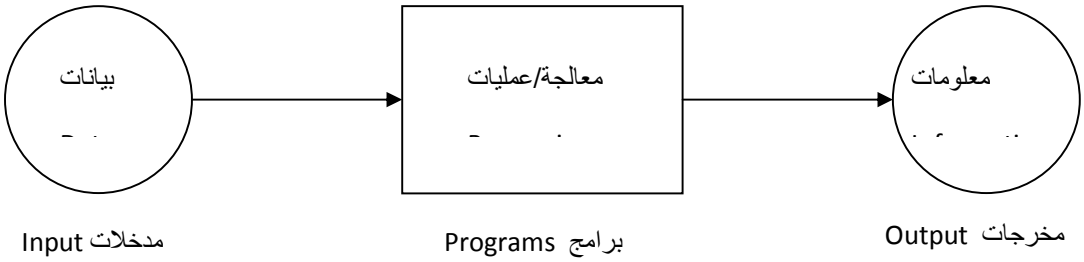
حاسب آلي

مقدمة إلى الحاسب الآلي والبرمجة

1-1 تعريف الحاسب الآلي

الحاسب الآلي هو كيانة عن مجموعة وحدات إلكترونية وميكانيكية (Hardware) تستعمل لإدخال وتنفيذ البرامج (Software) بهدف تحقيق أو إنجاز عمليات حسابية ومنطقية.

تختصر وظيفة الحاسب الآلي الأساسية بمعالجة البيانات بدقة فائقة وبسرعة كبيرة لإستنتاج المعلومات المطلوبة.



2-1 تاريخ الحاسب الآلي

يبدأ تاريخ المعلوماتية وعلوم الحاسوب مع اختراع الآلة 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 الآلة 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.

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.

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 و 1834 وصمم أول آلة حياكة قابلة للبرمجة (ميكانيكياً)، حيث استُعملت نفس التقنية بعدها لبناء العديد من الآلات الحربية.

أجيال الحاسوب

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.

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.

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.

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.

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.

واعتمدت شركة IBM عام 1980 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 Macintosh على نطاق واسع. وقد استخدمت حواسيب Apple Macintosh واجهات بيانية رسومية تعمل بالمؤشر، بدلاً من لوحة المفاتيح، كما كان الأمر عليه في نظام DOS.

ي نفس الوقت أصدرت Microsoft النسخة الأولى من نظام Windows وطورته عبر عقدين لتحوله من نظام خاص بحاسوب شخصي إلى نظام يمكن استخدامه ضمن شبكات حاسوبية في المؤسسات.

وفي عام 1991 طور Linus Torvald نظام التشغيل LINUX المجاني ذو الرماز 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 من الأخصائيين المشعوذين!!!

3-1 العتاديات (Hardware)

يتكون الحاسب الآلي من الوحدات الأساسية التالية:

وحدة الإدخال (Input Unit)

تستعمل هذه الوحدة لإدخال البيانات للحاسب من قبل المستخدم.

مثال: لوحة المفاتيح – الفارة – الماسح الضوئي

وحدة الإخراج (Output Unit)

تستعمل هذه الوحدة لإخراج المعلومات من الحاسب للمستخدم.

مثال: الشاشة – الطابعة

وحدة التخزين الرئيسة (Memory Unit)

تستعمل هذه الوحدة لتخزين البيانات المدخلة أو المخرجة وغالباً ما تكون سعتها منخفضة.

مثال: Random Access Memory (RAM) – Read Only Memory (ROM)

وحدة التخزين الثانوية (Secondary Storage Unit)

تستعمل هذه الوحدة لتخزين البيانات بشكل دائم وغالباً ما تكون سعتها كبيرة.

مثال: القرص الصلب (Hard Disk) – القرص المدمج (Compact Disc)

وحدة المعالجة المركزية (CPU – Central Processing Unit)

تستعمل هذه الوحدة لإدارة الحاسب وتنسيق أعماله، كما وأنها تشرف على عمل الوحدات الأخرى.

وحدة الحساب والمنطق (Arithmetic and Logic Unit)

تستعمل هذه الوحدة لتنفيذ العمليات الحسابية والمقارنات المنطقية.

4-1 البرمجيات (Software)

إنها البرامج أو الأوامر أو التعليمات التي تمكن عتاد الحاسب الآلي من العمل.

هنالك ثلاثة أنواع رئيسة للبرمجيات:

برامج النظم (System Software)

إنها البرامج التي تتحكم بعمل الحاسب الآلي.

مثال:

نظم التشغيل (Operating Systems) لتنسيق كافة العمليات والأوامر ضمن الحاسب الآلي

قائد جهاز (Device Drivers) لتأمين الإتصال بعتاد الحاسب الآلي وتحديد خصائص هذا الإتصال

برنامج مُساعد (Utility Software) لتجهيز عتاد الحاسب الآلي قبل إستعمالها كتقسيم وformating وحدات التخزين الثانوية.

البرامج التطبيقية (Application Software)

إنها البرامج التي تستخدم لإنجاز الأعمال بواسطة الحاسب الآلي.

مثال:

البرامج الإنتاجية: برامج معالجة النصوص (Microsoft Word) - برامج الجداول الإلكترونية (Microsoft Excel)

البرامج الإدارية: برامج المحاسبة – برامج حجز الرحلات الجوية

البرامج الترفيهية: Video Player – MP3 Player – Games

البرامج التثقيفية: Tutorials – Help

برامج اللغات (Programming Software)

إنها الأدوات التي يستخدمها المبرمج لكتابة وتحرير وتنفيذ البرامج (Code) في أي من لغات البرمجة المعروفة.

مثال: محرّر (Editor) – مترجم (Compiler) – مفسر (Interpreter)

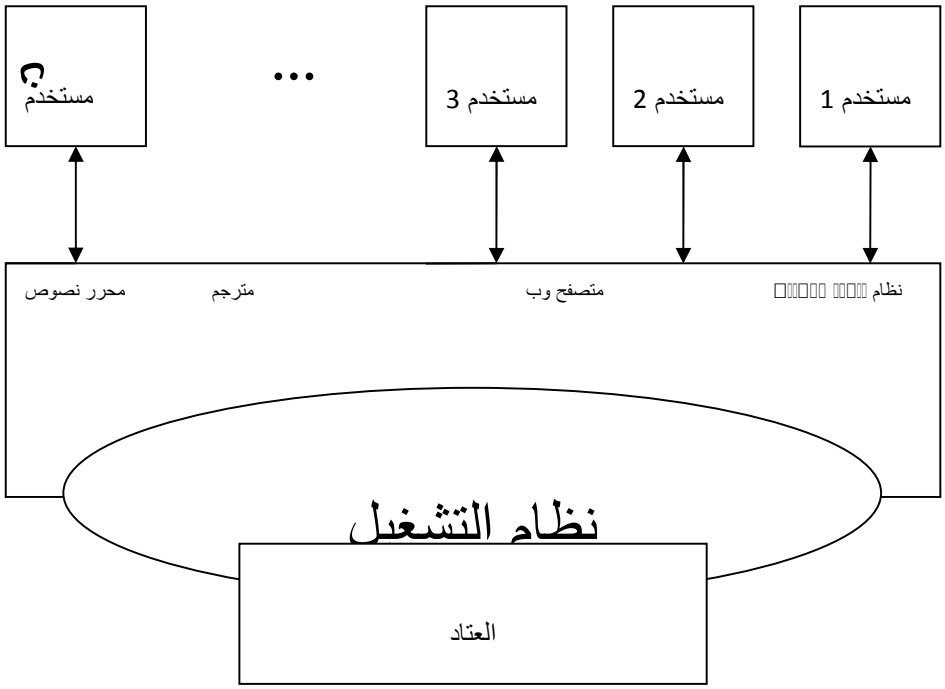
5-1 نظم التشغيل وتطورها

يُعرَّف نظام التشغيل بأنه برنامج يدير عتاد الحاسوب بحيث يوفر البرمجيات والتطبيقات الضرورية لتشغيل هذه العتاديات، كما يعمل كوسيط بين المستخدم والحاسوب بحيث يسمح للمستخدم باستثمار الحاسوب وتطبيقاته.

يختلف تصميم نظام التشغيل حسب البيئة التي يُفترض أن يعمل عليها، إذ يُصمم نظام التشغيل الذي يعمل على المخدمات 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 المخدمات.

الحاسب الآلي ونظام التشغيل

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



مهمة نظام التشغيل:

- نظام التشغيل كمحصص للموارد
- نظام التشغيل كبرنامج تحكم
- نظام التشغيل كنواة.

يعتبر نظام التشغيل جزءاً هاماً من كافة الأنظمة الحاسوبية، بحيث يمكن أن نقسم النظام الحاسوبي إلى أربعة مكونات رئيسية وهي:

- العتاديات
- نظام التشغيل
- التطبيقات
- المستخدمين

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 كمحصص للموارد، وكنظام تحكُّم، وكنواة لتشغيل التطبيقات الحاسوبية:

- نظام التشغيل كمحصص 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 تحصيص أو أداة تحكم يُولّد 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.**

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، وتُمثَّل 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، كما كانت وسائط التخزين تتمثل عموماً بالبطاقات المثقبة؛

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 الأعمال في قرص تخزين، كما يجري انتقاء مجموعة من تلك الأعمال ونقلها إلى الذاكرة لكي يجري تنفيذها معاً، ولا يجري نقل كافة الأعمال المخزنة لأنه غالباً ما تكون المعطيات المخزنة على القرص أكبر من سعة التخزين في الذاكرة؛ تسمى عملية انتقاء الأعمال 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

6-1 ترميز المعلومات

نُعرِّف الترميز على أنه تابع تقابل بين معلومة وبين سلسلة من 0 و 1 تمثل هذه المعلومة وتكون قابلة للتخزين ضمن الآلة.

أنواع الترميز:

- ترميز المحارف: الترميز ASCII (American Standard Code for Information Interchange)، و الترميز UNICODE (Universal Code)
- ترميز الأعداد الصحيحة: الترقيم

تعريف:

تكون المعلومات المُخزَّنة في الحاسوب على شكل سلسلة من 0 و 1 (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 (Universal Code)، مما ساعد على توفير إمكانية تمثيل 65536 حرف، وأدى لفتح المجال أمام تمثيل الأحرف العربية، والصينية، والكورية وغيرها.

ترميز الأعداد الصحيحة: الترقيم

يمكن ترميز الأعداد الصحيحة كمحارف، إلا أن مثل هذا الترميز سيُعقّد 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 و 9، وعندما تكون b=2 ندعو قاعدة الترقيم، قاعدة ثُنائيّة، وتكون الأرقام التي تؤلف الأعداد محصورة بين 0 و 1، وعندما تكون b=8 ندعو قاعدة الترقيم قاعدة ثُمانّية وتكون الأرقام التي تؤلف الأعداد محصورة بين 0 و 7.

بالنتيجة، تكون ارتباطات الأرقام التي تؤلف العدد 010010 الممثل ثنائياً كما يلي من اليسار إلى اليمين:

$2^5 \rightarrow 0$

$2^4 \rightarrow 1$

$2^3 \rightarrow 0$

$2^2 \rightarrow 0$

$2^1 \rightarrow 1$

$2^0 \rightarrow 0$

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

$(0 * 2^5) + (1 * 2^4) + (0 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0) = 18$

7-1 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 التي تعبر عن شيفرة يفهمها 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 الحية عند بدء تنفيذ البرنامج وذلك لتسريع عملية التنفيذ.

تكون وحدة المعالجة المركزية مسؤولة عن معالجة الشيفرة الناتجة عن عملية ترجمة البرنامج والمكتوبة بلغة الآلة من خلال:

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

8-1 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، وغيرها) وتدعى عادةً بلغة المُجمِّع وتَستخدِم رموزاً تمثل عمليات الحاسوب، ويتوجب ترجمة كافة الرموز المكتوبة بلغة المُجمِّع إلى لغة الآلة الممثلة بشيفرة وسلاسل ثنائية (مؤلفة من 0 و1). 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، ولغات التوصيف. ويتوجب ترجمة كافة الرموز المكتوبة بلغة برمجة عالية المستوى إلى لغة الآلة الممثلة بشيفرة وسلاسل ثنائية (مؤلفة من 0 و1). تجري عملية الترجمة باستخدام برامج خاصة تُدعى المُترجمات.

اللغات البرمجية عالية المستوى: لمحة تاريخية

تضمن اللغات البرمجية عالية المستوى تحقيق مجالٍ واسعٍ من المهام البرمجية المختلفة.

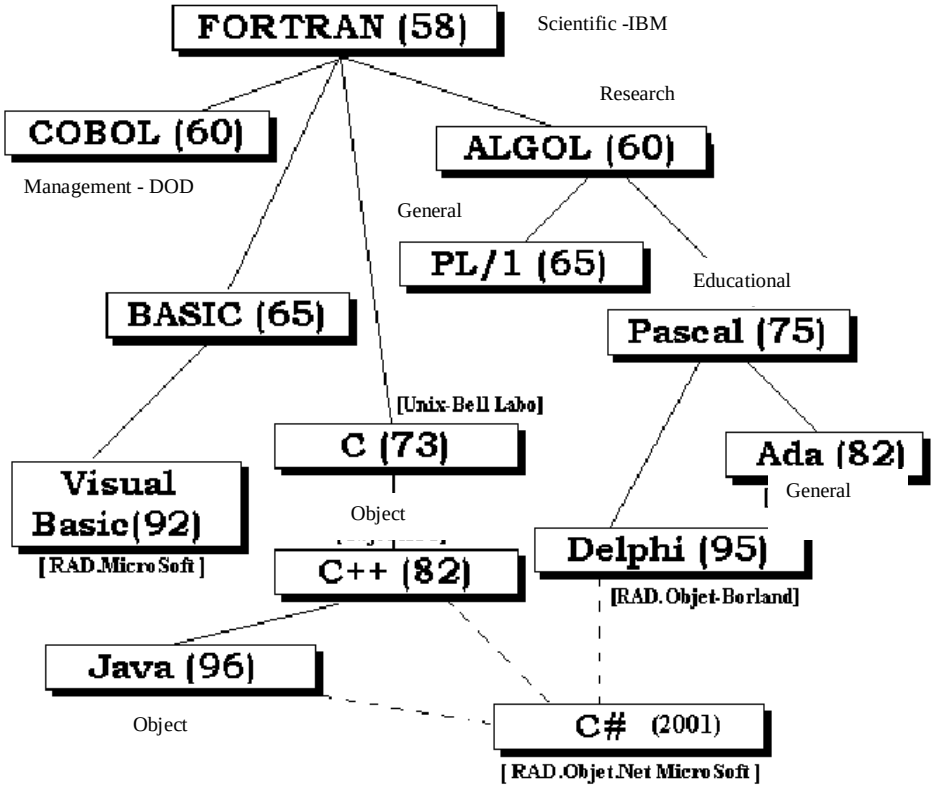
لقد جرى تطوير العديد من لغات البرمجة المختلفة على مر السنين بهدف تلبية الاحتياجات المتغيرة في تقنيات المعلومات:

- عام 1958: لغة Fortran؛
- عام 1964: لغة BASIC؛
- عام 1970: لغة ADA؛
- عام 1971: لغة Pascal؛
- عام 1972: لغة C؛
- عام 1986: لغة ++C؛
- عام 1991: لغة Visual Basic؛
- عام 1995: لغة Java.

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 كلغة Fortran 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 1970 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.

اللغات البرمجية عالية المستوى: اللغات الإجرائية (1)



مثال عن برنامج بلغة إجرائية هي لغة FORTRAN

```
C   *** FORTRAN ***
    integer i,j,k
    write(5,50)
5   format(2X,6HGood Day)
    i=15
    if (i.GT.10) goto 10
    read(6,80) j
```

```
6   format(i4)
    do 10 k=1,i-1,2
     j=j+1
10  continue
    end
```

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 محدّد من الكلمات والتعليمات الخاصة، ومثال ذلك: تعليمة الإنتقال GOTO، والإسناد LET والإنهاء END، بالإضافة إلى التعليمات الشرطية IF، والحلقية WHILE.

أنماط المعطيات:

تعبر أنماط المعطيات عن حجم الذاكرة المخصصة لتخزين قيمة محددة أو مجموعة من القيم، إذ يعبر نمط العدد الصحيح أو Integer على سبيل المثال، عن 16 بت أو 32 بت من مساحة الذاكرة المخصصة لتخزين عدد صحيح. ندعو الأنماط الأساسية (أعداد صحيحة، أعداد حقيقية،

محارف، ... الخ) بالأنماط البسيطة، في حين ندعو مصفوفات الأعداد وسلاسل المحارف بالأنماط المُركبة.

العمليات:

توجد، بالإضافة إلى التعليمات، رموز أخرى تدعى بالعمليات يتم استخدامها للإشارة إما إلى عملية حسابية، أو إلى علاقة منطقية.

الدخل والخرج:

يتم تنفيذ عملية الدخل باستخدام تعليمات محددة مثل 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.

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 محارف، ويجري تنميط الرقمين 2 و 3 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 و 3 بالشكل (3 2 +).

اللغات البرمجية عالية المستوى: اللغات المنطقية

جرى اشتقاق اللغات المنطقية وعلى رأسها لغة PROLOG (PROgrammation en LOGique) 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 محرك خاص بالتنفيذ، استخلاص النتيجة وهي (سقراط فان)، بحيث يمكن إضافة النتيجة إلى المقدمات لإغنائها.

ندعو المحرك الذي يستخلص النتائج من المقدمات **بمحرك استدلال**.

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.

تُعتبر لغات مثل C++، Java، C# من أشهر اللغات الغرضية التوجه، وسنستعرض مفاهيم التصميم والبرمجة الغرضية التوجه لاحقاً في فصل خاص كما سنركز في فصولنا اللاحقة على لغة C#.

المترجمات والمفسرات

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.

المفسر هو شبيه بالمترجم إلا أنه يعمل على ترجمة كل تعليمة على حدة قبل التنفيذ ويقوم بتحويل البرنامج إلى حالة وسطية بين لغة البرمجة المعتمدة ولغة الآلة.

9-1 الخوارزميات

تُعتبر الخوارزميات أدوات رياضية مساعدة على إيجاد حلول منهجية للعديد من المسائل. ويجري تعريف الخوارزمية على أنها سلسلة من التعليمات، أو الخطوات الإجرائية لحل مشكلة ما. إذ يمكن توصيف أي مشكلة، حتى تلك التي لاتتعلق بالحوسبة، باستخدام أحد أساليب وضع الخوارزميات.

مثال 1: خوارزمية استخدام الحاسب:

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.

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.

مثال 3: خوارزمية تحديد الرقم الأكبر من مجموعة أرقام:

1. احتفظ بالأرقام ضمن جدول؛
2. أدخل أول أرقام الجدول؛
3. أدخل ثاني أرقام الجدول؛
4. إذا كان الرقم الأول أكبر من الرقم الثاني: احتفظ بالأول ضمن الجدول واحذف الثاني؛
5. وإلا: احتفظ بالثاني ضمن الجدول واحذف الأول؛
6. العودة إلى الخطوة 1 وتكرارها حتى يتبقى رقم واحد في الجدول؛
7. الرقم المتبقي في الجدول هو الرقم الأكبر.

مثال 4: خوارزمية عدّ الأرقام الموجودة ضمن خانات جدول:

بفرض أن لدينا جدول يحوي على عدد من الخانات. ولنفرض أنه قد جرى ملئ عدد من الخانات بأرقام في حين بقيت بقية الخانات فارغة. سنضع خوارزمية لعدّ الخانات المشغولة.

1. احتفظ ضمن عداد بالقيمة 0؛
2. ابدأ بالخانة الأولى؛
3. تحقق من الخانة؛
4. إذا كانت الخانة مملوءة إجمع القيمة واحد إلى القيمة المحتواة ضمن العداد؛
5. إذا انتهت الخانات اذهب إلى الخطوة 8؛
6. إذا لم تنته الخانات انتقل لفحص الخانة التالية؛
7. العودة إلى الخطوة 3؛

أخرج رقم العداد الذي يعبِّر عن عدد الخانات المشغولة.

1-10 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.

عموماً، تمر عملية تطوير نظام برمجي بعدة مراحل أهمها:

1. فهم احتياجات المستخدم بدقة ووضوح؛
2. كتابة وصف النظام البرمجي المطلوب (أي توصيفه).
3. استخدام أدوات التصميم والتحليل، بما في ذلك المخططات التدفقية وغيرها.
4. كتابة تعليمات وإجرائيات النظام؛
5. اختبار جميع مكونات وواجهات النظام، وذلك قبل وضعه في حيز التنفيذ الكامل بهدف التحقق من نجاحه وعمله بالشكل المطلوب.
6. إنشاء تطبيق خاص بإعداد وتنصيب النظام، يتضمن الملفات التنفيذية وملحقاتها؛

تَتَبَع 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؛

1-11 تقنيات التصميم: شبه التشفير

يمكننا اعتماداً على شبه التشفير تحويل خوارزمية إلى شبه برنامج حقيقي وذلك باستخدام تعليمات أساسية تعبر عما يحتاجه المُبرمج من تعليمات أساسية لكتابة البرنامج:

1. تعليمة الدخل: **read**

```
read X;
```

2. تعليمة الخرج: **write**

```
write X;
```

3. عملية إسناد قيمة إلى متحول: **x←5**

```
X←X+5;
```

4. تعليمة الشرط: **if … then begin … end else begin … end**

```
if X>5 then
begin
    write X
end;
```

5. تعليمة تكرار: **while … do begin … end**

```
while X>5 then
begin
   X=X-1
end;
```

ملاحظات:

- تنتهي كل تعليمة من تعليمات البرنامج بفاصلة منقوطة تعبر عن نهاية التعليمة.
- يبدأ البرنامج بكلمة Program مع إسم البرنامج أو الإجرائية، وتكون تعليماته محاطة بكلمتي begin و end للدلالة على بداية ونهاية البرنامج.
- تُعامَل الإجرائية معاملة البرنامج.
- تُستخدم العمليات الحسابية (+ للجمع، - للطرح، * للضرب، / للقسمة) بين المتحولات؛
- تُستخدم عمليات المقارنة (= يساوي، > أصغر، => أصغر أو يساوي، < أكبر، =< أكبر أو يساوي، =!= لايساوي) بين المتحولات.

مثال1: استخدام شبه التشفير لكتابة خوارزمية حساب المتوسط الحسابي لعددين يُدخلهما المُستخدِم

```
Program Average;
begin
read n1;
read n2;
av←(n1+n2)/2;
write av;
end;
```

مثال2: استخدام شبه التشفير لكتابة خوارزمية حساب المتوسط الحسابي لـ 100 عدد يُدخلها المُستخدِم

```
Program Average;
Begin
counter ←0;
sum←0;

while counter<100 do
begin
read n;
sum=sum+n;
end;
av=sum/100;
write av;
end;
```

هناك العديد من تقنيات التصميم التي تُستخدم على عدة مستويات، لكننا سنركز في هذا الفصل على شبه التشفير كونه يُعتبر مرحلة انتقالية بين التوصيف المبدئي، وبين عملية البرمجة الفعلية للبرنامج.

تبدو عملية شبه التشفير مشابهة جداً لتعليمات البرنامج التي ستجري كتابتها فعلياً، حيث يجري استخدام عبارات شبيهة بالعبارات البرمجية، مثل: IF و WHILE، ويجري تحديد تعليمات البرمجة المطلوبة، دون الانتباه للقواعد الحرفية للغة البرمجية المستخدمة أو لاستكمال كافة إجرائيات البرنامج.

يمكننا اعتماداً على شبه التشفير تحويل خوارزمية إلى شبه برنامج حقيقي وذلك باستخدام تعليمات أساسية تعبر عما يحتاجه المُبرمج من تعليمات أساسية لكتابة البرنامج:

1. تعليمة الدخل: **read**
2. تعليمة الخرج: **write**
3. عملية إسناد قيمة إلى متحول: **x←5**.
4. تعليمة الشرط: **if ... then begin ... end else begin ... end**
5. تعليمة تكرار: **while ... do begin ... end**

ملاحظات:

- تنتهي كل تعليمة من تعليمات البرنامج بفاصلة منقوطة تعبر عن نهاية التعليمة.
- يبدأ البرنامج بكلمة Program مع إسم البرنامج أو الإجرائية، وتكون تعليماته محاطة بكلمتي begin و end للدلالة على بداية ونهاية البرنامج.
- تُعامَل الإجرائية معاملة البرنامج.
- تُستخدم العمليات الحسابية (+ للجمع، - للطرح، * للضرب، / للقسمة) بين المتحولات؛
- تُستخدم عمليات المقارنة (= يساوي، > أصغر، => أصغر أو يساوي، < أكبر، =< أكبر أو يساوي، =!= لايساوي) بين المتحولات.

12-1 استراتيجيات وضع الحلول البرمجية

تتضمن استراتيجية وضع حلّ برمجي:

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

تتضمن استراتيجية وضع حلّ برمجي:

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 كالخوارزميات، والمخططات التدفقية 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 وشبه التشفير؛
- 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.

13-1 التصميم من القمة إلى القاعدة (Top-Down Design)

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 والفرضيات الأساسية الخاصة به؛
- كتابة شبه شيفرة الخاص بكل قسم من أقسام النظام البرمجي؛
- ترجمة شبه الشيفرة إلى لغة برمجية.

مثال:

نحتاج لكتابة برنامج لحل معادلات من الدرجة الثانية.

بما أن معادلة من الدرجة الثانية تُكتب على الشكل $ax^2 + bx + c = 0$، لذا فإن حلها يتطلب معرفة 3 عناصر هي: a، و b، و c. إذاً:

دخل البرنامج: ثلاث أعداد حقيقية هي a، b، c تمثل أمثال حدود المعادلة.

بما أن المطلوب هو حلّ المعادلة، فمن الممكن أن يكون للمعادلة حلّ وحيد، أو حلين، أو لايكون لها حلول. إذاً:

خرج البرنامج: إحدى الحالات التالية:

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

لنسأل نفسنا الآن عن الفرضيات التي يجب معالجتها، والحقيقة أنه لا توجد أية فرضيات على قيم a و b و c التي يمكن أن تكون حقيقية أو صحيحة، سالبة أو موجبة.

بالنتيجة تكون خوارزمية الحل بشبه التشفير كما يلي:

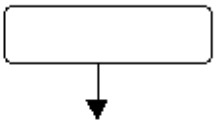
```
program equation;
begin
        read a;
        read b;
        read c;

        delta = b^2 - 4×a×c;

          if delta<0 then
        begin
                write “No Solution”;
        end;

          if delta = 0 then
        begin
                write “One Solution:”;
                sol = -b / (2×a);
                    write sol;
        end;
```

```
        if delta >0 then
        begin
            write “Two Solutions:”;
            sol1 = (-b + sqrt(delta)) / (2 × a);
                sol2 = (-b - sqrt(delta)) / (2 × a);
                write sol1,sol2;
        end;
end;
```

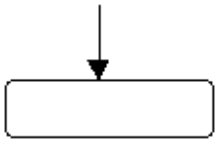
14-1 المخططات التدفقية

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

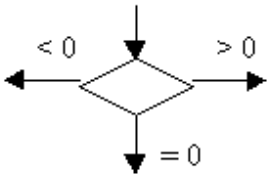
1. البداية:



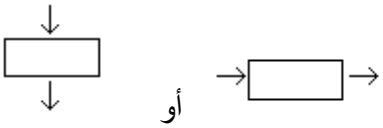
2. النهاية:



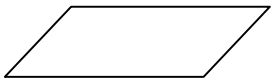
3. الاقتضاء الشرطي:



4. العمليات:



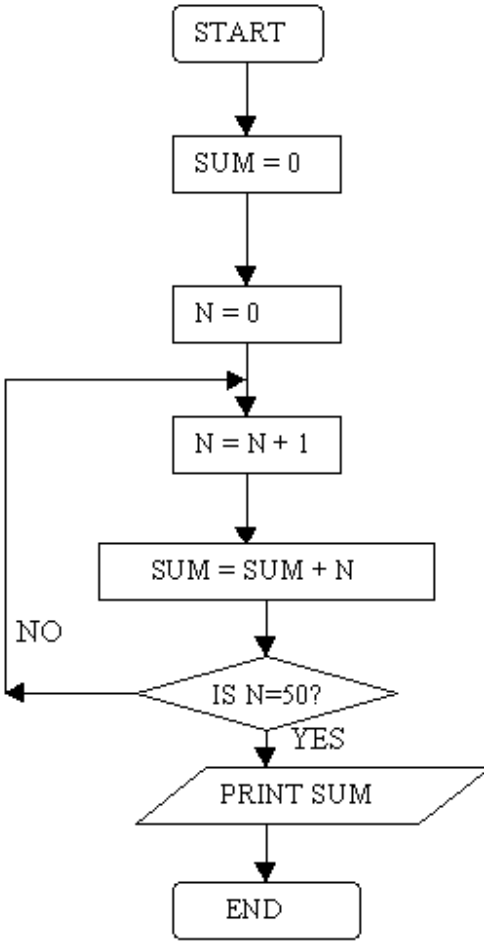
5. الدخل والخرج:



مثال1:

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

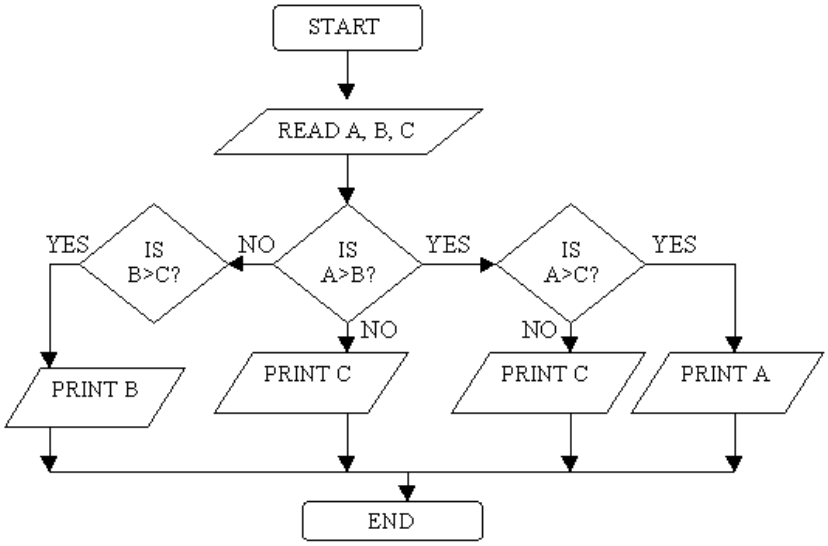
الحلّ:



مثال2:

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

الحلّ:

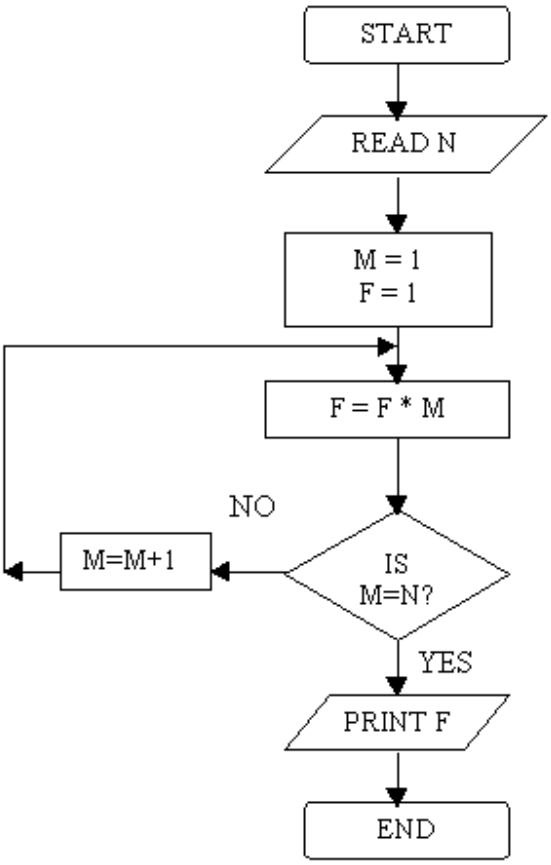


مثال3:

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

$$(N! = N * (N-1) * (N-2) * (N-3) * \ldots * 3 * 2 * 1)$$

الحلّ:



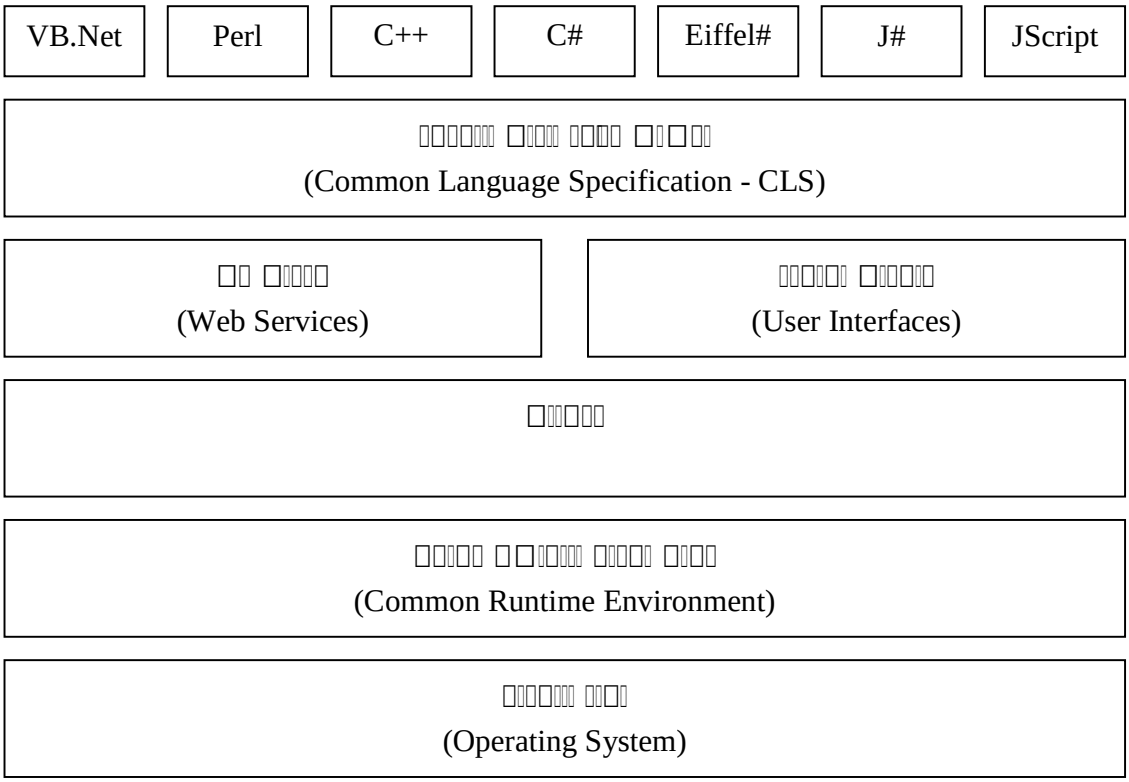
Microsoft Dot Net 1-2

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 المؤسِسة 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 مكونات 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 Web تعمل ضمن محيط مخدم وب.

2-2 بنيان إطار العمل Dot Net Framework



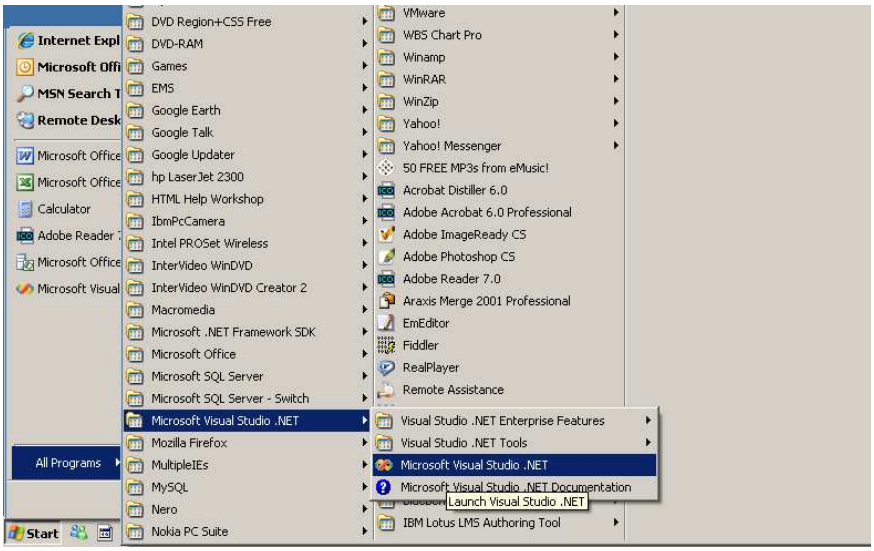
يُقسم بنيان إطار العمل Dot Net إلى مجموعة من الطبقات التي تساعد المُبرمج على كتابة برامجه وتحويلها إلى برامج تنفيذية.

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.

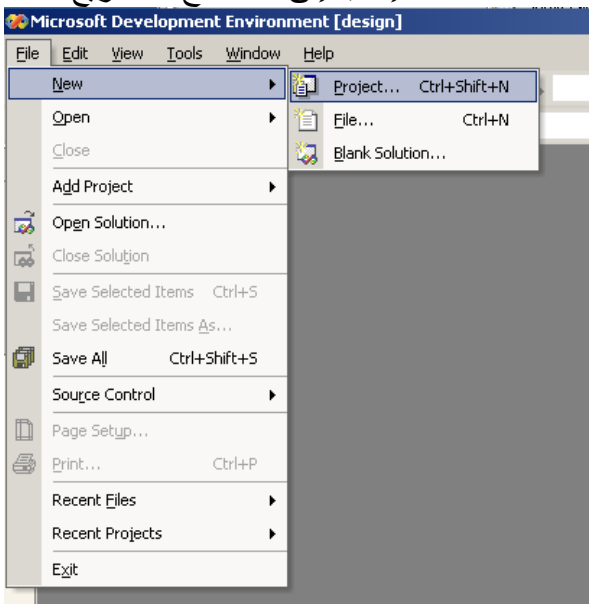
3-2 بداية سريعة مع C++

بعد تثبيت Visual Studio.net نفذ العمليات التالية:

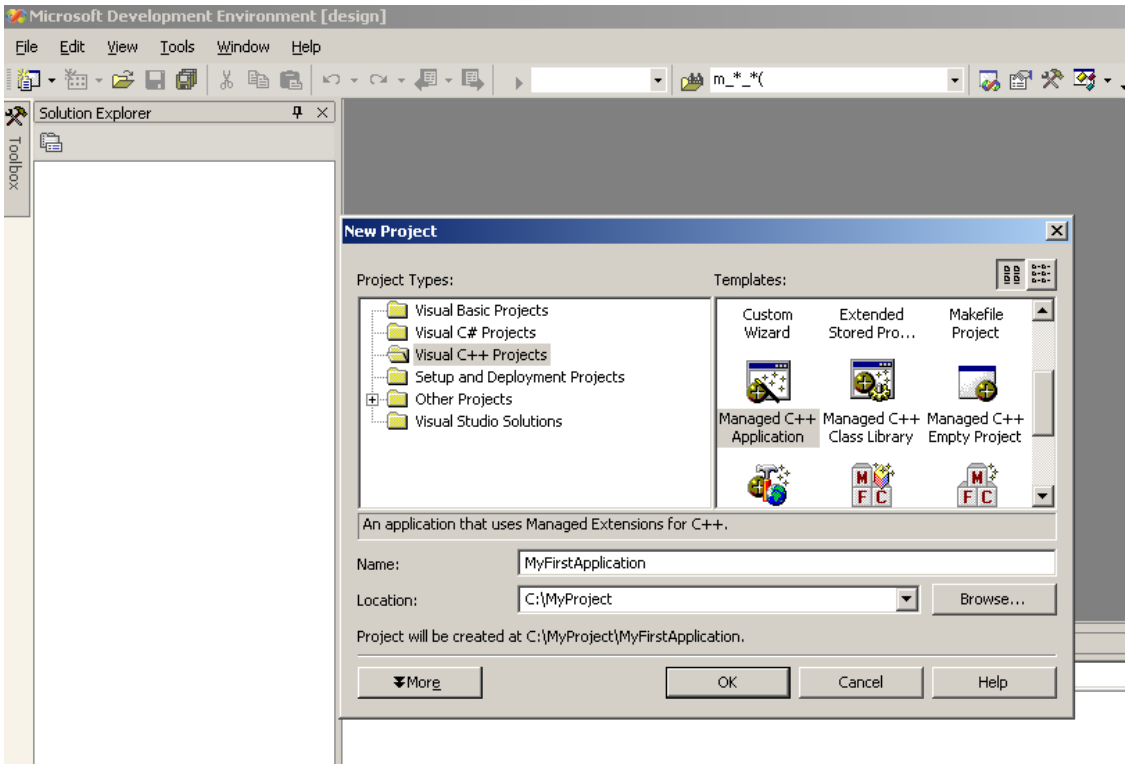
1. إذهب إلى زر البداية Start وإلى مكان إقلاع Microsoft Visual Studio .Net:



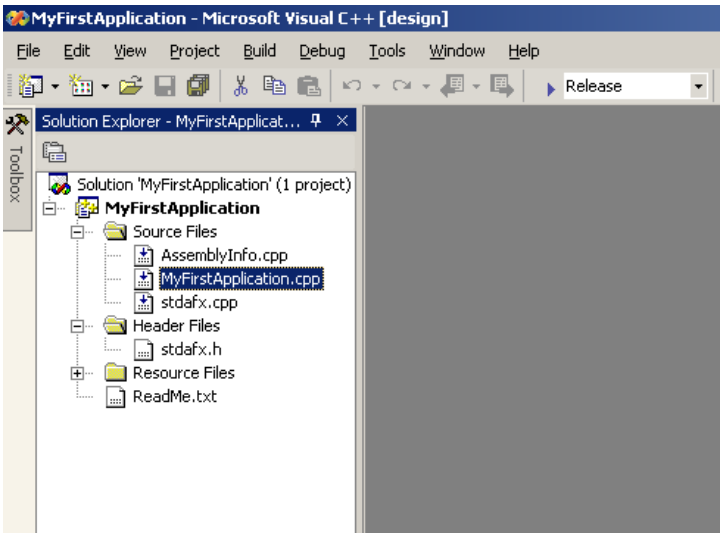
2. عند إقلاع Visual studio .Net إذهب إلى نافذة فتح المشاريع الظاهرة في الشكل:



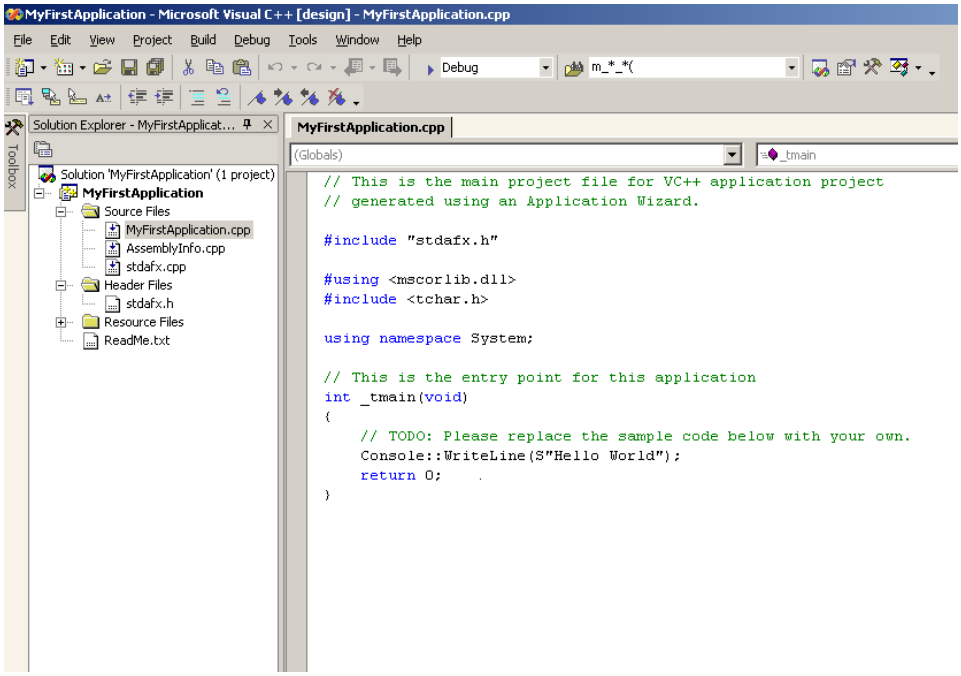
3. يمكنك عندها اختيار C+ Project من النافذة اليسارية، وConsol Application من النافذة اليمينية مع تحديد اسم التطبيق ومكان تخزينه في الأسفل، كما هو موضح في الشكل:



4. عند حصولك على واجهة التطبيق، انقر مرتين على الصف MyFirstApplication.cpp، واذهب إلى واجهة View Code:



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

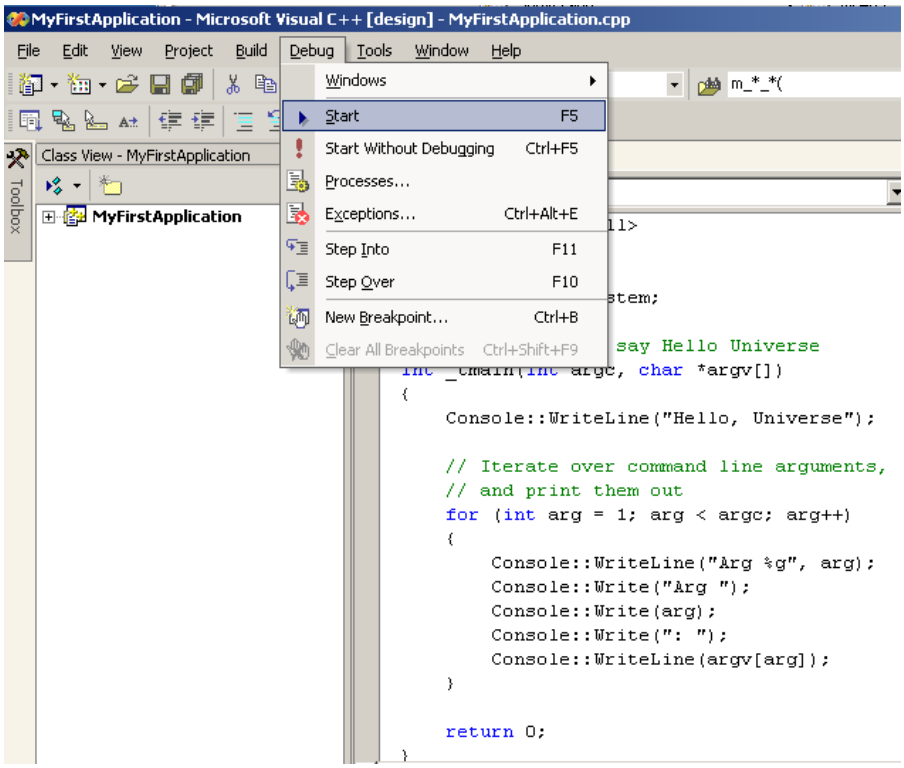


6. يمكنك كتابة برنامجك الأول الموضح فيما يلي ضمن إطار النص البرمجي المُعطى:

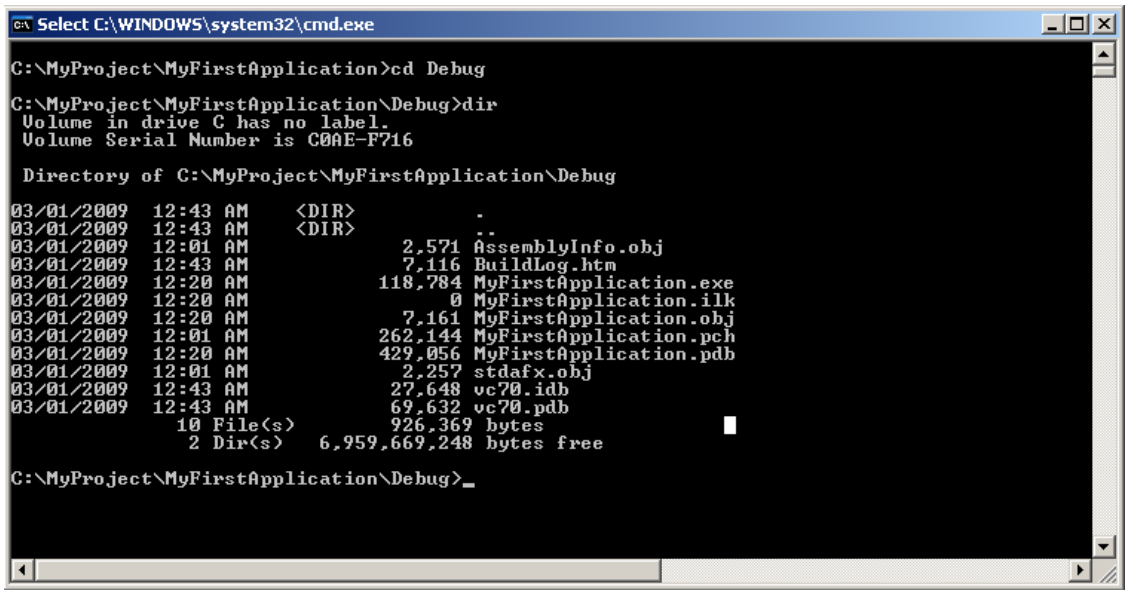
```
#include "stdafx.h"

#using <mscorlib.dll>
#include <tchar.h>

using namespace System;

// I would like to say Hello Universe
int _tmain(int argc, char *argv[])
{
	Console::WriteLine("Hello, Universe");

	// Iterate over command line arguments,
	// and print them out
	for (int arg = 1; arg < argc; arg++)
	{
		Console::Write("Arg ");
		Console::Write(arg);
		Console::Write(": ");
		Console::WriteLine(argv[arg]);
	}
```

```
    return 0;
}
```

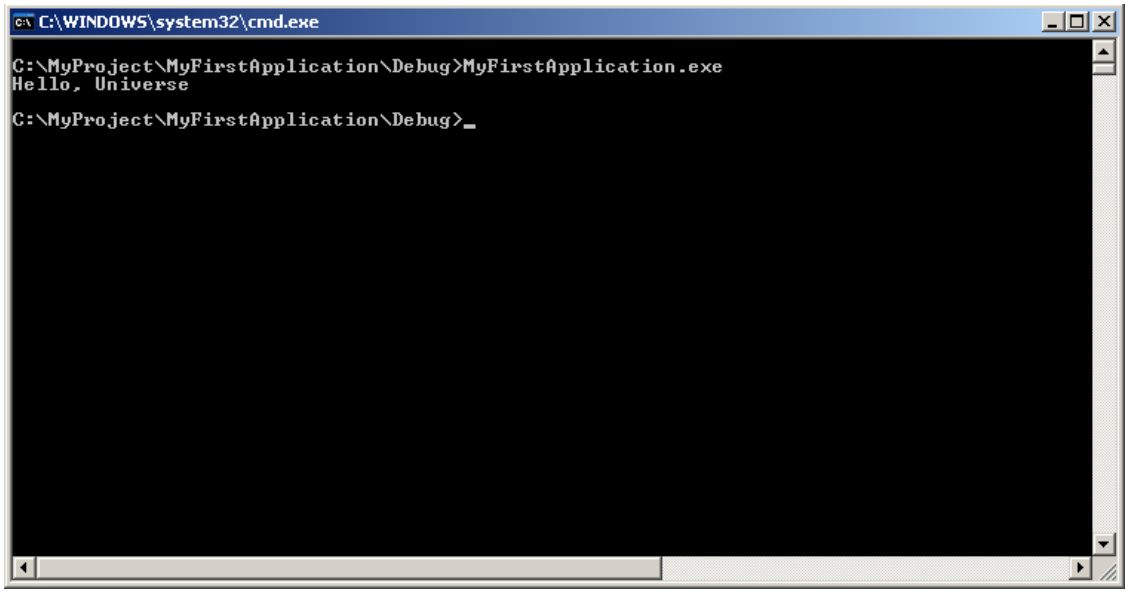
7. يمكنك ترجمة برنامجك والتحقق من صحته بالذهاب إلى واجهة التنفيذ Debug ومن ثم Start:



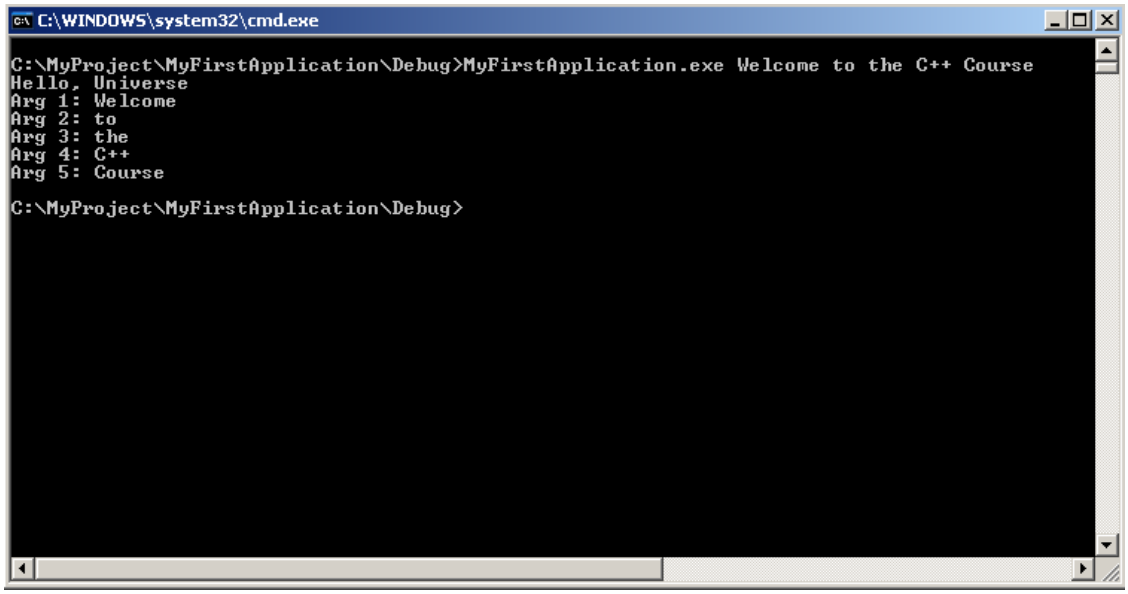
8. يمكن بعد ذلك تنفيذ البرنامج اعتباراً من Dos Command Prompt كما يظهر من الشكل تحت اسم MyFirstApplication.exe:



9. يمكن بعد ذلك تنفيذ البرنامج بدون مُعاملات:



أو مع معاملات:



4-2 تحليل النص البرمجي

```
#include "stdafx.h"


#using <mscorlib.dll>
#include <tchar.h>


using namespace System;


// I would like to say Hello Universe
int _tmain(int argc, char *argv[])
{
      Console::WriteLine("Hello,
Universe");


        // Iterate over command line
arguments,
        // and print them out
        for (int arg = 1; arg < argc;
arg++)
        {
                Console::WriteLine("Arg
```

```
%g", arg);
                Console::Write("Arg ");
                Console::Write(arg);
                Console::Write(": ");

        Console::WriteLine(argv[arg]);
        }

    return 0;
}
```

- تُستَخدَم العبارة "#include" للدلالة على ملفات محددة تقدم مجموعة من الإجرائيات والصفوف الجاهزة والمعرفة التي يمكن إستخدامها في البرنامج، وغالباً ما تكون هذه الملفات من مكتبة C++ المعيارية.

- تُستَخدَم العبارة "#using" للدلالة على مكتبات محددة تحتوي كل منها على ملفات تقدم مجموعة من الإجرائيات والصفوف الجاهزة والمعرفة التي يمكن إستخدامها في البرنامج. فعبارة "#using <mscorlib.dll>" تستعمل للدلالة على إحدى المكتبات الرئيسة التي تتضمن تعريف ماندعوه فضاء الأسماء "system"

- تُستَخدَم العبارة "using system" للدلالة على 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 واجهة التعليمات النصيّة التي ندعوها "Command Prompt" كما لاحظنا عند تشغيل البرنامج.

- يستخدم الصف "Console" الإجرائية "Write" والإجرائية "Writeline" لكتابة سلسلة محارف وإظهارها على واجهة التعليمات النصيّة.

- يُعرِّف المثال إجرائية "_tmain" التي يمكن اعتبارها نقطة إنطلاق لتنفيذ المثال؛

- تقوم الإجرائية بإظهار عبارة "Hello, Universe" بالإضافة إلى أية عبارة أخرى يُدخلها المُستثمر عند استدعائه للتطبيق من واجهة التعليمات النصيّة.

يجدر الإشارة هنا إلى أنه يمكن تحرير البرنامج المذكور باستخدام الطريقة التقليدية لإخراج البيانات والمعلومات عوضاً عن إستخدام الصف Console، وذلك بإعتماد التعليمة cout المعرّفة في ملف iostream.h من مكتبات لغة C المعيارية.

```
#include "stdafx.h"


#include <iostream.h>


// I would like to say Hello Universe

int main(int argc, char *argv[])

{

      cout << "Hello,Universe\n";


        // Iterate over command line arguments,

        // and print them out

        for (int arg = 1; arg < argc; arg++)

        {
```

```
                cout << "Arg ";
                cout << arg;
                cout << ": ";
            // \n is used to move to a new line
                cout << argv[arg] << "\n";
        }


    return 0;
}
```

5-2 الأنماط الأساسية

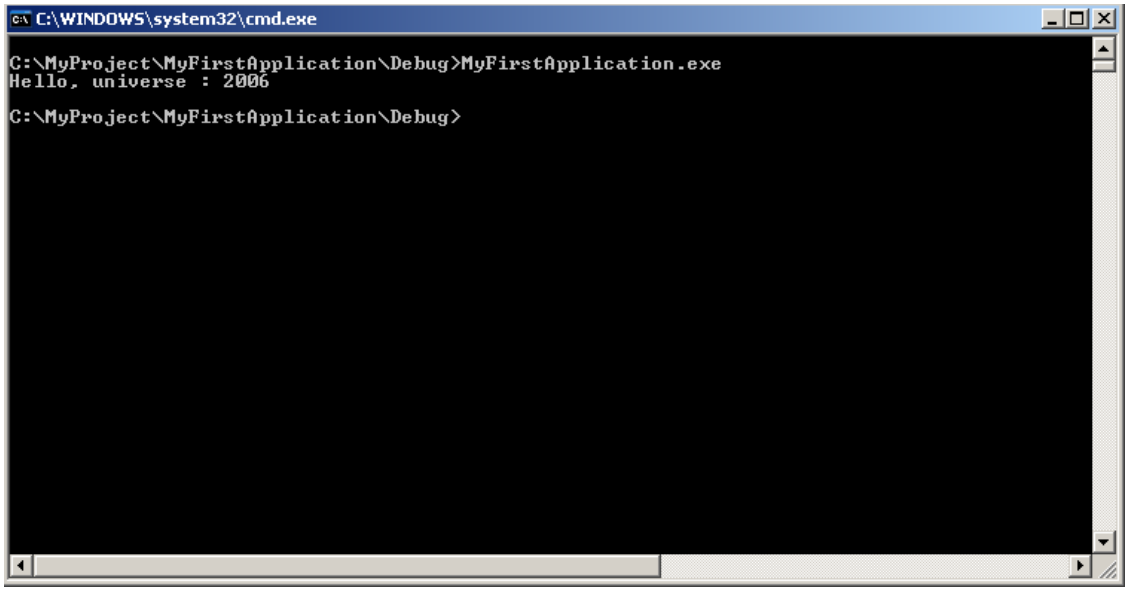
يمكن للمُبرمج استخدام أحد الأنماط البسيطة التي تظهر في الجدول لتعريف متحولاته. ويؤدي تعريف كل نوع من أنواع المتحولات الظاهرة إلى حجز جزء من الذاكرة يُقدر بالبايت (Byte) ويتعلق بالنمط المُستَخدَم.

النمط	عدد الـ Byte التي يحجزها في الذاكرة	توصيف
short	2	قيمة صحيحة ذات إشارة تتراوح بين -2^{15} و $+2^{15}$
unsigned short	2	قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{16}-1$
int	4	قيمة صحيحة ذات إشارة تتراوح بين -2^{31} و $+2^{31}$
unsigned int	4	قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{32}-1$
long	8	قيمة صحيحة ذات إشارة تتراوح بين -2^{63} و $+2^{63}$
unsigned long	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
char	1	حرف يتبع الترميز ASCII (قيمته بين -128 و 127)
bool	1	يأخذ إحدى القيمتين TRUE أو FALSE

مثال:

```
#include "stdafx.h"


#using <mscorlib.dll>
#include <tchar.h>


using namespace System;


int _tmain(int argc, char *argv[])
{
        int y = 2006;
        char* s = "Hello, universe
: ";
        Console::Write(s);
        Console::Write(y);
        Console::WriteLine();


    return 0;
}
```

لنحصل على النتيجة التالية عند التنفيذ:



6-2 الأنماط المُنَمَّرة ENUM

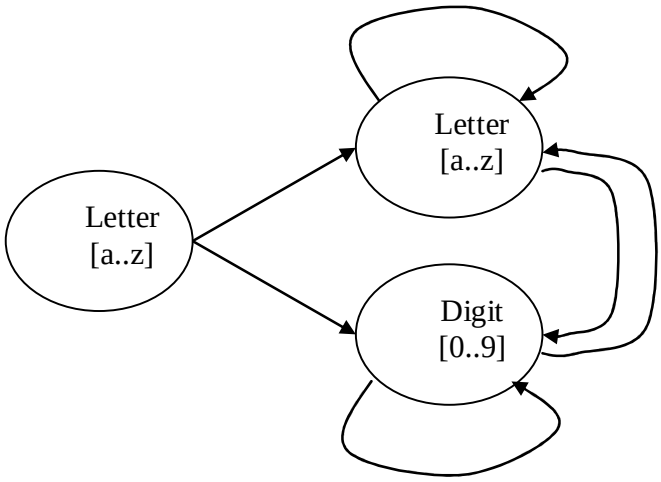
يُعتبر النمط enum من الأنماط البسيطة التي تساعد في تعريف مجموعة من القيم الثابتة بأسمائها الحقيقية:

enum Day {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
By default : Monday=0, Tuesday=1, Wednesday=2, Thursday=3, Friday=4, Saturday=5, Sunday=6

- يمتلك كل عنصر من عناصر النمط enum نمطه الخاص الذي ينتمي إلى أحد الأنماط الصحيحة: short، أو unsigned short، أو int، أو unsigned int، أو long، أو unsigned long.
- يكون النمط int هو النمط التلقائي لعناصر النمط enum، ويأخذ العنصر الأول من عناصر النمط enum الرقم 0 ويستمر ترقيم العناصر حتى القيمة n-1 في حال وجود n عنصر.

7-2 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 لما سبق، يمكن استخدام أحد الحروف التالية: "_"، أو "μ" ، أو الأحرف ذات الإشارات (Characters with Accents) كالأحرف الفرنسية.

مثال:

تعريف متحولات بدون قيم أولية:

```
int i;
int j;
double d;
char ch;
bool b;
```

تعريف متحولات مع إسناد قيم أولية لها:

```
int i=50;
int j=10;
double d=2.3;
char ch='K';
bool b=false;
```

8-2 الثوابت في C++

تمتلك لغة C++ كلمة مفتاحية للدلالة على المتحولات التي لا يمكن لقيمها أن تتغير: "const"، حيث تُستَخدَم هذه الكلمة عند تعريف المتحول أو الثابت قبل أي كلمة مفتاحية أخرى.

يجب على الكلمات المُعرَّفة كـ const أن تأخذ قيم عند تعريفها مباشرةً، ويمكن تعريف عضو من صف أو متحول ضمن إجرائية كـ const فيصبح ثابتاً.

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:

```
#include "stdafx.h"
#using <mscorlib.dll>
#include <tchar.h>
using namespace System;

int _tmain(int argc, char *argv[])
{
        const int num=0;
        num = 10;

        if ( num < 0 )
                Console::WriteLine("Negative");
```

```
        else
                Console::WriteLine("Positive");


    return 0;
}
```

9-2 العمليات في C++ وأفضلياتها

العمليات الحسابية

تقدم C++ كغيرها من لغات البرمجة مجموعة العمليات الحسابية الاعتيادية التي نستعرضها في هذه الشريحة.

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;

عمليات المُقارنة

تقدم C++ كغيرها من لغات البرمجة مجموعة عمليات المقارنة المنطقية التي نستعرضها في هذه الشريحة.

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);
!=	لا يساوي	6	a!=b; (x+y) != (z+r);

العمليات المنطقية

تقدم C++ كغيرها من لغات البرمجة مجموعة العمليات المنطقية الاعتيادية التي نستعرضها في هذه الشريحة.

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);

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

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 z < (x+y)؛
  - 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 &&؛

10-2 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 تمتلك 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 قرائة قيمة ذات نمط محرفي ومحاولة اسنادها لمتحول يعبر عن عدد صحيح) في حال نفذنا البرنامج التالي:

```
#include "stdafx.h"
#using <mscorlib.dll>
#include <tchar.h>
using namespace System;

int _tmain(int argc, char *argv[])
{
        int num;
        Console::WriteLine("Enter The Requested Value:
");

        num = Console::ReadLine();
```

```
        if ( num < 0 )
                Console::WriteLine("Negative");
        else
                Console::WriteLine("Positive");

    return 0;
}
```

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:

```
#include "stdafx.h"
#using <mscorlib.dll>
#include <tchar.h>
using namespace System;


int _tmain(int argc, char *argv[])
{
        int num;
        String* s;
        Console::WriteLine("Enter The Requested Value:
");
```

```
        s = Console::ReadLine();
        num = Int32::Parse(s);

        if ( num < 0 )
                Console::WriteLine("Negative");
        else
                Console::WriteLine("Positive");

    return 0;
}
```

ملاحظة: String هو نمط من ضمن فضاء الأسماء System يستعمل لتعريف المتحولات التي تحتوي على سلسلة محارف. "*" قد تستخدم إلى جانب أي نمط من الأنماط للدلالة على أن المتحوّل يخزّن المؤشّر إلى القيمة وليس القيمة في حد ذاتها. فالمتحول في هذه الحالة يحتوي على عنوان الذاكرة حيث تمّ تخزين القيمة الفعلية.

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

الصف	النمط
Int16	short
UInt16	unsigned short
Int32	int

UInt32	unsigned int
Int64	Long
UInt64	unsigned long
Single	Float
Double	Double

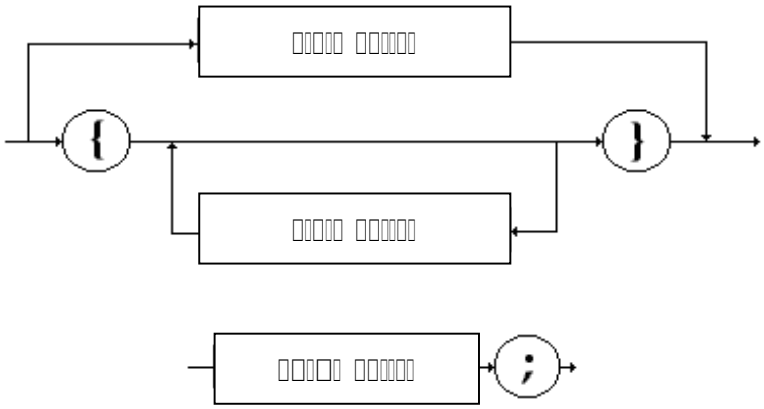
يجدر الإشارة هنا إلى أنه يمكن تحرير البرنامج المذكور أعلاه باستخدام الطريقة التقليدية لإدخال البيانات والمعلومات عوضاً عن إستخدام الصف Console، وذلك بإعتماد التعليمة cin المعرّفة في ملف iostream.h من مكتبات لغة C المعيارية.

```
#include "stdafx.h"
#include <iostream.h>

int main(int argc, char *argv[])
{
        int num;

    // \n is used to move to a new line
        cout << "Enter The Requested Value: \n";

        cin >> num;

        if ( num < 0 )
                cout << "Negative\n";
```

```
        else
                cout << "Positive\n";


    return 0;
}
```

1-3 قواعد عامة

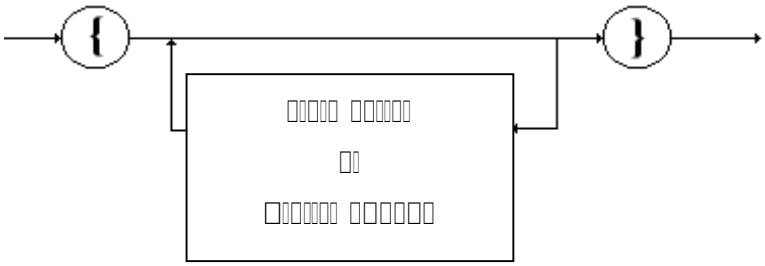
- تم الأخذ بقسم كبير من المعيار ANSI الخاص بلغة C في لغة C++؛
- يمكن أن نكتب تعليمة كاملة محتواة بين قوسين { } أو كتبها دون أقواس؛
- تنتهي أي تعليمة بسيطة (كتعليمة الإسناد) بفاصلة منقوطة؛



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

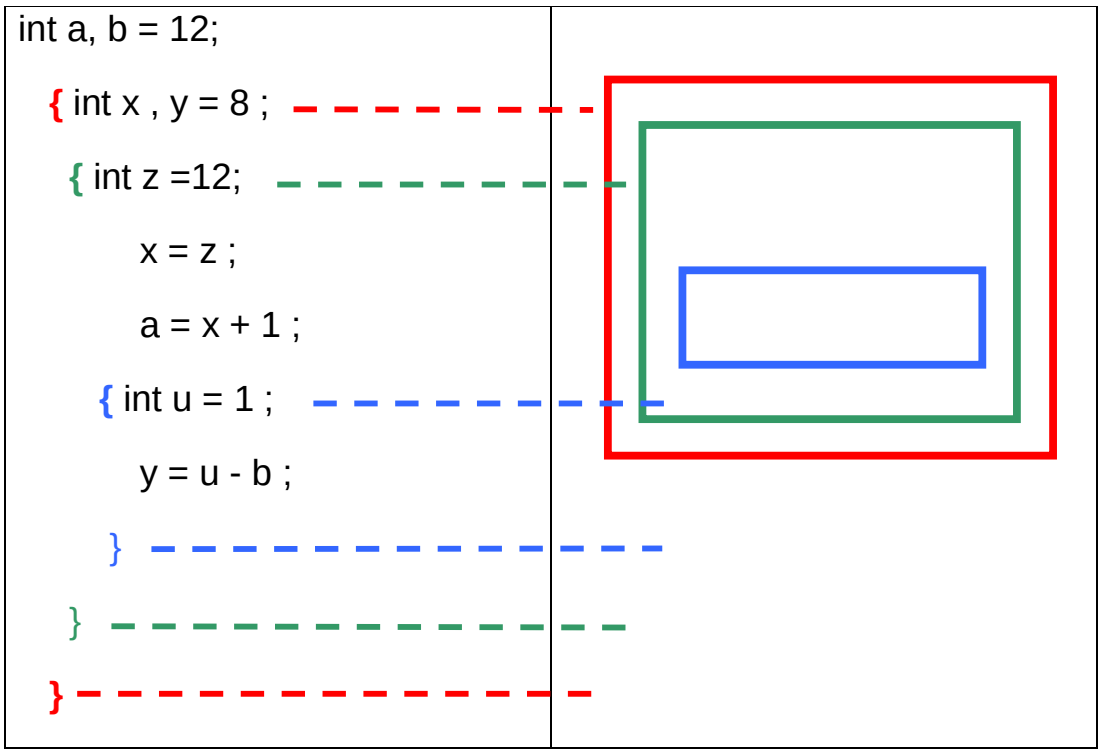
2-3 كتل التعليمات وتعريف مدى المتحول

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.

مثال:



نُعرِّف مدى المتحول بالكتلة التي يكون المتحول فيها مُعرفاً، 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.

مثال:

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.

3-3 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 العبارات 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 العبارة، وتأخذ b قيمة جديدة هي 8 في الحالة الثالثة ولكن ضمن عمليات 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=12
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
```

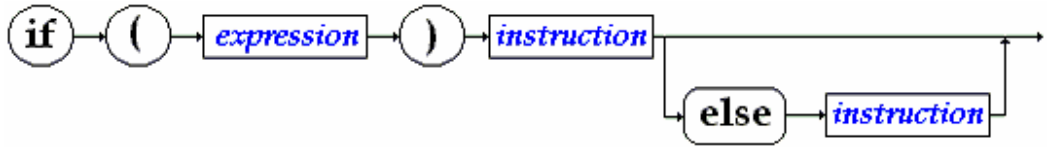
تكون قيم المتحولات في الحالات الثلاث بعد انتهاء عمليات الإسناد:

int a , b = 56 ;	a=???, b=56
a = -8 ;	a=-8
a += b ;	a=48
b *= 3 ;	b=168

يمكن استخدام الإسناد ضمن العبارات الحسابية والمنطقية، وعلى شكل إسناد متعدد. كما يمكن استخدام إسناد مزود بعملية من العمليات الحسابية أو المنطقية بشكل مكافئ للإسناد البسيط.

4-3 العبارة الشرطية

تأخذ العبارة الشرطية الشكل القواعدي التالي:



بحيث يكون لأي شرط أحد شكلين:

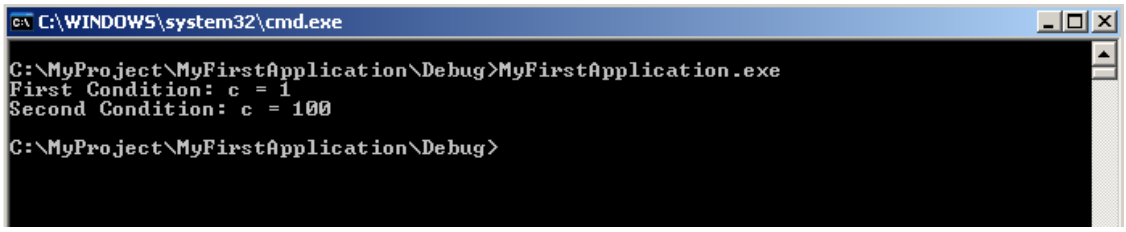
```
if ( Expression ) Instructions Bloc ;

                أو

if ( Expression ) Instructions Bloc ; else Instructions
Bloc ;
```

مثال:

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

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

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

Console::Write("Second Condition: c = ");
```

```
Console::WriteLine(c);
```

وتكون النتيجة:



5-3 غموض العبارة الشرطية

يمكن أن تظهر العبارة 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 عبارة else لعبارة if.

بشكل عام تكون عبارة else تابعة لعبارة 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;
```

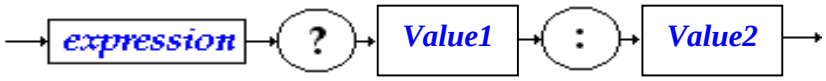
طبعاً، يمكن للأقواس في حال استخدامها أن تحلّ الغموض وفقاً لتوزعها. ففي الحالتين التاليتين تُزيل الأقواس الغموض تماماً وتحدد تبعية عبارة else لعبارة if:

```
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;
```

6-3 اختضار العبارة الشرطية

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



في حال تحقق الشرط المحدد في *expression*

يجري إرجاع *Value1*

وإلا يجري إرجاع *Value2*.

وتكافئ التعليمة السابقة التعليمة:

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

مثال:

نقرأ العبارة الشرطية المُختصرة التالية:

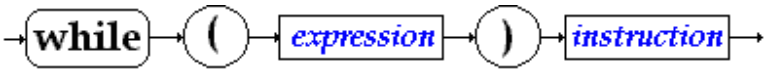
```
int a,b,c ;

....

c =   a == 0  ?  b  :  a+1 ;
```

كما يلي:

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

7-3 عبارات التكرار: while، do، for

1. يكون لعبارة التكرار الحلقية while الشكل القواعدي التالي:

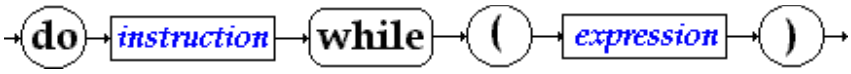


- يجري أولا اختبار الشرط *expression* الذي يعيد قيمة منطقية صح أو خطأ
  - فإذا كانت صح يجري تنفيذ *instruction*؛
  - وإلا يجري الخروج دون تنفيذ *instruction*.
- بعد كل تنفيذ للتعليمات *instruction*، يجري اختبار الشرط *expression* للتأكد من أن قيمته المنطقية مازالت صح:
  - فإذا كانت صح نستمر في تكرار *instruction* مرة أخرى؛
  - وإلا يجري التوقف عن تكرار *instruction*.

مثال:

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

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

2. ويكون لعبارة التكرار الحلقية do الشكل القواعدي التالي:

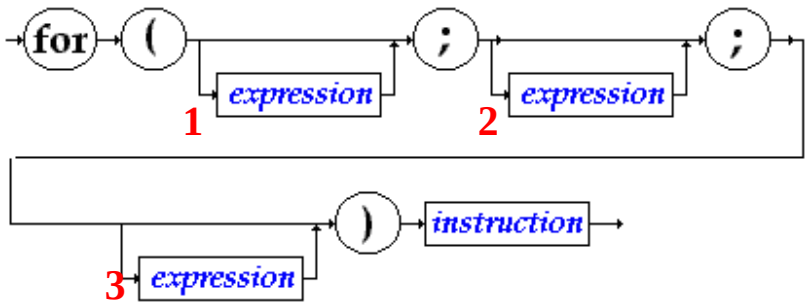


- يجري أولا تنفيذ التعليمات *instruction*
- يجري بعدها اختبار الشرط *expression*:
  - فإذا كان صحيحاً نستمر في تنفيذ *instruction*.
  - وإلا يجري التوقف.
- بعد كل تنفيذ للتعليمات *instruction*، يجري اختبار الشرط *expression* للتأكد من أن قيمته المنطقية مازالت صح:
  - فإذا كانت صح نستمر في تكرار *instruction* مرة أخرى؛
  - وإلا يجري التوقف عن تكرار *instruction*.

مثال:

```
int j=0;
int y=0;

do
{
        y=y+10;
        j++;
} while (j<=10);
```

3. ويكون لعبارة التكرار الحلقية 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;
```

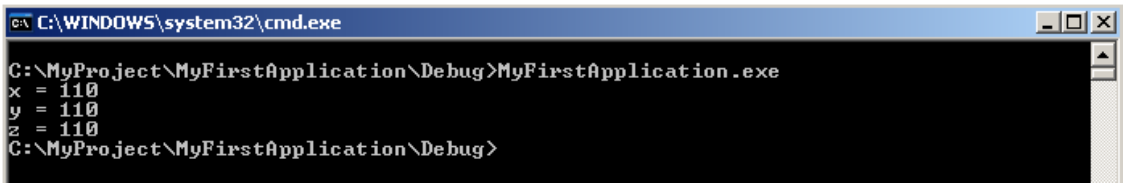
مثال:

تكون العبارات الثلاث متكافئة في البرنامج التالي:

```
int x=0;
int y=0;
int z=0;

int i=0;
while (i<=10)
{
        x=x+10;
        i++;
}
Console::Write("x = ");
Console::WriteLine(x);

i=0;
do
{
        y=y+10;
        i++;
```

```
} while (i<=10);
Console::Write("y = ");
Console::WriteLine(y);

for (i=0; i<=10; i++)
        z=z+10;
Console::Write("z = ");
Console::Write(z);
```

وتكون النتيجة:

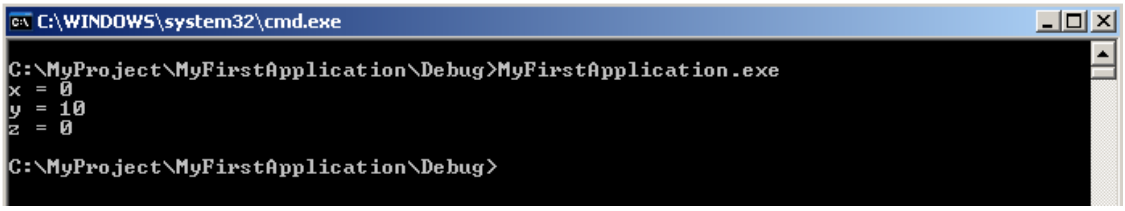


في حين لا يكون هناك تكافؤ في الحالة التالية:

```
int x=0;
int y=0;
int z=0;

int i=11;
while (i<=10)
{
        x=x+10;
        i++;
}
Console::Write("x = ");
Console::WriteLine(x);

i=11;
do
{
        y=y+10;
        i++;
} while (i<=10);
```

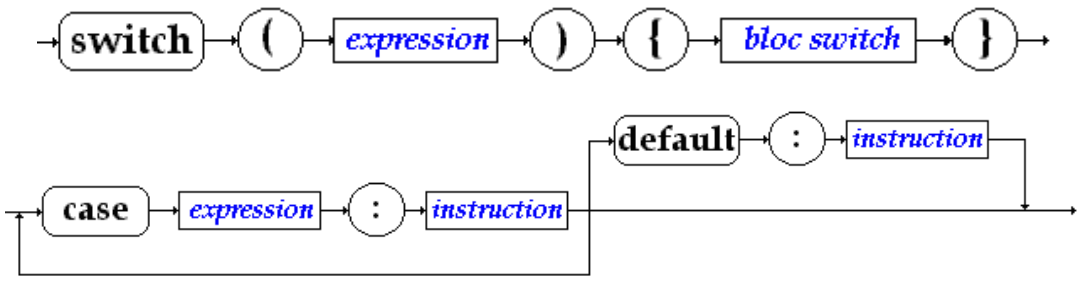
```
Console::Write("y = ");
Console::WriteLine(y);


for (i=11; i<=10; i++)
        z=z+10;
Console::Write("z = ");
Console::WriteLine(z);
```

وتكون النتيجة:



8-3 عبارة الوصل: switch … case

يكون لعبارة switch … case الشكل القواعدي التالي:



نقرأ العبارة كمايلي:

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

مثال:

```
int  x = 10;
switch (x+1)
{
        case 11 :    Console::WriteLine(">> case 11");
                     break;

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

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

تكون النتيجة هي تطابق العبارة (x+1) مع الحالة الأولى أي (case 11).

9-3 تمارين

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

الحل:

```
#include "stdafx.h"
#using <mscorlib.dll>
#include <tchar.h>
using namespace System;

int _tmain(int argc, char *argv[])
{
        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::Write("\nClass average is ");
        Console::WriteLine(average);

    return 0;
}
```

بفرض أن لديك مجموعة من 10 طلاب، اكتب برنامج 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:

```
#include "stdafx.h"
#using <mscorlib.dll>
#include <tchar.h>
using namespace System;

int _tmain(int argc, char *argv[])
{
        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::WriteLine( "Enter 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::Write("Passed: ");
      Console::WriteLine(passes);
      Console::Write("Failed: ");
      Console::WriteLine(failures);

      Console::Write("Percentage of passing students:
");
      Console::WriteLine(passes * 10);

   return 0;
}
```

1-4 مقدمة

- **المنهجية غرضية التوجه:** عبارة عن منهجية للتطوير والنمذجة مبنية على مفاهيم وأغراض.
- **البرمجة غرضية التوجه:** هي طريقة بديلة عن البرمجة التقليدية، تعتمد على أغراض، حيث يحتوي كل غرض على معطيات وطرائق للتعامل معه.
- **الغرض:** يُمثّل كل غرض كيان من العالم الحقيقي مُميَّز الهوية مع واصفات مميزة وإمكانية العمل بنفسه والتفاعل مع الأغراض الأخرى.
- **مثال:**
  - الغرض: شخص
  - واصفاته: رقم الضمان الاجتماعي، الاسم الأول، الاسم الثاني، تاريخ الميلاد، العنوان.

تُعتبر البرمجة غرضية التوجه طريقة بديلة عن البرمجة التقليدية، وتعتمد هذه البرمجة على استخدام مفهوم الغرض، حيث يُمثّل كل غرض كيان من العالم الحقيقي مُميَّز الهوية مع واصفات مميزة، وإمكانية العمل بنفسه والتفاعل مع الأغراض الأخرى.

يتم تعريف الغرض من خلال مُميِّز الهوية الخاص به، حيث يُسنَد للغرض لحظة بنائه ولايمكن تغييره أبداً، ويُحذف لحظة حذف الغرض.

يجري تحديد كل غرض من خلال مجموعة من الواصفات، وتتميز النسخ المختلفة من الغرض عن طريق القيم المختلفة للواصفات، حيث تملك كل نسخة قيم مختلفة لهذه الواصفات.

2-4 أساسيات التقانة غرضية التوجه

- يتألف العالم من حولنا من أغراض يكون كل منها في حالة محددة تحددها القيم الحالية لصفات الغرض.

- لكل غرض من الأغراض الحقيقية هوية (identity)، وهي خاصة ثابتة نميز من خلالها بين غرض وآخر.

قد يكون إجراء محاكاة مع أغراض حسية من الحياة الواقعية طريقة جيدة لشرح وتوضيح المفاهيم غرضية التوجه، إذ يتألف العالم من حولنا من أغراض يكون كل منها في حالة محددة تحددها القيم الحالية لصفات الغرض.

فعلى سبيل المثال يتواجد فنجان القهوة على مكتبي في الحالة "مملوء" لأنه مصمم بحيث يستوعب السوائل ومازالت القهوة موجودة فيه، وعندما لا تبقى هناك قهوة في الفنجان يصبح في الحالة "فارغ" وإذا سقط على الأرض وتحطم سيصبح في الحالة "مكسور".

إلا أن فنجان القهوة كائن سلبي، فهو لا يتميز بسلوك خاص (behavior)، بالمقابل لا يمكن قول الأمر نفسه بالنسبة لكلب أو شجرة، فالكلب ينبح، والشجرة تنمو، وللأغراض الحقيقية عادة سلوك.

لكل غرض من الأغراض الحقيقية هوية (identity)، وهي خاصة ثابتة نميز بواسطتها بين غرض وآخر، فإذا كان على مكتبي فنجانا قهوة من المجموعة نفسها يمكنني القول إن الفنجانين متساويان لكنهما غير متطابقين، فهما متساويان لأن لهما قيم الخصائص نفسها و لهما الشكل والحجم نفسه، وكلاهما أسود مثلاً، لكنهما ليسا متطابقين في اللغة غرضية التوجه، لأن هناك اثنان يمكنني أن أستخدم أياً منهما وفقا لرغبتي.

الغرض

- يتألف النظام غرضي التوجه من مجموعة أغراض متعاونة، فكل شيء في النظام غرضي التوجه هو عبارة عن غرض.

- يُعبَّر عن الغرض في لغة غرضية التوجه كمستطيل من خلال الصف الذي ينتمي إليه؛

- يجري تدوين العمليات الخاصة بغرض ضمن الصف الذي ينتمي إليه الغرض لأنها مشتركة بين جميع الأغراض.

الغرض هو مثل من "شيء"، فقد يكون من أمثلة عديدة للشيء نفسه، ففنجان القهوة الموجود لدي هو مثل من مجموعة الفناجين الموجودة.

يتألف النظام غرضي التوجه من مجموعة أغراض متعاونة، فكل شيء في النظام غرضي التوجه هو عبارة عن غرض.

من المهم أن نشير هنا إلى أن تدوين الغرض لا يحوي جزءاً للعمليات (Operations) التي يمكن أن ينفذها الغرض، ويعود هذا إلى كون العمليات متطابقة في كل الأمثال ولن يكون تكرار ذكرها في كل مثل مناسباً. لذلك يجري تخزين العمليات على مستوى الصف.

واصفات الغرض

- يمكن أن تأخذ الواصفة قيمة واحدة أو عدة قيم.

- يمكن أن تُشير واصفات الغرض إلى غرض أو عدة أغراض أخرى.

- حالة الغرض: هي عبارة عن مجموعة القيم التي تأخذها واصفاته في لحظة معينة، ويمكن أن تتغير هذه الحالة، في حين يبقى مميز هوية الغرض نفسه.

يمكن أن تُشير واصفات الغرض إلى غرض أو عدة أغراض أخرى، كما يمكن أن تأخذ قيمة واحدة أو عدة قيم.

يُستخدَم مميز هوية الغرض من أجل الربط بين غرضين، فمثلاً لدينا الغرض (طالب) وواصفة لهذا الغرض (المواد الدراسية) تشير إلى مجموعة من المواد. عندها تحوي الواصفة (المواد الدراسية) على مميز هوية غرض يحوي على قائمة من أغراض المواد، وبذلك يتم الربط بين الغرضين.

طرائق الغرض

- الطريقة: هي عبارة عن رماز يُستخدَم لتنفيذ عملية معينة على واصفات الغرض، ولكل طريقة اسم ويمكن أن تملك مجموعة معاملات.

- كل العمليات التي يمكن تنفيذها على الغرض يجب أن تتحقق من خلال طرائق.

- الكبسلة: هي إمكانية اخفاء البنية الداخلية للغرض (الواصفات والطرائق) عن الأغراض الأخرى.

يجري طلب تنفيذ طريقة معينة من غرض معين من خلال إرسال رسالة إلى الغرض تحوي اسم الطريقة والمعاملات المطلوبة لتنفيذها، حيث يقوم الغرض المعني بتنفيذ الطريقة وإعادة النتيجة في حال وجودها.

من الواضح أن أي غرض لا يمكن أن يصل للبنية الداخلية لغرض آخر، إنما يتم التخاطب بينهما عن طريق رسائل تتضمن طلب تنفيذ طرائق معينة. وهذا ما يدعى بمفهوم الكبسلة أي إمكانية إخفاء البنية الداخلية للغرض (الواصفات والطرائق) عن الأغراض الأخرى.

3-4 تمارين

1- اقترح غرضاً لتمثيل الوقت يحوي على مجموعة من الواصفات التي تعبر عن الوقت على نحو دقيق وعلى مجموعة من الطرائق التي تتعامل مع هذه الواصفات وتسمح بقراءتها أو تعديلها.

الحل:

Class Time:

- Class Attributes:
  - Hours;
  - Minutes;
  - Seconds;
- Class Methods:
  - Set_Time(H, M, S);
  - Get_Time( );

2- اقترح غرضاً لتمثيل شخص (ذاتية شخص) واقترح مجموعة من الطرائق للتعامل مع واصفاته.

الحل:

Class Person:

- Class Attributes:
  - First_Name;
  - Second_Name;
  - National_ID;
  - Address;
  - Date_Of_Birth;
- Class Methods:
  - Set_FullName(FN, SN);
  - Get_FullName( );
  - Set_NationalID( );
  - Get_NationalID( );
  - Set_Address(A);
  - Get_Address( );
  - Set_Date_Of_Birth(DOB)
  - Get_Date_Of_Birth( );

1-5 مقدمة

الصف هو عبارة عن مجموعة من الأغراض المتشابهة مع بنية متشابهة (واصفات) وسلوك متشابه (طرائق).

الصف هو الواصف لمجموعة من الأغراض لها الصفات نفسها والعمليات نفسها، وهو يلعب بذلك دور قالب لإنشاء الأغراض.

يدعى كل غرض من الصف بـ(نسخة الصف)، حيث تملك كل نسخة مميز هوية فريد، وتستطيع أن تستدعي الطرائق المعرّفة ضمن الصف.

هنالك نوعان لطرائق الصف، طرائق عامة يمكن طلبها من أغراض أخرى، وطرائق خاصة لا يمكن أبداً طلب تنفيذها من أغراض أخرى.

مثال 1

- نستعرض في هذا المثال الصف Point الذي يحتوي على واصفتين X و Y تمثلان إحداثيات النقطة، والطريقة Distance لحساب البعد بين نقطتين، والطريقة Equals لمقارنة نقطتين (متساويتين أم لا).

```
Class Point {
        //variables
        double X;
        double Y;

        //methods
        //computes the distance between two points
        double Distance (Point aPoint);
        //determines if two points have the same coordinates
        bool Equals (Point aPoint);

};
```

- سنقوم باستخدام الصف Point في بناء الصف Rectangle الذي يحتوي على واصفتين من النمط Point (UpperLeftCorner و LowerRightCorner) بالإضافة إلى الطرائق (Area و Length و Height).

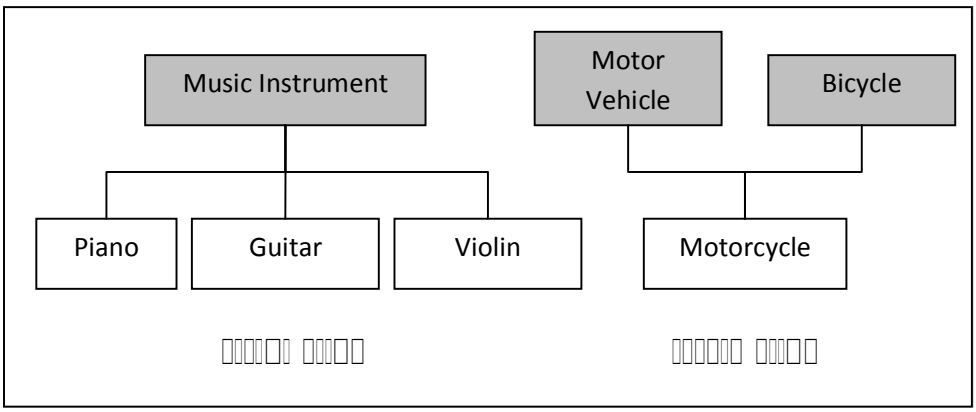
```
Class Rectangle {
        //variables
        Point UpperLeftCorner;
        Point LowerRightCorner;

        //methods
        //computes the area of the rectangle
        double Area ();
        //compute the length
        double Length ();
        //compute the height
        double Height ();
};
```

2-5 علاقات الصفوف

- الصف الأعلى: هو عبارة عن تصنيف أعم للصفوف الجزئية منه.

- الصفوف الجزئية: تحتوي على مركبات مخصصة من التصنيف الأعم للصف الأعلى.

- الوراثة: هي قدرة الغرض في الهرمية على وراثة بنية المعطيات والسلوك للصفوف الأعلى منه في الهرمية، وهي نوعان: وراثة أحادية ووراثة متعددة.



الصف (أداة موسيقية) هو صف أعلى للصفوف (بيانو، غيتار، فيولون)، وبالتالي فإن الصفوف الأخيرة هي صفوف جزئية من صف الأداة الموسيقية.

جميع الصفوف في الهرمية موروثة من الصف الجذر للهرمية.

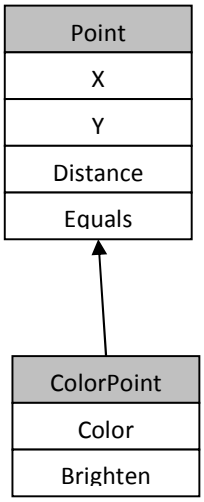
هنالك نوعان من الوراثة:

- وراثة أحادية: وهي موجودة عندما يكون للصف أب واحد فقط (صف أعلى)، وعندما يُرسِل النظام طلب تنفيذ طريقة معينة إلى غرض معين يتم البحث أولاً عن هذه الطريقة في الصف الذي ينتمي إليه الغرض ومن ثم في حال عدم وجودها يتم البحث في الصفوف الأعلى في الهرمية.

- وراثة متعددة: وهي موجودة عندما يكون للصف أكثر من أب واحد.

مثال 2

- نعود في هذا المثال إلى الصف Point وسنقوم بتطبيق مفهوم الوراثة لبناء صف جديد هو ColorPoint حيث يتضمن هذا الصف الجديد نفس واصفات وطرائق الصف Point، ولكن إضافة إليها الواصفة Color والطريقة Brighten.

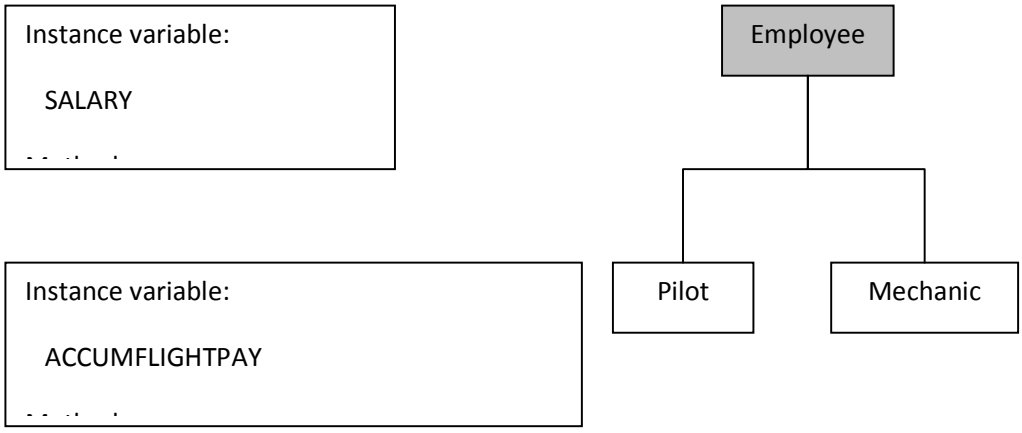


- تعريف الصف ColorPoint:

```
class ColorPoint : Point{
        //variables
        int Color;
        Point LowerRightCorner;

        //methods
        //computes a new color that is brighter
        int Brighten ();
};
```

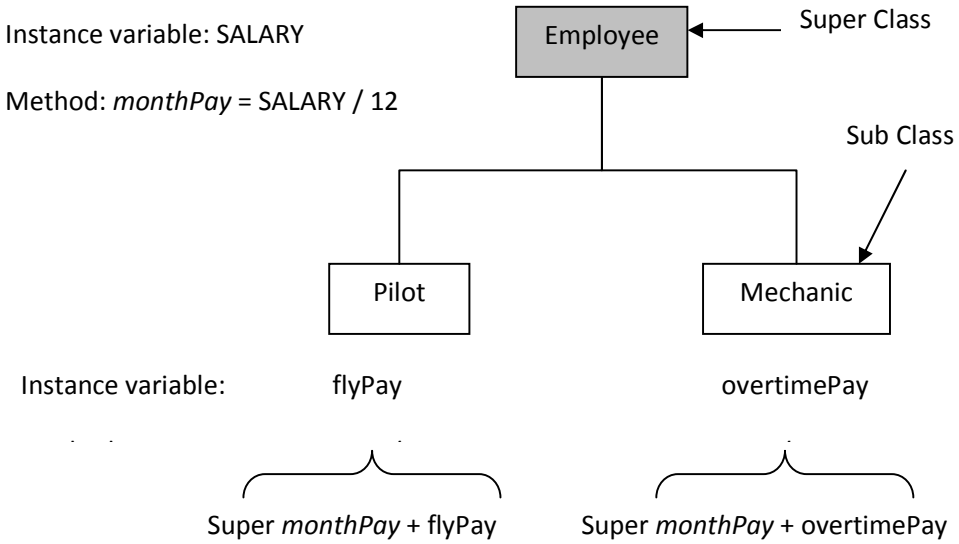
3-5 إعادة تعريف الطرائق

- إعادة تعريف الطريقة: يمكن أن نعيد تعريف طريقة معرفة في الصف الأب، بإعادة تعريفها في الصفوف الجزئية منه.
- مثال: لدينا صف أب (موظف)، وصفوف جزئية منه (طيار وميكانيكي)، نلاحظ أنه أعدنا تعريف الطريقة (*Bonus*) في الصف *طيار* ولم نقم بإعادة تعريفها في الصف *ميكانيكي*، فجميع الموظفين لديهم طريقة واحدة في حساب المكافآت ماعدا الطيار لذلك قمنا بإعادة تعريفها في الصف الخاص به.



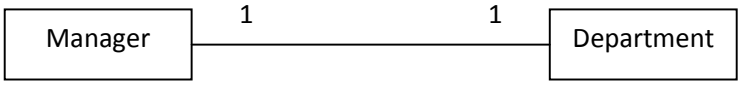
4-5 تعددية الأشكال

- تمكن تعددية الأشكال الغرض من السلوك بحسب معطياته الخاصة.
- مثال: بالعودة إلى نفس المثال السابق فإن حساب الراتب الشهري من خلال طلب نفس الطريقة (*monthPay*) من الصف *ميكانيكي* والصف *طيار* ولكن سيتم حسابها بطريقة مختلفة بكل صف وسيتم إعادة النتيجة الصحيحة.

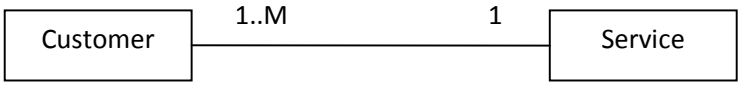


5-5 العلاقات بين الصفوف

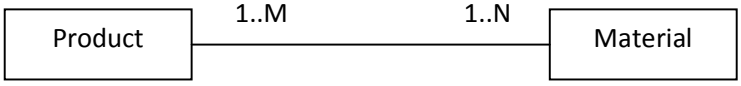
- علاقة واحد لواحد (1:1): علاقة غرض لغرض. مثال كل مدير يرأس قسم واحد وكل قسم يرأسه مدير واحد.



- علاقة واحد لكثير (1:M): كل غرض من الصف الأول يرتبط بـM غرض من الصف الثاني. مثال الموظف والخدمة، يعمل الموظف على خدمة واحدة، بينما تحوي الخدمة أكثر من موظف (علاقة 1:M).

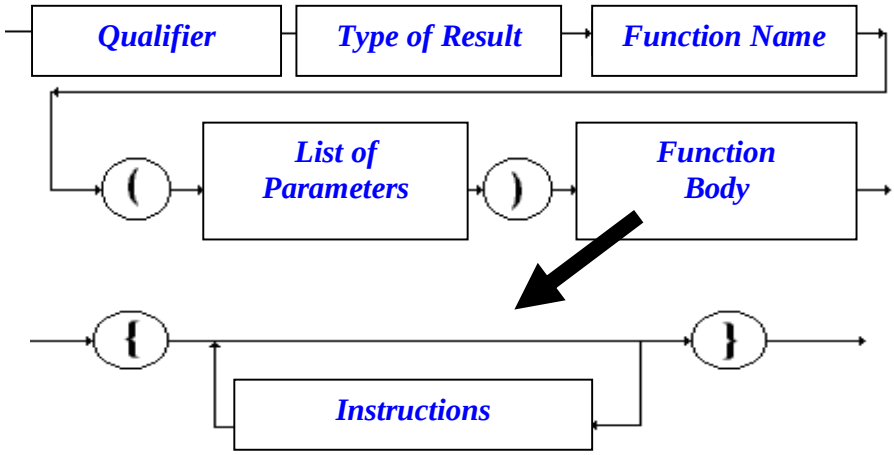


- علاقة كثير لكثير (M:N): يرتبط كل غرض من الصف الأول بـM غرض من الصف الثاني، وكذلك يرتبط كل غرض من الصف الثاني بـN غرض من الصف الأول. مثال كل منتج يحتاج إلى مجموعة مواد أولية، وكل مادة أولية تُستخدَم في إنتاج أكثر من منتج.



6-5 التصريح عن طريقة وتعريفها

يحتاج التصريح عن طريقة إلى تعريف رأس يحتوي 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 تصريح الطريقة وفق الشكل القواعدي التالي:

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

وتكون دلالة هذا التصرح كما يلي:

- حتى الآن سنكتفي بالكلمة المفتاحية **static** ككلمة تعبر عن <Qualifier>، وهي كلمة تشير إلى أن الطريقة هي طريقة تابعة للصف في الصف المُعرَّفة فيه وليس لغرض معرّف بموجب هذا الصف. 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 التي يعرفها المُبرمج. ولايمكن إهمال هذا الجزء من التصريح.
- ويشبه 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:

```
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) {
            //...
        }
}
```

7-5 تمرير المعاملات

تمثل المعاملات المُستخدَمة عند تعريف الإجرائيات، متحولات صماء.

تشبه هذه العملية، عملية تعريف تابع رياضي مثل $f(x)=x+5$

$f(2)=7$

$f(7)=12$

...

تمثل المعاملات المُستخدَمة عند تعريف الإجرائيات، متحولات صماء تساعد في تفسير عمل البرنامج من أجل متحولات حقيقية ستحل مستقبلاً (عند استدعاء الإجرائية وتنفيذها) محل هذه المُعاملات.

تشبه هذه العملية، عملية تعريف تابع رياضي مثل f(x)=x+5 حيث يلعب f هنا دور الإجرائية، وتلعب x دور المُعامل الأصمّ. تظهر المتحولات الحقيقية عند تنفيذ التابع، فعندما تحل القيمة 2 محل x يجري تنفيذ f فنحصل على القيمة 7.

عند الإعلان عن الطرائق ومعاملاتها، وعند استدعائها، يكون أسلوب تمرير قيمة المتحولات صالحاً من أجل كافة المعاملات ذات الأنماط البسيطة في C++، بالإضافة إلى المعاملات التي تكون على شكل أغراض (أنماطاً مركبة).

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

تلقائياً يكون تمرير معاملات الطرائق، تمريراً للقيمة.

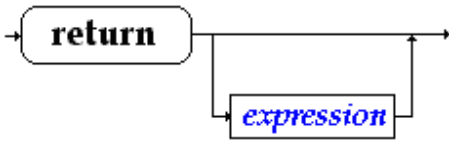
مثال:

```
#include "stdafx.h"
#using <mscorlib.dll>
#include <tchar.h>
using namespace System;


class Computation {


        public:
                static int Sum (int a, int b)
                {
                        int c=a+b;
                        return c;
                }
};

int _tmain(int argc, char *argv[])
{
    int a = 10;
    int b = 20;
```

```
    int c = Computation::Sum(a,b);
    Console::Write("The sum is: ");
    Console::WriteLine(c);

    return 0;
}
```

8-5 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 الإعلان عن الطريقة وتعريفها؛
- عند الوصول إلى return أثناء تنفيذ تعليمات الطريقة، يتوقف تنفيذ بقية التعليمات الموجودة ما بعد return؛

9-5 تحديد مدى تعريف ورؤية 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.

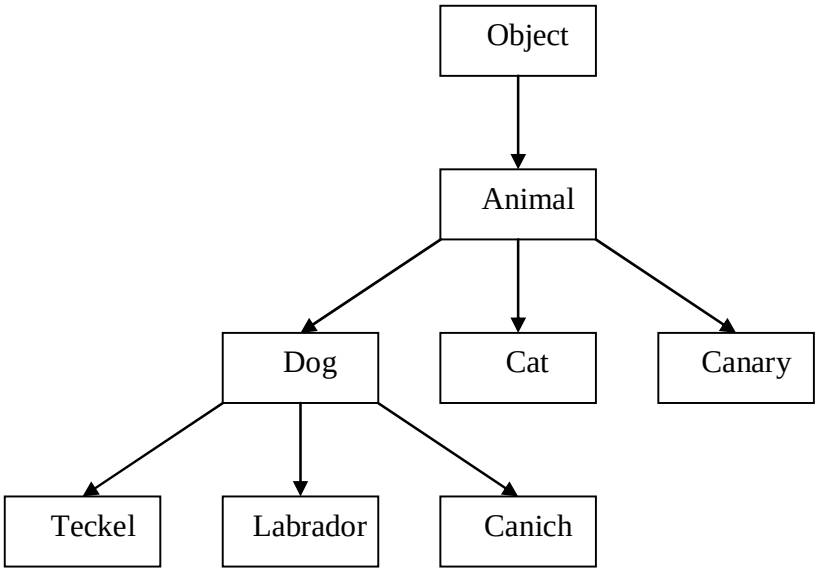
Public	تكون العناصر المسبوقة بكلمة public قابلة للاستخدام من قبل جميع الصفوف والإجرائيات الأخرى.
Private	تكون العناصر المسبوقة بكلمة private قابلة للاستخدام من قبل عناصر الصف الذي تتعرف فيه فقط.
Protected	تكون العناصر المسبوقة بكلمة protected قابلة للاستخدام من قبل عناصر الصف الذي تتعرف فيه ومن قبل الصفوف والعناصر المُشتقة منه والتي ترثه.

10-5 الأغراض

- تتعامل لغة C++ مع كل العناصر المُستَخدَمة كأغراض؛
- الغرض هو مِثْل أونسخة من صف؛
- كل صف مُشتق من صف أب؛
- يكون أي صف مُعرَّف بدون تحديد سلفه، صفاً خلفاً للصف Object.

مثال:

لنأخذ المثال التالي التي يعبر عن هرمية بين الحيوانات:



إن الهرمية الموضحة في المثال هي هرمية (is a) وليست هرمية (has a)، فكل Teckel هو كلب (Dog)، وكل كلب هو حيوان (Animal)، وكل حيوان هو كائن أو غرض (Object)، فنستنتج أن كل Teckel هو حيوان، وأن كل Teckel هو أيضاً غرض.

تتعامل لغة C++ مع كل العناصر المُستَخدَمة كأغراض، حتى المتحولات الأولية يمكن معالجتها كأغراض بعد تغليفها بالصفوف المُناسبة.

والغرض في C++ هو مِثل أونسخة من صف، وكل صف مُشتق من صف في مستو أعلى، ونسميه أحياناً الصف الأب، والصف الوحيد المُستثنى هو الصف Object الذي يُعتبر السلف الأقدم بين الصفوف. ويكون أي صف مُعرَّف بدون تحديد سلفه، صفاً خلفاً للصف Object.

من ناحية أخرى، يكون مثل من صف هو غرض من النمط الموافق، ومن أنماط كل الصفوف الأسلاف.

11-5 البنّاء

- طرائق خاصة تحمل أسماء الصفوف التي تنتمي إليها؛
- يجري تنفيذها تلقائياً عند إنشاء الغرض؛
- يمكن للصف الواحد أن يحتوي أكثر من بنّاء، يكون لكل منها مُعاملات مختلفة.

مثال:

ليكن لدينا الصف التالي:

```
class one {
        int a;
}
```

تقوم C++ أوتوماتيكياً بتوليد بنّاء خاص بهذا الصف هو:

```
public one() { }
```

لذا، يمكن عند تعريف الصف، تعريف بنّاء له:

```
class one {
        int a;
        public one() { }
}
```

بحيث يمكن الاستفادة منه في عملية إعداد عناصر الصف:

```
class one {
        int a;
        public one() {
                a=100;
        }
}
```

كما يمكن تعريف البنّاء مع معاملات:

```
class one {
        int a;
```

```
        public one(int b) {
                        a=b;
        }
}
```

ومن الممكن تعريف أكثر من بنّاء للصف الواحد تختلف باختلاف معاملاتها:

```
class one {
        int a;

        public one() {
        }

        public one(int b) {
                        a=b;
        }

        public one(short b) {
                        a=b;
        }
}
```

12-5 إنشاء الأغراض

يتطلب إنشاء المثل من صف تنفيذ طريقة بنّاء الصف من خلال تعريف متحوّل ذات نمط مركّب، ألا وهو الصف المعني.

مثال:

```
class Computation {
        private:
                //member variables
                int a,b,c;
        public:
                //constructor
                Computation(int x, int y)
                {
                        a = x;
                        b = y;
                }

                // function to compute the sum
                int Sum ()
                {
                        c=a+b;
                        return c;
```

```
            }
};


int _tmain(int argc, char *argv[])
{
        //declare a variable named comp of type Computation and
initialize
        //it by calling the constructor of the class Computation through
passing the
        //integer values 10 and 20.
        Computation comp(10,20);
        Console::Write("The sum is: ");
        Console::WriteLine(comp.Sum());


    return 0;
}
```

هنالك طريقة أخرى لإنشاء المثل من صف وذلك من خلال استخدام الكلمة المحجوزة new على أن يليها اسم طريقة بناء الصف. لن نتطرّق إلى هذه الطريقة ضمن هذا المقرّر.

13-5 تمرين

اكتب برنامج Square يحفظ طول ضلعه ويمتلك طريقة لحساب مساحته. اشتق منه صف Cube الذي يمتلك طريقة تعريف لمساحة المكعب.

الحلّ:

```
#include "stdafx.h"
#using <mscorlib.dll>
#include <tchar.h>
using namespace System;



class Square
{
	protected:
		int length;

	public:
		Square() {}

		Square(int len)
		{
			length=len;
		}
```

```
        int getSurface()
        {
            return length*length;
        }
};

class Cube:Square
{
    public:
        Cube(int len)
        {
            length=len;
        }

        int getSurface()
        {
            return 6*length*length;
        }

};
```

```
int _tmain(int argc, char *argv[])
{
	Cube c(3);
	Square s(3);
	Console::Write("Surface of square s(3): ");
	Console::WriteLine(s.getSurface());
	Console::Write("Surface of cube c(3): ");
	Console::WriteLine(c.getSurface());

	return 0;
}
```