

الفصل الاول

أساليب برمجة الحاسب

مفهوم برمجة الحاسب

مقدمة . البرمجيات و أنواعها . مميزات البرنامج الناجح . المراحل الأساسية في برمجة الحاسب . أساس لغة برمجة . مناقشة عامة .

مقدمة :-

تستخدم كلمة برمجة في كثير من الأحيان مرادفة لكلمة ترميز coding إي كتابة التعبير بلغة حاسوبية معروفة ، و لكن في كثير من الأوساط التعليمية تفهم برمجة الحاسب بأنها سلسلة من التعليمات و التركيبات للغة محددة ... و في الحقيقية أن هذا المفهم خاطئ حيث أنه يوجد الكثير من الأعمال التمهيدية و التحضيرات التي يجب أن تسبق الترميز لإيجاد حل ممكن لمسألة ما و التي تضمن خطوات حل المسألة و الطريقة الصحيحة .

مما تقدم فإن برمجة الحاسب هي عملية صعبة جدا بمعظم مراحلها و التي تكون جميعها مهمة و تؤدي إلي حل المسألة ، كما أنه يجب عدم الخلط بين مفهوم البرمجة و أي مرحلة منفردة من مراحلها ، لأن ذلك يؤدي إلي استبعاد المراحل الأخرى ، و أن البرمجة بتعريفها الشامل هي أن يقوم شخص ما (مبرمج) بتوصيف مهمة معينة للآلة (الحاسب) لتقوم بها آليا و الحصول على النتيجة بتغذية الآلة إي الحاسب بقيم خاصة تسمى المعطيات أو المدخلات و يوضح المخطط التالي مفهوم برمجة الحاسب .

الخلاصة ... أنه عندما تقوم بتصميم برنامجك الخاص فإن الترميز بأي لغة برمجة يجب أن يكون مسبقا بمقدار كبير من الأعمال التمهيدية ، و أن عدم فهم هذا المبدأ يشكل أول و أكبر خطأ يمكن أن يرتكبه المبرمج أثناء تعلمه برمجة الحاسب.

البرمجيات :-

تعرف البرمجيات بأنها الكيان المعنوي للحاسوب و تطلق علي مجموعة البرامج اللازمة للتخاطب و التفاهم و الاتصال بين الجهاز و الشخص المشغل له و هذه

البرامج تقوم بدور كبير في تسهيل التعامل مع هذه الأجهزة حيث يقوم الإنسان بكتابتها و تأخذ حيزا في ذاكرة الحاسب لا يستطيع العمل بدونها .

أنواع البرمجيات :-

تنقسم البرمجيات إلي عدد من البرامج و منها

البرنامج المصدري Source Program

و يعرف بأنه عبارة عن مجموعة من الأوامر و التعليمات المكتوبة بشكل منطقي و متسلسل بإحدى لغات البرمجة الحاسب و هي تقوم بتوجيه الحاسب لأداء عملية معينة من المسألة المعطاة .

البرنامج الهدفي Object Program

هذا النوع من البرامج خالي من الأخطاء و مكتوب بلغة الآلة قابل للتنفيذ على الحاسب و قابل لاستقبال البيانات للقيام بالعملية المطلوبة أو طباعة المعلومات و هذا النوع من البرامج لا يمكن التعديل بسهولة فيه لأنه مكتوب بشفرات معقدة.

برنامج المترجم Compiler Program

مهمته القيام بقراءة البرنامج المصدري المكتوب بإحدى لغات البرمجة الراقية و راجعته و اكتشاف الأخطاء فيه حسب قواعد لغة البرمجة و تصحيحها تم يقوم بترجمة هذا البرنامج إلي لغة الآلة التي يفهمها الحاسب .

برامج النظام System Programs

هي عبارة عن مجموعة من البرامج يقوم بكتابتها شخص أو مجموعة من الأشخاص بقصد مساعدة مستخدم جهاز الحاسب الآلي مثل أنظمة التشغيل OS و التي مهمتها التنظيم و التحكم في تشغيل و إدارة جهاز الحاسب الآلي و تكون مخزنة داخل الحاسب .

برامج تطبيقية Application Programs

مثل هذه البرامج تعتبر أكثر انتشارا و مهمتها حل بعض المسائل التي تتعلق بمؤسسة ما مثل المستشفيات و الشركات و غيرها من القطاعات و النشاطات .

مميزات البرنامج الناجح :-

بما أن العديد من البرامج معقدة نوعا ما فإن عملية تطوير البرامج تخضع لقواعد ضبط و تنظيم حتى تكون النتائج صحيحة و مفيدة و من خنا فإن مميزات البرنامج الناجح هو ذلك البرنامج الذي تتحقق فيه الشروط التالية :

الصحة **Validity** إي يجب أن يقوم البرنامج بما يفترض أن يقوم به . الاستخدام **Usability** و هي أن يكون البرنامج سهل الاستخدام . الموثوقية **Reliability** هي جعل البرنامج يعمل بكفاءة و لا يسبب ضياع بعض أو كل الحقوق إي يجب أن يعمل دون فشل . الوضوح **Understandability** يجب أن يقرأ البرنامج بسهولة . قابلية التعديل **Updatability** سهولة تطوير و توسيع مهام البرنامج . سهولة الصيانة **Maintainability** كلما كانت شفرة البرنامج سهلة كلما كانت عملية الصيانة و التطوير ممكنة . الكفاءة **Efficiency** يجب أن يتصف البرنامج بالكفاءة العالية . أخفاء البيانات **Information Hiding** التصميم الجيد للبرنامج هو ذلك التصميم الذي تقلل من ارتباط المستخدم بالتفاصيل الدقيقة للبرنامج المراحل الأساسية في برمجة الحاسب :-

لحل إي مشكلة باستخدام الحاسب نتبع عددا من الخطوات و حسب درجة كفاءة هذه الخطوات تتحدد كفاءة تشغيل المسألة حاسوبيا و فيما يلي إليكم أهم الخطوات التي تم إتباعها لحل إي مسألة :

فهم و تعريف المسألة . مخطط الحل . اختبار و عرض الخوارزميات . الترميز . اكتشاف الأخطاء و تصحيحها . الاختبار و الصلاحية . التوثيق . صيانة البرنامج .

فهم و تعريف المسألة أو المشكلة : -

في هذه المرحلة يجب علينا أن نعرف بالضبط ما نريد أن نعمل حيث يتم تعريف المشكلة و توضيح مواصفات المسألة بكل دقة ، و لا يجب على المبرمج إهمال هذه المرحلة و الانتقال إلي مراحل أخرى دون مراعاته لهذه المرحلة لأن ذلك يمكن أن يؤدي إلي عدم الثقة و التي بدورها تؤدي إي حل خاطئ في أغلب الأحيان . من هنا فإن فهم المسألة و التعرف على المشكلة يؤدي بالمبرمج إلي تحديد معطيات الحل ، و كذلك التعرف على القوانين و الشروط اللازم استخدامها لحل المسألة بالإضافة إلي المعرفة المسبقة بشكل المخرجات و التي في مجملها تمثل مستلزمات حل إي مسألة .

مخطط الحل : -

أن إي برنامج لن يكون مكونا من عمل واحد و لكن من مجموعة من الأعمال مرتبطة ببعضها فمثلا نظام محاسبة الرواتب الذي يحتوي على عدة مقاطع برمجية تعطي بيانات الدخل و الخرج و برامج حساب و طباعة التقارير و سجلات الأخطاء و

حفظ المعلومات في المشاريع الكبيرة تتطلب عددا من البرامج و عددا من المبرمجين و أنه من المهم جدا تحديد كل برنامج و كيفية ترابطه مع البرامج الأخرى و يكون ذلك برسم مخطط واضح للحل و تحديد كل هذه المتطلبات .

اختبار و عرض الخوارزميات :-

تعرف الخوارزمية بأنها مجموعة من الخطوات و التعليمات اللازم تنفيذها بشكل متسلسل للوصول إلي الحل المطلوب و منها فإن الخوارزميات الجيدة يجب أن تكون ذات قدرة و مهارة خاصة خصوصا إذا كان هنالك أكثر من خوارزمية لتنفيذ المهمة المطلوبة .

مرحلة الترميز Coding

بعد التعريف الواضح للمسألة و تنظيم الحل و إعطاء تفاصيل الخوارزمية خطوة بخطوة تأتي مرحلة كتابة البرنامج و هي عبارة عن عملية التعبير عن الخوارزمية كاملة بإحدى لغات البرمجة المعروفة . و بما أن هنالك العديد من لغات البرمجة فإن أول مهمة في كتابة البرنامج هي اختيار إحدى هذه اللغات فهنالك كثير من اللغات التي طورت لتلبية الاحتياجات المختلفة في التطبيقات المتعددة و أن اختيار لغة البرمجة ممكنو يخضع للاعتبارات التالية :

طبيعة المسألة . لغة البرمجة المتوفرة لديك . الحدود و الإمكانيات التي يوفرها الحاسب إليك .

حيث أن بعض لغات البرمجة تكون لأغراض عامة و الأخرى تخصصية تهتم بحل مسألة معينة .

اكتشاف الأخطاء و تصحيحها :-

يقصد بذلك عملية اكتشاف الخطأ المنطقي و تحديد موقعه و من ثم تصحيحه و ذلك بتنفيذ البرنامج المترجم لفحص نتائج البرنامج المكتوب بلغة برمجة عالية المستوى و ذلك من خلال تتبع مراحل هذا البرنامج بشكل كامل ، حيث يجب على المبرمج أن يخضع البرنامج إلي اختبار يشابه الحقيقة من خلال ما يدخله من معطيات و مراجعة النتائج بعد المعالجة باستخدام الحاسب .

الاختبار و الصلاحية :-

لا يكفي أن يحصل المبرمج من برنامجه علة نتائج بل يجب أن نضمن أنه يعطي البرنامج نتائج صحيحة .

التوثيق :-

أن عملية التوثيق هي عملية مهمة و استمرارية و هي عبارة عن وصف تفصيلي لخوارزمية البرنامج و تصحيحه و طريقة كتابه و فحصه و كيفية استخدامه بالشكل الصحيح .

صيانة البرنامج :-

تأتي ضرورة صيانة البرنامج من حقيقة أن البرنامج ليس كمية ثابتة بل قابل للتحديث و التعديل كلما اكتشف أخطاء أو جدت مسائل جديدة تتطلب حلا أو حصلت على تجهيزات جديدة .

أسس لغة البرمجة : -

كما هو الحال في إي لغة مثل اللغة العربية أو الانجليزية هنالك مجموعة من العناصر الأساسية نحتاجها وقت التعامل مع هذه اللغة ... و من هنا فأن لغة البرمجة أيضا تحتاج لمثل هذه العناصر و التي تتمثل في الأرقام و الحروف و الرموز و غيرها من الأمور .

البيانات و أنواعها : -

أن البيانات في إي لغة هي مجموعة من القيم العددية أو الحرفية أو الرموز غير مبوبة و تختلف عن المعلومات التي تعد حقائق مبوبة و منها فأن البيانات هي حقائق غير مصنفة إي تشبه المادة الخام و قد يحتاجها البرنامج أثناء التنفيذ بغية معالجتها ، حيث يتم إدخالها إلي الحاسب عن طريق البرنامج و يوجد فيها عدة أنواع.

الأعداد الرقمية Digits Number

و هي قيم ثابتة لا تتغير قيمتها و تتكون من مجموعة من الأرقام ذات حد أدنى و حد أقصى و لها الأشكال التالية

أرقام صحيحة Integer Number... و هي أرقام موجبة و سالبة بشرط أن تنطبق عليها الشروط التالية : لا تحتوي على فاصلة عشرية أو قيمة آسية . الرقم السالب يجب أن يسبقه علامة (-) . لا يكون فيه إي رمز خاص أو إي حرف هجائي . أرقام حقيقة Real Number.... و هي تلك الأرقام التي لها كسور عشرية و لا يكون بها إي رمز خاص أو حروف هجائية . الأرقام الآسية Exponential Number...

يمكن تمثيل الرقم لحقيق على شكل أسي في حالة ما إذا كانت الأرقام كبيرة جدا أو صغيرة جدا و ذلك باستخدام الحرف (E).
مثال على ذلك

E4 Or 1234E-2٠,٠٠٠١٢٣٥

بيانات السلسلة String Data

هي بيانات وصفية تتكون من مجموعة من الرموز المتجاورة سواء أن كانت تلك الرموز حروف أو أرقام أو حروف و أرقام أو قد تكون رموز بشرط ألا تكون من الرموز أو الكلمات الخاصة أو المحجوزة في لغة البرمجة المستخدمة في كتابة البرنامج كما يشترط فيها إن تكتب بين علامتي تنصيص مفردة (') ... مثال على ذلك

'ED١٢٥٥' 'Employee' 'Remas' '12354'

المتغيرات Variables

المتغير هو عبارة عن عنوان في ذاكرة الحاسب يمكن التغيير في قيمته حسب الحاجة و يشترط في المتغير الآتي :-

أن يكون المتغير من حروف أو حروف و أرقام . يجب أن يبدأ المتغير بحرف من جهة اليسار و يشترط فيه أن يكون باللغة الانجليزية . يمكن أن يكون مجموعة من الحروف و الأرقام أو الرموز على أن يكون ليس من الكلمات المحجوزة أو الرموز الخاصة . يجب ألا يحتوي على فراغ و يمكن أن يحتوي المتغير على علامة (underscore) التي الشرطة التي على السطر _)

أنواع المتغيرات :-

المتغير الصحيح Integer المتغير الحقيقي أو الكسري
Real المتغير السلسلة String المتغير الحرفي Character المتغير
Boolean المنطقي

٤,١,٤ الكلمات المحجوزة Reserved Words

هناك عدد من الكلمات المحجوزة في كل لغة برمجة و المبرمج مراعاتها عند كتابة برنامجها و استخدامها بالطريق لأمثل لكي يحصل على برنامج صحيح ... لأنها قد تتسبب في أرباك المترجم الخاص بلغة البرمجة عن تطبيق البرنامج و ترجمته إلي لغة الآلة .

العمليات الحسابية و المنطقية

عندما تتخلل إي عبارة حسابية العديد من الإشارات الحسابية فأن عملية تنفيذها تتم حسب ترتيبها من اليسار إلي اليمين و ذلك وفقا للترتيب التالي :

١. الأس . ٢. الضرب و القسمة. ٣. الجمع و الطرح .

*** العلامات الحسابية المنطقية المستخدمة في الحاسب :-

العملية الحسابية أو المنطقية الإشارة عملية الأس ^ الضرب * القسمة / الجمع +
الطرح - يساوي = أصغر من < أكبر من > أكبر من أو يساوي => أصغر من أو
يساوي <= لا يساوي <>

مثال (١) لديك العملية الحسابية التالية المطلوب إيجاد قيمة A

حيث ...

الحل

$$A=5*2^3$$

$$A = 5*8$$

$$A= 40$$

من النتائج نلاحظ أن تم إجراء عملية تحرير الأس أولاً ثم عملية الضرب ليكون قيمة
A=40.

مثال (٢) أو جد قيمة Y حيث أن

$$Y = (4+3)*(5-2)$$

الحل

$$Y = 7*(5-2)$$

$$Y = 7*3$$

$$Y = 21$$

نلاحظ أن قم تم في هذه العملية تحرير الأقواس أولاً ثم إجراء عملية الضرب لتكون قيمة $Y=21$... من فإن عملية الحسابية التي بها أقواس تجرى أولاً وفي حالة وجود أقواس وأس فإن الأولوية تكون للأس ثم الأقواس ثم تجرى باقي العمليات بالترتيب من اليسار إلى اليمين مع مراعاة مذكر أعلاه من حيث أسبقية العمليات الحسابية .

٥. أمثلة عامة

أمثلة عن الكلمات المحجوزة و المتغيرات بأنواعها و كذلك بعض أنواع المتغيرات السليمة و التي تتوفر فيها شروط المتغير .

×××× الكلمات المحجوزة :

**FOR ,READ , ADD , OR, PRINT,DO , WHILE
. ,NEXT,WRITE,AND,IF , THEN , LOOP**

××× المتغير الصحيح و الذي لا يقبل قيمة حرفية أو قيمة كسرية و يشغل حيزاً أقل في الذاكر مثال على ذلك

٥٢٤

٦٥٧

٥٢

١

××× المتغير الكسري أو الحقيقي هو ذلك المتغير الذي يقبل قيمة صحيحة أو كسرية من الأرقام و لا يقبل قيمة حرفية مثال على ذلك

٥٢٤,٠٠

٦٥٨,٥١

٥٢,٠٤

١,٠٠

××× المتغير الحرفي أو السلسلة و ذلك الحيز من الذاكرة الذي قبل قيمة
حرفية أو رقمية أو مجموعة من الحروف و الأرقام بشرط أن تكتب بين علامتي
تنصيص فردية (')

مثال على ذلك 'AHMED' أو 'ALI125'

××× المتغير المنطقي و ذلك الحيز من الذاكرة الذي له قيمتان فقط هما نعم /
لا إي TRUE / FALSE ××× أوجد قيمة العمليات الحسابية التالية :

١ . $Y = 4 + 3 * 5 - 2$

٢ . $Y = (6 + 3) / 4 - 2$

٣ . $Y = (15 - 3) / (7 - 5)$

٤ . $Y = (15 - 3)^2 * (4 + 3)$

٥ . $Y = 40 / 2 / 2^2$

٦ . $Y = 2^2 / (4 - 2) * 3$

٧ . $6 >= 4 / 20$

٨ . $5 - 8 > 2 / 8 + 4$

××× بين أي المتغيرات التالية صحيح و أيها خطأ من حيث شروط المتغير

اسم المتغير الجواب السبب S40
Z
سأل ALI_225 %HH DDRESS1 FOR 55J5K K55
م DO # 898AD_

أساليب حل المشكلات باستخدام الحاسب
تظهر الضرورة لعمل برنامج الحاسب الآلي عند ظهور مشكلة لها صفة التكرار مثل
(حساب رواتب الموظفين ، إدارة المخازن ، استخراج نتائج الطلاب ،
..... إلخ) مما يساعد على توفير الوقت والجهد من الأعمال الروتينية التكرارية ،
أو عند الرغبة في حساب بعض العمليات الرياضية المعقدة والتي تتطلب دقة
وسرعة للحصول على نتائجها مثل (حساب المساحات أو الأحجام لبعض الأشكال
الهندسية وتحديد مسارات سفن الفضاء والصواريخ إلخ).
عند استخدام الحاسب الإلكتروني في حل مسألة ما (مشكلة) ، فإن هناك عدداً
من الخطوات التي ينبغي إتباعها، وفقاً لدرجة كفاءة تنفيذ هذه الخطوات، تتحدد

كفاءة تشغيل المسألة على الحاسب. والجدير بالذكر أن أهم هذه الخطوات يتم تنفيذها وإنجازها خارج الحاسب وبدون استخدامه إذ أنها تمثل منطق حل المسألة، وفيما يأتي عرض لهذه الخطوات حسب ترتيبها المنطقي:

Problem Definition & Analysis

تعريف المسألة وتحليلها

Algorithm

وضع خوارزمية الحل

Writing the Program

كتابة البرنامج بإحدى لغات الحاسب

Compilation

ترجمة البرنامج إلى لغة الآلة

Execution

تنفيذ البرنامج

البرنامج (Program):

هو عبارة عن مجموعة من الأوامر (Instructions) المرتبة منطقياً التي تخبر الكمبيوتر بما يجب أن يقوم به للحصول على النتائج المطلوبة.

وتتمثل وظيفة المبرمج (Programmer) الأساسية في كتابة البرامج التي توجه عمل الكمبيوتر. ويجب أن تكون هذه البرامج صحيحة وواضحة وقادرة على إنتاج معلومات ذات جدوى تخدم المستخدم النهائي.

وتنقسم المراحل التي يمر بها برنامج الحاسب الآلي إلى مرحلتين رئيسيتين هما دور الإنسان في حل المسألة، ثم يليها دور الحاسب في الحل.

المراحل المختلفة لبرنامج الحاسب الآلي:

أولاً: دور الإنسان في حل المسألة:

ونعني بدور الإنسان المراحل اليدوية لحل المشكلة (المسألة) التي تتم خارج جهاز الحاسب الآلي وينحصر هذا الدور في عدة خطوات يمكن حصرها في النقاط التالية:

1. تحديد معالم المسألة
 2. تحليل عناصر المسألة وذلك بمعرفة معطياتها (المدخلات) والهدف الأساسي لها، والنتائج المطلوبة (المخرجات) و الطريقة المطلوبة لعرضها.
 3. البحث والتفكير في طريقة حل المسألة
 4. تدوين الحل في خطوات متسلسلة متعاقبة، يعبر عنها باللغة العادية، محكمة بالمنطق الرياضي، هذه الخطوات في مجموعها تسمى بالخوارزم Algorithm، كما يمكن تمثيل هذه الخطوات والارتباط فيما بينها بما يعرف بخريطة التدفق أو خريطة سير العمليات أو خريطة المسار flowchart ، وذلك لكي تساعد في تسلسل المنطق العام لحل المسألة.
 5. كتابة البرنامج الذي يمثل المسألة وفي هذه الخطوة يتم ترجمة خطوات الخوارزم وخريطة المسار إلى مجموعة من التعليمات والأوامر وإدخالها إلى الحاسب الآلي باستخدام إحدى لغات البرمجة ثم يحفظ البرنامج داخل الحاسب الآلي ويسمى في هذه الحالة بالبرنامج المصدري Source Program.
- ثانياً: دور الحاسب الآلي في حل المسألة:

ونعني هنا المراحل التي يمر بها البرنامج المصدري حتى استخراج النتائج وتنحصر في الخطوات التالية:

١. ترجمة البرنامج المصدري إلى لغة الآلة ويتم ذلك باستخدام مترجم اللغة (مع ملاحظة أن كل لغة برمجة لها مترجم خاص بها الذي يكتشف الأخطاء الإملائية أو المطبعية Syntax Errors في البرنامج المصدري (إن وجدت) ولا بد من تصحيحها من قبل المبرمج ثم تحويل البرنامج المصدري بعد ذلك إلى لغة الآلة Machine Language ويسمى البرنامج في هذه الحالة بالبرنامج الهدفي

. Object Program

٢. تشغيل البرنامج الهدفي المكتوب بلغة الآلة، ويتم استقبال البيانات واستخراج المعلومات والنتائج.

١- تعريف المسألة وتحليلها:

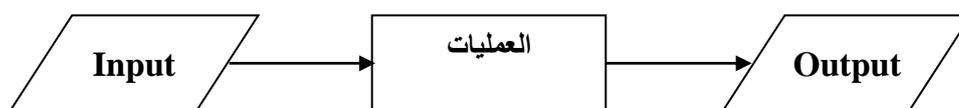
في هذه المرحلة يتم تحديد أبعاد المسألة، وتحديد الهدف المطلوب الوصول إليه وذلك بتحليل مفردات المسألة ووصفها ومن ثم إنجاز المهام التالية:

- تعريف المخرجات، وشكلها بدقة ... ونقصد بالمخرجات هنا، النتائج المراد تحقيقها من حل المسألة، وينبغي هنا أن نوضح أن المخرجات يجب أن يتم تعريفها أولاً لكي يمكن تحديد المدخلات والعمليات اللازمة لتحقيقها؛ فالنتائج تعرف وتحدد أولاً ونحصل عليها أخيراً.

- بناء على المخرجات والنتائج المطلوبة يمكن تحديد المدخلات اللازمة للحصول على هذه المخرجات، وذلك من بيانات ومتغيرات وكذلك تحديد شكلها ومواصفاتها بدقة.

- حصر طرق الحل المختلفة من وجهة نظر الحاسب، وتقييمها لاختيار أفضلها. ذلك أن أي مسألة قد يكون لها أكثر من طريقة للحل، ومن ثم يجب دراسة هذه الطرق واختيار أكثرها ملائمة للتنفيذ باستخدام الحاسب من حيث سهولتها، وسرعة تنفيذها، والمساحة التي تحتاجها من ذاكرة الحاسب.

● والشكل الآتي يوضح العلاقة بين النقاط السابقة واللازمة لتعريف المسألة:



٢- وضع خوارزمية الحل :

بعد اختيار الطريقة الأمثل لتناول المسألة، فإن الخطوة التالية هي التعبير عن هذه الطريقة في شكل خطوات متسلسلة متعاقبة ومتراصة منطقياً تؤدي إلى الوصول إلى حل للمسألة ...

وتسمى هذه الخطوات بالخوارزمية Algorithm - وذلك نسبة إلى العالم المسلم المشهور الخوارزمي (أبو جعفر محمد بن موسى الخوارزمي) - ويمكن تعريف الخوارزمية كالاتي:

" مجموعة متسلسلة من الخطوات اللازمة تحدد الأسلوب المستخدم لحل مسألة معينة"

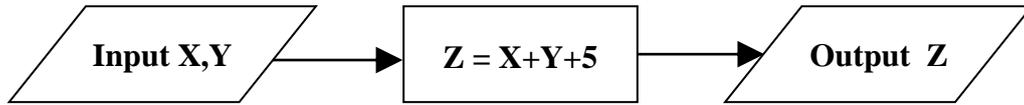
مثال:

اكتب خوارزمية الحل اللازمة للمعادلة الآتية:

$$Z = X+Y+5$$

الحل:

- ١- يجب أولاً أن نحدد الهدف من المشكلة (المسألة) وفي المثال فإن الهدف هو إيجاد مجموع قيم X ، Y مضافاً إليها ٥ . واعتبار قيمة الجمع هي قيمة المتغير Z
- ٢- الخوارزمية:
 - بداية
 - ادخل قيمة X ، Y .
 - اجمع هذه القيم مضافاً لها العدد ٥
 - اطبع حاصل الجمع المسمى (Z) .
 - نهاية



☒ الثوابت والمتغيرات:

نلاحظ من المثال السابق أن قيم Z تعتمد على قيم X , Y فمثلاً لو افترضنا أن

$$X = 1 \quad \& \quad Y = 2$$

فإن قيم Z ستكون:

$$Z = X+Y+5$$

$$Z = 1+2+5$$

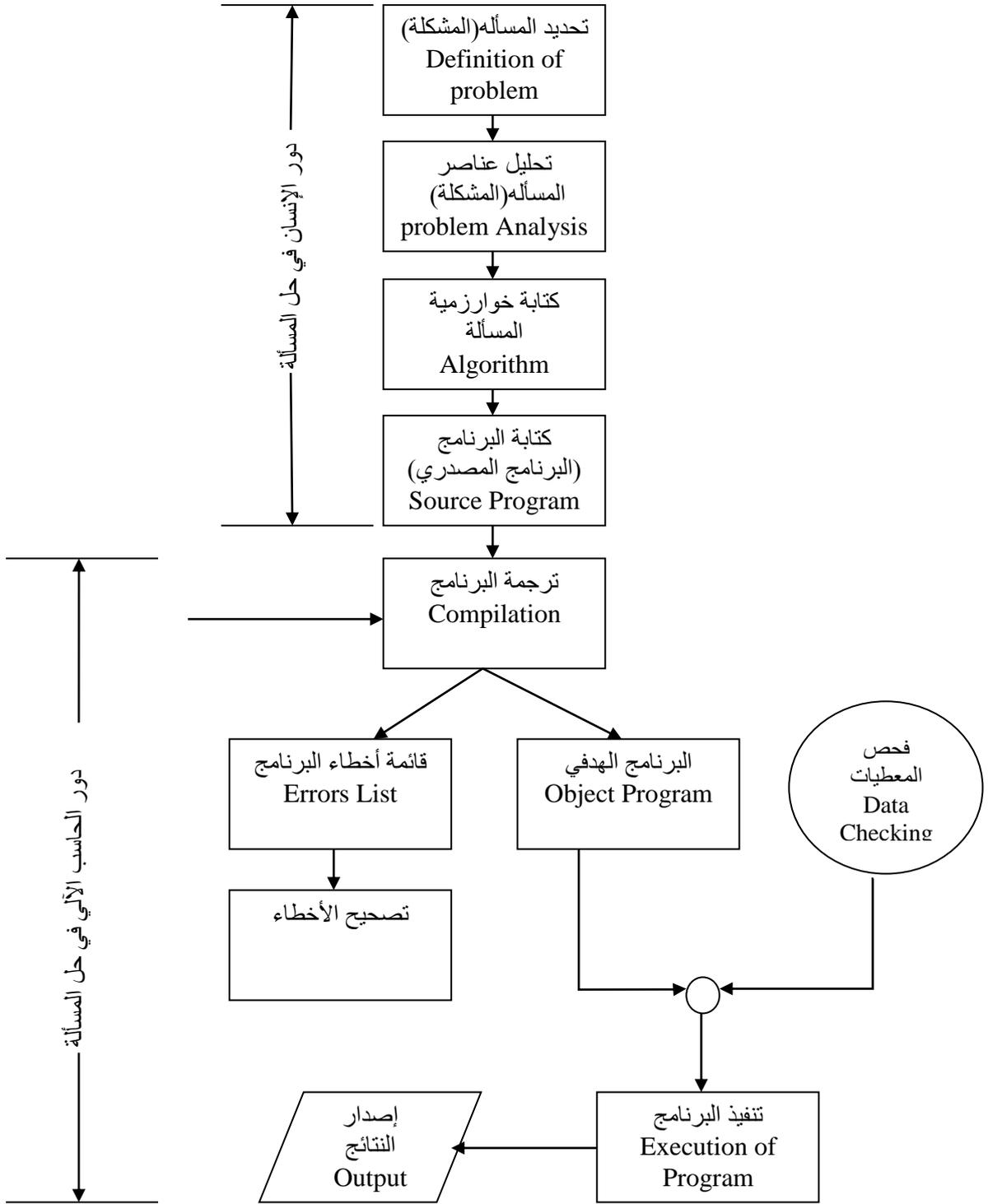
$$Z = 8$$

وإذا تم تغيير قيم X ، Y فإن قيمة Z ستتغير وفي هذه الحالة نطلق على كل من X ، Y اسم متغيرات وتأخذ قيم رقمية إي أن نوعها رقمي. بينما نطلق على (٥) في المعادلة باسم ثابت المعادلة أي أن القيمة لن تتغير. والثابت قد تكون قيمة عددية أو غير عددية وكلاهما لا تتغير قيمته.

أنواع الأخطاء المحتملة في البرنامج المصدري:

١. أخطاء مطبعية Syntax Errors: يحدث نتيجة خطأ في الهجاء عند كتابة تعليمات أو أوامر البرنامج باحدى لغات البرمجة ويتم اكتشافه عن طريق المترجم الخاص للغة البرمجة التي كتب بها البرنامج عند تحويل البرنامج المصدري إلى برنامج هدي.
٢. أخطاء منطقية Logical Errors: يحدث نتيجة خطأ في التسلسل المنطقي للبرنامج أو في القواعد الحسابية للعمليات وتسلسلها ولا يكتشف إلا بعد تشغيل البرنامج واستخراج نتائج خاطئة ويتم اصلاحه باتباع خطوات البرنامج لمعرفة مصدر الخطأ وتصحيحه وتسمى هذه العملية Tracing.
٣. أخطاء اثناء التشغيل Run-Time Errors: تظهر عند تنفيذ البرنامج مثل عدم حجز مساحة كافية للمدخلات او الدخول في دوران بلا نهاية، وتظهر رسالة بنوع الخطاء.

الشكل التالي يوضح المراحل المختلفة التي يمر بها البرنامج:



المراحل المختلفة التي يمر بها البرنامج

الخوارزميات Algorithm وخرائط المسار Flowchart:

١ - الخوارزميات (Algorithms):

كلمة خوارزم تعني الوصف الدقيق لتنفيذ مهمة من المهمات أو حل مسألة من المسائل . واشتقت هذه الكلمة من اسم عالم الرياضيات العربي محمد بن موسى الخوارزمي . وتستخدم كلمة خوارزم على نطاق واسع في علوم الرياضيات والحاسب الآلي . الخوارزم Algorithm : هي مجموعة من التعليمات (الخطوات) المرتبة ترتيباً منطقياً بشكل تتابعي متسلسل ومنظم لتنفيذ عمليات حسابية أو منطقية أو غيرها .
مثال: اكتب الخوارزم اللازم لحساب المتوسط الحسابي لثلاثة أرقام مدخلة بواسطة المستخدم

T_1, T_2, T_3 ؟

Write an Algorithm to calculate an Average of three numbers T_1, T_2, T_3 input by the user?

الحل:

- 1- Start ١ - ابدأ
- 2- Read T_1, T_2, T_3 ٢ - اقرأ قيم الأعداد T_1, T_2, T_3
- 3- calculate ٣ - احسب المتوسط الحسابي للأعداد AV
AV=($T_1+T_2+T_3$)/3 من المعادلة
AV=($T_1+T_2+T_3$)/3
- 4- Print AV ٤ - اطبع النتيجة
- 5- Stop ٥ - توقف

ملاحظات:

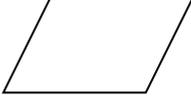
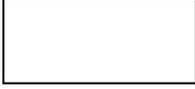
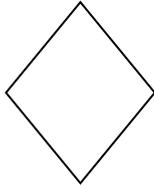
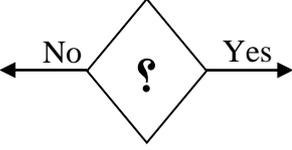
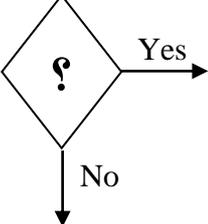
❖ عند كتابة الخوارزم يجب أن تبدأ الخطوة الأولى بكلمة " ابدأ Start " على أن تنتهي آخر خطوات الخوارزم بكلمة " توقف Stop " ونضع بينهما الخطوات الرئيسية لحل المشكلة.

❖ الرموز AV, T_1, T_2, T_3 جميعها رموز اختيارية تسمى بالمتغيرات Variables وهي عبارة عن مساحات يتم حجزها عن طريق المترجم ونظام التشغيل على الذاكرة لاستقبال البيانات المختلفة.

2.1- إعداد خريطة سير العمليات Flowchart

يمكن التعبير عن الخوارزمية السابقة بالرسم بدلاً من الكتابة وذلك باستخدام رموز وأشكال اصطلاحية - عملاً بالحكمة الصينية القائلة "الصورة تغني عن ألف كلمة" - ويطلق على الرسم التوضيحي الناتج باسم "خريطة سير العمليات".

ويمكن تعريف خريطة سير العمليات على أنها رسم توضيحي يبين التسلسل المنطقي لسير العمليات اللازمة لحل مسألة محددة وذلك باستخدام رموز وأشكال هندسية متفق عليها حيث تصف هذه الرموز والأشكال العمليات المحددة. ومن أهم هذه الرموز ما يلي:

الشكل	توضيح الشكل
<p>-١</p> <p>START / END</p> 	<p>رمز طرف المخطط (بداية أو نهاية) ويستعمل ليدل على بداية ونهاية مخطط سير العمليات.</p>
<p>INPUT / OUTPUT-٢</p> 	<p>رمز إدخال وإخراج يستعمل لإدخال البيانات أو لاستخراج النتائج.</p>
<p>PROCESSING-٣</p> 	<p>رمز المعالجة يستعمل للعمليات الحسابية ويكون في داخله معادله مثل :</p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin: 5px;">Z = X+Y</div> <div style="border: 1px solid black; padding: 5px; margin: 5px;">SUM=0</div> </div>
<p>DECISION-٤</p> 	<p>رمز اتخاذ القرار يستعمل في حالات فحص قيمة معينة لاتخاذ قرار معين بالاعتماد على القيمة المفحوصة. ويكون مخرجاتها إما (YES , NO) وتكون كالاتي :</p> <p style="text-align: center;">أو</p> <div style="display: flex; justify-content: space-around; align-items: center;">   </div>
<p>Connector-٥</p> 	<p>رمز التوصيل (الربط)</p>
<p>Flow Lines -٦</p> 	<p>خطوط التوصيل واتجاه السير</p>

الشكل	توضيح الشكل

← تستخدم مخططات سير العمليات لتحقيق الفوائد التالية:

- تسهيل عملية دراسة البرنامج من قبل المستخدمين
- تسهيل عملية مراجعة البرنامج لتعديله أو لاكتشاف الأخطاء الموجودة فيه.
- يستخدم كوسيلة لتوثيق البرامج حيث يعكس المخطط كافة العمليات من إدخال، إخراج ومعالجة الممثلة في البرنامج.

← هناك جملة خطوات يجب أن نراعيها في إعداد خريطة سير العمليات:

- تحديد المدخلات والمخرجات مع تخصيص أسماء المتغيرات المناظرة.
- تحديد خطة تحويل المدخلات إلى مخرجات أو منطق حل المشكلة.
- رسم خريطة سير العمليات.
- اختبار صحة خريطة سير العمليات بادخال (تجريب) ببيانات من عندك.

2.1.1-أنواع خرائط سير العمليات:

يمكن تصنيف خرائط سير العمليات إلى أربعة أنواع رئيسية وهي:

Simple Sequential Flowcharts	خرائط التتابع البسيط
Branched Flowcharts	خرائط ذات الفروع
Simple Loop Flowcharts	مخططات سير العمليات ذات التكرار البسيط
Multi Loop Flowcharts	خرائط الدورانات المتعددة (المتداخلة)

ويمكن للبرنامج الواحد أن يشمل أكثر من نوع واحد من هذه الأنواع وسنتناول فيما يلي شرح النوعان الأول والثاني فقط.

أولاً) خرائط التتابع البسيط:

ويتم ترتيب خطوات الحل لهذا النوع من الخرائط بشكل سلسلة مستقيمة من بداية البرنامج حتى نهايته، بحيث تنعدم فيها أية تفرعات على الطريق.

مثال :

❖ ارسم مخطط سير العمليات اللازمة لإيجاد متوسط درجات ثلاث مواد دراسية؟

الخوارزمية:

البداية

اقرأ الدرجات الثلاثة $M1, M2, M3$

احسب متوسط الدرجات الثلاثة من المعادلة : $AV = (M1 + M2 + M3) / 3$

اطبع نتيجة المتوسط AV

النهاية

خريطة سير العمليات : كما هو موضح في الشكل التالي :-



ثانياً) خرائط ذات الفروع :

إي تفرع يحدث في البرنامج إنما يكون بسبب الحاجة لاتخاذ قرار أو المفاضلة بين اختيارين أو أكثر، فيسير كل اختيار في تفرع مستقل عن الآخر.

مثال :

في المثال السابق إذا كان متوسط درجات الطالب أكبر من ٩٠ درجة يطبع رسالة أن مستواه ممتاز ؟

• الخوارزمية :

١- البداية

٢- اقرأ الدرجات الثلاثة $M1, M2, M3$

٣- احسب متوسط الدرجات الثلاثة من المعادلة : $AV = (M1 + M2 + M3) / 3$

٤- هل المتوسط أكبر من ٩٠

إن كان نعم فأذهب إلى الخطوة ٥

وإن كان لا فإذهب إلى الخطوة ٦

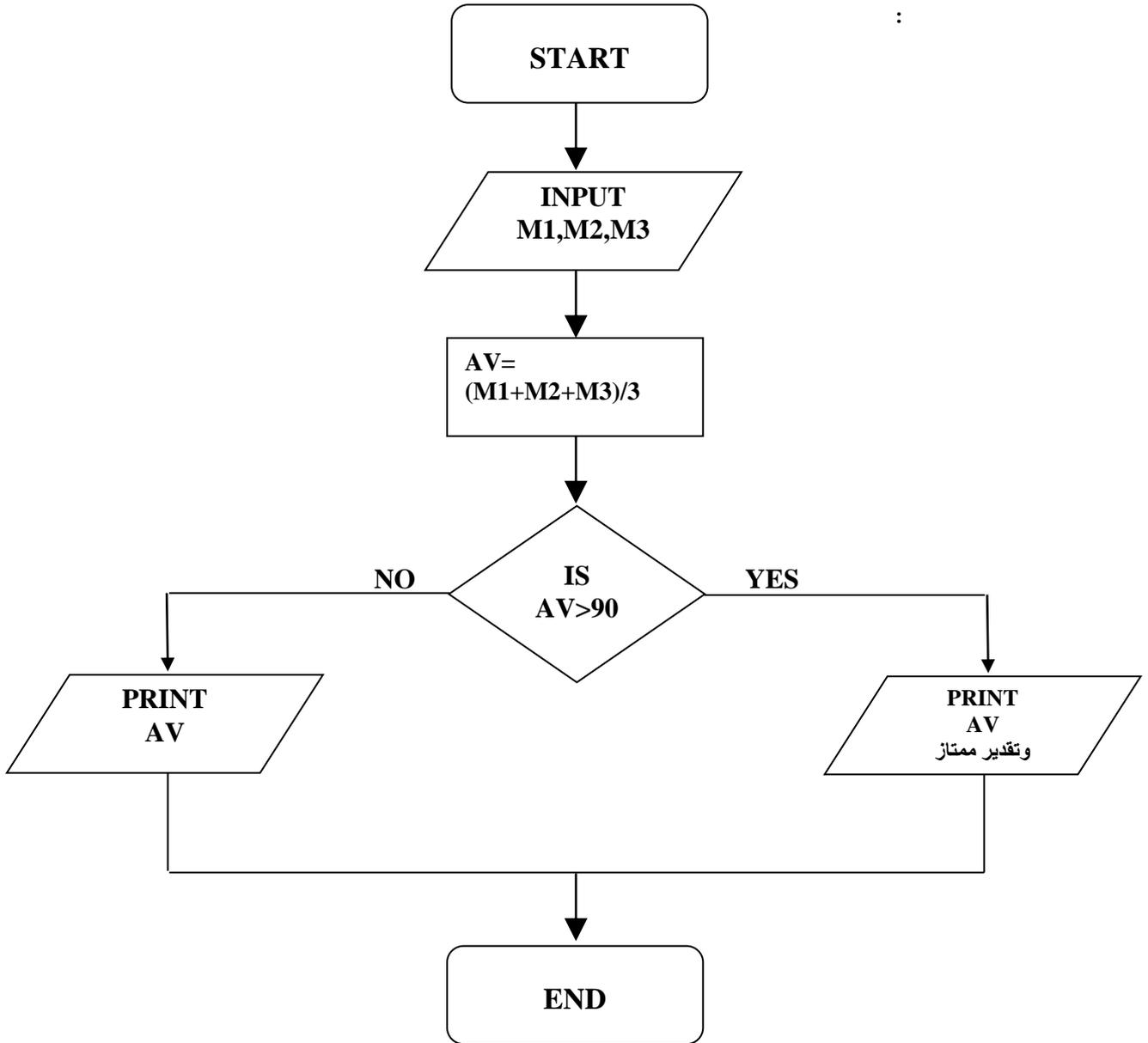
٥- اطبع نتيجة المتوسط AV والتقدير "ممتاز"

٦- اطبع نتيجة المتوسط AV

٧- النهاية

• خريطة سير العمليات

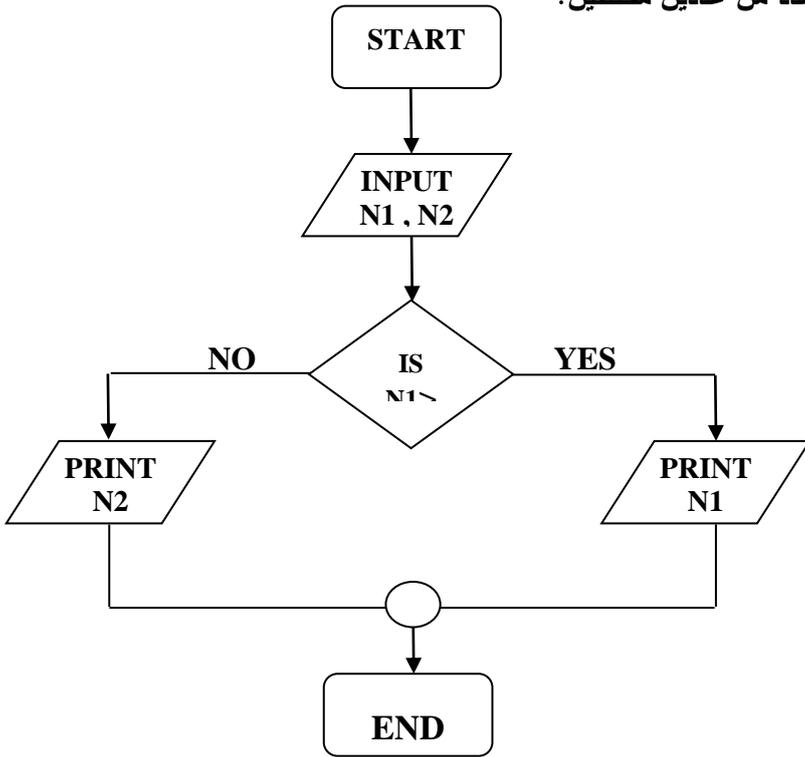
:



مثال ٢ :

ارسم خريطة سير العمليات لإيجاد أكبر عدد من عددين مختلفين؟

مخطط جدول المتغيرات :



الخوارزمية:

١- البداية

٢- اقرأ قيمة العددين N1, N2

٣- هل N1 أكبر من N2

إذا كانت نعم فإذهب إلى الخطوة ٤

وإذا كانت لا فإذهب إلى الخطوة ٥

٤- اطبع العدد N1

٥- اطبع العدد N2

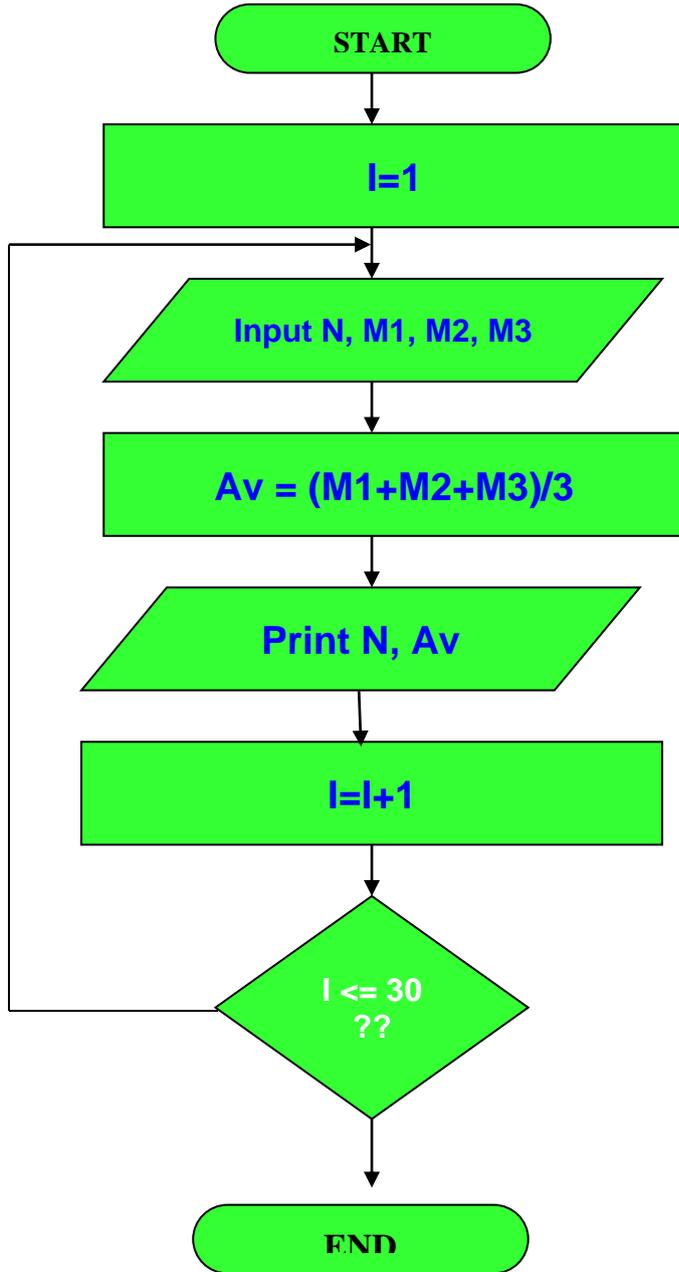
٦- النهاية

(خريطة سير العمليات)

ثانياً) مخططات سير العمليات ذات التكرار:

يستخدم هذا النوع من مخططات سير العمليات في الحالات التي تحتاج فيها إلى تكرار بعض العمليات عدداً من المرات. يتم الخروج من التكرار في هذا النوع من المخططات باستخدام عداد يبدأ بقيمة، ويضاف إليه واحد في نهاية كل دورة، حتى تصل قيمة العداد عدد التكرارات المطلوبة.

مثال يوضح مخطط سير عمليات لقراءة معلومات عن ٣٠ طالب مكونة من رقم الطالب N وثلاث علامات في القيم M1, M2, M3 ثم طباعة أرقام الطلاب ومعدلاتهم في الثلاث قيم.



هي عبارة عن مجموعة من الرموز والأشكال الاصطلاحية تستعمل لتوضيح الخطوات الرئيسية لحل مسألة ما بشكل رسومي متفق عليه (ومعلوم) لجميع مبرمجي الحاسب لتبين جميع الخطوات والعلاقات المنطقية للبرنامج.

فوائد استخدام خرائط سير العمليات أو خرائط المسار Flowchart:

١. تعطي صورة متكاملة للخطوات المطلوبة لحل المسألة في ذهن المبرمج بحيث تمكنه من الإحاطة الكاملة بكل أجزاء المسألة من بدايتها وحتى نهايتها.

٢. تساعد المبرمج على تشخيص الأخطاء المنطقية التي تقع

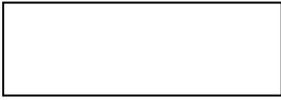
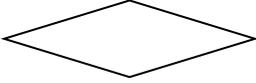
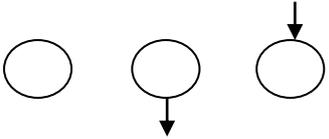
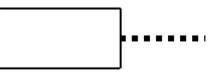
عادة في البرنامج والتي يعتمد اكتشافها على وضوح التسلسل المنطقي لحل المسألة عن طريق المبرمج.

٣. تسير للمبرمج إدخال أي تعديلات في أي جزء من أجزاء المسألة بسرعة وبدون الحاجة لإعادة دراسة المسألة بكاملها من جديد.

٤. تساعد المبرمج على متابعة دقة التسلسل خاصة في المسائل التي تكثر فيها الاحتمالات والتفرعات حيث يصبح هذا الأمر شاقا إذا لم يستعن بخرائط المسار.

٥. تعتبر خرائط المسار المستعملة في حلول بعض المسائل مرجعا للمسائل المشابهة ومفتاحا لحل مسائل جديدة لها علاقة بالمسائل القديمة المحلولة .

الجدول التالي يوضح الرموز والأشكال الاصطلاحية المستخدمة في خرائط المسار
:Flowchart

الشكل الاصطلاحي(الرمز)	معنى الرمز
	يستخدم هذا الرمز للدلالة على بداية البرنامج ونهايته (Start / Stop)
	يمثل هذا الرمز كل من عمليتي الإدخال (قراءة البيانات تمهيدا لمعالجتها) والإخراج (عرض النتائج على الشاشة، طباعة، ...) (Input / Output)
	رمز المعالجة، وقد يحتوي هذا الرمز على عملية حسابية أو عملية تخزين (Calculate and Store)
	عملية استدعاء (نداء) لبرنامج فرعي (Call Sub-Routine)
	رمز اتخاذ قرار، ويستخدم هذا الرمز في خطوات المعالجة التي تتطلب إجراء عملية منطقية كالمقارنة أو عملية اختبار Decision
	اتجاه سير البرنامج. تستخدم الأسهم لبيان حركة واتجاه خريطة التدفق. Directions
	نقطة توصيل وربط (Connector).
	تكرار أو دوران (Looping)
	تعليق وإيضاح (Comment)

أنواع خرائط سير العمليات:

١. خرائط سير النظم System Flowchart:

يستخدم هذا النوع من الخرائط عند تصميم الأجهزة الهندسية ، وعند تصميم نظام للمصانع أو الشركات وغيرها، والتي تستعمل أنظمة تحكم ذاتية ، مثل العوامات في خزانات المياه، وإشارات المرور الضوئية، و أجهزة ضبط الضغط ودرجات الحرارة في أبراج تقطير البترول، فتعتبر خرائط المسار هنا بمثابة المخطط الكامل الذي يبين ترتيب وعلاقة ووظيفة كل مرحلة بما قبلها وبما بعدها داخل إطار النظام المتكامل ويمكن تلخيص الدور الذي تقدمه هذه الخرائط بما يأتي:

١. تبين موقع كل خطوة من الخطوات الأخرى المكونة للنظام بحيث يسهل اكتشاف أي خلل يحدث في النظام كله بمجرد النظر ، مما ييسر عمليات صيانة الأجهزة وبأقل التكاليف.
٢. تسهل إجراء التعديلات التي قد تطرأ مستقبلاً على برنامج النظام في أي جزء منه.
٣. بيان التفصيلات عن المعطيات المطلوب إدخالها إلى النظام.
٤. بيان التفصيلات عن أنواع النتائج المتوقعة أو المطلوبة من البرنامج المعد للنظام.
٥. بيان طرق ربط النظام ببيقة الأنظمة الموجودة في المؤسسة المعنية.

٢. خرائط سير البرنامج Programs Flowchart:

ويستعمل هذا النوع من الخرائط لبيان الخطوات الرئيسية التي توضع لحل مسألة ما وذلك بشكل رسوم اصطلاحية تبين العلاقات المنطقية بين سائر خطوات الحل وموقع ووظيفة كل منها في إطار الحل الشامل للمسألة، ويكن تصنيف خرائط سير البرنامج إلى أربعة أنواع رئيسية:

- أ- خرائط التتابع البسيط (Simple sequential Flowchart).
 - ب- الخرائط ذات الفروع (Branched Flowchart).
 - ج- خرائط الدوران الواحد (simple-Loop Flowchart).
 - د- خرائط الدورانات المتعددة (Multi-Loop Flowchart).
- ويمكن للبرنامج الواحد أن يشمل أكثر من نوع واحد من هذه الأنواع .

أ- خرائط التتابع البسيط (Simple sequential Flowchart):

يتم ترتيب خطوات الحل لهذا النوع من الخرائط بشكل سلسلة مستقيمة من بداية البرنامج حتى نهايته ولا تحتوي على أية تفرعات أو دورانات .

مثال: اكتب الخوارزم للقيام بحساب مساحة ومحيط الدائرة بمعلومية نصف قطرها ، ثم ارسم خريطة سير البرنامج Flowchart المناظرة له؟

Write an Algorithm and Draw a Flowchart to calculate circumference and area of circle and display result?

الحل:

$$\text{مساحة الدائرة} = \pi R^2$$

$$\text{محيط الدائرة} = 2\pi R$$

حيث $\pi =$ النسبة التقريبية وقيمتها العددية ثابتة وتساوي 3.14 ، R نصف القطر وهو متغير

(معطيات)

خريطة سير البرنامج Flowchart	الخوارزم Algorithm	
<pre> graph TD Start([Start]) --> ReadR[/Read R/] ReadR --> PI[PI=3.14] PI --> A["A=PI*R*R"] A --> C["C=2*PI*R"] C --> Print["Print A, C"] Print --> Stop([Stop]) </pre>	<p>1- Start</p> <p>2- Read R</p> <p>3- put PI=3.14</p> <p>4- Calculate circle's area from the formula: A= PI*R*R</p> <p>5- Calculate circle's circumference from the formula: C=2*PI*R</p> <p>6- Print A, C,R</p> <p>7- Stop</p>	<p>١- ابدأ</p> <p>٢- اقرأ قيمة R</p> <p>٣- ضع قيمة PI= 3.14</p> <p>٤- احسب مساحة الدائرة A من المعادلة : A= PI*R*R</p> <p>٥- احسب محيط الدائرة C من المعادلة: C=2*PI*R</p> <p>٦- اطبع قيم كل من A,C,R</p> <p>٧- توقف</p>

ملاحظات:

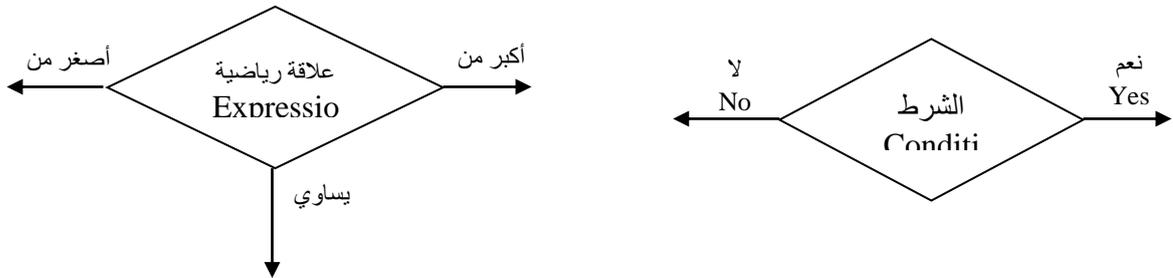
❖ تحديد رمز المساحة A والمحيط C اختياري ولكن من الأفضل أن تكون الرموز معبرة لمجتمويات المتغير فالرمز A يعبر عن المساحة Area ، والرمز C يعبر عن المحيط Circumference.

❖ عملية الضرب داخل خرائط سيرالعمليات يستخدم لها الرمز * أما القسمة فيستخدم لها الرمز / .

ب- الخرائط ذات الفروع (Branched Flowchart):

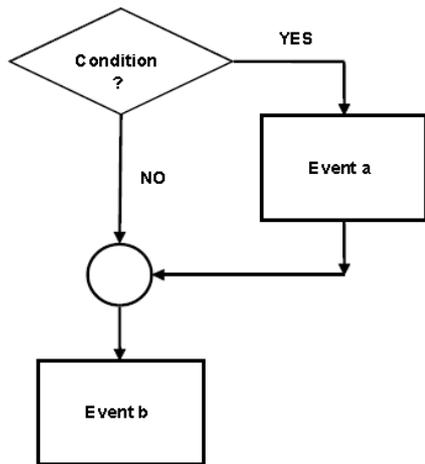
يحدث التفرع داخل البرنامج عند الحاجة لاتخاذ قرار أو مفاضلة بين اختياريين أو أكثر فيسير كل اختيار في طريق مستقل (تفرع) عن الآخر. وهناك نوعان من القرارات يمكن للمبرمج استعمال أحدها حسب المسألة المراد حلها.

أ) قرار ذو تفرعين ب) قرار ذو ثلاثة تفرعات

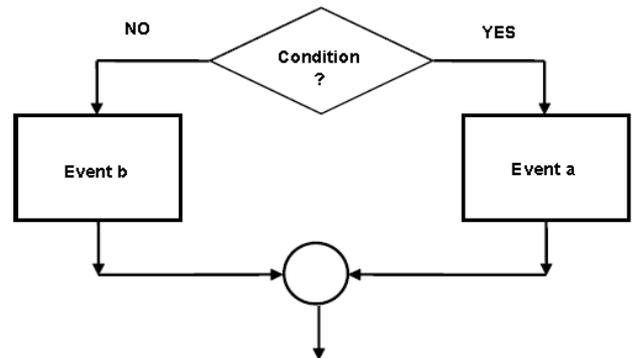


وبشكل عام فإن خرائط التفرع يمكن أن تأخذ إحدى الصورتين التاليتين:

شكل (ب)



شكل (أ)



❖ في شكل (أ) يبين أنه إذا كان جواب الشرط "نعم" فإن الحدث التالي في التنفيذ هو الحدث (a) ، أما إذا كان جواب الشرط "لا" فإن الحدث التالي في التنفيذ هو الحدث (b).

❖ في شكل (ب) يبين أنه إذا كان جواب الشرط "نعم" فإن الحدث التالي في التنفيذ هو الحدث (a) يليه الحدث (b)، أما إذا كان جواب الشرط "لا" فإن الحدث التالي في التنفيذ هو الحدث (b) مباشرة.

مثال: يوضح القرار ذو التفرعين

اكتبي الخوارزم للقيام بحساب قيمة الدالة $F(x)$ المعرفه كمايلي

$$F(x) = \begin{cases} x, & x \geq 0 \\ -x, & x < 0 \end{cases}$$

الحل:

خريطة سير البرنامج Flowchart	الخوارزم Algorithm	
<pre> graph TD Start([Start]) --> ReadX[/Read X/] ReadX --> Decision{X >=} Decision -- yes --> CalcYes[F(x) = x] Decision -- No --> CalcNo[F(x) = -x] CalcYes --> Join(()) CalcNo --> Join Join --> Print[/Print F(x), x/] Print --> Stop([Stop]) </pre>	1- Start	١- ابدأ
	2- Read x	٢- اقرأ قيمة X
	3- If x greater than or equal to Zero then go to step (4), else go to step (5).	٣- إذا كانت X أكبر من أو تساوي صفر اذهب إلى الخطوة (٤). وإلا فإذهب إلى الخطوة (٥).
	4- calculate $F(x) = x$, and then go to step (6)	٤- احسب قيمة الدالة $F(x)$ من $F(x) = x$ ثم اذهب إلى الخطوة (٦).
	5- calculate $F(x) = -x$	٥- احسب قيمة الدالة $F(x)$ من $F(x) = -x$.
	6- print $F(x)$, x	٦- اطبع قيمة $F(x)$, x
	7- Stop.	٧- توقف.

ملاحظة: عند تنفيذ الخطوة (٥) فإن الخطوة المنطقية التالية لها هي خطوة الطباعة (الخطوة (٦))، ولذلك لم يتم توجيه البرنامج بالعبارة اذهب إلى خطوة (٦)، كما أشرنا في الخطوة (٤) حيث تم توجيه البرنامج إلى الخطوة (٦) وتجاوز الخطوة (٥) حيث أن الخطوتان (٤)، (٥) لا تحققان معا.

مثال: يوضح القرار ذو ثلاثة تفرعات.

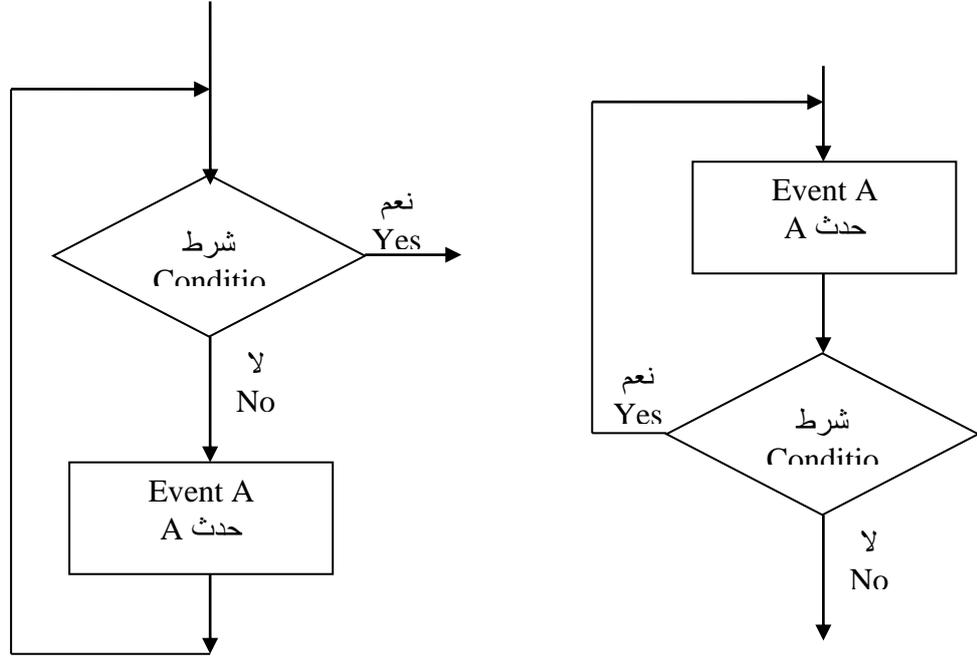
اكتبي الخوارزم و ارسمي خريطة سير العمليات لحساب قيمة W المعطاة من العلاقة التالية

$$W = \begin{cases} X + 1, & X > 0 \\ \sin(X) + 5, & X = 0 \\ 2X - 1, & X < 0 \end{cases}$$

Flowchart خريطة سير البرنامج	الخوارزمAlgorithm	
<pre> graph TD Start([Start]) --> ReadX[/Read X/] ReadX --> Decision{x} Decision -- "> 0" --> W1[W=X+1] Decision -- "= 0" --> W2[W=sin(X)+5] Decision -- "< 0" --> W3[W=2*X-1] W1 --> Merge(()) W2 --> Merge W3 --> Merge Merge --> Print[/Print W.X/] Print --> Stop([Stop]) </pre>	<p>1- Start</p> <p>2- Read x</p> <p>3- If x greater than Zero then go to step (4), If x equal to Zero then go to step (5), If x less than Zero then go to step (6).</p> <p>4- calculate $W = X + 1$, and then go to step (7)</p> <p>5- calculate $W = \sin(X) + 5$, and then go to step (7)</p> <p>6- calculate $W = 2 * X - 1$, and then go to step (7)</p> <p>7- print W , x</p> <p>8- Stop.</p>	<p>١- ابدأ</p> <p>٢- اقرأ قيمة X</p> <p>٣- إذا كانت X أكبر من صفر اذهب إلى الخطوة (٤). وإذا كانت X تساوي صفر فإذهب إلى الخطوة (٥)</p> <p>أما إذا كانت X أقل من صفر اذهب إلى خطوة (٦).</p> <p>٤- احسب قيمة W من المعادلة: $W = X + 1$ ثم اذهب إلى خطوة (٧).</p> <p>٥- احسب قيمة W من المعادلة: $W = \sin(X) + 5$ ، ثم اذهب إلى خطوة (٧).</p> <p>٦- احسب قيمة W من المعادلة: $W = 2 * X - 1$ ، ثم اذهب إلى خطوة (٧).</p> <p>٧- اطبع قيم كل من X, W</p> <p>٨- توقف.</p>

ج- خرائط الدوران الواحد (simple-Loop Flowchart):

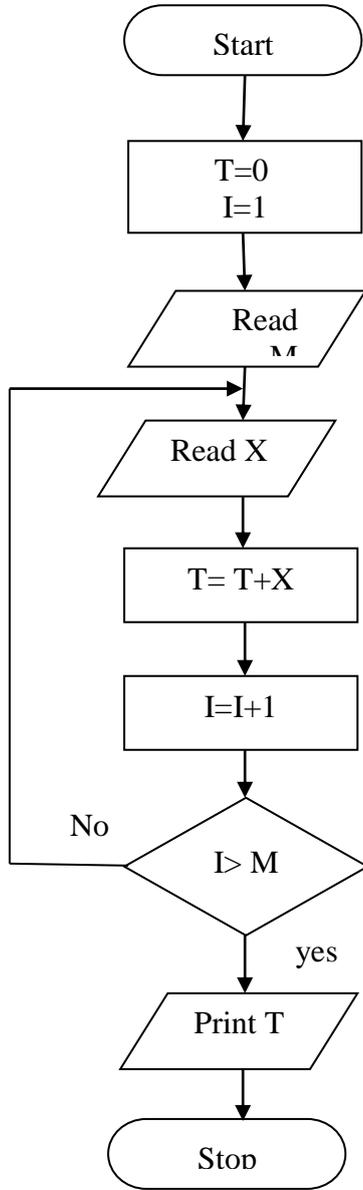
نحتاج هذا النوع من الخرائط لإعادة عملية أو مجموعة من العمليات في البرنامج عددا محدودا أو غير محدود من المرات ويكون الشكل العام لمثل هذه الخرائط كما يلي:



وقد سميت هذه الخرائط بخرائط الدوران الواحد لأنها تستعمل حلقة واحدة للدوران ، وتسمى أحيانا بخرائط الدوران البسيط.

مثال:

ارسم خريطة المسار لإيجاد مجموع m من الأعداد الحقيقية (X_1, X_2, \dots, X_m)
الحل:



النتيجة المطلوبة هي مجموع الأعداد T حيث أن :

$$T = \sum_{i=1}^m (X_i)$$

خطوات الحل يمكن أن تسيّر حسابيا على النحو:

$$T_0=0 , \quad T_1=T_0+X_1 , \quad T_2=T_1 + X_2$$

$$T_m=T_{m-1} + X_m = X_1 + X_2 + \dots + X_{m-1} + X_m$$

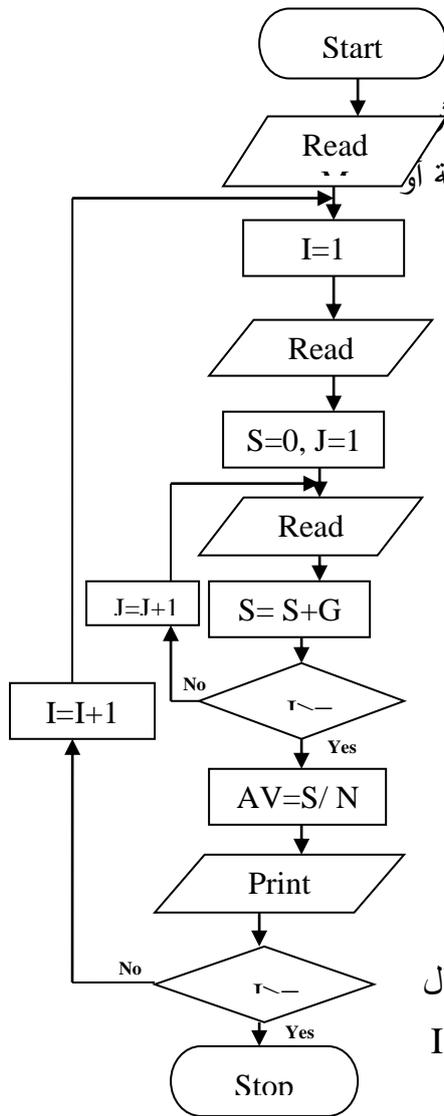
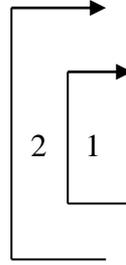
ونموذج الحل هذا يمكن أن يختصر بنموذج مكافئ هو

$$T = T + X_i , \quad i= 1,2, \dots, m$$

على أن تكون القيمة الأولى للمجموع T هي $T=0$

د- خرائط الدورانات المتعددة (Multi-Loop Flowchart):

في هذه الحالة تكون الدورانات داخل بعضها البعض بحيث لا تتقاطع ، فإذا كان لدينا دورانان من هذا النوع كما في الشكل التالي فيسمى الدوران رقم (١) دورانا داخليا Inner Loop بينما الدوران رقم (٢) دورانا خارجيا outer Loop ، ويتم التنسيق بين مثل هذين الدورانين ، بحيث تكون أولوية التنفيذ للدوران الداخلي.



وقد سميت هذه الخرائط بخرائط الدورانات المتعددة لأنها تستعمل أكثر من حلقة دوران واحدة، وقد تسمى أحيانا بخرائط الدورانات المتداخلة أو المتراكبة أو الضمنية .

مثال:

ارسمي خريطة المسار لإيجاد معدل درجات كل طالب إذا كان عدد المواد لكل طالب N ، وعدد طلاب الفرقة M ؟

ملاحظات:

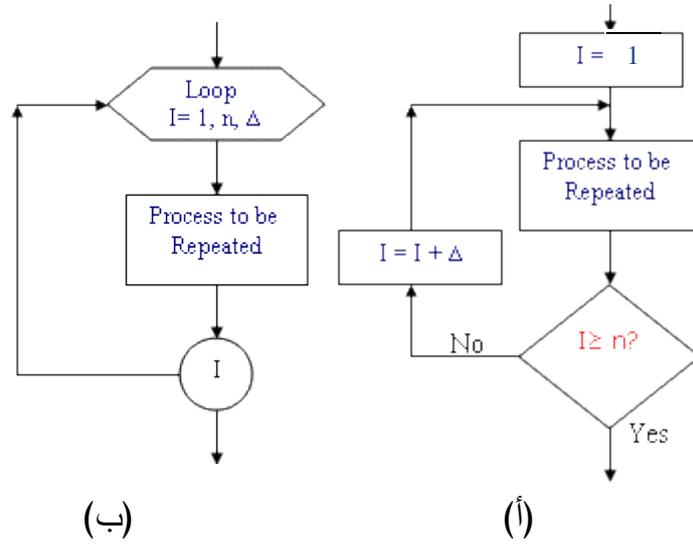
سوف نستخدم المتغير S كمخزن لتجميع درجات الطالب، والمتغير G يعبر عن درجة الطالب في المادة ، AV يمثل متوسط الدرجات، N عدد المواد المقررة لكل طالب ، M عدد الطلاب في كل فرقة ، J عداد لحصر عدد المواد N ، I عداد لحصر عدد الطلاب M .

يلاحظ من الشكل أن : الدوران الداخلي يقوم بقراءة وجمع درجات المواد لكل طالب على حده ، بينما الدوران الخارجي يقوم بحساب متوسط درجات الطالب الكلية ثم طباعته قبل البدء في إدخال درجات طالب آخر ، العداد J للخروج من الدوران الداخلي ، العداد I للخروج من الدوران الخارجي .

صيغة الدوران باستعمال الشكل الاصطلاحي (الدوران التكراري Loop):

في الفقرتين السابقتين تعلمنا مفهوم الدوران البسيط والدورانات المتعددة (المتداخلة) ويمكننا الآن استخدام الشكل الاصطلاحي (الدوران التكراري Loop). نلاحظ من المثال السابق أننا نحتاج إلى العناصر الآتية:

- ❖ القيمة الأولية للعداد I (هنا $I=1$).
- ❖ القيمة النهائية للعداد I (هنا $I=N$).
- ❖ قيمة الزيادة عند نهاية كل دورة Δ .

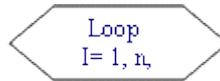


نلاحظ في الشكل السابق الجزء (أ) أن إجراءات الدوران كانت تتم طبقاً للخطوات الآتية والمفصلة من قبل المبرمج:

١. أعط العداد I قيمة أولية.
٢. أتم الإجراءات المطلوب إعادةتها.
٣. اتخاذ قرار: إذا كانت قيمة العداد I وصلت إلى القيمة النهائية N فاخرج إلى الخطوة (٦) في البرنامج وإلا فإذهب إلى الخطوة (٤).
٤. زد I بمقدار Δ .
٥. عد إلى الخطوة (٢)
٦. أكمل ما تبقى من البرنامج.

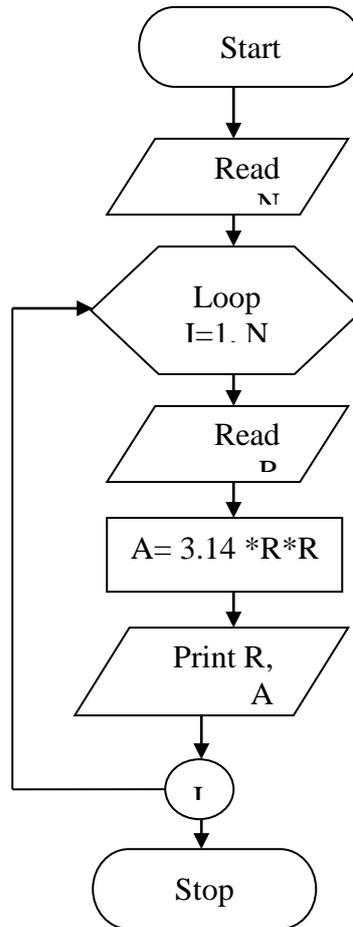
يمكننا استبدال الخطوات المفصلة (١، ٣، ٤، ٥) في الشكل (أ) بخطوة مجملة واحدة مبيّنة في الشكل الاصطلاحي للدوران (ب) حيث تنفذ هذه العملية بصورة أوتوماتيكية من قبل الحاسب، وهذا من شأنه تسهيل عملية البرمجة واختصار عدد التعليمات في البرنامج وتجنب بعض الأخطاء.

ملحوظة: تعتبر قيمة n تساوي ١ دائماً إذا لم تعط قيمة أخرى بخلاف ذلك، وفي حالة عدم ذكر قيمة n يصبح الشكل الاصطلاحي (الدوران المتكرر) الوارد في الشكل (ب) كما يلي حيث تكون قيمة n تساوي ١ وبصورة أوتوماتيكية.



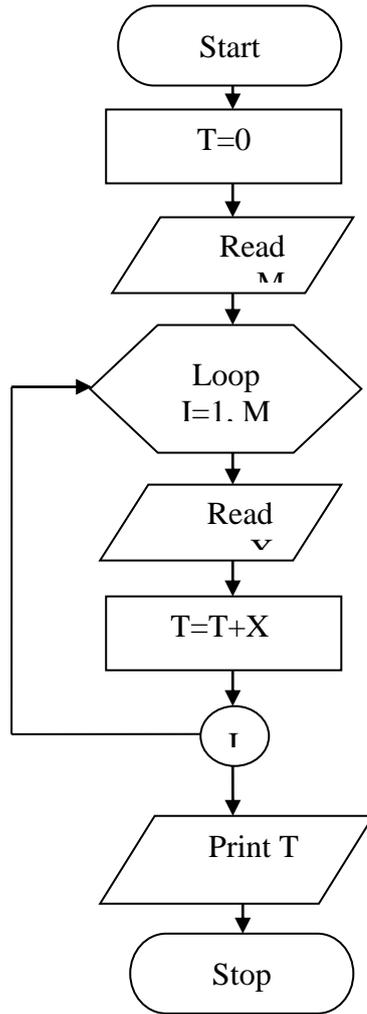
مثال: ارسم خريطة المسار لإيجاد مساحة N من الدوائر باستخدام الشكل الاصطلاحي للدوران.

الحل:



مثال: ارسم خريطة المسار لإيجاد مجموع M من الأعداد الحقيقية X_1, X_2, \dots, X_m

الحل:



٤- مرحلة الترجمة

اللغة الوحيدة التي يفهمها الحاسب هي لغة الآلة ولكي يفهم الحاسب البرنامج الذي تم كتابته بإحدى لغات البرمجة فإنه يتم تحويله إلى لغة الآلة وذلك من خلال استخدام المترجمات، وعلى حسب لغة البرمجة يتم استخدام (المجمع أو المترجم أو المفسر) ليقوم بتحويل البرنامج إلى لغة الآلة.

٥- تنفيذ البرنامج

بعد الحصول على برنامج الهدف (مكتوب بلغة الآلة) تتم تجربة البرنامج للتأكد من صحته منطقياً وذلك بتنفيذه باستخدام عينه من البيانات الاختبارية Test Data، فإذا ما ثبت صحة البرنامج منطقياً فإن ذلك يعني صحة طريقة الحل المقترحة ويمكن بعدها تنفيذ البرنامج على البيانات الحقيقية.

Some Questions:

1. Draw a flowcharts that find out the roots of a second degree

$$\text{polynomial } X_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Where $b^2 - 4ac > 0$ and $a \neq 0$

Draw a flowcharts for the following equations:

2. $M = \frac{\sum_{i=1}^N X_i}{N}$ where n is an integer number

3. $Y = (x+1) \times (x+2) \times (x+3) \times \dots \times (x+N)$

4. $Y = n!$ where n is an integer number

الفصل الثانی

البرمجة بلغة الـ C#

C# Programming Language

✚ ان لغة السي شارب تتبع مجموعة لغات .Net Frame Work وهذه البيئه
✚ تحتوى على عدة لغات لكن كل منهم له الكود الخاص به وتوجد بها لغه وسيطيه
Microsoft Intermediate Language (MSIL) وبالتالى هى لغه لا تعتمد
على الاله فهى تعمل على اى جهاز

✚ ان C# تضم مميزات VB - C++ بما لها من سهولة التعامل فى الواجهات
الرسوميه بالاضافه الى قوة البرمجه .

✚ ان الفرق الجوهرى بين C# و C++ هو كالتالى :

C++	C#
هى لغة تستطيع بها ان تستخدم مفهوم البرمجه كائنية التوجه	هى لغة مبنيه بالاساس على مفهوم البرمجه كائنية التوجه او Object Oriented

الهيكـل التنظيمي لبرنامج C# :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            Main()
        }
    }
}
```

namespace
الاساسيه والتي تستدعي عند كتابه اي برنامج

class
الرئيسي للبرنامج ويوجد بداخله الداله الرئيسيه

يكتب هنا داخل الداله الرئيسيه الكود الذي سوف ينفذ فهو يبدء التنفيذ من الداله Main()

نأتي لتفسير التكوين الرئيسي لبرنامج C#

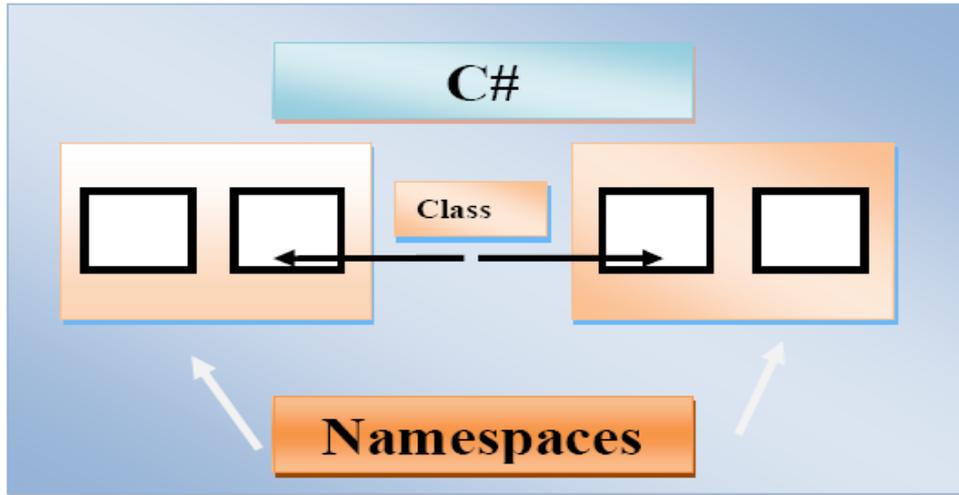
Using هي كلمه محجوزه داخل اللغه وتستخدم للنداء على اسم معين ليصبح

جزء من البرنامج

Namespace ويعرف مجازا بفضاء الاسماء وهو عباره عن مجلد يحتوى

بداخله على Classes جاهزه للعمل عليها ولها اسماء محدده

ونستطيع انه نمثله بالشكل التالي :



Classes هي وحدات محجوزه داخل اللغة وتحتوي على جزئين رئيسيين:

1- **Data Member** وهي المتغيرات والتي تحتوي على بيانات

2- **Function Member** دوال تقوم بتنفيذ عمليات على الدوال المخزنه

ان اي برنامج داخل C# قد يحتوي على اكثر من **namespace** على الاقل

لابد من وجود واحد فقط وقد يحتوي على اكثر من **Class** لكن على الاقل

يجب ان يحتوي على **Class** واحد

وبعد **System** هو **namespace** الرئيسي بصفته يحتوي على **classes**

تحتوي على دوال التوصيف للدخل والخرج داخل اي برنامج

ان وظيفة الداله الرئيسييه Main() ان Compiler بيبدء دائما بتنفيذ خطوات

البرنامج من عندها ايا كان موضعها ولا بد ان تكون موجوده داخل class

الرئيسي للبرنامج والذي يعطى اسم الافتراضى Program ومن الممكن

وضع تلك الداله داخل اى class وسعتبره انه الرئيسى الذى يحتوى على

تلك الداله

ان لغة C# حساسه لحالة الاحرف بمعنى انه يجب ان يراعى فيها capital

او small

شرح الاوامر البرمجيه داخل لغة C# :-

اوامر الطباعه :

هناك امران اساسيان داخل C# وهما كيفية التعامل مع الدخل والخرج والذي

لاغنى عنهم داخل اى برنامج وهما ممثلان فى دالتين يتبعان نفس class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            string myname;
            myname = Console.ReadLine();
            Console.WriteLine("hello", myname);
            Console.ReadLine();
        }
    }
}
```

الداله ReadLine()
خاصه باستلام المدخلات داخل اى برنامج

الداله WriteLine()
خاصه باظهار المخرجات

دوال الدخل والخرج والتي تتبع جميعها التصنيف Console	
هذه الدالة تقوم بطباعة الخرج ويظل المؤشر في مكانه	Write()
هذه الدالة تقوم بطباعة الخرج وينتقل المؤشر الى سطر جديد اسفل الخرج	WriteLine()
تقوم هذه الدالة بقراءة المدخلات ودائما ما تعتبر لغة السي شارب ان جميع المدخلات على انها نصوص لذلك يجب تحويل المدخلات بعد قراتها	ReadLine()

معامل تنظيم ظهور الخرج على الشاشة وما يعرف Replacement Operator

```

string myname;
string hellomessage;
myname = Console.ReadLine();
hellomessage = "hello mr";
Console.WriteLine("hello{0}, {1}", myname, hellomessage);
Console.ReadLine();

```

Replacement Operator
والذي يشير الى ترتيب ظهور قيم الخرج على الشاشة حيث يبدأ بالرقم 0 ثم 1 وهكذا

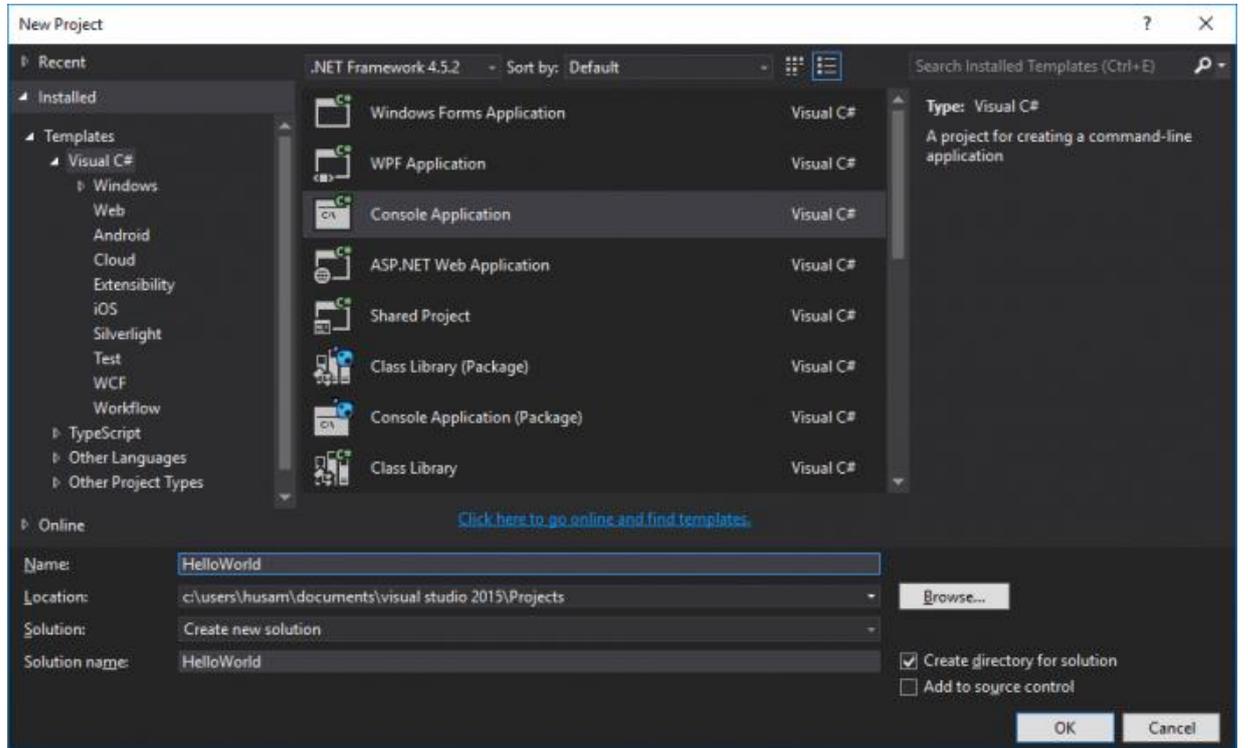
مكنك تنفيذ البرامج الموجودة ضمن هذه السلسلة (كما وسبق أن ذكرنا في المقدمة (بطريقتين مختلفتين):

الأولى هي تحميل وتنصيب بيئة التطوير المجانية Visual Studio 2015 Community من مايكروسوفت. توفر هذه البيئة خدمات عظيمة للمطور وتسهل عملية كتابة البرامج إلى حد كبير. ويمكنك الاستفادة من مزايا تنقيح الأخطاء debugging المتقدمة التي يوفرها المنقح debugger المرفق ضمن هذه البيئة. يمكنك تحميل نسختك المجانية من هنا.

الطريقة الثانية هي في استخدام الموقع .NET Fiddle الذي يوفر مزية تنفيذ البرامج التي تكتبها على خادم خاص به، ومن ثمّ يعرض لك خرج البرنامج، بدون أن تمتلك نظام تشغيل ويندوز حتى.

البرنامج الأول

سنبدأ بمثال عمليّ لنسبر سريعاً أغوار هذه اللغة. شغل برنامج Visual Studio 2015 Community ثم من القائمة **File** اختر الأمر **New > Project** من نافذة مشروع جديد **New Project**، اختر من القسم الأيسر **Visual C#** ومن القسم الأيمن اختر **Console Application** اكتب **HelloWorld** ضمن حقل الاسم **Name** في القسم السفلي، ثم اضغط زر **OK**. انظر الشكل التوضيحي التالي :



قد تبدو الصورة مختلفة بعض الشيء لديك بحسب إعدادات الإظهار التي اخترتها. سيعمل Visual Studio على إنشاء هذا التطبيق وفتح ملف مُجهّز خصيصاً لك اسمه **Program.cs**. امسح محتويات هذا الملف بالكامل ثم انسخ الشيفرة التالية ضمنه:

```

1  using System;
2
3  namespace HelloWorld
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10         }
11     }
12 }

```

اضغط المفاتيح **Ctrl+F5** معاً لتبدأ عملية بناء **build** البرنامج وتنفيذه لتحصل على العبارة **Hello World!** في خرج البرنامج (أو من القائمة **Debug > Start Without Debugging**) رغم أنّ البرنامج السابق بسيط جداً إلا أنه يحتوي على الكثير من المفاهيم الجديدة التي سنتناولها تباعاً في هذه السلسلة.

يبدأ هذا البرنامج بتعريف نطاق اسم (**namespace** السطر ٣) اسمه **HelloWorld** وهو نفس الاسم الذي زوّدناه للبرنامج، سنتكلم عن نطاقات الأسماء في درس لاحق، ولكن يكفيك أن تعرف الآن أنّ نطاقات الأسماء هي وسيلة لتنظيم الأصناف ضمن مجموعات مترابطة منطقياً. يأتي بعد ذلك تعريف صنف **class** جديد اسمه (**Program** السطر ٥). يحتاج أيّ برنامج مكتوب بالسي شارب إلى نقطة دخول **entry point** لكي يبدأ تنفيذه. نقطة الدخول يجب أن تكون عبارة عن تابع (**method** اسمه **Main** السطر ٧)، تكون التتابع عادةً ضمن الأصناف، يكفيك الآن أن تفهم التابع على أنه شبيه بالدالة **function** في لغات البرمجة الأخرى. أي هو جزء من الشيفرة يمكن استدعاؤه لتنفيذ ناحية وظيفية مُحدّدة في البرنامج وقد يُرجع قيمة ما أو لا يُرجع أيّ شيء. العبارة الموجودة في السطر ٩ هي عبارة برمجيّة قياسية في لغة سي شارب. وظيفة هذه العبارة استدعاء التابع **WriteLine** من الصنف **Console** وتمرير النص **!Hello World** له لكي يُظهر النص **Hello World!** في خرج البرنامج.

أيّ عبارة برمجيّة في سي شارب يجب أن تنتهي بفاصلة منقوطة (**;**)، وقد تكون العبارة البرمجيّة مجرد استدعاء تابع أو أن تكون عملية إسناد إلى متغيّر، أو قد تكون مزيجاً بينهما.

إذا كانت لديك معرفة سابقة بلغات برمجة مثل C أو ++C أو Java ستلاحظ أن الصيغة النحوية syntax للغة سي شارب تشبه إلى حد كبير الصيغة النحوية لهذه اللغات. حيث تُستخدم الحاضنات { } مثلًا لتحديد البداية والنهاية للتابع method وللصنف class ولنطاق الاسم namespace وحتى أنهما يشكّان حدود أيّ بنية برمجية في لغة سي شارب مثل العبارات التكرارية. انظر على سبيل المثال إلى السطر ٦ لتجد الحاضنة "}" الخاصة بالصنف Program وإلى السطر ١٢ لتجد حاضنة الإغلاق "{" له. كما ينبغي التنبّه أيضًا إلى كون لغة سي شارب حساسة لحالة الأحرف كما هو الحال في لغات البرمجة C و ++C و Java.

ملاحظة

يمكن استخدام المفتاح F6 في بيئة Visual Studio أو من القائمة **Build > Build Solution** لبناء البرنامج دون تشغيله (تنفيذه) وذلك اعتبارًا من الشيفرة والحصول على ملف تنفيذي منه له الامتداد exe في حال كان البرنامج لا يحتوي على أيّ خطأ.

برنامج بسيط لجمع عددين صحيحين

لنعمل الآن على برنامج عملي أكثر. سنكتب برنامج يعمل على جمع عددين صحيحين وإظهار النتيجة للمستخدم. اتبع نفس الخطوات التي أجريناها في البرنامج السابق لإنشاء برنامج جديد

باسم SumTwoNumbers، انسخ محتويات الشيفرة التالية إلى الملف Program.cs:

```
1 using System;
2
3 namespace SumTwoNumbers
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int a;
10            int b;
11            int c;
12
13            a = 3;
```

```

14         b = 4;
15
16         c = a + b;
17
18         Console.WriteLine("3 plus 4 equals: " +
c.ToString());
19
20     }
21 }
22 }

```

يُقدّم هذا البرنامج البسيط مفهوم التصريح عن المتغيّرات والتعامل معها. صرّحنا في الأسطر من ٩ إلى ١١ عن ثلاثة متغيّرات *a* و *b* و *c* من النوع `int`.

يجب التصريح في لغة سي شارب عن كل متغيّر قبل استخدامه في البرنامج. لاحظ أنّ التصريح عن متغيّر يتمّ بذكر نوعه ومن ثمّ اسمه. يستطيع المتغيّر من النوع `int` استيعاب أي عدد صحيح (دون فاصلة عشرية) يقع بين $-2,147,483,648$ و $2,147,483,647$. لاحظ أنّه قد أسندنا القيمتين ٣ و ٤ إلى المتغيّرين *a* و *b* على الترتيب، وذلك في السطرين ١٣ و ١٤. نجري عملية الجمع والإسناد إلى المتغيّر *c* في السطر ١٦. وفي السطر ١٨ نعرض رسالة للمستخدم .

يمكن إسناد قيمة للمتغيّر مباشرةً عند التصريح عنه. فمن الممكن مثلاً إسناد القيمتين ٣ و ٤ للمتغيّرين *a* و *b* على الترتيب عند التصريح عنهما وذلك بالشكل التالي:

```

int a = 3;
int b = 4;

```

كما يمكن استخدام عبارة تصريح واحدة للتصريح عن عدّة متغيّرات بنفس الوقت. فمثلاً كان من الممكن التصريح عن المتغيّرات *a* و *b* و *c* السابقة بعبارة برمجية واحدة على الشكل التالي:

```

int a, b, c;

```

في الواقع هناك شكلان من أنواع المتغيّرات تدعمهما منصّة دوت نت. أنواع القيمة `value types` والأنواع المرجعية `reference types`. سنتحدّث عنهما لاحقاً.

يُعتبر النوع `int` نوع قيمة. يجب إجراء عملية إسناد واحدة على الأقل إلى متغيّر قيمة قبل القراءة منه. وإلا سنحصل على خطأ. جرّب حذف العبارة البرمجية الموجودة في السطر ١٦ والمسؤولة عن إسناد قيمة المجموع إلى المتغيّر `c`. نفذ البرنامج وستحصل على الخطأ التالي:

```
Use of unassigned local variable 'c'
```

سبب ذلك أننا حاولنا قراءة المتغيّر `c` في السطر ١٨ دون أن نُسند أيّ قيمة له. العبارة البرمجية الموجودة في السطر ١٨ مسؤولة عن عرض الرسالة إلى المستخدم كما أسلفنا. ستلاحظ أننا مررنا التعبير `expression` التالي إلى التابع `WriteLine`:

```
"3 plus 4 equals: " + c.ToString()
```

التعبير البرمجي هو مفهوم موجود في جميع لغات البرمجة تقريباً، وهو ببساطة ناتج عملية برمجية باستخدام عامل `operator` واحد أو أكثر، أو استدعاء إلى تابع أو مزيج بينهما. نسمي عملية معالجة التعبير بتقييم التعبير. `expression evaluation` العامل المُستخدم هنا هو عامل ضمّ النصوص `(+)`، ولعلّك تستغرب لماذا أدعوه بعامل ضمّ النصوص رغم أنّه يشبه عامل الجمع العادي الذي يجمع عددين مع بعضهما (انظر السطر ١٦). يعود السبب في ذلك إلى نوع المُعاملين `operands` الموجودين على طرفيه. يمكنك أن تلاحظ بسهولة أنّ المعامل الأيسر هو النص:

```
"3 plus 4 equals: "
```

أمّا المعامل الأيمن فهو:

```
c.ToString()
```

وهو أيضاً نص ويعود سبب ذلك إلى استدعاء التابع `ToString` من المتغيّر `c` المعرّف أصلاً أنّه متغيّر من نوع `int`. ولكنّ استدعاء هذا التابع من المتغيّر `c` يؤدّي إلى الحصول على التمثيل النصّي للقيمة العددية الموجودة ضمنه أصلاً. فإذا كان المتغيّر `c` يحمل القيمة العددية ٧، فإنّ التابع `ToString` سيُرجع النص "٧"، الذي يعمل عامل الضمّ `+` على ضمّه مع النص الذي يسبقه لتكون نتيجة التعبير ككل هي:

```
"3 plus 4 equals: 7"
```

سيُمرّر هذا النص إلى التابع `WriteLine` لعرضه للمستخدم.

أعتقد أنك قد بدأت بفهم طريقة الوصول إلى التتابع واستدعائها في لغة سي شارب. فنحن نستخدم اسم الصنف (أو الكائن object كما سنرى لاحقاً) الذي يحوي التابع المراد استدعاؤه متبوعاً بنقطة ثم باسم التابع المطلوب، وبعد ذلك قوسين هلاليين نمزّر بينهما الوسائط التي يطلبها التابع إذا اقتضت الضرورة لذلك.

برنامج محسن أكثر لجمع عددين

لقد تعلمنا العديد من المفاهيم الجديدة من خلال البرنامجين السابقين. ولكن لعلك قد لاحظت من برنامج جمع العددين السابق أنّ البرنامج جامد بعض الشيء. فهو يجمع عددين مُحدّدين سلفاً. سنعمل في هذه النسخة المطوّرة من البرنامج على استقبال العددين المراد جمعهما من المستخدم ومن ثمّ إجراء عملية الجمع عليهما وعرض النتيجة على المستخدم، مع بعض التحسينات الإضافية الأخرى.

أنشئ مشروعاً جديداً باسم EnhancedSumTwoNumbers ثمّ استبدل محتويات الملف Program.cs بالشفيرة التالية:

```
1 using System;
2
3 namespace EnhancedSumTwoNumbers
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             string str1, str2, result;
10            double num1, num2, sum;
11
12            //prompt user to get input for first value.
13            Console.WriteLine("Input first number: ");
14            str1 = Console.ReadLine();
15
16            //prompt user to get input for second value.
17            Console.WriteLine("Input second number: ");
18            str2 = Console.ReadLine();
19
```

```

20         //convert the input values to double numbers.
21         num1 = double.Parse(str1);
22         num2 = double.Parse(str2);
23
24         //perform sum operation.
25         sum = num1 + num2;
26
27         /*concatenate strings to form output
28         message which contains the result.*/
29         result = num1.ToString() + " + " +
num2.ToString() + " = " + sum.ToString();
30
31         Console.WriteLine(result);
32
33     }
34 }
35 }

```

نفذ البرنامج بضغط المفاتيح **Ctrl+F5** معًا. سيطلب منك البرنامج في البداية إدخال قيمة العدد الأول. أدخل القيمة المرغوبة ثم اضغط مفتاح الإدخال **Enter**. بعد ذلك سيطلب منك البرنامج إدخال قيمة العدد الثاني. أدخلها واضغط **Enter** سيعرض البرنامج بعد ذلك النتيجة المطلوبة على شكل رسالة مناسبة .

تحتوي هذه النسخة من برنامج جمع الأعداد على عدّة تحسينات :

أصبح برنامجنا يدعم جمع أعداد تقبل فاصلة عشرية من خلال التصريح عن متغيرات من النوع **double** (النوع **double** انظر السطر ١٠). والنوع **double** هو النوع الذي يقبل أعدادًا ذات فاصلة عائمة مزدوجة الدقة. مجال الأعداد التي يقبلها يقع بين $\pm 5.0 \times 10^{-324}$ حتى $\pm 1.7 \times 10^{308}$.

أصبح بإمكان مستخدم البرنامج أن يُدخل الأعداد المراد جمعها مباشرة من لوحة المفاتيح، وذلك من خلال التابع **ReadLine** (من الصنف **Console** انظر السطرين ١٣ و ١٦) يُوقف هذا التابع تنفيذ البرنامج وينتظر المستخدم أن يُدخل قيمة ما ويضغط مفتاح الإدخال **Enter** ليتابع البرنامج التنفيذ .

أضفنا تعليقات توضيحية ضمن البرنامج. هذه التعليقات لا يكثر بها المترجم، ووظيفتها هي جعل الشيفرة البرمجية مقروءة وسهلة الفهم والتعديل لاحقاً. في الحقيقة تُعتبر عملية كتابة التعليقات البرمجية فناً بحد ذاته، وأنصح أن يمارسها كل مبرمج بأي لغة برمجة كانت. في الواقع ليس مطلوباً كتابة التعليقات البرمجية قبل كل عبارة برمجية، فعلى المرء أن يكون حكيماً في استخدامها بالشكل الذي يحافظ فيه على التوازن بين جعل الشيفرة واضحة ومقروءة، وعدم الإفراط في كتابة التعليقات بدون ضرورة. تدعم لغة سي شارب نوعين من التعليقات: التعليقات على مستوى السطر، حيث يتجاهل المترجم compiler كل ما يقع على يمين الرمز // والتعليقات التي تمتد على عدة أسطر، حيث يتجاهل المترجم المحتوى الموجود بين الرمز /* والرمز */. انظر الأسطر ١٢ و ١٦ و ٢٠ و ٢٤ من أجل التعليقات على مستوى السطر، والسطرين ٢٧ و ٢٨ من أجل التعليقات التي تمتد على عدة أسطر .

استخدمنا التابع Write بدلاً من التابع WriteLine انظر السطرين ١٣ و ١٧)، والسبب في ذلك هو أننا نريد أن يطلب البرنامج من المستخدم إدخال القيمة على نفس السطر الذي تُعرض فيه الرسالة وليس على سطر منفصل. فالتابع Write يعرض النص المُمرّر إليه ولا يُحدث سطرًا جديدًا. في حين يسلك التابع WriteLine نفس سلوك التابع Write ولكن ينتقل إلى سطر جديد بعد عرض النص. يمكنك أن تجرب استبدال التابع WriteLine بالتابع Write لترى الفرق .

جعلنا عملية تشكيل النص الذي سيُعرض على المستخدم ضمن سطر منفصل (السطر ٢٩) وأسندنا هذا النص إلى المتغير result من النوع string. الهدف من هذا الأمر هو جعل الشيفرة نظيفة وواضحة وسهلة للقراءة. النوع string هو من الأنواع المرجعية reference types ويُستخدم للتعبير عن النصوص.

ولكن تبقى هناك بعض العيوب التي لم نعالجها والتي قد تسبب توقّف البرنامج عن العمل: تُعتبر القيم التي يدخلها المستخدم بواسطة التابع ReadLine أنها قيم نصية. وحتى نستطيع التعامل معها كأعداد تقبل فاصلة عشرية يجب تحويلها إلى قيم عددية من النوع double. نستطيع ذلك بسهولة من خلال التابع Parse من الصنف double. يقبل هذا التابع أن تُمرّر إليه قيمة نصية ليعيد إلينا التمثيل العددي لها من النوع double. ولكن السؤال هنا أنه ماذا لو أدخل المستخدم بشكل غير مقصود (أو مقصود) القيمة النصية التالية "abc" للعدد الأول؟

عندما يصل تنفيذ البرنامج إلى السطر ٢١ سيعمل التابع Parse على تحويل القيمة "abc" إلى التمثيل العددي من النوع double وهذا ما لا يمكن حدوثه بالطبع، لذلك فسيؤدي التابع Parse استثناءً Exception سيؤدي إلى توقّف البرنامج عن العمل فوراً! سنتحدّث عن الاستثناءات في درس لاحق. وعلى أيّة حال يمكن حلّ هذه المشكلة بطريقتين مختلفتين سنتحدّث عنهما لاحقاً في هذه السلسلة. ولكنّ المغزى هنا هو أنّه لا تثق بمدخلات المستخدم مطلقاً.

تُعتبر عملية ضم النصوص التي أجريناها في السطر ٢٩ غير عمليّة وعادة برمجيّة غير جيّدة. يتعلّق هذا الأمر بالحقيقة طريقة تعامل سي شارب مع النصوص، سأترك الحديث عن هذه المشكلة وطرق حلّها عندما نتحدّث عن النصوص والتعامل معها في الدرس السادس.

تمارين

تمرين ١

اكتب برنامجاً يطبع العبارات التالية كما يلي على الشاشة:

```
Today is Sunday.  
Tomorrow is Monday.  
Yesterday is Saturday.
```

تمرين ٢

اكتب برنامجاً يطلب من المستخدم إدخال قيمتين عدديتين، ومن ثمّ يوجد حاصل الضرب لهما (استخدام العامل *) ويعرض النتيجة على الشاشة. يجب أن يدعم البرنامج الأعداد ذات الفاصلة العشرية.

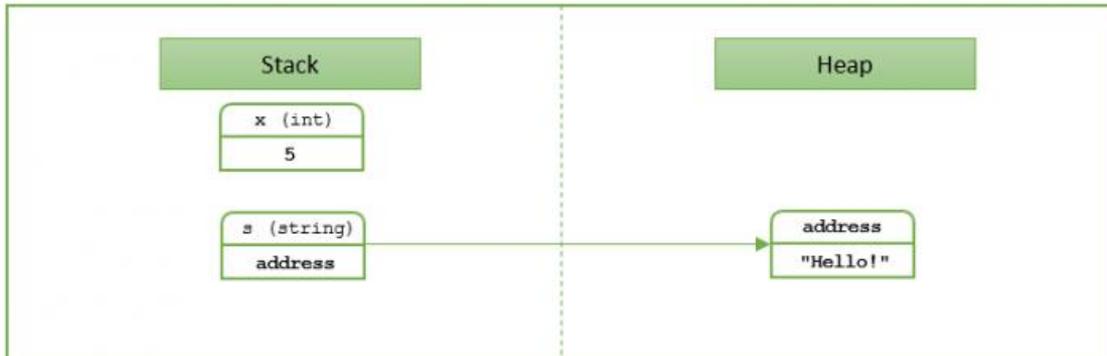
المتغيرات

يوجد نمطان أساسيان لأنواع المتغيرات في سي شارب، وهما: أنواع قيمة value types وأنواع مرجعية reference types.

تشتمل أنواع القيمة على الأنواع المُدمجة built-in في اللغة مثل int و float و decimal و double و bool وجميع الأنواع المُعرّفة كبنية struct.

في حين تشتمل الأنواع المرجعية على أي نوع آخر وهذا يشتمل على عدد لا يحصى من الأنواع، فيكفيك أن تعرف مثلاً أن جميع الأنواع الموجودة في مكتبة FCL هي أنواع مرجعية، بالإضافة إلى أن أي نوع جديد (على شكل صنف class) ينشئه المستخدم يُعتبر نوعاً مرجعياً. ومن المفيد أن تعلم أن النوع المُضمّن string هو نوع مرجعي أيضاً.

يُمكن الفرق الأساسي بين أنواع القيمة والأنواع المرجعية في مكان وطريقة تخزين قيم المتغيرات المصرّح عنها بواسطتها. فعند التصريح عن متغير من نوع قيمة، تُخزن أي قيمة يتم إسنادها إليه ضمن المتغير نفسه أو بمعنى أدق تُخزن في ذاكرة المُكدّس Stack Memory، أما المتغير المصرّح عنه على أنه نوع مرجعي فالذي يُخزن ضمنه هو العنوان إلى موقع في الذاكرة. هذا الموقع موجود ضمن ما يُسمّى بذاكرة الكومة Heap Memory. انظر إلى الشكل التوضيحي التالي.



حيث صرّحنا عن المتغير x من النوع int وخزّننا القيمة ٥ ضمنه. وبما أن int هو نوع قيمة، لذلك سيكون المتغير مع القيمة المخزّنة فيه ضمن المكدّس Stack.

أما بالنسبة للمتغير s فهو من النوع string وقد أسندنا إليه النص "Hello!" وبما أن النوع string هو نوع مرجعي كما أسلفنا لذلك فالقيمة التي ستكون مخزّنة ضمن المتغير s في

الحقيقة ليست النص "Hello!" إنما العنوان address الذي يُشير إلى موقع ضمن الكومة Heap موجود ضمنه النص "Hello!"، وهذا بالمناسبة ليس سلوكًا تنفرد به سي شارب، بل هو موجود في لغات أخرى مثل C++ و C.

سنستوع في هذا الموضوع قليلًا عندما نتحدّث عن الأصناف والكائنات لاحقًا في هذه السلسلة.

يمكن استخدام أيّ مزيج من الحروف والأرقام عند تسمية المتغيّرات، بشرط أن يكون أوّل محرف في اسم المتغيّر حرفًا وليس رقمًا. كما لا يجوز أن يحتوي اسم المتغيّر على فراغات ولا يجوز أيضًا أن يكون مماثلًا لكلمة محجوزة في سي شارب مثل new أو class أو غيرها، ولا يجوز أن يحتوي على رموزًا خاصّة مثل & و\$ و#، ولكن يُعتبر الرمز (_) underscore حرفًا ويجوز الابتداء به.

١. أنواع البيانات

الأنواع المُضمّنة هي الأنواع الموجودة ضمناً في لغة سي شارب وفي إطار عمل دوت نت عموماً. سنستعرض في الجدول التالي هذه الأنواع. لاحظ أنّ عمود "الاسم" يحتوي على أسماء الأنواع المستخدمة في سي شارب، في حين يحتوي العمود الذي يليه مباشرةً على اسم نفس النوع ولكن ضمن منصّة دوت نت. في الحقيقة يمكننا استخدام أي تسمية نرغبها ولكنّ الأسهل والأفضل هي في استخدام أسماء الأنواع في سي شارب. يعود سبب وجود أسماء أنواع مختلفة في إطار عمل دوت نت هو أنّه بإمكان أي لغة برمجة ضمن منصّة دوت نت استخدام نفس هذه الأنواع ضمنها.

الاسم	النوع الموافق في منصّة دوت نت	القيم التي يقبلها	الحجم في الذاكرة
bool	System.Boolean	true أو false	
sbyte	System.SByte	من ١٢٨ - حتى ١٢٧	٨ bits
byte	System.Byte	من ٠ حتى ٢٥٥	٨ bits

bits ١٦	من ٣٢,٧٦٨ - حتى ٣٢,٧٦٧	System.Int16	short	٤
bits ١٦	من ٠ حتى ٦٥,٥٣٥	System.UInt16	ushort	٥
bits ٣٢	من ٢,١٤٧,٤٨٣,٦٤٨ - حتى ٢,١٤٧,٤٨٣,٦٤٧	System.Int32	int	٦
bits ٣٢	من ٠ حتى ٤,٢٩٤,٩٦٧,٢٩٥	System.UInt32	uint	٧
bits ٦٤	من ٩,٢٢٣,٣٧٢,٠٣٦,٨٥٤,٧٧٥,٨٠٨ - حتى ٩,٢٢٣,٣٧٢,٠٣٦,٨٥٤,٧٧٥,٨٠٧	System.Int64	long	٨
bits ٦٤	من ٠ حتى ١٨,٤٤٦,٧٤٤,٠٧٣,٧٠٩,٥٥١,٦١٥	System.UInt64	ulong	٩
bits ١٦	من U+0000 حتى U+ffff	System.Char	char	١٠
bits ٣٢	من ١.٣٨*٣,٤ - حتى ١.٣٨*٣,٤ +	System.Single	float	١١
bits ٦٤	من ١.٣٠٨*١,٧± حتى ٣٢٤-١.٠*٥,٠±	System.Double	double	١٢
128 bits	(-7.9*1028 to 7.9*1028)/(100 to 28)	System.Decimal	decimal	١٣
	حسب حدود الذاكرة	System.String	string	١٤
	يمكن تخزين بيانات من أي نوع ضمن المتغيرات من النوع object	System.Object	object	١٥

الأنواع من ٢ حتى ٩ هي أنواع صحيحة لا تقبل أعدادًا ذات فاصلة عشرية. أما الأنواع من ١١ حتى ١٣ فهي أنواع تقبل أعداد ذات فاصلة عشرية وتختلف فيما بينها في مجالات الأعداد التي تقبلها ودقة تلك الأعداد بالنسبة لعدد الخانات على يمين الفاصلة العشرية.

النوع char مخصص للمتغيرات التي تسمح بتخزين حرف character واحد فقط، وهو نوع يدعم ترميز Unicode، يُعتبر أي حرف موضوع ضمن علامتي اقتباس مفردتين مثل 'a' من نوع char. في الحقيقة أنّ النوع string يُعبّر عن سلسلة من المحارف من نوع char.

النوع object هو الأب العام لجميع الأنواع في إطار العمل دوت نت ومنه تنحدر جميع الأنواع الأخرى مهما كانت، سنصادفه في هذه السلسلة مرّة أخرى.

٢. العوامل Operators

تدعم سي شارب نوعين من العوامل بشكل أساسي: عوامل أحادية unary operators وعوامل ثنائية binary operators. سنتحدّث في هذا الدرس عن أكثر العوامل استخدامًا في سي شارب.

العوامل الأحادية

لهذه العوامل أسبقية أعلى في الحساب من العوامل الثنائية، وهي تشمل العديد من العوامل يلخّص الجدول التالي أهمّها، انظر إلى الأمثلة العملية التي ستأتي بعد الجدول لمعرفة كيفية استخدامها:

العامل	الوصف	الاستخدام
!	عامل النفي المنطقي وهو عامل يُطبّق على القيم المنطقية من النوع bool.	!x
~	عامل المتمم الثنائي bitwise complement وهو عبارة عن عامل نفي ولكن على مستوى	~x

	البيّات bits.	
<p>++x عامل زيادة بادئ.</p> <p>x++ عامل زيادة لاحق.</p>	<p>لهذا العامل شكلان يعمل كلّ منها على زيادة قيمة متغيّر عددي بمقدار ١، ويختلفان فقط في توقيت هذه الزيادة.</p>	++
<p>--x عامل إنقاص بادئ.</p> <p>x-- عامل إنقاص لاحق.</p>	<p>لهذا العامل شكلان أيضًا، يعمل كلّ منها على إنقاص قيمة متغيّر عددي بمقدار ١، ويختلفان فقط في توقيت هذا الإنقاص.</p>	--
<p>طريقة استخدامه هو في وضع النوع المراد التحويل إليه</p> <p>بين قوسين ونضعها جميعًا قبل القيمة التي نريد تحويلها</p> <p>مثل int(x) لتحويل قيمة x إلى قيمة من النوع int.</p>	<p>وهو عامل التحويل بين الأنواع casting.</p> <p>وهو عامل مهم جدًا سنصادفه مرارًا في هذه السلسلة.</p> <p>يمكن استبدال الحرف T باسم أيّ نوع يخطر على بالك</p> <p>مثل int و double و string وغيرها.</p>	(T)

عاملي الزيادة والإنقاص

شغل برنامج Visual Studio 2015 Community وأنشئ مشروعًا جديدًا من النوع Console Application سمّه UnaryOperatorsTest1 ثم استبدل محتويات الملف Program.cs بالشفيرة التالية:

```

1 using System;
2
3

```

```

4 namespace UnaryOperatorsTest
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10            int i = 1;
11
12            Console.WriteLine("Using of pre-increment operator
13 (++i):");
14            Console.WriteLine("Current value of i is {0}, and
15 after applying ++i, the value of i becomes {1}", i, ++i);
16            Console.WriteLine(new string('-', 40));
17
18            Console.WriteLine();
19            i = 1;
20
21            Console.WriteLine("Using of post-increment
22 operator (i++):");
23            Console.WriteLine("Current value of i is {0}, and
24 after applying i++, the value of i becomes {1}", i, i++);
25            Console.WriteLine(new string('-', 40));
26        }
27    }
28 }

```

نفذ البرنامج باستخدام (Ctrl+F5 أو من القائمة **Debug > Start Without Debugging** ستحصل على الخرج التالي:

```

sing of pre-increment operator (++i):
Current value of i is 1, and after applying ++i, the value of i
becomes 2
-----

Using of post-increment operator (i++):
Current value of i is 1, and after applying i++, the value of i
becomes 1

```

يوضّح هذا البرنامج البسيط استخدام عامل الزيادة البادئ وعامل الزيادة اللاحق. يبدأ البرنامج بالتصريح عن المتغيّر `i` من النوع `int` وإسناد القيمة ١ إليه. تعمل العبارة في السطر ١٣ على إظهار قيمتين، الأولى هي القيمة الحالية للمتغيّر `i` وتساوي ١، والقيمة الثانية هي قيمة المتغيّر `i` مضافاً إليها ١ باستخدام عامل الزيادة البادئ `++i` أي هي القيمة ٢، إذاً يقوم هذا العامل بزيادة قيمة المتغيّر `i` بمقدار ١ قبل تمرير القيمة النهائية إلى التابع `WriteLine` لذلك نحصل على الخرج:

```
Current value of i is 1, and after applying ++i, the value of i becomes 2
```

ولكن على النقيض من ذلك، نلاحظ أنّ العبارة الموجودة في السطر ٢٠ تعمل على إظهار قيمتين أيضاً، الأولى هي القيمة الحالية للمتغيّر `i` وتساوي ١ (أعدنا إسناد القيمة ١ للمتغيّر `i` في السطر ١٧)، والقيمة الثانية هي قيمة المتغيّر `i` مضافاً إليها ١ باستخدام الزيادة اللاحق `i++` ولكن لن تمرّر القيمة ٢ هذه المرّة إلى التابع `WriteLine`. والسبب في ذلك أنّ البرنامج سيعمل على تمرير قيمة `i` الأصلية (القيمة ١) ثم يطبّق بعد ذلك عامل الزيادة اللاحق. وهذا هو سبب الحصول على الخرج التالي:

```
Current value of i is 1, and after applying i++, the value of i becomes 1
```

لعلّ هذا السلوك يُسبّب بعض الإرباك للمبرمجين الجدد في سي شارب، وعلى أيّة حال أنصح بتجنّب تمرير القيمة إلى التوابع عمومًا بهذا الأسلوب. إذا احتجت لزيادة (أو إنقاص) قيمة متغيّر ما قبل تمرير لأحد التوابع فاعمل على ذلك ضمن سطر منفصل قبل استدعاء هذا التابع وأرح نفسك. في الحقيقة يُطبّق نفس المفهوم السابق بالنسبة لعاملَيّ الإنقاص البادئ والإنقاص اللاحق.

ملاحظة: انظر إلى طريقة التنسيق الجديدة التي استخدمتها في السطر ١٣:

```
Console.WriteLine("Current value of i is {0}, and after applying ++i, the value of i becomes {1}", i, ++i);
```

مررت إلى التابع WriteLine ثلاثة وسائط: الأول هو النص التنسيقي وقد حُجز ضمنه مكانين مخصّصين لقيمتين سأمررهما لاحقاً لهذا التابع، هذان المكانان على الشكل {0} و {1}. الوسيط الثاني هو المتغيّر i، والوسيط الثالث هو ++i. سيعمل البرنامج على وضع القيمة الممرّرة للتابع WriteLine والتي تلي النص التنسيقي مباشرةً (في حالتنا هذه قيمة i) في المكان {0}، أمّا المكان {1} فسيُوضع ضمنه القيمة التالية وهي ++i. ينطبق نفس الكلام تمامًا على العبارة الموجودة في السطر ٢٠.

كما يحتوي السطران ١٤ و ٢١ على أسلوب جميل لطباعة سطر فاصل في خرج البرنامج بغرض توضيحه. أنشأنا كائنًا من النوع string باستخدام العامل new ومررنا لبانيته وسيطين: الأول المحرف '-' من نوع char والثاني القيمة ٤٠:

```
new string('-', 40)
```

سيؤد ذلك نصًا يحتوي على ٤٠ محرف '-' مكرّر (لاحظ علامتي الاقتباس المفردتين ')، يمرر هذا النص بعد ذلك إلى التابع WriteLine. لا تقلق إن بدا هذا الكلام غير مفهومًا الآن، فسنحدّث عن الكائنات فيما بعد.

فهم عامل النفي المنطقي وعامل التحويل بين الأنواع

أنشئ مشروعًا جديدًا من النوع Console Application سمّه UnaryOperatorsTest2 ثم استبدل محتويات الملف Program.cs بالشيفرة التالية:

```
1 using System;
2
3
4 namespace UnaryOperatorsTest2
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             bool b = true;
11             double d = 8.9;
12             int i;
13
14             Console.WriteLine("b = {0}, !b = {1}", b, !b);
```

```

15
16         i = (int)d;
17         Console.WriteLine("d = {0}, after applying casting
to (int), i = {1}", d, i);
18     }
19 }
20 }

```

نفذ البرنامج باستخدام Ctrl+F5 لتحصل على الخرج التالي:

```

b = True, !b = False

d = 8.9, after applying casting to (int), i = 8

```

استخدمنا في هذا البرنامج المتغير `b` من النوع `bool` وهو نوع منطقيّ تحمل المتغيرات المصرّح عنها بواسطة إحدى قيمتين `true` أو `false`. أسندنا للمتغير `b` القيمة `true` عند التصريح عنه في السطر ١٠، ثمّ عرضنا للمستخدم قيمة `b` الأصلية وقيمته بعد تطبيق عامل النفي المنطقي عليه `!b` لنحصل على الخرج التالي:

```

b = True, !b = False

```

يعكس هذا العامل الحالة المنطقية، فإذا كانت `true` تصبح `false`، أمّا إذا كانت `false` فتصبح `true`. ولكن إذا لاحظت أنّ الخرج يُظهر القيمتين المنطقيتين `true` و `false` بحرفين كبيرين في بداية كل منهما: `True` و `False`. السبب في ذلك أن التابع `WriteLine` في السطر ١٤ يعمل بشكل ضمني على استدعاء التابع `ToString` لكل من الوسيطين المرّرين له، أي الوسيطين `b` و `!b` فيحصل بذلك على التمثيل النصّي للقيمة المنطقية الموجودة في كلّ منهما، والذي يبدأ بحرف طباعي كبير. جرّب استبدال العبارة البرمجية في السطر ١٤ بالعبارة التالية:

```

Console.WriteLine("b = {0}, !b = {1}", b.ToString(),
(!b).ToString());

```

التعديل الذي أجريناه في السطر السابق هو استدعاء التابع `ToString` بشكل صريح لكلّ وسيط قبل تمريره إلى التابع `WriteLine`. ستحصل بذلك على نفس الخرج دون أيّ تغيير.

بالنسبة لعملية التحويل بين الأنواع فقد أجريناها بين المتغير d من النوع double (السطر ١١) والمتغير i من النوع int (السطر ١٢)، حيث سنحوّل القيمة ذات الفاصلة العشرية ٨,٩ الموجودة في d إلى قيمة صحيحة بدون فاصلة ونخزنها ضمن i. تجري عملية التحويل هذه في السطر ١٦ على الشكل التالي:

```
i = (int)d;
```

لاحظ أنّ القوسين المحيطين بـ int ضروريين. إذا حاولت إزالة عامل التحويل (int) من العبارة السابقة وحاولت تنفيذ البرنامج فستحصل على الخطأ التالي:

```
CS0266 Cannot implicitly convert type 'double' to 'int'. An explicit conversion exists (are you missing a cast?)
```

يُشير هذا الخطأ إلى عدم إمكانية إسناد قيمة متغير من النوع double إلى متغير من النوع int مباشرةً بدون تحويل لأنّ ذلك سيؤدّي إلى ضياع في البيانات (ستضيع القيمة ٩,٠). يقترح عليك هذا الخطأ استخدام التحويل بين الأنواع cast في الجزء الأخير من الرسالة. أعد وضع عامل التحويل (int) أمام المتغير d ونفّذ البرنامج لتحصل في الخرج على ما يلي:

```
d = 8.9, after applying casting to (int), i = 8
```

انظر كيف أصبحت قيمة i تساوي ٨. في الواقع سيصادفنا عامل التحويل كثيرًا في هذه السلسلة.

العوامل الثنائية

تشتمل هذه العوامل على معظم العوامل الموجودة في سي شارب ولها العديد من الأصناف، تحتاج هذه العوامل إلى وجود مُعاملين operands على طرفيها لكل تعمل، يلخص الجدول التالي أهم هذه العوامل مع التصنيف الذي تقع ضمنه.

العامل	الوصف	الاستخدام	التصنيف
+	عملية الجمع العددي	$x + y$	عوامل
-	عملية الطرح العددي	$x - y$	حسابية
*	عملية الضرب العددي	$x * y$	
/	عملية القسمة العددية (إذا كان كل من المعاملين من نوع صحيح فسيكون ناتج القسمة صحيحًا بدون فاصلة، حيث تُهمل الأجزاء العشرية في حال وجودها).	x / y	
%	عملية باقي القسمة	$x \% y$	
<	عامل اختبار "أصغر من" يُرجع القيمة true إذا كان المُعامل الأيسر أصغر من الأيمن، وإلا يُرجع false.	$y > x$	عوامل مقارنة
>	عامل اختبار "أكبر من" يُرجع القيمة true إذا كان المُعامل الأيسر أكبر من الأيمن، وإلا يُرجع false.	$y < x$	
<=	عامل اختبار "أصغر من أو يساوي"	$y => x$	

		يُرجع القيمة true إذا كان المُعامل الأيسر أصغر من أو يساوي الأيمن، وإلا يُرجع false.	
	$y \leq x$	عامل اختبار "أكبر من أو يساوي" يُرجع القيمة true إذا كان المُعامل الأيسر أكبر من أو يساوي الأيمن، وإلا يُرجع false.	\geq
عوامل اختبار المساواة	$x == y$	عامل اختبار "المساواة" بين قيمتين، يُرجع true إذا كانت القيمتين متساويتين وإلا يُرجع false.	$==$
	$x != y$	عامل اختبار "عدم المساواة" بين قيمتين، يُرجع true إذا كانت القيمتين غير متساويتين وإلا يُرجع false.	$!=$
العوامل	$\&\& x$ y	تطبيق منطوق AND على قيمتين (أو تعبيرين) منطقيين.	$\&\&$
الشرطية	$x \ \ y$	تطبيق منطوق OR على قيمتين (أو تعبيرين) منطقيين.	$\ \ $
عوامل إسناد	$x = y$	عامل الإسناد للقيمة (أو التعبير) الموجودة في اليمين إلى المتغير الموجود في اليسار.	$=$
	$x += y$	عامل الجمع ثم الإسناد.	$+=$
	$x -= y$	عامل الطرح ثم الإسناد.	$-=$
	$x *= y$	عامل الضرب ثم الإسناد.	$*=$
	$x /= y$	عامل القسمة ثم الإسناد.	$/=$

	x %= y	عامل باقي القسمة ثم الإسناد.	%=
--	--------	------------------------------	----

العوامل الحسابية

تُعتبر هذه العوامل بسيطة وواضحة وهي مشتركة بين جميع لغات البرمجة. على أيّة حال إليك برنامجًا بسيطًا يتعامل معها ويوضّح وظائفها.

```

1  using System;
2
3  namespace ArithmeticOperators
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              int x, y;
10             string str_x, str_y;
11
12             //input operands.
13             Console.WriteLine("Input left operand (x) : ");
14             str_x = Console.ReadLine();
15
16             Console.WriteLine("Input right operand (y) : ");
17             str_y = Console.ReadLine();
18
19             //convert each operand to integer representative.
20             x = int.Parse(str_x);
21             y = int.Parse(str_y);
22
23
24             Console.WriteLine();
25
26             //perform arithmetic calculations and display
27             results.
28             Console.WriteLine("x + y = {0}", x + y);
29             Console.WriteLine("x - y = {0}", x - y);
30             Console.WriteLine("x * y = {0}", x * y);

```

```

30         Console.WriteLine("x / y = {0}", x / y);
31         Console.WriteLine("x % y = {0}", x % y);
32
33     }
34 }
35 }

```

نفذ البرنامج باستخدام `Ctrl+F5` سيطلب منك البرنامج إدخال المُعامل الأيسر `left operand` ، ثم المُعامل الأيمن `right operand` ، وبعدها ينفذ العمليات الحسابية الأربع عليهما. جرّب إدخال القيمتين ٩ و ٢ على الترتيب لتحصل على الخرج التالي:

```

Input left operand (x) : 9
Input right operand (y) : 2

x + y = 11
x - y = 7
x * y = 18
x / y = 4
x % y = 1

```

العمليات الثلاث الأولى واضحة. بالنسبة لعملية القسمة يجب أن يكون الناتج ٤,٥، ولكن بما أنّ عملية القسمة تجري بين قيمتين صحيحتين فإنّ النتيجة يجب أن تكون صحيحة، وبالتالي يُهمل الجزء العشري ٥,٠ ويكون الناتج ٤ فقط. بالنسبة لعملية باقي القسمة `x % y` فإنّ النتيجة ١ هي باقي قسمة ٩ على ٢.

ملاحظة: إذا لم ترغب بحذف الجزء العشري من ناتج عملية القسمة الصحيحة ودون أن تغيّر أنواع المتغيّرات، يمكنك استخدام عامل التحويل بين الأنواع (T). استبدال العبارة الموجودة في السطر ٣٠ بالعبارة التالية:

```

Console.WriteLine("x / y = {0}", x / (double)y);

```

وضعت عامل التحويل (double) قبل المتغيّر `y` لتحويل قيمته العددية إلى قيمة من نوع `double` (دون المسّ بقيمة الأصلية بالطبع)، فعندما يرى البرنامج أنّه يُجري عملية القسمة بين قيمة صحيحة (قيمة `x`) وقيمة من النوع `double` فسيعطي الناتج على شكل قيمة من

نوع double تُمرّر بدورها إلى التابع WriteLine ليعرض القيمة ٤,٥ بدلاً من ٤. ويمكن فعل نفس الأمر مع المتغيّر x بدلاً من y إذا أحببت.

عوامل المقارنة

سنتناول عوامل المقارنة < و > و <= و >= و == و != في البرنامج التالي:

```
1  using System;
2
3
4  namespace RelationalOperators
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             int x, y;
11             string str_x, str_y;
12
13
14             //input operands.
15             Console.Write("Input left operand : ");
16             str_x = Console.ReadLine();
17
18             Console.Write("Input right operand : ");
19             str_y = Console.ReadLine();
20
21             //convert each operand to integer representative.
22             x = int.Parse(str_x);
23             y = int.Parse(str_y);
24
25             Console.WriteLine();
26
27             //perform comparing operations and display
28             Console.WriteLine("{0} == {1} evaluates to {2}",
29             x, y, x == y);
30             Console.WriteLine("{0} != {1} evaluates to {2}",
31             x, y, x != y);
```

```

30         Console.WriteLine("{0} > {1} evaluates to {2}", x,
y, x > y);
31         Console.WriteLine("{0} >= {1} evaluates to {2}",
x, y, x >= y);
32         Console.WriteLine("{0} < {1} evaluates to {2}", x,
y, x < y);
33         Console.WriteLine("{0} <= {1} evaluates to {2}",
x, y, x <= y);
34     }
35 }
36 }

```

نفذ البرنامج وأدخل القيمتين ٣ و ٤ على الترتيب لتحصل على الخرج التالي:

```

Input left operand : 3
Input right operand : 4

3 == 4 evaluates to False
3 != 4 evaluates to True
3 > 4 evaluates to False
3 >= 4 evaluates to False
3 < 4 evaluates to True
3 <= 4 evaluates to True

```

تكون نتيجة تنفيذ عوامل المقارنة قيمة منطقية true أو false. جرب إدخال قيم متنوعة، كما جرب إدخال قيمتين متساويتين وانظر إلى الخرج.

العوامل الشرطية

العاملين الشرطيين &&(AND) و ||(OR) هما عاملان مهمّان جدًّا ويستخدمان بكثرة في بنى القرار في سي شارب. ولهما وجود في جميع لغات البرمجة. يوضّح البرنامج التالي استخدام هذين العاملين بصورة مبسّطة.

```
1  using System;
2
3
4  namespace RelationalOperators
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             int a, b, c, d;
11             bool and_operator, or_operator;
12
13             a = 1;
14             b = 2;
15             c = 5;
16             d = 9;
17
18             and_operator = (a > b) && (c <= d);
19             Console.WriteLine("{0} > {1}) && ({2} <= {3})
evaluates to {4}", a, b, c, d, and_operator);
20
21             or_operator = (a > b) || (c <= d);
22             Console.WriteLine("{0} > {1}) || ({2} <= {3})
evaluates to {4}", a, b, c, d, or_operator);
23         }
24     }
25 }
```

لا نستخدم العوامل الشرطية بهذا الأسلوب في البرامج الحقيقية، ولكنّ هذا الأسلوب مفيد في توضيح آلية عمل العوامل الشرطية وتفاعلها مع عوامل المقارنة. نفّذ البرنامج لتحصل على الخرج التالي:

```
(1 > 2) && (5 <= 9) evaluates to False
```

```
(1 > 2) || (5 <= 9) evaluates to True
```

تفسير هذا الخرج يسير للغاية. لنبدأ بالسطر الأول، نتيجة حساب التعبير الأول هو false:

```
(1 > 2) && (5 <= 9)
```

وسبب ذلك هو أنّ نتيجة التعبير (1 > 2) هو false، أمّا نتيجة حساب (5 <= 9) هو true وبالتالي سيعمل العامل الشرطي && بالنتيجة على حساب التعبير && false true والذي يعطي بكلّ تأكيد القيمة المنطقية false. بالنسبة للسطر الثاني، وهو التعبير:

```
(1 > 2) || (5 <= 9)
```

والذي يعطي true. والسبب هو أنّ العامل الشرطي || سيعمل على حساب التعبير false || true والذي يعطي القيمة المنطقية true.

لاحظ استخدام الأقواس على أطراف عوامل المقارنة، يمكن الاستغناء عنها، ولكن لا أنصح بذلك، استخدم الأقواس دومًا حتى ولو لم يكن استخدامها ضروريًا لتوضيح منطق البرنامج، ولكن استخدمها بحكمة. السبب في انتفاء الحاجة إلى استخدام الأقواس في هذا البرنامج، هو أنّ عوامل المقارنة لها أسبقية تنفيذ أعلى من العوامل الشرطية، لذلك فهي تُقيم قبل تقييم العوامل الشرطية.

عوامل الإسناد

استخدمنا حتى الآن عامل الإسناد (=). توجد عوامل إسناد أخرى تُسهّل البرمجة في سي شارب وهي += و -= و *= و /= و %= . الأمر بسيط، بالنسبة لعامل الإسناد += يمكن توضيح عمله بالشفيرة التالية:

```
int x = 3;

x += 5;
```

بعد تنفيذ الشيفرة السابقة ستصبح قيمة x تساوي ٨. لأنّ العبارة $x += 5$ تكافئ تمامًا العبارة $x = x + 5$ ويمكننا استبدالها بها. يُطبّق نفس الأسلوب تمامًا على العوامل الباقية. فمثلًا انظر إلى الشيفرة التالية:

```
int x = 21;

x /= 4;

x %= 3;
```

هل تستطيع تخمين قيمة x بعد تنفيذ هذه الشيفرة؟ إذا كانت النتيجة ٢ فقد أصبت. السبب في ذلك بسيط. فقد بدأنا بقيمة x تساوي ٢١ ثم نفذنا العبارة $x /= 4$ التي تكافئ العبارة $x = x / 4$ وهي قسمة صحيحة، لذلك سيحمل x القيمة ٥ (بدون فاصلة عشرية). بعد تنفيذ العبارة الأخيرة $x %= 3$ التي تكافئ العبارة $x = x \% 3$ ستصبح قيمة x تساوي ٢ لأنّ باقى قسمة ٥ على ٣ يساوي ٢. وهذا كلّ ما في الأمر.

تمارين

تمرين ١

حاول تخمين القيمة المنطقية التي ستُطبع على الشاشة باستخدام القلم والورقة فقط:

```
int a = 30;

a /= 3;

a %= 3;

Console.WriteLine(a == 1);
```

تمرين ٢

حاول تخمين قيمة f التي ستُطبع على الشاشة باستخدام القلم والورقة فقط:

```
int x;  
double f;  
  
x = 9;  
f = (double)x / 2;  
f *= 10;  
  
Console.WriteLine("f = {0}", f);
```

الفصل الثالث

الجملة الشرطية

الجمل الشرطية

العبارات الشرطية في البرنامج من الأمور الأساسية في البرمجة كما هو معلوم. تمتلك لغة سي شارب نوعين من العبارات الشرطية وهما: بنية if-else وبنية switch-case.

العبارة الشرطية if-else

وهي بنية مألوفة في معظم لغات البرمجة، تشبه هذه البنية في تشكيلها تلك الموجودة في لغات أخرى مثل C++ و Java. تمتلك هذه البنية ثلاثة أشكال سنتحدث عنها تباعاً.

شكل الأول لبنية if

الشكل الأبسط لبنية if هي:

```
if ([condition])
{
    statement1;
    statement2;
    ...
}
```

إذا كان تقييم evaluate الشرط [condition] يعطينا true (أي تحقق الشرط) عندها سننفذ العبارات البرمجية الموجودة ضمن الحاضنة {}, وإلا (أي لم يتحقق الشرط) فلن يُنفذ أي منها. أنشئ مشروعاً جديداً سمّه Lesson03_1 واستبدل محتويات الملف Program.cs بالبرنامج البسيط التالي الذي يعمل على مقارنة القيمة المدخلة من المستخدم مع العدد ٥ ويظهر الخرج المناسب:

```
1 using System;
2
3 namespace Lesson03_1
4 {
5     class Program
6     {
7         static void Main(string[] args)
```

```

8      {
9          double x;
10         string str_x;
11
12         Console.WriteLine("Input a number: ");
13         str_x = Console.ReadLine();
14
15         x = double.Parse(str_x);
16
17         if(x > 5)
18         {
19             Console.WriteLine("The value {0} is greater
than 5", x);
20         }
21
22         Console.WriteLine("Goodbye!");
23     }
24 }
25 }

```

جرّب تنفيذ هذا البرنامج باستخدام (Ctrl+F5 أو من القائمة **Debug > Start Without Debugging**) سيطلب منك البرنامج إدخال قيمة عددية، أدخل العدد 6، سيعرض البرنامج

الخرج التالي:

```
The value 6 is greater than 5
```

```
Goodbye!
```

أعد تنفيذ البرنامج وأدخل هذه المرة القيمة 3 لتحصل على الخرج التالي:

```
Goodbye!
```

لاحظ بأن خرج البرنامج قد اختلف باختلاف القيم المدخلة، أي أنّ هناك اختلاف في العبارات البرمجية التي تمّ تنفيذها في كلّ مرّة. يعود سبب ذلك إلى البنية if الموجودة بين السطرين ١٧ و ٢٠. يختبر الشرط الموجود بعد كلمة if في السطر ١٧ فيما إذا كانت قيمة المتغيّر x أكبر تمامًا من ٥. فإذا كانت نتيجة تقييم التعبير $x < ٥$ تساوي true فهذا يعني أنّ الشرط قد تحقّق وبالتالي تنفّذ جميع العبارات البرمجية الموجودة في الحاضنة (بين السطرين ١٨ و ٢٠). أمّا إذا كانت نتيجة تقييم التعبير $x < ٥$ تساوي false فعندها سيتجاوز تنفيذ البرنامج البنية if إلى العبارات التي تأتي بعد السطر ٢٠.

الشكل الثاني لبنية if

هذا الشكل للعبارة الشرطية if مفيد أيضًا، ويُستخدم عندما نريد الاختيار بين مجموعتين من العبارات البرمجية، والشكل العام له:

```

if ([condition])
{
    statement1;
    statement2;
    ...
}
else
{
    Statement3;
    Statement4;
    ...
}

```

لقد أضفنا القسم else مع حاضنته. المنطق هنا بسيط يمكننا قراءته بالشكل التالي:

"إذا تحقّق الشرط [condition] عندها تنفّذ الحاضنة الموجودة بعد if مباشرةً، وإلاّ يتمّ تنفيذ الحاضنة الموجودة بعد else مباشرةً"

لكي نتعرّف على كيفية التعامل مع هذا الشكل، أنشئ مشروعًا جديدًا سمّه Lesson03_2 وانسخ الشيفرة التالية إلى الملف Program.cs:

```

1  using System;
2
3  namespace Lesson03_2
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              double x;
10             string str_x;
11
12             Console.Write("Input a number: ");
13             str_x = Console.ReadLine();
14
15             x = double.Parse(str_x);
16
17             if (x > 5)
18             {
19                 Console.WriteLine("The value {0} is greater
20 than 5", x);
21             }
22             else
23             {
24                 Console.WriteLine("The value {0} is smaller
25 than or equals 5", x);
26             }
27             Console.WriteLine("Goodbye!");
28         }
29 }

```

هذا البرنامج مطابق للبرنامج الذي رأيناه قبل قليل باستثناء القسم else مع حاضنته. يسلك هذا البرنامج نفس السلوك الذي يسلكه البرنامج السابق باستثناء أنه لو أدخل المستخدم قيمة مثل ٣ سيعمل البرنامج على طباعة الخرج التالي:

```
The value 3 is smaller than or equals 5
```

Goodbye!

لاحظ أنّ البرنامج Lesson03_1 كان يطبع العبارة Goodbye! فقط عند إدخال القيمة ٣. السبب في ظهور الخرج الجديد هو وجود القسم else في بنية if السابقة، فعندما يُقِيم الشرط $x < ٥$ في السطر ١٧ وتكون نتيجة تقييمه false سينتقل البرنامج فوراً إلى تنفيذ العبارات البرمجية الموجودة ضمن حاضنة else وهذا هو سبب ظهور هذا الخرج.

العيب الوحيد في هذا البرنامج أنّه لا يستطيع التمييز بين الحالة التي تكون فيها القيمة المدخلة تساوي ٥ وبين الحالة التي تكون فيها أصغر تماماً من ٥، ففي كلّ من هاتين الحالتين يعرض البرنامج نفس الخرج عن طريق تنفيذ العبارة الموجودة في السطر ٢٣.

ملاحظة: في حال كانت أيّة حاضنة تحوي عبارة برمجية واحد فقط، فعندها يمكن عدم استخدام قوسي الحاضنة {} مع أنني أفضل استخدامها لجعل البرنامج أكثر وضوحاً.

الشكل الثالث لبنية if

وهو الشكل الأكثر شمولاً وفيه نستخدم القسم else if على الصورة التالية:

```
if ([condition])
{
    statement1;
    statement2;
    ...
}
else if ([condition1])
{
    Statement3;
    Statement4;
    ...
}
else if ([condition2])
{
    Statement3;
```

```

        Statement4;
        ...
    }
    ...
else
{
    Statement3;
    Statement4;
    ...
}

```

يمكننا قراءة المنطق هنا على الشكل التالي:

"إذا تحقّق الشرط [condition] عندها تنفّذ الحاضنة الموجودة بعد if مباشرةً، وإلا إذا (else if) تحقّق الشرط [condition1] يتم تنفيذ الحاضنة الموجودة بعد else if الأولى مباشرةً، وإلا إذا تحقّق الشرط [condition2] يتم تنفيذ الحاضنة الموجودة بعد else if الثانية مباشرةً، وإلا (else) يتم تنفيذ الحاضنة الموجودة بعد else مباشرةً"

نلاحظ أنه يمكننا استخدام أقسام else if بقدر ما نريد، ولكن يمكن استخدام قسم else وحيد. ونلاحظ أيضاً أنه بالنتيجة ستنفّذ مجموعة واحدة فقط ضمن حاضنة ما. وواضح أيضاً أن أقسام else if و else هي أقسام اختيارية ووجودها غير مرتبط ببعضها، ولكن إذا حوت بنية if قسم else if فيجب أي يكون القسم else (في حال وجوده) هو القسم الأخير.

لكي نتبّت هذا المفهوم بشكل جيّد انظر البرنامج Lesson03_3 التالي:

```

1  using System;
2
3  namespace Lesson03_3
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              double x;

```

```

10         string str_x;
11
12         Console.WriteLine("Input a number: ");
13         str_x = Console.ReadLine();
14
15         x = double.Parse(str_x);
16
17         if (x > 5)
18         {
19             Console.WriteLine("The value {0} is greater
than 5", x);
20         }
21         else if (x == 5)
22         {
23             Console.WriteLine("The value {0} is equals 5",
x);
24         }
25         else
26         {
27             Console.WriteLine("The value {0} is smaller
than 5", x);
28         }
29
30         Console.WriteLine("Goodbye!");
31     }
32 }
33 }

```

يشبه هذا البرنامج سابقه إلى حدٍ بعيد، فهو يقارن القيمة المدخلة مع العدد ٥ ويعرض رسالة مناسبة نتيجة عملية المقارنة. الشيء الجديد هنا هو التمييز بين الحالة التي تكون فيها القيمة المدخلة تساوي العدد ٥ والحالة التي تكون فيها أصغر من العدد ٥. قمنا بذلك من خلال إضافة القسم `else if` جديد يختبر حالة المساواة مع العدد ٥. الآن أصبح منطق البرنامج كالتالي:

"إذا كانت القيمة المدخلة أكبر تمامًا من ٥ (السطر ١٧) عندها تُنفَّذ العبارة الموجودة في السطر ١٩، وإلا إذا كانت القيمة المدخلة تساوي ٥ (السطر ٢١) عندها تُنفَّذ العبارة الموجودة في السطر ٢٣، وإلا ستكون القيمة المدخلة أصغر من ٥ حتمًا، وتُنفَّذ العبارة الموجودة في السطر ٢٧".

العبارة الشرطية switch-case

تفيد هذه البنية في الاختيار من بين عدّة حالات منفصلة. لهذه البنية الشكل العام التالي:

```
switch (expression)
{
    case [A]:
        [statements]
        break;

    case [B]:
        [statements]
        break;

    ...

    [default:]
        [statements]
        break;
}
```

القسم الأخير default هو قسم اختياري، كما يجب أن يكون هناك قسم case واحد على الأقل. إليك الآن البرنامج Lesson03_4 لفهم كيفية استخدام هذه البنية:

```
1 using System;
2
3 namespace Lesson03_4
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
```

```

9         double x, y;
10        string str_x, str_y, operation;
11
12        Console.WriteLine("Input first number: ");
13        str_x = Console.ReadLine();
14
15        Console.WriteLine("Input second number: ");
16        str_y = Console.ReadLine();
17
18        Console.WriteLine("Choose operation (+, -, *, /): ");
19        operation = Console.ReadLine();
20
21        x = double.Parse(str_x);
22        y = double.Parse(str_y);
23
24        switch (operation)
25        {
26            case "+":
27                Console.WriteLine("{0} + {1} = {2}", x, y,
x + y);
28                break;
29            case "-":
30                Console.WriteLine("{0} - {1} = {2}", x, y,
x - y);
31                break;
32            case "*":
33                Console.WriteLine("{0} * {1} = {2}", x, y,
x * y);
34                break;
35            case "/":
36                Console.WriteLine("{0} / {1} = {2}", x, y,
x / y);
37                break;
38            default:
39                Console.WriteLine("Unsupported
operation.");
40                break;
41        }

```

```
42     }
43     }
44 }
```

البرنامج السابق عبارة عن برنامج آلة حاسبة بسيطة تدعم العمليات الحسابية الأربعة: الجمع والطرح والضرب والقسمة. يطلب البرنامج من المستخدم إدخال قيمتين عدديتين، بعد ذلك يطلب اختيار العملية الحسابية المراد إجراؤها على هاتين القيمتين (+ ، - ، * ، /) وتخزين العملية المختارة ضمن المتغير النصي operation وذلك في السطر ١٩. تعمل البنية switch في السطر ٢٤ على مقارنة قيمة المتغير النصي operation مع القيم الموجودة في أقسام case (الأسطر ٢٦ و ٢٩ و ٣٢ و ٣٥) فإذا تطابقت القيمة الموجودة في operation إحدى تلك القيم، فإن العبارات البرمجية الموجودة ضمن هذا القسم سيتم تنفيذها. أما إذا لم يحدث مثل هذا التطابق، فستنفذ العبارات البرمجية الموجودة في القسم الاختياري default والتي ستخبر المستخدم (في هذا المثال) بأن العملية الحسابية التي يرغبها لا يدعمها البرنامج.

نستفيد من القسم default في تنفيذ عبارات برمجية في حال لم يحدث التطابق مع أي قسم case سابق. كما نلاحظ أن العبارة break الموجودة في كل قسم من أقسام case بالإضافة إلى قسم default هي عبارة ضرورية وتؤدي إلى انتقال تنفيذ البرنامج إلى خارج بنية switch أي إلى السطر ٤٢.

جرب تنفيذ البرنامج وإدخال قيم متنوعة بالإضافة إلى تجريب العمليات الحسابية الأربعة. جرب إدخال عامل باقي القسمة مثلاً (%) وانظر كيف سيجيب البرنامج بالرسالة Unsupported operation.

تمارين

تمرين ١

في البرنامج Lesson03_4 السابق إذا أدخل المستخدم القيمة ٠ للعدد الثاني، ثم اختار عملية القسمة (/) سيؤدي ذلك إلى القسمة على صفر، وهذا يسبب خطأ أثناء التنفيذ runtime

error يؤدي إلى رمي استثناء وتوقف البرنامج عن العمل. أجر تعديلًا على البرنامج ليأخذ هذا الأمر بالحسبان.

(تلميح: أضف شرط if ضمن قسم case الموافق للعملية (/) لاختبار قيمة المتغير y فيما إذا كانت تساوي الصفر أم لا).

تمرين ٢

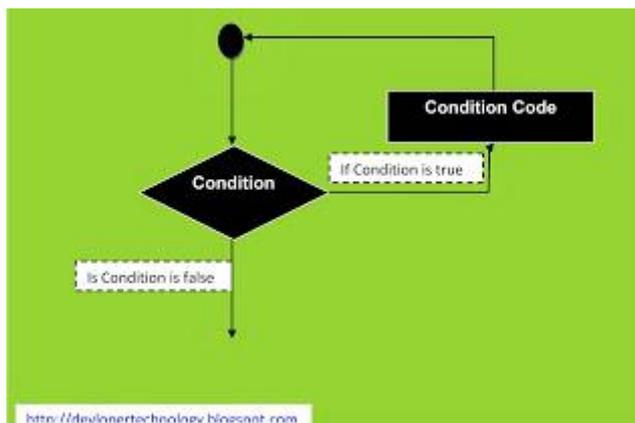
اكتب برنامجًا يطلب من المستخدم إدخال درجة الحرارة الحالية. فإذا كانت درجة الحرارة أقل من ٤ مئوية يعرض البرنامج الرسالة "Cold Very". أما إذا كانت درجة الحرارة بين ٤ وأقل من ١٠ مئوية يعرض الرسالة "Cold". وفي حال كانت درجة الحرارة بين ١٠ وأقل من ٣٠ مئوية يعرض الرسالة "Normal". أما إذا كانت درجة الحرارة ٣٠ فما فوق فيعرض البرنامج الرسالة "Hot".

الفصل الرابع

جمل التكرار

الدورة أو حلقات التكرار

Loops - C# الدورة :- ال loop عبارة عن امر برمجي يقوم بعمل تكرار او يدوار لفحص شرط ما ويخرج منه او لا ينفذ اذا كان الشرط غير صحيح. وتوجد للدورة او اللوب ال loop صيغ عديدة وكلها تؤدي نفس الناتج عند تطبيقها. فعلى سبيل المثال انك تريد طباعة الارقام من ١ الى ١٠٠ فمن الصعب ان تقوم بكتاب الارقام من ١ الى ١٠٠ كي تطبعها ولكن نستخدم الدورة فنكتب كود بسيط يطبع هذه الارقام. ويمكنك طباعة مثلا الارقم الزوجية التي بين ١ و ١٠٠ وذلك حسب الشرط وغيرها.



الحلقات التكرار التي تتعامل معها لغة سي شارب.

Loop Type نوع حلقة التكرار	Description الوصف
while loop	يتم تكرار الجملة او مجموعة الجمل - الأكواد - بينما الشرط المعطى صحيح معنا الحلقة بالعامية بينما كذا قم بتنفيذ الجملة حتى تصبح الشرط خطأ في هذا النوع من حلقات يتم فحص الشرط قبل التنفيذ بمعنى اذا كان الشرط خطأ من البداية لا ينفذ
for loop	يتم تنفيذ سلسلة من الجملة عدة مرات ويختصر تكرار
do...while loop	<u>while</u> ولكن يتم تنفيذ الكود بمعنى يطبع لمرة واحدة ثم يفحص الشرط يشبه نفس جملة التكرار في حال كان صحيح ينفذ جملة الشرط مرة <u>loop</u> اخرى حتى يصبح الشرط خطأ
nested loops	يمكنك استخدام حلقتين او اكثر من حلقات التكرار بحيث تكون حلقة داخل حلقة وهو ما يعرف بحلقات التكرار المتداخل

Loop Control Statements التحكم بجملة التكرار:-

تتيح الس شارب عوامل هروب عند استخدام حلقات التكرار فمثلا عند استخدام حلقة التكرار وبدخلها جملة شرطية وبفرض ان الجملة الشرطية تحققت فان حلقة التكرار ستستمر حتى يبطل الشرط ولذلك لا حاجة للاستمرار فنتيح اللغة التوقف او الهروب عند التحقق ومثلا ربما تدخل حلقة تكرار لانهاية اي تستمر الى ما لانهاية وهذا خطأ فيمكنك توقيفه.

اليك جمل التحكم او الهروب.

Control Statement جمل التحكم او الهروب من التكرار للانتهائي	Description الوصف
<u>break statement</u>	تمكنك من انهاء حلقة التكرار والقفز الى الجملة البرمجية مباشرة بعد حلقة التكرار
<u>continue statement</u>	تقوم هذه الجملة بتخطي تنفيذ الكود واعادة اختباره وتنفيذه لاحقة

The Infinite Loop التكرار للانتهائي أو الى مالانتهائية.

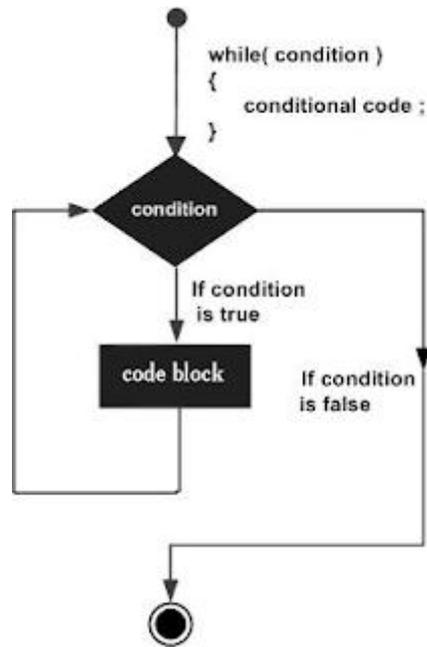
يمكن حلقة التكرار ان تتكرر بدون توقف وهذا يصبح خطأ برمجي يتسبب في استهلاك الذاكرة الرام ويحدث التكرار للانتهائي عندما الشرط لا يصبح خطأ اي يبقى صحيح . ويمكن عمل تكرار للانتهائي بترك الشرط فارغ كما في المثال التالي.

```
using System;

namespace Loops
{
    class Program
    {
        static void Main(string[] args)
        {
            for ( ; ; )
            {
                Console.WriteLine("Hey! I am Trapped");
            }
        }
    }
}
```

أساسيات البرمجة حلقة التكرار while:

حلقة التكرار while تقوم بتكرار جملة معينة طالما بان الشرط صحيح .



الصيغة العامة لجمل التكرار While

```

while(الشرط)
{
    الكود ;
}
  
```

مثال على جملة التكرار While:-

```

using System;

namespace Loops
{
    class Program
    {
        static void Main(string[] args)
        {
            /* متغيرات خاصة او محلية */
            int a = 10;

            /* حلقة التكرار */
            while (a < 20)
            {
                Console.WriteLine("value of a: {0}", a);
                a++;
            }
            Console.ReadLine();
        }
    }
}
  
```

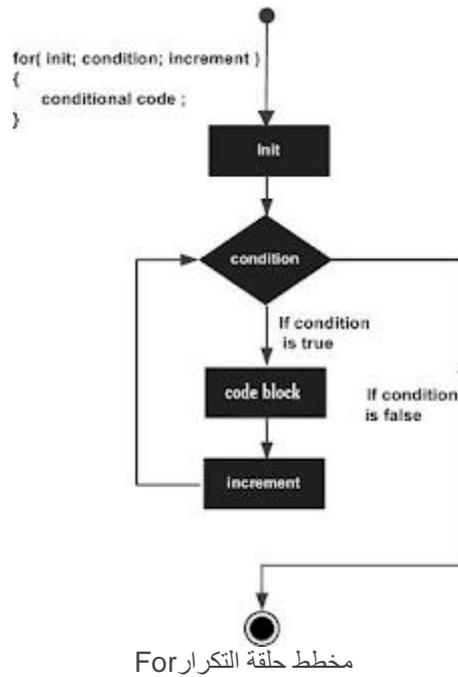
```
    }  
  }  
}
```

الناتج مثال على جملة التكرار While:-

```
value of a: 10  
value of a: 11  
value of a: 12  
value of a: 13  
value of a: 14  
value of a: 15  
value of a: 16  
value of a: 17  
value of a: 18  
value of a: 19
```

أساسيات البرمجة حلقة التكرار

For loop حلقة التكرار فور تقوم بتكرار الجملة المعطاة عدد من المرات وفقا للشرط حتي يصبح قيمته خطأ.



الصيغة العامة لحلقة التكرار for

```

for ( init; condition; increment )
{
  statement
}

```

بعض القواعد التي يجب اتباعها عند التعامل مع جملة التكرار for.

١- int هو عدد صحيح حيث يعتبر العداد الذي سيبدأ العد من عنده للتكرار ويجب ان يكون عدد صحيح.

٢- ثانيا الشرط condition اذا كان صحيح يبدأ بتنفيذ الشرط ام اذا كان خطأ لا ينفذ.

٣- بعد تنفيذ الشرط تكرر يقفز جملة التحكم بالشرط الى جملة الزيادة بالموجب او السالب . increment statement

٤- بعد تنفيذ الشرط والزيادة على العداد يتم تقييم الشرط مع القيمة الجديد التي تم الزيادة عليها وفحص الشرط مرة اخرى اذا كان صحيح ينفذ ويقوم بعملية الزيادة وهكذا حتى يصبح الشرط خطأ.

مثال على حلقة التكرار *FOR*:-

```
using System;

namespace Loops
{

class Program
{

static void Main(string[] args)
{

/*تنفيذ حلقة التكرار*/

for (int a = 10; a < 20; a = a + 1)
```

```
{  
  
Console.WriteLine("value of a: {0}", a);  
  
}  
  
Console.ReadLine();  
  
}  
  
}  
  
}
```

نتائج حلقة التكرار:-

```
value of a: 10  
value of a: 11  
value of a: 12  
value of a: 13  
value of a: 14  
value of a: 15
```

value of a: 16

value of a: 17

value of a: 18

value of a: 19

أساسيات البرمجة حلقة التكرار

حلقة التكرار أو الدوارة

حلقة التكرار `do while` تقوم بفحص الشرط بأسفل الكود أي بعد تنفيذه مرة واحدة حتى لو كان الشرط خطأ بعكس جملة التكرار `while` ، `for` حيث يتم فحص الشرط بالأعلى ولا ينفذ إذا كان الشرط خطأ.

الصيغة العامة `do while`:-

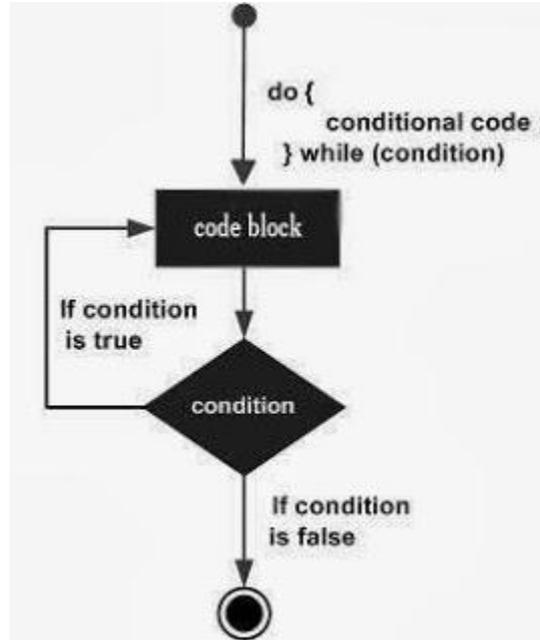
do

{

جملة الشرط او الكود ينفذ اولاً

while(الشرط);

مخطط التدفق لجملة الشرط do while او الخوارزمية



مخطط التدفق لجملة الشرط do while او الخوارزمية

مثال do while :-

```
using System;
```

```
namespace Loops
```

```
{
```

```
class Program
{
    static void Main(string[] args)
    {
        /* او خاصة متغيرات محلية */
        int a = 10;
        /* جملة الشرط */
        do
        {
            Console.WriteLine("value of a: {0}", a);
            a = a + 1;
        }
    }
}
```

```
} while (a < 20);
```

```
Console.ReadLine();
```

```
}
```

```
}
```

```
}
```

النتائج الكود -:do while

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

أساسيات البرمجة حلقة التكرار

حلقة التكرار أوالدوارة

حلقة التكرار do while تقوم بفحص الشرط باسفل الكود أي بعد تنفيذه مرة واحدة حتى لو كان الشرط خطأ بعكس جملة التكرار while ، for حيث يتم فحص الشرط بالاعلى ولا ينفذ اذا كان الشرط خطأ.

الصيغة العامة do while :-

do

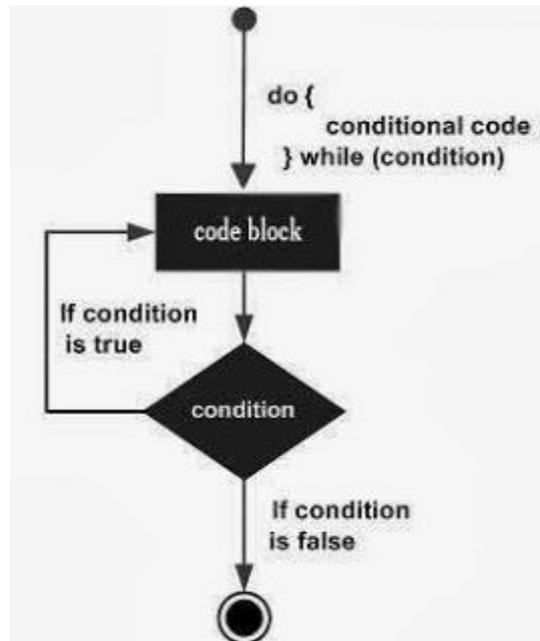
{

جملة الشرط او الكود ينفذ اولاً

```
}while(الشرط);
```

Do.....while loop

مخطط التدفق لجملة الشرط do while او الخوارزمية



مخطط التدفق لجملة الشرط do while او الخوارزمية

مثال do while :-

```
using System;
```

```
namespace Loops
```

```
{
```

```
class Program

{

static void Main(string[] args)

{

/* خاصة متغيرات محلية */

int a = 10;

/* جملة الشرط */

do

{

Console.WriteLine("value of a: {0}", a);
```

```
a = a + 1;  
  
} while (a < 20);  
  
Console.ReadLine();  
  
}  
  
}  
  
}
```

الناتج الكود -:do while

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

الفصل الخامس

المصفوفات

المصفوفات Arrays

تعتبر المصفوفات من بنى المعطيات المهمة في أي لغة برمجة. سيفترض هذا الدرس أنه لديك خبرة مسبقة عن مفهوم المصفوفة. المصفوفات في سي شارب هي عبارة عن نوع مرجعي `reference type`، وهي ترث من الصنف الأب `System.Array`. تقدّم لنا سي شارب المصفوفات بأسلوب مبسّط وواضح. فلتعريف مصفوفة يمكنها استيعاب ١٠ عناصر من النوع `int` مثلاً يكفي أن نكتب ما يلي:

```
int[] arrIntegers = new int[10];
```

للعبارة السابقة في الواقع وظيفتان: الأولى هي التصريح عن المتغيّر `arrIntegers` على أنه مصفوفة عناصرها من النوع `int` وذلك عن طريق كتابة `int[]` أول العبارة. والثانية هي إنشاء كائن المصفوفة وحجز ١٠ أماكن في الذاكرة بحيث يستطيع كلّ مكان منها استيعاب قيمة من النوع `int` وذلك عن طريق التعبير `new int[10]` ومن ثمّ إسناد المرجع لهذا الكائن إلى المتغيّر `arrIntegers`. ويمكن كما نعلم أن نجري هذه العملية على شكل عبارتين منفصلتين. يمكننا إنشاء أيّ نوع من المصفوفات نرغبه. فيمكننا إنشاء مصفوفات عناصرها نصوص `string[]`، ومصفوفات عناصرها أعداد ذوات فاصلة عائمة مثل `float[]`، وحتى يمكننا إنشاء مصفوفات عناصرها كائنات من أصناف ننشئها نحن بأنفسنا. فمثلاً إذا أنشأنا الصنف `Car` فيمكننا إنشاء مصفوفة من العناصر التي يقبل كل عنصر منها أن يخزن مرجع لكائن من الصنف `Car` وذلك على الشكل التالي:

```
Car[] arrCars = new Car[5];
```

تُنشئ العبارة السابقة المصفوفة `arrCars` والتي تحوي ٥ عناصر يمكنها تخزين مراجع لكائنات من الصنف `Car`.

استخدام المصفوفات مع أنواع مضمنة

لكلّ عنصر في مصفوفة دليل `index`، ويُعبّر عن ترتيب هذا العنصر ضمن المصفوفة. يبدأ ترقيم الأدلة في أيّ مصفوفة بالصفري. أي أنّ دليل العنصر الأول هو الصفري. فالمصفوفة `arrCars` التي صرّحنا عنها قبل قليل تحتوي على خمسة عناصر، دليل العنصر الأول هو ٠، أما دليل العنصر الأخير فهو ٤ كما هو واضح.

يمكن المرور على عناصر أيّ مصفوفة باستخدام الدليل. فمثلاً يمكننا الوصول إلى العنصر الثاني في المصفوفة `arrCars` عن طريق كتابة `arrCars[1]`.

يطلب البرنامج Lesson09_01 التالي من المستخدم إدخال درجات ٥ طلاب في إحدى المواد الدراسية ومن ثمّ يحسب معدّل هؤلاء الطلبة في هذه المادّة، على افتراض أنّ الدرجة العظمى هي ١٠٠. ومن ثمّ يطبع المعدّل مع أسماء الطلاب ودرجاتهم على الشاشة:

```
1  using System;
2
3  namespace Lesson09_01
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              int[] arrMarks = new int[5];
10             string[] arrNames = new string[5];
11             int sum = 0;
12
13             Console.WriteLine("Input Students Marks");
14             Console.WriteLine("=====");
15
16             //input loop.
17             for(int i = 0; i < arrMarks.Length; i++)
18             {
19                 Console.Write("Input student {0} th name: ",
i + 1);
20                 arrNames[i] = Console.ReadLine();
21
22                 Console.Write("Input student {0} th mark: ",
i + 1);
23                 string tmpMark = Console.ReadLine();
24                 arrMarks[i] = int.Parse(tmpMark);
25
26                 Console.WriteLine();
27             }
28
29             Console.WriteLine();
30             Console.WriteLine("Students Marks Table");
31             Console.WriteLine("=====");
32             Console.WriteLine("No\tName\tMark");
```

```

33
34         //calculating sum and display output loop.
35         for (int i = 0; i < arrMarks.Length; i++)
36         {
37             sum += arrMarks[i];
38             Console.WriteLine("{0}\t{1}\t{2}", i + 1,
arrNames[i], arrMarks[i]);
39         }
40
41         Console.WriteLine("-----");
42         Console.WriteLine("Sum\t\t{0}", sum);
43         Console.WriteLine("Average\t\t{0}",
sum/(double)arrMarks.Length);
44         Console.WriteLine();
45     }
46 }
47 }

```

يبدأ البرنامج السابق بالتصريح عن المصفوفتين `arrMarks` لتخزين علامات الطلاب و `arrNames` لتخزين أسمائهم. كما يصرّح عن المتغيّر `sum` لتخزين مجموع الدرجات. يعرض البرنامج عبارتين توضيحيّتين في السطرين ١٣ و ١٤، ثم تبدأ حلقة `for` في السطر ١٧ بجمع أسماء ودرجات الطلاب في هذه المادّة. لاحظ كيف أنّنا وضعنا شرط استمرار الحلقة $i < arrMarks.Length$ (السطر ١٧). تعطينا الخاصيّة `Length` للمصفوفة `arrMarks` عدد العناصر ضمن هذه المصفوفة (عددها ٥ في مثالنا). سيضمن ذلك تنفيذ حلقة `for` لخمس مرّات فقط. نبدأ اعتبارًا من السطر ٢٩ بالتجهيز لعرض النتائج، حيث سنظهرها على شكل جدول يضم ثلاثة أعمدة الرقم المتسلسل للطالب `No` والاسم `Name` والدرجة `Mark`. يطبع السطر ٣٢ ترويسة هذا الجدول من خلال النص `"No\tName\tMark"` نستخدم المحرف `t` في النص السابق لترك مسافة جدولة `tab` تفصل بين كل عمودين. يدخل البرنامج بعد ذلك إلى حلقة إظهار النتائج اعتبارًا من السطر ٣٥. لاحظ النص التنسيقيّ `" {0} \t{1} \t{2}"` في السطر ٣٨، وظيفته أيضًا ترك مسافة جدولة بين كل عمودين. بعد الانتهاء من الحلقة نُظهر المجموع `Sum` والمعدّل `Average` بقسمة المجموع `Sum` على عدد الطلاب.

أمر أخير تجدر ملاحظته، في السطر ٤٣ عند حساب المعدّل استخدمنا التعبير التالي `sum/(double)arrMarks.Length`؛ ويعود سبب وجود عامل التحويل (`double`) أمام `arrMarks.Length` إلى جعل القسمة تجري بين قيمة من نوع (`int`) قيمة (`sum`) وقيمة من نوع `double` لكي يصبح الناتج من نوع `double`. لأنّه بدون عامل التحويل هذا، ستجري عملية القسمة بين قيمتين من نوع (`int`) الخاصية `Length` هي من نوع (`int`) وبالتالي سيكون الناتج من نوع `int` ويُهمل أي أجزاء عشريّة وهذا أمر غير مرغوب. لقد نفذت البرنامج وقمت بتزويده ببعض البيانات، وحصلت على الخرج التالي:

```

C:\WINDOWS\system32\cmd.exe
Input Students Marks
=====
Input student 1 th name: Ahmad
Input student 1 th mark: 90

Input student 2 th name: Yaseen
Input student 2 th mark: 80

Input student 3 th name: Nour
Input student 3 th mark: 45

Input student 4 th name: Amjad
Input student 4 th mark: 78

Input student 5 th name: Sawsan
Input student 5 th mark: 64

Students Marks Table
=====
No      Name    Mark
1       Ahmad   90
2       Yaseen  80
3       Nour    45
4       Amjad   78
5       Sawsan  64
-----
Sum          357
Average     71.4
Press any key to continue . . .

```

استخدام المصفوفات مع أنواع من إنشاءنا

لا تختلف طريقة التعامل مع المصفوفات عناصرها من أنواع مضمّنة مع مصفوفات عناصرها من أصناف موجودة في مكتبة FCL أو حتى من أصناف ننشئها نحن، باستثناء أمرٍ مهمٍ واحد سنعرّض له.

سننشئ لهذا الغرض صنف جديد اسمه Student، يحتوي هذا الصنف على خاصيتين: الاسم Name والدرجة Mark. سنصرّح بعد ذلك عن المتغيّر arrStudents ليكون مصفوفة من النوع Student[]. سيسلك هذا البرنامج نفس سلوك البرنامج Lesson09_01 تمامًا، أي سيطلب درجات خمسة طلاب ليعرضهم ويحسب مجموع درجاتهم ومعدّلهم. انظر البرنامج Lesson09_02:

```
1  using System;
2
3  namespace Lesson09_02
4  {
5      class Student
6      {
7          public string Name { get; set; }
8          public int Mark { get; set; }
9      }
10
11     class Program
12     {
13         static void Main(string[] args)
14         {
15             Student[] arrStudents = new Student[5];
16             int sum = 0;
17
18             Console.WriteLine("Input Students Marks");
19             Console.WriteLine("=====");
20
21             //input loop.
22             for (int i = 0; i < arrStudents.Length; i++)
23             {
24                 arrStudents[i] = new Student();
25
26                 Console.Write("Input student {0} th name: ",
27 i + 1);
28
29                 arrStudents[i].Name = Console.ReadLine();
30
31                 Console.Write("Input student {0} th mark: ",
32 i + 1);
```

```

30         string tmpMark = Console.ReadLine();
31         arrStudents[i].Mark = int.Parse(tmpMark);
32
33         Console.WriteLine();
34     }
35
36     Console.WriteLine();
37     Console.WriteLine("Students Marks Table");
38     Console.WriteLine("=====");
39     Console.WriteLine("No\tName\tMark");
40
41     //calculating sum and display output loop.
42     for (int i = 0; i < arrStudents.Length; i++)
43     {
44         sum += arrStudents[i].Mark;
45         Console.WriteLine("{0}\t{1}\t{2}", i + 1,
arrStudents[i].Name, arrStudents[i].Mark);
46     }
47
48     Console.WriteLine("-----");
49     Console.WriteLine("Sum\t\t{0}", sum);
50     Console.WriteLine("Average\t\t{0}", sum /
(double)arrStudents.Length);
51     Console.WriteLine();
52     }
53 }
54 }

```

كلّ من البرنامجين Lesson09_01 و Lesson09_02 متشابهان من حيث الخرج. ولكن يتعامل البرنامج Lesson09_02 مع المصفوفة arrStudents التي عناصرها من النوع Student. الصنف Student مصرّح عنه في الأسطر من ٥ إلى ٩. والمصفوفة arrStudents مصرّح عنها في السطر ١٥. يُعتبر السطر ٢٤ مهمًا جدًا وفيه يتم إنشاء كائن جديد من الصنف Student وإسناده إلى كل عنصر من عناصر المصفوفة arrStudents في كل دورة من دورات حلقة for. إذا حاولت إزالة عبارة إنشاء الكائن من الصنف Student في السطر ٢٤ فسيعمل البرنامج ولكنه سيتوقّف عن التنفيذ ويصدر

خطأ عندما يصل التنفيذ إلى السطر ٢٧. لأنه عندما نصرّح عن مصفوفة عناصرها من أنواع ليست مضمّنة، فنحن في الحقيقة نصرّح عن متغيّرات فقط دون إنشاء كائنات ضمن هذه العناصر (المتغيّرات). وبالتالي لا يحق لنا الوصول إلى أعضاء كائن غير مُنشأ أصلاً، فالتعبير arrStudents.Name سيولّد خطأً ما لم يكن هناك كائن فعلي ضمن العنصر arrStudents[i] (أي عنصر المصفوفة ذو الدليل i).

حلقة foreach التكرارية

حلقة foreach من الحلقات التكرارية المفيدة والتي تتسم بأسلوب عمل أقرب إلى المؤلف. يمكن استخدام حلقة foreach على المصفوفات والمجموعات كما سنرى لاحقاً. طريقة استخدام foreach بسيطة سنتناولها من خلال البرنامج Lesson09_03 التالي:

```
1 using System;
2
3 namespace Lesson09_03
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             int[] arrNumbers = new int[10];
10            Random random = new Random();
11
12            for(int i = 0; i < arrNumbers.Length; i++)
13            {
14                arrNumbers[i] = random.Next(100);
15            }
16
17            foreach(int n in arrNumbers)
18            {
19                Console.WriteLine(n);
20            }
21        }
22    }
23 }
```

ملاحظة: البرنامج السابق بسيط، حيث يصرّح عن المصفوفة arrNumbers بعشرة عناصر، ويعمل على تعبئة عناصرها بقيم عشوائية يحصل عليها من كائن من النوع Random. صرّحنا عن المتغيّر random وأسندنا إليه مرجع لكائن من الصنف Random وذلك في السطر ١٠. في حلقة for (الأسطر من ١٢ إلى ١٥) قمنا بتوليد أرقام عشوائية عن الطريق التابع Next(100) من الكائن random والذي يولّد أرقامًا عشوائية صحيحة غير سالبة أقل من ١٠٠. بعد ذلك سنطبع هذه الأرقام العشوائية على الشاشة عن طريق حلقة foreach. مبدأ هذه الحلقة بسيط، منطق عملها يتلخّص على النحو التالي: "من أجل كل عنصر n موجود ضمن المصفوفة arrNumbers نفّذ الشيفرة الموجودة في حاضنة foreach". في مثالنا هذا تحتوي الحاضنة على عبارة برمجية واحدة (السطر ١٩).

في البرنامج Lesson09_03 السابق المتغيّر n المصرّح عنه في السطر ١٧ والذي سيجمل قيمة مختلفة من عناصر المصفوفة arrNumbers في كل دورة للحلقة، هو متغيّر للقراءة فقط read-only لا يمكن تغيير قيمته ضمن حاضنة foreach.

تمارين

تمرين ١

اكتب برنامجًا يعرّف مصفوفة من نوع int بعشرة عناصر. ثم يطلب من المستخدم إدخال قيم لهذه العناصر. بعد ذلك يعمل البرنامج على ترتيب عناصر هذه المصفوفة تصاعديًا.

ملاحظة: تمتلك سي شارب أساليب جاهزة وسريعة لمثل عمليّات الترتيب هذه، ولكننا نريد في هذا المثال التدرّب على الحلقات والمصفوفات.

تمرين ٢

عدّل البرنامج Lesson09_02 السابق لكي يسمح للمستخدم بإدخال بيانات عدد كفي من الطلاب.

ملاحظة: ستحتاج لأن تطلب من المستخدم إدخال عدد الطلاب المراد إدخال بياناتهم أولًا، ومن ثمّ تصرّح عن المصفوفة arrStudents بالعدد المطلوب.

الواجهات Interfaces والمجموعات Collections. تُعتبر المجموعات collections من المزايا القويّة والمهمّة في سي شارب، وهي وسيلة لتخزين العناصر ضمن بنية قابلة للتوسّع

تلقائياً وسهولة التعامل. إذاً فهي تشبه المصفوفات، باستثناء أنّ المصفوفات ذات حجم ثابت وهي تبقى كذلك منذ لحظة إنشائها، أمّا المجموعات فهي على النقيض من ذلك، فهي بنية قابلة للتوسّع بقدر ما نرغب. بالإضافة إلى أنّ للمجموعات أشكال عديدة مفيدة جداً، فهناك المكدّس stack والرتل queue والقاموس dictionary وغيرها من البنى المهمّة.

الفصل السادس

المؤشرات

المؤشرات pointers

يشير المؤشر إلى عنوان الذاكرة المرتبط بالمتغير، تُدعم المؤشرات بشكل مباشر بدون وجود قيود في لغات برمجة مثل PL/I و C و C++ و Pascal وأغلبية لغات التجميع. كما أنها تُستخدم بشكل أساس لتراكيب المراجع، والتي بدورها خطوة أساسية لبناء كل تراكيب البيانات تقريباً، تماماً كما يحدث في عملية تمرير البيانات بين الأجزاء المختلفة من البرنامج.

في اللغات البرمجة الوظيفية التي كثيراً ما تعتمد على القوائم، يتم التحكم بالمؤشرات والمراجع بشكل نظري من خلال اللغة باستخدام التراكيب الداخلية مثل cons.

تستخدم المؤشرات لتمرير المعاملات باستخدام مراجع لها. ويكون هذا مفيداً إذا أراد المبرمج أن يكون تعديل الدالة للمعامل مرئياً للجزء الذي قام باستدعائها. ويُفيد ذلك أيضاً في استرجاع العديد من القيم من الدالة. يُمكن استخدام المؤشرات أيضاً للذاكرة المخصصة وغير المخصصة للمتغيرات الديناميكية والمصفوفات داخل الذاكرة. في أغلب الأحيان ما يتحول المتغير إلى كائن زائد عن الحاجة بعد تحقيق الغرض منه، فإن الإبقاء عليه إهداراً للذاكرة، ولذا فمن الأفضل عدم تخصيصه أو توزيعه (مستخدماً مرجع المؤشر الأصلي) عندما انتهاء الحاجة إليه. قد يؤدي عدم القيام بهذه الخطوة إلى تسرب الذاكرة (memory leak) حيث تقل مساحة الذاكرة الفارغة بشكل تدريجي، أو في الحالات الخطير بشكل سريع، ويرجع ذلك إلى تراكم أعداد كبير من قوالب الذاكرة.

الصيغة الأساسية للإعلان عن مؤشر هي:

```
int *money;
```

يقوم هذا بالإشارة إلى القيمة "money" كمؤشر إلى عدد صحيح. حيث أن محتويات الذاكرة لا تضمن ما هي القيمة المحتمل أن تحتويها في لغة السي C، لذا يجب الحرص على التأكد من أن العنوان الذي يُشير إليه "money" أنه عنوان صحيح ومناسب. ولهذا السبب يُقترح في بعض الأحيان استهلال المؤشر بالقيمة NULL مع ذلك، يُمكن لاستهلال المؤشرات تورية التحليل وإخفاء الأخطاء

```
int *money = NULL;
```

عند إتمام عملية تتبع أحد المؤشرات له القيمة NULL فسوف يحدث خطأ في التشغيل ويتوقف التنفيذ ويحدث ذلك عادة مع أخطاء التخصيص. segmentation fault وعقب الإعلان عن المؤشر فإن الخطوة المنطقية التالية هي جعله يشير إلى شيء ما.

```
int a = 5;
int *money = NULL;

money = &a;
```

ويقوم هذا بإسناد عنوان الذاكرة للمتغير الصحيح a إلى قيمة المؤشر " money " ، على سبيل المثال إذا كان المتغير a مخزن في الذاكرة عند الموقع 0x8130 فستكون قيمة " money " هي 0x8130 بعد الإسناد. ولتتبع المؤشر يتم استخدام علامة النجمة * مرة أخرى:

```
*money = 8;
```

ويُشير مترجم الكود بأن يأخذ محتويات المؤشر money والتي هي 0x8130 وينتقل إلى ذلك العنوان ويعين القيمة المخزنة به إلى 8 a. فسوف تتضح قيمته ب 8. قد يتخلى هذا المثال بشكل أكثر وضوحاً ما إذا تم فحص الذاكرة بشكل مباشر. افترضاً أن a له عنوان الذاكرة 0x8130 و money له العنوان 0x8134، وافترضاً أيضاً أن الحاسوب المستخدم هو حاسوب 32 بت حيث أن سعة نوع البيانات int هي 32 بت. فإن الشكل التالي هو ما سوف يظهر في الذاكرة بعد تنفيذ الكود التالي:

```
int a = 5;
int *money = NULL;
```

العنوان	المحتويات
0x8130	0x00000005
0x8134	0x00000000

المؤشر ذو القيمة NULL المعروضة هنا هو 0x00000000. وبتعيين عنوان المتغير a للمؤشر money باستخدام مؤثر التعيين &

```
money = &a;
```

يكون الناتج في الذاكرة كما يلي :

العنوان	المحتويات
0x8130	0x00000005
0x8134	0x00008130

Then by dereferencing `money` by coding

```
*money = 8;
```

سيأخذ الحاسب الآلي محتويات المؤشر والتي هي 0x8130 وينتقل إلى ذلك العنوان ويعين القيمة 8 إلى ذلك الموقع ويكون الناتج في الذاكرة كما يلي:

العنوان	المحتويات
0x8130	0x00000008
0x8134	0x00008130

وكما هو واضح فإن استدعاء قيمة المتغير a سوف ينتج القيمة 8 لأن التعليمة السابقة قامت بتعديل محتويات المتغير a عن طريق المؤشر money.

مصفوفات لغة السي C#

تتم فهرسة المصفوفة في لغة السي C بشكل أساسي باستخدام المؤشر الحسابي، حيث أن معايير اللغة التي تتطلب أن يكون معادل إلى (array+i) * [٢] لذا يُمكن اعتبار المصفوفات كمؤشرات لمناطق متتابعة من الذاكرة (بدون أي فجوات [٢])، والصيغة المُستخدمة للولوج إلى المصفوفات ماثلة لتلك التي يُمكن أن تُستخدم في عملية تتبع المؤشر، ومثال على ذلك يُمكن الإعلان عن المصفوفة array واستخدامها على النحو التالي:

```
int array[5]; /* Declares 5 contiguous (per Plauger Standard C
1992) integers */
int *ptr = array; /* Arrays can be used as pointers */
ptr[0] = 1; /* Pointers can be indexed with array syntax */
*(array + 1) = 2; /* Arrays can be dereferenced with pointer
syntax */
*(1 + array) = 3; /* Pointer addition is commutative */
2[array] = 4; /* Subscript operator is commutative */
```

يقوم هذا بحجز مكان خاص لخمسة أعداد صحيحة ويُطلق على هذا قالب اسم المصفوفة التي تعمل كمؤشر لهذا المكان. من الاستخدامات الشائعة أيضاً للمؤشرات هي الإشارة للذاكرة المحجوزة ديناميكياً باستخدام دالة malloc التي تُعيد قالب متصل من الذاكرة بحيث لا يقل حجمه عن الحجم المطلوب الذي يُمكن استخدامه كمصفوفة. بينما معظم المؤشرات على المصفوفات والمؤشرات متكافئة إلا أنه من المهم الإشارة إلى أن دالة sizeof سوف تكون مختلف. في هذا المثال سوف تكون قيمة (المصفوفة sizeof (يُساوي ٥) sizeof (int)*حجم المصفوفة)، بينما سوف تكون قيمة (ptr sizeof (int) تساوي أي حجم المؤشر نفسه. يُمكن الإعلان عن القيم الافتراضية للمصفوفة مثل:

```
int array[5] = {2,4,3,1,5};
```

إذا افترضت أن المصفوفة array تقع في الذاكرة بدايةً من العنوان $\cdot x1000$ على جهاز الحاسب الآلي ٣٢ بايت وترتيب البيانات داخله little-andian فسوف تحتوي الذاكرة على التالي: (كون القيم الموجودة في نظام عد السادس عشري hexadecimal ، مثل تلك الموجودة في العناوين

	0	1	2	3
1000	2	0	0	0
1004	4	0	0	0
1008	3	0	0	0
100C	1	0	0	0
1010	5	0	0	0

يُوضح الشكل خمسة أعداد صحيحة ١ و ٢ و ٣ و ٤ و ٥. تشغل هذه الأرقام الخمسة ٣٢ بت (أي ٤ بايت) ويخزن كل منها البايت الأقل أهمية أولاً (وهذه هي بنية المعالجة المركزية-little-endian) وتخزن بشكل متتالي بدءاً من العنوان $\cdot x1000$ الصيغة المستخدمة في لغة السي C مع المؤشرات هي:

- المصفوفة array تعني $\cdot x1000$
- المصفوفة array +1 تعني $x1004$ لاحظ أن معنى "+١" هو إضافة مرة واحدة عدد حجم متغير int ولا يُقصد بذلك المعنى الحرفي ("١")
- المصفوفة *array تعني تتبع محتويات المصفوفة. array بمعنى اعتبار هذه المحتويات كعنوان الذاكرة ($\cdot x1000$ والبحث عن القيمة في هذا العنوان ($\cdot x1000$).
- array[i] تعني القسم i من المصفوفة والذي يترجم إلى (array + i)*
يوضح المثال الأخير طريقة الولوج إلى محتويات المصفوفة. وتفتيتها:
- Array + i هو موقع الذاكرة الخاص بعامل المصفوفة (i+1)th
- (array + i)* يأخذها عنوان الذاكرة لوصول إلى القيمة.
- على سبيل المثال array[3] :يترادف مع (array+3)* يعني ($\cdot x1000 + 3*\text{sizeof}(\text{int})$ ، والذي يُعلن "تتبع القيمة المخزنة في $\cdot x100C$ في هذه الحالة تكون القيمة $\cdot x0001$.

القوائم المرتبطة في لغة السي #C

فيما يلي مثال يوضح تعريف القوائم المرتبطة في لغة السي #C.

```
/* أو قيمة إشارة NULL القائمة المرتبطة الفارغة تمثل باستخدام */
/* أخرى */
تعريف القائمة المرتبطة الفارغة#

struct link {
    void *data; /* بيانات هذا الكائن */
    *next; /* القائمة EMPTY_LIST الكائن التالي؛ */
    /* فارغة إذا كان الأخير */
};
```

يُمكن ملاحظة أن تعريف المؤشر التكراري هذا هو نفسه تعريف المرجع التكراري من لغة البرمجة Haskell: `Data link a = Nil |Cons a (link a) Nil` هي القائمة الفارغة و `Cons` هي `(Link a)` هي خلية `cons` من النوع `a` مع رابط آخر من النوع `a`. مع ذلك التعريف باستخدام المراجع تُفحص ولا تستخدم قيم الإشارة التي تحتمل أن تكون مربكة. لهذا السبب، عادةً ما يتم التعامل مع هيكل البيانات في لغة `C` عن طريق وظائف التضمين، والتي فُحصت بعناية للتأكد من صحتها.

التمرير باستخدام المؤشرات

يُمكن استخدام المؤشرات لتمرير المتغيرات باستخدام مراجع لها، بما يسمح لقيمتها بأن تتغير، على سبيل المثال انظر إلى كود لغة `C++` التالي:

```
/* a copy of the int n is changed */
void not_alter(int n) {
    n = 360;
}
/* the actual variable passed (by address) is changed */
void alter(int *n) {
    *n = 120;
}

void func(void) {
    int x = 24;

    /*pass x's address as the argument*/
    alter(&x);

    /* x now equal to 120 */
    not_alter(x);
    /* x still equal to 120 */}
```

الفصل السابع

الواجهات والمجموعات في السي شارب

الواجهات والمجموعات في السي شارب

سنتناول في هذا الدرس المجموعات العادية، والمجموعات العمومية generics collections. سنبدأ هذا الدرس بالحديث المختصر عن الواجهات، وهي تشبه الأصناف ولكن بصورة مجردة، حيث تصف سمات عامة ينبغي أن تتحلّى بها الأصناف التي تحقّقها.

كلمة عن الواجهات

ليست فكرة الواجهات Interfaces محصورةً بـ سي شارب، فهي موجودة أيضاً في لغات برمجة أخرى مثل جافا Java. تشبه الواجهة Interface الصنف Class بشكل كبير، فهي تضم خصائص وتوابع، ولكن كتصاريح فقط، بدون شيفرة تحوي عبارات برمجية، وبدون محدّدات وصول. كما لا تحتوي على حقول fields ولا على بواني. تستطيع القول أنّ الواجهة تعرّف الهيكل العام بشكل مجرد فحسب. يمكن الاستفادة من الواجهة عندما نحقّقها. implement فعندما يرث صنف ما من واجهة، فيجب عليه أن يعيد التصريح عن جميع الخصائص والتوابع الموجودة ضمن هذه الواجهة ولكن مع تزويدها بحواضن تحوي الشيفرة المطلوب تنفيذها.

فكأنّ الواجهة تحقّق مبدأ التعددية الشكلية بشكل ممتاز، فهي تصرّح عن تابع أو خاصية، وتترك للصنف الذي يحقّقها (يرث منها) حرية التعبير عن هذا التابع أو هذه الخاصية بالشكل الملائم. تُستخدم الواجهات على نحو واسع جداً في سي شارب، وتتبع تسميتها لنفس أسلوب تسمية الأصناف، مع الاصطلاح على أن يكون الحرف الأوّل من اسم أيّ واجهة هو الحرف الطباعي الكبير I وهو الحرف الأوّل من كلمة Interface. في الحقيقة لقد حلّت الواجهات مشكلة الوراثة المتعدّدة (الوراثة من أكثر من صنف بنفس الوقت) والتي كانت تتمتع بها لغة C++ فقط. الآن أصبح بإمكان أي صنف أن يرث من صنف آخر واحد فقط، بالإضافة إلى تحقيقه لأي عدد يرغبه من الواجهات.

يُصرّح عن الواجهة بالكلمة المحجوزة interface. انظر البرنامج Lesson10_1 الذي يوضّح استخدام الواجهات.

هل تذكر الأصناف Animal و Bird و Frog و Fish؟ لقد استعرت الصنفين Animal و Frog لتوضيح فكرة استخدام الواجهات، تذكر أنّ الصنف Frog كان يرث من الصنف Animal. في الحقيقة لقد استفدت من فكرة أنّ الكائنات الحيّة تتنفس Breathing. لذلك أنشئت واجهة اسمها IBreathing لتعبّر عن عملية التنفس، وبما أنّ الضفدع Frog هو

كائن حيّ، فمن الطبيعي أن يحقق هذه الواجهة. تحتوي الواجهة IBreathing على تابع وحيد اسمه (TakeBreath السطر ٩) يقبل وسيطاً واحداً من النوع double يُعبّر عن كمية الأكسجين التي سيحصل عليها الكائن الحيّ عند التنفّس، كما يُرجع هذا التابع قيمة من النوع double أيضاً تمثل كمية الأكسجين التي بقيت بعد عملية التنفّس.

يرث الصنف Frog هذه الواجهة في السطر ٢٠، ويحقّقها من خلال إعادة تعريف التابع TakeBreath في الأسطر من ٢٧ حتى ٣٠ حيث يُعبّر عن عملية التنفّس بالشكل الذي يناسبه (سيستهلك في مثالنا هذا ٢٠% من كمية الأكسجين التي يحصل عليها). في الحقيقة يمكن استخدام الواجهة IBreathing مع أيّ "كائن حيّ" بصرف النظر عن كونه يرث من الصنف (Animal الذي يمثّل الصنف الحيواني) أو من الصنف (Mammal الثدييات) مثلاً أو غيره، وذلك لأنّ جميع الكائنات الحيّة تشترك معاً بخاصيّة التنفّس، وهنا تمكن قوّة الواجهات.

```
1 using System;
2
3 namespace Lesson10_01
4 {
5     class Program
6     {
7         interface IBreathing
8         {
9             double TakeBreath(double oxygen_amount);
10        }
11
12        class Animal
13        {
14            public virtual void Move()
15            {
16                Console.WriteLine("Animal: Move General
17Method");
18            }
19
20        class Frog : Animal, IBreathing
21        {
22            public override void Move()
```

```

23         {
24             Console.WriteLine("Frog - Move: jumping 20
cm");
25         }
26
27         public double TakeBreath(double oxygen_amount)
28         {
29             return oxygen_amount * 0.8;
30         }
31
32     }
33
34     static void Main(string[] args)
35     {
36         double oxygent_to_breath = 10;
37
38         IBreathing frog = new Frog();
39
40         Console.WriteLine("Oxygen amount before
breath: {0}", oxygent_to_breath);
41         Console.WriteLine("Oxygen amount after breath:
{0}", frog.TakeBreath(oxygent_to_breath));
42     }
43 }
44 }

```

هناك أمر آخر جدير بالملاحظة. انظر إلى السطر ٣٨ ستجد أننا قد صرّحنا عن المتغيّر frog من النوع IBreathing ثمّ أسندنا إليه مرجعًا لكائن من النوع Frog، وهذا أمر صحيح تمامًا وشائع جدًا لأنّ الصنف Frog يرث من الواجهة IBreathing. عند تنفيذ البرنامج ستحصل على الخرج التالي:

```

Oxygen amount before breath: 10
Oxygen amount after breath: 8

```

توجد الأصناف المعبّرة عن هذه المجموعات ضمن نطاق الاسم `System.Collection`. تلعب نطاقات الأسماء دورًا تنظيميًا للأصناف، وسنتحدّث عنها بشكل أكبر في درس لاحق. من أبرز المجموعات في نطاق الاسم هذا هو الصنف `ArrayList`. يحقّق الصنف `ArrayList` كلّ من الواجهات `ICollection` و `IEnumerable` و `ICloneable`. جميع هذه الواجهات تقع في مكتبة FCL في إطار عمل دوت نت، حيث تعرّف هذه الواجهات العمليّات الأساسيّة التي ينبغي أن يتمتّع بها الصنف `ArrayList`.

تسمح الكائنات من هذه المجموعة بإضافة أي نوع من العناصر لها، حيث من الممكن أن نضيف عناصر من النوع `object`. يمكن إضافة العناصر إلى هذه المجموعة باستخدام التابع `Add` الذي يقبل وسيطًا من النوع `object`. أي أننا فعليًا نستطيع أن نضيف عناصر من أنواع مختلفة لنفس المجموعة. أيّ عنصر تتمّ إضافته سيوضع آخر المجموعة التي هي ذات حجم مرّن، فمن الممكن إضافة أي عدد نرغبه من العناصر. أمّا إذا أردنا إضافة عنصر إلى مكان محدّد ضمن القائمة، فعلىنا استخدام التابع `Insert` الذي يحتاج إلى وسيطين، الأوّل هو الدليل المراد إدراج العنصر الجديد ضمنه، والثاني هو العنصر نفسه.

انظر البرنامج Lesson10_02 البسيط التالي:

```
1 using System;
2 using System.Collections;
3
4 namespace Lesson10_02
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             ArrayList values = new ArrayList ();
11
12             values.Add("My ");
13             values.Add("age: ");
14             values.Add(36);
```

```

15
16         foreach(object item in values)
17         {
18             Console.Write(item);
19         }
20
21         Console.WriteLine();
22     }
23 }
24 }

```

استطعنا الحصول على عناصر هذه المجموعة باستخدام حلقة `foreach`. ولكن إذا أردنا الوصول إلى عنصر محدد فحسب، ولنقل أنه العنصر الثالث (القيمة ٣٦) في مثالنا السابق، فيمكن ذلك من خلال الشكل التالي:

```
values[2]
```

تذكر دومًا أن دليل العنصر الأول هو ٠. في الواقع ستكون القيمة التي سنحصل عليها من `values[2]` هي قيمة من نوع `object` رغم أنها في حقيقة الأمر تحوي القيمة ٣٦ وهي قيمة من نوع `int` بطبيعية الحال. السبب في ذلك منطقي وهو أننا عندما أضفنا القيمة ٣٦ إلى المجموعة كان ذلك باستخدام التابع `Add` الذي يقبل وسيطًا من النوع `object`. إذا أردنا الاستفادة من القيمة الفعلية المخزنة ضمن `values[2]` فعلينا هنا أن نستخدم عامل التحويل (`int`) على الشكل التالي:

```
int age = (int) values[2];
```

ملاحظة: عند تمرير القيمة ٣٦ في المثال السابق إلى التابع `Add` الذي يتوقع وسيط من نوع `object` تحدث ظاهرة نسميها التعليب `boxing`. حيث تُعلَب القيمة ٣٦ ليصبح بالإمكان تمريرها مكان وسيط يتطلب النوع (`object`) يبقى هذا الأمر صحيحًا من أجل أي قيمة `value` (`type`). عندما نريد استرجاع القيمة الفعلية فإننا نقوم بعملية معاكسة تدعى بإلغاء التعليب `unboxing` باستخدام عامل التحويل بين الأنواع كما فعلنا بالعبارة البرمجية الأخيرة:

```
int age = (int) values[2];
```

هناك العديد من المجموعات الأخرى الموجودة ضمن نطاق الاسم System.Collection، ولكن لن أتحدث عنها هنا. في الحقيقة إذا أردت نصيحتي حاول ألا تستخدم المجموعات العادية أبداً! يكمن السبب في ذلك في الفقرة التالية عندما نتحدث عن المجموعات العمومية generic collection، حيث سنطّلع على مجموعات تشبه إلى حدّ بعيد المجموعات العادية الموجودة هنا، ولكنّها عملية وأكثر أماناً.

المجموعات العمومية

تشبه المجموعات العمومية generic collections من حيث المبدأ المجموعات العادية باستثناء أنها أكثر أمناً وأفضل أداءً. حيث ينبغي تعيين نوع العناصر التي ستتعامل معها المجموعة عند التصريح عنها، فتتعامل المجموعة في هذه الحالة مع نوع مُحدّد. من أشهر المجموعات العمومية هي المجموعة List<T> وهي تعبّر عن القائمة. list قد يبدو الشكل السابق غريباً قليلاً، ولكنّه في الحقيقة بسيط. استبدل الحرف T بأيّ نوع (صنف) ترغبه وستقبل المجموعة نتيجة لذلك أن يكون عناصرها من هذا النوع. تقع المجموعات العمومية في نطاق الاسم System.Collections.Generic.

سنعدّل البرنامج Lesson09_02 من الدرس السابق الذي كان يسمح بإدخال أسماء ودرجات خمسة طلاب فقط، ويخزنها على شكل كائنات Student ضمن مصفوفة من النوع Student [] وذلك لإيجاد مجموع الدرجات والمعدّل. سنجعل هذا البرنامج يستخدم المجموعة العمومية List<Student> (المجموعة يمكن لعناصرها تخزين مراجع لكائنات من النوع Student)، سننشئ البرنامج Lesson10_03 لهذا الغرض.

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace Lesson10_03
5 {
6     class Student
7     {
8         public string Name { get; set; }
9         public int Mark { get; set; }
10    }
11
```

```

12     class Program
13     {
14         static void Main(string[] args)
15         {
16             List<Student> listStudents = new
List<Student>();
17             int sum = 0;
18             bool continueCondition = true;
19             int counter = 0;
20             string response;
21
22             Console.WriteLine("Input Students Marks");
23             Console.WriteLine("=====");
24
25             //input loop.
26             while(continueCondition)
27             {
28                 Student student = new Student();
29
30                 Console.Write("Input student {0} th name:
", counter + 1);
31                 student.Name = Console.ReadLine();
32
33                 Console.Write("Input student {0} th mark:
", counter + 1);
34                 string tmpMark = Console.ReadLine();
35                 student.Mark = int.Parse(tmpMark);
36
37                 listStudents.Add(student);
38
39                 Console.WriteLine();
40                 Console.Write("Add another student? (y/n)
: ");
41                 response = Console.ReadLine();
42
43                 if(response=="n" || response == "N")
44                 {
45                     continueCondition = false;

```

```

46         }
47
48         counter++;
49     }
50
51     Console.WriteLine();
52     Console.WriteLine("Students Marks Table");
53     Console.WriteLine("=====");
54     Console.WriteLine("No\tName\tMark");
55
56     //calculating sum and display output loop.
57     for (int i = 0; i < listStudents.Count; i++)
58     {
59         sum += listStudents[i].Mark;
60         Console.WriteLine("{0}\t{1}\t{2}", i + 1,
listStudents[i].Name, listStudents[i].Mark);
61     }
62
63     Console.WriteLine("-----");
64     Console.WriteLine("Sum\t\t{0}", sum);
65     Console.WriteLine("Average\t\t{0}", sum /
(double)listStudents.Count);
66     }
67 }
68 }

```

لقد أجرينا هنا بعض التحسينات. بدأنا البرنامج في السطر ١٦ بالتصريح عن المتغير `listStudents` من النوع `List<Student>` وإنشاء كائن من هذا النوع وإسناده لهذا المتغير. تقبل المجموعة العمومية `List<Student>` بتخزين كائنات من النوع `Student` ضمنها. لاحظ أننا لم نحدّد عدد الكائنات مسبقاً (مع أنه يمكن ذلك بهدف تحسين الأداء لا غير). وضعنا حلقة `while` في السطر ٢٦ بدلاً من حلقة `for` القديمة وذلك لأننا لا نعرف على وجه التحديد عدد الطلاب الذين يرغب المستخدم بإدخال بياناتهم. لاحظ شرط استمرار الحلقة `continueCondition` الذي يحمل القيمة `true` بشكل افتراضي.

أصبح البرنامج غير مقيد بعدد محدد من الطلاب، فبعد إدخال بيانات كل طالب، سيعرض البرنامج رسالة يختار فيها المستخدم في إضافة المزيد أم التوقف (السطر ٤٠) فإذا اختار المستخدم التوقف بإدخاله النص "N" أو "n" عندها سيسند البرنامج القيمة false للمتغير continueCondition مما يؤدي إلى الخروج من حلقة while عند بدء الدورة التالية. تنحصر وظيفة المتغير counter الذي صرّحنا عنه في السطر ١٩ في إظهار ترتيب الطالب الحالي على الشاشة.

نستخدم الخاصية Count للمجموعة listStudents لمعرفة عدد العناصر الفعلية المخزنة ضمنها (تذكر الخاصية Length المماثلة لها في المصفوفات

بعد تنفيذ البرنامج وإدخال بيانات ستة طلاب، ستحصل على شكل شبيه بما يلي:

```
C:\WINDOWS\system32\cmd.exe
Input Students Marks
-----
Input student 1 th name: Ahmad
Input student 1 th mark: 90

Add another student? (y/n) : y
Input student 2 th name: Yaseen
Input student 2 th mark: 80

Add another student? (y/n) : y
Input student 3 th name: Nour
Input student 3 th mark: 45

Add another student? (y/n) : y
Input student 4 th name: Amjad
Input student 4 th mark: 78

Add another student? (y/n) : y
Input student 5 th name: Sawsan
Input student 5 th mark: 64

Add another student? (y/n) : y
Input student 6 th name: Nader
Input student 6 th mark: 67

Add another student? (y/n) : n

Students Marks Table
-----
No      Name      Mark
1       Ahmad     90
2       Yaseen    80
3       Nour      45
4       Amjad     78
5       Sawsan    64
6       Nader     67
-----
Sum      424
Average  70.66666666666667
Press any key to continue . . .
```

يوجد تابع اسمه RemoveAt ضمن هذه المجموعة يسمح بإزالة عنصر من القائمة، حيث نمرّر لهذا التابع دليل index العنصر المراد إزالته (دليل العنصر الأوّل هو ٠) ليعمل هذا التابع على إزالته وإعادة تعيين أدلّة جميع العناصر بعد إزالة العنصر المطلوب. انظر الشيفرة التالية:

```
List<string> listStrings = new List<string>();  
  
listStrings.Add("Bird");  
listStrings.Add("Fish");  
listStrings.Add("Frog");  
  
listStrings.RemoveAt(1);
```

أنشأنا في الشيفرة السابقة مجموعة من النوع (List<string> عناصرها من النوع string، ثم أضفنا إليها ثلاثة عناصر. يؤدّي استدعاء التابع RemoveAt(1) إلى إزالة العنصر ذو الدليل ١ من هذه المجموعة، أي أنّ العنصر ذو القيمة Fish سيُزال من هذه القائمة.

يوجد تابع مشابه لهذا التابع اسمه Remove يتطلّب أن تمرّر إليه مرجعًا لكائن موجود في هذه المجموعة لتتم إزالته. فإذا كان النوع العمومي لهذه المجموعة عبارة عن نوع قيمة مثل List<int> أو List<double> فعندها يكفي تمرير القيمة المراد إزالتها للتابع Remove فحسب. علمًا أنّ هذا التابع يزيل أوّل نتيجة تطابق يصادفها في هذه المجموعة.

يوجد أيضًا التابع Reverse الذي يعمل على عكس ترتيب العناصر الموجودة في المجموعة، حيث يصبح العنصر الأوّل هو الأخير، والعنصر الأخير هو الأوّل. كما يوجد التابع Sort الذي يعمل على ترتيب العناصر ضمن المجموعة وفق الترتيب الافتراضي (بالنسبة لأنواع المضمّنة) أو وفق ترتيبٍ كيفيٍّ يمكن للمبرمج أن يختاره.

وهناك تابع مفيد آخر وهو BinarySearch الذي يجري خوارزمية البحث الشهيرة على عناصر المجموعة، حيث نمرّر إليه القيمة المراد البحث عنها (أو مرجع الكائن الذي نريد البحث عنه) ويُرجع هذا التابع دليل العنصر ضمن المجموعة في حال وجدته. مع الانتباه إلى أنّ هذا الدليل يمثّل دليل العنصر ضمن المجموعة على اعتبارها مرتّبة. إذ أنّه يقوم بترتيبها بشكلٍ داخليّ قبل أن يجري عملية البحث. إذا أردت الحصول على نتائج منطقيّة، فاعمل على ترتيب مجموعتك باستخدام التابع Sort قبل استدعاء BinarySearch.

يمكننا الوصول إلى عنصر محدّد ضمن مجموعة عموميّة بنفس الأسلوب التي تحدّثنا عنه في المجموعات العاديّة، مع ملاحظة أنّنا لن نحتاج إلى عامل التحويل بين الأنواع، وبالتالي التخلّص من عمليتيّ التعليب boxing وإلغاء التعليب unboxing. كما يمكن الاستفادة أيضًا من التابع Insert للإدراج ضمن موقع مُحدّد، والذي تحدّثنا عنه أيضًا في المجموعات العاديّة. توجد توابع أخرى مفيدة ضمن المجموعة `List<T>` ولكننا لن نستطيع الخوض فيها قبل أن نتحدّث عن النّوَاب delegates في درس لاحق.

ملاحظة: توجد طريقة سريعة وفعّالة لإنشاء مجموعة `List<T>` وإسناد عناصر إليها مباشرةً في حال كان عدد العناصر محدّد ومعروف سلفًا. فإذا أردنا إنشاء مجموعة من النوع `List<int>` تحوي العناصر ١، ٢، ٥، ١٠ يمكنك كتابة العبارة التالية لهذا الغرض:

```
List<int> listNumbers = new List<int>() { 10, 5, 2, 1 };
```

تنشئ هذه العبارة مجموعة من النوع `List<int>` وتضيف إليها العناصر ١٠ و٥ و٢ و١ ثمّ تسند هذه المجموعة إلى المتغيّر `listNumbers`.

تمارين

تمرين ١

اكتب برنامجًا يطلب من المستخدم إدخال خمس قيم نصيّة، ويخزنها ضمن مجموعة من النوع `List<string>`. ثمّ استخدم التابع `Reverse` لعكس ترتيب العناصر ضمن هذه المجموعة، ثمّ اطبع النتائج على الشاشة.

تمرين ٢

اكتب برنامجًا يطلب من المستخدم إدخال قيم عدديّة من النوع `double` بقدر ما يريد، وبعد أن يفرغ من الإدخال، احسب المتوسط الحسابي (المعدّل) لهذه الأعداد، ورتّبها باستخدام التابع `Sort`، ثم اطبعها على الشاشة، مع المتوسط الحسابي لها.