعليمة g ← b.

الجواب:

لا تغيير، لأن هذه التعليمة تلي تعليمة التكرار، التي لا ننتقل إلى ما بعدها إلا بعد عدم تحقق الشرط (a لا تساوي b). وبالتالي عند وصولنا إلى تعليمة الإسناد، تكون قيمة a و b متساويتان بالضرورة.

مثال 2:

حساب العدد الأكبر بين مجموعة أعداد يدخلها المستخدم.

المعطيات: مجموعة الأعداد التي يجري إدخالها
النتيجة: العدد الأكبر.

لنتأمل قليلاً، هل المسألة موصفة جيداً؟ ما المقصود بمجموعة، ما هو عدد عناصر المجموعة، أو كيف لنا أن نعرف أن المجموعة التي يجري إدخالها قد انتهت.
إذن، لتحديد المجموعة التي ندخلها، نعطي أولاً عدد العناصر، ثم العناصر. أو نقوم بإدخال العناصر ونصطلح على رمز يدل على نهاية المجموعة (عنصر غير مألوف مثل 999).

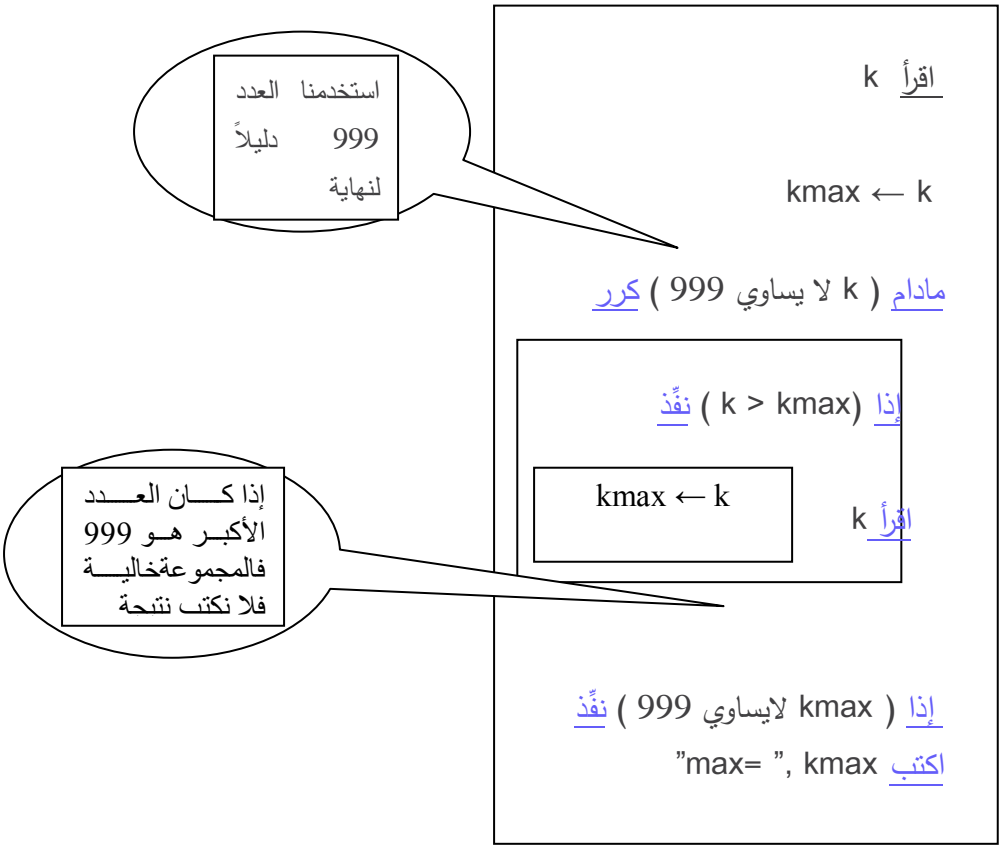
و لنفكر معاً بالخوارزمية:
أرجو ألا تبدأ التفكير بحل مسألة بواسطة الحاسوب، لا بلغة الخوارزميات، وقطعاُ في المستقبل لا تفكر بلغة برمجة معينة، بل دع تفكيرك الطبيعي الحدسي يتلمس ويتخيل المسألة وارسم وشطب على مسودتك والمهم جداً أن تكون قادراً على حل المسألة يدوياً (في الحالات البسيطة)، تذكر لن يقدم الحاسوب الحل.
نؤكد لك أن تفكيرك العادي للحل يمكن بجهد قليل صياغته بلغة خوارزمية، سيمكن فيما بعد نقلها بجهد قليل أيضاً إلى أية لغة برمجة عالية المستوى.

ففي مسألتنا هنا، تخيل أنك تريد معرفة الشخص الأطول بين مجموعة أشخاص يردون إلى غرفتك بالتتالي، وبالطبع لا يمكن لغرفتك أن تستوعب إلا عدداً محددو جداً من المجموعة، أي لا تستطيع النظر إليهم جميعاً لتقول بنظرة خاطفة هذا هو الأطول.

الحل هو كالتالي، تدخل العنصر الأول، تعتبره الأطول إلى أن يثبت العكس، أي تكرر إدخال العناصر، تقارن العنصر المدخل مع من اعتبرته الأطول فإن كان أطول منه تجعله يحل مكانه، وهكذا حتى نهاية المجموعة.
إنها الخوارزمية، وما علينا إلا التعبير عما كتبناه بخطوات محددة، وبهيكلية واضحة ودقيقة لا لبس فيها، بعد أن نحدد المتحولات المعبرة عما نعالجه.

ليكن k هو عنصر المجموعة المدخل، وkmax هو المتحول الذي نحتفظ فيه بالعنصر الأكبر.

الخوارزمية:



تدريب: اختبر الخوارزمية يدوياً في حالات مجموعات مختلفة. (لا تنس اختبار الحالات الحدية: حالة مجموعة من عنصر واحد، وحالة المجموعة الخالية حيث الدخل هنا يقتصر فقط على نهاية المجموعة أي999).

30. أسئلة

أجب بصح أو خطأ:

1. ترتبط البرامج المكتوبة بلغات عالية المستوى بالعتاد خطأ
2. تكون كتابة البرامج بلغات عالية المستوى أصعب من كتابتها بلغات ذات مستوى منخفض خطأ
3. يكون تشغيل البرامج المكتوبة بلغات عالية المستوى أبطأ من البرامج المكتوبة بلغات ذات مستوى منخفض صح
4. يكون تصحيح البرامج المكتوبة بلغات عالية المستوى أسهل من البرامج المكتوبة بلغات ذات مستوى منخفض صح
5. يتحكم نظام التشغيل بالموارد الحاسوبية صح
6. إن نظام التشغيل هو Windows خطأ
7. يعد نظام التشغيل جزءاً من العتاد خطأ
8. يمكن تحميل أكثر من برنامج في الذاكرة في آن واحد صح
9. يمكن تنفيذ وتشغيل أكثر من برنامج في آن واحد صح
10. يمكن أن تستخدم عدة برامج الطابعة في آن واحد صح

رتب ما يلي وفق التتالي الزمني الصحيح:

1. الاختبارات (5)
2. البرمجة (4)
3. كتابة توصيف البرنامج (2)
4. التوثيق (6)
5. التصميم (3)
6. فهم المشكلة (1)

31. نشاط

إن المسائل في هذا النشاط من النوع البسيط، ولا شك أنك تستطيع الحصول على الحل بسهولة بواسطة أدوات الجدولة مثل Excel، ولكن الهدف هنا تدريبك على التعبير عن الحل باستخدام التعليمات الأساسية (الخمسة) بلغة الخوارزميات.

بعد تعبيرك عن الحل، حاول أن تنفِّذ الخوارزمية يدوياً بتغيير قيم المتحولات حسب تسلسل تنفيذ التعليمات. ابدأ بالتنفيذ على عينة صغيرة من قيم الدخل، تأكد من قيمة الخرج باستخدام Excel إن أردت.

المسألة الأولى:

نريد كتابة برنامج حساب المتوسط الحسابي:

$$\text{Average} = \frac{\sum_{i=1}^{n} x_i}{n}$$

حيث تعبّر x_i عن معدلات طلاب صف من صفوف الجامعة الافتراضية. مع العلم أن عدد طلاب الصف الواحد (المُشار إليه بالمتحول n) يُعطى من الدخل، وأن المعدلات محسوبة من 100 علامة وأن المُستخدم يقوم بإدخال المعدلات عند تنفيذ البرنامج.

حدد عند توصيفك ومعالجتك للمطلوب:

1. دخل البرنامج.
2. خرج البرنامج.
3. الفرضيات الأساسية التي يجب معالجتها .
4. خوارزمية الحل بلغة الخوارزميات.

مثال عن دخل/خرج البرنامج:

2

70

60

الخرج

65

المسألة الثانية:

نريد كتابة برنامج حساب الإنحراف المعياري:

$$StandardDeviation = \frac{\sum_{i=1}^{n}(x_i - Average)}{n}$$

حيث تعبّر x_i عن معدلات طلاب صف من صفوف الجامعة الافتراضية، ويعبر Average عن المتوسط الحسابي للمعدلات. مع العلم أن عدد طلاب الصف الواحد (المُشار إليه بالمتحول n) يُعطى من الدخل وأن المعدلات محسوبة من 100 علامة وأن المُستخدم يقوم بإدخال المعدلات عند تنفيذ البرنامج.

حدد عند توصيفك ومعالجتك للمطلوب:

1. دخل البرنامج .
2. خرج البرنامج .
3. الفرضيات الأساسية التي يجب معالجتها .
4. خوارزمية الحل بلغة الخوارزميات.

المسألة الثالثة:

بفرض أن لديك معادلة مستقيم:

$$ax + by = c$$

تعبر a,b,c عن قيم صحيحة أوحقيقية، في حين يعبر كل من x و y عن متحولين، إذ يعبر x عن محور الفواصل (المحور OX) ويكون y هو المتحول المعبّر عن محور التراتيب (المحور OY).

نقول عن نقطة (x_0,y_0) أنها منتمية إلى المستقيم ax+by=c إذا تحققت معادلة المستقيم بتعويض x و y بـ x_0 وy_0 على الترتيب، كمايلي:

$$a \times x_0 + b \times y_0 = c$$

فعلى سبيل المثال، ومن أجل المستقيم $5x + 4y = 13$ تكون النقطة $(1,2)$ منتمية إلى المستقيم لأن:

$$5 \times 1 + 4 \times 2 = 13$$

المطلوب كتابة خوارزمية التحقق من انتماء نقطة إلى مستقيم بحيث تحدد:

1. دخل البرنامج (مساعدة: المستقيم والنقطة).
2. خرج البرنامج (مساعدة: انتماء أو عدم انتماء).
3. الفرضيات الأساسية التي يجب معالجتها في البرنامج.
4. التعبير عن الحل بلغة الخوارزميات.

الفصل الثاني:
أساسيات لغة C#

الكلمات المفتاحية:

برنامج، متحول، ثابت، نمط ، صف، فضاء الأسماء، كلمات محجوزة (مفتاحية)

ملخص:

نتعرف في هذا القسم على أساسيات لغة البرمجة C#؛

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- تشغيل محيط التطوير Dot Net بهدف برمجة تطبيقات بسيطة بلغة C#
- البنية العامة لنص برنامج في C#
- الأنماط الأساسية (البسيطة)
- المتحولات والثوابت
- العمليات الحسابية والمنطقية وعمليات المقارنة
- أفضليات العمليات الأساسية

المخطط:

1. Microsoft Dot Net
2. بنيان إطار العمل Dot Net Framework
3. بداية سريعة مع C#
4. تحليل النص البرمجي
5. الكلمات المحجوزة (المفتاحية) C# Reserved words (Keyword)
6. نماط الأساسية
7. المتحولات في C#
8. الثوابت في C#
9. العمليات في C# وأفضلياتها-1
10. العمليات في C# وأفضلياتها-1
11. العمليات في C# وأفضلياتها-3
12. العمليات في C# وأفضلياتها-4
13. العمليات في C# وأفضلياتها-5
14. تعليمة القراءة
15. تمارين للتجريب

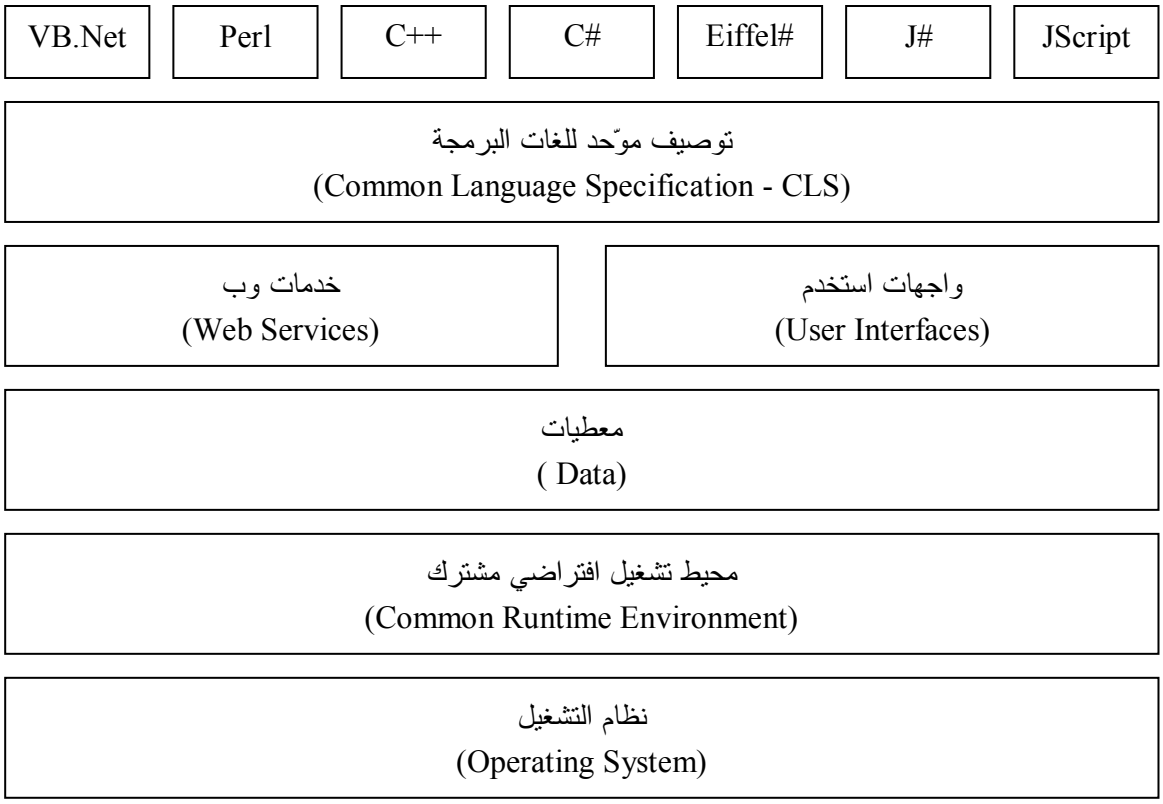
1. Microsoft Dot Net

اقترحت Microsoft استراتيجية جديدة لتوزيع عملية معالجة المعطيات في إطار بنيان برمجي متكامل، تحت اسم "Dot Net". يرتكز البنيان الآنف الذكر على مجموعة من الأفكار المؤسِّسة التي تطمح للوصول إلى بيئة برمجية تتمتع بالمواصفات التالية:

- شفافية التعامل مع التطبيق من ناحية كونه تطبيق محلي أو تطبيق إنترنت
- توزيع المعطيات على عدد من المخدمات عوضاً عن تركيزها ضمن مخدم واحد، وبحيث يحتوي كل مخدم على الخدمات اللازمة للتعامل مع جزئه الخاص من المعطيات
- تحويل عملية شراء تطبيق برمجي وتثبيته على مخدمات محلية إلى عملية استئجار خدمة برمجية تقدمها مجموعة مخدمات على الإنترنت
- تحويل الحاسب الشخصي إلى طرفية ذكية تساعد في البحث عن الخدمة المطلوبة وتشغيلها عن بعد
- تأمين مكوِّنات برمجية جاهزة يمكن لمطوري البرامج مكاملتها ضمن برامجهم دون الحاجة لإعادة برمجتها

بالنتيجة، تقدم Microsoft من خلال Dot Net محيطاً برمجياً يُدعى (Dot Net Framework) يساعد المُبرمج في برمجة وتشغيل تطبيقاته سواءاً كانت تطبيقات كلاسيكية تعمل ضمن محيط نظام التشغيل Windows أو تطبيقات وب تعمل ضمن محيط مخدم وب.

2. بنيان إطار العمل Dot Net Framework



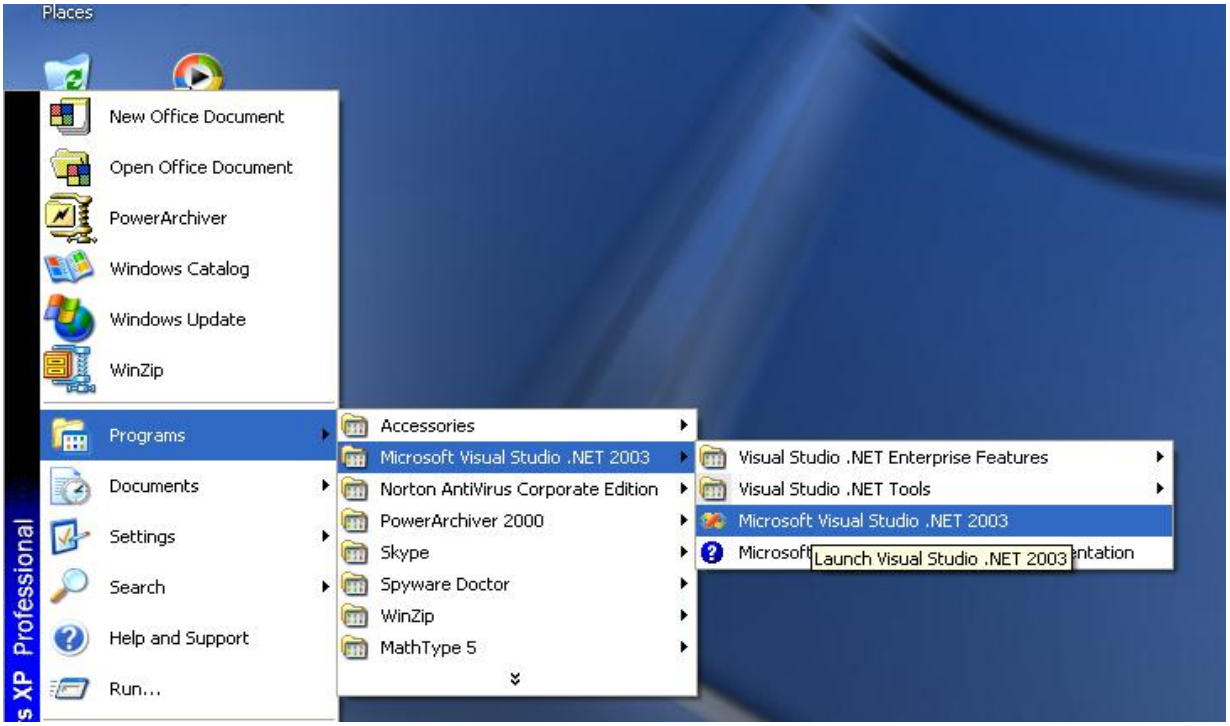
يُقسم بنيان إطار العمل Dot Net إلى مجموعة من الطبقات التي تساعد المُبرمج على كتابة برامجه وتحويلها إلى برامج تنفيذية.

سنستعرض في هذا الشكل مجموعة الطبقات وسنركز في بقية القسم على عمل الطبقتين الأولى والثانية التي تهمنا كمبرمجين وخصوصاً بالنسبة للغة C# وأسلوب استثمارها لهذا المحيط.

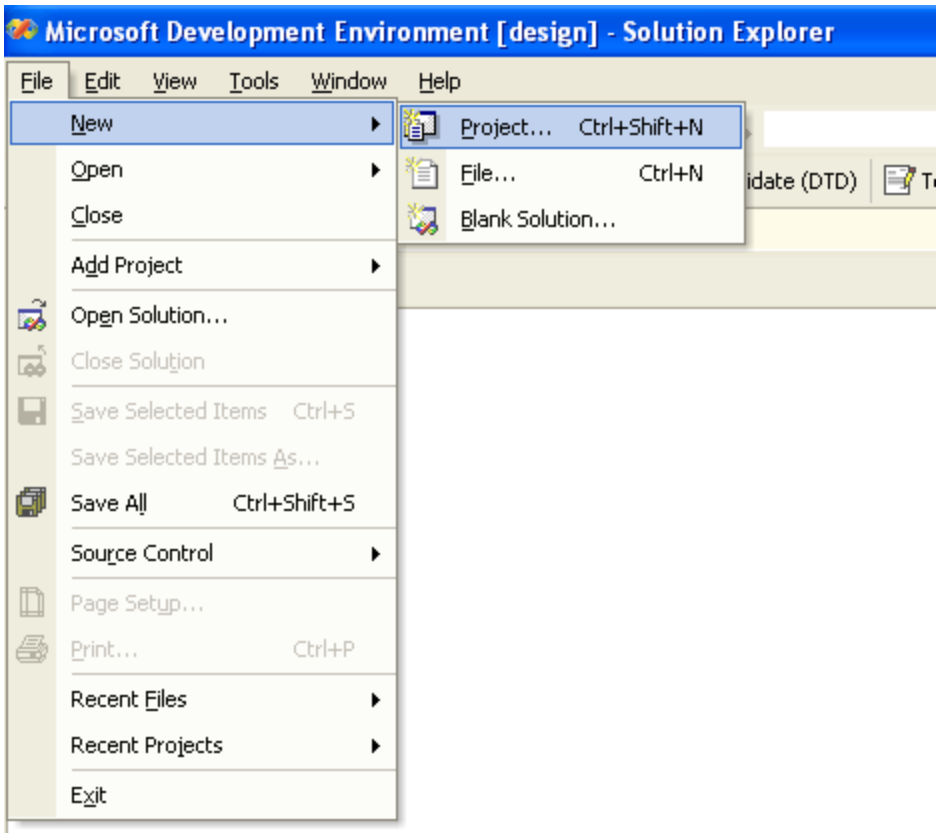
3. بداية سريعة مع C#

بعد تثبيت Visual Studio.net نفذ العمليات التالية:

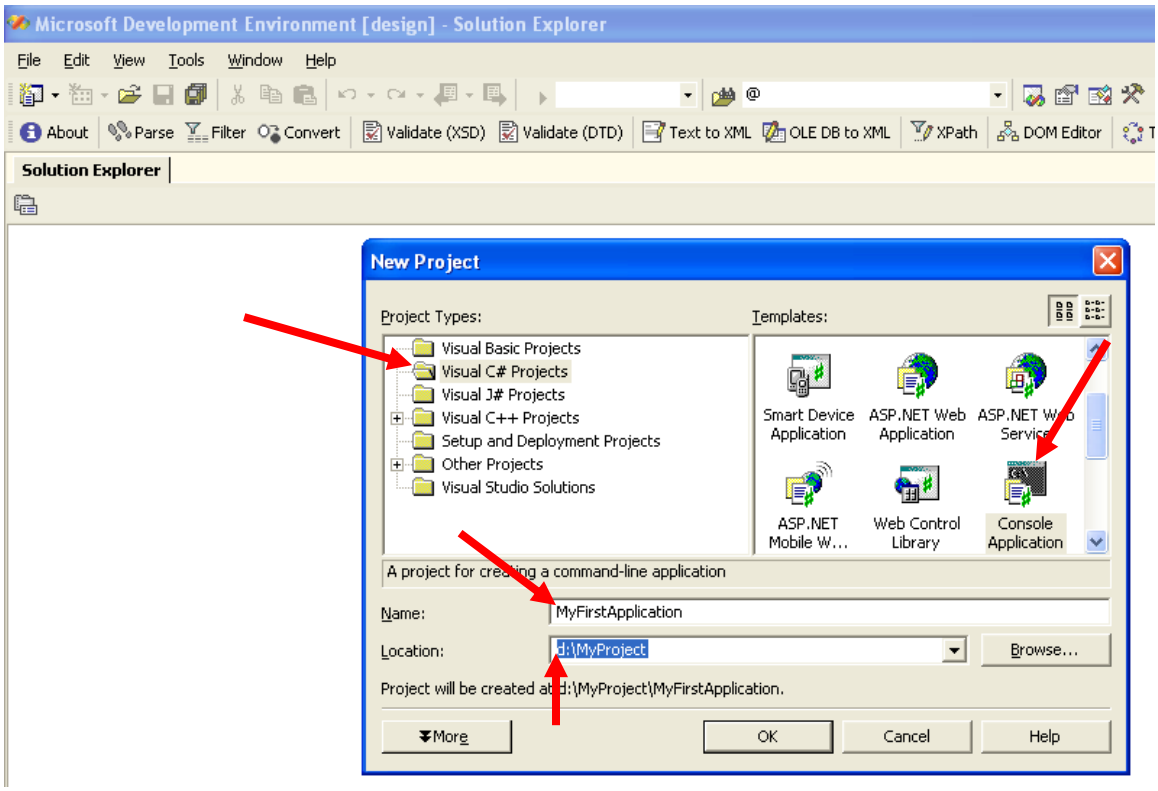
1. اذهب إلى زر البداية Start وإلى مكان إقلاع Microsoft Visual Studio .Net:



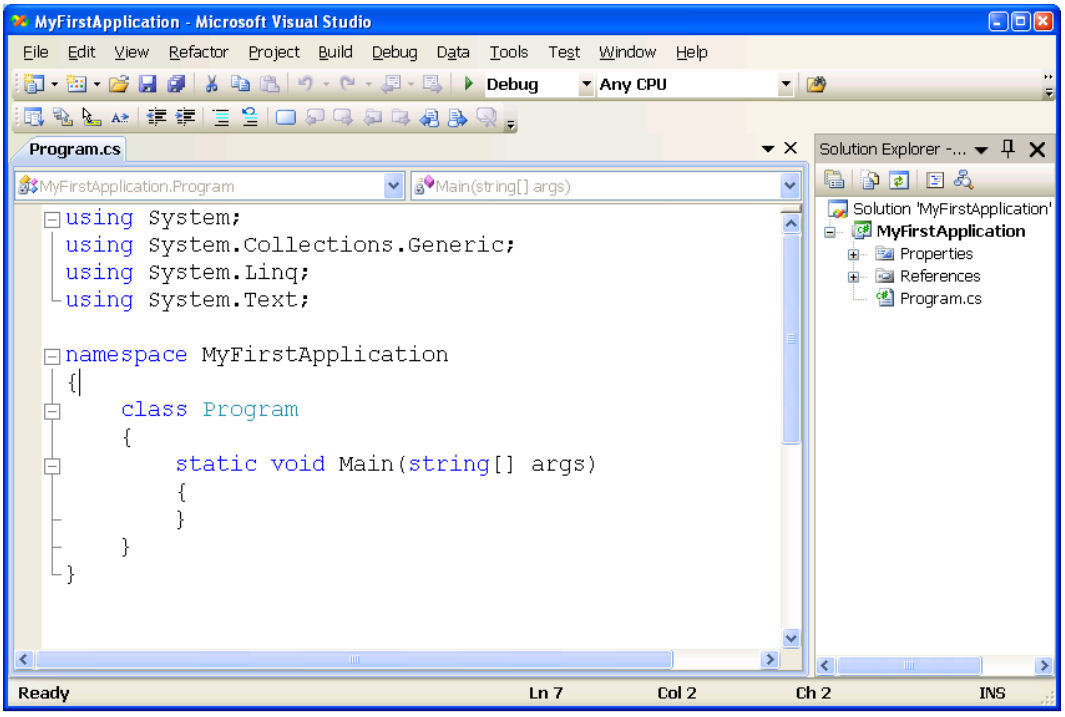
2. عند إقلاع Visual studio .Net إذهب إلى نافذة فتح المشاريع الظاهرة في الشكل:



3. يمكنك عندها اختيار C# Project من النافذة اليسارية، و Consol Application من النافذة اليمينية مع تحديد اسم التطبيق ومكان تخزينه في الأسفل، كما هو موضح في الشكل:

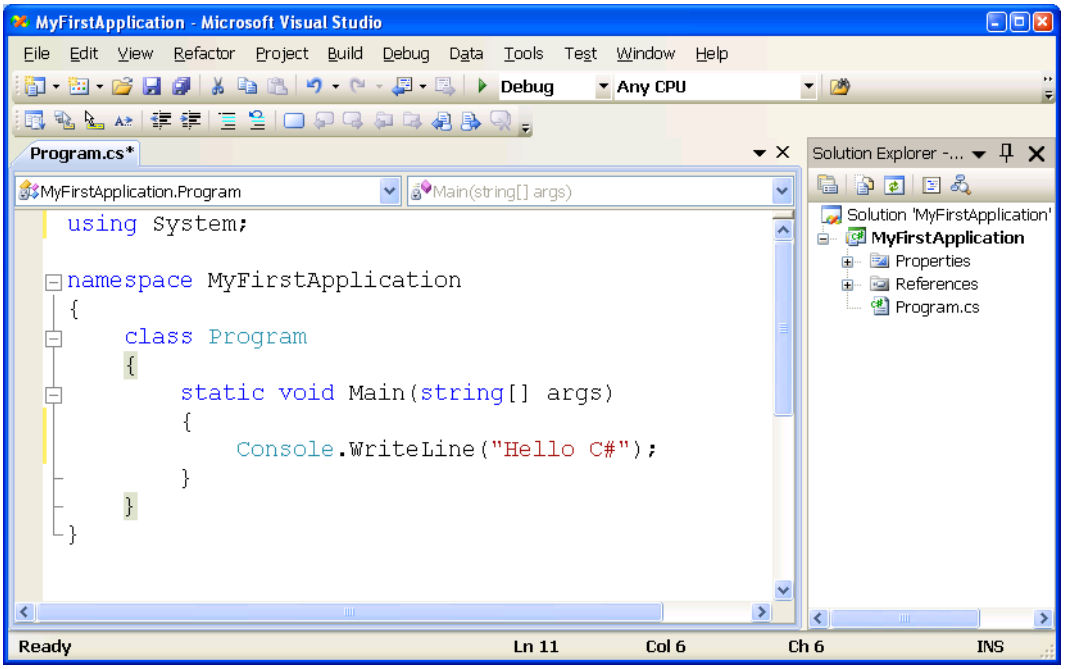


4. ستحصل على الواجهة التالية التي تؤطر البرنامج الذي ستكتبه:

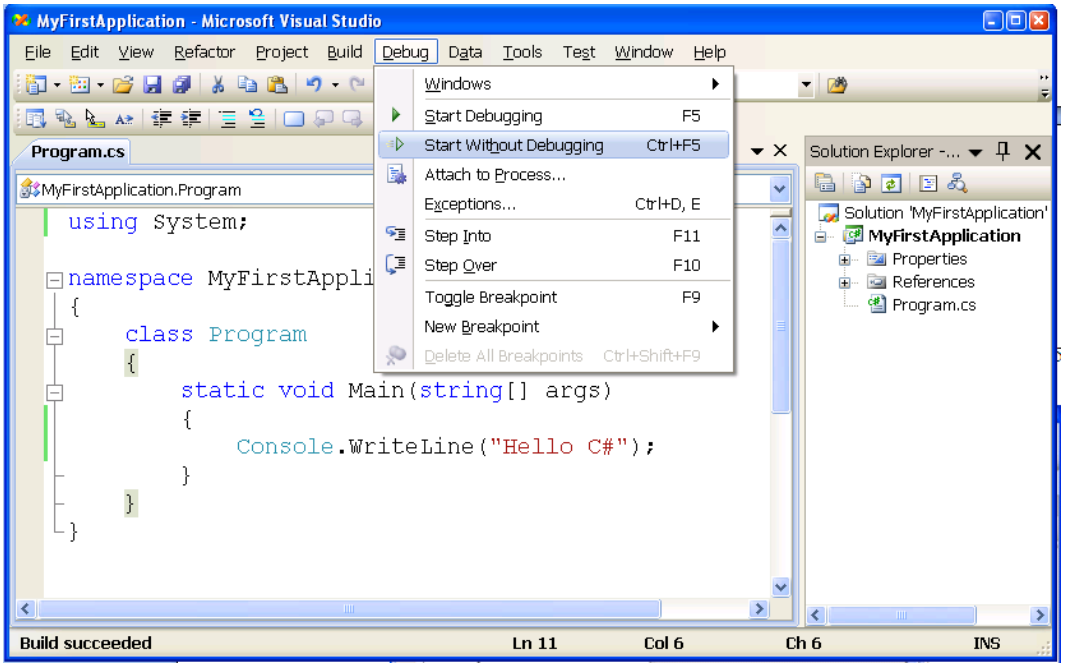


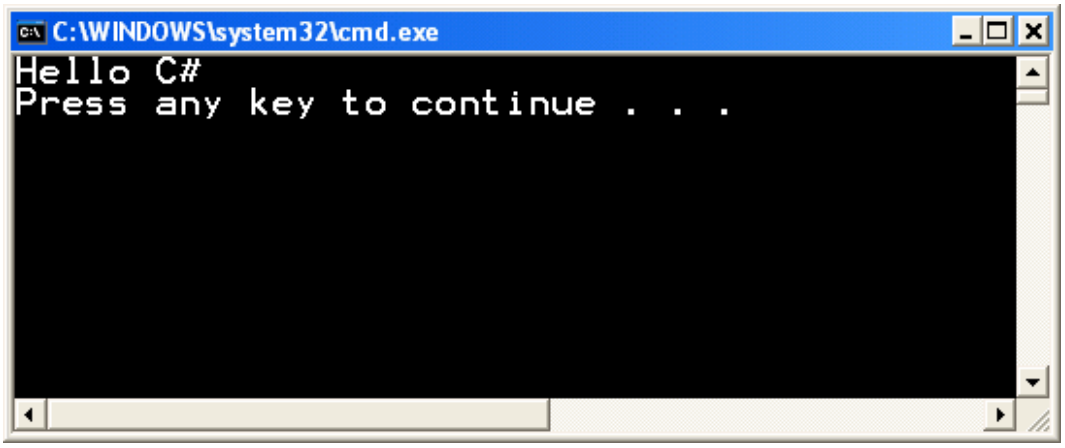
5. يمكنك كتابة برنامجك الأول الموضح فيما يلي ضمن إطار النص البرمجي المُعطى:

```
using System;

namespace MyFirstApplication
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine("Hello C#");
}
}
}
```



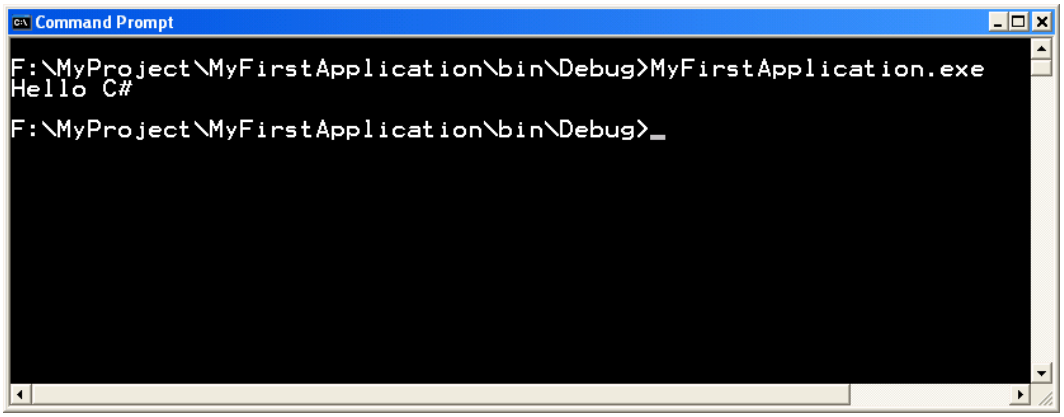
6. يمكنك ترجمة برنامجك والتحقق من صحته بالذهاب إلى واجهة التنفيذ Debug ومن ثم Start:





7. كما يمكن بعد ذلك تنفيذ البرنامج اعتباراً من Dos Command Prompt كما يظهر من الشكل تحت اسم MyFirstApplication.exe

(لقد جرى توليد البرنامج التنفيذي داخل المجلد MyFirstApplication \bin\Debug):



4. تحليل النص البرمجي

```
using System;

namespace MyFirstApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello C#");
        }
    }
}
```

لاحظ أولاً الكلمات الملونة بالأزرق: using, namespace, class, static, void, string هي كلمات مخصصة/محجوزة للغة البرمجة C# وهي التي تساهم في التعبير عن البنية القواعدية للغة (غالباً ما تُسمّى أيضاً كلمات مفتاحية keyword).

- نُستَخدَم العبارة "using System" للدلالة على ماندعوه فضاء الأسماء "System" الذي يُقدم مجموعة من الصفوف الجاهزة والمُعرّفة التي يمكن استخدامها ضمن التطبيق مباشرةً
- ينتمي الصف "Console" إلى فضاء الأسماء "System" ويُستَخدَم للتعامل مع الواجهة النصيّة (الشاشة أسود/أبيض تسمح بإظهار نصي فقط) كما لاحظنا عند تشغيل البرنامج
- يستخدم الصف "Console" التعليمة/الإجرائية "WriteLine" لكتابة سلسلة محارف وإظهارها على الواجهة النصيّة
- يُعرِّف المثال صفاً يُدعى "Program"، يحتوي على إجرائية "Main" يمكن اعتبارها نقطة إنطلاق لتنفيذ المثال
- تقوم التعليمة/الإجرائية WriteLine بإظهار عبارة "Hello C#" عند استدعاء التطبيق من واجهة التعليمات النصيّة

ملاحظة هامة جداً:

- إن لغة C# هي لغة غرضية التوجه، وبالتالي تعتمد في تأطير البرمجة، على ما يسمى الصف class لكننا في أساسيات البرمجة، لسنا معنيين بمعرفة آلية البرمجة في C# لبناء الصفوف... بقدر ما نحن معنيون باستخداماتها

- سنعرِّف الصف (بما يعنينا في أساسيات البرمجة) بأنه تجميع لمعطيات ووظائف.
لوظائف (هي غالباً مانسميها توابع، إجرائيات، طرائق)
من الناحية العملية البرمجية والقواعدية نطلب الوظيفة من صف بالشكل: ClassName.F()
كما في المثال:
Console.WriteLine();
- أخيراً فضاء الأسماء namespace هو تجميع لعدد من الصفوف.

5. Reserved words (Keyword) C# الكلمات المحجوزة (المفتاحية)

abstract	extern	operator	throw
as	false	out	true
Bab base	finally	override	try
bool	fixed	params	typeof
break	float	partial	uint
byte	for	private	ulong
case	foreach	protected	unchecked
catch	get	public	unsafe
char	goto	readonly	ushort
checked	if	ref	using
class	implicit	return	value
const	in	sbyte	virtual
continue	int	sealed	void
decimal	interface	set	volatile
default	internal	short	where
delegate	is	sizeof	while
do	lock	stackalloc	yield
double	long	static	
else	namespace	string	
enum	new	struct	

event	null	switch	
explicit	object	this	

يتضمن الجدول حميع الكلمات المحجوزة للغة C# . ولكن في أساسيات البرمجة لا نحتاج إلا معرفة ودلالة عدد محدد منها.

6. الأنماط الأساسية

يمكن للمُبرمج استخدام أحد الأنماط البسيطة التي تظهر في الجدول لتعريف متحولاته. ويؤدي تعريف كل متحول إلى حجز جزء من الذاكرة يُقدر بالبايت ويتعلق بالنمط المُستَخدَم.

النمط	عدد الـ Byte التي يحجزها في الذاكرة	توصيف
byte	1	قيمة صحيحة بدون إشارة تتراوح بين 0 و 255
sbyte	1	قيمة صحيحة ذات إشارة تتراوح بين −128 و +127
short	2	قيمة صحيحة ذات إشارة تتراوح بين -2^{15} و $2^{-15}-1$
ushort	2	قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{16}-1$
int	4	قيمة صحيحة ذات إشارة تتراوح بين -2^{31} و $2^{31}-1$
uint	4	قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{32}-1$
long	8	قيمة صحيحة ذات إشارة تتراوح بين -2^{63} و $2^{63}-1$
ulong	8	قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{64}-1$
float	4	فاصلة عائمة بدقة بسيطة ~ 7 أرقام عشرية
double	8	فاصلة عائمة بدقة مُضاعفة ~ 15 رقم عشري
decimal	16	دقة تصل إلى 28 رقم عشري على الأكثر
string		سلسلة محارف
char	2	حرف يتبع الترميز Unicode (قيمته بين 0 و 65536)
bool		يأخذ إحدى القيمتين TRUE أو FALSE

مثال:

```
namespace MyFirstApplication
{
    class Program1
    {
        static void Main(string[] args)
        {
            int Y = 2015;
            string s = "Hello C#: ";
            Console.WriteLine(s + Y);
        }
    }
}
```

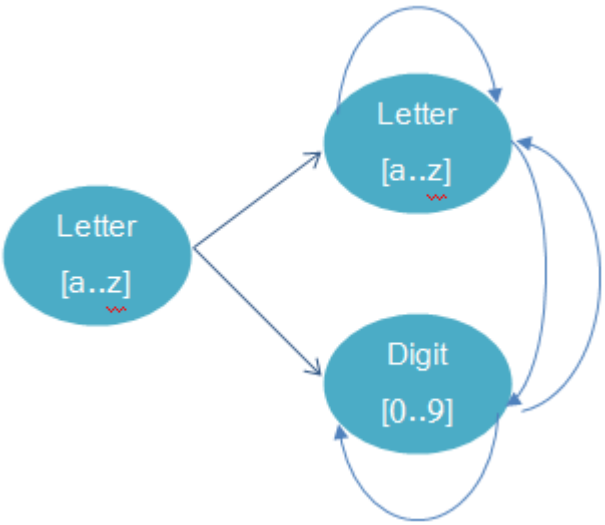
لنحصل على النتيجة التالية عند التنفيذ:

تذكَّر u هي اختصار لـ unsigned بلا إشارة (أي عدد موجب)

انتبه: لا يمكننا تصنيف النمط string (سلسلة محارف) كنمط بسيط، فهو مركَّب من مجموعة من الحروف، ولكنه وُضع في الجدول لأن الجدول يضم جميع الأنماط المتاحة في لغة C# . وجميع أسماء هذه الأنماط هي كلمات محجوزة keyword في لغة C#.

7. المتحولات في C#

تبدأ أسماء المتحولات بحرف من الحروف الأبجدية اللاتينية (من a إلى z) تليها اختيارياً سلسلة من الحروف والأرقام وفقاً للمخطط التالي:



بالإضافة لماسبق، يمكن استخدام "_" (Underscore).

- يجب ألا يكون اسم المتحول مطابقاً لأي من الكلمات المحجوزة keyword للّغة
- يجب الانتباه أن C# مثلها مثل جميع اللغات الشيهة بلغة C تفرّق بين الأحرف الصغيرة والكبيرة. فاسم المتحول n هو غير الاسم N . (كثير من لغات البرمجة لا تفرّق)

أمثلة عن تعريف المتحولات:

- تعريف متحولات بدون قيم أولية:

```
int MaxN;
double Value2 ;
char aChar ;
bool Test ;
long N_F ;
```

- تعريف متحولات مع إسناد قيم أولية لها:

```
int MaxN=50 ;
double Value2=2.3;
char aChar='K' ;
bool Test=false ;
long N_F= 1600000000000;
```

8. الثوابت في C#

تستخدم لغة C# الكلمة المفتاحية const للدلالة على القيم الثابتة أي المتحولات التي لا يمكن لقيمها أن تتغير، حيث تُستَخدَم عند تعريف المتحول أو الثابت قبل اسم النمط.

يجب على الكلمات المُعرَّفة كـ const أو تأخذ قيم عند تعريفها مبشرةً

مثال:

```
const int Num=0;  // Accepted
const int Num; // Error – without Initialization
```

وتسبب عبارة إسناد للثابت Num بحدوث خطأ ناجم عن عدم إمكانية تعديل محتواه:

```
namespace MyFirstApplication
{
     class Test1
     {
          public static void Main()
          {
               const int Num=0;

               Num = 10;

               if ( Num < 0 )

System.Console.WriteLine("Negative");
               else

System.Console.WriteLine("Positive");
          }
     }
}
```

9. العمليات في C# وأفضلياتها-1

العمليات الحسابية

العملية	التوصيف	الأفضلية	مثال
+	إشارة موجبة	1	+a; +b; +1; x=+z;
-	إشارة سالبة	1	-a; -b; -(5+x); y=-(6*u);
++	زيادة بقيمة واحد	1	a++; ++y; v=++c;
--	نقصان بقيمة واحد	1	x--; --z; x=--p;
*	عملية الضرب	2	(a*b); x*y; 6*z; x=a*b;
/	عملية القسمة	2	(a/b); x/5; y=u/y;
%	عملية باقي القسمة	2	x%5; z%a; z=u%t;
+	عملية الجمع	3	a+b; (x+y)+z; t=x+y;
-	عملية الطرح	3	-a-b; x-y-z; t=x-y-z;

10. العمليات في C# وأفضلياتها-2

عمليات المُقارنة:

العملية	التوصيف	الأفضلية	مثال
>	أكبر تماماً	5	a>b; a>(x+y);
>=	أكبر أو يساوي	5	a>=b; a>=(x+y);
<	أصغر تماماً	5	a<b; a<(x+y);
<=	نقصان بقيمة واحد	5	a<=b; a<=(x+y);
==	يساوي	6	a==b; (x+y) == (z+r);
!=	لايساوي	6	a!=b; (x+y) != (z+r);

11. العمليات في C# وأفضلياتها-3

العمليات المنطقية:

العملية	التوصيف	الأفضلية	مثال
!	نفي	1	! ((a+b) < 6);
&	و	7	((a+b)<6) & ((x+y)>7);
|	أو	9	((a+b)<6) | ((x+y)>7);
&&	"و" مُحسَّنة	10	((a+b)<6) && ((x+y)>7);
||	"أو" مُحسَّنة	11	((a+b)<6) || ((x+y)>7);

12. العمليات في C# وأفضلياتها-4

ملاحظات على العمليات المنطقية

تُعبِّر عملية ! عن نفي عبارة منطقية وتفترض عدم تحقق الطرف حتى تكون محققة، بحيث يكون جدول الحقيقة للعبارة (!((a+b)>8) هو:

a+b > 8	! ((a+b) > 8)
True	False
False	True

تُعبِّر عملية & عن "و" منطقية وتفترض تحقق الطرفين حتى تكون محققة، بحيث يكون جدول الحقيقة للعبارة ((a+b)<6) & ((x+y)>7) مثلاً:

(a+b)<6	(x+y)>7	((a+b)<6) && ((x+y)>7)
True	True	True
True	False	False
False	True	False
False	False	False

تُعبِّر عملية | عن "أو" منطقية وتفترض تحقق أحد الطرفين حتى تكون محققة، بحيث يكون جدول الحقيقة للعبارة ((a+b)<6) | ((x+y)>7) مثلاً:

(a+b)<6	(x+y)>7	((a+b)<6) | ((x+y)>7)
True	True	True
True	False	True
False	True	True
False	False	False

تُعبِّر عملية && عن "و" منطقية أمثلية، كما تُعبِّر عملية || عن "أو" منطقية أمثلية، إذ تمتلك كلتا العمليتان نفس جدول الحقيقة لكل من & و | على الترتيب، إلا أن أهمية هذه العمليات أنه في حالة && مثلاً لاتتم بالضرورة عملية تقييم الطرفين، بل يكفي أن يكون أحد طرفي العبارة (الذي جرى تقييمه أولاً) خطأ حتى يجري اعتبار العبارة خاطئة بكاملها.

13. العمليات في C# وأفضلياتها-5

أفضليات العمليات:

- يمكن للأقواس أن تحل مشكلة الأفضليات

- على سبيل المثال يكون للعبارة ((x+y) > z) && (z<8) التفسير التالي:
  - يجري أولاً حساب x+y ومقارنة النتيجة بقيمة z لتحديد خطأ أو صحة العبارة .((x+y) > z)
  - يجري بعدها مقارنة قيمة z بـ 8 لتحديد خطأ أو صحة العبارة .(z<8)
  - يجري بعد ذلك التحقق من صحة أو خطأ ((x+y) > z) && (z<8) تبعاً لجدول الحقيقة الخاص بالعملية .&&

- في حال عدم وجود أقواس يتم تنفيذ العمليات تبعاً للأفضليات (العمليات ذات الأفضلية 1 لها أسبقية على العمليات ذات الأفضلية 2 وهكذا دواليك)

- أما في حال تسلسل عمليتين لهما نفس الأفضلية، فتكون الأسبقية للعملية الموجودة على اليسار

- على سبيل المثال يكون للعبارة (x+y > z && z<8) التفسير التالي:
  - بما أن عملية "الجمع" تمتلك أسبقية (ذات الأفضلية 3) بالنسبة لعملية المقارنة "أكبر تماماً" (ذات الأفضلية 5) يجري أولاً حساب x+y ومقارنة النتيجة بقيمة z لتحديد خطأ أو صحة العبارة (x+y) > z
  - بما أن عملية المقارنة "أصغر تماماً" تمتلك أسبقية (ذات الأفضلية 5) بالنسبة لعملية الـ "و" المنطقية (ذات الأفضلية 10) يجري أولاً حساب z<8 ومقارنة النتيجة بقيمة z لتحديد خطأ أو صحة العبارة .z<8
  - يجري بعد ذلك التحقق من صحة أو خطأ (x+y > z && z<8) تبعاً لجدول الحقيقة الخاص بالعملية .&&

14. تعليمة القراءة

يمكن لقراءة قيمة متحول ذو نمط بسيط أن نستخدم تعليمة *Read* أو ReadLine التابعة للصف Console وإسنادها للمتحول المطلوب

إلا أن القيمة التي ترجعها Read أو ReadLine هي من نمط سلسلة المحارف. فإذا أدخلنا 123 تمت قراءتها من قبل التعليمة على أنها سلسلة المحارف "123".

يؤدي استخدام التعليمة ReadLine دون أخذ الملاحظة الآنفة الذكر بعين الاعتبار إلى حدوث خطأ (عدم توافق الأنماط الناتج عن قراءة قيمة ذات نمط محرفي ومحاولة اسنادها لمتحول يعبر عن عدد صحيح) في حال نفذنا البرنامج التالي:

```
using System;
namespace MyFirstApplication
{
      class Test1
      {
            public static void Main()
            {
                  int Num;
                  Console.WriteLine("enter The Requested Value: ");

                  Num=Console.ReadLine();

                  if ( Num < 0 )
                        System.Console.WriteLine("Negative");
                  else
                        System.Console.WriteLine("Positive");
            }
      }
}
```

لذا يتوجب في حال أردنا أن نقرأ متحول من نمط عدد صحيح، أو حقيقي أن نستخدم إجرائيات تحويل خاصة كإجرائية Parse المرتبطة بكل نمط من الأنماط البسيطة والتي تقوم بتحويل سلسلة محارف مثل "123" إلى قيمة هي 123، كما هو الحال في البرنامج التالي:

```
namespace MyFirstApplication
{
      class Test1
      {
            public static void Main()
            {
                  string s;
                  int Num;
                  Console.WriteLine("enter The
Requested Value: ");
                  s=Console.ReadLine();
                  Num=Int32.Parse(s);

                  if ( Num < 0 )

      System.Console.WriteLine("Negative");
                  else

      System.Console.WriteLine("Positive");
            }
      }
}
```

عموماً، يكون لكل نمط بسيط صف مقابل يمتلك الإجرائية Parse. نورد هذه الصفوف فيمايلي:

الصف	النمط
Byte	byte
SByte	sbyte
Int16	short
UInt16	ushort
Int32	int
UInt32	uint
Int64	long
UInt64	ulong
Single	float
Double	double

ملا حظة هامة:

يمكن في مترجمات C# حالياً الاستغناء عن استخدام الصف المقابل (الصف المغلِّف) للنمط البسيط، فالمترجم يغلف النمط البسيط بدلاً عنا!

فتعليمات القراءة التالية صالحة (وأسهل للتذكر)

```
string s = Console.ReadLine();
int Num= int.Pars(s);
```

أو

```
string s = Console.ReadLine();
float f =float.Parse(s);
```

15. تمارين للتجريب

تمرين 1– نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;
namespace HelloWorld3s
{
    class Welcome3
    {
       static void Main( string[] args )
       {

          Console.WriteLine("Welcome\nto\nC#\nProgramming!" );
       }
    }
}
```

تمرين 2– نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;
using System.Windows.Forms;

namespace WelcomeGUI
{
   class Welcome4
   {
      static void Main( string[] args )
      {
                  MessageBox.Show( "Welcome\nto\nC#\nprogramming!" );
      }
   }
}
```

تمرين 3– نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;

namespace AdditionProgram
{
   class Addition
   {
      static void Main( string[] args )
      {
         string firstNumber,   // first string entered by user
            secondNumber;  // second string entered by user

         int number1,          // first number to add
            number2,          // second number to add
            sum;              // sum of number1 and number2

         // prompt for and read first number from user as string
         Console.Write( "Please enter the first integer: " );
         firstNumber = Console.ReadLine();

         // read second number from user as string
         Console.Write( "\nPlease enter the second integer: " );
         secondNumber = Console.ReadLine();

         // convert numbers from type string to type int
         number1 = Int32.Parse( firstNumber );
         number2 = Int32.Parse( secondNumber );

         // add numbers
         sum = number1 + number2;

         // display results
         Console.WriteLine( "\nThe sum "+sum );

      } // end method Main

   } // end class Addition

} // end namespace AdditionProgram
```

تمرين 4– نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;

namespace ComparisonApplication
{
   class Comparison
   {
      static void Main( string[] args )
      {
         int number1,           // first number to add
            number2;           // second number to add

         // read in first number from user as a string
         Console.Write( "Please enter first integer: " );
         number1 = Int32.Parse( Console.ReadLine() );

         // read in second number from user as a string
         Console.Write( "\nPlease enter second integer: " );
         number2 = Int32.Parse( Console.ReadLine() );

         if ( number1 == number2 )
            Console.WriteLine( number1 + " == " + number2 );

         if ( number1 != number2 )
            Console.WriteLine( number1 + " != " + number2 );

         if ( number1 < number2 )
            Console.WriteLine( number1 + " < " + number2 );

         if ( number1 > number2 )
            Console.WriteLine( number1 + " > " +  number2 );

         if ( number1 <= number2 )
            Console.WriteLine( number1 + " <= " + number2 );

         if ( number1 >= number2 )
            Console.WriteLine( number1 + " >= " + number2 );

      } // end method Main

   } // end class Comparison

} // end namespace ComparisonApplication
```

الفصل الثالث:
التعليمات في لغة C#

الكلمات المفتاحية:

تعليمة شرطية، تعليمة تكرارية، تعليمة وصل.

ملخص:

نتعرف في هذا القسم على التعليمات الأساسية في لغة البرمجة C# كالتعليمة الشرطية، والتعليمة التكرارية (الحلقة)، وغيرها.

أهداف تعليمية:

يتعلم الطالب في هذا الفصل استخدام البنى البرمجية الأساسية في لغة C#:

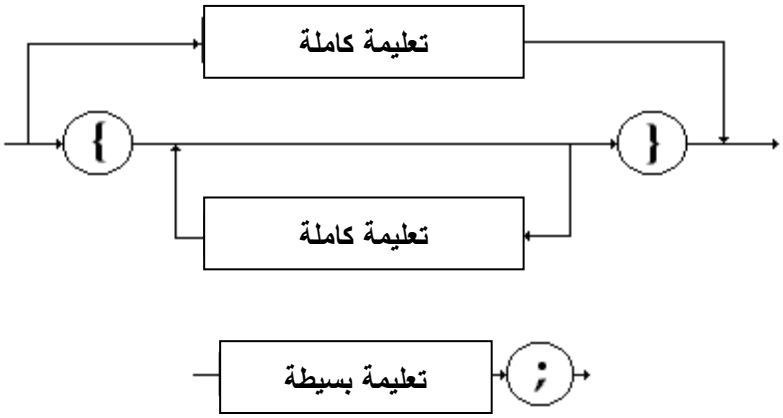
- مفهوم كتلة التعليمات
- التعليمة الشرطية وغموضها واختصارها
- التعليمة التكرارية الأساسية

المخطط:

1. قواعد عامة
2. كتل التعليمات ومدى المتحول
3. تعليمة الإسناد
4. التعليمة الشرطية
5. غموض التعليمة الشرطية
6. تعليمة الإسناد الشرطية
7. التعليمة التكرارية: while
8. التعليمات الخوارزمية الأساسية الخمسة في C#
9. تدريبات
10. تدريب
11. تمرين-1
12. تمرين-2
13. تمرين-3
14. تمرين-4
15. تمرين-5
16. تمرين-6
17. مسائل للحلّ خوارزمياً، ومن ثم بلغة C#

1. قواعد عامة

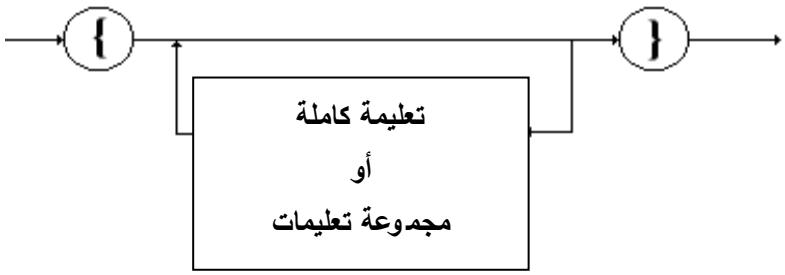
- تم الأخذ بقسم كبير من المعيار ANSI الخاص بلغة C في لغة C#
- يمكن أن نكتب تعليمة كاملة محتواة بين قوسين { } أو كتبها دون أقواس
- تنتهي أي تعليمة بسيطة (كتعليمة الإسناد) بفاصلة منقوطة



- يمكن استخدام تعليقات سطرية تبدأ بالإشارة .... //
- يمكن أن استخدام تعليقات نصيّة في برنامج مكتوب بلغة C# بإحاطتها بـ /* ... */

2. كتل التعليمات ومدى المتحول

نُعرِّف كتلة تعليمات كمايلي:



يمكن لكتلة التعليمات أن تحتوي كتلة تعليمات أخرى على أن تكون الكتل معلّبة ببعضها البعض تماماً، إذ لايمكن أن تتقاطع كتلتي تعليمات جزئياً بأن تكون بداية الثانية بعد بداية الأولى وأن تكون نهاية الثانية بعد نهاية الأولى مثلاً. (هذه هي البرمجة المهيكلة structured programming "هياكل"/ كتل محدّدة تماماً"معششة" داخل بعضها)

مثال1:

```
int a, b = 12;
  { int x , y = 8 ;
    { int z =12;
        x = z ;
        a = x + 1 ;
      { int u = 1 ;
        y = u – b ;
      }
    }
  }
```

نُعرِّف مدى المتحول بالكتلة التي يكون المتحول فيها مُعرَّفاً، ويكون المتحول مُعرّفاً في الكتلة التي تم تعريفه فيها وفي جميع الكتل المحتواة في كتلته، ولكنه لايكون مُعرّفاً في الكتل التي تحوي كتلته أو الكتل الموازية لكتلته.

مثال2:

ليكن لدينا البرنامج التالي:

```
//Block0
int a, b = 12;

//Block1
{
      int x , y = 8 ;
}

// Block2
{
      int z =12;
       a = b + z;
      x = z ; //error
      a = x + 1 ;      //error
}
//Block3
{
      int u = 1 ;
       a = b - u;
      y = u - b ; //error
}
```

تظهر الأخطاء نتيجة عدم وجود أي تعريف لكل من x و y في الكتل التي تظهر بها، في حين لا توجد أي مشكلة في استخدام المتحولات a و b في هذه الكتل.

ملاحظات هامة:

في اللغات شبيهات C (C-like: C, C++, Java, C#) من المسموح تعريف المتحولات في أي كتلة، ومدى المتحول المصرح داخل كتلة يشمل كافة الكتل المحتواة فيها (كل الكتل الداخلية). يمكن أن يكون سطر تعريف المتحول في أي مكان في الكتلة، ولكن الممارسة البرمجية الأفضل هي في وضع تعريف متحولات الكتلة في بدايتها.

من حيث المبدأ يمكن أن نعرِّف متحولات بنفس الاسم في أي من الكتل المتداخلة. لأن ذلك يعني استقلالية المتحولات في كل كتلة (كل متحول، يعرَّف في كتلة، تُحجَز له ذاكرته الخاصة). لكن في C#، لا يمكن استخدام تعريف بنفس الاسم في الكتل المتداخلة، إلا في حالة كتلة الصف وكتلة البرنامج الجزئي/التابع (كما سنرى لاحقاً).

تمرين: أين ستظهر الأخطاء الناجمة عن تعريف مدى المتحولات في البرنامج التالي:

```
//Block0
int a, b = 12;
//Block1
{
	int x , y = 8 ;
	{
		int k=0;
		x = a * y;
		k=k+z;
	}
}

// Block2
{
	int z =12;
	a = z + b;
	{
		int u=0;
		x = a * y;
		u++;
	}
}
//Block3
{
	int u = 1 ;
	a= u - b;
	z = u - b ;
}
```

الحلّ:

```
//Block0
int a, b = 12;
//Block1
{
	int x , y = 8 ;
	{
		int k=0;
		x = a * y;
		k=k+z; //error
	}
}

// Block2
{
	int z =12;
	a = z + b;
	{
		int u=0;
		x = a * y; //error
		u++;
	}
}
//Block3
{
	int u = 1 ;
	a= u - b;
	z = u - b ; //error
}
```

3. تعليمة الإسناد

- الإسناد البسيط:

يكون رمز الإسناد هو "=" حيث نكتب:

x=y

يجب في هذه الحالة أن تكون x هي مُعرِّف متحول.

يمكن استخدام الإسناد ضمن التعابير الحسابية والمنطقية، وعلى شكل إسناد متعدد.

مثال:

```
int a , b = 56, c, d ;
a = (b = 12)+8;  // New value of b
a = b = c = d =8; // Multiple Assignment
```

في الحالة الأولى تأخذ b قيمة أولية عند تعريفها، أما في الحالة الثانية فتأخذ b قيمة جديدة عند استخدامها داخل التعليمة، وتأخذ b قيمة جديدة هي 8 في الحالة الثالثة ولكن ضمن عملية إسناد متعددة.
وتكون قيم المتحولات في الحالات الثلاث بعد انتهاء عمليات الإسناد:

int a , b = 56, c, d ;	a=???, b=56, c=???, d=???
a = (b = 12)+8;	**a=20, b=8**
a = b = c = d =8;	**a=8, b=8, c=8, d=8**

- الإسناد المزود بعملية:

لتكن op إحد العمليات التالية: {+, -, *, /, &, |}.
من الممكن أن نستخدم إسناد مزود بعملية من العمليات السابقة له الشكل:
x op= y;
بشكل مكافئ للإسناد البسيط التالي:
x = x op y;

مثال:

```
int a , b = 56 ;
a = -8 ;
a += b ;  // a = a + b
b *= 3 ;  // b = b * 3
```

تكون قيم المتحولات في الحالات الثلاث بعد انتهاء عمليات الإسناد:

int a , b = 56 ;	a=???, b=56
a = -8 ;	**a=-8**
a += b ;	**a=46**
b *= 3 ;	**b=168**

تمرين:

18. احسب قيم المتحولات a, b, c, d, e بعد تنفيذ كل سطر من أسطر العمليات التالية يدوياً دون كتابة برنامج لحسابها:

```
int a=0, b=0, c=8, d=0, e=0;
a=b=c;
b*=5;
a/=2;
e= ((a+b) * (d=6))/2;
d %= 5;
a= b * c - d;
e=d=c=b=(a = (a%31-9));
```

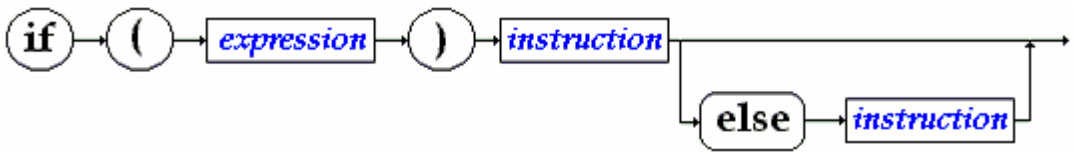
19. اكتب برنامج بسيط بلغة C# للتأكد من النتائج التي حسبتها يدوياً.

الحلّ:

int a, b, c=8, d, e;	a=0, b=0, c=8, d=0, e=0
a=b=c;	a=8, b=8, c=8, d=0, e=0
b*=5;	a=8, b=40, c=8, d=0, e=0
a/=2;	a=4, b=40, c=8, d=0, e=0
e= (a + b * (d=6))/2;	a=4, b=40, c=8, d=6, e=122
d %= 5;	a=4, b=40, c=8, d=1, e=122
a= b * c - d;	a=0, b=0, c=0, d=0, e=0

4. التعليمة الشرطية

تأخذ التعليمة الشرطية الشكل القواعدي التالي:



بحيث يكون للتعليمة أحد شكلين:

الشكل if

```
if (expression)  instruction;
```

- يجري اختبار الشرط المنطقي logical expression الذي يعيد قيمة منطقية صح أو خطأ
  - فإذا كانت صح يجري تنفيذ instruction
  - وإلا يجري الخروج دون تنفيذ instruction

الشكل if else

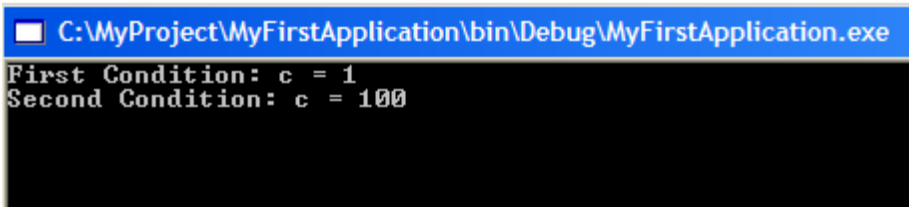
```
if (expression) instructionsBloc1 ; else instructionsBloc2;
```

- يجري اختبار الشرط المنطقي logical expression الذي يعيد قيمة منطقية صح أو خطأ
  - فإذا كانت صح يجري تنفيذ instructionsBloc1
  - وإلا يجري تنفيذ instructionsBloc2

مثال:

```
int a=100, b=0, c;

if ( b == 0 )
{
     c = 1;
     System.Console.WriteLine("First Condition:
c = " + c);
}
else
{
     c = a / b;
     System.Console.WriteLine("First Condition:
c = " + c);
}

if ((c = a*b) != 0)
     c += b;
else
        c = a;

System.Console.WriteLine("Second Condition: c =
" + c);
```

وتكون النتيجة:

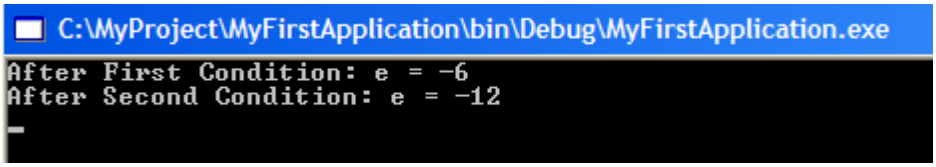


تمرين:

أوجد نتيجة البرنامج التالي المكتوب بلغة C#:

```
int a=100, b=0, c=8, d, e=100;
if ( b == 0 )
{
     a=b=c;
     b*=3;
     a/=2;
     e= (a - b * (d=4))/2;
     d %= 3;
     a= b * c + d;
     e=d=c=b=(a = (a%5-9));
     System.Console.WriteLine("After First
Condition: e = " + e);
}
else
{
     c = a / b;
     System.Console.WriteLine("After First
Condition: c = " + c);
}

if (e != 0)
     e += b;
else
     e = a;
System.Console.WriteLine("After Second
Condition: e = " + e);
```

الحلّ:



5. غموض التعليمة الشرطية

يمكن أن تظهر التعليمة الشرطية بشكل غامض كما هو الحال في المثال التالي:

```
if ( x>0 )
if ( y+z>10 )
x=x+5 ;
else
x=x-5;
```

في هذه الحالة يظهر الغموض في تحديد تبعية تعليمة else لتعليمة if.

بشكل عام تكون تعليمة else تابعة لتعليمة if الأقرب إلا إذا حددت الأقواس عكس ذلك. ففي الحالة السابقة تكون التبعية كمايلي:

```
if ( x>0 )
        if ( y+z>10 )
                x=x+5 ;
        else
                x=x-5;
```

طبعاً، يمكن للأقواس في حال استخدامها أن تحلّ الغموض وفقاً لتوزعها. ففي الحالتين التاليتين تُزيل الأقواس الغموض تماماً وتحدد تبعية تعليمة else لتعليمة if:

```
if ( x>0 )
{
        if ( y+z>10 )
                x=x+5 ;
        else
                x=x-5;
}
```

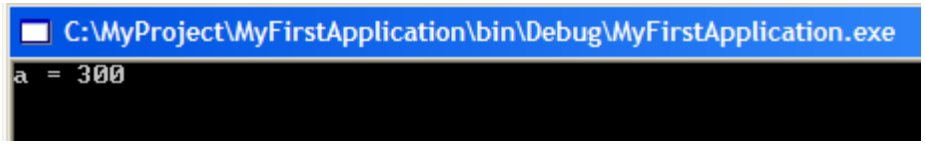
```
if ( x>0 )
{
        if ( y+z>10 )
                x=x+5 ;
}
else
        x=x-5;
```

تمرين:

أوجد نتيجة البرنامج التالي المكتوب بلغة C#:

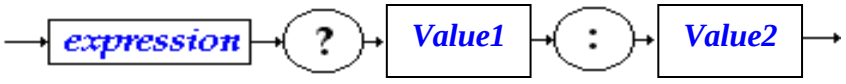
```
int a=100, b=0, c=8, d, e=100;
if ( b == 0 )
     b+=3;
if ( a <= 0)
     a= a + b;
else
     a = a * b;

System.Console.WriteLine("a = " + a);
```

الحلّ:



6. تعليمة الإسناد الشرطية

يمكن في بعض الحالات عندما يكون الهدف من التعليمة الشرطية تنفيذ عملية إسناد أن نبني تعليمة شرطية مختصرة لها الشكل القواعدي التالي:



في حال تحقق الشرط المنطقي المحدد في logical expression
يجري إرجاع Value1
وإلا يجري إرجاع Value2.

وتكافئ التعليمة السابقة التعليمة:

```
if (expression) Value1 else Value2
```

مثال 1:

نقرأ التعليمة الشرطية المُختصرة التالية:

```
int a,b,c ;
….
c = ( ( a == 0 ) ?  b  :  a+1 ) ;
```

كما يلي:

في حال تحققت التعليمة (a==0) يجري إرجاع القيمة الأولى (b) وإسنادها إلى (c) وإلا فإن القيمة الثانية (a+1) هي التي يجري إرجاعها وإسنادها إلى (c).

مثال 2:

ماذا تمثِّل تعليمة الإسناد الشرطية التالية:

```
float x, absx ;
absx = ( ( x >= 0 ) ?  x  :  -x ) ;
```

(تابع القيمة المطلقة لـ x)

تمارين

1. اكتب مكافئات التعليمات التالية بصيغة if ... else:

```
c =    a == 0   ?   b   :   a+1 ;
d =    a >= 0   ?   b-1   :   b+1 ;
e =   (a+b)>0 ? a++ : a--;
```

2. أعط نتيجة تنفيذ البرنامج التالي المكتوب بلغة C#:

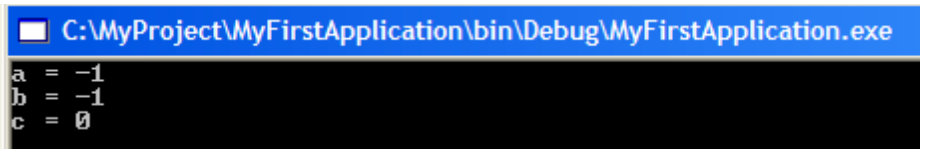
```
int a=0,b=0,c=0 ;

a =    a == 0   ?   b   :   a+1 ;
b =    a >= 0   ?   b-1   :   b+1 ;
c =    (a+b)>0 ? a++ : a--;

System.Console.WriteLine("a = " + a);
System.Console.WriteLine("b = " + b);
System.Console.WriteLine("c = " + c);
```

الحلّ:

1.

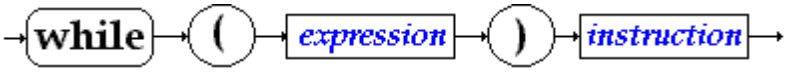
`c =  a == 0  ?  b  :  a+1 ;`	`if (a==0) c=b;` `else c=a+1;`
`d =  a >= 0  ?  b-1  :  b+1 ;`	`if (a>=0) c=b-1;` `else c=a+1;`
`e =  (a+b)>0 ? a++ : a--;`	`if (a+b>0) c=a++;` `else c=a--;`

2.



7. التعليمة التكرارية: while

يكون لتعليمة التكرار الحلقية while الشكل القواعدي التالي:



- يجري أولا اختبار الشرط المنطقي logical expression الذي يعيد قيمة منطقية صح أو خطأ
  - فإذا كانت صح يجري تنفيذ instruction
  - وإلا يجري الخروج دون تنفيذ instruction
- بعد كل تنفيذ للتعليمات instruction، يجري اختبار الشرط المنطقي logical expression للتأكد من أن قيمته المنطقية مازالت صح:
  - فإذا كانت صح نستمر في تكرار تنفيذ instruction مرة أخر
  - وإلا يجري التوقف عن تكرار تنفيذ instruction. والانتقال إلى التعليمة التي تلي while

مثال:

```
int i=0;
int x=0;

while (i<=10)
{
      x=x+10;
       i++;
}
```

8. التعليمات الخوارزمية الأساسية الخمسة في C#

مثال: برنامج حساب القاسم المشترك الأعظم وفق خوارزمية إقليدس (طرح الأصغر من الأكبر)

```
using System;
namespace Instructions
{
class GCD     // compute Greatest Common Divisor
{

    public static void Main(String[] arg)
    {

        int a; string Sa;
        int b; string Sb;
        int g;

        // Read a, b
        Console.Write("a =");
        Sa = Console.ReadLine(); a = Int32.Parse(Sa);

        Console.Write("b =");
        Sb = Console.ReadLine(); b = Int32.Parse(Sb);

        // Compute GCD of a and b
        while (a != b)
        {
            if (a > b)
                a = a - b;
            else
                b = b - a;
        }
        g = a;

        Console.WriteLine("GCD =" + g);
    }
}
}
```

التعليمات الأساسية الخمسة للغة الخوارزميات (pseudo code) ومقابلاتها في C#

1. تعليمة القراءة أو إدخال المعطيات وتكتب على أحد الأشكال	Sa = Console.ReadLine(); a = Int32.Parse(Sa); Sa = Console.ReadLine(); a = int.Parse(Sa); a = int.Parse(Console.ReadLine());
2. تعليمة الكتابة أو إظهار النتائج	Console.WriteLine("GCD =" + g); "كما لو أن" + هنا هي إشارة الفصل بين الأشياء التي أريد أن أكتبها!!
3. تعليمة الإسناد أو وضع قيم تعبير في متحول	a = a - b;

4. التعليمة الشرطية	if (a > b)
	a = a - b;
	else
	b = b - a;

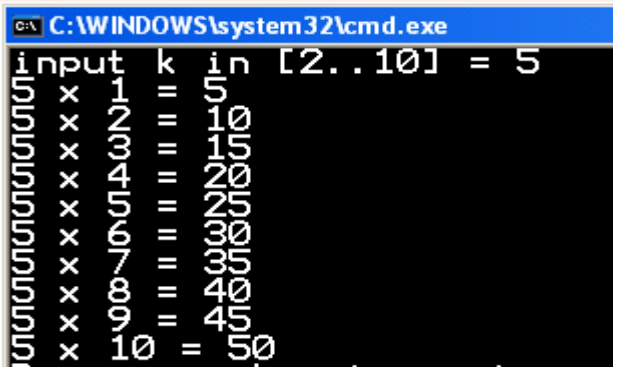
5. التعليمة التكرارية	while (a != b)
	{
	}

9. تدريبات

تدريب1: كتابة جدول الضرب المدرسي لعدد k في المجال [2..10] ، حيث نقرأ قيمة k من الدخل.

```
using System;
namespace ExampleWhile
{
class Multiply
{
 public static void Main(string[] arg)
 {

    int k, i; string Sk;

    Console.Write("input k in [2..10] = ");
    Sk = Console.ReadLine(); k = Int32.Parse(Sk);

  // Multiplication Table of number k

    i = 1;
    while (i <= 10)
    {
       Console.WriteLine(k + " x " + i + " = " + k * i);
       i = i + 1;
    }
}
}
}
```

التنفيذ (عند قيمة الدخل 5):



تدريب2:

كتابة كل جداول الضرب للأعداد من 1 حتى 10

الدخل: لايوجد.

الخرج: جداول الضرب.

الخوارزمية:

{ كتابة جدول الضرب k }

```
i ← 1
مادام ( i ≤ 10 ) كرر
   بداية
      اكتب k, " x ", i ," = ", k*i
      i ← i +1
   نهاية
```

{ تكرار كتابة جدول الضرب k مادام k<=10 }

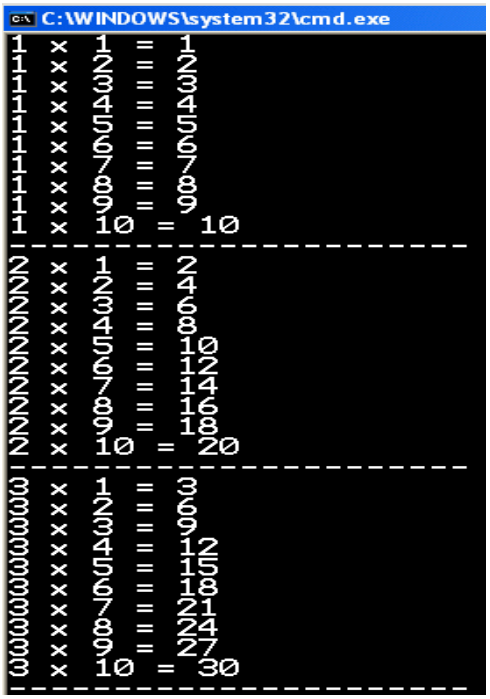
```
k ← 1
مادام ( k ≤ 10 ) كرر
   بداية
      { كتابة جدول الضرب k }
      k ← k +1
   نهاية
```

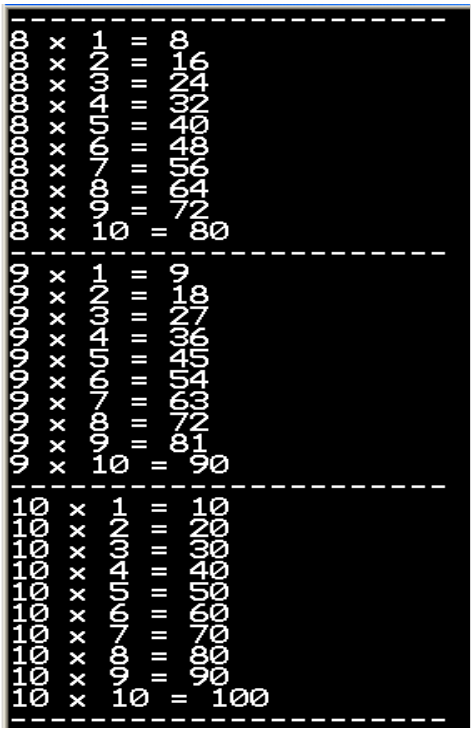
لاحظ فكرة البرمجة المهيكلة

```
using System;
namespace ExampleWhile
{
class MultiplyAll
{
public static void Main(string[] arg)
{
    int k, i;

    k = 1;
    while (k <= 10)
    {
        // Multiplication Table of number k
        i = 1;
        while (i <= 10)
        {
            Console.WriteLine(k + " x " + i + " = " + k *
i);
            i = i + 1;
        }
        Console.WriteLine("----------------------");
        k = k + 1;
    }
}
}
}
```

لاحظ فكرة البرمجة المهيكلة: استخدام تعليمة while داخل كتلة تعليمات while.

مقاطع من شاشة التنفيذ:





10. تدريب

حل معادلات من الدرجة الثانية.

$a.x^2 + b.x + c = 0$ ،

دخل البرنامج: ثلاث أعداد حقيقية هي a، b، c تقابل حدود المعادلة. $a \neq 0$

خرج البرنامج: إحدى الحالات التالية:

1. رسالة تعلن عدم وجود حلّ حقيقي.
2. رسالة تعلن عن وجود حل وحيد مع إظهار الحلّ.
3. رسالة تعلن عن وجود حلين مع إظهار الحلّين.

النص البرمجي:

```
using System;
namespace Examples
{
class Solution2degree
{// Solution of second degree equation : a.x2 + b.x + c = 0
    public static void Main(string[] arg)
    {
        double a; String Sa;
        double b; String Sb;
        double c; String Sc;
        double delta, x, x1, x2;

        // Read a,b, c
        Console.Write(" Input a =");
        Sa = Console.ReadLine(); a = Double.Parse(Sa);
        Console.Write(" Input b =");
        Sb = Console.ReadLine(); b = Double.Parse(Sb);
        Console.Write(" Input c =");
        Sc = Console.ReadLine(); c = Double.Parse(Sc);

        delta = b * b - 4 * a * c;

        //  compute the equation roots
        if (delta < 0)
            Console.WriteLine(" No Real Solution ");

        if (delta == 0)
        {
            Console.WriteLine(" One Solution ");
            x = -b / (2 * a);
            Console.WriteLine(" x= " + x);
        }

        if (delta > 0)
        {
            Console.WriteLine(" Two Solutions ");
            x1 = (-b - Math.Sqrt(delta)) / (2 * a);
            x2 = (-b + Math.Sqrt(delta)) / (2 * a);
            Console.WriteLine(" x1= " + x1 + " x2= " + x2);
        }

    }
}
}
```

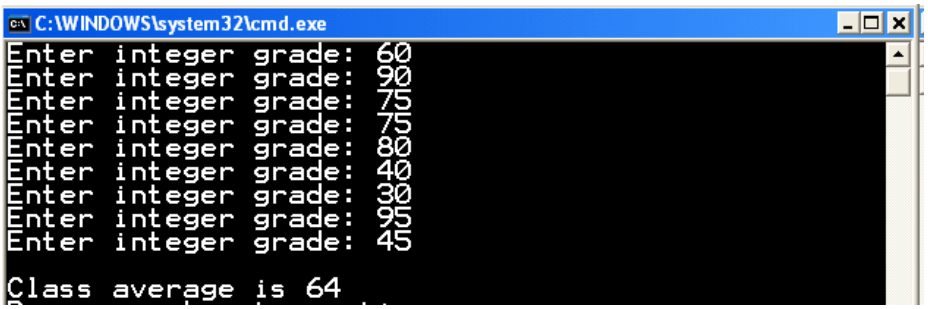
11. تمرين-1

اكتب صف Average1 بلغة C#، يساعد في حساب المتوسط الحسابي لعشرة أرقام صحيحة يجري طلبها من البرنامج وإدخالها من قبل المُستَخدِم.

الحل:

```
using System;
namespace Average
{
   class Average1
   {
      static void Main( string[] args )
      {
         int total,         // sum of grades
            gradeCounter,   // number of grades
entered
            gradeValue,     // grade value
            average;        // average of all grades

         // initialization phase
         total = 0;          // clear total
         gradeCounter = 1;   // prepare to loop

         // processing phase
         while ( gradeCounter <= 10 )  // loop 10
times
         {
            // prompt for input and read grade from
user
            Console.Write( "Enter integer grade: "
);
            // read input and convert to integer
            gradeValue = Int32.Parse(
Console.ReadLine() );
            // add gradeValue to total
            total = total + gradeValue;
            // add 1 to gradeCounter
            gradeCounter = gradeCounter + 1;
         }

         // termination phase
         average = total / 10;  // integer division
         // display average of exam grades
         Console.WriteLine( "\nClass average is "+
average );

      } // end Main

   } // end class Average1
}
```

تنفيذ البرنامج:



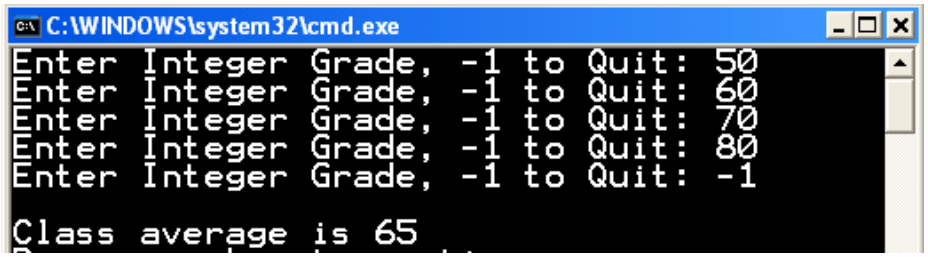
12. تمرين-2

اكتب برنامجاً بلغة C#، يساعد في حساب المتوسط الحسابي لعدد من الأرقام صحيحة يجري طلبها من البرنامج وإدخالها من قبل المُستَخدِم وبحيث يتوقف طلب الأعداد عند إدخال المستخدم للقيمة 1-.

الحلّ:

```
using System;
namespace Average
{
class Average2
{
    static void Main(string[] args)
    {
        int total,           // sum of grades
           gradeCounter,    // number of grades entered
           gradeValue;      // grade value
        double average;      // average of all grades

        // initialization phase
        total = 0;           // clear total
        gradeCounter = 0;    // prepare to loop

        // processing phase
        // prompt for input and convert to integer
        Console.Write("Enter Integer Grade, -1 to Quit: ");
        gradeValue = Int32.Parse(Console.ReadLine());

        // loop until a -1 is entered by user
        while (gradeValue != -1)
        {
            // add gradeValue to total
            total = total + gradeValue;

            // add 1 to gradeCounter
            gradeCounter = gradeCounter + 1;

            // prompt for input and read grade from user
            // convert grade from string to integer
            Console.Write("Enter Integer Grade, -1 to Quit: ");
            gradeValue = Int32.Parse(Console.ReadLine());
        } // end while

        // termination phase
        if (gradeCounter != 0)
        {
            average = (double)total / gradeCounter;
            // display average of exam grades
            Console.WriteLine("\nClass average is "+ average);
        }
        else
        {
            Console.WriteLine("No grades were entered.");
        }

    } // end method Main
} // end class Average2
}
```

تنفيذ البرنامج:



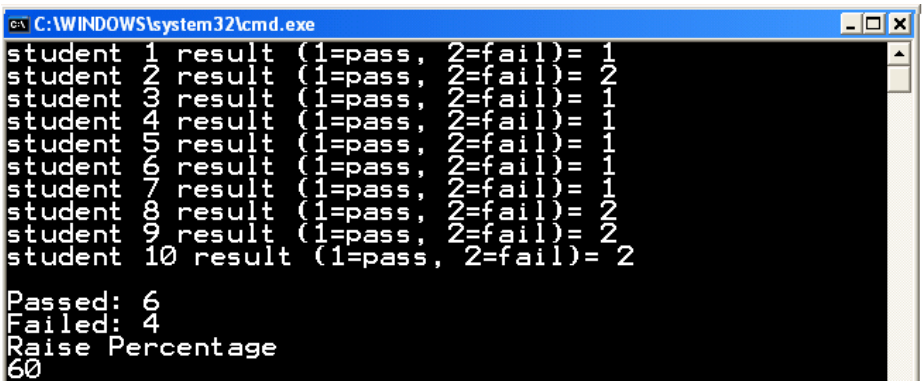
13. تمرين-3

بفرض أن لديك مجموعة من 10 طلاب، اكتب برنامجاً بلغة C# يستعرض أرقام الطلاب من 1 إلى 10 رقماً رقماً، بحيث يقوم المستخدم من أجل كل طالب بإدخال رقم 1 في حال كان الطالب ناجحاً، وإدخال رقم 2 في حال كان الطالب راسباً، وبحيث يعطي البرنامج في النهاية عدد الناجحين وعدد الراسبين والنسبة المئوية للنجاح.

الحل:

```
using System;

namespace Passes_and_Failures
{
class Analysis
{
static void Main(string[] args)
{
    int passes = 0,            // number of passes
       failures = 0,          // number of failures
       student = 1,           // student counter
       result;                // one exam result

    // process 10 students; counter-controlled loop
    while (student <= 10)
    {
        Console.Write("student "+student+" result (1=pass, 2=fail)= ");
        result = Int32.Parse(Console.ReadLine());

        if (result == 1)
            passes = passes + 1;

        else
            failures = failures + 1;

        student = student + 1;
    }

    // termination phase
    Console.WriteLine();
    Console.WriteLine("Passed: " + passes);
    Console.WriteLine("Failed: " + failures);

    Console.WriteLine("Raise Percentage\n" + passes * 10);

} // end of method Main

} // end of class Analysis

}
}
```

تنفيذ البرنامج:



14. تمرين-4

جرّب البرنامج التالي وأبد ملاحظاتك على عمليات الزيادة بقيمة 1.

```
using System;

namespace Increment
{
   class Increment
   {
      static void Main(string[] args)
      {
         int c;

         c = 5;
         Console.WriteLine( c );    // print 5
         Console.WriteLine( c++ ); // print 5 then
postincrement
         Console.WriteLine( c );    // print 6

         Console.WriteLine();       // skip a line

         c = 5;
         Console.WriteLine( c );    // print 5
         Console.WriteLine( ++c ); // preincrement
then print 6
         Console.WriteLine( c );    // print 6s

      } // end of method Main

   } // end of class Increment
                                                    }
```

15. تمرين-5

اكتب برنامج بلغة C# لحساب مجموع الأعداد الزوجية المحصورة بين 0 و 100.

الحلّ:

```
using System;

namespace MySum
{
class Sum
{
    static void Main(string[] args)
    {
       int sum = 0;
       int number = 2;
       while ( number <= 100)
        {
            sum += number;
            number += 2;
        }
        Console.WriteLine("The sum is " + sum +
                          " Sum Even Integers from 2 to
100");

    } // end of method Main

} // end of class Sum
}
```

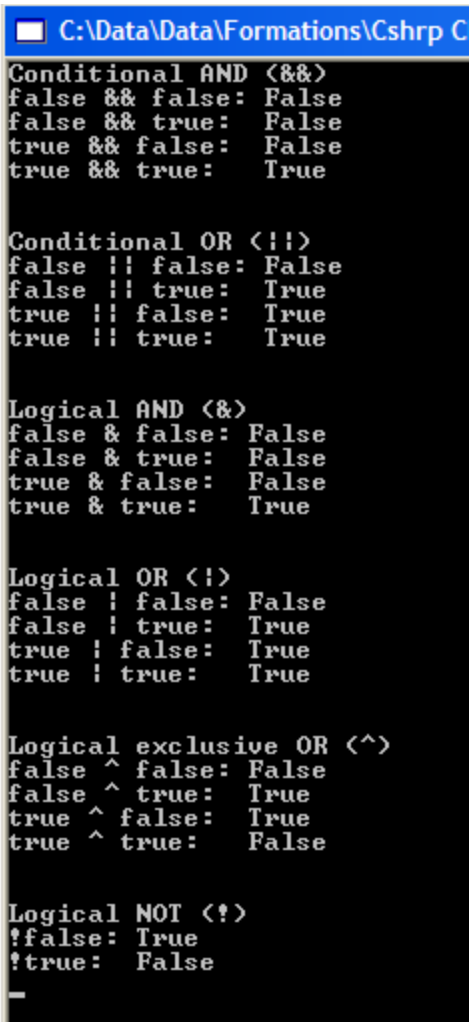
نتيجة التنفيذ:

16. تمرين

نفذ البرنامج التالي وأعط نتيجته:

```
using System;
namespace LogicalOperators
{
   class LogicalOperators
   {
     static void Main(string[] args)
      {
         // testing the conditional AND operator (&&)
         Console.WriteLine( "Conditional AND (&&)" +
            "\nfalse && false: " + ( false && false ) +
            "\nfalse && true:  " + ( false && true ) +
            "\ntrue && false:  " + ( true && false ) +
            "\ntrue && true:   " + ( true && true ) );

         // testing the conditional OR operator (||)
         Console.WriteLine( "\n\nConditional OR (||)" +
            "\nfalse || false: " + ( false || false ) +
            "\nfalse || true:  " + ( false || true ) +
            "\ntrue || false:  " + ( true || false ) +
            "\ntrue || true:   " + ( true || true ) );

         // testing the logical AND operator (&)
         Console.WriteLine( "\n\nLogical AND (&)" +
            "\nfalse & false: " + ( false & false ) +
            "\nfalse & true:  " + ( false & true ) +
            "\ntrue & false:  " + ( true & false ) +
            "\ntrue & true:   " + ( true & true ) );

         // testing the logical OR operator (|)
         Console.WriteLine( "\n\nLogical OR (|)" +
            "\nfalse | false: " + ( false | false ) +
            "\nfalse | true:  " + ( false | true ) +
            "\ntrue | false:  " + ( true | false ) +
            "\ntrue | true:   " + ( true | true ) );

         // testing the logical exclusive OR operator (^)
         Console.WriteLine( "\n\nLogical exclusive OR (^)" +
            "\nfalse ^ false: " + ( false ^ false ) +
            "\nfalse ^ true:  " + ( false ^ true ) +
            "\ntrue ^ false:  " + ( true ^ false ) +
            "\ntrue ^ true:   " + ( true ^ true ) );

         // testing the logical NOT operator (!)
         Console.WriteLine( "\n\nLogical NOT (!)" +
            "\n!false: " + ( !false ) +
            "\n!true:  " + ( !true ) );

      }
   }
}
```

الحلّ:


C:\Data\Data\Formations\Cshrp C
Conditional AND (&&)
false && false: False
false && true: False
true && false: False
true && true: True
Conditional OR (||)
false || false: False
false || true: True
true || false: True
true || true: True
Logical AND (&)
false & false: False
false & true: False
true & false: False
true & true: True
Logical OR (|)
false | false: False
false | true: True
true | false: True
true | true: True
Logical exclusive OR (^)
false ^ false: False
false ^ true: True
true ^ false: True
true ^ true: False
Logical NOT (!)
!false: True
!true: False

17. مسائل للحلّ خوارزمياً، ومن ثم بلغة C#

مسألة 1:

نريد كتابة برنامج لحلّ للمعادلة من الشكل $a.x^2+b.x+c=0$.

مهما تكن قيم الحدود a,b,c. أي يمكن أن تكون a=0 وبالتالي لن تكون معادلة درجة ثانية....

(فكّر بالخوارزمية أولاً ومن ثم اكتب البرنامج بعد وضعها بلغة الخوارزميات)

الحلّ:

```
using System;
namespace Examples
{
class Solution2degreeAll
{ // Solution of second degree equation : a.x2 + b.x + c = 0
    public static void Main(string[] arg)
    {
        double a; String Sa;
        double b; String Sb;
        double c; String Sc;
        double delta, x, x1, x2;

        // Read a,b, c
        Console.Write(" Input a =");
        Sa = Console.ReadLine(); a = Double.Parse(Sa);
        Console.Write(" Input b =");
        Sb = Console.ReadLine(); b = Double.Parse(Sb);
        Console.Write(" Input c =");
        Sc = Console.ReadLine(); c = Double.Parse(Sc);

        if (a == 0)
        {
            if (b == 0)
            {
                if (c == 0)
                {
                    Console.WriteLine("Each Real is a solution");
                }
                else
                {// c ≠ 0
                    Console.WriteLine("Incorrect Equation");
                }
            }
            else
            { // b ≠ 0
                x = -c / b;
                Console.WriteLine("The Solution is : " + x);
            }
```

```
        }
        else
        { // a ≠ 0

            delta = b * b - 4 * a * c;

            if (delta < 0)
                Console.WriteLine(" No Real Solution ");

            if (delta == 0)
            {
                Console.WriteLine(" One Solution ");
                x = -b / (2 * a);
                Console.WriteLine(" x= " + x);
            }

            if (delta > 0)
            {
                Console.WriteLine(" Two Solutions ");
                x1 = (-b - Math.Sqrt(delta)) / (2 * a);
                x2 = (-b + Math.Sqrt(delta)) / (2 * a);
                Console.WriteLine(" x1= " + x1 + " x2= " + x2);
            }

        }
    }
   }
}
```

مسألة 2:

نقول عن عدد أنه perfect إذا كان يساوي حاصل جمع جميع قواسمه بما فيهم الواحد. اكتب برنامج بلغة C# يأخذ عدد n ويبحث عن أول n عدد Perfect من مجموعة الأعداد الطبيعية. (فكّر بالخوارزمية أولاً ومن ثم اكتب البرنامج بعد وضعها بلغة الخوارزميات)

الحلّ:

```
using System;
namespace Perfect
{
      class ApplicationPerfect
      {
            static void Main (string[ ] args)
            {
                  int compt = 0, n, k, sumdiv, nbr;
                  Console.Write("Number of Perfect numbers you wish find : ");

                  n = Int32.Parse( Console.ReadLine( ) ) ;
                  nbr = 2;

                  while (compt != n)
                  {
                        sumdiv = 1;
                        k = 2;

                        while(k <= nbr/2 )
                        {
                              if (nbr % k == 0)
                                          sumdiv += k ;

                              k++;
                        }

                        if (sumdiv == nbr)
                        {
                              Console.WriteLine(nbr+" is a Perfect number");
                              compt++;
                        }
                        nbr++;
                  }
            }
      }
}
```

الفصل الرابع:
تتمات في لغة الخوارزميات

المخطط:

1. تعليمات التحكم المشتقة من التعليمات الأساسية
2. التعليمة التكرارية بعدّاد for
3. تعليمة التكرار بعدًّاد في C#
4. التعليمة التكرار بعدّاد for في C ,C#
5. أمثلة عن تعليمةfor
6. التعليمة التكرارية: كرّر_مرة_واحدة_على_الأقل
7. مثال: التكرار_لمرة_واحدة_على_الأقل
8. مثال: نص برمجي do { } while
9. كسر البرمجة المهيكلة
10. تعليمات كسر البرمجة المهيكلة في لغات البرمجة
11. مثال تعليمة break
12. مثال تعليمة break ضمن كتلة تعليمات for
13. التعليمة continue في C, C#
14. تعليمة التفريع (التعليمة الشرطية متعددة الاختيار)
15. تعليمة التفريع: switch … case
16. أمثلة برمجية switch case
17. تمارين

1. تعليمات التحكم المشتقة من التعليمات الأساسية

يمكن أن نعبِّر عن بعض تعليمات التحكم الأساسية (الشرطية والتكرارية) بشكل مختلف يصف ماهية العمل الذي نقوم به "خوارزمياً"، أو ما نفكر به في خطوات الحل، وصفاً أدق وأسهل من تعبيرنا بالتعليمات الأساسية. هذا الشكل المختلف للتعبير عن التحكم بمسار التنفيذ قد نجده مباشرةً في أغلب لغات البرمجة. مثال ذلك، تعليمة for (ولنسميها تعليمة تكرارية بعدّاد). هذه التعليمة تجد تعليمةً مكافئةً لها في كل لغات البرمجة الإجرائية (حتى أنها وُجدَت في لغة Fortran حين لم تكن تعليمة while موجودة فيها).

لكنك تسأل هنا، ألم نقل أن التعليمات الأساسية الخمسة كافية للتعبير عن خوارزميتنا والتحكم بمسارات تنفيذ الحل. نعم، يبقى ما ذكرناه صحيحاً : إن تعليمات التحكم الأساسية (الشرطية والتكرارية) كافية للتعبير عن مسار الحل في إي من خوارزميتنا، ولذلك سنرى أن جميع هذه البنى الجديدة للتحكم، التي سنطرحها هنا، جرى اشتقاقها من تعليمات التحكم الأساسية (الشرطية والتكرارية). ولكننا نشجع على استخدام تعليمات البنى الجديدة عند اللزوم، فهي تسهِّل التعبير عن حلنا، وتزيد من وضوح القراءة لخوارزميتنا و برامجنا.

2. التعليمة التكرارية بعدّاد for

لاحظنا في الكثير من خوارزمياتنا، حين استخدام التكرار:
(مادام <شرط> كرِّر <مجموعة تعليمات>)، أن تكرار التنفيذ مشروط بقيمة عدّاد
(متحول عددي صحيح تتزايد قيمته خطوة خطوة) تقع بين قيمة بدائية وقيمة نهائية.

ليكن counter "العدّاد" (ويمكننا تسميته متحول التكرار).
وليكن لدينا تجمع التعليمات التالي:

<القيمة 1> ← counter
ما دام (counter أصغر أو يساوي < القيمة 2>) **كرِّر**

<مجموعة تعليمات>
counter ← counter + 1

لاحظ أن التكرار يبدأ مع قيمة العدَّاد عند < القيمة 1> ويتوقف التكرار عند < القيمة 2>

نعبِّر عن ذلك كما يلي:

من أجل قيم counter تبدأ من <القيمة 1> وتنتهي إلى < القيمة 2> كرِّر

<مجموعة تعليمات>

نكتب ذلك على نحوٍ مهيكل أسوةً ببقية تعليماتنا المهيكلة الأخرى:

من أجل <قيمة1> ← counter حتى <قيمة 2> كرِّر

<مجموعة تعليمات>

إذن، تعمل التعليمة التكرارية بعدّاد كما يلي:

مادامت قيمة العدّاد counter أصغر أو تساوي القيمة العليا لمجال التكرار < القيمة 2> تُنفَّذ <مجموعة تعليمات>.

مع كل تكرار يزداد العدّاد آليا خطوة، أي تصبح قيمة counter تساوي counter+1، تُختبَر القيمة الجديدة إن كانت ضمن مجال التكرار، تنفَّذ <مجموعة تعليمات>. من جديد، يزداد العدّاد مع كل تنفيذ.

إذن يجري تكرار مايلي:

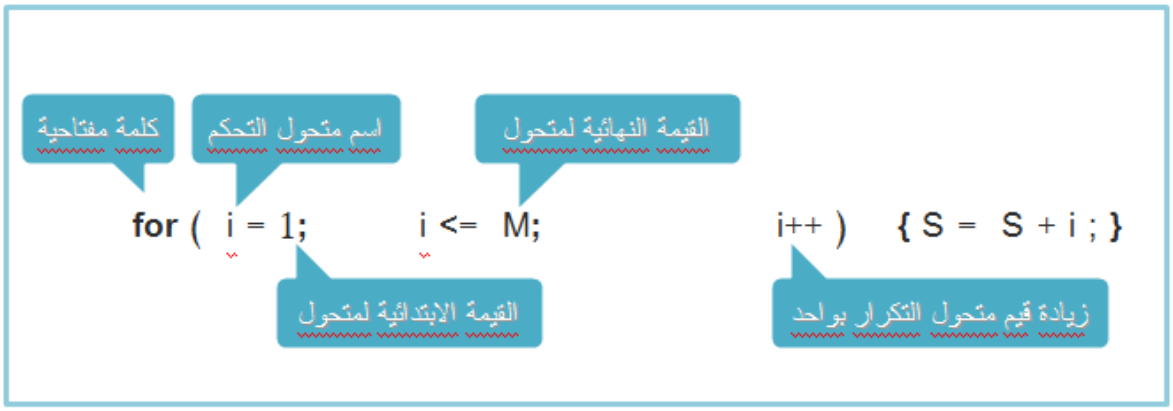
1. اختبار قيمة العدّاد.
2. تنفيذ التعليمات.
3. زيادة العدّاد مع كل تكرار حتى يصل العدّاد نهاية مجال التكرار أي القيمة <القيمة 2>.

ملاحظات:

- إن شكل التعليمة التكرارية بعدّاد، عدا عن تعبيره الواضح أن التكرار مشروط بقيمة متحول–تكرار مُحَّدد المجال، فإنه يُلزمنا بعدم نسيان إعطاء القيمة البدائية لهذا المتحول، ويضمن لنا زيادة هذا المتحول (العداد) خطوة، والذي كان من الممكن أن نسيانه في الحلقة التكرارية العادية، وندخل التنفيذ في حلقة تكرار لا نهائية
- ينصح بشدة استخدام هذه الحلقة التكرارية إن كان التكرار مشروطاً بعدّاد يتزايد خطوة خطوة مع كل تكرار. وينصح بشدة عدم تغيير قيمة العدّاد (متحول–التكرار) ضمن مجموعة التعليمات
- إن لم نغيّر قيمة متحول–التكرار خلال تكرار تنفيذ مجموعة التعليمات(وهذا ما نصحنا به)، وإن تُرِك تكرار التنفيذ يجري بشكل طبيعي بلا إنقطـاع، فستكون قيمة العدّاد (متحول–التكرار) يساوي القيمة <القيمة 2> +1، عند انتهاء هذه التعليمة التكرارية

3. تعليمة التكرار بعدَّاد في C#

تقوم تعليمة for بدور التكرار بعدّاد، وهذه تعليمة ذات عمومية أكبرفي C, C#، ولكننا سنرى هنا استخدامها للتكرار بعدّاد خطوة خطوة:



الأجزاء المؤلفة لتعليمة التكرار for (حالة تكرار بعدّاد يزداد1)

هناك العديد من الخوارزميات التي كتبناها نستطيع الآن كتابتها بإيجاز باستخدام التعليمة التكرارية بعدّاد.

مثال:

حساب مجموع الأعداد 1, 2, 3,...M

حيث M تُعطى كدخل.

خوارزمية الحل السابقة باستخدام تعليمة التكرار الأساسية

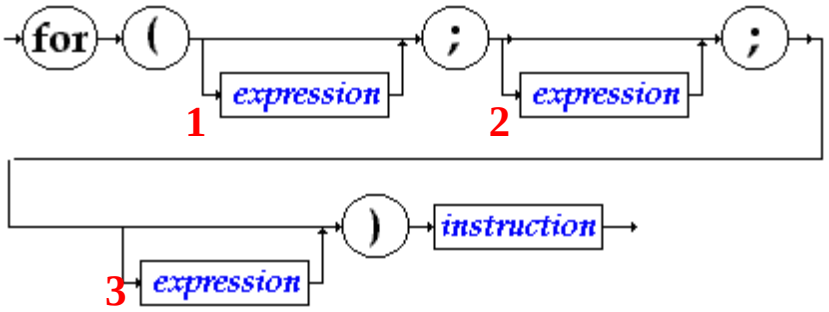
```
اقرأ M
S ← 0
i ← 1
مادام ( i ⩽ M ) كرِّر
    S ← S + i
    i ← i +1
اكتب S
```

خوارزمية الحل باستخدام تعليمة التكرار بعدّاد

```
اقرأ M
S ← 0
من أجل i ← 1 حتى M كرِّر
    S ← S + i
اكتب S
```

4. التعليمة التكرار بعدّاد for في C ,C#

يكون للتعليمة التكرار بعدّاد for الشكل القواعدي التالي:



- يجري أولا تنفيذ عبارة الإعداد expression المُشار إليها بالرقم 1
- يجري بعدها اختبار الشرط الموجود في العبارة الشرطية expression المُشار إليها بالرقم 2:
  - فإذا كان الشرط صحيحاً نبدأ في تنفيذ instruction.
  - وإلا يجري التوقف.
- بعد كل تنفيذ للتعليمات instruction، يجري تنفيذ التعليمات الموجودة في عبارة التعديل expression المُشار إليها بالرقم 3 وإعادة اختبار الشرط الموجود في العبارة الشرطية expression المُشار إليها بالرقم 2 للتأكد من أن قيمتها المنطقية مازالت صح:
  - فإذا كانت صح نستمر في تكرار instruction مرة أخرى.
  - وإلا يجري التوقف عن تكرار instruction.

مثال:

```
int z=0;

for (k=0; k<=10; k++)
     z=z+10;
```

5. أمثلة عن تعليمة for

مثال1: حساب مجموع الأعداد 1, 2, 3,...M حيث M تُعطى كدخل.

```
using System;
namespace Instructions
{
class SumMfor
{
    // Compute the Sum 1+2+.....M  using (for) statement
    public static void Main(string[] arg)
    {

        int M; String SM;
        int s, i;

        // Read M
        Console.Write(" Input M : ");
        SM = Console.ReadLine(); M = Int32.Parse(SM);

        // variables initialisation
        s = 0;

        //  compute the Sum in s
        for (i = 1; i <= M; i++)
        {
            s = s + i;
        }

        // Write s
        Console.WriteLine("1+2+...+"+M+" = "+s);
    }
}
}
```

التنفيذ:

يمكنك التحقق من صحة البرنامج: رياضياً $1+2+..+M = M*(M+1)/2$

مثال2: حساب ! M حيث M تُعطى كدخل.

M! =1*2*3…*M

نعرف أن العاملي لـ M وتكتب !M تعرف بالعلاقة السلبقة، إذن هي "مضروب" الأعداد من 1 حتى M.

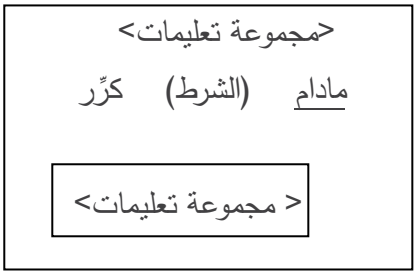
إذن نستخدم نفس النص البرمجي السابق، فقط نستبدل الجمع + بالضرب * ولاننسى القيمة البدائية لـ s هي 1 بدل 0.

```
using System;
namespace Instructions
{
class FactMfor
{
    // Compute the M!  using (for) statement
    public static void Main(string[] arg)
    {
        int M; String SM;
        int s, i;

        // Read M
        Console.Write(" Input M : ");
        SM = Console.ReadLine(); M = Int32.Parse(SM);

        // variables initialisation
        s = 1;

        //  compute the M! in s
        for (i = 1; i <= M; i++)
        {
            s = s * i;
        }

        // Write s
        Console.WriteLine(" "+M+"! = "+s);
    }
}
}
```

التنفيذ:

6. التعليمة التكرارية: كرّر_مرة_واحدة_على_الأقل

عندما تأخذ التعليمة التكرارية مادام ... كرر الإطار التالي: (1)



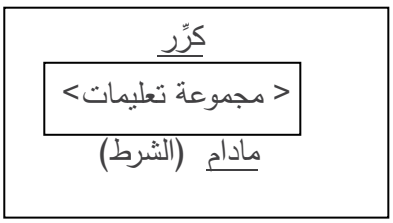
نرى هنا أننا ننفذ <مجموعة تعليمات> ثم نختبر صحة (الشرط) لنتابع تكرار تنفيذ نفس <مجموعة تعليمات> مادام (الشرط) صحيحاً.

إذن ستُنفَّذ < مجموعة تعليمات> مرة واحدة على الأقل ، ثم يتكرر تنفيذ <مجموعة تعليمات> 0 مرة أو أكثر حتى يصبح (الشرط) غير صحيح.

يمكن قراءة السابق كما يلي:

كرّر تنفيذ < مجموعة تعليمات> **مادام** (الشرط) مُحقَّق.

ويمكن أن نعطي ذلك الإطار التالي: (2)



إن الإطار (1) و(2) متكافئان، وكلاهما يجمع فكرة التنفيذ لِـ <مجموعة تعليمات> أولاً، ثم تكرار التنفيذ لهذه المجموعة ولكن بشرط.

إن التعبير "الخوارزمي" في الإطار (2) أكثر إيجازاً، وقد يكون أوضح تعبيراً.

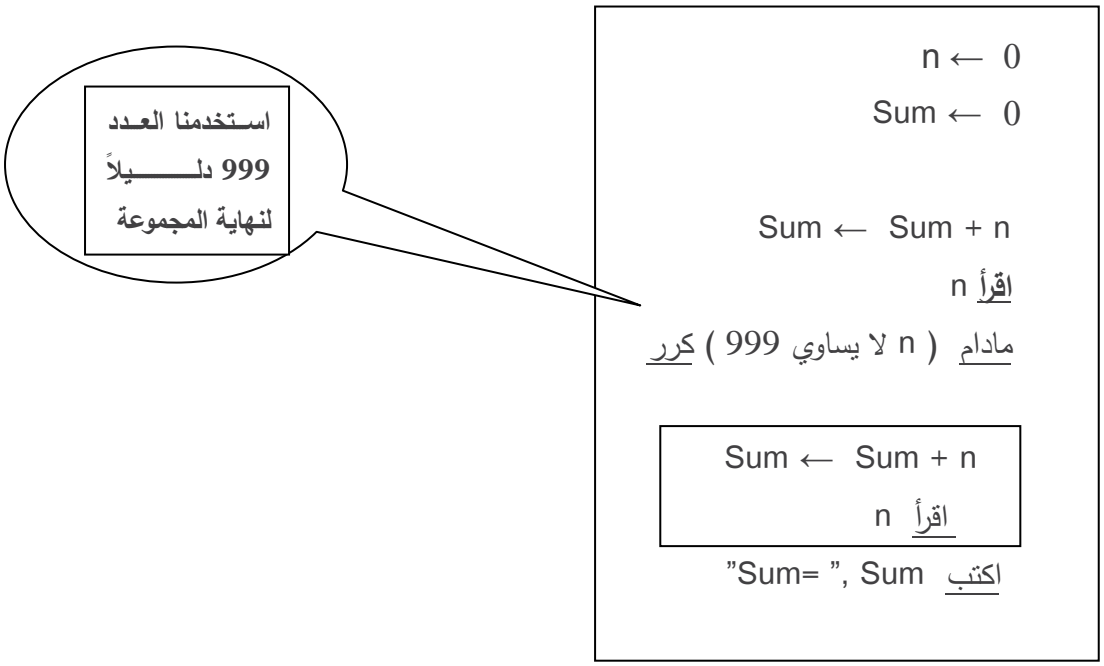
والفكرة الأساسية هي تكرار تنفيذ مجموعة تعليمات لمرة واحدة على الأقل (ثم اختبار الشرط) ثم تكرار التنفيذ مادام الشرط صحيحاً.

تنَّفذ <مجموعة تعليمات> أول مرة، يتكرر تنفيذ <مجموعة تعليمات> مادام الشرط صحيحاً.

7. مثال: التكرار_لمرة_واحدة_على_الأقل

حساب المجموع لمجموعة من الأعداد الصحيحة يجري إدخالها للبرنامج، ينتهي الإدخال بالقيمة 999.

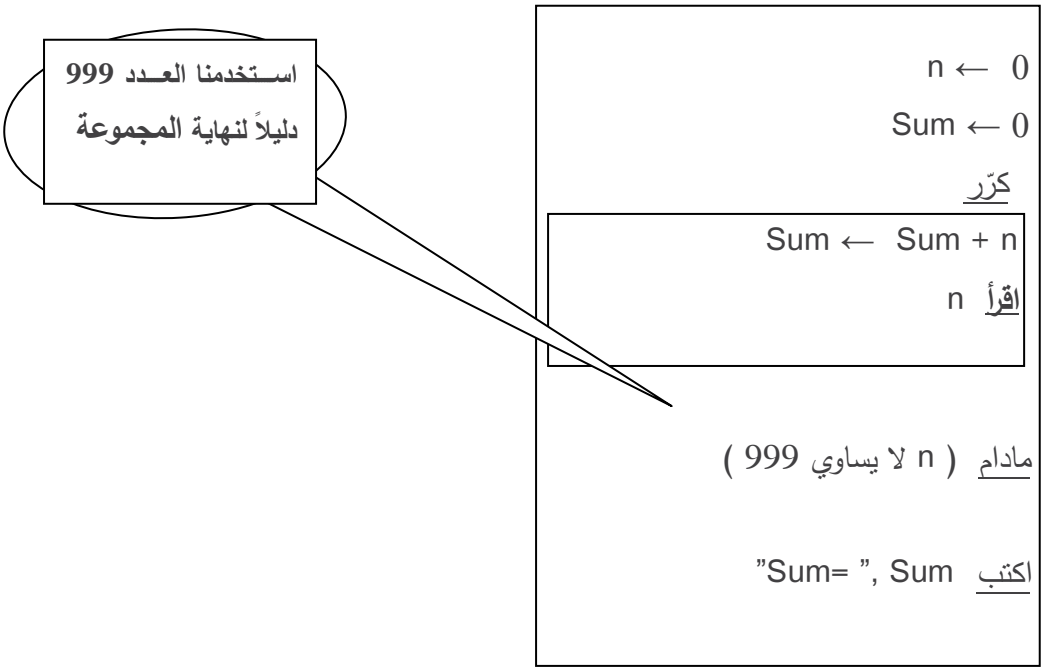
خوارزمية الحل باستخدام التعليمة التكرارية الأساسية:



ملاحظة:

للانتظام بالتعليمات، أعطينا جميع المتحولات القيم البدائية 0، تعليمة الإسناد Sum ← Sum + n قبل التكرار، يمكن الاستغناء عنها، لأن قيمة Sum قبل وبعد الإسناد هي 0. ولكننا تركناها للانتظام فهي تشكل مع تعليمة القراءة، مجموعة التعليمات المطلوب تكرار تنفيذها لمرة واحدة على الأقل.

خوارزمية الحل السابقة باستخدام تعليمة التكرار لمرة واحدة على الأقل :

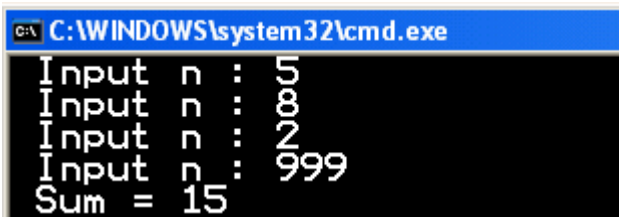


8. مثال: نص برمجي do { } while

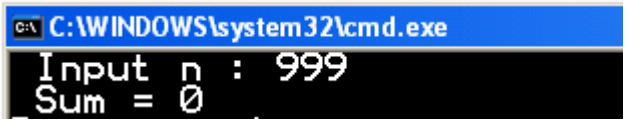
حساب المجموع لمجموعة من الأعداد الصحيحة يجري إدخالها للبرنامج، ينتهي الإدخال بالقيمة 999.

```
using System;
namespace Instructions
{
    class SumInDoWhile
    {  // compute Sum of user input (end value = 999)

        public static void Main(string[] arg)
        {

            int n = 0; String Sn;
            int sum = 0;

            do
            {
                sum = sum + n;
                // Read n
                Console.Write(" Input n : ");
                Sn = Console.ReadLine();
                n = Int32.Parse(Sn);
            }
            while (n != 999);

            Console.WriteLine(" Sum = " + sum);
        }
    }
}
```

التنفيذ:



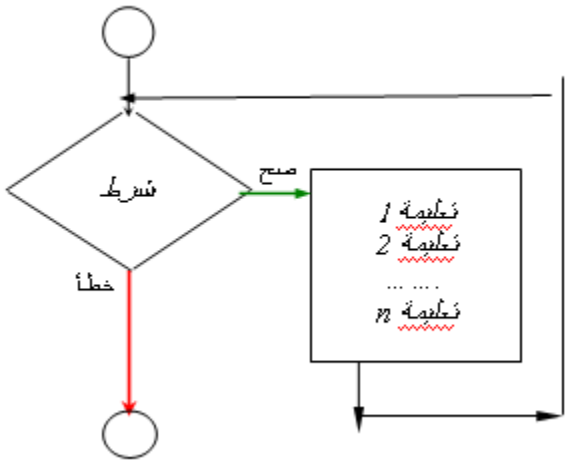
المجموعة الخالية !



9. كسر البرمجة المهيكلة:

منذ الخطوات الأولى، في تعليمك البرمجة، أكدنا أن تدريس أسس البرمجة، استقر باستخدام البرمجة المهيكلة structured programming، فلكل تعليمة تحكم هيكلٌ يتضمن كتلةَ تعليماتٍ تنفذها التعليمة، وفي كتلة_التعليمات تُعطى كل تعليمة في الكتلة هيكل.....وهكذا.

إن الميزة الأهم في البرمجة المهيكلة أن لكل تعليمة بداية واحدة ونهاية واحدة، مما يسمح بوضوح مسار تنفيذ البرنامج ومتابعته.
فمثلاً في البنية التالية للتعليمة التكرارية، إذا دخلنا لتنفيذ كتلة التعليمات فلا نخرج إلا مع تنفيذ آخر تعليمة في الكتلة أي تعليمة n. ولايمكن أن نخرج من الكتلة من أي تعليمة أخرى.



ولكن علي ألا أخفي عليك يا صديقي أننا لا نحب الإنتظام دوماً، ونقول للضرورة أحكام.
وهكذا، أيضاً في البرمجة، يرى البعض أن البرمجة المهيكلة أمر جيد وفوائدها التي عددناها سابقاً مُتفق عليها، ولكنها ليس مقدسة بحيث لا يمكن تجاوزها في حالات معينة استثنائية.

وهكذا، غالباً تتيح لغات البرمجة المهيكلة تعليمات تحكُّم بإمكانها كسر قواعد البرمجة المهيكلة بحيث تسمح بالخروج من كتلة تعليمات قبل انتهائها. ولكن يُنصح دوماً باستخدامها في الحالات الاستثنائية فقط.

10. تعليمات كسر البرمجة المهيكلة في لغات البرمجة:

كنا قد نوهنا أن لغات البرمجة المهيكلة هي التي تقدم تعليمات التحكم الأساسية التي عرفناها في لغة الخوارزميات، والتي تكفي كاملاً للتعبير عن مسار تنفيذ البرنامج بحيث نستغني تماماً عن تعليمة القفز (GOTO) التي كانت هي الأساس للتحكم في الأجيال الأولى للغات البرمجة عالية المستوى. إلا أن العديد من لغات البرمجة المهيكلة قد احتفظت بها لتبقى متوافقة مع الإصدارات السابقة، أو وضعت ما يكافؤها الذي يسمح فقط بالخروج إلى مابعد نهاية كتلة التعليمات المهيكلة.

كسر البرمجة المهيكلة في C, C#:

تؤدي التعليمة break إلى الخروج إلى ما بعد نهاية كتلة تعليمات.

11. مثال تعليمة break:

إنه مثال بسيط يهدف فقط إلى توضيح استخدام التعليمة break للخروج من كتلة تعليمات.

المطلوب من البرنامج تكرار تنفيذ تعليمات مادامت قيم i أصغر أو يساوي قيمة معينة (القيمة 13 في مثالنا).

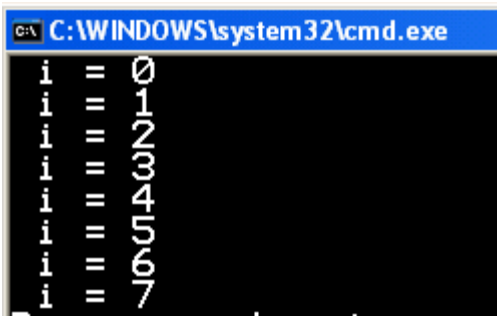
وللتبسيط أكثر في مثالنا، تقتصر التعليمات المطلوب تكرار تنفيذها على تعليمة إظهار قيمة i.

هذا هو المسار الأساسي لعمل برنامجنا. أما الاستثناء فهو قيم خاصة للمتحول i، نريد إيقاف تنفيذ التعليمات بسببها. القيم الخاصة في مثالنا هي i من مضاعفات 4. وما نقصده هنا بمضاعفات 4، هو أي عدد أكبر من 4، ويقسم 4 بلا باقي (باقي قسمته على 4 تساوي 0).

النص البرمجي:

```
using System;
namespace Instructions
{
class UseBreak
{
    public static void Main(string[] arg)
    {
        int i = 0, n = 13;

        while (i <= n)
        {
            if ((i % 4 == 0) && (i > 4))
                break;

            Console.WriteLine(" i = " + i);

            i = i + 1;
        }
    }
}
}
```

التنفيذ:



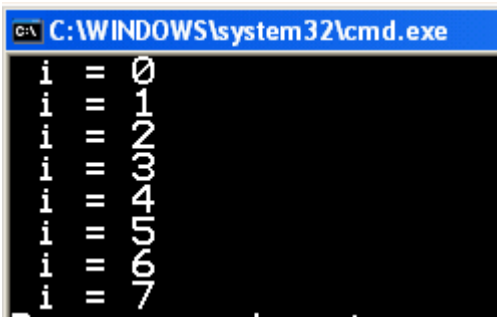
ملاحظات:

- لاحظ أن تكرار تنفيذ مجموعة/كتلة تعليمات while توقَّف عند القيمة 8، لأنها من مضاعفات 4، فقد وضعنا ذلك شرطاً ننفذ عنده تعليمة break للخروج إلى مابعد كتلة التعليمات.
- لاحظ أنه بإمكاننا تحقيق النتيجة نفسها بكتابة البرنامج دون استخدام تعليمة كسر الحلقة break، كيف؟ ببساطة لا نريد استمرار تكرار التنفيذ عندما يكون الشرط الذي استخدمناه لكسر الحلقة صحيحاً، نضيف ذلك كشرط إلى شرط تكرار الحلقة.

```
using System;
namespace Instructions
{
class UseBreakNo
{

public static void Main(string[] arg)
{
    int i = 0, n = 13;

    while ((i <= n) && !((i % 4 == 0) && (i > 4)))
    {
        Console.WriteLine(" i = " + i);

        i = i + 1;

       //if (( i % 4 == 0) && (i > 4))
       //     break;
    }
}
}
}
```

لاحظ وضعنا تعليمة الخروج من الكتلة كتعليق، ونقلنا شرطها إلى شرط التعليمة المهيكلة while

التنفيذ:



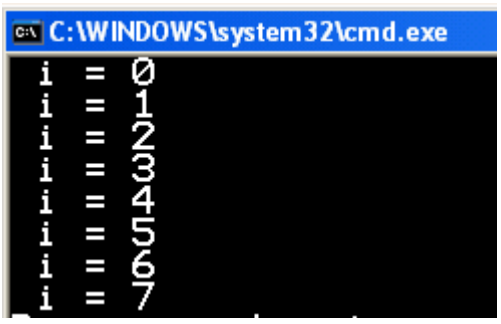
12. مثال تعليمة break ضمن كتلة تعليمات for:

- بالطبع يمكننا أيضاً استخدام التعليمة break لكسر (للخروج) من الحلقة التكرارية بعدّاد أي من كتلة تعليمات for (لا تنس التعليمة for مكافئة للتعليمة التكرارية while عندما الشرط متحول عدّاد).

- النص البرمجي:

```
using System;
namespace Instructions
{
    class UseBreakFor
    {
        public static void Main(string[] arg)
        {
            for (int i = 0; i <= 13; i++)
            {

                if ((i % 4 == 0) && (i > 4))
                    break;

                Console.WriteLine(" i = " + i);
            }
        }
    }
}
```

التنفيذ:



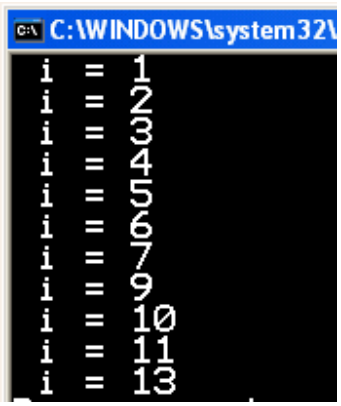
13. التعليمة continue في C, C#:

تُستخدم تعليمة continue للاستمرار في التنفيذ ضمن الحلقة التكرارية، ولكن بالانتقال إلى التكرار التالي. نريد القيام بنفس العمل الذي قمنا به عند استخدام break في الأمثلة السابقة، أي تجنب إظهار مضاعفات 4، ولكن أريد لبرنامجي ألا يتوقف عن التكرار، أي لا أريد الخروج من تعليمة التكرار، بل أريد منه الانتقال بالتنفيذ إلى التكرار التالي مباشرةً دون تنفيذ بقية كتلة التعليمات.

النص البرمجي:

```
using System;
namespace Instructions
{
class UseContinueFor
{
 public static void Main(string[] arg)
 {
    for (int i = 0; i <= 13; i++)
    {

        if ((i % 4 == 0) && (i > 4))
            continue;

        Console.WriteLine(" i = " + i);
    }
 }
}
}
```

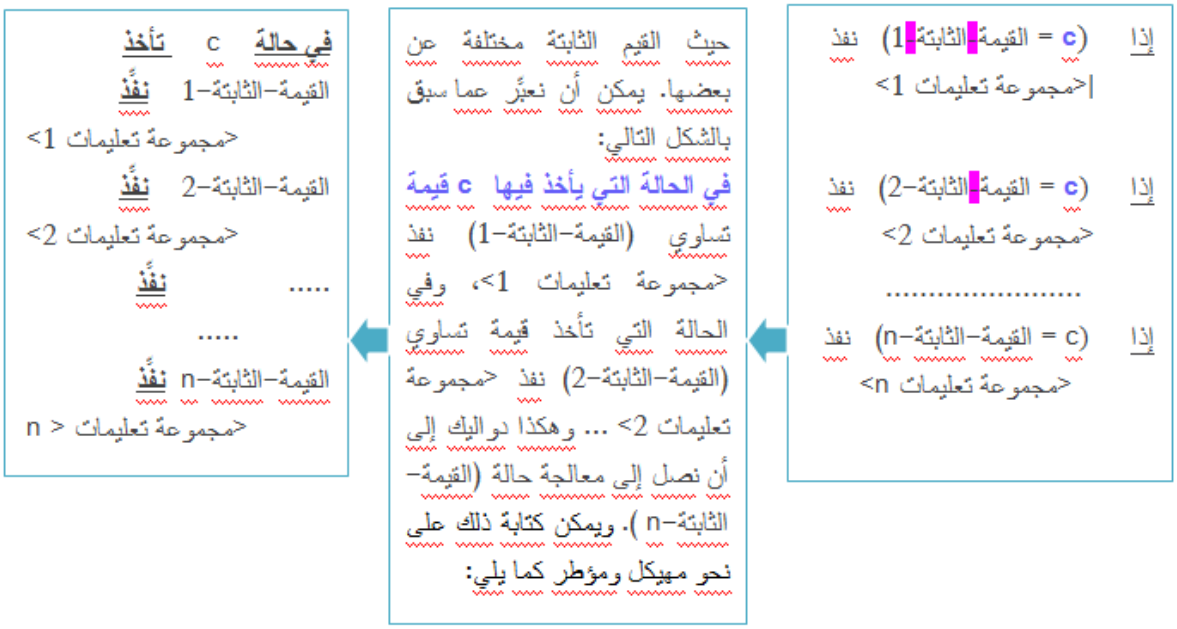
التنفيذ:



ملاحظات: لاحظ أن تنفيذ تعليمات الحلقة (إظهار قيمة i) جرى من أجل القيم من 1 حتى 7، ولكن عند القيمة 8 (مضاعفات 4) لم يتم التنفيذ/الإظهار بل تابعنا مباشرةً اختبار شرط التكرار نتيجة تنفيذ continue، هنا صارت i=9 ، شرط الحلقة صحيح، إذن ننفذ تعليمة الحلقة أي إظهار i، وهكذا يتكرر الإظهار للقيم 9، 10، 11 .ثم، تأخذ i القيمة 12(عدد من مضاعفات 4)، إذن تنفَّذ continue، فننتقل مباشرةً إلى اختبار شرط الحلقة مع القيمة 13...

14. تعليمة التفريع (التعليمة الشرطية متعددة الاختيار):

إذا كان c متحولاً نقرِّر وفقاً لقيمته اختيارات متعددة. أي كان لدينا تجميع التعليمات على الشكل التالي:



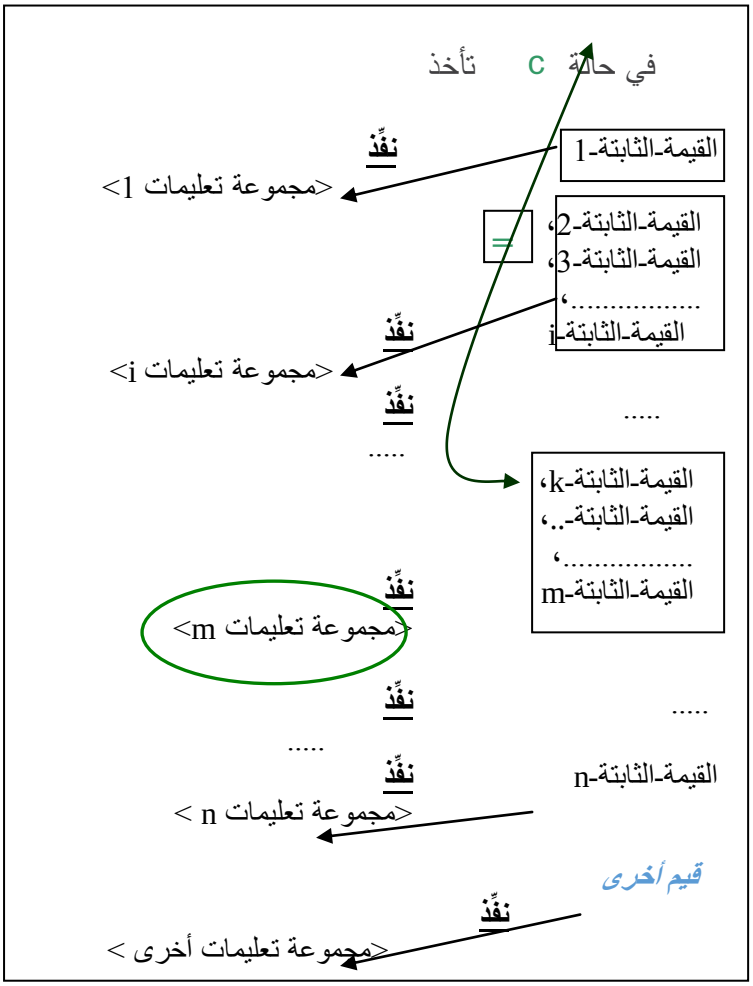
لاحظ أن التعليمة الشرطية هي تعليمة يتم وفقها اختيار مسار تنفيذي:

1. يمكننا تسمية التعليمة إذا "بالتعليمة الشرطية وحيدة الاختيار Single-selection structure لأنها تسمح باختيار أو تجاهل فعل وحيد.
2. يمكننا تسمية التعليمة إذا ..وإلا " بالتعليمة الشرطية مضاعفة الاختيار double-selection structure لأنها تسمح بالاختيار ما بين فعلين مختلفين.
3. في يمكننا تسمية تعليمة التفريع في حالة" بالتعليمة متعددة الاختيار multiple-selection structure لأنها تسمح باختيار فعل محدد من بين مجموعة ممكنة من الأفعال المختلفة، وذلك حسب قيمة "متحول" معين.

الشكل العام لتعليمة التفريع (الاختيار المتعدد):

ربما تكون معالجتنا للاختيار المتعدد، بأن تكون مجموعة التعليمات التي ستنفِّذ هي نفسها من أجل عدة قيم ثابتة. ولكي تكون معالجتنا شاملة للحالات المختلفة لقيم c، يمكن أن يكون لدينا معالجة خاصة <مجموعة تعليمات أخرى> عندما تأخذ c قيم أخرى مختلفة عن جميع القيم الثابتة التي ميّزنا حالاتها.

نعطي الصياغة العامة التالية للتعبير عن تعليمة الاختيار المتعدد في حالة :



إن صياغة الحل بالبنية الجديدة (تعليمة التفريع)، مع كونه مكافئاً لتجميع تعليمات باستخدام إذا، إلا أنه أكثر انتظاما في التعبير عن ماهية ما نريد عمله. إذ بهذه البنية (في حالة متحول يأخذ القيم التالية نفِّذ وفقاً لكل قيمة...) نحدد تماما أن تفريعات المعالجة هنا متوقفة على حالة c أي على القيم التي يأخذها c.

توجد هذه البنية الشرطية متعددة الاختيار، التحكم واختيار التنفيذ حسب حالة متحول، في العديد من لغات البرمجة (Visual Basic, Ada, Pascal, C, Java).

ومتحول التعليمة الذي نفرّع التنفيذ حسب قيمته/حالته يمكن أن يكون من **نمط عدد صحيح** أو **نمط حرفي**.

في لغة C, Java :

تستخدم التعليمة المعرَّفة بالكلمتين المفتاحيتين switch و case. تنفَّذ هذه التعليمة كالتالي: عندما يأخذ متحول التعليمة قيمة أحد الثوابت، تقوم بتنفيذ التعليمات الموافقة له، ولكنها ، بخلاف الكثير من لغات البرمجة، لا تنهي تنفيذ تعليمة الاختيار المتعدد آلياً بعد ذلك، لذا لا بد من تنفيذ تعليمة break بعد كل تفريع.

<< تذكَّر: تؤدي التعليمة break إلى الخروج إلى ما بعد نهاية كتلة تعليمات.>>

مثال برمجي:

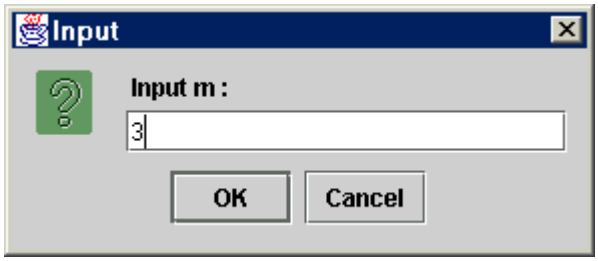
نريد كتابة عدد الأيام الموافق لكل شهر.

الدخل: رقم الشهر

الخرج: عدد الأيام.

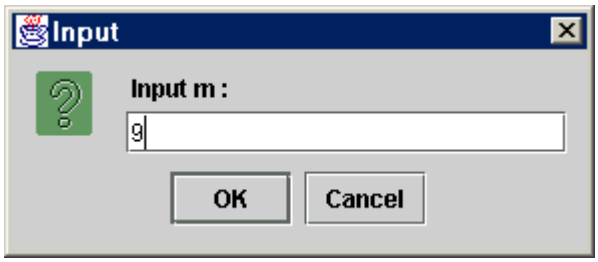
لا يوجد أي علاقة حسابية بين الخرج والدخل، ولا بد من وضع تعداد الحالات التي يكون فيها الشهر مساوياً 31 يوم، والحالات 30 يوم، وحالة الشهر 2 (شباط) 28 يوم.

```java
import javax.swing.JOptionPane;
class SwichMonth31 {

  public static void    main( String[] arg ) {

    int m;   String Sm;
    int mDays = 0;

    // Read m
     Sm = JOptionPane.showInputDialog(" Input m : ");
     m  = Integer.parseInt(Sm);
     // give the number of days  (mDays) per month  (m)
     switch ( m ) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
                mDays = 31;
                break;
        case 4:
        case 6:
        case 9:
        case 11:
                mDays = 30;
                break;
        case 2:
                mDays = 28;
                break;
    }

      System.out.println(" Month  " + m + " has " +  mDays + " days");
  }
 }
```



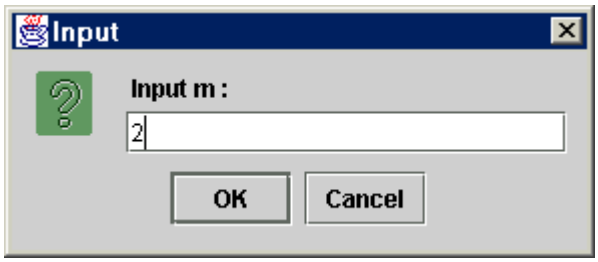
نتيجة تنفيذ البرنامج

Month 3 has 31 days



نتيجة تنفيذ البرنامج

Month 9 has 30 days



نتيجة تنفيذ البرنامج

Month 2 has 28 days

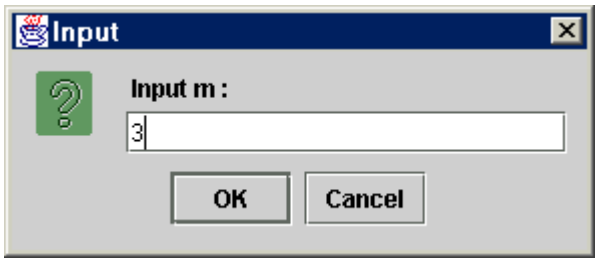
مثال برمجي:

نريد كتابة الاسم المختصر (الحروف الثلاثة الأولى) لشهور السنة باللغة الإنكليزية: January تكتب Jan ....

الدخل: رقم الشهر

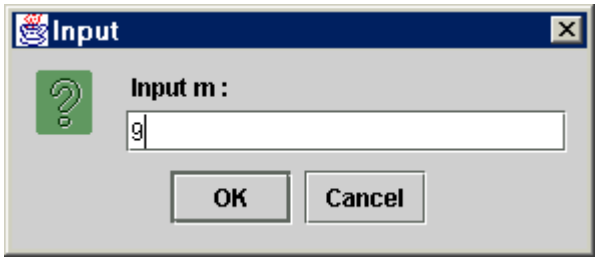
الخرج: اسم الشهر المختصر.

```
import javax.swing.JOptionPane;

class SwichMonth {
  public static void    main( String[] arg ) {

    int m;   String Sm;
    String month;
    // Read m
     Sm = JOptionPane.showInputDialog(" Input m : ");
     m  = Integer.parseInt(Sm);
     // give month string abreviation corresponding to number: m
     switch ( m ) {
        case 1: month = "Jan";
                break;
        case 2: month = "Feb";
                break;
        case 3: month = "Mar";
                break;
        case 4: month = "Apr";
                break;
        case 5: month = "May";
                break;
        case 6: month = "Jun";
                break;
        case 7: month = "Jul";
                break;
        case 8: month = "Aug";
                break;
        case 9: month = "Sep";
                break;
        case 10: month = "Oct";
                break;
        case 11: month = "Nov";
                break;
        case 12: month = "Dec";
                break;
        default:
                month = "";
    }
    // write month
    System.out.println(" Month : " + m + " is " +  month);
  }
}
```



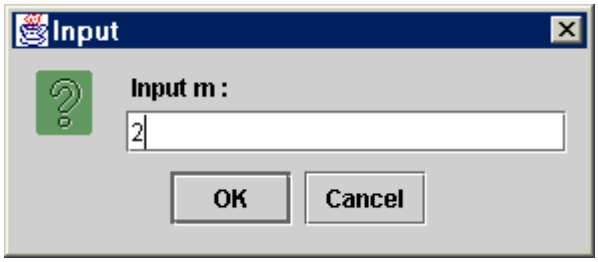
نتيجة تنفيذ البرنامج

Month : 3 is Mar



نتيجة تنفيذ البرنامج

Month : 9 is Sep



نتيجة تنفيذ البرنامج

Month : 2 is Feb

ملاحظات:

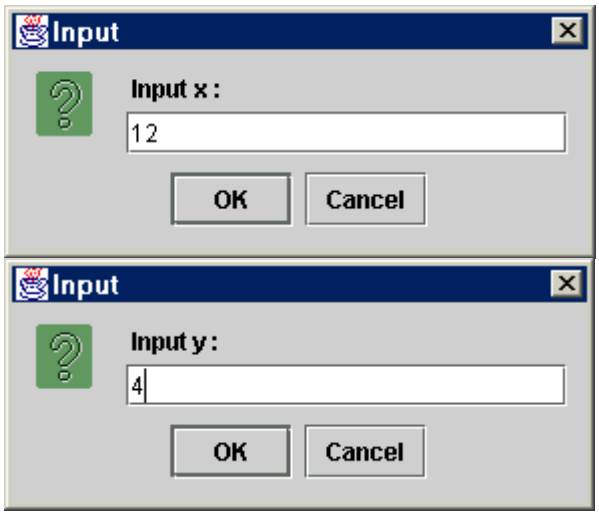
نستخدم هنا تعليمة switch case مع تمييز الحالات الأخرى بالكلمة المفتاحية default.

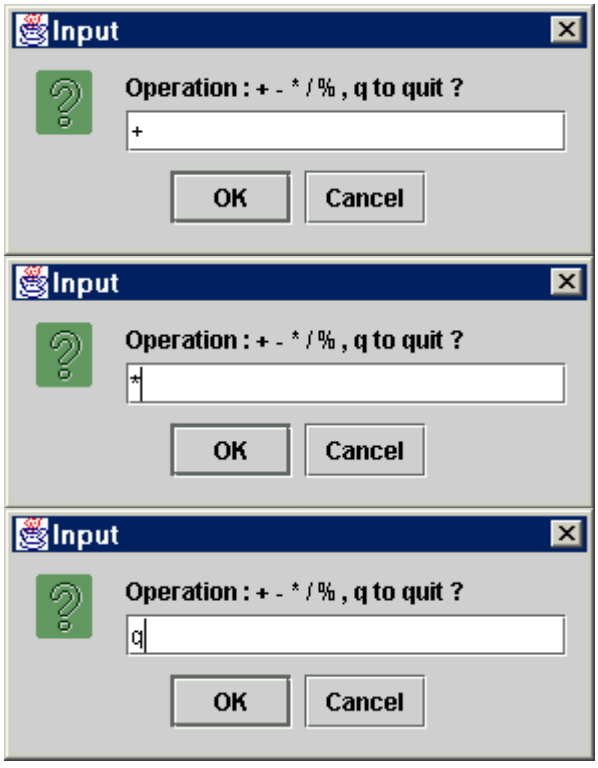
لاحظ أن هذا العمل البسيط، الذي نريده من البرنامج، يستدعي منا كل هذه الأسطر البرمجية. ولكن بما نعرفه حتى الآن من بنى معطيات لا يمكن لنا الكتابة بشكل أفضل. سنرى عند استخدام المصفوفات (الجداول) أنه يمكننا تحقيق ذلك، على نحو موجز وواضح.

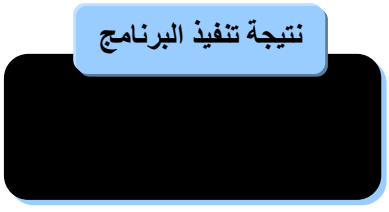
تمارين:

تمرين 1: باستخدم تعليمة التكرار بخطوة، اكتب برنامجاً يقوم بكتابة جميع جداول الضرب من 1 حتى 10.البرنامج لا يأخذ أي دخل.

تمرين 2:اكتب برنامجاً يقوم بقراءة عددين صحيحين، ثم يكرر إظهار لائحة خيارات menu لإدخال حرف يدل على العملية الحسابية التي يريد المستخدم تنفيذها على العددين. يقوم البرنامج بقراءة (الحرف) اختيار المستخدم وينفذ العملية حسب قيمة الحرف المدخل ويظهر النتيجة. يتكرر نفس العمل حتى يعطي المستخدم اختيار الخروج من البرنامج q.

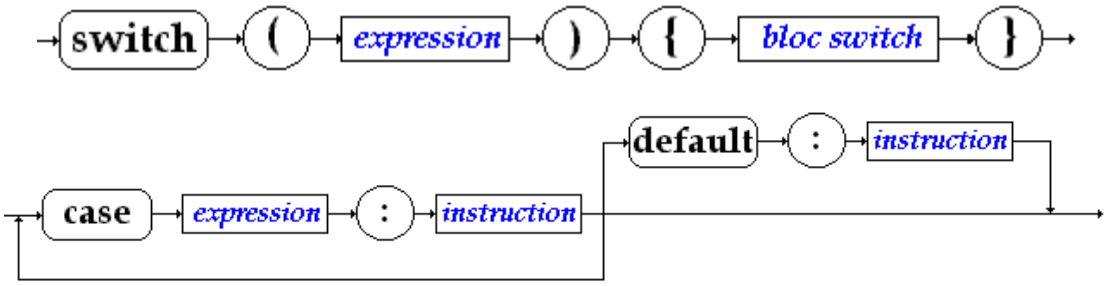






15. تعليمة التفريع: switch … case

يكون لتعليمة switch … case الشكل القواعدي التالي:



نقرأ التعليمة كمايلي:

- بعد التعرُّف على تعليمة expression ندخل إلى bloc switch:
  - نطابق بين التعليمة expression الخارجية التي تعرفنا عليها عند مدخل bloc switch مع التعليمات expression الداخلية المُحددة داخله
  - عند حدوث تطابق يجري تنفيذ التعليمات instruction المقابلة
  - في حال وجود عدة حالات تطابق بين تعليمة expression الخارجية مع تعليمات expression في الداخل، يجري تنفيذ التعليمات المرتبطة بتعليمات expression الداخلية المطابقة حسب تسلسل ظهورها
  - إذا أردنا أن يقتصر التنفيذ على التعليمة المطابقة الأولى فقط، توجب إضافة تعليمة break إلى نهاية التعليمات المرتبطة بكل تعليمة expression داخلية؛
  - في حال عدم وجود أي تطابق مع التعليمات expression الداخلية، يجري تنفيذ التعليمات instruction المرتبطة بالتعليمة default.

مثال:

```
int  x = 11;
switch (x)
{
     case 11 : Console.WriteLine(">> case 11");
          break;

     case 12 : Console.WriteLine(">> case 12");
          break;

     default : Console.WriteLine(">> default");
          break;
}
```

تكون النتيجة هي تطابق القيمة (x) مع الحالة الأولى أي (case 11).

16. أمثلة برمجية switch case:

مثال1: كتابة عدد الأيام الموافق لكل شهر.

الدخل: رقم الشهر

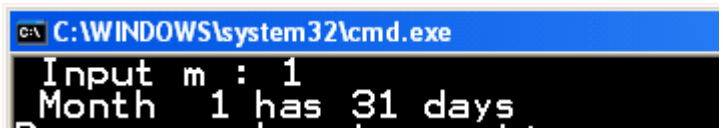
الخرج: عدد أيام الشهر.

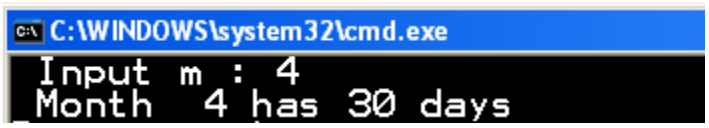
لا يوجد أي علاقة حسابية بين الخرج والدخل، ولا بد من وضع تعداد الحالات التي يكون فيها الشهر مساوياً 31 يوم، والحالات 30 يوم، وحالة الشهر 2 (شباط) 28 يوم.

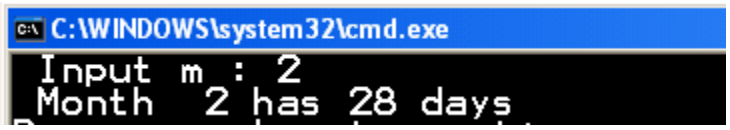
النص البرمجي:

```
using System;
namespace Instructions
{
class SwichMonth
{
public static void Main(string[] arg)
{
    int m; String Sm;
    int mDays = 0;

    // Read m
    Console.Write(" Input m : ");
    Sm = Console.ReadLine();  m = Int32.Parse(Sm);

    // give the number of days  (mDays) per month  (m)
    switch (m)
    {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            mDays = 31;
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            mDays = 30;
            break;
        case 2:
            mDays = 28;
            break;
    }

    Console.WriteLine(" Month  " + m + " has " + mDays + " days");
}
}
}
```

التنفيذ:


C:\WINDOWS\system32\cmd.exe
Input m : 1
Month 1 has 31 days


C:\WINDOWS\system32\cmd.exe
Input m : 4
Month 4 has 30 days


C:\WINDOWS\system32\cmd.exe
Input m : 2
Month 2 has 28 days

مثال2: كتابة الاسم المختصر للشهر (بالإنكليزية):

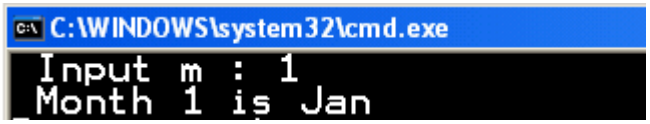
نريد كتابة الاسم المختصر (الحروف الثلاثة الأولى) لشهور السنة باللغة الإنكليزية: January تكتب Jan، February تكتب Feb....

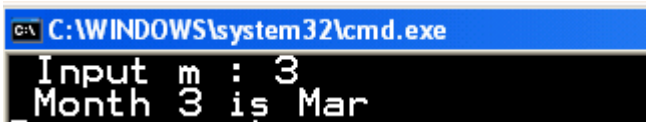
الدخل: رقم الشهر، الخرج: اسم الشهر المختصر.

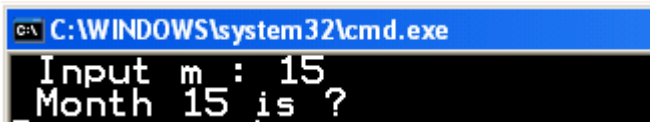
النص البرمجي:

```
using System;
namespace Instructions {
class SwichMonthName {
 public static void Main(string[] arg) {
    int m; string Sm;
    string month = "";

    // Read m
    Console.Write(" Input m : ");
    Sm = Console.ReadLine(); m = Int32.Parse(Sm);

    // give month string abreviation corresponding to number: m
    switch (m){
        case 1: month = "Jan";
            break;
        case 2: month = "Feb";
            break;
        case 3: month = "Mar";
            break;
        case 4: month = "Apr";
            break;
        case 5: month = "May";
            break;
        case 6: month = "Jun";
            break;
        case 7: month = "Jul";
            break;
        case 8: month = "Aug";
            break;
        case 9: month = "Sep";
            break;
        case 10: month = "Oct";
            break;
        case 11: month = "Nov";
            break;
        case 12: month = "Dec";
            break;
        default : month = "?";
            break;
    }
    // write month
    Console.WriteLine(" Month " + m + " is " + month);
}
}
}
```

التنفيذ:







ملاحظات:

- نستخدم هنا تعليمة switch case مع تمييز الحالات الأخرى(غير قيم الأشهر 1..12) بالكلمة المفتاحية default

- لاحظ أن هذا العمل البسيط، الذي نريده من البرنامج، يستدعي منا كل هذه السطور البرمجية. ولكن، بما نعرفه حتى الآن من بنى معطيات، لا يمكن لنا الكتابة بشكل أفضل. سنرى عند استخدام المصفوفات (الجداول) أنه يمكننا تحقيق ذلك، على نحو موجز وواضح

17. تمارين:

تمرين 1:

باستخدم تعليمة التكرار بخطوة for، اكتب برنامجاً يقوم بكتابة جميع جداول الضرب من 1 حتى 10.
البرنامج لا يأخذ أي دخل.

تمرين 2:

اكتب برنامجاً يقوم بقراءة عددين صحيحين، ثم يكرر إظهار لائحة خيارات menu لإدخال حرف يدل على العملية الحسابية التي يريد المستخدم تنفيذها على العددين. يقوم البرنامج بقراءة (الحرف) اختيار المستخدم وينفذ العملية حسب قيمة الحرف المدخل ويظهر النتيجة. يتكرر نفس العمل حتى يعطي المستخدم اختيار الخروج من البرنامج q. (استخدم تعليمة switch case)

نتيجة تنفيذ البرنامج

```
Input x : 12
Input y : 4
Operation: + - * / %, q  to quit?  +

4 + 12 = 16

Operation: + - * / %, q  to quit?  %
0 = 12 % 4

Operation: + - * / %, q  to quit?  q
```

الفصل الخامس:
أنماط (بنى) معطيات مُركّبة

الكلمات المفتاحية:

سلسلة حرفية، جدول، مصفوفة، صف جاهز.

ملخص:

يتعرف الطالب في هذا القسم على أنماط (بنى) معطيات مُركّبة المقدّمة في بعض الصفوف الجاهزة للإستخدام كسلاسل المحارف والجداول والمصفوفات. ويتعلم كيفية استخدتم توابع (طرائق) هذه الصفوف.

أهداف تعليمية:

يتعلم الطالب في هذا الفصل استخدام:

- سلاسل المحارف
- الجداول والمصفوفات

المخطط:

1. أنماط المعطيات المركَّبة
2. أنماط معطيات مركَّبة: سلاسل المحارف
3. أنماط معطيات مركَّبة: سلاسل المحارف
4. أنماط معطيات مركَّبة: سلاسل المحارف– التمثيل الداخلي لسلسلة محارف والوصول إلى مِحرف من محارف السلسلة
5. أنماط معطيات مركَّبة: سلاسل المحارف– التعديل: الحشر Insert
6. أنماط معطيات مركَّبة: سلاسل المحارف– التعديل: الدمج باستخدام عملية "+"
7. أنماط معطيات مركَّبة: سلاسل المحارف– التعديل: الحصول على موقع سلسلة جزئية من سلسلة محارف IndexOf
8. أنماط معطيات مركَّبة: سلاسل المحارف– التعديل: تحويل سلسلة محارف إلى جدول محارف ToCharArray
9. أنماط معطيات مركَّبة: سلاسل المحارف– التعديل: الإسناد والمقارنة
10. أنماط معطيات مركَّبة: سلاسل المحارف– تعريف جدول
11. أنماط معطيات مركَّبة: سلاسل المحارف– استخدام الجداول والمصفوفات
12. أنماط معطيات مركَّبة: سلاسل المحارف– أمثلة برمجية
13. أنماط معطيات مركَّبة: سلاسل المحارف– تعريف مصفوفة (جدول متعدد الأبعاد)، توليد خانات مصفوفة لم تتحدد أبعادها عند التعريف

1.أنماط المعطيات المركَّبة

تعريف

أنماط المعطيات المركّبة هي أنماط معطيات غير بسيطة. أي هي نمط معطيات مختلف عن الأنماط الأساسية التي عرفناها.

نسميها مركّبة لأنها بخلاف الأنماط البسيطة/الأساسية تتكون من مجموعة من العناصر، يمكن الوصول لأي عنصر منها.

الأمثلة الواضحة عن أنماط المعطيات المركّبة المتاحة في معظم لغات البرمجة هي السلاسل الحرفية string والجداول array .

فالسلاسل الحرفية مركبة من مجموعة حروف.

والجداول هي تجميع لعناصر من النمط نفسه يمكن الوصول إلى أي عنصر منه برقم العنصر ضمن الجدول.

في لغة C# جميع الأنماط المختلفة عن الأنماط البسيطة هي صفوف class.

- إن لغة C# هي لغة غرضية التوجه، و تعتمد في تعريف أنماط المعطيات المركَّبة على ما يسمى الصف class . لكننا في أساسيات البرمجة، لسنا معنيين بمعرفة كيفية بناء الصفوف في C#... بقدر ما نحن معنيون باستخداماتها.
- سنعرِّف الصف (بما يعنينا في أساسيات البرمجة) بأنه نمط معطيات مركَّب يسمح لنا بتعريف متحولات من هذا النمط كما في حالة الأنماط البسيطة.
- وكنا قد عرفنا الصف بأنه تجميع لمعطيات ووظائف. والوظائف (هي غالباً مانسميها توابع، إجرائيات، طرائق)، تُستخدَم هذه الوظائف على متحولات الصف فتنجز العديد من العمليات التي نحتاجها من هذا النمط (الصف).

- من الناحية العملية البرمجية والقواعدية نستخدم/نستدعي وظيفة/تابع على متحول من صف بالشكل varName.F()

كما في المثال:

```
string  s="abcdef"
s.IndexOf(def)
```

فضاء الأسماء namespace هو تجميع لعدد من الصفوف. أهمها هو System الذي يضم عدداً من الصفوف ذات الاستخدام الكبير كصف التوابع الرياضية Math وصف السلاسل string.

2. أنماط معطيات مركَّبة: سلاسل المحارف

يُعتبر نمط المعطيات String (المُعبر عن سلسلة محارف) صفاً من صفوف فضاء الأسماء (المكتبة المرجعية) System، في إطار العمل DotNet.

بالتالي، لايمكن استخدام أي سلسلة محارف من نمط string إلا من خلال مجموعة الطرائق/التوابع التابعة لهذا الصف.

3. أنماط معطيات مركَّبة: سلاسل المحارف

التصريح عن سلسلة محارف

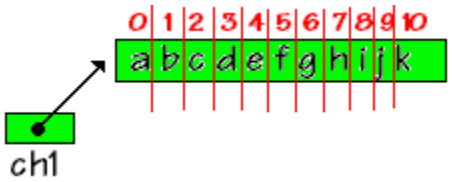
يجري التصريح عن سلسلة محارف وفقاً لمايلي:

```
string ch1 ;
ch1 = "abcdefghijk";

string ch2 = "abcdefghijk";
```

4. أنماط معطيات مركَّبة: سلاسل المحارف

التمثيل الداخلي لسلسلة محارف والوصول إلى مِحرف من محارف السلسلة

يكون لسلسة المحارف التمثيل الداخلي التالي:



يمكن الوصول إلى أحد محارف السلسلة باستخدام العملية "[ ]" (حيث تُقرأ السلسلة كجدول محارف)، وفقاً للمثال التالي:

```
string ch1 = "abcdefghijk";
char car = ch1[4] ; // contains the character
'e'
```

ولكن عملية الوصول تلك تقتصر على القراءة فقط، بحيث لايمكن تعديل محرف من محارف السلسلة باستخدام المؤشر [i]، وإنما اعتماداً على أدوات أخرى سنراها في الفقرات اللاحقة. فعلى سبيل المثال، إذ يعطي المترجم خطأً عند كتابة:

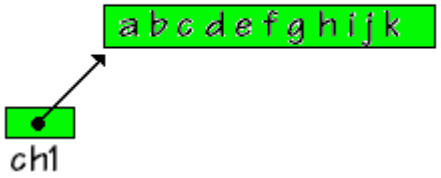
```
ch1[7]=car; // … Error
ch1[8]='x'; // … Error
```

5. أنماط معطيات مركَّبة: سلاسل المحارف

التعديل: الحشر Insert

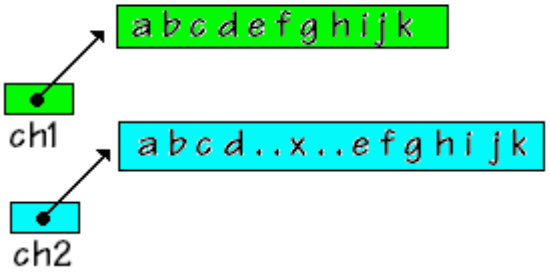
ليكن لدينا السلسلة ch1 التالية:

```
ch1 = "abcdefghijk" ;
```



ولننفذ عليها عملية الحشر الظاهرة في الشكل، فنحصل على السلسلة ch2 الناتجة التالية:

```
ch2 = ch1.Insert(4,"..x..") ;
```



يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

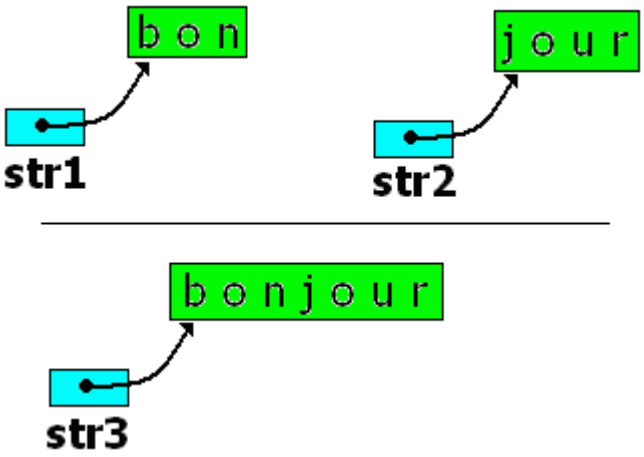
6. أنماط معطيات مركَّبة: سلاسل المحارف

التعديل: الدمج باستخدام عملية "+"

ليكن لدينا الوضع التالي:

```
string str1,str2,str3 ;

str1 = "bon" ;
str2 = "jour" ;

str3 = str1+str2 ;
```

نحصل بنتيجة الجمع (الدمج) على:



ونحصل على طول السلسلة باستخدام "الخاصة" Length كما يلي:

```
string str4 = "abcdef";
int Len;
Len = str1.Length ; // length = 6
```

يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

7.أنماط معطيات مركَّبة: سلاسل المحارف

التعديل: الحصول على موقع سلسلة جزئية من سلسلة محارف IndexOf

يمكن أن نحصل على موقع بداية السلسلة الجزئية "cde" مثلاً ضمن السلسلة "abcde" باستخدام التابع/الطريقة IndexOf التي تعيد عدد صحيح برقم موقع أول ظهور للسلسلة الجزئية كمايلي:

```
string str5 = "abcdef" , ssch="cde";

int ord;

ord = str1.IndexOf ( ssch );
```

يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

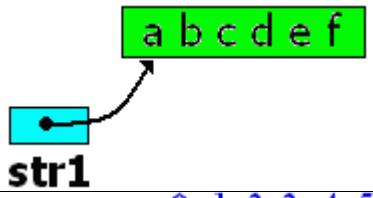
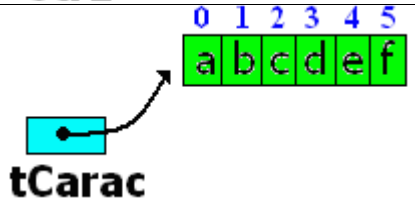
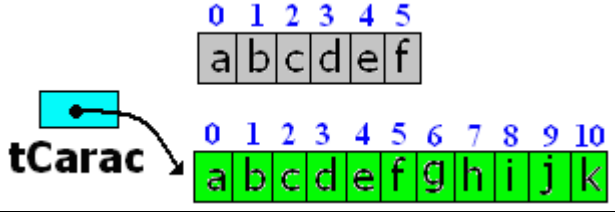
8. أنماط معطيات مركَّبة: سلاسل المحارف

التعديل: تحويل سلسلة محارف إلى جدول محارف ToCharArray

يمكن أن نرغب باستخدام سلسلة المحارف كجدول للمحارف لكي يتسنى لنا تطبيق عمليات خاصة بالجداول عليها. نستخدم في هذه الحالة الطريقة ToCharArray التي يمكن أن نستخدمها كمايلي:

```
string str6 = "abcdef" ;
char [ ] tCarac ;

tCarac = str6.ToCharArray( ) ;

tCarac = "abcdefghijk".ToCharArray( );
```

يوضح الجدول التالي التبديل الذي حصل مع تنفيذ كل تعليمة من التعليمات السابقة:

`string str6 = "abcdef" ;`	
`char [ ] tCarac ;` `tCarac = str6.ToCharArray( ) ;`	
`tCarac="abcdefghijk".ToCharArray();`	

يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

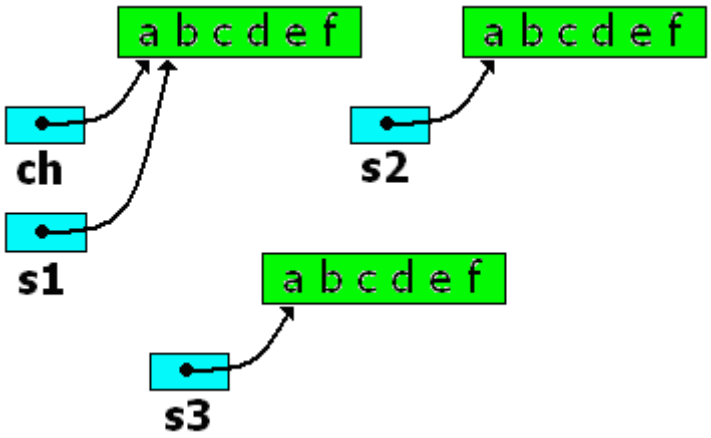
9.أنماط معطيات مركَّبة: سلاسل المحارف

التعديل: الإسناد والمقارنة

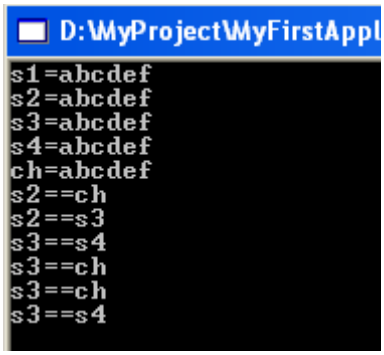
يمكن أن نستخدم عمليات الإسناد "=" مع سلاسل المحارف، كما يمكن أن نستخدم عملية مقارنة سلسلتين متساويتين "==" أو باستخدام الطريقة "Equals". بالإضافة إلى ماسبق، يمكننا اعتباراً من محتوى سلسلة أن نبني سلسلة جديدة مساوية لها باستخدام تعليمة "new" وهي عملية مختلفة عن عملية الإسناد حسب ما سنوضحه فيما يلي:

```
string s1,s2,s3,s4, ch;

ch = "abcdef";
s1 = ch;
s2 = "abcdef";
s3 = new string("abcdef".ToCharArray( ));
s4 = new string(s3.ToCharArray( ));

Console.WriteLine("s1="+s1);
Console.WriteLine("s2="+s2);
Console.WriteLine("s3="+s3);
Console.WriteLine("s4="+s4);
Console.WriteLine("ch="+ch);

if( s2 == ch )Console.WriteLine("s2==ch");
else Console.WriteLine("s2<>ch");

if( s2 == s3 )Console.WriteLine("s2==s3");
else Console.WriteLine("s2<>s3");

if( s3 == s4 )Console.WriteLine("s3==s4");
else Console.WriteLine("s3<>s4");

if( s3 == ch )Console.WriteLine("s3==ch");
else Console.WriteLine("s3<>ch");

if( s3.Equals(ch) )Console.WriteLine("s3==ch");
else Console.WriteLine("s3<>ch");

if( s3.Equals(s4) )Console.WriteLine("s3==s4");
else Console.WriteLine("s4<>ch");
```

يكون الفرق بين عملية الإسناد (حالة ch و s1) وعملية نسخ المحتوى (حالة s3 و s4) موضحاً فيما يلي:



ويعطي تنفيذ البرنامج الخرج التالي:



يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

10. انماط معطيات مركَّبة: الجداول والمصفوفات

تعريف جدول

- تعريف جدول بدون تحديد حجمه:

```
int [ ] table1;      // Table of integre
char [ ] table2;     // Table of char
float [ ] table3;    // Table of float
string [ ] tableStr;// Table of string
```

- تعريف جدول مع تحديد حجمه:

```
int [ ] table1 = new int [5];
char [ ] table2  = new char [12];
float [ ] table3  = new float [8];
string [ ] tableStr  = new String [9];
```

حيث تشير new إلى بناء غرض جديد من النمط المُحدد (int، أو char أو غيره) بعدد خانات مُحدد بالعدد الموضوع ضمن قوسين.

- تعريف جدول مع إعطائه قيماً ابتدائية مباشرةً:

```
int [ ] table1 = {17,-9,4,3,57};
char [ ] table2  = {'a','j','k','m','z'};
float [ ] table3  = {-15.7f, 75, -22.03f, 3 ,57 };
string [ ] tableStr  = {"cat","dog","mouse","cow"};
```

11. أنماط معطيات مركَّبة: الجداول والمصفوفات

استخدام الجداول والمصفوفات

يمكن تنفيذ عمليات اسناد على الجداول والمصفوفات واستخدامها ضمن تعليمات مختلفة مع الإنتباه إلى أن الجدول الذي طوله n، تكون خاناته موزعة بين الخانة رقم 0 والخانة رقم n−1.

أمثلة:

```
int [ ] table1 = new int [5];

// Assignment Operations:
table1[0] = -458;
table1[4] = 5891;
table1[5] = 72;  // Error table1[5] does not exists

// Loop:
for (int i = 0 ; i<= table1.Length-1;  i++)
     table1[i] = 3*i-1;
// Result: table1 = {-1,2,5,8,11}
```

```
char [ ] table2  = new char [7];

// Assignment Operations:
table2[0] = '?' ;
table2[4] = 'a' ;
table2[14] = '#' ; // Error table1[5] does not exists

//Loop
for (int i = 0 ; i<= table2.Length-1;  i++)
     table2[i] =(char)('a'+i);
// Result:  table2 = {'a', 'b', 'c' ,'d', 'e', 'f'}
```

12. أنماط معطيات مركَّبة: الجداول والمصفوفات

أمثلة برمجية

مثال 1: كتابة عدد الأيام الموافق لكل شهر.

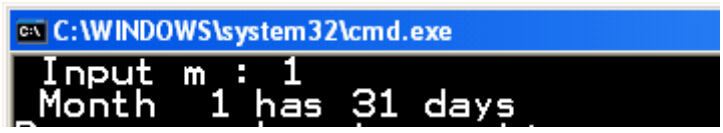
الدخل: رقم الشهر، الخرج: عدد أيام الشهر.

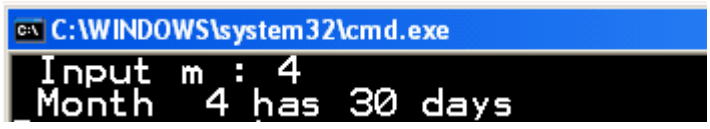
لا يوجد أي علاقة حسابية بين الخرج والدخل، ولا بد من وضع تعداد الحالات التي يكون فيها الشهر مساوياً 31 يوم، والحالات 30 يوم، وحالة الشهر 2 (شباط) 28 يوم.

النص البرمجي:

```
using System;
namespace ExampleArray
{

class ArrayMonth
{

 public static void Main(String[] arg)
 {

    int m; string Sm;
    int[] mDays = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

    // Read m
    Console.Write(" Input m : ");
    Sm = Console.ReadLine(); m = Int32.Parse(Sm);


    // give the number of days  (mDays) per month  (m)
    Console.WriteLine(" Month  "+m+" has " + mDays[m   1] + " days");
 }
}
}
```

التنفيذ:





ملاحظات:

- كم هو مختصر هذا النص البرمجي! قارن مع نفس البرنامج باستخدام التعليمة switch case
- أرقام العناصر في جداول اللغات الشبيهة بلغة C تبدأ من 0! لذلك كتبنا `mDays[m - 1]` لأخذ عدد أيام الشهر m.

تدريب: كتابة الاسم المختصر للشهر (بالإنكليزية) باستخدام الجداول:

كتابة الاسم المختصر (الحروف الثلاثة الأولى) لشهور السنة باللغة الإنكليزية: January تكتب Jan، February تكتب Feb....

مثال 2: قراءة قيم جدول وحساب القيمة الصغرى والعظمى والمتوسط الحسابي

الدخل: عدد عناصر الجدول، قيم الجدول (علامات طلاب مثلاً)

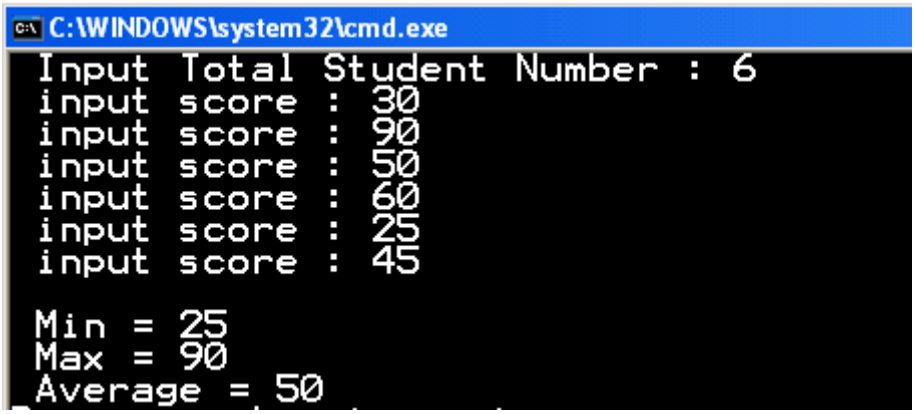
الخرج: القيمة الصغرى والعظمى والمتوسط الحسابي

النص البرمجي:

```
using System;
namespace ExampleArray
{
class ArrayStat
{
public static void Main(String[] arg)
{
    int N; string SN;
    int[] T;
    int i, minT, maxT;
    float sum;

    Console.Write(" Input Total Student Number : ");
    SN = Console.ReadLine(); N = int.Parse(SN);

    T = new int[N];

    // input Table
    for (i = 0; i <= N - 1; i++)
    {
        Console.Write(" input score : ");
        SN = Console.ReadLine();
        T[i] = int.Parse(SN);
    }
```

```
    sum = minT = maxT = T[0];

    for (i = 1; i <= T.Length - 1; i++)
    {
        sum = sum + T[i];
        if (T[i] < minT)
            minT = T[i];
        if (T[i] > maxT)
            maxT = T[i];
    }
    Console.WriteLine();
    Console.WriteLine(" Min = " + minT);
    Console.WriteLine(" Max = " + maxT);
    Console.WriteLine(" Average = " + (sum / N));
}
}
}
}
```

التنفيذ:



ملاحظات:

- لاحظ فكرة الحجم الديناميكي للجدول: نقرأ N طول الجدول ثم ننشئ جدول بهذا الحجم `T = new int[N].`
- لاجديد خوارزمياً هنا، فقط استخدمنا ماتعلمناه سابقاً: إيجاد أصغر عنصر (أو أكبر عنصر) و حساب المجموع لمجموعة أعداد (هذه المجموعة هنا هي عناصر الجدول)

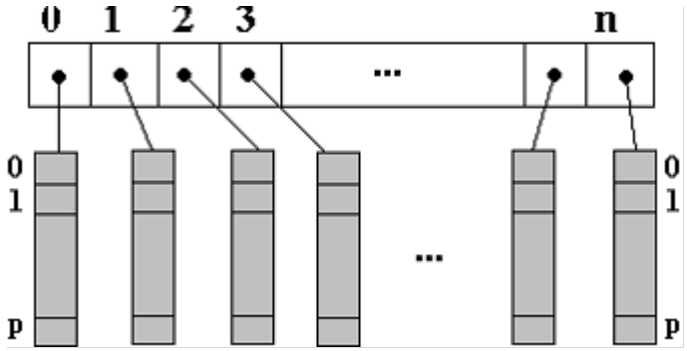
13.أنماط معطيات مركَّبة: الجداول والمصفوفات

تعريف مصفوفة (جدول متعدد الأبعاد)، توليد خانات مصفوفة لم تتحدد أبعادها عند التعريف:

يجري توليد خانات مصفوفة لم تتحدد أبعادها وتهيأتها يدوياً وفقاً للمثال التالي والذي نريد فيه توليد خانات المصفوفة t[n+1][p+1]:

```
int n=10, p=8;

int [ ][ ] table = new int [n+1][ ];

for (int i=0; i<n+1; i++)
     table[i] = new int [p+1];
```

ويمكن تمثيل مثل هذه المصفوفة بالشكل:



الفصل السادس:
مقدمة عن التوابع والإجرائيات (الطرائق)

الكلمات المفتاحية:

صف، تابع، إجرائية، طريقة،

ملخص:

يتعلم الطالب في هذا القسم بناء واستدعاء التوابع والإجرائيات (طرائق الصفوف).

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- بنية النص البرمجي في C# (عندما يكون مؤلفاً من صفّ واحد)
- تعريف تابع/إجرائية (طريقة تابعة للصف)
- استدعاء التابع
- تمرير المُعاملات
- تعريف مدى المتحولات
- مسائل للحلّ

المخطط:

1. بنية النص البرمجي في C#
2. التوابع والإجرائيات (الطرائق)
3. التصريح عن طريقة وتعريفها في C#
4. استدعاء طريقة
5. تمرير المعاملات– مقدمة
6. تمرير المعاملات– تجانس الأنماط البسيطة
7. تمرير المعاملات– تمرير القيمة
8. تمرير المعاملات– تمرير العنوان
9. إرجاع نتيجة طريقة
10. مدى تعريف المتحولات
11. عناصر الصف، ومتحولات الطرائق
12. تمارين للتجريب

1. بنية النص البرمجي في C#

يمكننا، وحسب مالاحظنا في بعض الأمثلة التي وردت في الأقسام السابقة، أن بإمكان برنامج كامل مكتوب بلغة C# أن يكون مضَّمناً في صف برمجي واحد، وأن يلعب Main دور البرنامج الرئيسي فيه، وبذلك نتعامل في C# بكتابة البرامج كما في أي لغة برمجة إجرائية أخرى مثل لغة C.

يتكون قالب النص البرمجي من:

- عبارة using، لاستحضار المكتبات (فضاء الأسماء) التي يمكن أن نستخدمها (مثل وظائف الدخل/الخرج Read, Write).
- كلمة namespace يليها إسم تعريف وجسم ويكون مُحاطاً بقوسين من الشكل "{" و "}"

- كلمة class يليها اسم الصف وجسمه البرمجي أي كتلة الصف ويكون مُحاطاً بقوسين من الشكل "{" و"}".

- ترويسة البرنامج الرئيسي Main : يليه جسم البرنامج أي كتلة البرنامج (تعليماته) مُحاطاً بقوسين من الشكل "{" و "}".

بعد أن يجري حفظ الصف في ملف "xxx.cs"، تولِّد عملية ترجمة الملف الآنف الذكر الملف "xxx.exe" الجاهز للتنفيذ.

أمثلة:

```
using System;
namespace Example
{

      class Example1
      {
            static void Main(string[ ] args)
            {
                  // You can add your instructions
here
            }
      }
}
```

```
using System;
namespace Example
{

      class Example2
      {
            static void Main(string[ ] args)
            {
                  Console.WriteLine("Hello C# ");
            }
      }
}
```

تمارين

- التمرين الأول:

اكمل البرنامج التالي بحيث نستبدل الحرفين الأول والأخير ببعضهما في جدول المحارف Tablechar، وأظهر النتيجة. (ابحث عن البرنامج التنفيذي *.exe وشغّله مباشرةً)

ملاحظة: يحول التابع newstring(Tablechar) جدول المحارف إلى سلسلة محارف.

```
using System;
class Application1
{
     static void Main(string[ ] args)
     {
          char [ ] Tablechar ={'a','b','c','d','e','f'} ;
          int i, j ;
          Console.WriteLine("Before : " + new
string(Tablechar));

          // Add Code Here

          Console.WriteLine("After : " + new string(Tablechar));
     }
}
```

الحلّ:

```
using System;
class Application1
{
     static void Main(string[ ] args)
     {
          char [ ] Tablechar ={'a','b','c','d','e','f'} ;
          int i, j ;

          Console.WriteLine("Before : " + new
string(Tablechar));
          for ( i = 0 , j = 5 ;  i<j ; i++ , j-- )
          {
               char car ;
               car = Tablechar[i];
               Tablechar[i ]= Tablechar[j];
               Tablechar[j] = car;
          }

          Console.WriteLine("After : " + new string(Tablechar));
     }
}
```

- التمرين الثاني:

اكمل البرنامج التالي الذي يساعدك في البحث عن رقم elt ضمن جدول table وعن إعطاء رسالة تؤكد وجود أوعدم وجود elt في الجدول مع إعطاء ترتيبه في حال وجوده. (ابحث عن البرنامج التنفيذي *.exe وشغّله مباشرةً)

```
using System;

class Application2
{
     static void Main(string[ ] args)
     {
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          int elt = 4, i ;

          //Add Code Here
     }
}
```

الحلّ:

```
using System;

class Application2
{
     static void Main(string[ ] args)
     {
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          int elt = 4, i ;

          for ( i = 0 ; i<8 ; i++ )
               if (elt == table[i]) break ;

          if (i == 8)
          Console.WriteLine("Value : " + elt + " Not Found.");

          else
          Console.WriteLine("Value : " + elt + " ... Order : " +i);
     }
}
```

2. التوابع والإجرائيات (الطرائق)

إن لغة C# هي لغة غرضية التوجه، وبالتالي تعتمد في تأطير البرمجة، على ما يسمى الصف class. لكننا في أساسيات البرمجة، تعلمنا حتى الآن، جميع التقنيات في البرمجة المهيكلة، باستخدام C#. لم نكن معنيين بمعرفة آلية برمجة وبناء الصفوف في C#... بقدر ما كنا معنين باستخداماتها.

ولكن تعلم أساسيات البرمجة لا تكتمل إلا بإمكانية كتابة برامج جزئية Subprogram . فمن المنطقي ألا يكون النص البرمجي كاملاً (جميع التعليمات) مضمَّناً في كتلة واحدة، فنحن حتىالآن نضمِّن كل التعليمات ضمن جسم البرنامج الرئيسي Main. ومن هنا نرى الحاجة إلى تجزئة البرنامج الكلي إلى برامج جزئية.

إذن، كبداية، البرنامج الجزئي هو تجميع لمجموعة من التعليمات، يُعطى اسم تعريف، ويصبح بالإمكان طلب تنفيذه (استدعاؤه call) بشكل مباشر أو غير مباشر من البرنامج الرئيسي.

يمكن أن نتخيل معاً، أن تجزئة/تقسيم البرنامج سيكون موجّه وظيفياً، أي أن يكون لكل برنامج جزئي وظيفة معينة: حساب قيمة تابع رياضي مثلاً، أو إجراء عمل معين كإظهار على الشاشة لمجموعة قيم في جدول.

هناك مصطلحات للبرامج الجزئية: توابع، إجرائية function, procedure. في البرمجة الغرضية التوجه تُسمى البرامج الجزئية طرائقmethods.

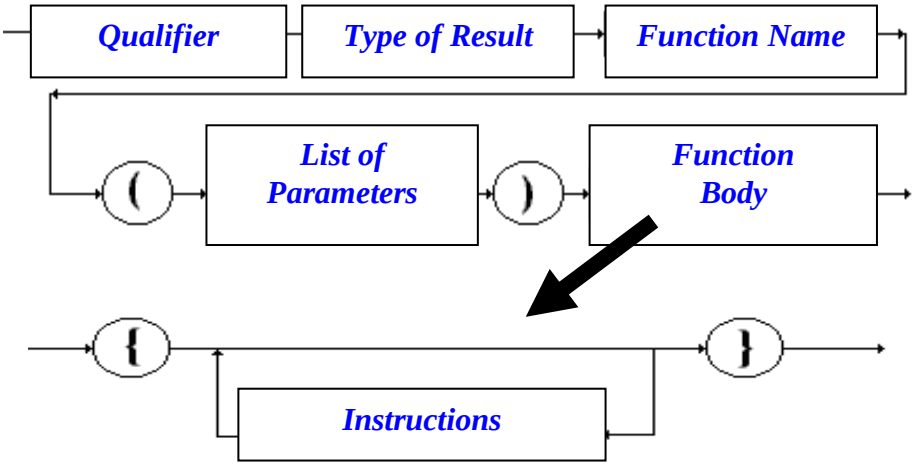
ولكي نبقى في تعلم أساسيات البرمجة، التي تسمح لنا بالتعامل مع أي لغة برمجة "إجرائية" مثل Pascal Basic, Php, C، ولكي نستطيع تعلم هذه الأساسيات في بيئات برمجية حديثةإطارها العام لغة غرضية التوجه مثل C#, Java يكفينا صف واحد يتضمن البرنامج الرئيسي Main وجميع البرامج الجزئية (الطرائق) التي نحتاجها.

تميز C# نوعين من الطرائق: "طريقة الصف" نفسه والتي تظهر عند تعريف الصف، و"نسخة الطريقة" التابعة لغرض والناتجة عن نسخ طريقة الصف ضمن غرض (متحول) له نمط الصفّ.

سنكتفي في هذه المرحلة بطرائق الصف لتسهيل العمل. بالنتيجة، عندما نستخدم في الفقرات اللاحقة كلمة "طريقة" دون أية صفات أخرى لها، فإننا نعني بها "طريقة صف" وذلك لأن تطبيقاتنا لاتملك إلا صفاً واحداً بحيث يمكن اعتبار طرائق هذا الصف بمثابة برامج جزئية للتطبيق في حالة البرمجة العادية.

3. التصريح عن طريقة وتعريفها في C#

يحتاج التصريح عن طريقة إلى تعريف ترويسة تحتوي على صورة عن المُعاملات اللازمة لعمل الطريقة، يجري بعدها تعريف جسم الطريقة الذي يحوي التعليمات التي سيجري تنفيذها عند استدعاء الطريقة. بشكل عام، يكون التصريح عن الطريقة وجسم الطريقة متتاليان بحسب الشكل القواعدي التالي:



برمجياً، يجري التصريح طريقة وفق الشكل القواعدي التالي:

<Qualifier><Type of Result><Function Name> (<Formal List of Parameters>)

وتكون دلالة هذا التصرح كمايلي:

- حتى الآن سنكتفي بالكلمة المفتاحية **static** ككلمة تعبر عن <Qualifier>، وهي كلمة **تشير إلى أن الطريقة هي طريقة تابعة للصف في الصف المُعرَّفة فيه.** ويمكن إهمال هذا الجزء من التصريح
- تعيد الطريقة نتيجة تنتمي إلى أحد الأنماط البسيطة التي تعرفها C#، أو void (في حال لم يكن هناك نتائج إرجاع)، أو الأنماط المركبة الجاهزة أو التي يعرفها المُبرمج. ولايمكن إهمال هذا الجزء من التصريح
- يشبه التصريح عن المُعاملات، تعريف المتحولات والتصريح عنها، ويمكن لهذه اللائحة أن تكون فارغة
- ويمكن لجسم الطريقة أن يكون فارغاً مُعبراً عن طريقة لاقيمة لها

مثال:

```
using System;

class Application
{
      // Methods without parameters
      int compute1( ){
            //.....
      }

      bool test1( ){
            //.....
      }

      void recompute1( ){
            //.....
      }

      // Methods with parameters
      int compute2(byte a, byte b, int x ) {
            //.....
      }

      bool test2(int k) {
            //.....
      }

      void recompute2(int x, int y, int z ) {
            //.....
      }

      static void Main(string[ ] args) {
            //...
      }
}
```

4.استدعاء طريقة

يجري استدعاء طريقة بلغة C# بأسلوب كلاسيكي مع مجموعة المُعاملات الضرورية لعمل الطريقة بحيث يتوجب استخدام نفس عدد المعاملات تبعاً لعدد المُعاملات المُصرَّح عنه، واستخدام نفس الأنماط تبعاً للأنماط المُصرَّح عنها.

أمثلة:

نفِّذ البرامج التالية التي تستدعي طرائق ولاحظ نتائجها:

```
using System;
class Application
{
     static void Main(string[ ] args)
     {
          Display( );
     }

     static void Display( )
     {

          Console.WriteLine("Good Morning");
     }
}
```

```
using System;
class Application
{
     static void Main(string[ ] args)
     {
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          long elt = 4;
          int i ;

          for ( i = 0 ; i<8 ; i++ )
               if (elt==table[i]) break ;
          Display(i,elt);
     }
     static void Display (int ord , long val)
     {
          if (ord == 8)
          Console.WriteLine("Value: " + val + " Not Found.");
               else
          Console.WriteLine("Value: " + val + " Order: " + ord);
                                                  }
}
```

تمرين1:

عدّل في المثال الثاني بحيث يقوم المستثمر بإدخال الرقم الذي نبحث عنه في الجدول (المحتوى في المتحول elt) واجعل البرنامج يتكرر حتى يقوم المستثمر بإدخال الرقم 0 فيتوقف البرنامج عن العمل.

الحلّ:

```
using System;
class Application
{
     static void Main(string[ ] args)
     {
          string s;
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          long elt=-1000;
          int i ;

          do
          {
               Console.Write("\nPlease enter the number: " );
               s=Console.ReadLine();
               elt=Int16.Parse(s);

                    for ( i = 0 ; i<8 ; i++ )
                  if (elt==table[i]) break ;

                    Display(i,elt);

          }while (elt != 0);
     }

     static void Display (int ord , long val)
     {
          if (ord == 8)
          Console.WriteLine("Value: " + val + " Not Found.");

               else
          Console.WriteLine("Value: " + val + " Order: " + ord);
                                                  }
}
```

5. تمرير المعاملات

مقدمة

تمثل المعاملات المُستخدَمة عند تعريف الطرائق ، متحولات شكلية تساعد في تفسير عمل البرنامج من أجل قيم فعلية ستحل مستقبلاً (عند استدعاء الطريقة وتنفيذها) محل هذه المُعاملات.

تشبه هذه العملية، عملية تعريف تابع رياضي مثل f(x)=x+5 حيث يلعب f هنا دور الطريقة، وتلعب x دور المُعامل الشكلي. توضع القيم الفعلية عند تنفيذ التابع، فعندما تحل القيمة 2 محل x يجري تنفيذ f فنحصل على القيمة 7.

f(2)=7

f(7)=12

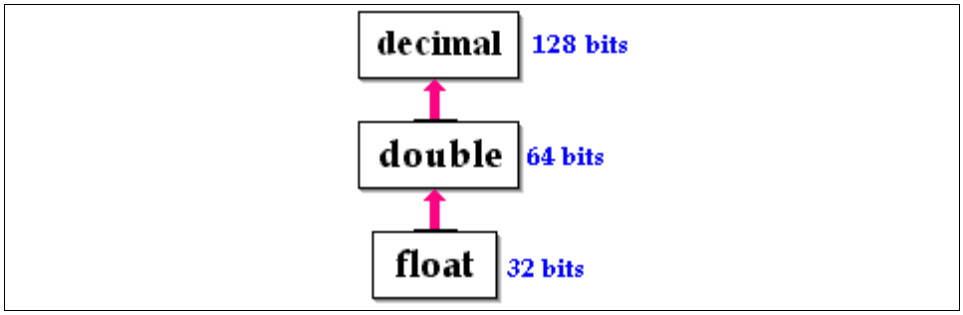
6.تمرير المعاملات

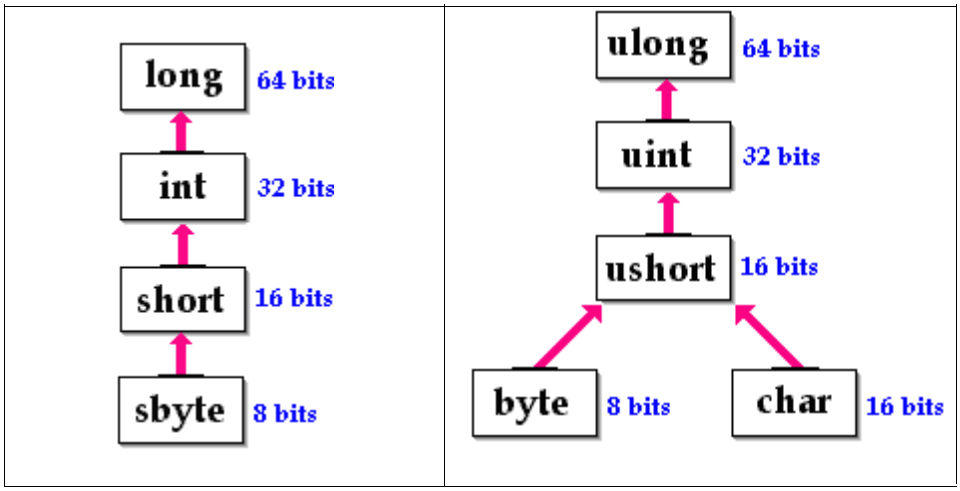
تجانس الأنماط البسيطة:

يمكن، تبعاً لتسلسل الأنماط المتجانسة الظاهر في كل شكل من أشكال الشريحة أن يتمكن المتحول ذو النمط صاحب الحجم الأكبر أن يستوعب قيمةً تنتمي إلى نمط متجانس معه ذو حجم أصغر.

فعلى سبيل المثال، يمكن لمتحول من نمط short أن يحتوي قيمة (أن نسند له متحولاً) من نمط sbyte ولكن لايمكن أن يقبل قيمة من نمط char.

ينطبق هذا الكلام على مُعاملات، ففي حال تعريف إجرائية (طريقة) تمتلك معاملاً من النمط short مثلاً، يمكننا عندها استخدامها مع متحول عددي صحيح من نمط sbyte كبديل عن المُعامل.





مثال:

```
using System;

class Application
{
     static void Main(string[ ] args)
     {
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          sbyte elt = 4;
          short i ;
          for ( i = 0 ; i<8 ; i++ )
               if (elt==table[i]) break ;
          display (i,elt);
     }
     static void display (int ord , long val)
     {
          if (ord == 8)
          Console.WriteLine("Value : " + val + "Not Found.");
          else
          Console.WriteLine("Value : " + val + " Order :" + ord);
     }
}
```

تمرين

حدد فيما إذا كانت الأنماط التالية صالحة لكي يأخذها المُعامل val دون ظهور أي أخطاء عند ترجمة البرنامج الآتي:

- short;
- int;
- sbyte;
- char;
- ushort;
- ulong;
- double;

```
using System;

class Application
{
     static void Main(string[ ] args)
     {
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          sbyte elt = 4;
          short i ;

          for ( i = 0 ; i<8 ; i++ )
               if (elt==table[i]) break ;

          afficher(i,elt);
     }
     static void afficher (int ord , sbyte val)
     {
          if (ord == 8)
          Console.WriteLine("Value : " + val + "Not Found.");
          else
          Console.WriteLine("Value : " + val + " Order :" + ord);
     }
}
```

الحل:

- short; → Yes
- int; → Yes
- sbyte; → Yes
- char; → No
- ushort; → No
- ulong; → No
- double; → No

7. تمرير المعاملات

تمرير القيمة:

عند استدعاء (طلب تنفيذ) الطرائق، يجري التنفيذ بتمرير قيمة المتحولاتً من أجل كافة المعاملات. وهذا مانسميه تمرير الوسطاء بالقيمة Parameter passing by value وهذا ينطبق على جميع المعاملات ذات الأنماط البسيطة في C#.

عند استدعاء طريقة تمتلك معامل ممرر بالقيمة، من أجل متحول ما، يقوم البرنامج ببناء نسخة من المتحول (نسخة من قيمته) وتمريرها إلى الطريقة بحيث لايؤثر أي تعديل على النسخة المُمررة، على المتحول الأصلي.

تلقائياً يكون تمرير معاملات للطرائق، تمريراً للقيمة.

مثال:

```
using System;

class Application
{
     static void Main(string[ ] args)
     {
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          int res = 0;

          short i ;
          for ( i = 0 ; i<8 ; i++ )
               res=Compute(i,res, table[i]);


          Console.WriteLine("Result : " + res);
     }

     static int Compute (short s, int r , int val)
     {
          int k=r+s*val;
               return k;
     }
}
```

تمرين:

ماهي قيمة k التي تظهر نتيجة تنفيذ البرنامج التالي؟ علق على النتيجة:

```
using System;
class Application
{
     static void Main(string[ ] args)
     {
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          int k = 0;

          short i ;
          for ( i = 0 ; i<8 ; i++ )
               Increment(k);

          Console.WriteLine("Result : " + k);
     }

     static void Increment(int val)
     {
          val=val+1;
     }
}
```

الحلّ:

Result : 0

8. تمرير المعاملات

تمرير العنوان :

عند التصريح عن الطرائق ومعاملاتها، يمكن أن نحّدد إذا كنا نريد طريقة استدعائها بتمرير الوسطاء بالعنوان (بالمرجع/ بالمؤشِّر).

في C# :

- يكون أسلوب تمرير عناوين المتحولات صالحاً من أجل كافة الأنماط البسيطة للمعاملات بوضع الكلمة المفتاحية ref
- لجميع الأنماط المركَّبة (الصفوف) للمعاملات يجري تلقائياً تمرير العنوان

عند استدعاء طريقة – تمتلك معامل ممرر بالعنوان– من أجل متحول ما، لايبني البرنامج نسخة عن المتحول، وإنما يستخدم المتحول نفسه، بحيث يؤثر أي تعديل على النسخة المُمررة، على المتحول الأصلي.

تأخذ عملية تعريف المُعامل المُمرر بالعنوان، وعملية التمرير بالعنوان، الشكل التالي:

```
static int mymethod (int a , ref char b)
{
 //.....
  return a+b;
}


….

int x = 10, y = '$', z = 30;
z = mymethod(x, ref y) ;
```

ملاحظة هامة:

يجب الانتباه أن الاستدعاء للطريقة يجب أن يتضمَّن أيضاً الكلمة المفتاحية ref

تمرين:

ماهي قيمة k التي تظهر نتيجة تنفيذ البرنامج التالي؟ علق على النتيجة:

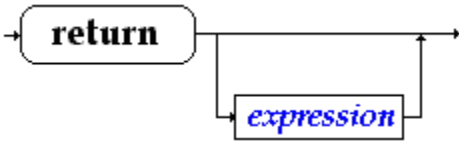
```
using System;

class Application
{
     static void Main(string[ ] args)
     {
          int [ ] table= {12,-5,7,8,-6,6,4,78,2};
          int k = 0;

          short i ;
          for ( i = 0 ; i<8 ; i++ )
               Increment(ref k);

          Console.WriteLine("Result : " + k);
     }

     static void Increment(ref int val)
     {
          val=val+1;
     }
}
```

الحلّ:

Result : 8

9. إرجاع نتيجة طريقة:

يمكن لأي طريقة أن تعيد قيمة من نمط محدد وذلك اعتماداً على الكلمة المفتاحية return التي تأخذ الشكل القواعدي التالي:



دلالياً، يجب أن تحقق **return** مايلي:

- **يجب** أن تعيد تعبير له نفس نمط الإرجاع المُعرَّف عند التصريح عن الطريقة وتعريفها
- عند الوصول إلى **return** أثناء تنفيذ تعليمات الطريقة، يتوقف تنفيذ بقية التعليمات الموجودة ما بعد **return** ؛ ويجري الخروج من البرنامج الجزئي والعودة بالتنفيذ إلى مكان استدعاء الطريقة
- عند وجود أكثر من مسار تنفيذي ممكن ضمن برنامج واحد (كوجود تعليمة if else ، حيث تشكل if مسار تنفيذ لحالات معينة، وتشكل else مسار تنفيذ آخر لحالات أخر) يجب وضع تعليمة return في نهاية كل مسار تنفيذي. انظر المثال 2

مثال1:

يحسب البرنامج التالي التابع f(x)=3x−7 من أجل x=4 ومن أجل x=5

```
using System;
class Application
{
     static void Main(string[ ] args)
     { // ...
          int x , y ;
          x = 4 ;
          y = f(5) ;
          y = f(x) ;

          Console.WriteLine("f(x)=" + f(x) );
          Console.WriteLine("f(5)=" + f(5) );
        }

      static int f (int x )
      {
               return 3*x-7;
      }
}
```

مثال2:

تقوم الطريقة increment بإضافة 1 إلى قيمة المتحول إذا كان لايساوي الصفر.

```
using System;
class Application
{
     static void Main(string[ ] args)
     {
          int a = 0  ;
          a=Increment ( a );
          a=Increment ( a+4 );
     }
          static int Increment(int x)
     {
          if (x == 0)
          {
               Console.WriteLine("The Case of 0 ...");
               return x;
          }
          else
          {
               Console.WriteLine("The other cases");
               return x++;
          }
     }
}
```

10. مدى تعريف المتحولات

تقضي القاعدة الأساسية، بأن يكون المتحول مرئي (قابل للاستخدام) ضمن المقطع أو الكتلة التي جرى تعريفه فيها.

نعني بالمقاطع أو كتل التعليمات في لغة C# مايلي:

- الصفوف
- الطرائق
- التعليمات الأساسية المُركبة (تعليمة if else، أو تعليمة while، أو تعليمة do، أو تعليمة for)

بشكل عام:

- لايمكن تعريف متحول ضمن طريقة، إذا سبق وجرى تعريف معامل للطريقة أو متحول محلي للطريقة بنفس الإسم
- لايمكن تعريف متحول ضمن مقطع (if، أو while، أو for، أو do)، إذا سبق وجرى تعريف متحول بنفس الإسم في أي مقطع يحتوي المقطع المذكور

تمرين:

هل تعطي البرامج التالية أخطاء عند ترجمتها، وأين هي هذه الأخطاء في حال وجودها؟

برنامج 1:

```
using System;
class Application
{
      static void Main(string[ ] args)
      {
            int a = 0  ;
            a=Increment ( a );
            a=Increment ( a+4 );
      }
      static int Increment(int x)
      {
            if (x == 0)
            {
                  int x;
                  Console.WriteLine("The Case of 0 ...");
                  return x;
            }
            else
            {
                  Console.WriteLine("The other cases");
                  return x++;
            }
      }
}
```

الحلّ:

```
using System;
class Application
{
      static void Main(string[ ] args)
      {
            int a = 0  ;
            a=Increment ( a );
            a=Increment ( a+4 );
      }
      static int Increment(int x)
      {
            if (x == 0)
            {
                  int x; //error
                  Console.WriteLine("The Case of 0 ...");
                  return x;
            }
            else
            {
                  Console.WriteLine("The other cases");
                  return x++;
            }
      }
}
```

برنامج 2:

```
using System;
class Application
{
     static void Main(string[ ] args)
     {
          int a = 0  ;
          a=Increment ( a );
          a=Increment ( a+4 );
     }

     static int Increment(int x)
     {
          int a=0;
          if (x == 0)
          {
               a++;
               Console.WriteLine("The Case of 0 ...");
               return x;
          }
          else
          {
               Console.WriteLine("The other cases");
               return x++;
          }
     }
}
```

الحلّ:

صحيح لايوجد خطأ.

البرنامج 3:

```
using System;
class Application
{
      static void Main(string[ ] args)
      {
            int a = 0  ;
            a=Increment ( a );
            a=Increment ( a+4 );
      }
      static int Increment(int x)
      {
            int a=0;
            if (x == 0)
            {
                  a++;
                  Console.WriteLine("The Case of 0 ...");
                  return x;
            }
            else
            {
                  int a=x;
                  Console.WriteLine("The other cases");
                  return x++;
            }
      }
}
```

الحلّ:

```
using System;
class Application
{
      static void Main(string[ ] args)
      {
            int a = 0  ;
            a=Increment ( a );
            a=Increment ( a+4 );
      }
      static int Increment(int x)
      {
            int a=0;
            if (x == 0)
            {
                  a++;
                  Console.WriteLine("The Case of 0 ...");
                  return x;
            }
            else
            {
                  int a=x; //error
                  Console.WriteLine("The other cases");
                  return x++;
            }
      }
}
```

11. عناصر الصف، ومتحولات الطرائق

تكون المتحولات المُعرَّفة كعناصر ضمن الصف، قابلة للاستخدام من قبل جميع طرائق الصف.

مثال1:

نلاحظ أن المتحول a هو أحد عناصر الصف، وهو مرئي في الطريقة g وفي الطريقة f. ففي الطريقة g يُستخدَم لحساب التعبير 3x–a أما في الطريقة f فيجري إخفاؤه بالمُعامل a الذي يُستَخدَم لحساب التعبير 3x–a.
(جرّب البرنامج ولاحظ النتيجة في حالتي f و g).

```
using System;
class Visiblity1
{
     static int a = 10;

     static void Main(string[ ] args)
     {
          int result_g, result_f;
          int x=10;

          result_g=g(x);
          result_f=f(x,5);

          Console.WriteLine("g(10)="+ result_g);
          Console.WriteLine("f(10,5)="+result_f);
     }

     static int g (int x )
     {
               return  3*x-a;
     }

     static int f (int x, int a )
     {
               return 3*x-a;
     }
}
```

ملاحظة:

تخضع عناصر الصف والمتحولات المُعرَّفة في طرائق الصف، إلى نفس القواعد الكلاسيكية المُستَخدَمة لتحديد مدى تعريف ورؤية المتحول

تمرين:

نفذ البرنامج التالي وأجب عن الأسئلة التالية:

1. هل نستطيع تعريف المتحول car في كل من g وf؟ ولماذا؟
2. ما هو برأيك الخطأ الموجود في تعريف طريقة الصف g والذي سيظهر عند محاولة ترجمة وتنفيذ البرنامج؟
3. هل تؤثر قيمة عنصر الصف a المُعرَّف في بداية الصف، على حساب كل من g، وf؟
4. صحح البرنامج ونفذه لتحصل على نتيجته.

البرنامج:

```
using System;

class Visiblity
{
     static int a = 10;

     static void Main(string[ ] args)
     {
          int result_g, result_f;
          int x=10;

          result_g=g(x);
          result_f=f(x,5);

          Console.WriteLine("g(10)="+ result_g);
          Console.WriteLine("f(10,5)="+result_f);
     }

     static int g (int x )
     {
          char car = 't';
          long a = 12345;

          return  3*x-a;
     }

     static int f (int x, int a )
     {
          char car ='u';

          return 3*x-a;
     }
}
```

الحلّ:

1. نعم لأن car متحول محلي بالنسبة للطريقة في كل حالة.
2. إرجاع 3x–a التي تأخذ النمط long (بسبب تعريف المتحول المحلي a كمتحول من النمط long)، بالرغم من أن نمط الإرجاع المُعلَن عنه هو int.
3. كلا، ففي كلا الحالتين تكون هذه القيمة مخفية بتعريف المتحول a المحلي بالنسبة للطريقة في كل حالة، والذي يجري اعتماده لحساب نتيجة الطريقة.
4. يمكن تصحيح البرنامج كما يلي:
5.

```
using System;

class Visiblity
{
     static int a = 10;

     static void Main(string[ ] args)
     {
          int result_g, result_f;
          int x=10;

          result_g=g(x);
          result_f=f(x,5);

          Console.WriteLine("g(10)="+ result_g);
          Console.WriteLine("f(10,5)="+result_f);
     }

     static int g (int x )
     {
          char car = 't';
          int a = 12345; //Correction

          return  3*x-a;
     }

     static int f (int x, int a )
     {
          char car ='u';

          return 3*x-a;
     }
}
```

12. تمارين للتجريب

تمرين1:

اكتب برنامج لحساب مربعات الأعداد من 1 إلى 100 وذلك من خلال تعريف طريقة تدعى Square لحساب المُربَّع.

الحلّ: بسيط جداً !!!!

تمرين2:

أعط نتيجة تنفيذ البرنامج التالي وعلل النتيجة التي تظهر في كل مرة وبعد كل استدعاء لطريقة من الطريقتين A و B.

```
using System;

class Scoping
{
      static int x=1;

      static void Main(string[ ] args)
      {
            Console.WriteLine("local x in method " + x);

            MethodA();
            Console.WriteLine("local x in 1st call of A: " + x);

            MethodB();
            Console.WriteLine("local x in 1st call of B: " + x);

            MethodA();
            Console.WriteLine("local x in 2nd call of A: " + x);

            MethodB();
            Console.WriteLine("local x in 2nd call of B: " + x);
      }

      static void MethodA()
      {
            int x = 25;
            ++x;
      }

      static void MethodB()
      {
            x *= 10;
      }
                                                                  }
```

الحلّ: 1، 10، 10، 100.

تمرين 3:

نُعرِّف تابع العاملي بالشكل 1*2*3 .... *(x-2)*(x-1)*x=x!.

1. اكتب برنامج، يكرر الطلب من المُستخدم إدخال قيمة x من نمط ushort ومن ثم يستدعي طريقة Fact(x) بحيث تقوم هذه الطريقة بحساب x! (يستمر البرنامج بالعمل مع تكرار طلبات إدخال أرقام حتى يُدخل المُستخدِم الرقم 0). (مساعدة: استخدم لحساب Fact(x) إحدى تعليمات التكرار مثل while، أو do، أو for)
2. من أجل أي قيمة للمتحول x يبدأ البرنامج بإعطاء قيمة Fact(x) تساوي 0 ولماذا؟

الحلّ: البرنامج المطلوب تنفيذه:

```
using System;
class Factorial
{
     static int x=1;
     static void Main(string[ ] args)
     {
          string s;
          ushort a=1000;
          long f;
               s=Console.ReadLine();
               a=UInt16.Parse(s);

          while (a !=0)
          {
               f=Fact(a);

               Console.WriteLine("Fact of " + a + " = " + f);

                       s=Console.ReadLine();
                       a=UInt16.Parse(s);
          }
     }

     static long Fact(ushort a)
     {
          long res=a;

          while (a>1)
          {
               a--;
               res=res*a;
          }

          return res;
     }
}
```

تمرين 4:

نُعرِّف متتالية Fibonacci كمايلي:

$$u(n)=\begin{cases} n, \text{ if } n=0 \text{ or } n=1 \\ u(n-1)+u(n-2) \end{cases}$$

اكتب برنامج، يكرر الطلب من المُستخدم إدخال قيمة n من نمط ومن ثم يستدعي طريقة Fibonacci(n) بحيث تقوم هذه الطريقة بحساب قيمة u(n). (يستمر البرنامج بالعمل مع تكرار طلبات إدخال أرقام حتى يُدخل المُستخدِم عدداً سالباً).

الحلّ:

```
class Scoping
{
      static int x=1;
      static void Main(string[ ] args)
      {
            string s;
            int n;
            long f;

            s=Console.ReadLine();
            n=Int32.Parse(s);

            while (n >= 0)
            {
                  f=Fibonacci(n);

                  Console.WriteLine("U("+n+")=" + f);

                  s=Console.ReadLine();
                  n=Int32.Parse(s);
            }
      }
      static long Fibonacci( int number )
      {
            if ( number == 0 || number == 1 )
            return number;

            else
            return Fibonacci( number - 1 ) + Fibonacci( number - 2 );
      }
}
```

الفصل السابع:
تمارين ومسائل للمناقشة والحلّ

ملخص:

يهدف هذا القسم إلى تقديم مجموعة من التمارين والمسائل حول مجموعة من الخوارزميات الأساسية التي ينبغي فهمها وحلها وتطبيقها بلغة C#.

أهداف تعليمية:

يتعامل الطالب في هذا الفصل مع مجموعة من التمارين التطبيقية التي تركز على ما تعلمه خلال الفصول السابقة من المادة.

تمارين ومسائل للمناقشة والحلّ

التمرين الأول:

اكتب بلغة C# برنامجاً لحساب القاسم المشترك الأعظم لعددين صحيحين لايساويان الصفر (خوارزمية إقليدس).

اكتب بلغة الخوارزميات قبل المباشرة بكتابة البرنامج بلغة C# مستخدماً بنية التابع (الطريقة).

الحلّ1: استخدام خوارزمية إسناد (قيمة باقي القسمة الأكبر على الأصغر) إلى (العدد الأكبر).

```
using System;
namespace Excersise
{
class ApplicationEuclide
{
static void Main(string[] args)
{
 Console.Write("First Number : ");
 int p = Int32.Parse(Console.ReadLine());

 Console.Write("Second Number : ");
 int q = Int32.Parse(System.Console.ReadLine());

 if (p * q != 0)
  Console.WriteLine("mgcd of " + p + " and " + q + " = " + mgcd(p, q));
 else
  Console.WriteLine("One of the numbers is null !");
}

static int mgcd(int a, int b)
{
    int r;

    while ((a != 0) && (b != 0))
    {
        if (a > b)
            a = a % b;
        else
            b = b % a;
    };
    r = (a != 0) ? a : b;
    return r;
}
}
}
```

الحلّ2: استخدام خوارزمية إسناد (قيمة حاصل طرح الأصغر من الأكبر) إلى (العدد الأكبر)

```
using System;
namespace Exercice1
{
 class ApplicationEgyptien
 {
       static void Main (string[ ] args)
       {
             Console.Write("First Number : ");
             int p = Int32.Parse( Console.ReadLine( ) ) ;
             Console.Write("Second Number : ");
             int q = Int32.Parse( Console.ReadLine( ) ) ;

             if ( p*q != 0 )
                   System.Console.WriteLine("mgcd of "+p+" and "+q+" is
"+mgcd(p,q));
             else
                   Console.WriteLine("One of the numbers is null !");
       }

       static int mgcd (int p, int q)
       {
             while ( p != q)
             {
                   if (p > q) p -= q;
                   else q -= p;
             }
             return p;
       }
 }
}
```

التمرين الثاني:

اكتب بلغة C# برنامجاً لإظهار أول n عدد أولي من مجموعة الأعداد الصحيحة الموجبة. اقترح الخوارزمية المناسبة لتنفيذ العمل، واكتبها بلغة الخوارزميات قبل المباشرة بكتابة البرنامج بلغة C#.

الحلّ:

```
using System;
namespace Exercice2
{
     class ApplicationPrem
     {
          static void Main(string[ ] args)
          {
               int divis, nbr, n, count = 0 ;
               bool is prem;

               Console.Write("How much numbers to Display ? ");
               n = Int32.Parse( Console.ReadLine( ) ) ;

               Console.WriteLine( 2 );
               nbr = 3;
               while (count < n-1)
               {
                    divis = 2 ;
                    is_prem = true;
                    do
                    {
                         if (nbr % divis == 0) is_prem=false;
                         else divis = divis+1 ;
                    }
                    while ((divis <= nbr/2) && (is_prem == true));
                    if (is_prem)
                    {
                         count++;
                         Console.WriteLine( nbr );
                    }
                    nbr++ ;
               }
          }
     }
}
```

التمرين الثالث:

اكتب بلغة C# برنامجاً للتحقق من أن سلسلة محارف تمتلك صفة "التناظر" PALINDROME، أي أنها تبقى نفسها سواء قرأناها من اليمين إلى اليسار أو من اليسار إلى اليمين.

مثال: abcddcba، acca

اقترح الخوارزمية المناسبة لتنفيذ العمل، واكتبها بلغة الخوارزميات قبل المباشرة بكتابة البرنامج بلغة C#.

الحلّ:

```
using System;
namespace Excersise
{
class palindrome
{
static String inverse(string s)
{
    string r = ""; ;
    int L = s.Length;
    for (int i = 0; i <= L - 1; i++)
        r = r + s[L - 1 - i];
    return r;
}

public static void Main(String[] args) {
    string s ;

    Console.Write(" Enter String:  ");
     s = Console.ReadLine();

    string invs = inverse(s);

   Console.WriteLine("Your  string : "+s);
   Console.WriteLine("Invers string : "+invs);

   if (s==invs)
        Console.WriteLine("palindrome !");
   else
       Console.WriteLine("Not palindrome !");
}
}
}
```

التمرين الرابع:

ندعو عدد Armstrong، كل عدد يكون مساوياً لحاصل جمع مكعبات الأرقام التي تؤلفه. مثال:

$$153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27.$$

هناك عدة أعداد Armstrong وكلها من مرتبة المئات، اكتب برنامج بلغة C# لتحديدها. اقترح الخوارزمية المناسبة لتنفيذ العمل، واكتبها بلغة الخوارزميات قبل المباشرة بكتابة البرنامج بلغة C#.

الحلّ:

```
using System;
namespace Armestrong
{
class ApplicationArmstrong
{
    static void Main(string[] args)
    {
        int i, j, k, n, sumcube;
        Console.WriteLine("Number of Armstrong:");

        for (i = 1; i <= 9; i++)
            for (j = 0; j <= 9; j++)
                for (k = 0; k <= 9; k++)
                {
                    n = 100 * i + 10 * j + k;
                    sumcube = i*i*i + j*j*j + k*k*k;

                    if (sumcube == n)
                        Console.WriteLine(n);
                }
    }
}
}
```

التمرين الخامس:

اكتب بلغة C# برنامجاً يقوم بعملية بحث خطي تسلسلي عن عنصر x ضمن جدول T مؤلف من n عنصر ويعطي ترتيبه في حال وجوده. اقترح الخوارزمية المناسبة لتنفيذ العمل، واكتبها بلغة الخوارزميات قبل المباشرة بكتابة البرنامج بلغة C#.

الحلّ: سنقترح هنا الخوارزمية فقط بلغة الخوارزميات ونترك للطالب تطبيقها كبرنامج.

```
i ←1;

while (i < n) and (T[i] <> x) do
        i ←i+1;
end_while

if T[i] = x then
        ord←i;
        write(“the element exists, order:”,ord);
else
        write(“element not found”)
end_if
```

الحلّ:

```
using System;
namespace ExampleArray
{
class ArrayFind
{
public static void Main(String[] arg)
{
    int x; string Sx;
    int[] T = { 12, 10, 25, 16, 40, 45, 51, 60, 75, 90, 140, 120 };
    int i, ord, N;
    N = T.Length;
    // Display Table
    for ( i = 0; i <= N - 1; i++)
        Console.WriteLine("[" + i + "]\t" + T[i]);
    // Read x
    Console.Write(" choose number of the table: ");
    Sx = Console.ReadLine(); x = Int32.Parse(Sx);
    // Find x
    i = 1;
    while ((i < N) && x != T[i])
        i = i + 1;
    if (i < N)
    {
        ord = i;
        Console.WriteLine("the element exists, order:" + ord);
    }
    else
        Console.WriteLine("element not found");
}
}
}
```

التمرين السادس:

ليكن لدينا جدول T مرتب ترتيباً تصاعدياً ويحتوي على N عنصر، وليكن x عنصر من هذا الجدول. اشرح هدف وعمل الخوارزمية التالية، وطبقها كبرنامج بلغة C#.

```
Bottom, Middle, Top, Order : Integer;

Bottom←1;
Top←N;
Order←-1 ;

repeat
   Middle←(Bottom + Top) div 2;
   if x = T[Middle] then
     Order←Middle;
   else
      if T[Middle] < x then
         Bottom←Middle + 1;
      else
          Top←Middle-1;
      end_if
   end_if
while ( x ≠ T[Middle] )
```

الحلّ:

```
using System;
namespace ExampleArray
{

class ArrayFind
{

 public static void Main(String[] arg)
 {

    int x; string Sx;
    int[] T = {12, 18, 25, 30, 40, 45, 51, 60, 75, 90, 100, 120 };
    int bottom, middle, top, ordre, N;

    N=T.Length;

    // Display Table
    for (int i = 0; i <= N-1; i++)
        Console.WriteLine("["+i+"]\t"+T[i]);

    // Read x
    Console.Write(" choose number of the table: ");
    Sx = Console.ReadLine(); x = Int32.Parse(Sx);

   // Find x
    ordre = -1;    bottom = 1; top = N-1;
    do
    {
        middle = (bottom + top) / 2;
        if (x == T[middle])
            ordre = middle;
        else
            if (x > T[middle])
                 bottom = middle + 1;
            else
                 top = middle - 1;
    }
    while (x != T[middle]);

    Console.WriteLine("\n ordre (index) of " + x + " = "+ordre);
 }
}
}
```

التمرين السابع:

اكتب برنامج بلغة C# يقرأ جدولاً من n رقم ويرتبها بالترتيب التصاعدي أو التنازلي حسب خوارزمية الترتيب بالتعويم Bubble Sort.

الخوارزمية التي يجب فهمها وتحويلها إلى برنامج بلغة C#:

```
Algorithm Bubble_Sort;

local: i , j , n, temp: Integer;
Input-Output : Tab : Table of n Integers;

Begin

for i=1 to n Do

   for j= 2 to i Do

      if Tab[ j-1 ] > Tab[ j ] then
         temp=Tab[ j-1 ] ;
         Tab[ j-1 ]= Tab[ j ] ;
         Tab[ j ] = temp ;
      End_if

   End_for

End_for

End_Bubble_Sort
```

التمرين الثامن:

اكتب برنامج بلغة C# يقرأ جدولاً من n رقم ويرتبها بالترتيب التصاعدي أو التنازلي حسب خوارزمية الترتيب بالحشر Sort by Insertion.

الخوارزمية التي يجب فهمها وتحويلها إلى برنامج بلغة C#:

```
Algorithm Sort_by_Insertion;

local: i , j , n, v: Integer (positive);
Input-Output : Tab : Table of n Integers;

Begin

for i=2 to n do

    v = Tab[ i ] ;
    j= i ;

    while (Tab[ j-1 ]> v)
        Tab[ j ] = Tab[ j-1 ];
        j = j-1;
    End_While ;

    Tab[ j ] = v ;

End_for

End_Sort_by_Insertion
```