C++

الدليل السريع

ترجمة وتأليف باسل الرمو سمير عزو

C++

الدليل السريع

فهرس المحتويات

المقدّمة:

بسم الله الرّحمن الرّحيم

الحمد لله ربّ العالمين الرحمن الرحيم مالك يوم الدين، والصلاة والسلام على سيّد الأوّلين والآخرين محمّد النبيّ الأمين وعلى آله وصحبه الغُرِّ الميامين ومن اهتدى بهديهم واستنَّ بسنتهم أجمعين.

قال الله تعالى[1]: "وقل ربِّ زِدني علماً". قال ابنُ كثير في تفسيره: "أي: زدني منك علماً".

قال ابن عُيَيْنَةَ رحمه الله: "ولم يزل رسول الله صلى الله عليه وسلم في زيادة [من العلم] حتّى توفّاه الله عزّ وجلّ".

وعن أبي هريرة ، رضي الله عنه، أنّ رسول الله، صلى الله عليه وسلم، قال: "ومن سلك طريقاً يلتمس فيه علماً، سهّل الله له به طريقاً إلى الجنة".

وعنه قال: سمعت رسول الله، صلى الله عليه وسلم، يقول: "الدنيا ملعونةٌ، ملعونٌ ما فيها، إلا ذكر الله تعالى، وما والاه، وعالماً، أو متعلماً".[2]

وعن أنس، رضي الله عنه قال: قال رسول الله، صلى الله عليه وسلم: "من خرج في طلب العلم، فهو في سبيل الله حتّى يرجع".[3]

وعن أبي سعيدٍ الخدري، رضي الله عنه، عن رسول الله، صلى الله عليه وسلم، قال: "لن يشبع مؤمنٌ من خيرٍ حتّى يكون منتهاه الجنة".[4]

وعن أبي أمامة، رضي الله عنه، أنّ رسول الله، صلى الله عليه وسلم، قال: "فضل العالم على العابد كفضلي على أدناكم ثمّ قال رسول الله، صلى الله عليه وسلم: إن الله وملائكته وأهل السموات والأرض حتى النملة في جحرها وحتى الحوت ليصلون على معلمي الناس الخير".[5]

وعن أبي هريرة، رضي الله عنه، قال: قال رسول الله صلى الله عليه وسلم: "من سئل عن علمٍ فكتمه، ألجم يوم القيامة بلجامٍ من نارٍ".[6]

إنّ ما تقدّم من الآيات والأحاديث إنّما يدّل على فضل العلم وطلّابه في الدنيا والآخرة ولكي تُحصّل الأجر العظيم من الله تعالى عليك أن تعلم أنّ نيّتك في طلب العلم مهمّة فقد روى إماما المحدِّثين البخاريّ ومسلم رضي الله عنهما في صحيحيهما -اللذين هما أصحّ الكتب المصنّفة- عن أمير المؤمنين أبي حفصٍ عمر بن الخطاب رضي الله عنه، قال: "سمعت رسول الله صلى الله عليه وسلم يقول: إنّما الأعمال بالنيّات، وإنّما لكلّ امرئٍ ما نوى فمن كانت هجرته إلى الله ورسوله فهجرته إلى الله ورسوله، ومن كانت هجرته لدنيا يصيبها، أو امرأةٍ ينكِحُها فهجرته إلى ما هاجر إليه".[7]

إنّما جُعِل هذا الكتابُ ليكون مساعداً للذين يودّون تعلّم شيء من علوم البرمجة (برمجة الحاسب) وفي محاولةٍ لزيادة الكتب العربيّة التي تهتمّ بهذا العلم الذي لا يشكُّ أحدٌ بإهميته للبشرية جمعاء وللأمّة الإسلاميّة على وجه الخصوص تلك الأمّة التي خاطبها الله تعالى فقال: "كُنْتُمْ خَيْرَ أُمَّةٍ أُخْرِجَتْ لِلنَّاسِ تَأْمُرُونَ بِالْمَعْرُوفِ وَتَنْهَوْنَ عَنِ الْمُنْكَرِ وَتُؤْمِنُونَ بِاللَّهِ وَلَوْ آمَنَ أَهْلُ الْكِتَابِ لَكَانَ خَيْرًا لَهُمْ".[8]

سائلين المولى عزّ وجلّ أن تعود هذه الأمّة إلى عهدها قائدةً ورائدةً في كل المجالات ومشكاةً للنور ومنبعاً للفكر العذب الصافي.

إنّ الذيْنِ كتبا هذا الكتاب هما من البشر لذلك نرجو منك أيُّها القارئُ الفاضلُ أن تعفوَ عمّا كان من الخطأ أو النسيان وأن تذكر أنّ الكمال لله وحده. أخيراً هذا الكتاب ليس مجانيّاً بالكامل وثمنُه دعوةٌ بالغيب لنا وللمسلمين فلا تنسونا من دعائكم.

---

[1] سورة طه الآية : 114
[2] رواه الترمذي وقال: حديثٌ حسنٌ. قوله وما والاه أي: طاعة الله.
[3] رواه الترمذي وقال: حديثٌ حسنٌ.
[4] رواه الترمذي وقال: حديثٌ حسنٌ.
[5] رواه الترمذي وقال: حديثٌ حسنٌ.
[6] رواه أبو داود والترمذي وقال: حديثٌ حسنٌ.
[7] متفقٌ على صحته.
[8] سورة آل عمران الآية : 110

إهداء

إلى المعلّم الأوّل محمّد صلى الله عليه وسلم.

إلى والدِينَا الأعزّاء.

إلى أصحاب الفضل علينا من إِخوانِنا ومعلّمِينا الفضلاء.

إلى كلّ المسلمين.

إلى من كان السبب في إخراج هذا الكتاب

المؤلفان

في يوم السبت 18 ذي القعدة سنة 1432 هجريّة

15 تشرين الأوّل سنة 2011 ميلاديّة

مقدّمة إلى البرمجة

تحتلّ علوم الحاسب الإلكترونيّ في أيامنا هذه مكانة عالية بين العلوم وهي ترتبط بشكل وثيق جدّا مع كلّ العلوم الأخرى بدءً بالعلوم الرياضيّة والفيزيائيّة مروراً بعلوم اللغات والعلوم الطبيّة وغيرها.

وممّا هو معلوم عند الناس أنّ الحاسبات تؤدّي العديد من الوظائف المهمّة في حياتنا اليومية بسرعة وكفاءة عاليتين وبدقة كبيرة في النتائج الأمر الذي ساعد العلماء على اختراع الكثير من الإبداعات العلمية بسرعة كبيرة تفوق تصوّرات الكثيرين من البشر ابتداءً بالآلات الميكانيكية كالسيارات إلى عالم الإنترنت المليء بالمفاجآت وصولاً إلى القمر حيث استطاع البشر الوصول إليه بإذن الله تعالى ثمّ بما أبدعته عقول العلماء وبمساعدة هذه الحاسبات.

إلّا أنّ الحاسب لا يستطيع أن يفعل أي شيء مالم يقم الإنسان بتعليمه وتدريبه على القيام به هذا الأمر من خلال برمجة الحاسب على القيام بما نريده أن يقوم به وهذا الكتاب صُنِع خِصّيصاً ليرافقك عزيزي القارئ في رحلة- نرجو من الله أن تكون ممتعة بالنسبة لك– في رحاب إحدى أهمّ وأقوى لغات البرمجة لتتمكّن بعدها من قيادة حاسبك وجعلِه يفعل كل الأشياء التي تريدها منه.

يشبه هذا الأمر الطريقة التي يتّبعها الناس في تربية وتعليم أبنائهم حيث يُولُونَهم الحبّ والرحمة ويحاولون جاهدين أن يجعلوا من أولادهم أفضل ما يمكن، وما نريده منك في هذه الرحلة أن تعزم على أن تقرأ هذا الكتاب كاملاً وأن تتحلّى بالصبر قال الله تعالى : "واستعينوا بالصبر والصلاة وإنّها لكبيرة إلا على الخاشعين"، فالصبرَ الصبر.

قد يبدو الأمر صعباً في أوّله ولكنّه سيسهل شيئاً فشيئاً مع تقدّمك في صفحات هذا الكتاب وتمرّنك على ما فيه من التمارين والأسئلة ولا تكتفِ بذلك بل اكتب كل ما يخطر ببالك من برامج وإن لم تتمكّن من ذلك بسهولة، و حاول أن تحلّ كلّ المشاكل التي تواجهها باستخدام البرمجة ليتثنّى لك ما تريد ، وتذكّر دائماً ما قاله أجدادنا الحكماء: "إهمال ساعة يفسد رياضة دهر" فإيّاك والإهمال وواصل العمل بجدّ فمن جدّ وجد ومن سار على الدرب وصل، ولكي تكون على الدرب عليك أوّلاً أن تعرف الدرب وهذا لا يكون إلّا بالتخطيط وتحديد الأهداف ثمّ الأسباب ثمّ تضع الخطة وتبدأ بتنفيذها وتذكّر هدفك دوماً –الغاية الأسما لكل مسلم إرضاء الله عز وجل– واعمل له كلّ ساعة.

أخيراً أقول لك: "اِعمل لدنياك كأنّك تعيش أبداً واعمل لآخرتك كأنّك تموت غداً".

البداية مع البرمجة

نقصد بالبرمجة صناعة البرامج وليس شرطاً أن تكون برامجاً تعمل فقط على الحاسبات أو على الآلات الإلكترونية. البرامج هي مجموعة من التعابير المنطقيّة والرياضيّة الخاصة المرتبطة مع بعضها لتكوّن حلولاً لمشاكل معينة.

ولكي نكتب برنامجاً ما علينا أن نقوم بخطوات محدّدة قبل ذلك وهذه الخطوات تبدأ أوّلاً **بتحليل المشكلة** المعطاة إلى مشاكل أبسط ثمّ نقوم **بوضع خطوات الحل** وذلك وفق ترتيب معيّن يلائم التحليل الذي حصلنا عليه وهذا الترتيب لخطوات الحل يدعى بالخوارزمية، ثمّ كتابة البرنامج بلغة البرمجة.
الخوارزمية: هي خطوات منطقيّة مرتّبة ترتيباً صحيحاً لحلّ مشكلة معيّنة.
لا بدّ في كل برنامج من مرحلتي التحليل وكتابة الخوارزمية بعد ذلك يأتي دور الحاسب. حيث يتبقّى أن تقوم بترجمة الخوارزمية التي صغتها إلى برنامج مكتوب بإحدى لغات البرمجة ثمّ يقوم الحاسب بعمله ليعطيك البرنامج الذي قمت بكتابه.
مثلاً لنفترض أنّنا نريد أن نقوم بحساب مساحة مستطيل.
أوّلاً: لنحلّل هذه المشكلة
ما هو المستطيل و كيف نحسب مساحته ؟ المسطيل هو شكل هندسي له أربعة أضلاع اثنان منها يمثلان الطول والاثنان الآخران يمثلان العرض أمّا مساحته فهي تنتج من ضرب الطول بالعرض.
الخوارزمية:

1- اقرأ الطول ثمّ العرض
2- اضرب الطول بالعرض
3- أعطني الناتج

هذه الخطوات تمثّل خوارزميّة لحل مشكلتنا السابقة. سنعتمد في هذا الكتاب على كتابة الخوارزميات بنفس الطريقة السابقة. ولن نكثر من الشرح حول الخوارزميّات فهناك كتب متخصّصة في هذا المجال أمّا هذا الكتاب فهو يناقش البرمجة وبعض البرامج وما يهمّنا هو توضيح الفكرة من الخوارزميات فقط ولن نحتاج إليها كثيراً خلال هذه الرحلة ويمكنك التوسّع في ذلك المجال بقراءة بعض الكتب المختصة. الآن بعد أن قمنا بوضع الخوارزميّة بقيت المرحلة الأخيرة لكتابة البرنامج ألا وهي ترجمتها إلى لغة برمجة يعرفها الحاسب.

لغة البرمجة
يواجه المبرمجون المشكلة ذاتها التي يواجهها المسافرون إلى بلدان أجنبيّة وهي مشكلة اختلاف لغة التخاطب بين المسافر والبلد الذي سافر إليه فكان من المناسب إيجاد حلّ لهذه المشكلة وهو اليوم اتقان اللغة الأكثر شهرة مثل اللغة الإنكليزيّة مثلاً فمعظم الناس يستطيعون التحدّث بهذه اللغة، وكذلك الأمر عند المبرمجين تمّ إيجاد لغات خاصّة للتخاطب بين البشر(المبرمجين) والحاسب والتي تعرف بلغات البرمجة programming languages وهذه اللغات صنعت لهذا الغرض ولن نطيل الكلام حول هذا الموضوع فالكلام فيه طويل وليس له أهميّة كبيرة في تعلّم البرمجة وتستطيع عزيزي القارئ الحصول على المعلومات الكاملة حول هذا الموضوع من خلال الإنترنت ففيه الكثير ممّا يجب عليك أن تتعلّمه أمّا ما يهمّنا هنا هو أنّ ما يكتب باستخدام لغة البرمجة يسمى شيفرة.

لكي يستطيع الحاسب معرفة هذه اللغة تمّ تصنيع برامج أخرى سمّيت مترجمات compilers تقوم هذه المترجمات بترجمة الشيفرة التي تكتب بلغة البرمجة إلى لغة أخرى هي لغة الآلة إذ أنّ الحاسب يمتلك لغة خاصّة حيث يمثّل كل البيانات والمعلومات فيه على شكل سلاسل من الأرقام الثنائيّة (0 و 1) ما يجعل التخاطب معه بلغته أمراً في غاية الصعوبة والتعقيد.

لمحة تاريخيّة عن C++

طوّرت C++ من لغة C التي طوّرت بدورها من لغتين سابقتين هما BCPL و B .
قام مارتين ريكارد Martin Richards عام 1967م بتطوير BCPL كلغة لكتابة أنظمة التشغيل والتطبيقات و المترجمات.

وأضاف كِن ثومبثون Ken Thompson العديد من الميزات الجديدة إلى لغته B التي تشبه BCPL إلى حدّ كبير واستخدمها في إنتاج النسخ الأولى من نظام التشغيل Unix الشهير وذلك في مختبرات Bell عام 1970م باستخدام الحاسب المعروف بـ DEC DPD-7 كلتا اللغتين كانتا عديمتي الأنواع في البداية.

طوّر دينِس ريتشي Dennis Retchie لغة C من لغة B في مختبرات Bell باستخدام الحاسب DEC DPD-11 عام 1972م.

حيث استخدمت C العديد من الأفكار الموجودة في اللغتين B و BCPL و أضافت إليهما أنواع البيانات وبعض الميزات الأخرى.

ومن هذه اللغة التي تلقّت العديد من التعديلات والتطويرات حتّى نالت درجة القياسية العالمية ISO(International Standards Organaizatoion) تمّ تطوير لغة C++ في مختبرات Bell من خلال Bjarne Stroustrup في مطلع الثمّانينات.

زوّدت C++ بمجموعة من المزايا إلّا أنّ أهمّها كان دعمها لمبادئ البرمجة كائنيّة التوجه OOP(Object-Oriented Programming)

لمن هذا الكتاب ؟

هذا الكتاب لكلّ من يريد أن يتعلّم البرمجة ولا يفترض هذا الكتاب وجود أيّ معرفة لدى القارئ لذلك ليس مهمّاً إن كانت هذه أوّل مرّة تقرأ فيها كتاباً في هذا المجال أو إن كانت المرّة الأولى التي تقرّر فيها تعلّم البرمجة ، وكذلك للأشخاص الذين يودّون الانتقال من لغات أخرى إلى هذه اللغة يمكن أن يقدّم هذا الكتاب المبادئ الأساسيّة للبرمجة باستخدام لغة C++ ولكنّ هذا الكتاب لا يعطي للأشخاص المتقدّمين في البرمجة سوى مراجعة لبعض معلوماتهم السابقة وقد يضيف بعض الأفكار البسيطة إن لم يكونوا قرؤوها من قبل.

كيف أقرأ هذا الكتاب ؟
إن كنت مبتدئاً ولم يسبق لك أن كتبت برنامجا من قبل فعليك أن تبدأ من حيث أنت أمّا إن كان لديك تصوّر كافٍ عن البرمجة بهذه اللغة فيمكنك أن تختار الموضوع الذي تودّ البدء منه فالكتاب مقسّم إلى ثلاثة أقسام رئيسيّة القسم الأول و الذي يحتوي على المبادئ الأساسية لكل المبتدئين في البرمجة والقسم الثاني الذي يحتوي على مواضيع أكثر تقدّماً كالمصفوفات والذاكرة الديناميكيّة والمؤشّرات أمّا القسم الثالث فقد تمّ تخصيصه للحديث عن البرمجة الكائنيّة التوجه وهي مرحلة متقدّمة بعض الشيء لذلك إن لم تتقن الفصول التي تسبقها فستواجه العديد من المشاكل.

فصول الكتاب

الفصل الأوّل : بنية البرنامج في لغة C++

نتحدّث في هذا الفصل عن البنية الأساسيّة للبرنامج وفق قواعد اللغة ونتناول أهمّ العناصر المشكّلة للبرنامج ونشرح كلّ واحدة منها من خلال البرنامج الأوّل لنا في هذا الكتاب.

الفصل الثاني: المتغيّرات

نطرح في هذا الفصل فكرة المتغيّرات التي تشبه خلايا الذاكرة في دماغ الإنسان فهي خلايا في ذاكرة الحاسب يستخدمها الحاسب لتخزين القيم التي يحتاجها أثناء تنفيذ البرنامج وسنناقش كيفية التعامل معها أيضاً.

الفصل الثالث : الإدخال و الإخراج

هذا الفصل هو بدايتك لكي تكون قادراً على التواصل مع الحاسب بشكل مباشر حيث ستتمكّن من الطباعة على الشاشة و القراءة من لوحة المفاتيح وهذا هو التخاطب الذي نبحث عنه بين المبرمج والحاسب.

الفصل الرابع : الثوابت

في هذا الفصل سنشرح فكرة استخدام الثوابت و الفرق بينها و بين المتغيّرات.

الفصل الخامس : المعاملات

نقصد بها الرموز التي تعني للحاسب عمليّات معيّنة إما رياضيّة أو منطقيّة أو عمليّات خاصّة تقوم بوظائف من أنواع أخرى والمعاملات ضروريّة جدّاً ولا يخلو برنامج منها أبداً لذلك ننصحك بالتمعّن أثناء قراءتها و التمرّن عليها بشكل جيّد.

الفصل السادس : بنى التحكم

نقدّم هنا أهمّ البنى البرمجيّة وهي تعليمات خاصّة تقوم بوظائف معينة منها ما يدعى ببنى الشرط و منها ما يدعى ببنى التكرار وهي أنواع نتعرّف عليها في حينها ثمّ نشرح فيه بعض الكلمات المستخدمة مع هذه البنى لمساعدتنا في إدارة برامجنا بشكل كامل.

الفصل السابع : التوابع

تمثّل التوابع البنية الرئيسيّة للبرنامج ولها فوائد أخرى حيث تسهّل التوابع كتابة البرامج وتقسّم البرامج إلى أقسام منظّمة واضحة للمبرمج ولقارئ الشيفرة.

الفصل الثامن : التعاودية

أسلوب خاصّ لكتابة التوابع يتمثّل بتكرار التابع عدداً منتهياً من المرّات دون استخدام بنى التكرار وهذا الفصل من أكثر الفصول صعوبة وفيه من المتعة الشيء الكثير وتجدر الإشارة إلى أنّ هذا الفصل غير مهمّ كثيراً لأنّه يستغنى عنه باستخدام البنى التكرارية.

الفصل التاسع : المصفوفات

بنى معطيات تسهّل التعامل مع البيانات التي لها نفس النوع كالبحث والترتيب والتخزين

الفصل العاشر : سلاسل الرموز

نناقش هنا كيفية التعامل مع النصوص حيث يتمّ تمثيل الكلمات والنصوص بسلاسل من الأحرف (الرموز).

الفصل الحادي عشر : المؤشّرات

تعتبر المؤشّرات من أقوى ميزات لغة الـ C++ حيث تسمح لنا بالتعامل مع الذاكرة مباشرةً وتمكّننا من إنشاء بنى معطيات ديناميكية. أحياناً تكون المؤشّرات صعبة الفهم ولكنّ مفهومها بسيط ويصبح التعامل معها سهلاً كلّما زاد التمرّن وكتابة الأكواد المتعلقة بها.

الفصل الثاني عشر : الذاكرة الديناميكية

مفهوم الذاكرة الديناميكيّة مرتبط جدّاً بالمؤشّرات حيث يتمّ حجز الذاكرة أثناء تنفيذ البرنامج باستخدام المؤشّرات

الفصل الثالث عشر : التراكيب

هي بنى معطيات خاصّة تسمح لنا بجمع عدّة أنواع من المعطيات تحت نوع واحد وهي مهمّة في قواعد البيانات

الفصل الرابع عشر : الأنواع المعرّفة من قبل المستخدم

نعرّف هنا أنواع بيانات جديدة غير موجودة بالاعتماد على أنواع موجودة ومعرّفة مسبقاً.

الفصل الخامس عشر : الإدخال و الإخراج في الملفات

نتعلم هنا كيفية التعامل مع الملفّات سواءً قراءة محتوياتها أو الكتابة عليها أو البحث فيها.

الفصل السادس عشر : معالجة الاستثناءات

هذا الفصل هو عبارة عن إضافة جديدة إلى لغة C++ نتعرّف من خلاله على طريقة عمليّة في منع تدمير البرنامج الناجم عن أخطاء معيّنة في العمليّات داخل البرنامج كأن يدخل المستخدم حرفاً أو اسماً بدلاً من أن يدخل رقماً. مثل هذه الأخطاء قد تتسبب بتوقف البرنامج وتدميره لذلك سيكون من الأفضل التخلّص منها بطريقة آمنة.

الفصل السابع عشر : مقدمة إلى البرمجة كائنية التوجه

يعتبر هذا الفصل مدخلاً إلى الصّفوف والكائنات ويقدّم بعض المفاهيم التي تخصّ هذا النمط من البرمجة ويتحدّث عن أسلوب بناء الصّفوف (الأنواع) الجديدة وأشياء أخرى تتعلّق بها.

الفصل الثامن عشر : مبادئ البرمجة كائنية التوجه

نتعرّض في هذا الفصل إلى المبادئ الثلاث للبرمجة كائنيّة التوجّه ونشرحها بشكل سهل و مختصر ونقدّم بعض الأمثلة عليها في محاولة لتقديم فكرة بسيطة عن هذا النمط من البرمجة ولا يعتبر هذا الكتاب كافياً في هذا المجال.

الفصل التاسع عشر : القوالب

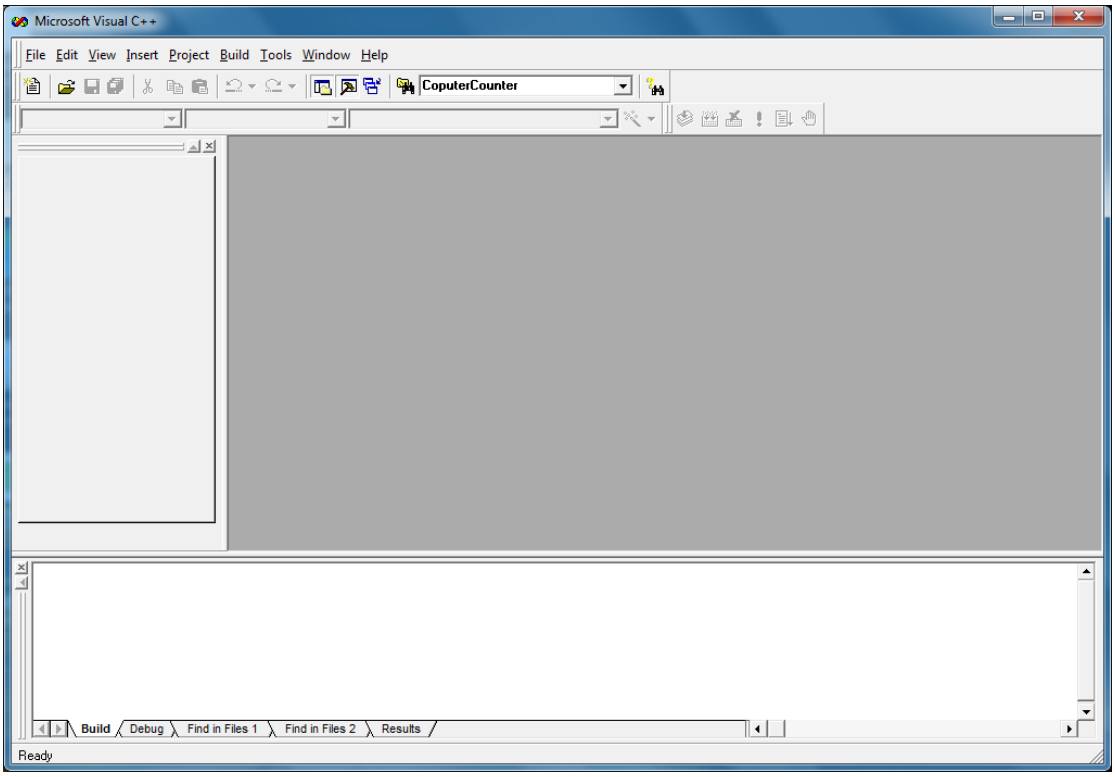
أسلوب قويّ جدّاً للتعامل مع الصفوف والتوابع واختصار الوقت والجهد الأمر الذي ستتعرّف عليه عندما تنهي آخر صفحة من صفحات البرمجة الكائنيّة.

الفصل العشرون : فضاءات الأسماء

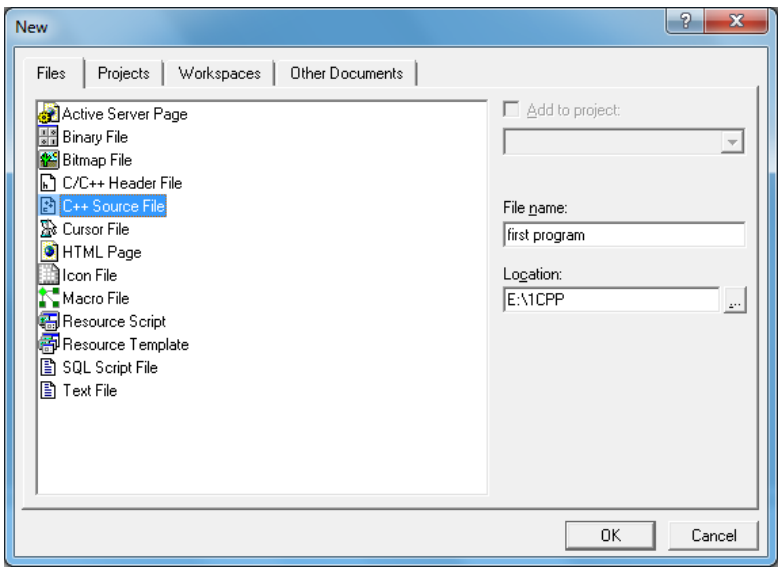
مكتبات قياسيّة موجودة في لغة C++ القياسيّة تحتوي على الكثير من المزايا و التوابع التي توفّر عليك الوقت و الجهد فهي تحوي المئات من التوابع الجاهزة والتي ستحتاجها في كلّ برامجك. سنتّعلم بسرعة كيف يمكننا أن نصنع فضاءات الأسماء الخاصّة بنا.

التعامل مع المترجم visual c++ 6.0

نقوم بفتح البرنامج بعد تنصيبه على الحاسب فتظهر النافذة التالية

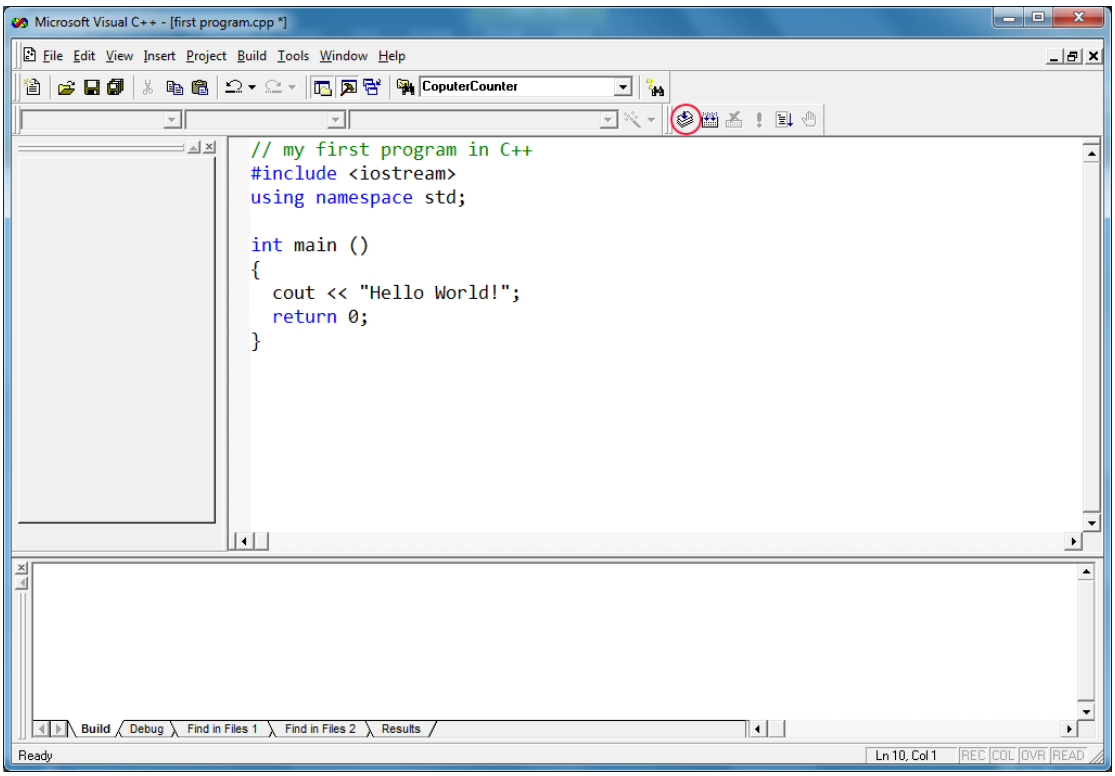


نختار New من القائمة File ثمّ نختار التبويب Files ثمّ نختار C++ Source File ثمّ نحدد اسم الملف ومكان الحفظ[9] ثمّ نضغط الزر OK

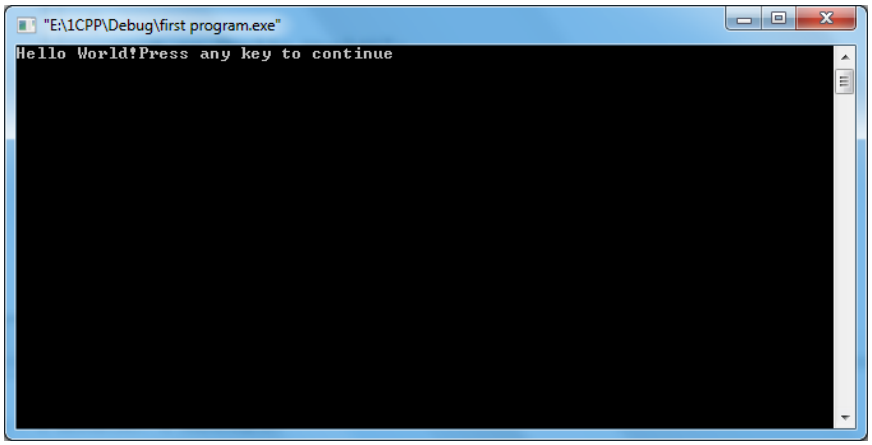


[9] أحياناً يجب تغيير مكان الحفظ عند العمل على Windows 7

نكتب البرنامج في المحرّر الذي يظهر ونضغط الزر (compile) أو ctrl+f7 لترجمة الشيفرة كما في الشكل



نضغط الزر أو ctrl+f5 لتنفيذ البرنامج كما في الصورة



القسم الأول

ما الفَضلُ إلّا لِأَهلِ العِلمِ إِنَّهُمُ عَلى الهُدى لِمَنِ اِستَهدى أَدِلّاءُ
وَقَدرُ كُلِّ اِمرِئٍ ما كانَ يُحسِنُهُ وَلِلرِجالِ عَلى الأَفعالِ اسماءُ
فَفُز بِعِلمٍ وَلا تَطلُب بِهِ بَدَلاً فَالناسُ مَوتى وَأَهلُ العِلمِ أَحياءُ

للإمام عليّ كرّم الله وجهه

الفصل الأوّل:

بنية البرنامج في لغة الـ C++

لعلّ أفضل ما نبدأ به رحلتنا مع هذه اللغة هو البرنامج الشهير الذي يتمتّع ببساطته إضافة لاحتوائه على المكوّنات الرئيسيّة التي لابدّ منها في كل برنامج، وليس عليك الآن لكي تكون مبرمجاً سوى أن تضع أناملك الناعمة تلك على لوحة المفاتيح. الآن هل أنت مستعدّ لكتابة أوّل برنامج لك؟ إذن هيّا بنا.

افتح المترجم واكتب فيه النصّ الذي على يسار الشكل ثمّ اضغط على أزرار التشغيل

```
// my first program in C++
#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

الشيفرة

```
Hello World!
```

ناتج التنفيذ(الخرج)

تهانينا! كان هذا أوّلَ برنامج لنا في C++، والآن دعنا نبدأ بشرح هذه العبارات

// my first program in C++

هذا السطر يدعى تعليقاً، فكل السطور التي تبدأ بـ // تعتبر تعليقات وهذه التعليقات ليس لها أي تأثيرعلى البرنامج.لقد جرت عادة المبرمجين على استخدام التعليقات لتوضيح ومراقبة عمل الشيفرة من داخل ملف المصدر.

#include <iostream>

الجمل التي تبدأ بالرمز(#) هي عبارات تقوم بتوجيه المترجم ولا يتمّ تنفيذها أثناء تنفيذ الشيفرة التي نكتبها لكنّها تعمل كدليل للمترجم، ففي مثالنا الجملة #include <iostream> تخبر المترجم بأن يقوم بتضمين الملف المصدري iostream وهو ملف معرّف سابقاً يحتوي على تصريحات للمكتبة القياسية الخاصة بالإدخال والإخراج في لغة C++ وقد استدعينا هذه المكتبة لأنّنا سنستخدمها لاحقاً في برنامجنا، وكأننا نقول للمترجم: "إننا سوف نستخدم التوابع المعرفة في هذا الملف المصدري لذلك اذهب وأحضره"، وسنأتي على شرح ذلك لاحقاً.

using namespace std;

جميع مكوّنات المكتبة القياسية للغة C++ معرّفة فيما يدعى بفضاء الاسم (namespace) الذي سنشرحه لاحقاً. ومن أجل الوصول إلى العديد من التعليمات والأوامر الجاهزة فإنّ هذا السطر يتكرّر كثيراً في البرامج التي تستخدم المكتبة القياسية حيث سترى هذا السطر في معظم البرامج الموجودة في هذا الكتاب.

int main()

يمثل هذا السطر بداية التصريح عن التابع الرئيسي main.
إنّ التابع main هو النقطة الأولى لبداية أي برنامج مكتوبٍ بلغة C++. وهذا لايعتمد على كون هذا التابع مكتوباً في أوّل الشيفرة أو في وسطها أو في آخرها فأينما كان هذا التابع مكتوباً فإنّ التنفيذ سيبدأ منه أي أنّه ضروريّ جدّاً ليعمل البرنامج ومن دونه لا يوجد برنامج.
في التصريح عن التابع main وضعنا القوسين ( ) بعد اسم التابع وذلك لأنّ الصيغة العامّة للتصريح عن التوابع تقضي بذلك كما سيرد معنا فيما بعد.
كلّ التعليمات التي سيقوم التابع main بتنفيذها محصورة بين القوسين{ } كما هو مبين في المثال السابق.

cout << "Hello World!";
هذه التعيلمة هي الأكثر أهمية في هذا البرنامج. تعتبر cout تعليمة قياسية في C++ وهي تقوم بطباعة الجملة Hello World على الشاشة.
ومن الجيد التنويه إلى أن التعليمة cout معرّفة في الملف المصدري iostream وقد قمنا بتضمينه في بداية برنامجنا كي نتمكن من استخدامها.

ملاحظة: من المهمّ أن تعرف أنّ كلّ تعليمة في C++ **يجب** أن تنتهي بفاصلة منقوطة (;).
(من الأخطاء الشائعة بين مبرمجي C++ أنّهم ينسَون وضع الفاصلة المنقوطة في نهاية بعض التعليمات ممّا يُحدث خطأ يمنع البرنامج من العمل، **فانتبه** لذلك).

return 0;
تقوم التعليمة return بإنهاء التابع main() وتعيد التعليمة التي تأتي بعدها كقيمة للتابع، في حالتنا هذه تعيد 0.

هذه هي الطريقة المعتادة لإنهاء البرنامج فعندما يعيد التابع main() القيمة **0** نعلم أنّ البرنامج انتهى دون أيّة أخطاء أثناء التّنفيذ.

ومن ثمّ لابد أنّك قد لاحظت أنّ كل سطر في هذا البرنامج قام بعمله، فالتعليق الذي بدأ بالرمز (//) الذي وجّه المترجم لأن يتجاهل هذا السطر، وهناك السطر الذي بدأ بالرمز(#) الذي وجّه المترجم كي يجلب الملف المصدري iostream، وكان هناك السطر الذي صرّحنا به عن التابع main()، وأخيراً التعليمة cout التي وضعناها بين القوسين { } الخاصّين بالتابع main.

تنسيق الكتابة

يمكننا أن نكتب:

```
int main () { cout << " Hello World "; return 0; }
```

كلّ تعليمة في C++ تنتهي بالفاصلة المنقوطة كما ذكرنا. لذلك لا فرق في أن تكتب التّعليمات على سطر واحد أو في عدّة أسطر، فطالما أنّك لم تضع فاصلة منقوطة فهذا يعني أنّ التّعليمة لم تنتهِ بعد.
لكنّنا كتبناه كما في الجدول لكي تسهل قراءته.

والآن سنكتب المثال التالي بطريقة أخرى ممكنة

```
// my second program in C++
#include <iostream>
using namespace std;
int main ()
{
  cout <<
        "Hello World! ";
  cout <<
        "I'm a C++ program";
  return 0;
}
```

```
Hello World! I'm a C++ program
```

و**يجب** الانتباه إلى أنّ الموجّهات (الأسطر التي تبدأ بـ #) تشذُّ عن هذه القاعدة لأنّها في واقع الأمر لا تمثّل تعليمات حقيقيّة فهي أسطر يقرؤها المترجم لكنّها لا تمثّل أيّ جزء من شيفرة البرنامج، فهذه الأسطر **يجب** أن تكتب في مكانها الخاص في أعلى البرنامج وهي لا تحتاج إلى فواصل منقوطة في نهاياتها.

التعليقات

التّعليق هو قطعة من الشيفرة مهملة من قبل المترجم. إذن التعليقات لا تقوم بأيّة وظيفة برمجية، وإنّما هي موجودة فقط لتذكّر المبرمج ببعض الملاحظات حول شيفرته.

تقدّم C++ نوعين من التعليقات هما كالآتي:

تعليق على نفس السطر // line comment

تعليق على سطر أو أكثر /* block comment */

والآن لنضف بعض التعليقات إلى برنامجنا السابق

```
/* my second program in C++
   With more comments */
#include <iostream>
using namespace std;
int main ()
{
//says hello world
  cout<<"Hello World! ";
//says I'm a C++ program
  cout<<"I'm a C++ program";
  return 0;
}
```

```
Hello World! I'm a C++ program
```

ملاحظة: عندما تريد أن تضيف تعليقاً ما فعليك ألّا تنسَ استخدام أحد الرمزين // أو /* */ لأنّ ذلك سيتسبب بأخطاء. إذ أنّ المترجم سيعتبر هذه الأسطر تعليمات بدلاً من أن يعتبرها تعليقات وهذا ما لا تريده أنت ولا المترجم طبعاً.

تمارين

1. املأ الفراغات التالية :
   - كل برنامج في C++ يبدأ من التابع _____.
   - كل عبارة في C++ تنتهي بـ _____.
   - الرمز // يدل على أن هذا السطر _____.
   - التعليقات هي _____.
   - التعليمة return 0; تعمل على _____.
   - التعليمات الخاصة بالتابع main موجودة بين _____.
   - التعليمة #include<iostream.h> تعني _____.
2. اجعل البرنامج السابق يطبع العبارة التالية :

In the name of Allah. The peace upon you, Hello world!

الفصل الثاني:

المتغيّرات (Variables)

إنّ الفائدة من برنامج hello world الذي قمنا بإنشائه في الدرس السابق هي أن نعرض نصّاً ما على شاشة الحاسب، **لكن هل البرمجة محدودة إلى هذه الدرجة؟**

لنفترض أنّني طلبت منك أن تحتفظ بالرقم 4 في ذهنك ثمّ طلبت منك أن تحتفظ بالرقم 1، الآن رجاءً أضف 3 إلى الرقم الأوّل، ثمّ اجمع الرقم الأوّل إلى الرقم الثاني وأعطني الناتج.
يمكن أن نعبّر برمجياً عن ذلك كما يلي:

```
a = 4 ;
b = 1 ;
a = a + 3 ;
result = a + b ;
```

إنّ الحاسب يقوم بإجراء هذه العمليات تماماً كما تقوم أنت بها، لكنه يتميّز عنك بأنّه قادر على الاحتفاظ بآلاف القيم وإجراء عشرات العمليات الحسابية المعقّدة في وقت قصيرٍ جدّاً وهذا يتمّ من خلال إدخال القيم إلى الحاسب وتخزينها في أماكن مخصّصة في الذاكرة ندعوها بالمتغيّرات.

المتغيّرات: هي أماكن (خلايا) في الذاكرة المؤقّتة تخزّن فيها البيانات، ويتمّ الوصول إليها من خلال اسم المتغيّر الذي يميّزه عن باقي المتغيّرات في الذاكرة.

كيف نسمي المتغيّرات؟
إنّ الاسم الصحيح للمتغيّر يتألف من سلسلة متّصلة مكوّنة من محرف أو أكثر، و يمكن أن تحوي أرقاماً أو الرمز(_). شريطة أن يبدأ اسم المتغيّر بحرف أو بـ (_)، وألّا يبدأ برقم أو يحوي على فراغات أو رموز خاصّة مثل: @ -. # $....
هل هناك طول معين لاسم المتغيّر؟
إنّ طولَ اسمِ المتغيّر ليس محدوداً، ولكنّ بعض المترجمات تقبل فقط الـ 32 حرفاً الأولى من اسم المتغيّر وتهمل باقي الأحرف.
قاعدة أخرى **يجب** عليك أن تضعها في حسبانك: اسم المتغيّر لا يمكن أن يكون إحدى الكلمات المحجوزة في لغة C++ و هذه الكلمات هي:

```
asm, auto, bool, break, case, catch, char, class, const,
const_cast, continue, default, delete, do, double,
dynamic_cast, else, enum, explicit, extern, false, float, for,
friend, goto, if, inline, int, long, mutable, namespace, new,
operator, private, protected, public, register,
reinterpret_cast, return, short, signed, sizeof, static,
static_cast, struct, switch, template, this, throw, true, try,
typedef, typeid, typename, union, unsigned, using, virtual,
void, volatile, wchar_t
```

ويضاف إليها مجموعة العمليات المنطقية التالية:

```
and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq,
xor, xor_eq
```

حساسية اللغة لحالة الأحرف:

إنّ لغة C++ حسّاسة لحالة الأحرف فعندما تكتب اسم المتغيّر عليك أن تراعي حالة كلّ حرف من حروفه والأمثلة التالية كفيلة بشرح هذه الفكرة إن شاء الله:

في المثال السابق المتغيّر a يختلف تماماً عن المتغيّر A، وكذلك كل المتغيّرات التالية مختلفة

Result , result , RESULT …

أي أنّه لو قمنا بالتصريح عن متغيّر ما بالاسم a فلن يكون بمقدورنا أن نستدعيه إلا بالاسم a ولو استدعيناه بالاسم A فلن يعرفه المترجم لأنّه سيعتبره متغيّراً آخرَ وهو غير موجود.

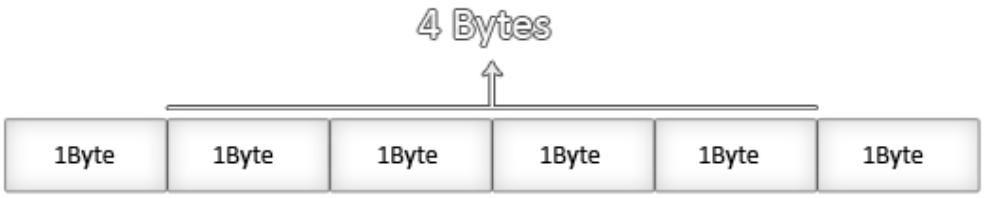
أنواع البيانات (Data types)

عندما نكتب برنامجا فإنّنا سنحتاج إلى تخزين البيانات في الذاكرة باستخدام المتغيّرات مما يعني أنّنا سنحجز مساحةً في الذاكرة لكلّ واحدٍ من المتغيّرات التي سنتعامل معها.

لكن هل لكلّ المتغيّرات نفس الحجم في الذاكرة؟ لتوضيح هذه المسألة لنفترض أنّنا نريد أن نخزّن كلّاً من القيم التالية 10 و 45.123 و Hi Everybody.

إنّ ذاكرة الحاسب مقسّمة إلى وحدات تدعى بايتات Bytes. والبايت هو وحدة قياس الحجم الصغرى التي سنتعامل معها. نستطيع أن نخزّن في البايت الواحد عدداً صحيحاً بين 0 و 255 أو حرفاً أو رمزاً واحداً فقط.

واضح تماماً أنّنا لن نستخدم فقط الأرقام من 0 إلى 255، ولن ندخل حرفاً واحداً فقط فنحن بحاجة لأكثر من ذلك بكثير لذلك يوفّر لنا الحاسب أنواعاً أخرى أكثر تعقيداً من النوعين السابقين وذلك بتجميع عدّة بايتات إلى بعضها لتتّسع لحجم أكبر من البيانات. (كما هو موضح في الشكل)



نعرض في الجدول التالي عدّة أنواع من البيانات المستخدمة في C++.

نوع المتغيّر	الحجم بالبايت	المدى (يمثل القيم التي يمكن للمتغيّر أن يأخذها)	ملاحظات
char	1	بإشارة [-128,127] دون إشارة [0,255]	**حرف واحد أو رقم طوله 8 بِت**
short	2	بإشارة [-32768 , 32767] دون إشارة [0,65535]	**رقم صحيح طوله 16 بِت**
long	4	بإشارة [-2147483648 , 2147483647] دون إشارة [0,4294967295]	**رقم صحيح طوله 32 بِت**
int	2 أو 4	إما short أو long	**رقم صحيح**
float	4	$3.4e^{-38}$ إلى $3.4e^{+38}$ (7 أرقام)	**رقم عشري**
double	8	$1.7e^{-308}$ إلى $1.7e^{+308}$ (15 رقم)	**رقم ذو دقة عشرية مضاعفة عن float**
long double	10	$1.7e^{-4932}$ إلى $1.2e^{+4932}$ (15 رقم)	**رقم ذو دقة عشرية مضاعفة عن double**
bool	1	true أو false	**متغيّر منطقي بإحدى القيمتين 0 أو 1**

والآن لنعد إلى موضوعنا الذي كنّا بدأنا الحديث عنه سابقاً.

كنّا نقول: **كيف يمكننا أن نصرّح عن المتغيّرات؟**
تفرض علينا اللغة أن نذكرَ نوع البيانات (int, short, float,...) التي سيأخذها المتغيّر، ثمّ نذكرَ اسمه وفقاً لقواعد التسمية التي أوردناها للتو. أي أنّ التصريح عن المتغيّرات يخضع للقاعدة التالية:

```
data_type  variable_Identifier ;
```

أو

```
data_type  variable_Identifier  =  initial_value  ;
```

أمثلة:

```
int  a ;
float  mynumber ;
long   mysecondnumber = 5 ;
```

كما تلاحظ بدأ التصريح عن المتغيّر `a` بالنوع `int` أي أنّنا نقول للمترجم: "إنّ المتغيّر هو من نوع الأعداد الصحيحة `int` و اسمه `a`"، وكذلك الأمر في المثال الثّاني. أمّا المثال الثّالث فقد تابعنا الخطوات بأن قمنا بتبدئته (أي إسناد قيمة ابتدائية للمتغير أثناء التصريح عنه) على القيمة 5.

تمكّننا C++ من التصريح عن عدّة متغيّرات من نفس النوع وتبدئتها إن أردنا في سطر واحد.

مثال:

```
int  a, b = 3, c = 5 ;
```

وهذا مكافئ تماماً للشّكل التالي:

```
int  a ;
int  b = 3 ;
int  c = 5 ;
```

لعلّك لاحظت من الجدول أنّ الأنواع (int , short , long, char) تكون بإشارة أو بدون إشارة وهذا تابع للمجال الذي نرغب بأن يضمّه المتغيّر ففي بعض الحالات نكون **واثقين** بأنّ القيم التي سيأخذها متغيّر عدديّ هي من النّوع الصّحيح و الموجب فقط عندها قد لا نرغب بأن يكون المتغيّر معرّفاً ليستقبل الأعداد السالبة.

تستطيع أن تحدّد فيما إذا كان المتغيّر من الأنواع الأربعة السابقة يمتلك إشارة أو لا وذلك باستخدم إحدى الكلمتين التاليين:

unsigned short NumberOfSons ;
signed int MyAccountBalance ;

في الحالة الأولى من المعروف أنّ عدد الأبناء لا يمكن أن يكون أقل من الصفر فإمّا أن يكون هناك أبناء أو لا يكون لذلك صرّحنا عن NumberOfSons على أنّه بدون إشارة سالبة.

أمّا في الحالة الثانية فلا شيء يضمن عدم إدخال قيمة سالبة على المتغيّر MyAccountBalance ***(دلالة على الخسارة مثلاً)*** لذلك صرّحنا عنه على أنّه من النوع `signed` الذي يستطيع أن يحمل أرقاماً سالبة.

وبشكل افتراضي إذا لم تحدد فيما إذا كان المتغيّر من النوع signed أو unsigned فإنّ المترجم سيعتبره من النوع signed لذلك يمكن التصريح عن المتغيّر في الحالة الثانية من المثال السابق بالشكل:

Int MyAccountBalance ;

وهذا هو الشكل المعتاد لهذه الحالة.

والآن لنكتب المثال الذي بدأنا به درسنا

```
// operating with variables
#include <iostream>
using namespace std;
int main ()
{
  // declaring variables:
  int a, b;
  int result;
  // process:
  a = 10;
  b = 4;
  a = a + 2;
  result = a + b;
  // print out the result:
  cout << result;
  // terminate the program:
  return 0;
}
```

```
16
```

ملاحظة: إنّ الطريقة السابقة بتبدئة المتغيّرات مأخوذة من لغة C، وهذا ما يدعى (C-Like)، لكنّ C++ تمتلك طريقة أخرى لفعل ذلك وهي :

```
data_type  variable_Identifier (initial_value) ;
```

أي أنّ:

```
int a = 5 ;   ↔   int a (5) ;
```

مقدّمة إلى السلاسل (strings)

إنّ النوع **string** يعبّر عن سلسلة من الرموز المحفوظة في حجرات متتالية من الذاكرة. توجد في لغة C++ طريقتان للتعامل مع السلاسل، الأولى مأخوذة من لغة C وتدعى C-style string سنشرحها لاحقاً في فصل سلاسل الرموز أمّا الطريقة الثانية فتعتمد على الصفّ **string** ومكتبة القوالب القياسية STL.

لكي نستطيع أن نتعامل مع السلاسل **string** يجب أن نضمّن الملف المصدري **<string>**

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string name;
  cout<<"enter your name: ";
  cin>>name;
  cout << "Hello "<<name<<endl;
  return 0;
}
```

```
enter your name: Basil
Hello Basil
```

التعليمة **endl** تقوم بنقل المؤشّر إلى السطر التالي كما سنرى فيما بعد.

هناك مجموعة من العمليّات المعرّفة في الصفّ **string** كالجمع (+) والإضافة (=+) والمقارنة (<,>)

مثال:

```
// my second string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
   string firstName,lastName,fullName;
   cout<<"enter your first name: ";
   cin>>firstName;
   cout<<"enter your last name: ";
   cin>>lastName;
       fullName=firstName+" "+lastName;
       cout<<"Hello "<<fullName<<endl;

       if(firstName<lastName)
             cout<<firstName<<endl;
return 0;
}
```

```
enter your first name: Ahmad
enter your last name: Sami
Hello Ahmad Sami
Ahmad
```

تمارين

1. أجب عن الأسئلة التالية :
   - هل المتغيّر Num هو نفس المتغيّر num في C++ ؟
   - هل يجب أن يعطى المتغيّر نوعاً من البيانات عند التصريح عنه ؟
   - هل يمكن أن تكون أسماء المتغيّرات أي مجموعة من الرموز ؟
   - هل نستطيع أن نعرّف متغيّراً ونعطيه قيمة في نفس اللحظة ؟
   - ما الفرق بين المتغيّرات signed و unsigned .
2. املأ الفراغات التالية :
   - **المتغيّرات** هي _____ في ذاكرة الحاسب _____فيها البيانات، ويتمّ الوصول إليها من خلال _____المتغيّر الذي _____عن باقي المتغيّرات في _____.
3. اختر الإجابة الصحيحة؟

a) unsigned long f = -1.25 ;
b) int 4men ;
c) double d = 'S';
d) long _2u ;
e) short hi. ;
f) char u = "U" ;

4. ما هي الأخطاء في البرنامج التالي:

```
#include <iostream>
using namespace std;
int main ()
{
   float a = 1 , b = 0 ;
   unsigned int result;
   a = a - 6;
   result = A + b;
   cout << result;
}
```

الفصل الثالث :

الإدخال و الإخراج (Input and Output)

توفّر المكتبة iostream التعليمتين cin و cout حيث تمّ تصميمُهُمَا من أجل التخاطب بين المستخدم والحاسب، وذلك أنّ العمليّة cout تقوم بالطباعة على الشاشة بينما تقوم العملية cin بإدخال القيم التي يحتاجها البرنامج من لوحة المفاتيح.

تعليمة الإخراج (cout)

تعاملنا مع هذه التعليمة منذ أوّل درس لنا في C++ لذلك فأنا واثق بأنّك تدرك وظيفتها، ولكنَّه من الأفضل أن أقدّم إليك تذكيراً مع شرحٍ لائقٍ بصاحبة الجلالة cout.

تعمل هذه التعليمة على إخراج قيم المتغيّرات والثوابت التي تأتي مباشرةً بعد معامل الحشر **>>**، ولها الشكل التالي:

```
cout<< "Islam is a way of life" ; // print  Islam is a way of life  on screen

cout<< 120 ;                      // print number 120 on screen

cout<< x ;                        // print  the content of variable x on screen
```

مالذي تفعله هذه التعليمات؟

1- **التعليمة الأولى :** تطبع هذه التعليمة العبارة **Islam is a way of life** المحصورة بين علامتي التنصيص " " على شاشة الخرج. **(عندما نرغب بطباعة نص ما فعلينا أن نضعه بين قوسي التنصيص وكذلك الأمر عند إسناد نص إلى متغيّر نصي لذلك فالعبارتين الآتيتين مختلفتين تماماً).**

```
cout<< "Hello" ; // Hello
```
تطبع العبارة بين قوسي التنصيص وهي الكلمة **Hello**

```
cout<< Hello ;  // Hello
```
تطبع قيمة المتغيّر

2- **التعليمة الثانية:** تطبع الرّقم **120** على شاشة الخرج.
3- **التعليمة الثالثة:** تطبع القيمة المخزّنة في المتغيّر x على شاشة الخرج.

تستطيع أن تستخدم أكثر من معامل حشر واحد **>>** في نفس التعليمة:

```
cout<< "The peace upon you, " << "I am " << "a C++ sentence. " ;
```

هذه العبارة تطبع الجملة التالية:

The peace upon you, I am a C++ sentence.

تكمن الفائدة من استخدام أكثر من معامل حشر في نفس التعليمة في طباعة عبارة مركّبة من قيم ثابتة ومتغيّرات كما في المثال التالي:

```
cout<<"I am "<< age << " years old and my zipcode is "<< zipcode ;
```

لو افترضنا أن قيمة المتغيّر age هي 20 والمتغيّر zipcode يحمل القيمة 60094 عندها فإنّ ناتج التعليمة السابقة سيكون:

I am **20** years old and my zipcode is **60094**

يجب أن تعلم أنّ التعليمة cout لا تقوم بإضافة رمز انتهاء السطر \n **(\new line)** فهي تطبع على سطر واحد. لذلك عندما ترغب في الطباعة على سطر جديد فعليك أن تضيف هذا الرّمز بنفسك. لتوضيح هذه المسألة:

لنقل إنّنا نريد أن نطبع الشكل التالي:

Go forward young Muslims, wherever you are
In the shadow of the sun or by the light of a star

عندها قد نكتب التالي:

```
cout<< "Go forward young Muslims, wherever you are";
cout<< " In the shadow of the sun or by the light of a star";
```

لكنّ المفاجأة عندما نشغل البرنامج فشكل الطباعة سيكون كالآتي:

Go forward young Muslims, wherever you are In the shadow of the sun or by the light of a star

وهذا ما لا نريده فالعبارتان على سطر واحد بينما نريد أن تكون كل واحدة على سطر، والحل يكمن كما قلنا للتوّ بإضافة الرمز **\n** في نهاية التعليمة الأولى لتصبح بالشكل:

```
cout<< " Go forward young Muslims, wherever you are\n";
```

عندها سيعلم المترجم أنّك تريد منه أن يبدأ سطراً جديداً، ثمّ يقومَ بتنفيذ التعليمة الثانية:

```
cout<< " In the shadow of the sun or by the light of a star";
```

هناك تعليمة بديلة عن رمز الانتهاء **\n** وهي التعليمة **endl** **(end line)** وعندها سيكون شكل البرنامج كالآتي:

```
cout<< " Go forward young Muslims, wherever you are"<<endl;
cout<< " In the shadow of the sun or by the light of a star";
```

سنقدّم مثالاً نذكر من خلاله بعض الطرق الممكنة لاستخدام هاتين التعليمتين:

```
#include<iostream>
using namespace std;
int main()
{
      /*    ****  1  ****    */
       cout<< "Welcome to you\n";
       cout<<"C++ You are elegant"<< endl;
      /*    ****  2  ****    */
       cout<< "Welcome to you "<< endl;
       cout<<"C++ You are elegant\n";
      /*    ****  3  ****    */
       cout<< "Welcome to you\nC++ You are elegant"<< endl;
      /*    ****  4  ****    */
       cout<< "Welcome to you"<< endl <<"C++ You are elegant\n";
       /* … */
       return 0;
}
```

```
Welcome to you
C++ You are elegant
```

تعليمة الإدخال input (cin)

تستخدم هذه التعليمة لإدخال القيم المطلوبة من **لوحة المفاتيح**، ونستخدم لذلك معاملاً يدعى معامل الإدخال **>>** ثمّ نتبعه باسم المتغيّر الذي نرغب بإعطائه القيمة المقروءة من لوحة المفاتيح، وذلك كما يلي:

```
int  age ;
cin>>  age ;
```

مثال:

```
#include <iostream>
using namespace std;
int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

```
Please enter an integer value: 702
The value you entered is 702 and its double is 1404.
```

ملاحظة 1: قد يكون المستخدم أحد أهمّ العوامل التي تسبب فشل برنامجك ففي المثال السابق طلبنا من المستخدم أن يدخل رقماً صحيحاً، **ولكن ماذا لو أدخل المستخدم اسمه؟** بالطبع لن تكون مسروراً لذلك.
طبعاً هناك حل لمثل هذه المشاكل ولكنّنا لن نتحدث عنه هنا وإنّما أردنا التنبيه لمثل هذه المشاكل حتى تضعها في حسبانك.

ملاحظة 2: يمكن أن نستخدم التعليمة `cin` لإدخال أكثر من قيمة كما يلي:

```
cin >> a >> b ;
```

بعض الرموز الخاصة

\n	New line	سطر جديد
\r	Enter	يكافئ الضغط على زر
\t	Tabulation	مسافة جدولة أفقية
\b	Back space	تراجع
\a	Beep	يصدر صوتاً
\'	'	تطبع الرمز
\"	"	تطبع الرمز

تمارين

1. مالذي تطبعه التعليمة التالية:

```
cout<<"*\n**\n***\n****\n***** " ;
```

2. اكتب برنامج يقرأ درجة الحرارة بالسيليسوس (c)، ثمّ يحولها إلى الفهرانهايت وفق العلاقة

$$f = \frac{9}{5}.c + 32$$

3. اكتب برنامج يقرأ درجة الحرارة بالفهرانهايت (f)، ثمّ يحولها إلى مئوية وفق العلاقة

$$c = \frac{5}{9}(f - 32)$$

4. اكتب برنامجاً يقرأ تسارع جسم ما a و الزمن t الذي استغرقه في الحركة ثمّ يحسب:

المسافة المقطوعة $d = \frac{1}{2}.a.t^2$ meters

و السرعة النهائية $v = a.t$ $m.\sec^{-1}$

5. اكتب برنامجاً يحسب قيمة التابع التالي

$$f(x) = 5 - x^2 \quad ; x \epsilon R$$

6. إذا علمت أن مجموع الأعداد من 1 إلى العدد n يعطى بالعلاقة التالية

$$sum = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

اكتب برنامجاً باستخدام هذه العلاقة لحساب مجموع الأعداد من 1 إلى n التي يدخلها المستخدم ثمّ احسب المتوسط الحسابي لها $average = \frac{sum}{n}$.

7. اكتب برنامجاً يطبع العبارة: 'Just remember "Allah created You'.
8. اكتب برنامجاً يحسب المتوسط الحسابي لثلاثة أعداد يدخلها المستخدم.

الفصل الرابع :

الثوابت (Constants)

في كثير من الأحيان نحتاج إلى أرقام أو نصوص ثابتة لاستخدامها في برامجنا. قد لا يكون ذلك واضحاً في البرامج العادية أو البسيطة لكنّه يتّضح في البرامج الكبيرة التي تتعامل مع الدوالّ الرياضيّة والرسوم وغيرها.

ولنأخذ كمثال الرقم المعروف π الذي كثيراً ما يُستخدم في الحسابات الرياضية. نعلم أنّه كلّما زادت دقّة الأرقام العشريّة زادت دقّة الحسابات، ولنفترض أنّنا سنستخدم الرقم π في برنامجنا عدّة مرّات فهل سنكون مضطرّين لكتابة هذا الرّقم عدّة مرّات. إنّ هذه الطريقة تعني زيادة احتمال وقوع أخطاء فقد نخطئ بكتابة الرّقم ممّا يعني خطأً حسابياً قد يتسبب بكارثة كبيرة في البرامج الهندسيّة أو الإنشائيّة لذلك وفرّت لغات البرمجة ما يسمّى بالثوابت فأنت تصرح عن الثابت وتعطيه القيمة التي سيأخذها والتي **لا يمكن أن تتغير طوال فترة التنفيذ** ثمّ تقوم باستدعاء الثابت في كل مرّة تودّ استخدامه فيها.

كيف نصرح عن الثوابت؟

هناك طريقتان :

- ✓ **الأولى** const:

```
const float pi = 3.14159265 ;

const char tab = '\t' ;
```

- ✓ **الثانية** #define:

```
#define  pi  3.14159265

#define  NewLine  '\n'
```

في هذه الحالة نستخدم التعليمة #define ثمّ نذكر اسم الثابت ثمّ القيمة التي سيأخذها. **انتبه**: في هذه الطريقة لا يوجد فاصلة منقوطة في نهاية التعليمة.

لعلّك لاحظت أنّنا لم نقم بتحديد نوع الثابت في الحالة الثانية كما فعلنا في الحالة الأولى.

الحالة الأولى	الحالة الثانية

`#include<iostream>`
`using namespace std;`
`const double pi = 3.14159265;`
`int main()`
`{`
`  float r ;`
`  r = 0 ;//initial to 0`
`  cout<<"enter circle radius plz: ";`
`  cin>> r ;`
`  cout<< "circle area = "`
`      << 0.5 * pi * r * r << endl;`
`return 0 ;`
`}`
	`#include<iostream>`
	`using namespace std;`
	`#define pi 3.14159265`
	`int main()`
	`{`
	`  float R ,r ;`
	`  r = 0 ;//initial to 0`
	`  cout<<"enter circle radius plz: ";`
	`  cin>> r ;`
	`  cout<< "circle area = "`
	`      << 0.5 * pi * r * r << endl;`
	`return 0 ;`
	`}`

الفصل الخامس:

المعاملات (Operators)

تعرّفنا في الدرس السابق على المتغيّرات والثوابت وقلنا: "إنّ المتغيّرات هي الأماكن التي يخزّن الحاسب فيها البيانات وهي مستعدّة لتغيير قيمتها أثناء تنفيذ البرنامج، بينما الثوابت فهي أماكن في الذاكرة تحتفظ بقيمة وحيدة لا يمكن تغييرها طوال فترة تشغيل البرنامج"، وتعرّفنا أيضاً على كيفية التّصريح عن كلّ منهما، ولكن **كيف يمكننا التحكم بالقيم التي تحتفظ بها المتغيّرات؟**

من أجل هذا تزوّدنا C++ بالمعاملات. والتي تُمثَّل في هذه اللغة بمجموعة من الكلمات المفتاحية المحجوزة والرموز الخاصة وهذه المعاملات تعتبر من أساسيّات اللغة لذلك لابدّ لك عزيزي القارئ من التعرف عليها و التعامل معها بشكل جيد.

1. الإسناد (=)

نستخدم معامل الإسناد = كي نعطي المتغيّر قيمة من نوع يقبله**(كأن نضع رقم صحيح في متغيّر من النوع العشري مثلاً)**

```
float a = 3 ;
```

أسند العدد الصحيح 3 إلى المتغيّرالعشري a. يقوم معامل الإسناد بإسناد القيمة على يمينه إلى المتغيّر الذي على يساره حيث أنّ القيمة على اليمين يمكن أن تكون متغيّراً أو عدداً ثابتاً أو نتيجةً لعدد من العمليات على قيم موافقة لنوع المتغيّر على يسارمعامل الإسناد. والإسناد المعاكس ليس ممكناً.

أمثلة:

```
int a , b ;              // a : ?    b : ?
a = 10 ;                 // a : 10   b : ?
b = 4 ;                  // a : 10   b : 4
a = b ;                  // a : 4    b : 4
b = 7 ;                  // a : 4    b : 7
a = 2 + ( b = 5 ) ; // 1) b = 5 ;  2) a = 2 + b ;  3) a = 7 ;
```

أسند القيمة 5 إلى المتغيّر b ثمّ أسند المجموع (2 + b) إلى المتغيّر a.

```
a = b = c = 5 ; //1) c =5 ;  2) b = c ; : c = 5  3) a = b ; : b = 5 ;
```

أسند القيمة 5 إلى المتغيّرات الثلاثة a , b , c

2. المعاملات الرياضية (+, -, *, /, %)

تدعم **C++** خمس معاملات رياضيّة وهي:

العملية	الوظيفة
+ Addition	يقوم بجمع القيمتين على طرفيه
- subtraction	يقوم بطرح القيمتين على طرفيه
* multiplication	يقوم بضرب القيمتين على طرفيه
/ division	يقوم تقسيم القيمة التي علي يساره على القيمة التي التي على يمينه
% module	يعيد باقي قسمة القيمة على يمينه على القيمة على يساره

3. معاملات الإسناد المركبة (+=, -=, *=, /=, %=, &=, ^=, |=)

value += increase;	**value = value + increase;**
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

وكذلك باقي المعاملات.

4. معاملات الزيادة و النقصان (++, --)

a++ ;	**a+= 1 ;**	**a = a + 1 ;**
a-- ;	a-= 1 ;	**a = a - 1 ;**

ملاحظة:

يمكن استخدام هذين المعاملين بطريقتين مختلفتين و المثال التالي يبين ذلك:

الطريقة الأولى	الطريقة الثانية	الفرق

b = 3;
a = b++ ;
// a = 3 , b = 4
 	
تُسند **b** إلى **a** ثمّ تزداد **b** بمقدار 1
	b = 3;
	a = ++b;
	// a = 4 , b = 4
	
	تزداد **b** بمقدار 1 ثمّ تُسند **b** إلى **a**
		في كلا الطريقتين بدأنا مع b = 3
		في الطريقة**1** تصبح a = 3 ثمّ تزداد قيمة b فتصبح b = 4 أما في الطريقة**2** تزداد قيمة b ثمّ تصبح a = b = 4

5. معاملات المقارنة (==, !=, >, <, >=, <=)

تستخدم هذه المعاملات عادةً للمقارنة بين عبارتين برمجيتين، وتعيدُ إحدى القيمتين **true** أو **false** وذلك حسب نتيجة المقارنة.

المعامل	وصفه	مثال
==	يساوي	(5 == 7) ستعيد **false**
!=	لا يساوي	(2 != 3) ستعيد **true**
<	أصغر من	(8 < 4) ستعيد **false**
>	أكبر من	(8 > 4) ستعيد **true**
<=	أصغر أو يساوي	(6 <= 6) ستعيد **true**
>=	أكبر أو يساوي	(6 >= 6) ستعيد **true**

طبعاً نستطيع استخدام متغيّرات أو تعابير للمقارنة بينها عوضاً عن استخدام أرقام ثابتة فقط.
لنفترض أنّ لدينا

a = 2, b = 3, c = 6

(a == 5)	تعيد false
(a * b >= c)	تعيد true
(b + 4 > a * c)	تعيد false
((b = 2) == a)	تعيد true

كن حذراً: استخدام المعامل = مرّة واحدة فقط يعني الإسناد أمّا استخدامه مرّتين == فيعني المقارنة والسطر الرابع من المثال السابق يوضح ذلك.

في العديد من المترجمات كما في لغة **C** مثلاً تعيد المعاملات المنطقيّة قيماً عدديّة صحيحة حيث تعيد القيمة 0 بدلا من **false** وتعيد قيماً مختلفة عن 0 بدلاً من **true** **(في الحالة العامة تعيد القيمة 1)**.

6. المعاملات المنطقية (!, &&, ||)

المعامل not ! :

!(5 == 5)	يعيد **false** لأن 5 تساوي 5
!(4 <= 6)	يعيد **true** لأن 6 ليست أصغر من أو تساوي الـ 4
!true	يعيد **false**
!false	يعيد **true**

المعاملان && و || :

&& يمثّل معامل الربط المنطقي (وَ) **and**

|| يمثّل معامل الربط المنطقي (أو) **or**

a	b	a && b	a || b	مثال ((a) && (b)) , ((a) || (b))
true	true	true	true	((5 == 5) && (3 < 6)), ((5 == 5) || (3 < 6))
true	false	false	true	((5 == 5) && (3 > 6)), ((5 == 5) || (3 > 6))
false	true	false	true	((5 != 5) && (3 < 6)), ((5 != 5) || (3 < 6))
false	false	false	false	((5 != 5) && (3 > 6)), ((5 != 5) || (3 > 6))

7. المعامل الشرطي (?)

يعمل هذا المعامل على تقييم تعبير ما ويعيد إحدى قيمتين وفقاً لتقييم التعبير. وله الشكل الآتي:

condition ? result1 : result2

وهو يكافئ التعليمة التالية

```
if (condition == true )
   return result1;
else
   return result2;
```

أي: إذا كان الشرط محققاً فإنّ المعامل سيعيد النتيجة الأولى وإلّا سيعيد النتيجة الثانية.

مثال:

```
a = 7, b = 3;
(a == b) ? return 1 : 2 ;
```

سيعيد العدد 2 لأن الشرط غير محقق (7 لا تساوي 3).

8. معامل تحويل البيانات

يسمح لك هذا المعامل بتحويل متغيّر من نوع إلى آخر وهناك عدّة طرق لفعل ذلك والأكثر شهرةً منها هو المعامل التالي: **(type)** أو **type (value)**

مثال:

```
Int  i ;
float  pi = 3.14 ;
i = (int)pi ; // pi = 3.14 ,  i = 3
```

السطر الثالث من المثال السابق يقوم بتحويل قيمة المتغيّر العشري إلى عدد صحيح، و يمكن أن يكتب بالشكل:

```
i = int(pi) ;
```

9. المعامل sizeof()

يقبل هذا المعامل وسيطاً واحداً إمّا أن يكون نوعاً من أنواع البيانات المألوفة أو متغيّراً بحدّ ذاته، وهو يعيد حجم الوسيط بالبايت.وله الشكل:

```
int a  = sizeof(char) ;
```

سيعيد هذا المعامل القيمة 1 لأن النوع **char** يأخذ 1 بايت في الذاكرة كما في الجدول في الدرس السابق.

أولويّات تنفيذ المعاملات

ليكن لدينا العبارة البرمجية التالية:

```
a = 5 + 7 % 2 ;
```

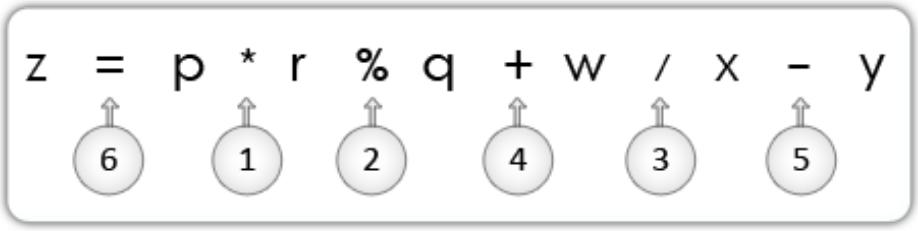
ما هي نتيجة هذا التعبير برأيك؟ طبعا جوابك هو 6 **ولكن لماذا؟**

يخضع ترتيب تنفيذ المعاملات للجدول التالي:

الأولوية	العملية	جهة الأولويّة
1	:: معامل المدى	من اليسار
	( ) الأقواس , sizeof()	من اليسار
2	* , / , %	من اليسار
3	++ , -- ~ ! * , & (type) - , +	من اليمين
4	< , > , <= , >=	من اليسار
5	== , !=	من اليسار
6 7	& , ^ , | && , ||	من اليسار
8	: ?	من اليمين
9	= , += , -= , *= , /= , %= , |= , ^= , &=	من اليمين
10	,	من اليسار

مثال:

z = p*r %q+ w/ x - y;



تمارين

1. ما هي قيمة العبارات التالية:

```
int a = 0 ;
long b = 1, c = 2, d = 3 ;
bool e = true , r = true , t = false ;
```

a) a += (b = c = d - 5) ;
b) d %= 2 * c / ++b ;
c) a = d – (b += (c * (1 – a))) ;
d) e = !e ;
e) r = e || t ;
f) t = e && r ;
g) e = !((e && r) || (e && t)) ;

2. حدد أولويات العمليات في التعابير التالية ثمّ أوجد قيمة **x** في كل مرة:

a) x = 7 + 3 * 6 / 2 – 1 ;
b) x = 2 % 2 + 2 * 2 – 2 / 2 ;
c) x = (3 * 9 * (3 + (9 * 3 / (3)))) ;
d) c = 2*b – 3/d + a++;

3. اكتب ناتج الجمل التالية:

a) cout<< ++a ;
b) cout<< a++ ;
c) cout<< a++ / 2 ;
d) cout<< ++a / 2 ;

الفصل السادس:

بنى التحكم (Control Structures)

إنّ البرنامج ليس عبارة عن مجموعة من تعليمات الإسناد أو المقارنات أو الإدخال والإخراج التي تعلّمناها حتّى الآن، فنحن وكما تعلم نتحدّث عن البرمجة التي إنّما نستخدمها كي تساعدنا في حل مشاكلنا اليومية ومن الأشياء المهمّة التي نحتاج إليها هي اتخاذ القرارت المناسبة لإجراء عمليات معيّنة دون أخرى. توفّر C++ مجموعة بُنى برمجية نسميها بنى التحكم، و **البنية البرمجية** هي عبارة عن مجموعة من التعليمات يفصل بين كل تعليمتين متتاليتين فيها فاصلة منقوطة، وتكون هذه المجموعة محصورة بين قوسي البنية **{ }.** معظم البنى التي سنتعرّف عليها الآن تحتاج إلى عبارة عامّة كوسيط يتمّ تمريره إليها. في حال كانت البنية تحوي تعليمة واحدة فقط فإنّها لا تحتاج إلى القوسين **{ }** أمّا إذا كانت البنية تحوي أكثر من تعليمة فإنّها تحتاج إليهما.

1) بنية الشرط condition structure

كثيراً ما نتّخذ قراراتنا باستخدام الشرط فعندما تقول: "إذا حصل هذا الشيء عندها سأفعل كذا وكذا". فإنّك تستخدم شرطاً تبني على تحقّقه قراراً في تنفيذ مجموعة من الأشياء أو عدم تنفيذها. لنتعرّف كيف نستطيع أن نفعل ذلك باستخدام C++

```
if ( condition )
   statement ;  // تعليمة واحدة فقط
```

أو

```
if ( condition )
{
   statements... // أكثر من تعليمة
}
```

حيث أنّ التعليمات التي داخل هذه البنية لا يتمّ تنفيذها إلّا إذا كان الشرط الذي بين القوسين (condition) محقّقاً.

مثال: نريد من المستخدم أن يدخل رقماً ما

فإذا كان **(الرقم يساوي 100)** عندها ***سنطبع العبارة:*** ***" x is 100 "***

```
if ( x == 100 )
        cout << " x is 100 " ;
```

في هذا المثال لدينا تعليمة واحدة فقط لذلك لم نستخدم القوسين {}، و لو أردنا أن نضيف تعليمات أخرى إلى هذه البنية عندها **يجب** أن نستخدم القوسين {} أي:

```
if ( x == 100 )
{
        cout << " x is " ;
        cout << x ;
        statements ...
}
```

هناك نوع آخر من الشرط نستخدمه في حياتنا فكثيراً ما نقول:" إذا حصل هذا الشيء عندها سأفعل هذه الأشياء و إلا سأفعل أشياءً أخرى ".

```
if ( x == 100 )
        statement1 ;
else
       statement2 ;
```

في المثال السابق نريد من المستخدم أن يدخل رقماً ما
فإذا كان **(الرقم يساوي 100)** عندها ***سنطبع العبارة: " x is 100 "***
و إلّا سنطبع العبارة : " x is not 100 "

```
if (x == 100)
 cout << "x is 100";
else
 cout << "x is not 100";
```

لايزال هناك حالة أخيرة للشرط وسنحتاجها بكثرة أيضاً وهي من الشكل:

إذا كان **(شرط1)** عندها ***اِفعل الأشياء 1***
و إلا إذا كان **(شرط2)** عندها ***اِفعل الأشياء 2***
.
.
.
وإلا ***اِفعل الأشياء n***

```
if (condition 1)
{
   statements 1 ;
}
else if (condition 2)
   statement 2 ;
.
.
else
   statement n;
```

مثال:

```
if (x > 0)
        cout << "x is positive";
else if (x < 0)
        cout << "x is negative";
else
        cout << "x is 0";
```

2) بنى التكرار loops or repetition structures

أثناء كتابة برامجنا قد نحتاج إلى تكرار مجموعة من التعليمات (عدداً من المرات أو حتّى يتحقّق شرطٌ ما) فعندما تريد أن تطبع الأرقام من 10 إلى 1 تنازليّاً فما الذي ستصنعه هل ستكتب 10 مرّات التعليمة cout لتخرج كل مرّة رقماً أقلّ من سابقه ماذا لو أردت طباعة 1000 رقم أو أكثر. إنّ استخدام البنى التكرارية سيسهّل هذا الأمر كثيراً. هناك ثلاثة أشكال من البنى التكرارية سنتعرف عليها تباعاً وهي:

البنية الأولى : مادام (الشرط محققاً) *نفذ التعليمات التالية*

while (**condition**) statements

البرنامج التالي يستخدم هذه الحلقة لطباعة الأرقام من 10 إلى 1 تنازليّاً كما هو موضح:

```
// custom countdown using while
#include <iostream>
using namespace std;
int main ()
{
  int n = 10 ;
  while (n>0)
  {
    cout<< n << ", ";
    n--;
  }
  cout<< "FIRE!";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

بدأ البرنامج من التابع main()، ثمّ صرّحنا عن المتغيّر n الذي حمل القيمة 10، ثمّ استخدمنا البنية while التي **تفحص الشرط n > 0 أولاً**

- فإذا كان الشرط محققاً فإنّها تقوم بما يلي:
  1- تطبع العدد n.
  2- تنقص العدد n بمقدار 1.
  3- نهاية البنية ثمّ العودة للتحقّق من الشرط مرّة أخرى فإذا كان محقّقاً عادت إلى الخطوة 1.
- و إلا سينتقل المترجم إلى التعليمات التي تقع بعد هذه البنية وهي طباعة العبارة FIRE!.

ثمّ نهاية البرنامج.

البنية الثانية: نفّذ *التعليمات التالية* مادام (الشرط محققاً)

do statements while (condition);

البرنامج التالي يستخدم هذه الحلقة لطباعة الأرقام من 10 إلى 1 تنازليّاً كما هو موضح:

```
// custom countdown using while
#include <iostream>
using namespace std;
int main()
{
  int n = 10 ;
  do
  {
    cout << n << ", ";
    n--;
  } while (n>0);
  cout << "FIRE!";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

بدأ البرنامج من التابع main()، ثمّ صرّحنا عن المتغيّر n الذي حمل القيمة 10، ثمّ استخدمنا البنية do التي **تنفذ التعليمات أولاً** وهي تقوم بالآتي:

1. تطبع العدد n.
2. تنقص العدد n بمقدار 1.
3. نهاية البنية ثمّ التحقق من الشرط
   - فإذا كان الشرط $n > 0$ محقّقا فإنّها تعود إلى الخطوة 1.
   - و إلا $n \leq 0$ سينتقل المترجم إلى خارج هذه البنية ويطبع عبارة FIRE!.

ثمّ نهاية البرنامج.

تنبيهات حول البنيتين السابقتين

1- **يجب** أن تضع شرطاً قابلاً للتحقق كي تضمن انتهاء الحلقة (البنية) التكرارية لأنّ عدم تحقق الشرط يعني الدخول في حلقة لا نهائية وبالتالي لن يتوقف البرنامج عن العمل.
فمثلاً لو وضعنا n++; بدلاً من n--; في أحد المثالين السابقين فإنّ الحلقة لن تنتهي أبداً.

2- هاتان البنيتان لا تمتلكان عدّادات زيادة أو نقصان في داخلهما لذلك فهما تقبلان الشروط المنطقية و العددية و**يجب** عليك أن تجد طريقةً تستطيع من خلالها أن تعرف فيما إذا تحقّق الشرط (المنطقي أو العددي) أوّلا، في المثال السابق استخدمنا المتغيّر n لهذه المهمة الذي أخذ يتناقص مع كل تكرار حتّى تحقق شرط التوقف $n \leq 0$.

3- البنية while تقوم باختبار الشرط **أوّلاً ثمّ تنفّذ** التعليمات التي بداخلها لو كان الشرط محققاً. أي أنّه قد لا يتمّ تنفيذ تعليمات هذه البنية ولا مرّة في حال كان الشرط غيرَ محققٍ. بينما تعمل البنية do بعكس أختها فهي تنفّذ تعليمات بنيتها **مرّةً أولى** ثمّ تتحقّق من الشرط فإذا تحقّق أعادت تكرار العمليات التي بداخلها و إلا خرجت من البنية و تابعت البرنامج.

البنية الثالثة:

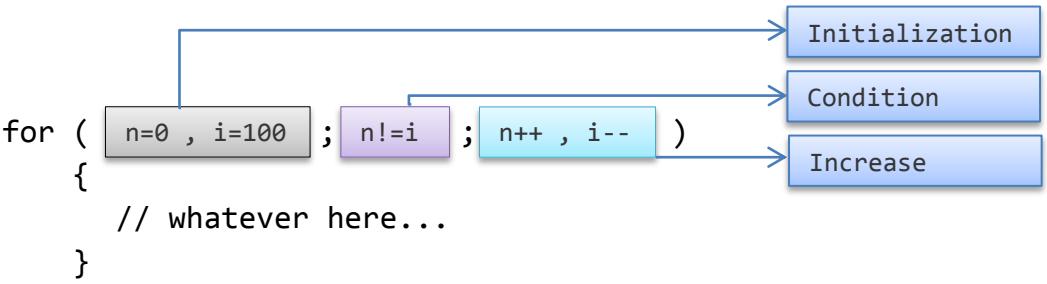
```
for (initialization ; condition ; increase)
      statement;
```

تتمتع هذه البنية بشهرة كبيرة لأنها من أسهل و أكثر الحلقات التكرارية استخداماً، فهي تعمل على تكرار التعليمات التي بداخلها ابتداءً من قيمة معيّنة وحتّى قيمة الانتهاء التي **يجب** أن تكون محدّدة، وهي تحتوي على عدّاد بداخلها يستطيع المبرمج أن يتحكّم بمقدار زيادته أو نقصانه.
البرنامج التالي يستخدم هذه الحلقة لطباعة الأرقام من 10 إلى 1 تنازليّاً كما هو موضح:

```
// custom countdown using for
#include <iostream>
using namespace std;
int main ()
{
  for ( int n=10 ; n>0 ; n-- )
      {
        cout << n << ", ";
      }
  cout << "Fire!";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

يمكن أن تضمّ هذه الحلقة عدّادين بنفس الوقت فتأخذَ الشكل التالي:



تبدأ هذه الحلقة بالقيم الابتدائية التالية `n = 0` و `i = 100` وتستمرّ مادام `n != i` أي أنّ الحلْقة ستتوقف بعد 50 تكراراً لأنّ `i = n = 50`

كما يمكن أن تكتب هذه الحلقة بالأشكال التالية أيضاً:

```
for (;n<10;)      // لم نعطي العداد قيمة ابتدائية ولم نحدد مقدار خطوة الزيادة

for (;n<10;n++)   // لم نعطي العداد قيمة ابتدائية فقط

for(; ;)          // break ; حلقة لا نهائية يمكن الخروج منها باستخدام التعليمة
```

يمكن أن نستخدم الحالة الأخيرة عندما لا نستطيع أن نعرف قيمة التوقف أو عندما نرغب باستخدام شرط منطقي بدلاً من العدّاد كما في المثال التالي:

مثال:

البرنامج التالي يقوم بقراءة رقم من لوحة المفاتيح وطباعته على الشاشة ويكرّر هذه العملية حتى يدخل المستخدم الرقم 0

```
// number echoer
#include <iostream>
using namespace std;
int main ()
{
  unsigned long n;
  cout << "Enter  number (0 to end): ";
  cin >> n;
  cout << "You entered: " << n << "\n";
  while(n != 0)
  {
    cout << "Enter number (0 to end): ";
    cin >> n;
    cout << "You entered: " <<n<< "\n";
  }
  return 0;
}
```

```
Enter  number (0 to end): 5
You entered: 5
Enter number (0 to end): 0
You entered: 0
```

```
// number echoer
#include <iostream>
using namespace std;
int main ()
{
  unsigned long n;
  do
  {
     cout << "Enter number (0 to end): ";
     cin >> n;
     cout << "You entered: " << n<< "\n";
  } while (n != 0);
  return 0;
}
```

```
Enter  number (0 to end): 5
You entered: 5
Enter number (0 to end): 0
You entered: 0
```

```
// number echoer
#include <iostream>
using namespace std;
int main ()
{
  unsigned long n;
  for (; ;)
  {
    cout << "Enter number (0 to end): ";
    cin >> n;
    cout << "You entered: " << n << "\n";
    if (n== 0)
       break;
  }
  return 0;
}
```

```
Enter  number (0 to end): 5
You entered: 5
Enter number (0 to end): 0
You entered: 0
```

3) التعليمة break: تستخدم هذه التعليمة للخروج من البنية التي كُتِبت فيها كما في الحالة الثالثة من المثال السابق، فنحن نريد أن نخرج من الحلقة(البنية)التكرارية بمجرّد أن يتحقّق الشرط `(n == 0)`

4) التعليمة continue: تستخدم هذه التعليمة عندما نرغب بأن يتجاهل البرنامج التعليمات التي تلي التعليمة `continue` والعودة إلى بداية البنية البرمجية التي كتبت فيها هذه التعليمة.

فلو أردنا أن نكتب برنامجاً يطبع الأعداد بين 1 و 10 ماعدا الرقم 5 باستخدام هذه التعليمة فسنكتب الآتي:

```
#include <iostream>
using namespace std;
int main()
{
for (int n=10; n>0;n--)
{
if (n==5)
continue;
cout << n << ", ";
}
cout << "FIRE!\n";
return 0;
}
```

```
10, 9, 8, 7, 6, 4, 3, 2, 1, FIRE!
```

البداية من التابع main() يدخل المترجم الحلقة التكرارية وعندها تكون قيمة العدّاد n = 10 ثمّ يختبر الشرط (n == 5) وبالطبع لن يكون الشرط محقّقاً لأنّ 10 != 5 وعندها يتجاهل التعليمة الموجودة داخل بنية الشرط if ثمّ ينتقل إلى التعليمة التالية وهي تعليمة طباعة العدد n وبعدها تنتهي الحلقة للمرّة الأولى فيعود المترجم إلى معامل النقصان في الحلقة ويغير قيمة العداد n بحسب قيمة النقصان المطلوبة (في حالتنا 1) ثمّ يختبر الشرط n>0 فإن كان محققاً فسيعود إلى داخل الحلقة مرّة أخرى ويستمر في هذه العمليات حتى تصبح قيمة n = 5 عندها سيختبرالشرط n == 5 والذي أصبح محققاً لذلك سينفّذ التعليمة الموجودة داخل تعليمة الشرط وهي التعليمة continue والتي ستجبر المترجم على تجاهل التعليمات التي تليها مما يعني تجاهل التعليمة cout << n << ", "; والعودة إلى بداية الحلقة لإنقاص العدّاد واختبار الشرط ثمّ الدخول في الحلقة من جديد وهكذا حتى نهاية الحلقة.

5) التعليمة goto: بهذه التعليمة تستطيع إجبار المترجم على القفز من السطر الذي هو فيه إلى السطر الذي تريده ليكمل التنفيذ من المكان الذي أرسلته إليه. لاستخدام هذه التعليمة نحتاج إلى لافتة label لاستعمالها كدليل على السطر الذي سنرسل المترجم إليه، وهذه اللافتة يجب أن تتبع بالنقطتين (:).

مثال:

```
 // goto loop example
#include <iostream>
using namespace std;
int main ()
{
  int n=10;
  loop:// goto label
  cout << n << ", ";
  n--;
  if (n>0)
     goto loop;
  cout << "FIRE!\n";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

في المثال السابق استطعنا أن نطبع الأعداد من 10 إلى 1 دون استخدام حلقات تكرارية وذلك باستخدام goto. كالمعتاد بدأ البرنامج من التابع main()، ثمّ صرّحنا عن المتغيّر n الذي أخذ القيمة 10 ثمّ بعد ذلك صرّحنا عن اللافتة loop: وبعدها طبعنا قيمة n ثمّ أنقصناها بمقدار 1 بعد ذلك وضعنا شرطاً لنعرف إذا ما كانت n>0 أو لا فإذا كانت n > 0 عندها سنعيد المترجم إلى السطر الذي يحوي اللافتة loop: ليعيد العمليات السابقة كلّها حتى تصبح n = 0 عندها لن يرجع المترجم إلى اللافتة وسيكمل البرنامج ليطبع العبارة FIRE! ثمّ ينتهي البرنامج.

6) التعليمة switch: تشبه هذه التعليمة في وظيفتها التعليمة if فهي تختبر قيمة المتغيّر الذي يتمّ تمريره لها كوسيط ثمّ تحدد العمل الذي ستقوم به من خلال قيمة المتغيّر.

ولها الشكل التالي:

```
switch(variable)
{
case value1:
        statements;
        break;
//other cases here…
default:   //optional
        statements;
        break;
}
```

مثال: نريد برنامجاً يخبر المستخدم عن اليوم الذي يقابل الرقم الذي أدخله المستخدم.
أولاً يجب أن نقرأ الرقم ونخزّنه في متغيّر عددي صحيح و موجب(لأن الأرقام هي بين 1 - 7)
ثانياً سنختبر قيمة المتغيّر ثمّ نقابله بالنصّ المناسب فالرقم 1 يقابل يوم الجمعة Friday وهكذا...
ثالثاً سنطبع اسم اليوم المناسب على شاشة الخرج.

```
// switch
#include<iostream>
using namespace std;
int main()
{ unsigned short dnum ;
cout<< "Enter number of day(1-7): ";
cin>> dnum;
cout<< "\n";
switch(dnum){
case 1: // if dnum = 1
cout << "the day is Friday";
break;
case 2: // if dnum = 2
cout << "the day is Saturday";
break;
case 3: // if dnum = 3
cout << "the day is Sunday";
break;
case 4: // if dnum = 4
cout << "the day is Monday";
break;
case 5: // if dnum = 5
cout << "the day is Tuesday";
break;
case 6: // if dnum = 6
cout << "the day is Wednesday";
break;
case 7: // if dnum = 7
cout << "the day is Thursday";
break;
default: // if dnum < 1 or dnum> 7
cout << "Sorry we're closed ";
break;
}
cout<< "\n";
return 0;
}
```

```
// if
#include<iostream>
using namespace std;
int main()
{
unsigned short dnum ;
cout<< "Enter number of day(1-7): ";
cin>> dnum;
cout<< "\n";
 if(dnum == 1)
   cout << "the day is Friday";
 else if(dnum == 2)
   cout << "the day is Saturday";
 else if(dnum == 3)
   cout << "the day is Sunday";
 else if(dnum == 4)
   cout << "the day is Monday";
 else if(dnum == 5)
   cout << "the day is Tuesday";
 else if(dnum == 6)
   cout << "the day is Tuesday";
 else if(dnum == 7)
   cout << "the day is Thursday";
 else
   cout << "Sorry we're closed ";
 cout<<'\n';
return 0;
}
```

لنشرح ما حصل في التعليمة switch

كما عوّدتنا c++ بدأ البرنامج من التابع main() ثمّ صرّحنا عن المتغيّر dnum، ثمّ قرأنا الرقم وخزّنّاه في المتغيّر dnum، والآن جاء دور التعليمة switch التي أخذت المتغيّر dnum وسيطاً. بعد ذلك قمنا بذكر الحالات التي يمكن للمتغيّر أن يحملها وهي الأرقام {1,2,..,7} وذكرنا كل عنصر منها لوحده بعد الكلمة case ثمّ وضعنا التعليمة التي تناسب كل حالة على حدة وفي نهاية كل حالة كتبنا التعليمة break التي شرحناها سابقاً والتي لها أهمّية كبيرة في التعليمة switch فعندما يقارن المترجم قيمة المتغيّر الوسيط مع الحالة الأولى و تكون القيمتين متساويتين عندها سينفذ التعليمات الخاصة بهذه الحالة وعندما سينتهي منها نريد منه أن يخرج من التعليمة switch وألّا يختبر باقي الحالات لذلك وضعنا التعليمة break ولو لم نضعها لاختبر المترجم كل الحالات الباقية وهذا يعني مضيعةً للوقت وبطأ للبرنامج لا نريده أن يحصل طبعاً. أمّا التعليمة default فهي تخصّ باقي الاحتمالات الممكنة إذ قد يدخل المستخدم رقماً غير الأرقام المذكورة لذلك نريد من البرنامج أن يخبر المستخدم أنّه أدخل رقماً خاطئاً. جرّب البرنامجين، وستجد أن لهما نفس الناتج تماماً كما في الشكل:

```
Enter number of day: 5
the day is Tuesday
```

تمرين:

باستخدام إحدى البنى التكرارية أو باستخدام التعليمة goto عدّل البرنامج السابق ليكرّر الإدخال حتّى يدخل المستخدم رقماً من خارج المجال 1 إلى 7 حيث يخرج من البرنامج.

7) الحلقات المتداخلة nested loops: قلنا عند الحديث عن البنى التكرارية: "إنّها تمكّننا من تكرار مجموعة من التعليمات عدداً من المرّات". أي أنّه يمكننا أن نطبع الشكل التالي باستخدام حلقة تكرارية كما بيّنا من قبل:

```
1 2 3 4 5
```

```
#include <iostream>
using namespace std;
int main ()
{
  for(int j = 1 ; j<=5 ; j++)
     cout << j << " ";
  return 0;
}
```

```
1 2 3 4 5
```

لكن ماذا لو أردنا أن نطبع الشكل التالي و الذي يمثل عناصر مصفوفة ثنائية الأبعاد.

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

يمكن فهم هذا الشكل على أنّه طباعة الأعداد من 1 إلى 5 ثلاث مرّات متتالية على ثلاثة أسطر متتالية أيضاً أي أنّنا سنقوم بتكرار عمل الحلقة السابقة ثلاث مرّات كما في الشكل التالي:

```
// nested loop
#include <iostream>
using namespace std;
int main ()
{
  for( int i = 1 ; i<=3 ; i++){
      for(int j = 1 ; j<=5 ; j++)
          cout << j << " ";

      cout<<"\n";
  }
  return 0;
}
```

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

تبدأ الحلقة الأولى عندما `i = 1` وبما أنّ 1<3 سيتّجه المترجم إلى داخل الحلقة الأولى لينفّذ ما فيها من تعليمات فيجد أنّ أول تعليمة أمامَه هي الحلقة الثانية التي تبدأ من `j = 1` وتطبع الأرقام من 1 إلى 5 على نفس السطر حيث يعلق المترجم في هذه الحلقة حتّى ينتهي منها عندما `j = 6`، ثمّ يتّجه المترجم إلى التعليمة الثانية في الحلقة الأولى وهي `cout<<"\n";` والتي تعني إبدأ سطراً جديداً، ثمّ ينتهي التكرار الأول لهذه الحلقة، وتتمّ زيادة العدّاد `i` بمقدار 1 ليصبح `i = 2` ويتمّ اختبار الشرط مرّة أخرى وبما أنّ 2<3 فإنّ المترجم سيعود ويدخل الحلقة الأولى ليقوم بنفس الخطوات السابقة، فيعيد قيمة `j` إلى 1 ليعلق مرّة أخرى في الحلقة الثانية فيقوم بطباعة الأرقام من جديد على السطر الجديد الذي جاء من التعليمة `cout<<"\n";`، وعندما ينتهي من الحلقة الثانية يكمل الحلقة الأولى فيطبع سطراً جديداً، ثمّ يكرّر ذلك مرّة أخيرة حيث تكون `i = 3` وبعدها تصبح `i = 4` ممّا يعني أنّ المترجم سيغادر الحلقة دون أن ينفذها مرّةً رابعةً وعندها نحصل على الشكل السابق.

مثال:اكتب برنامجاً يطبع الشكل التالي.

```
*
* *
* * *
* * * *
* * * * *
```

```
// nested loop
#include <iostream>
using namespace std;
int main ()
{
  for( int i = 1 ; i<=5 ; i++)
  {
      for(int j = 1 ; j<=i ; j++)
          cout << "* ";
   /* for(int j = 1 ; j<=5 ; j++)
          if(j<=i) cout << "* "; */

      cout<<"\n";
  }
  return 0;
}
```

```
*
* *
* * *
* * * *
* * * * *
```

المثال السابق باستخدام الحلقة `while` مع الحلقة `for`

```
// nested loop
#include <iostream>
using namespace std;
int main ()
{
  int j = 1;
  for( int i = 1 ; i<=5 ; i++)
  {
      j = 1 ;
      while( j<=i )
      {
          cout <<"* " ;
          j++;
      }
      cout<<"\n" ;
  }
  return 0;
}
```

```
*
* *
* * *
* * * *
* * * * *
```

تمارين محلولة

1. اكتب برنامجاً يطلب من المستخدم أن يدخل رقماً ثمّ عمليّة حسابيّة من العمليّات الأربع المعروفة ثمّ يطلب رقماً آخرَ ثمّ يظهر النتيجة وفقاً للعمليّة التي أدخلها المستخدم (باستخدام switch).

الحل:

1. **الخوارزمية**
   a) اقرأ الرقم الأول n1.
   b) اقرأ العملية op.
   c) اقرأ الرقم الثاني n2.
   d) افحص العملية op
   - ✓ إذا كانت op = '+' اطبع المجموع
   - ✓ وإلا إذا كانت op = '–' اطبع الفرق
   - ✓ وإلا إذا كانت op = '*' اطبع الجداء
   - ✓ وإلا إذا كانت op = '/'
     - إذا كان الرقم الثاني لا يساوي الصفر $n2 \neq 0$ اطبع القسمة
     - و إلّا اطبع عبارة Infinite .
2. **الشيفرة**

```
#include<iostream>
using namespace std;
int main()
{
	float n1 , n2;
	char op  = '+';
	cout<<"Enter n1:"; cin>> n1;
	cout<<"\nEnter Operator of {+,-,/,*}: ";
	cin>>op;
	cout<<"\nEnter n2: "; cin>> n2;
	switch(op)
	{case '+':
		cout<< n1<<op<<n2<<" = "<<n1+n2;
		break;
	case '-':
		cout<< n1<<op<<n2<<" = "<<n1-n2;
		break;
	case '*':
		cout<< n1<<op<<n2<<" = "<<n1*n2;
		break;
	case '/':
		if(n2 != 0)
		   cout<< n1<<op<<n2<<" = "<<n1/n2;
		else cout<<"Infinite";
		break;
	 }
	return 0;
 }
```

2. اكتب برنامجاً يحسب أصغر الأعداد التي يدخلها المستخدم وعددها n يحدّده المستخدم.

الحل:

1. **الخوارزمية**
   a) اقرأ n.
   b) اقرأ الرقم الأول a_1.
   c) اعتبر أنّ أصغر عدد هو الرقم الأول $\min = a_1$
   (وذلك لكي نقارن كل الأعداد التي سنقرأها به (min) فإن وجدنا (a_i) أصغر منه اعتبرناه هو الأصغر).
   d) من أجل i = 2 حتى i = n (قرأنا الرقم الأول سابقاً لذلك بدأنا من 2)
      - اقرأ a_i
      - إذا كان $a_i < min$ فاعتبر أن $\min = a_i$
   e) اطبع أصغر عدد min
2. **الشيفرة**

```
#include<iostream>
using namespace std;
int main()
{
	int n, a, min;
	cout<<"Enter n: ";
	cin>>n;
	cout<<"\nEnter number 1: ";
	cin>>a;
	min = a;
	for(int i =2 ; i <=n;i++)
	{
		cout<<"\nEnter number "<<i<<": ";
		cin>>a;
		if(a<min)
		  min = a;
	}
	cout<<"\nmin = "<< min << endl;
	return 0;
 }
```

3. اكتب برنامجاً يقرأ اسم طالب وعدد العلامات التي يرغب بحساب المتوسط الحسابي لها ثمّ يقرأ كلّ علامة منها وفي النهاية يطبع اسم الطالب ومتوسطه الحسابي في كلّ المواد.

الحل:

1. **الخوارزمية**
   a) اقرأ اسم الطالب ثم عدد العلامات.
   b) من أجل i = 1 حتى i = n بزيادة قدرها 1
      - اِقرأ العلامة رقم i
      - أضف العلامة رقم i إلى المجموع
   c) اطبع اسم الطالب ثم المتوسط الحسابي (المجموع ÷ n).

2. **الشيفرة**

```
#include<iostream>
#include<string>
using namespace std;
void main()
{ string name;
int n,mark,i = 0,sum = 0;
//important to initial sum and i to 0 here (it's local var).
cout<< "Enter student name: ";     cin>>name;
cout<< "Enter number of marks: ";  cin>>n;
while(i++ < n)
{      cout<< "Enter mark ("<<i<<") [between 0 and 100]: ";
       cin>> mark;
       sum += mark;
}; // you can do that
cout<< name << " Average = " << (float)sum/n <<'\n';
}
```

4. اكتب برنامجاً لحساب السلسلة التالية حيث تدخل n من لوحة المفاتيح

$$a = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

الحل:

1. **الخوارزمية**
   a) اقرأ n.
   b) من أجل القيم بين الـ i=1 حتى i=n فإن
      المجموع a = المجموع a + $\frac{1}{i}$
   c) اطبع المجموع.
2. **الشيفرة**

```
#include<iostream>
using namespace std;

int main()
{
       int  n ;
       float a = 0.0f ;//Initial a to 0 where f means float value
       cout<<"Enter n: ";
       cin>>n;
       for(int i=1;i<=n;i++)
              a+=1.0f/i;//like a = a + (float)1/i;
        cout<< a << endl;
       return 0;
}
```

ملاحظة: التعليمة float a = 0.0f ; تعني أنّ القيمة المخزّنة في المتغيّر a هي من النوع العشري و ليس الصحيح. وكذلك الأمر في التعليمة a+=1.0f/i; وكان بإمكاننا أن نكتبها كالتالي:

```
a += (float)1 / i ;
```

لأنّ ناتج عمليّة القسمة الموجودة على يمين المساواة السابقة ليس بالضرورة عدداً صحيحاً فمثلاً العدد 1/2 = 0.5 وهو عشريّ لذلك قمنا بتعريف المجموع على أنّه عشريّ ثمّ قمنا بتحويل الطرف الأيمن من المساواة إلى النوع العشريّ لنضمن عدم ضياع البيانات لأنّ قسمة عددين صحيحين هي عدد صحيح في c++. ولو لم نفعل ذلك فإن ناتج التعليمة سيكون كالآتي:

```
a += 1 / i ;
```

عندها i = 1 `a = 0.0 + 1 / 1 = 0.0 + 1 = 1.0`

عندها i = 2 `a = 1.0 + 1 / 2 = 1.0 + 0 = 1.0`

...

عندها i = n `a = 1.0 + 1 / n = 1.0 + 0 = 1.0`

$$int\ r = \frac{int\ a}{int\ b} = \frac{1}{2} = 0.5$$

$$float\ r = \frac{float\ a}{float\ b} = \frac{float\ a}{int\ b} = \frac{int\ a}{float\ b} \ : \ \frac{1.0}{2.0} = \frac{1}{2.0} = \frac{1.0}{2} = 0.5$$

5. اكتب برنامجاً يحسب السلسلة التالية من أجل أي عدد يدخله المستخدم

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

الحل:

1. الخوارزمية

a) اقرأ n.

b) من أجل القيم بين الـ `i=0` حتى `i=n` فإنّ

- $fact_i = 1$
- احسب fct_i=`i!`:
  - من أجل `j=1` حتى `j=i` فإن $fact_i$ = $fact_i$ * `j`
- المجموع `e` = المجموع `e` + $\frac{1}{\text{fact}_i}$

c) اطبع المجموع.

2. الشيفرة

```
#include<iostream>
using namespace std;
int main()
{
 int n;
 double e = 0.0f; //Initial a to 0 where f means float value
 double fact = 1;
 cout<<"Enter n: ";
 cin>>n;
 for(int i=0;i<n;i++)
 {
  fact = 1; //initial fact to 1 for each calculation of i!
  for(int j = 1; j<=i;j++)
      fact *= j;
  e += 1.0f/fact; //like e = e + (float)1/i;
 }
 cout<< e << endl;
 return 0;
}
```

ملاحظة: يجب عليك أن تقوم بتبدئة متغيّرات المجاميع والجداءات إلى القيمة الحياديّة في العمليّة فمثلاً المتغيّر e يمثّل مجموع السلسلة ونعلم أنّ الجمع يأخذ 0 كقيمة حياديّة أي أنّ إضافة 0 إلى المجموع لن يغيّر من النتيجة شيئاً.
أمّا المتغيّر fact فهو يمثّل جداءات أي أنّ القيمة الحياديّة هي 1 لأنّ ضربها بأيّ عدد لن يغيّر من النتيجة شيئاً. و لو جعلناها 0 فإنّ القيمة التي سنحصل عليها من أجل المتغيّر fact هي 0 لأنّ:
fact = 0 وهذا يعني أنّ ناتج التعليمة التالية هو كالآتي:

```
fact = fact * j;  →  fact = 0 * j;  →  fact = 0 ;
```

6. اكتب برنامجاً يحسب السلسلة التالية من أجل أي عددين x,n يدخلهما المستخدم.

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

الحل:

1. **الخوارزمية**
   a) اقرأ x,n.
   b) من أجل القيم بين الـ i=0 حتى i=n فإن
   - $fact_i = 1$ (كل مرة نعيد قيمة fac_i إلى 1)
   - احسب act_i=i!:
     من أجل j=1 حتى j=i فإن
     $fact_i$ = $fact_i$ * j
   - المجموع e^x = المجموع e^x + $\frac{x^i}{\text{fact}_i}$

   c) اطبع المجموع.
2. **الشيفرة**

```
#include<iostream>
#include<cmath> // pow عند استخدام التوابع الرياضية مثل
using namespace std;
int main()
{
int n,x ;
double e = 0.0f ;//Initial a to 0 where f means float value
double fact = 1;
cout<<"Enter n,x: ";
cin>>n>>x;
for(int i=0;i<n;i++)
{
fact = 1;//initial fact to 1 for each calculation of i!
for(int j = 1; j<=i;j++)
    fact *= j;
e += pow(x,i)/fact;
}
cout<< e << endl;
return 0;
}
```

7. اكتب برنامجاً لحساب العلاقتين التاليتين:

$$1)\ \bar{X} = \sum_{i=1}^{n} \frac{a_i}{n} \qquad 2)\ sum = \sum_{i=1}^{n} i^3$$

الحل 1:

1. الخوارزمية

a) اقرأ n.

b) من أجل القيم بين الـ `i=1` حتى `i=n`

- اقرأ a_i
- المجموع sum = المجموع sum + a_i
- المتوسّط الحسابي للأعداد = المجموع ÷ n. أي $average = \frac{sum}{n}$

c) اطبع المتوسّط الحسابي.

2. الشيفرة

```
#include<iostream>
using namespace std;
int main()
{
	float sum =0 , average=0 , a=0;
	int n;
	cout<<"Enter n: ";
	cin>>n;
	for(int i =1 ; i<=n;i++)
	{
		cout<<"Enter a"<<i<<" : ";
		cin>>a;
		sum += a;
	}
	average = sum / n;
	cout<<"\nAverage = "<<average;
	return 0;
}
```

الحل 2:

1. الخوارزمية

a) اقرأ n.

b) من أجل القيم بين `i = 1` حتى `i = n`

المجموع sum = المجموع sum + $i * i * i$

c) اطبع المجموع.

2. الشيفرة

```
#include<iostream>
using namespace std;
int main()
{
	float sum =0;
	int n;
	cout<<"Enter n: ";
	cin>>n;
	for(int i =1 ; i<=n;i++)
		sum += i*i*i;
	cout<<"\nSum = "<<sum;
	return 0;
}
```

8. اكتب برنامجاً لإيجاد جذور معادلة من الدرجة الثانية معتمداً على المعلومات التالية:

$$\boldsymbol{ax^2 + bx + c = 0} \qquad ; a, b, c \in R$$

$$\Delta = \sqrt{b^2 - 4.a.c}$$

$$if \begin{cases} \Delta > 0 & x1 = \frac{-b+\sqrt{\Delta}}{2.a}, x2 = \frac{-b-\sqrt{\Delta}}{2.a} \\ \Delta = 0 & x1 = x2 = \frac{-b}{2.a}\ doublicated\ root \\ \Delta < 0 & Impossible\ in\ R \end{cases}$$

الحل:

1. الخوارزمية

1) اقرأ a,b,c

2) أ) إذا كان $a \neq 0$

- إذا كان $b \neq 0$
  - إذا كان $c \neq 0$
    - ✓ $\Delta = \sqrt{b^2 - 4.a.c}$
    - ✓ إذا كان $\Delta > 0$ فإن:

$$x1 = \frac{-b + \sqrt{\Delta}}{2.a}, x2 = \frac{-b - \sqrt{\Delta}}{2.a}$$

- ✓ و إلّا إذا كان $\Delta = 0$ فإن:

$$x1 = x2 = \frac{-b}{2.a}\ doublicated\ root$$

- ✓ و إلّا اطبع المعادلة مستحيلة الحل.
- و إلّا(c=0) فإنّ : إما $x = 0$ أو $x = -\frac{b}{a}$
- و إلّا(b=0) فإنّ $x^2 = -\frac{c}{a} = t$
  - فإذا كان $t \geq 0$ فإنّ $x = \sqrt{t}$
  - و إلا $t < 0$ فاطبع المعادلة مستحيلة الحل.

ب) و إلا(a=0)

إذا كان $b \neq 0$ فإنّ $x = -\frac{c}{b}$.

وإلا اطبع it's not equation.

2. الشيفرة

```
#include<iostream>
#include<cmath> // or <math.h>
using namespace std;
int main()
{
  float a , b , c;
  float delta=0.0 , x1=0.0 , x2=.0;
  float dsqrt=.0;
  cout<<"Enter a , b , c:";
  cin>>a>>b>>c;
  cout<<"\n";
  if(a!=0) //1
  {
    if(b!=0) //2
    {
      if(c!=0) //3
      {
       delta =((b*b) - (4*a*c));
       cout<<"delta = "<<delta<<"\n";
       if(delta > 0) //4
       {
         x1 = (float)(-1*b + sqrt(delta)) / 2*a;
         x2 = (float)(-1*b - sqrt(delta)) / 2*a;
         cout<<"x1 = "<< x1<<"\nx2 = "<< x2 << endl;
       }
       else if(delta == 0)
       {
         x1 = x2 = (float)-1*b / 2*a ;
         cout<<"x1 = x2 = "<<x1<<" Doublicated root!\n";
       }
       else
         cout<<"Impossible in R\n";
    }
    else
    {
      x1 = 0;
      x2 = (float)-1*b/a;
      cout<<"Either x = "<<x1<<"\nOr x = "<<x2<<endl;
    }
 }
 else
 {
   float t = (float)-1*c/a;
   if(t>=0)
     cout<<"x = "<<sqrt(t)<<endl;
   else
     cout<<" Impossible in R\n";
 }
}
else{
    if(b!= 0) cout<<"x = "<< (float) -c/b<<endl;
    else cout<<"it\'s not equation.\n";
    }
return 0;
}
```

تمارين

1. اكتب الشيفرة الموافقة للخوارزميّة التالية: (باستخدام التعليمة if)
   i. أدخل المعدّل
   ii. إذا كان المعدّل أكبر من 89 و أصغر أو يساوي 100 اطبع Excellent
   iii. إذا كان المعدّل أكبر من 79 و أصغرأو يساوي 89 اطبع Very Good
   iv. إذا كان المعدّل أكبر من 69 و أصغرأو يساوي 79 اطبع Good
   v. إذا كان المعدّل أكبر من 59 و أصغرأو يساوي 69 اطبع Passed
   vi. وإلا اطبع failed
2. صحّح البرامج التالية:

a)
```
while ( c <= 5 )
{
 product *= c;
 ++c;
```

b)
```
if ( gender == 1 )
    cout<< "Woman" ;
else;
    cout<<"man";
```

c)
```
while ( z >= 0 )
        sum += z;
```

d)

```
// nested loop
#include <iostream>
using namespace std;
int main ()
{
  int j = 1;
  for( int i = 1 ; i<=3 ; i++)
  {
      while( j<i )
      {
          cout <<"* " ;
          j++;
      }
  }
  return 0;
}
```

```
*
* *
* * *
* * * *
* * * * *
```

3. اكتب برنامجاً يقرأ ثلاثة أعداد ثمّ يرتّبها تصاعديّاً.
4. اكتب برنامجاً لكلّ شكل من الأشكال التالية:

1
```
3 4 5 6
4 5 6 7
5 6 7 8
6 7 8 9
```

2
```
3
4 5
5 6 7
6 7 8 9
```

3
```
3 4 5 6
  5 6 7
    7 8
      9
```

4
```
      6
    6 7
  6 7 8
6 7 8 9
```

5
```
3 4 5 6
4 5 6
5 6
6
```

6
```
3 4 5 6
4     7
5     8
6 7 8 9
```

5. اكتب برنامجاً يقرأ عدداً ثمّ يحدد فيما إذا كان يقبل القسمة على 10 أو لا.
6. اكتب برنامجاً لحساب العلاقة التالية:

$$f(x) = \frac{x^3-7}{x^2-4x+3} : x \in R\backslash\{1,3\}$$

7. اكتب برنامجاً لحساب وطباعة مجموع الأعداد الزوجية القابلة للقسمة على 3 ما بين 1 و 99.
8. اكتب برنامجاً لحساب المضاعف المشترك الأصغر LCM لعددين مدخلين من لوحة المفاتيح.
9. اكتب برنامجاً لحساب العلاقة التالية $x^y : y \geq 0$ بطريقتين مختلفتين.
10. اكتب برنامجاً يطلب من المستخدم أن يدخل حرفاً ثمّ عدداً، ثمّ يقوم البرنامج بطباعة مثلث قائم ومتساوي الساقين من هذا الحرف وطول ساقه هو الرقم المدخل سابقاً.
11. اكتب برنامجاً يطلب من المستخدم إدخال عددين ثمّ يقوم بتقسيم الأوّل على الثاني (إذا كان العدد الثاني صفراً فإن البرنامج يطلب من المستخدم أن يعيد إدخال الرقم الثاني من جديد)، ثمّ يخير المستخدم بين إجراء عملية جديدة (وذلك بإدخال الحرف n) والخروجِ من البرنامج (و ذلك بإدخال الحرف e).
12. اكتب برنامجاً يطبع جميع قواسم عدد مدخل من لوحة المفاتيح.

الفصل السابع :

التوابع (functions)

لعلّك تذكر برنامجك الأوّل في هذا الكتاب Hello World قلنا حينها: "إنّ النقطة الأولى لبداية البرنامج هي التابع main()". لكن **ما هو التابع ولماذا نستخدمه؟**

التابع: هو بنية برمجية يتمّ تنفيذها عند استدعائه من أي نقطة في البرنامج.

ونستخدم التوابع لأنّها توفّر الكثير من المزايا ومنها:

1- تختصر من طول البرنامج و توفّر الوقت، فبدلاً من إعادة كتابة ماسيقوم به تابع ما عدّة مرّات فإنّنا نكتبه مرّة واحدة في تابع ثمّ نستدعي هذا التابع كلّما دعت إليه الحاجة.

2- تسهل قراءة البرنامج وتتبّع الأخطاء، لأنّه من غير الجيّد أن تكتب برنامجك كلّه داخل التابع main() فيصبح متشابكاً وتصعب قراءتُه.

3- تصوّر أنّك كتبت برنامجاً استخدمت فيه مجموعة من التعليمات عدّة مرّات ثمّ قرّرت أن تحسِّن من أداء هذه التعليمات أو أن تصحِّح خطأ ما فيها، فهل ستقوم بهذا التصحيح من أجل كل مرّة كتبت فيها تلك التعليمات. ماذا لو استخدمت تابعاً يقوم بهذه العمليات ثمّ استدعيته تلك المرّات العديدة؟ عندها سيكون كافياً أن تعدّل التابع مرّة واحدة فقط ليسري هذا التعديل في برنامجك كلّه دون عناء يذكر.

وتشبه التوابع البرمجية الدوال الرياضيّة في عملها إلى حدٍّ ما، فهي تقوم بمجموعة من العمليات ثمّ تعيد قيمة **وحيدة** تدعى قيمة التابع.

كيف نصرح عن التوابع؟ يتم التصريح عن التوابع وفقاً للقاعدة التالية:

```
return_value_type  function_name( argument1, argument2,.. .)
{
   Statements;
 . . .
  return function_value;
}
```

حيث أن:

- **return_value_type** نوع القيمة التي سيعيدها التابع.
- **function_name** اسم التابع، وبه يمكننا استدعاء التابع.
- **arguments** الوسطاء التي يتمّ تمريرها إلى التابع ليُعالجَها ويستخرجَ القيمةَ المطلوبة، وهذه الوسطاء اختياريّة أي أنّه يمكن الاستغناء عنها كلِّها، ولكن يجب الإبقاء على الأقواس () بعد اسم التابع، وكل وسيط يجب أن يسبِق اسمَه نوعُ البيانات التي يحملها كما لو كان متغيّراً. إلّا أنّه متغيّر شكليّ فقط، وعند استدعاء التابع سيتمّ استبدال الوسطاء بمتغيّرات أو ثوابت من نفس النوع. يفصل بين كل وسيطين متتابعين فاصلة من الشكل (,).
- **statements** هي التعليمات التي تشكل جسم التابع. قد تكون مؤلّفة من تعليمة واحدة أو أكثر وفي كلتي الحالتين **يجب** أن توضع بين القوسين { }. تقوم هذه التعليمات بإيجاد قيمة التابع المطلوبة.
- **return** function_value هذه التعليمة تعيد القيمة التي حُسِبَتْ في جسم التابع، وهذه القيمة **وحيدة**.

و الآن لنكتب أوّلَ تابعٍ لنا في هذا الكتاب:

```
// function example
#include <iostream>
using namespace std;
int addition (int a, int b)
{
  int r = a + b;
  return (r);
}
int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z;
  return 0;
}
```

```
The result is 8
```

أذكُرُ أنّنا قلنا في درسنا الأول: "إنّ البرنامج يبدأ من التابع main() دائماً". لذلك سنبدأ من هناك.
تستطيع أن ترى كيف أن التابع main() بدأ بالتصريح عن المتغيّر z من النوع الصحيح، وبعد ذلك نجد التعليمة
z = addition(5,3) ;

إنّ التعليمة السابقة تمثّل استدعاءً للتابع addition ولو أنّنا انتبهنا لوجدنا تشابهاً كبيراً بين استدعاء التابع و التصريح عنه كما في الشكل:

```
int addition(int a,int b)
               ↑    ↑
z = adition(    5 ,   3   );
```

حيث يتمّ استبدال الوسيطين الشكليين a, b بالقيمتين 5, 3 على الترتيب ثمّ يتوجّه المترجم إلى التابع addition تاركاً التعليمات التي لم ينفذّها بعدُ من التابع main(). يعرِّف التابع addition متغيّراً موضعيّاً(انظر فقرة المدى في نهاية الدرس) جديداً r ثمّ يسند إليه مجموع الوسيطين a, b.

```
r =  a + b ;
```

لكنّ a = 5, b = 3 وبالنتيجة فإنّ r = 8 ثمّ يعيد التابع القيمة r باستخدام التعليمة **return r ;** أي أنّ قيمة التابع هي العدد 8 بعدها يخرج المترجم من التابع addition ثمّ يعود إلى التعليمة

```
z = addition(5,3);
```

وبما أنّ قيمة التابع addition أصبحت 8 فهذا يعني أنّ z = 8 أيضاً. كما في الشكل:

```
int addition(int a,int b)
↓
z = adition(   5 ,  3   );
```

ثمّ يكمل المترجم ما تبقى له من تعليمات التابع main().

مثال:

```
// function example
#include <iostream>
using namespace std;
int sub(int a, int b)
{
  return (a-b);
}
int main ()
{
  int x=5, y=3, z;
  z = sub(7,2);
  cout<<"The 1st result is "<<z<<'\n';
  cout<<"The 2nd result is "<<sub(7,2)<<'\n';
  cout<<"The 3rd result is "<<sub(x,y)<<'\n';
  z = 4 + sub(x,y);
  cout << "The 4th result is " << z << '\n';
  return 0;
}
```

```
The 1st result is 5
The 2nd result is 5
The 3rd result is 2
The 4th result is 6
```

التوابع التي لا تعيد قيمة (استخدام الكلمة void)

في بعض الأحيان نحتاج إلى تابع يقوم بمجموعة من الأشياء دون أن يعيد قيمة أو قد نريد منه أن يعيد أكثر من قيمة لكن وكما تعلّمنا **فالتابع يجب أن يعيد قيمة وهذه القيمة وحيدة** لذلك كان الحل باستخدام الطرق (الطرق methods هو الاسم الذي تستخدمه لغة java لتعبّر به عن التوابع التي لا تعيد قيماً). **لكن ماهو الفرق؟** في واقع الأمر تشبه الطريقة التابع في كل شيء إلّا أنّها لا تملك نوعاً من البيانات لأنّها بالأصل لا تعيد قيمةً.

يتمّ التصريح عن الطريقة بنفس الأسلوب الذي صرّحنا به عن التابع مع اختلافين بسيطين هما:

1- لن نضع نوعاً للبيانات وبدلاً من ذلك سنستخدم الكلمة void دلالة على أنّ الطريقة التالية لن تعيد شيئاً.

2- بما أنّ الطريقة لا تعيد أي قيمة فلا داعي لأن نضع التعليمة return في نهايتها.

وبذلك يكون شكل الطريقة كالآتي:

```
void  method_name( void )
{
  statements ;
  . . ..
}
```

مثال:

```
#include <iostream>
using namespace std;
void dummyfunction (void)
{
cout << "I'm a function!\n";
}
int main()
{
  dummyfunction() ;
  return 0;
}
```

```
I'm a function!
```

ملاحظة: في C++ عندما نصرّح عن تابع أو طريقة دون وسطاء فليس من الضروري استخدام الكلمة void بين القوسين () لكنّنا نضعها لنشير إلى أنّ التابع لا يمتلك وسطاء.
وممّا **يجب** عليك أن تنتبه إليه عندما تستدعي أيّ تابع هو استخدام القوسين () بعد اسم التابع حتّى ولو لم يمتلك وسطاءً.

تمرير الوسطاء بالقيمة و بالمرجع(العنوان)

في الأمثلة السابقة قمنا بتمرير الوسطاء بالقيمة وهذا يعني أنّنا عندما نستدعي تابعاً ذا وسطاء بالقيمة فإنّنا نمرّر **قيم المتغيّرات** إلى التوابع ولا نمرّر المتغيّرات نفسها. كمثال افترض أنّنا استدعينا التابع addition في المثال الأول كالآتي:

```
int x = 5 , y = 3 , z ;
z = addition(x,y) ;
```

ما فعلناه هنا هو أنّنا استدعينا التابع addition ومرّرنا إليه قيمتي المتغيّرين x و y أي أنّنا مرّرنا إليه الرقمين 5 و 3 ولم نمرّر إليه المتغيّرين x و y. كما في الشكل.

```
int addition(int a,int b)
                 ↑5   ↑3
z = adition(    x ,   y   );
```

إنّ استدعاء الوسطاء بالقيمة يمنع حدوث أي تغيير على المتغيّر الذي تمّ تمريره إلى التابع خارج هذا التابع ولتوضيح هذه المسألة نقدّم هذا المثال:

```
// passing by values
#include<iostream>
using namespace std;
int add(int a,int b)
{
 int result = a + b;
 a--; b++; //changeing values of a and b
 return result;
}
int main()
{
 int a=5,b=3,z;
 cout<<"a and b before calling add function is: "<<a<<", "<<b<<endl;
 z = add(a,b);
 cout<<"value of add function is: "<<z<<endl;
 cout<<"a and b after calling add function is: "<<a<<", "<<b<<endl;
 return 0;
}
```

```
a and b before calling add function is: 5, 3
value of add function is: 8
a and b after calling add function is: 5, 3
```

لعلّك لاحظت أنّ قيمتي المتغيّرين a و b لم تتغيّرا في التابع main() حتّى بعد استدعاء التابع addition الذي قام بالتعديل على الوسيطين a و b المعرّفين فيه والذين تغيّرت قيمتاهما كما هو واضح في نتيجة البرنامج السابق. **لكن ماذا لو أردنا أن تتغير قيم المتغيّرات عند تمريرها إلى تابع ما؟** عندها لن يكون تمريرها بالقيمة مجدياً ويتحتّم علينا عندئذٍ أن نمرّرها بالمرجع (العنوان).

عندما نمرّر متغيّراً ما بالمرجع إلى أحد التوابع فإنّنا لا نمرّر قيمته وحسب بل نمرّر المتغيّر نفسه أي أنّ أيّ تغيير على المتغيّر الوسيط في التابع سيَحْدُثُ على المتغيّر حتّى خارج هذا التابع.
ويتمّ الاستدعاء بالمرجع من خلال استخدام المعامل **&** بعد نوع الوسيط الذي نرغب بتمريره بالمرجع في جسم التابع كما في الشكل:

```
void duplicate (int& a, int& b, int& c)
                    ↕x   ↕y   ↕z
     duplicate (   x   ,   y   ,   z )
```

المثال التالي يوضح ذلك:

```
// passing parameters by reference
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
  a*=2;
  b*=2;
  c*=2;
}
int main ()
{
  int x=1, y=3, z=7;
  cout<<"x,y,z before calling duplicate";
  cout <<"\nx="<<x<<", y=" << y
       <<", z=" <<z<<endl;
  duplicate (x, y, z);
  cout<<"x,y,z after calling duplicate\n";
  cout <<"x="<<x<<", y=" << y
       <<", z=" <<z<<endl;
  return 0;
}
```

```
x,y,z before calling duplicate
x=1, y=3, z=7
x,y,z after calling duplicate
x=2, y=6, z=14
```

يجب أن تنتبه إلى أنّ كلّ وسيط في التابع duplicate أُتْبِعَ نوعُه بالرمز **&** الذي يعني أنّ هذا الوسيط سيمرّر بالمرجع و ليس بالقيمة كما في الحالة الأولى.

كما قلنا في بداية هذا الدرس إنّ التوابع تعيد قيمة **وحيدة بينما لا تعيد الطرق أيّة قيمة**، ولكنّ تمرير الوسطاء بالمرجع يعدّ أداةً قويةً تستطيع من خلالها أن تجعل التابع (الطريقة) يعيد أكثر من قيمة.
والمثال التالي يوضّح هذا:
لدينا التابع prevnext الذي يقبل ثلاثة وسطاء حيث يأخذ الوسيط الأوّل ويعيد الرقم الذي يسبقه كقيمة له من خلال الوسيط الثاني كما يعيد الرقم الذي يلي الوسيط الأول كقيمة له من خلال الوسيط الثالث.

```
// more than one returning value
#include <iostream>
using namespace std;
void prevnext (int x,int& prev, int& next)
{
  prev = x-1;
  next = x+1;
}
int main ()
{
  int x=100, y, z;
  prevnext (x, y, z);
  cout<<"Previous=" << y << ", Next="<< z << endl;
  return 0;
}
```

```
Previous=99, Next=101
```

ملاحظة : يمكنك كتابة التابع prevnext بالشكل:

```
int prevnext (int x,int& prev, int& next)
{ prev = x-1; next = x+1; return 0;}
```

المدى Scope

كلّ المتغيّرات التي سنستخدمها في برامجنا **يجب** أن يتمّ التصريح عنها أولاً. يمكن التصريح عن المتغيّرات في أيّ مكان من صفحة الشيفرة. لكنْ هناك شيئ يجب أن تعلمه عزيزي القارئ، وهو ما يُعرَفُ بمجال الرؤيا للمتغيّر (المدى). تابع الشكل التالي أولاً:

```
#include<iostream>
using namespace std;
int Integer ;
char aCharacter ;
char string[20] ;

int main()
{
unsigned short Age ;
float ANumber , AnotherOne ;

cout << "Enter Your age: ";
cin >> Age;
…
 return 0 ;
}
```

Global Variables

Local Variables

Instructions

المتغيّرات الشاملة Global variables و المتغيّرات الموضعيّة Local variables:

المتغيّرت الشاملة هي المتغيّرات التي يُصرّح عنها أعلى البرنامج خارج كل التوابع كما هو واضح في الشكل السابق فالمتغيّرات `Integer , aCharacter , string [20]` كلّها متغيّرات شاملة.

يمكن الوصول إلى هذه المتغيّرات من أي مكان داخل الشيفرة، و من أي تابع أو طريقة(إجراء) معرّفة في البرنامج.

وهذا مالا تستطيع المتغيّرات الموضعيّة أن تفعله، و**المتغيّرات الموضعيّة** هي التي **يُصرّح عنها داخل القوسين {}** لتابع أو إجراء أو بنية برمجية ما في البرنامج.

في الشكل السابق المتغيّرات `Age , ANumber , AnotherOne` صُرِّح عنها في جسم التابع main() أي أنّنا قادرون على الوصول إليها من داخل التابع main() فقط. وهذا يعني أنّه لن يكون بمقدورنا أن نصل إلى أحد هذه المتغيّرات من أي تابع آخر معرّف في البرنامج.

في C++ المدى للمتغيّر الموضعيّ محدّد فقط في البنية البرمجية التي صُرِّحَ عنه فيها (البنية البرمجية المقصودة هي مجموعة التعليمات المحصورة بين القوسين {}) أي لو أنّه عُرِّف ضمن التابع فإنّ مداه هو التابع الذي عُرِّفَ فيه، ولو أنّه عُرِّفَ ضمن إحدى البنى التكرارية فإنّ مداه هو تلك البنية التكراريّة وحسب.

ملاحظة: المتغيرات الشاملة تتم تبدأتها مع بداية البرنامج **قبل** التابع main بشكل تلقائي إلى القيم الصفرية لكل نوع فمثلاً المتغير Integer يأخذ القيمة 0 تلقائياً (جرّب أن تطبع قيم المتغيرات الشاملة من المثال السابق ستجد أن المتغيرات الحرفية تطبع فراغات (أو لا تطبع شيئاً)) أمّا المتغيرات الموضعية فيتمّ تبدأتها عندما يصل المترجم إلى مكان التصريح عنها **وهي تأخذ قيماً عشوائية بشكل تلقائي ما لم تقم بتبدئتها بنفسك**.

المتغيرات الثابتة static

هذا النوع من المتغيرات تتمّ تبدأته مرة واحدة فقط ويمكن أن تتغير قيمته (يوجد منه نسخة واحدة فقط)

مثال:

```
// Example static variables
#include <iostream.h>
void f(int a)
{
  while(a-- != 0)
  {
   static int n = 0 ;// initialized once
   int x = 0 ;       // initialized n times
   cout<<"n == "<<n++<<", x == "<<x++<<"\n";
  }
}
int main ()
{
  f(4);
  return 0;
}
```

```
n == 0 , x == 0
n == 1 , x == 0
n == 2 , x == 0
n == 3 , x == 0
```

تعيين القيمة الافتراضية للوسطاء

عندما نصرّح عن متغيّر فإنّنا نستطيع أن نعطيه قيمة ابتدائية ثمّ قد نرغب بتغيير قيمته فيما بعد أثناء البرنامج لكن **هل نستطيع أن نعطي قيمة ابتدائية لوسيط معرف داخل تابع ما؟**

لا تتعجّب إن قلتُ لك: "نعم". في الحقيقة يعدّ هذا ممكناً من أجل كلّ وسيط تريد أن تعطيه قيمة ابتدائية لتكون قيمتَه الافتراضية في حال لم تقم بوضع أيّ قيمة في مكان هذا الوسيط عند استدعاء التابع، ولكن لو قمت بوضع قيمة أخرى أثناء استدعاء التابع فسيعتبر المترجم أنّ قيمة الوسيط هي القيمة التي أدخلتها عند استدعاء التابع وسيتجاهل القيمة الافتراضية تلك.

مثال:

```
// default values in functions
#include <iostream>
using namespace std;
int divide (int a, int b=2)
{
  int r;
  r=a/b;
  return (r);
}
int main ()
{
  cout << divide (12);
  cout << endl;
  cout << divide (20,4)<<endl;
  return 0;
}
```

```
6
5
```

كما ترى فإنّ ناتج التعليمة `divide(12)` هو العدد 6 لأنّنا لم نضع قيمة للوسيط الثاني b الذي يأخذ القيمة 2 كقيمة افتراضية له لذلك فإنّ `divide(12) = 12 / 2 = 6`

أمّا في الحالة الثانية فقد استدعينا التابع بالشكل `divide(20,4)` أي أنّنا طلبنا منه أن يغيّر القيمة الافتراضية للوسيط b لتصبح العدد 4 وعندها يكون الناتج `divide(20,4) = 20 / 4 = 5`.

إعادة تحميل التوابع functions overloading

لو كان لدينا تابعان يقومان بنفس التعليمات تماماً ويمتلكان نفس الاسم أيضاً لكنّهما يختلفان عن بعضيهما بالوسطاء وهذا الاختلاف قد يكون في الأنواع أو في عدد الوسطاء المطلوبة، فهذا ما يُعرَفُ بإعادة التحميل التوابع.

وإليك المثال التالي:

```
//overloaded function
#include<iostream>
using namespace std;
int divide (int a, int b)
{  return (a/b);  }
float divide (float a, float b)
{  return (a/b);  }
int main ()
{
  int x=5,y=2;
  float n=5.0,m=2.0;
  cout << divide (x,y);
  cout << "\n";
  cout << divide (n,m);
  cout << "\n";
  return 0;
}
```

```
2
2.5
```

يستطيع المترجم أن يحدّد أيّ التابعين سيستخدم من خلال نوع االوسطاء التي ستُمرَّرُها فإذا كانت أعداداً صحيحة سيستدعي التابع المعرّف أوّلاً والذي يقبل وسطاء من النوع `int` وإذا كانت الوسطاء من النوع `float` عندها سيستخدم التابع الثاني.

يفيدنا هذا الأسلوب في التقليل من أسماء التوابع التي سنستخدمها في برنامجنا، فطالما أنّ هذه التوابع تقوم بنفس العمل، فلِم لا تأخذ نفس الاسم.

التعريف المبدئي للتوابع functions prototype

حتّى الآن كلّ التوابع التي قمنا بكتابها صرّحنا عنها فوق التابع `main` لكي نستطيع استدعاءها في التابع `main()` ، **لكن ماذا لو أردنا أن نكتب تابعاً ما تحت التابع `main()`؟**

يمكننا فعل ذلك ولكن باستخدام الطريقة التالية:

نصرّح عن التابع في أعلى البرنامج قبل التابع `main()` بالشكل التالي:

data_type function_name (*argument_type1*, *argument_type2*...);

مع مراعاة الأمور التالية:

- **يجب** ألّا تكتب أيّة تعليمة من جسم التابع في هذا التصريح.
- **يجب** أن تضع فاصلة منقوطة في نهاية هذا التصريح.
- يُكتفى بوضع أنواع البيانات للوسطاء دون أسمائها، ويعدّ كتابة الأسماء أمراً اختيارياً وهو الأفضل.

ثمّ تحت التابع `main()` نقوم بكتابة جسم التابع.

مثال:

```
// prototyping
#include <iostream>
using namespace std;
void odd (int a);
void even (int a);

int main ()
{
  int i;
  do {
    cout << "Type a number(0 to exit): ";
    cin >> i;
    odd (i);
  } while (i!=0);
  return 0;
}
void odd (int a)
{
  if ((a%2)!=0)
     cout<<"Number is odd.\n";
  else even (a);
}
void even (int a)
{
  if((a%2)==0)
    cout <<"Number is even.\n";
  else odd (a);
}
```

```
Type a number (0 to exit): 9
Number is odd.
Type a number (0 to exit): 6
Number is even.
Type a number (0 to exit): 1030
Number is even.
Type a number (0 to exit): 0
Number is even.
```

الفصل الثامن:

التعاوديّة (Recursion)

التعاوديّة: هي خاصيّة يستطيع التابع من خلالها أن يستدعي نفسه.

وتستخدم هذه الطريقة لحل المشاكل التي يمكن ردّها إلى مشكلة واحدة مكرّرة عدّة مرّات، و يعتبر هذا الأسلوب مفيداً من أجل بعض طرق الترتيب أو لحساب عاملي عدد ما n! والذي يعطى بالعلاقة التالية:

$$n! = n.(n-1).(n-2).\ldots.1 \; if \; n > 0$$

ويُصطلح على أنّ:

$$0! = 1! = 1$$

لو تأملنا العلاقة جيّداً سنجد أنّها تكتب أيضاً بالشكل التالي:

$$n! = n.(n-1)!$$

وكمثال لنأخذ الرقم 4 ونحسب العاملي له:

$$4! = 4.(3.\ 2.\ 1) = 24$$

أو

$$4! = 4.\ 3!\ = 4.\ 3.\ 2!\ = 4.\ 3.\ 2.\ 1! = 24$$

والآن سنكتب برنامجاً يستخدم تابعاً تعاوديّاً لحلّ هذه المشكلة:

```
// factorial calculator
#include <iostream>
using namespace std;
long fact(long n)
{
  if (n > 1)
   return (n * fact(n-1));  // n.(n-1)!
  else
   return (1);
}
int main ()
{
  long m;
  cout << "Type a number: ";
  cin >> m;
  cout<<"\n"<<m<<"! = "<<fact(m)<<endl;
  return 0;
}
```

```
Type a number: 6
6! = 720
```

لعلّك لاحظت أنّ التابع fact قام باستدعاء نفسه من خلال التعليمة return (n * ***fact(n-1)***) وهذا ما يُعرف بالتعاوديّة. والمقصود بهذا أنّ التابع لن يكمل تنفيذ التعليمات الموجودة بعد هذه التعليمة وإنّما **سيعود** ليستدعيَ نفسه من جديد، وذلك لأنّ قيمة الاستدعاء الأوّل مرتبطة باستدعاء جديد للتابع نفسه وقيمة هذا الاستدعاء الجديد مرتبطة باستدعاء آخر سيتحقق بمجرّد وصول المترجم إلى نفس السطر من الاستدعاء الذي ينفذه حالياً وتستمر هذه الحلْقة من الاستدعاءات المترابطة مع بعضها ويستمر المترجم بحجز أماكن جديدة من الذاكرة لكل المتغيّرات الموجودة في التابع المُستَدْعى **ولكن إلى متى؟**

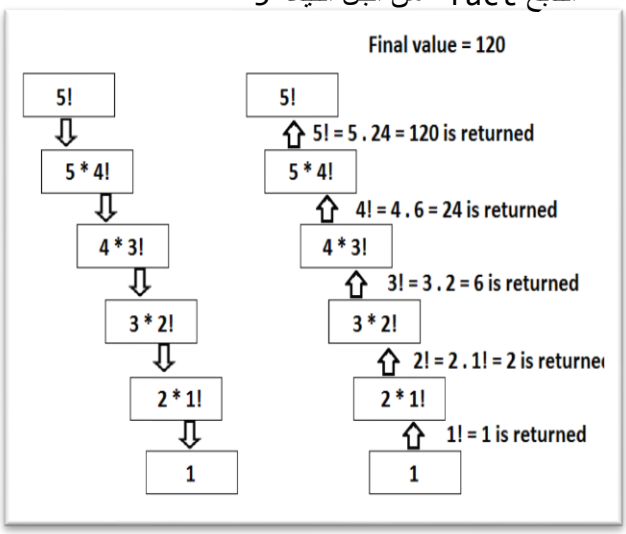
يعتبر *شرط التوقف* الذي **يجب** أن تضعه في التابع أهمّ ما **يجب** عليك فعله لأنّك لو لم تفعل فإنّ البرنامج سيدخل في حلقة تعاوديّة لانهائيّة نتيجتها توقّف البرنامج بخطأ من النوع `stack overflow`.

شرط التوقف هو شرط يؤدّي تحقّقه إلى تنفيذ تعليمة لا تحتوي على استدعاء جديد للتابع ومن المهمّ أن يكون هذا الشرط قابلاً للتحقّق لكي يتوقف التابع عن استدعاء نفسه وإلّا فليس لهذا الشرط أيّة قيمة. في مثالنا السابق الجزء الثاني من تعليمة الشرط هو شرط التوقف. الآن لنعد إلى برنامجنا السابق ولندخل عالم التعاوديّة المليء بالمغامرات.

التابع `fact` من أجل القيمة 3

```
fact(3) :
3 > 1 ?  yes
fact(3) = 3 * fact(2)
       fact(2) :
       2 > 1 ? yes
       fact(2) = 2 * fact(1)
              fact(1) :
                1 > 1 ? no
                return 1
         fact(2) = 2 * 1 = 2
         return 2
 fact(3) = 3 * 2 = 6
 return 6
```

التابع `fact` من أجل القيمة 5



كما تلاحظ تُحسب قيمةُ الاستدعاء الأخير أوّلاً ثمّ الذي قبله ثمّ الذي قبله حتّى يصل إلى الاستدعاء الأوّل مجسداً بذلك فكرة المكدّس `stack` وهي تقول: "آخرُ الداخلين هو أوّل الخارجين `(Last in first out)`".
هل يجب علينا أن نستخدم هذا الأسلوب دائماً؟ في الحقيقة إنّ استخدام هذا الأسلوب مربكٌ كثيراً ويكمن البديل في استخدام البنى التكراريّة، فالقاعدة تقول: "البرنامج الذي يمكن كتابته بشكل تعاوديّ يمكن كتابته بشكل تكراري وهذا أفضل وأسرع". لنكتب المثال السابق بشكل تكراري:

```
// factorial calculator
#include <iostream>
using namespace std;
long fact(int a)
{
  long f = 1;
  for(int i = a ; i>0 ; i--)
      f *= i ; //or  f = f * i;
  return f ;
}
int main ()
{
  long l;
  cout << "Type a number: ";
  cin >> l;
  cout << "\n" << l << "! = " << fact(l);
  cout<<endl;
  return 0;
}
```

```
Type a number: 9
9! = 362880
```

تمرين: اكتب التابع `fact` باستخدام الحلقتين التكراريتين الباقيتين.

مثال: البرنامج التالي يقوم بحساب مجموع السلسلة المكونة من الأعداد الطبيعية بالشكل:

$$1 + 2 + 3 + 4 + \cdots + n$$

```
#include<iostream>
using namespace std;
int sum(int n)
{
 if ( n > 1)
       return n+sum(n-1);
 else
       return 1;
}
int main()
{
 int n ;
 cout<< "Enter n: ";
 cin>> n;
 cout << "\nsum(" << n << ")= " << sum(n) << endl;
}
```

```
Enter n: 4
sum(4) = 10
```

```
sum(3) :
3 > 1 ?  yes
sum(3) = 3 + sum(2)
       sum(2) :
       2 > 1 ? yes
       sum(2) = 2 + sum(1)
              sum(1) :
                1 > 1 ? no
                return 1
         sum(2) = 2 + 1 = 3
        return 3
 sum(3) = 3 + 3 = 6
 return 6
```

سلسلة فيبوناكسي Fibonacci numbers:

سنكتب تابعاً تعاوديّاً لإيجاد الأرقام التالية: `0, 1, 1, 2, 3, 5, 8, 13, 21, 34,.. .`

حيث أنّ الرقم الأكبر من 1 ينتج من مجموع الرقمين الذين يسبقانه وفق العلاقات التالية:

`fib(0) = 0, fib(1) = 1`

`fib(n) = fib(n-1) + fib(n-2)` :n>1.

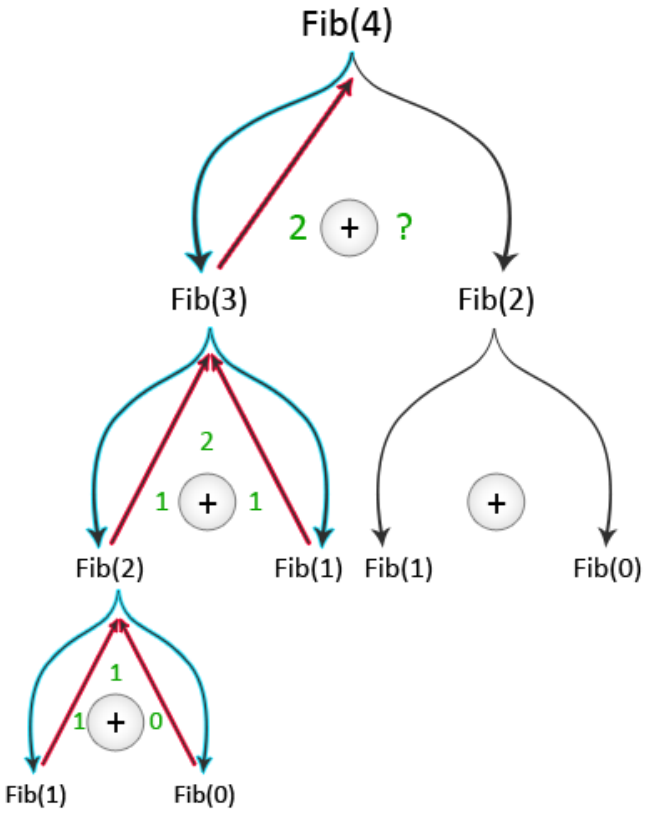
$\frac{fib(n)}{fib(n-1)} = 1.618 \ldots$ وهذه السلسلة تعطي النسبة الذهبية التي توجد بكثرة في الطبيعة وهي تنتج من القسمة

وهي المستخدمة في إنشاء النوافذ و الغرف و المباني حيث تمثل نسبة الطول إلى العرض.

```cpp
#include<iostream>
using namespace std;
int fib(int n)
{
 if ( n == 0 || n == 1 )
       return n;
 else
       return (fib(n-1)+fib(n-2));
}
int main()
{
 int n ;
 cout<< "Enter n: ";
 cin>> n;
 cout<< "\nfib(" << n << ") = " << fib(n) << endl;
}
```

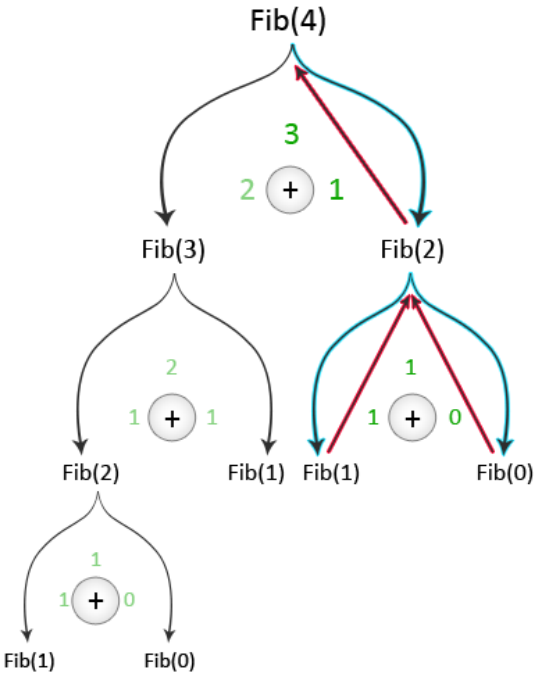
```
Enter n: 4
fib(4) = 3
```

1- fib(4):
   4 ?= 0 no, or 4 ?= 1 no
   fib(4) = fib(3) + fib(2)
2- fib(3):
   3 ?= 0 no, or 3 ?= 1 no
   fib(3) = fib(2) + fib(1)
3- fib(2):
   2 ?= 0 no, or 2 ?= 1 no
   fib(2) = fib(1) + fib(0)
4- fib(1):
   1 ?= 0 no, or 1 ?= 1 yes
   fib(1) = 1
   return fib(1)
5- fib(0):
   0 ?= 0 yes, or 0 ?= 1 no
   fib(0) = 0
   return fib(0)
   fib(2) = 1 + 0
   return fib(2)
   fib(3) = 1 + fib(1)
6- fib(1):
   1 ?= 0 no, or 1 ?= 1 yes
   fib(1) = 1
   return fib(1)
   fib(3) = 1 + 1 = 2
   return fib(3)
   fib(4) = 2 + fib(2)



```
7- fib(2):
   2 ?= 0 no, or 2 ?= 1 no
   fib(2) = fib(1) + fib(0)
8- fib(1):
   1 ?= 0 no, or 1 ?= 1 yes
   fib(1) = 1
   return fib(1)
9- fib(0):
   0 ?= 0 yes, or 0 ?= 1 no
   fib(0) = 0
   return fib(0)
   fib(2) = 1 + 0
   return fib(2)

fib(4) = 2 + 1 = 3
```



تمارين محلولة

1. ماهو ناتج تنفيذ التوابع التالية:

A	B	C

`#include<iostream>`
`using namespace std;`
`int sum(int a)`
`{`
`  if(a>1)`
`  return a+sum(--a);`
`  else if(a==1)`
`  return a;`
`  else return 0;`
`}`
`int main()`
`{`
`  cout<<sum(5)<<endl;`
`  return 0;`
`}`
	`#include<iostream>`
	`using namespace std;`
	`int sum(int a)`
	`{`
	`  if(a>1)`
	`  return a+sum(a--);`
	`  else if(a==1)`
	`  return a;`
	`  else return 0;`
	`}`
	`int main()`
	`{`
	`  cout<<sum(5)<<endl;`
	`  return 0;`
	`}`
		`#include<iostream>`
		`using namespace std;`
		`int sum(int a)`
		`{`
		`  if(a>1)`
		`  return a+sum(a-1);`
		`  else if(a==1)`
		`  return a;`
		`  else return 0;`
		`}`
		`int main()`
		`{`
		`  cout<<sum(5)<<endl;`
		`  return 0;`
		`}`

الحل

A	B	C
يطبع القيمة 11	يحدث خطأ من النوع stack overflow	يطبع القيمة 15

التعليل

A

الاستدعاء	a=	sum=
1	5	11
2	4	7
3	3	4
4	2	2
5	1	1

B

الاستدعاء	a=	sum=
1	5	not calculated
2	5	not calculated
3	5	not calculated
4	5	not calculated
...	...	not calculated

C

الاستدعاء	a=	sum=
1	5	15
2	4	10
3	3	6
4	2	3
5	1	1

الحالة A نجد التعليمة `return a+sum(--a);` في هذه التعليمة المعامل `--a` يأخذ أولوية التنفيذ حتّى ولو كان في يمين التعليمة لذلك فإنّ قيمة a ستتغيّر أوّلاً ثمّ سيتمّ الاستدعاء التالي، وسيكون ناتج التابع كما هو موضّح في الجدول A.

الحالة B نجد التعليمة `return a+sum(a--);` في هذه التعليمة المعامل `a--` يأخذ أولوية التنفيذ حتّى ولو كان في يمين التعليمة لكنّ قيمة a لن تتغيّر حتّى ينتقل المترجم إلى التعليمة التالية إلّا أنّ الاستدعاء التالي سيتمّ وستكون قيمة a عندها نفسها في الاستدعاء السابق أي أنّ شرط توقّف التعاوديّة لن يتحقّق أبداً ممّا يسبب الخطأ المذكور في الجدول B.

الحالة C نجد التعليمة `return a+sum(a-1);` في هذه التعليمة `a-1` تنقص قيمة a بمقدار 1 في الاستدعاء التالي، وسيكون ناتج التابع كما هو موضح في الجدول C.

(راجع فصل المعاملات. فقرة 4) معاملات الزيادة و النقصان)

2. يُقالُ عن عددٍ ما إنّه تامٌّ إذا كان يساوي مجموع قواسِمه كلِّها بالإضافة إلى العدد 1 ما عدا العدد نفسه.
مثال:

```
6 = 1 + 2 + 3
```

اكتب برنامجاً فيه إجراءٌ يطبع العبارة Is perfect إذا كان العدد المدخل تامّاً
ويطبع sn't perfect إذا لم يكن تامّاً.

الحل

1. **الخوارزمية (للتابع فقط)**
   1- من أجل i = 1 حتى i = a-1
   إذا كان i يقسم a أضفه إلى مجموع القواسم
   2- إذا كان المجموع يساوي a اطبع perfect
   و إلا فاطبع not perfect
2. **الشيفرة**

```
#include<iostream>
using namespace std;
void isperfect(int a);
int main()
{
       int n;
       cout<<"Enter number: ";
       cin>>n;
       isperfect(n);
        return 0;
}
void isperfect(int a)
{
      int sum = 0;
      for(int i = 1; i< a;i++)
        if(a%i == 0)
           sum+= i;
      (sum == a)? cout<<a <<" perfect\n":cout<< a<<" not perfect\n";
}
```

3. **أبراج هانوي (Towers of Hanoi)**

تعترض هذه المشكلة معظم متعلمي البرمجة وهي تعود إلى مناطق في شرق آسيا حيث كان في المعبد ثلاثة أعمدة الأول فيه 64 قرصاً مرتبة تصاعدياً من الأعلى إلى الأسفل بحيث يكون أكبرها في أسفل العمود وتقول الأسطورة: "إنّه عندما ينتهي الرهبان من نقل كل هذه الأقراص إلى العمود الثالث سينتهي العالم"، ولكي ينقل الرهبان هذه الأقراص يجب أن يراعوا القاعدة التالية:

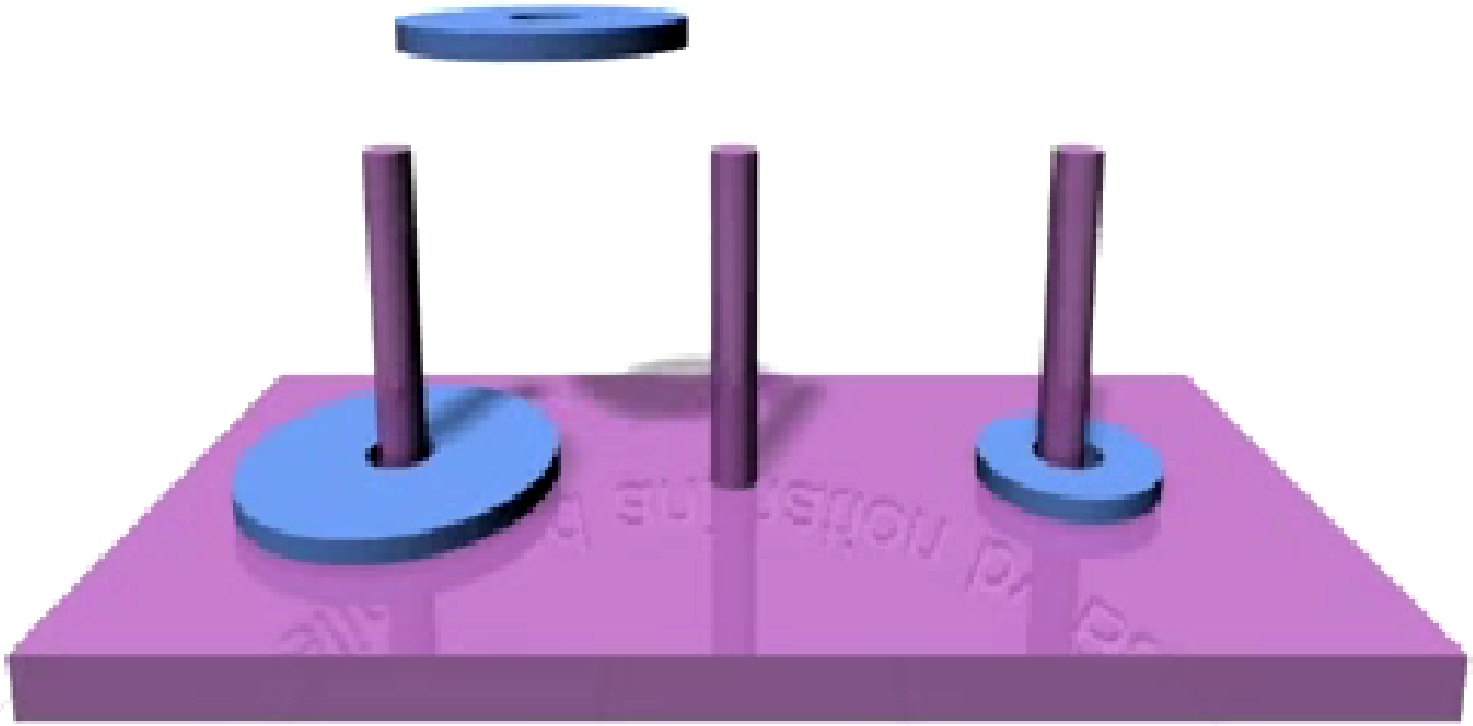
1- يجب نقل قرص واحد فقط في كلّ مرّة.
2- يجب استخدام أحد الأعمدة كوسيط مؤقت.
3- لا يجوز أن تضع قرصاً كبيراً فوق قرص أصغر منه أبداً.

مهمتّنا الآن أن نساعد هؤلاء الرهبان على إيجاد الطريقة الأفضل لحلّ هذه المشكلة.
حيث سنكتب تابعاً تعاوديّاً يقبل الوسطاء التالية:

- $n \geq 1$ عدد الأقراص
- from العمود الموجودة عليه الأقراص
- to العمود الذي سننقل الأقراص إليه
- temp العمود الوسيط المؤقت

ثمّ يطبع الخطوات اللازمة لنقل الأقراص اعتماداً على القاعدة السابقة.

modeling and animation by Bassel



مثال: لدينا ثلاثة أقراص نريد نقلها من العمود الأول إلى العمود الثالث بالاعتماد على العمود الثاني كوسيط مؤقت.

Move Disk (1) from 1 to 3
Move Disk (2) from 1 to 2
Move Disk (1) from 3 to 2
Move Disk (3) from 1 to 3
Move Disk (1) from 2 to 1
Move Disk (2) from 2 to 3
Move Disk (1) from 1 to 3

الحل

1. **الخوارزمية**

1- إذا كان n = 1
انقل القرص من العمود الأول إلى العمود الثالث من دون اللجوء إلى العمود الوسيط المؤقت.
2- و إلا افعل الخطوات التالية:
a- انقل القرص رقم n-1 من العمود الأول إلى العمود الثاني باستخدام العمود الثالث كوسيط مؤقت.
b- انقل القرص الأخير من العمود الأول إلى العمود الثالث.
c- انقل القرص رقم n-1 من العمود الثاني إلى العمود الثالث باستخدام العمود الأول كوسيط مؤقت.

2. **الشيفرة**

```
#include<iostream>
using namespace std;
int han(int n,int a,int b,int t);
int main()
{
   int n,from=1,to=3,temp=2;
   cout<<"Enter n, from, to, temp\ne.g: 3\n1\n3\n2\n\n\n";
   cin>>n>>from>>to>>temp;
   han(n,from,to,temp);
   return 0;
}

int han(int n,int from,int to,int temp)
{
    if(n ==1)
    cout<<"Move Disk ("<<n<<") from "<< from<<" to "<< to <<"\n";
    else
    {
        han(n-1,from,temp,to);
        cout<<"Move Disk ("<<n<<") from "<< from<<" to "<< to <<"\n";
        han(n-1,temp,to,from);
    }
  return 0;
}
```

تمارين

1. أوجد الأخطاء في التوابع التالية:

```
1. int Sum(int a , int b)
   {
      int sum = a + b;
   }
2. int divide(int a , int b)
   {
      return (float)a/b;
   }
3. void subtract(int a , int b)
   {
      return a-b;
   }
```

2. ما هو ناتج البرنامج التالي:

```
#include<iostream>
using namespace std;
int x = 0;
float b;
float d(float a , float& b)
{
	a--;
	b++;
	x = a/b;
	return (x+1)/2.0;
}
int main()
{
	float a;
	cout<<"x\tb\ta\n";
	cout<<x<<"\t"<<b<<"\t"<<a<<endl;
	a = b = ++x;
	cout<<d(a , b)<<endl;
	cout<<x<<"\t"<<b<<"\t"<<a<<endl;
	return 0;
}
```

3. ماذا سيكون ناتج البرنامج السابق لو استبدلنا التعليمة `return (x+1)/2.0;` بالتعليمة `return ++x/2.0;` ؟.
4. اكتب برنامجاً يقوم بالعمليّات الرياضيّة الأربع وذلك باستخدام التوابع.
5. باستخدام تابع تعاوديّ اكتب برنامجاً يحسب العدد x^y ثمّ اكتبه باستخدام تابع تكراري.

(**مع ملاحظة أنّ:** $x^y = x.x^{y-1}$ وَ $x^1 = x$ و $x^0 = 1$ و $x^{-y} = \frac{1}{x^y}$)

6. اكتب برنامجاً فيه تابع يقوم بإيجاد مجموع السلسلة التالية:

$$\pi \approx \sum_{i=0}^{n} \frac{4(-1)^i}{2i+1}$$

اعتمد على التمرين السابق لحساب -1^i.

7. اكتب برنامجاً لإيجاد القاسم المشترك الأكبر لعددين باستخدام تابع تكراري و آخر تعاودي علماً أنّ: `GCD(x , y) = GCD(x , x % y)`.

8. اكتب برنامجاً فيه تابع لحساب n! و آخر لحساب $\binom{n}{k} = \frac{n!}{(n-k)!.k!}$
و آخر لحساب $p(n,r) = \binom{n}{r} r!$

9. اكتب تابعاً (إجراءً) اسمه multiple بحيث يأخذ عددين صحيحين كوسيطين ثمّ يعيد (يطبع) true إذا كان العدد الثاني من مضاعفات العدد الأول وإلا فيعيد (يطبع) false.

10. اكتب برنامجاً فيه تابع يقوم بفحص رقم صحيح ممرّر إليه ويعيد قيمة منطقيّة true إذا كان أوليّاً و false إذا لم يكن، ثمّ استخدم إجراءً يقوم بطباعة كل الأرقام الأوليّة بين 1 والرقم المدخل.(يمكنك استخدام التابع السابق داخل هذا الإجراء).
ملاحظة : الرقم الأوّلي هو الرقم الذي يقبل القسمة على نفسه وعلى 1 فقط. والرقم 1 ليس أوليّاً.

11. اكتب برنامجاً فيه تابع لحساب القيمة المطلقة لعدد ما. علماً أنّها تحقّق الشكل التالي:

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

12. اكتب برنامجاً فيه ثلاثة توابع كما يلي:

- يحسب مساحة دائرة بعد إدخال نصف قطرها r $s = \pi.r^2$
- يحسب مساحة مستطيل بعد إدخال طولي ضلعيه a, b $s = a.b$
- يحسب مساحة مثلث بعد إدخال طول قاعدته b و ارتفاعه h $s = \frac{1}{2}b.h$

حيث يعرض البرنامج قائمة فيها الخيارات الممكنة ويختار المستخدم أحدها كما في الشكل التالي:

```
1- Calculate circle space
2- Calculate rectangle space
3- Calculate triangle space
Enter number from list above to select: 2

Calculating rectangle space …
Enter Rectangle width: 5
Enter Rectangle height: 3
S = 15
```

13. اكتب برنامجاً فيه تابع يأخذ وسيطاً يمثّل ارتفاع مثلث ثمّ يطبعه بالشكل التالي:

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

في هذا المثال الارتفاع h = 5 كل سطر يحوي 1- (رقم السطر)*2 نجمة.
اطبع فراغات حتّى أوّل نجمة في السطر ثمّ استخدم العلاقة السابقة لطباعة عدد النجم المطلوبة في كل سطر.

القسم الثاني

أخي لن تنال العِلمَ إلّا بستَّةٍ سأُنبِيكَ عن تفصيلها ببيانِ
ذكاءٌ وحرصٌ واجتهادٌ وبلغةٌ وصحبةُ أستاذٍ وطولُ زمانِ

للأمام الشافعي

الفصل التاسع:

المصفوفات (Arrays)

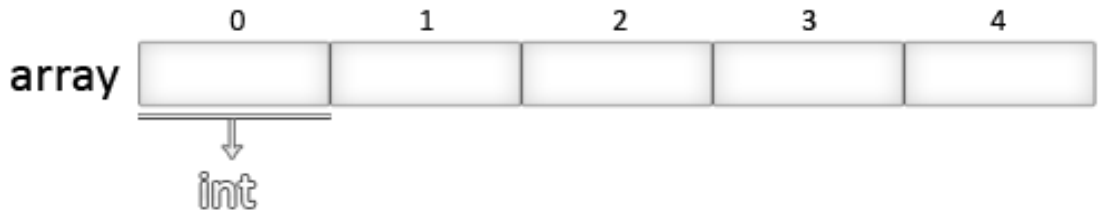
المصفوفة هي سلسلة من العناصر ذات النوع نفسه محفوظة في الذاكرة بشكل متتالي بحيث يمكن الوصول إلى أي عنصر بذكر اسم المصفوفة متبوعاً بالدليل. الشكل العام للتصريح عن المصفوفات هو

```
type name [elements];
```

type: نوع البيانات الخاص بالمصفوفة. **name:** اسم الصفوفة. **elements:** عدد العناصر.

فمثلاً يمكن أن نخزن 5 قيم من النوع int دون الحاجة إلى التصريح عن خمسة متغيّرات بأسماء مختلفة وذلك باستخدام المصفوفات كما يلي:

```
int  array[5] ;
```



أي المصفوفة array تحوي خمس قيم صحيحة (integer) كما في الشكل السابق.

ملاحظة: عند التصريح عن المصفوفة في لغة C++ يجب أن يكون عدد العناصر قيمة ثابتة

حيث إن المصفوفات هي حجرات ذاكرة ستاتيكية يجب أن يُعطى حجم هذه الحجرات

للمترجم compiler ليحدد كم من الذاكرة يجب أن يحجز للمصفوفة قبل تنفيذ البرنامج.

يمكن تعبئة المصفوفة (إسناد القيم لعناصرها) كما يلي

```
int array[5]={1,3,4,2,7};
```

ويمكن في هذه الحالة تجاوز الملاحظة السابقة أي يمكن كتابة

```
int array[]={1,3,4,2,7};
```

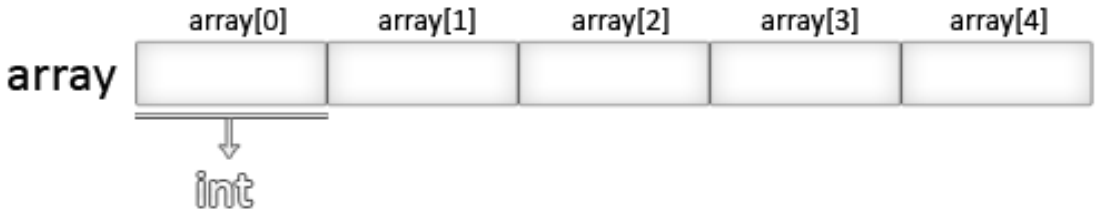
عندما نصرح عن المصفوفات داخل التوابع ولا نضع قيم لعناصرها فإن قيم العناصر ستبقى عشوائية حتى نضع قيم لهذه العناصر ،وعندما نصرح عن المصفوفات خارج أي تابع فإن قيم العناصر تكون صفرية.

التعامل مع المصفوفات

في أي جزء من البرنامج تكون فيه المصفوفة مرئيّة (راجع فقرة المدى) يمكن أن نصل إلى عناصرها للقراءة أو للتعديل على أيّ عنصر منها كما يلي:

اسم المصفوفة [الدليل] name[index]

من المثال السابق تكون العناصر كما في الشكل:



فإذا أردنا إسناد قيمة العنصر الثالث للمتحول a نكتب a=array[2] ;

ولوضع القيمة 10 في العنصر الأول نكتب array[0]=10 ;

للأقواس المربعة استخدامان مختلفان الأول لتحديد حجم المصفوفة عند التصريح عنها، والثاني لتحديد دليل العنصر المراد الوصول إليه من المصفوفة

```
int array[5];        // التصريح عن مصفوفة جديدة يبدأ بتحديد نوعها
array[0] = 10;       // الوصول إلى العنصر الأول من المصفوفة
```

هناك عبارات أخرى صحيحة مثل

```
array [0] = a;
array [a] = 75;
b = array [a+2];
array [array [a]] = array [2] + 5;
```

مثال

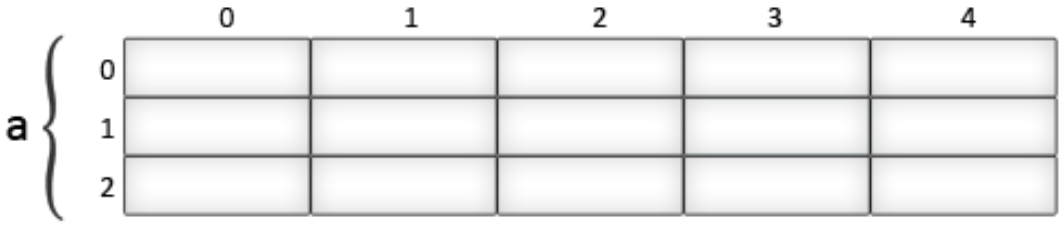
```
// arrays example
#include <iostream>
using namespace std;
int m[] = {16, 2, 77, 40, 12071};
int i,sum=0;
int main ()
{
 for ( i=0 ; i<5 ; i++ )
 {
     sum+= m[i];
 }
  cout <<sum;
 return 0;
}
```

```
12206
```

الخرج هو مجموع عناصر المصفوفة m

المصفوفات المتعددة الأبعاد

يمكن أن توصف المصفوفات المتعددة الأبعاد بأنّها مصفوفات لمصفوفات فالمصفوفة ثنائية البعد يمكن تصورها على أنّها جدول ثنائي البعد **لبيانات من نوع واحد**.

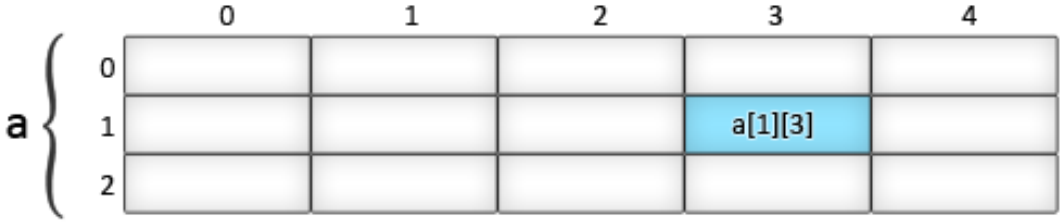


تمثل a مصفوفة ثنائية البعد مؤلفة من 3 أسطر و 5 أعمدة من النوع int تعرّف كما يلي :

```
int  a[3][5];
```

وللوصول إلى العنصر الواقع في السطر الثاني والعمود الرابع نكتب

```
a[1][3]
```



(تذكّر أنّ أدلّة المصفوفة تبدأ دائماً بـ 0)

المصفوفات المتعددة الأبعاد لا تقتصر على دليلين(بعدين) فقط بل يمكن أن تحوي أدلة بحسب الحاجة، لكنّنا نادراً ما نحتاج أكثر من 3 أبعاد حيث تكون كمية الذاكرة المطلوبة لتخزين هذه المصفوفات كبيرة جدّاً فمثلاً

```
char century [100][365][24][60][60];
```

هذه المصفوفة تحجز في الذاكرة متغيّر من نوع char لكل ثانية في القرن أي تحجز حوالي 3 مليارات بايت

إذا صرحنا عن هذه المصفوفة فإنّها ستستهلك حوالي 3 جيغا من الذاكرة RAM.

المصفوفات المتعددة الأبعاد هي مجرد تبسيط لمصفوفة من بعد واحد حيث المصفوفتان التاليتان متكافئتان

int a [3][5]; ↔ int a [15]; (3 * 5 = 15)

والشيفرتان التاليتان متكافئتان أيضاً

```
// multidimensional array
#include <iostream>
using namespace std;

#define WIDTH 5
#define HEIGHT 3

int a [HEIGHT][WIDTH];
int n,m;


int main ()
{
  for (n=0;n<HEIGHT;n++)
    for (m=0;m<WIDTH;m++)
    {
      a[n][m]=(n+1)*(m+1);
    }
  return 0;
}
```

```
// pseudo-multidimensional array
#include <iostream>
using namespace std;

#define WIDTH 5
#define HEIGHT 3

int a [HEIGHT * WIDTH];
int n,m;


int main ()
{
  for (n=0;n<HEIGHT;n++)
    for (m=0;m<WIDTH;m++)
    {
      a[n*WIDTH+m]=(n+1)*(m+1);
    }
  return 0;
}
```

كلا البرنامجين يحجزان ذاكرة للمصفوفة a كما في الجدول التالي :

a	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

استخدمنا #define ليصبح التعديل اللاحق على البرنامج أسهل فلو أردنا أن تكون قيمة البعد HEIGHT 4 بدلاً من 3 نكتب #define HEIGHT 4 بدلاً من #define HEIGHT 3
وذلك أبسط من تعريف المصفوفة بثوابت عددية (int a[3][5];) ثمّ تعديل 3 إلى 4 في كل مرّة تذكر فيها المصفوفة في البرنامج.
كما يمكن أن نسند قيم لعناصر المصفوفة أثناء التصريح كما يلي :

```
int a[3][5]={{1,2,3,4,5},{2,4,6,8,10},{3,6,9,12,15}};
```

حيث إنّ العناصر المحصورة بين الأقواس الداخلية هي عناصر السطر الأول والثاني والثالث بالترتيب

أمّا إذا كتبنا

```
int a[3][5]={{1,3,4,5},{10},{12,15}};
```

فإنّ باقي عناصر المصفوفة تكون أصفاراً كما في الجدول المبين :

a	0	1	2	3	4
0	1	3	4	5	0
1	10	0	0	0	0
2	12	15	0	0	0

المصفوفات كمعاملات(وسطاء) للتوابع

نحتاج أحياناً أن نمرّر مصفوفة كمعامل لتابع ما. في لغة C++ لا يمكن تمرير مجموعة حجرات ذاكرة بالقيمة كمعامل لتابع ما، وبالتالي لا يمكن تمرير كامل المصفوفة، ولكن يمكن تمرير عنوانها الذاكري.

ولكي تصبح المصفوفات معاملات يجب علينا عند التصريح عن التابع المراد التعامل معه أن نضع نوع البيانات للمصفوفة المراد استقبالها واسم للمصفوفة داخل التابع ثمّ أقواس مربعة فارغة فمثلاً التابع

```
void procedure (int arg[])
```

يقبل مصفوفة من النوع int وسيسميها arg ولنمرر مصفوفة لهذا التابع نصرح عنها

```
int myarray [40];
```

ثمّ نستدعي التابع كما يلي

```
procedure (myarray);
```

مثال

```
// arrays as parameters
#include <iostream>
using namespace std;
void printarray(int arg[], int length) {
  for (int n=0; n<length; n++)
    cout << arg[n] << " ";
  cout << "\n";
}
int main ()
{
  int firstarray[] = {5, 10, 15};
  int secondarray[] = {2, 4, 6, 8, 10};
  printarray (firstarray,3);
  printarray (secondarray,5);
  return 0;
}
```

```
5 10 15
2 4 6 8 10
```

كما ترى فإنّ المعامل الأول للتابع printarray هو (int arg[]) يقبل أي مصفوفة من النوع int أياً كان طولها، أمّا المعامل الثاني فهو طول المصفوفة المراد تمريرها وهذا يعطي الحلقة for شرط التوقف.

عند التصريح عن التابع يمكن أيضاً أن نمكّنه من أن يستقبل المصفوفات المتعددة الأبعاد ويكون التصريح عن التابع الذي يقبل المصفوفة

ثنائية الأبعاد

```
void procedure (int myarray[ ][3]);
```

وثلاثية الأبعاد

```
void procedure (int myarray[ ][3][4]);
```

لاحظ أنّ القوس الأول فارغ أمّا باقي الأقواس فيجب أن نحدّد عمقها لأنّ المترجم يجب أن يحدد خلال التابع ما هي قيمة كل بعد إضافي.
عندما تمرّر المصفوفات الأحادية أو المتعددة الأبعاد كمعاملات للتوابع تشكل مصدراً شائعاً للأخطاء عند المبرمجين قليلي الخبرة، ننصح هنا بقراءة فصل المؤشّرات لفهم أفضل لآلية عمل المصفوفات.

الأعداد العشوائية

- هناك العديد من التطبيقات البرمجية الشائعة التي تحتاج إلى المحاكاة كالألعاب والدراسات الاحتمالية والاحصائية.
- يمكن أن نولد الأعداد العشوائية في لغة C++ باستخدام التابع rand() الموجود في المكتبة القياسية <cstdlib> لذلك يجب أن نضمّنها في البرنامج.
- يولد التابع rand() أرقاماً صحيحة موجبة **(unsigned integer)** بين 0 و RAND_MAX الذي هو ثابت رمزي معرف في المكتبة القياسية تكون قيمته 32767 على الأقل.
- لكي نحدد مجال للأعداد التي سيتم توليدها نستخدم العلاقة

```
number = shiftingValue + rand() % scalingFactor;
```

number: العدد المولّد. **shiftingValue:** الإزاحة. **scalingFactor:** طول مجال التوليد.

مثال:

```
number = 1 + rand() % 6;
```

إنّ القيم المحتملة للمتغيّر number هي ضمن المجال [1,6].

- كل مرّة يتمّ فيها تنفيذ البرنامج يتمّ توليد نفس الأعداد لذلك يوجد تابع في المكتبة القياسية اسمه srand() يقوم بتغيير سلسلة الأرقام عند كل تنفيذ للبرنامج ويستقبل هذا التابع قيمة من النوع **unsigned integer** ولكي نضمن تغيّر القيمة المرسلة إلى srand نستخدم التابع time() كالتالي:

```
srand( time( 0 ) );
```

time(0) يقرأ الوقت عند التنفيذ بالثواني ويحوّله إلى **unsigned integer**.
التابع time() موجود في المكتبة <ctime> التي يجب أن نضمّها إلى البرنامج.

مثال:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main ()
{
       int num,guess,guessNum;
    cout<<"\tguessing game\n\n";
       srand(time(0));
       num=1+rand()%100;
       cout<<"enter your guess between 1 and 100: ";

       for(guessNum=0;guess!=num;guessNum++)
       {
              cin>>guess;
              if(guess<num)
                     cout<<"your guess is too low!.  try again: ";
               if(guess>num)
                     cout<<"your guess is too high!. try again: ";
       }
```

```
       cout<<"congratulations. you guessed it!!!. "
           <<"\nyou tried "<<guessNum<<" times to guess it\n";

       return 0;
}
```

```
        guessing game

enter your guess between 1 and 100: 50
your guess is too high!. try again: 25
your guess is too low!.  try again: 37
your guess is too high!. try again: 31
your guess is too high!. try again: 28
congratulations. you guessed it!!!.
you tried 5 times to guess it
```

تمارين محلولة

1- لتكن لدينا المصفوفة A ذات النوع الصحيح قم بطباعتها ثمّ اطبع مجموع عناصرها :

A= [5, 6, 1, 9, 12, 4, 7]

الحل:

```
#include <iostream>
using namespace std;
int main()
{
	int A[]={5,6,1,9,12,4,7};
	int sum=0;
	for(int i=0;i<7;i++)
	{
		cout<<A[i]<<" ";
	  sum+=A[i]; //sum =sum+A[i];
	}
	cout<<"\nsum= "<<sum<<endl;

	return 0;
}
```

2- اكتب برنامجاً لحساب سلسلة فيبوناكسي بطريقة تكراريّة.

الحل:

```
#include<iostream>
using namespace std;
int f[1000];
int fib(int n)
{
 f[0] = 0; f[1] = 1;
 for(int i=2; i<=n;i++)
     f[i] = f[i-1]+f[i-2];
 return f[n];
}
int main()
{
 int n ;
 cout<< "Enter n: ";
 cin>> n;
 cout<< "\nfib(" << n << ") = " << fib(n) << endl;
}
```

3- اكتب برنامجاً يطلب من المستخدم أن يدخل 10 أرقام صحيحة إلى مصفوفة وعدد V ثمّ يقوم بالبحث عن V ضمن عناصر المصفوفة المدخلة ويطبع "V is in the array" إذا كان V موجوداً فيها وإلّا فإنّه يطبع "V is not in the array".

الحل:

```
#include <iostream>
using namespace std;
int main()
{
	int arr[10],V,i;
	bool found=false;
    cout<<"enter 10 numbers please!\n";
	for(i=0;i<10;i++)
		{
		cout<<"number"<<i+1<<"= ";
		cin>>arr[i];
		}
		cout<<"enter V: ";
		cin>>V;
	for(i=0;i<10;i++)
			if(arr[i]==V)
			{
			   found=true;
			   break;
			}

		if(found) //same as if(found==true)
			cout<<"V is in the array\n";
		else
			cout<<"V is not in the array\n";
	return 0;
}
```

4- اكتب برنامجاً لقراءة مصفوفة ثنائية الأبعاد $X_{4\times4}$ ثمّ عمل مايلي :

- طباعة المصفوفة بشكل مناسب باستخدام تابع print().
- طباعة عدد العناصر(الموجبة ،السالبة ، الصفرية).
- طباعة عدد العناصر الزوجية والفردية.
- طباعة عناصر القطر الرئيسي ثمّ عناصر القطر الثانوي.

الحل:

```
#include <iostream>
using namespace std;

void print(int a[][4])
{
	int i,j;
	for(i=0;i<4;i++)
	{
		for(j=0;j<4;j++)
			cout<<a[i][j]<<"\t";
		cout<<"\n\n";
	}

}
int main()
{
```

```
	int X[4][4],i,j;
	int positives=0,negatives=0,zeros=0;
	int even=0,odd=0;
	for(i=0;i<4;i++)
		for(j=0;j<4;j++)
		{
			cout<<"X["<<i+1<<"]["<<j+1<<"]= ";
			cin>>X[i][j];
			if(X[i][j]>0)
				positives++;
			 if(X[i][j]<0)
				negatives++;
			if(X[i][j]==0)
				zeros++;
			if(X[i][j]%2==0)
				even++;
			if(X[i][j]%2!=0)
				odd++;
		}
    cout<<"\n";
    print(X);//printing X
	   cout<<"\nnumber of positive elements="<<positives;
	   cout<<"\nnumber of negative elements="<<negatives;
	   cout<<"\nnumber of zeroth elements="<<zeros;
	   cout<<"\nnumber of even elements="<<even;
	   cout<<"\nnumber of odd elements="<<odd;
	   cout<<"\nleading diagonal:  ";
	   for(i=0;i<4;i++)
			cout<<X[i][i]<<"\t";
	   cout<<"\ncross diagonal:    ";
	   for(i=0;i<4;i++)
	   {
		for(j=0;j<4;j++)
			if(i+j==4-1)
				cout<<X[i][j]<<"\t";
	   }
    cout<<endl;
	   return 0;
}
```

5- اكتب برنامجاً يقوم بجمع مصفوفتين ثنائيتي الأبعادعلماً أن شرط الجمع هو أن يكون للمصفوفتين نفس الأبعاد.

الحل:

```
// program to add matrix to another one
#include <iostream>
using namespace std;
int main()
{
	int a[10][10],b[10][10],c[10][10];
	int i,j,m,n;
	cout<<"number of rows=";cin>>m;
	cout<<"number of columns=";cin>>n;
	for(i=0;i<m;i++)
		for(j=0;j<n;j++)
```

```
            {
                  cout<<"a["<<i+1<<"]["<<j+1<<"]=";cin>>a[i][j];
                  cout<<"b["<<i+1<<"]["<<j+1<<"]=";cin>>b[i][j];
                  c[i][j]=a[i][j]+b[i][j];
             }

      for(i=0;i<m;i++)
      {
            for(j=0;j<n;j++)
              cout<<a[i][j]<<"  ";
        cout<<"\n";
       }
      cout<<"\n  +\n\n";
     for(i=0;i<m;i++)
      {
       for(j=0;j<n;j++)
            cout<<b[i][j]<<"  ";
      cout<<"\n";
      }
      cout<<"\n  =\n\n";
      for(i=0;i<m;i++)
      {
       for(j=0;j<n;j++)
            cout<<c[i][j]<<"  ";
       cout<<"\n";
      }
      return 0;
}
```

6- اكتب برنامجاً يقوم بضرب مصفوفتين ثنائيتي الأبعادعلماً أن شرط الضرب هو أن يكون عدد أعمدة المصفوفة الأولى يساوي عدد أسطر المصفوفة الثانية.

الحل:

```
//matrix multiplication
#include<iostream>
using namespace std;
#define M 20
int a[M][M],b[M][M],c[M][M];//all arrays' elements equal zero
void main()
{
      int ma,na,mb,nb,i,j,k;

      /*ma=number of A rows,na=number of A columns,
        mb=number of B rows,nb=number of B columns,
        i,j,k:counters */

      cout<<"enter dimensions of A(ma,na): ";cin>>ma>>na;
    cout<<"enter dimensions of B(mb,nb): ";cin>>mb>>nb;
      if(na!=mb)
            cout<<"verify multiplication condition na=mb\n";
      else
      {     cout<<"enter A elements\n";//reading A elements
       for(i=0;i<ma;i++)
              for(j=0;j<na;j++)
              { cout<<"a["<<i+1<<"]["<<j+1<<"]= ";cin>>a[i][j];}
```

```
          cout<<"enter B elements\n";//reading B elements
         for(i=0;i<mb;i++)
                for(j=0;j<nb;j++)
                { cout<<"b["<<i+1<<"]["<<j+1<<"]= ";cin>>b[i][j];}

          for(i=0;i<ma;i++)//multiplication
           for(j=0;j<nb;j++)
               for(k=0;k<na;k++)
                      c[i][j]+=(a[i][k]*b[k][j]);

          cout<<"the result C=\n\n";
           for(i=0;i<ma;i++)
              {for(j=0;j<nb;j++)
                 cout<<c[i][j]<<"\t";
                 cout<<"\n\n";}
       }
}
```

7- عرّف مصفوفة من النوع (string) تحوي أسماء الأشهر ثمّ مرر هذه المصفوفة إلى تابع (print) ليقوم بطباعتها.

الحل:

```
#include<iostream>
#include<string>
using namespace std;
void print(string m[],int n)
{
       for(int i=0;i<n;i++)
              cout<<m[i]<<"\n";
}
int main()
{
   string months[12]={"January","February","March"
                      ,"April","May", "June"
                      ,"July","August","September"
                       ,"October","November","December"};
       print(months,12);
       return 0;
}
```

تمارين

1- أوجد الأخطاء في كل من العبارات التالية:

a) بفرض
```
int b[ 10 ] = { 0 } ;
for ( int i = 0; i<= 10; i++ )
    b[ i ] = 1;
```

b) بفرض
```
int a[ 3 ];
cout<<a[1]<<" "<<a[2]<<" "<<a[3]<<endl;
```

c)
```
double f[ 3 ] = { 1.1, 10.01, 100.001, 1000.0001 };
```

d) بفرض
```
double d[ 2 ][ 10 ];
d[ 1, 9 ] = 2.345;
```

2- املأ الفراغات التالية :

a) أسماء العناصر الأربع للمصفوفة p (int p[4];) هي _____ و _____ و _____ و _____.

b) تسمية المصفوفة بتحديد نوعها وعدد عناصرها يدعى _____ المصفوفة.

c) اسم العنصر الواقع في السطر 3 والعمود 5 في المصفوفة d هو _____.

d) المصفوفة المؤلفة من 4 أسطر و 5 أعمدة تحوي _____ عنصراً.

3- اكتب التعليمة أو التعليمات اللازمة للقيام بالمهام التالية :

a) صرّح عن متغيّر ثابت باسم arraySize وأعطه القيمة 10.

b) صرّح عن المصفوفة fractions بعدد عناصر arraySize ونوع double وأعط عناصرها القيمة 0.

c) ضع القيمة 1.667 في العنصر التاسع والقيمة 3.333 في العنصر السابع.

d) اطبع المصفوفة باستخدام حلقة for.

4- رتب المصفوفة التالية تصاعدياً ثمّ تنازلياً واطبعها في الحالتين

A = [1,5,4,3,7,2,9,0]

5- اكتب برنامجاً يطلب من المستخدم إدخال مصفوفتين A و B (من النوع integer) كل منهما مكونة من 10 عناصر , يجب على البرنامج أن يضم المصفوفة B إلى نهاية المصفوفة A ويضع الناتج في المصفوفة C المؤلفة من 20 عنصر ثمّ يطبع المصفوفة C.

6- اكتب برنامجاً يطلب من المستخدم إدخال مصفوفة (integer) من 10عناصر ثمّ يطبع البرنامج المصفوفة تصاعدية " the array is growing " أو تنازلية "the array is decreasing" أو ثابتة "the array is constant" أو تصاعدية وتنازلية "the array is growing and decreasing."

7- اكتب برنامجاً يمكّن المستخدم من إدخال عدد الطلاب n وأسمائهم names ودرجاتهم marks في مقرر ما ثمّ يطبع أسماء الطلاب حسب درجاتهم في المقرر وبترتيب تنازلي.

8- اكتب برنامجاً يقوم بجمع مصفوفتين ثنائيتي الأبعاد علماً أن شرط الجمع هو أن يكون للمصفوفتين نفس الأبعاد باستخدام التوابع. (التمرين محلول في التمارين المحلولة بدون استخدام التوابع).

9- اكتب برنامجاً يقوم بتنفيذ اللعبة التالية

لعبة (ورقة – مقص – حجر)

تحتاج هذه اللعبة إلى لاعبين فقط حيث يخفي كل منهما يده ويختار أحد الخيارات الثلاث ويكون الرابح وفق الجدول التالي:

الخيار الأول	الخيار الثاني	النتيجة
ورقة	حجر	الورقة تغطي الحجر لذلك تربح نقطة
حجر	مقص	الحجر يكسر المقص لذلك يربح نقطة
مقص	ورقة	المقص يقص الورقة لذلك يربح نقطة

الفصل العاشر:

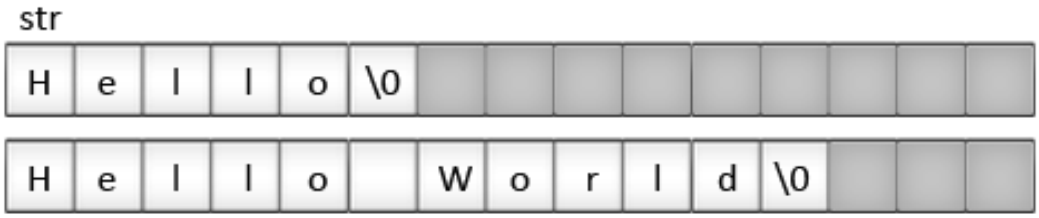
سلاسل الرموز (Strings of Characters)

تمكّننا السلاسل من التعامل مع النصوص سواءً منها الكلمات أو الجمل أو الأسماء.. .
السلسلة مؤلفة من تتابع من العناصر ذات النوع char حيث إنّ كل عنصر يمثل حرف في السلسلة. فمثلاً المصفوفة (سلسلة الرموز) التالية :

```
char  str [20];
```

يمكن أن تخزن سلسلة مؤلفة من 15 حرف كما في الشكل

ليس ضرورياً أن نملأ كل عناصر السلسلة فيمكن أن تحوي السلسلة في نقطة ما من البرنامج الكلمة "Hello" التي تملأ 5 خانات أو العبارة "Hello World" التي تملأ 12 خانة.
عند نهاية السلسلة المخزنة في كلا الحالتين نصل إلى الرمز null الذي يمكن أن يكتب 0 أو ' \0 '



أما العناصر الواقعة في المنطقة الرمادية فقيمها غير محددة.

التعامل مع السلاسل

بما أنّ السلاسل هي مصفوفات طبيعية فإنّها تخضع لنفس قواعد المصفوفات فإذا أردنا مثلاً أن نضع القيم في السلسلة أثناء التصريح نكتب :

```
char mystring[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

أو

```
char mystring[ ] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

ولكن هناك طريقة أخرى لعمل ذلك وهي:

```
char mystring [ ] = "Hello";
```

حيث إنّ السلاسل المحصورة بعلامات اقتباس مزدوجة يضاف إلى نهايتها الثابت null بشكل آلي.

إنّ إسناد الثوابت النصية مثل "Hello" للسلاسل صحيح فقط أثناء التصريح عن السلسلة (المصفوفة)
لذلك فالعبارات التالية غير صحيحة:

```
mystring = "Hello";
mystring[] = "Hello";
mystring = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

السبب في ذلك يعود إلى أنّ المصفوفة هي عبارة عن مؤشّر ثابت يشير إلى كتلة ذاكرة محجوزة وبسبب هذا الثبات (في قيمة المؤشّر) فلا يمكن أن نسند أيّة قيمة للمصفوفة بحدِّ ذاتها ولكن يمكن إسناد قيمة لأيّ عنصر من عناصر هذه المصفوفة.
لذلك فإنّ الطرف الأيسر للإسناد يمكن أن يكون فقط عنصر من مصفوفة وليس مصفوفة كاملة.
إذاً يمكن أن نكتب :

```
mystring[0] = 'H';
mystring[1] = 'e';
mystring[2] = 'l';
mystring[3] = 'l';
mystring[4] = 'o';
mystring[5] = '\0';
```

لا تبدو هذه طريقة عملية لذلك سنستخدم توابع جاهزة مثل التابع strcpy المعرّف في المكتبة cstring (string) ويُستدعَى بالطريقة التالية:

```
strcpy (string1, string2);
```

هذا التابع يقوم بنسخ محتوى string2 ويضعه في string1

string2 يمكن أن يكون مصفوفة أو مؤشّراً أوسلسلة ثابتة.

إذاً لإسناد "Hello" إلى السلسلة mystring نكتب:

```
strcpy (mystring, "Hello");
```

مثال

```
// setting value to string
#include <iostream>
using namespace std;
#include <string>
int main ()
{
  char MyName[20];
  strcpy (MyName,"Basil");
  cout << MyName;
  return 0;
}
```

```
Basil
```

استخدمنا المكتبة <string> لنتمكن من التعامل مع التابع strcpy
يمكن أن نكتب التابع setstring المؤدي نفس الوظيفة دون الاستعانة بـ strcpy

```
// setting value to string
#include <iostream>
using namespace std;
void setstring(char Out[],char In[])
{
  int n=0;
  do {
    Out[n] = In[n];
  } while (In[n++] != '\0');
}
int main ()
{
  char MyName [20];
  setstring (MyName,"Basil");
  cout <<MyName;
  return 0;
}
```

```
Basil
```

هناك طريقة أخرى تعتمد على تعليمة الإدخال cin في هذه الحالة يتمّ إدخال السلسلة من قبل المستخدم أثناء تنفيذ البرنامج حيث نستخدم هنا التابع getline حسب الشكل العام

cin.getline (char buffer[], int length, char delimiter = ' \n');

buffer: هو المصفوفة التي سيُخزَّنُ فيها الدخل.
length: هو الطول الأعظمي للمصفوفة buffer
delimiter: هو الرمز الذي يحدّد نهاية إدخال المستخدم (إذا لم نضع هذا المعامل فإنّه يكون بشكل افتراضي رمز السطر الجديد '\n' أي عندما نضغط enter)

فمثلاً يقوم هذا البرنامج بتكرار ماتكتبه على لوحة المفاتيح

```
// cin with strings
#include <iostream>
using namespace std;
int main ()
{
  char mybuffer [100];
  cout << "What's your name?\n";
  cin.getline (mybuffer,100);
  cout << "Hello " << mybuffer << ".\n";
  cout << "Which is your favourite book?\n";
  cin.getline (mybuffer,100);
  cout << "I like " << mybuffer << " too.\n";
  return 0;
}
```

```
What's your name?
Sameer
Hello Sameer.
Which is your favourite
book? The Holy Qura’an
I like The Holy Qura’an too.
```

عندما استخدمنا cin.getline لأول مرّة كانت قيمته تخزّن الاسم Sameer في المصفوفة mybuffer أما في المرّة الثانية فإنّ التابع يقوم بالكتابة فوق المحتوى الأول للمصفوفة أي يخزن The Holy Qura’an في المصفوفة.

ملاحظة: يمكن إدخال السلاسل باستخدام التعليمة ; cin >> mybuffer
هذه الطريقة مقبولة لكنها تسمح لنا بإدخال الكلمات فقط و لا تستقبل من الجملة الحاوية على فراغات إلا أول كلمة ، كما لا يمكن تحديد طول المصفوفة mybuffer التي سيدخلها المستخدم أي إذا أدخل عدداً من الأحرف أكبر من طول المصفوفة المحجوزة سيتوقف تنفيذ البرنامج لذلك من الأفضل استخدام cin.getline عند التعامل مع السلاسل.

الفصل الحادي عشر:

المؤشّرات (pointers)

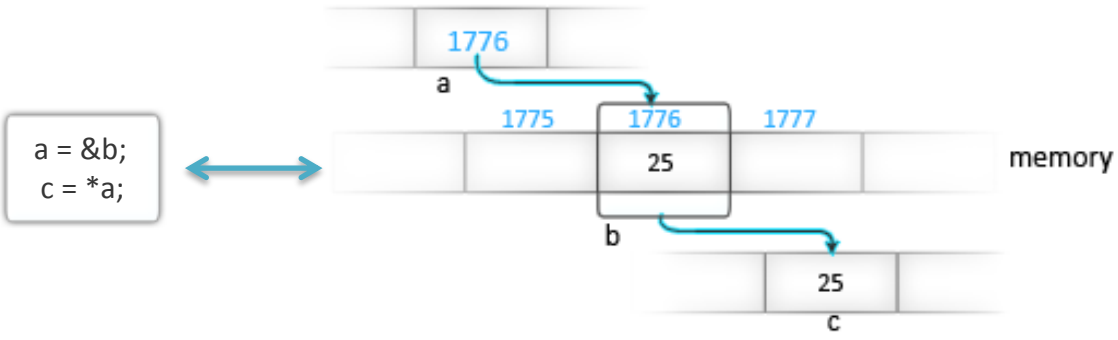
إنّ ذاكرة الحاسوب ماهي إلّا تتابع من خلايا حجم كل منها واحد بايت، ولكل واحدة من هذه الخلايا عنوان مميز حيث يتولى نظام التشغيل عنونة الذاكرة بأرقام متتابعة.
المؤشّر هو متغيّر كسائر المتغيّرات ولكنه يختلف عنها فيما يختزنه فهو لا يختزن البيانات العادية كالأرقام والرموز وإنّما يختزن عنوان المتغيّر الذي يشير إليه.

معامل العنوان (&)

عندما نصرّح عن متغيّر فإنّه سيخزّن في إحدى الخلايا المتاحة في الذاكرة بشكل آلي من قبل المترجم ونظام التشغيل، فإذا أردنا معرفة مكان تخزين هذا المتغيّر نسبق اسم المتغيّر بالرمز **&**.
مثلاً التعليمة a = &b; ستضع عنوان المتغيّر b في المتغيّر a.
إنّ المتغيّر الذي يحوي عنوان متغيّر آخر يدعى **مؤشّراً** (مثل المتغيّر a في السطر السابق).

معامل المرجع (*)

إذا سبقنا المؤشّر a بمعامل المرجع * أي إذا كتبنا *a فإنّ العبارة c = *a; تعني ضع القيمة التي يُشيرُ إليها المؤشّر a في المتغيّر c.



يجب الانتباه إلى الفرق بين العبارتين

```
c = a;   // c == 1176
c = *a;  // c = =25
```

مما سبق تكون العبارات التالية صحيحة

```
b == 25
&b == 1776
a == 1776
*a == 25
*a == b
```

التصريح عن المؤشّرات

الشكل العام للتصريح هو

type * pointer_name;

type: هو نوع البيانات التي سيشير إليها المؤشّر وليس نوع المؤشّر ذاته
pointer_name: اسم المؤشّر.

أمثلة

int * number;
char * character;
float * greatnumber;

لدينا ثلاثة مؤشّرات كل منها يشير إلى نوع مختلف من البيانات مع ذلك فإنّ كلّ واحد من هذه المؤشّرات يحجز نفس الحجم في الذاكرة بحسب نظام التشغيل أمّا البيانات التي تُشير إليها هذه المؤشّرات لها حجوم مختلفة في الذاكرة كما أنها من أنواع مختلفة.

مثال 1

```
// my first pointer
#include <iostream>
using namespace std;
int main ()
{
  int a = 5, b = 15;
  int *ptr;
   ptr= &a;
  *ptr = 10;
   ptr = &b;
  *ptr = 20;
  cout<<"a="<<a<<" , b="<<b;
  return 0;
}
```

```
a=10 , b=20
```

- ✓ لاحظ أنه تم تعديل قيم a و b بشكل غير مباشر حيث جعلنا المؤشّر ptr يشير إلى a عن طريق الرمز & ثمّ أسندنا القيمة 10 إلى مايشير إليه المؤشّر ptr وكذلك الأمر بالنسبة لـ b.
- ✓ كما يمكن أن يأخذ المؤشّر خلال البرنامج عناوين مختلفة حيث استخدمنا المؤشّر ptr ليشير إلى a ثمّ إلى b.

مثال 2

```
// more pointers
#include <iostream>
using namespace std;
int main ()
{
  int a = 5, b = 15;
  int *p1, *p2;
   p1 = &a;      // a عنوان = p1
   p2 = &b;      // b عنوان = p2
  *p1 = 10;      // 10=p1 القيمة التي يشير إليها
  *p2 = *p1;     // p1 القيمة التي يشير إليها = p2 القيمة التي يشير إليها
   p1 = p2;      // p1 = p2
  *p1 = 20;      //  20 = p1 القيمة التي يشير إليها
  cout << "a=" << a << " , b=" << b;
  return 0;
}
```

```
a=10 , b=20
```

المؤشّرات والمصفوفات

إنّ المصفوفة تشبه المؤشّر إلى حدّ كبير فعندما نصرح عن مصفوفة فإنّنا نضع في الذاكرة عنوان أول عنصر من عناصرها كما أنّنا عندما نصرح عن مؤشّر فإنّه يشير إلى أول عنصر لذلك فهما متشابهان.

لنفرض التصريحين التاليين:

```
int  numbers [20];
int * p;
```

إنّ الإسناد p = numbers; صحيح لأنّ p و numbers متكافئان ولهما نفس الخصائص، الاختلاف الوحيد هو أنّنا نستطيع أن نعطي قيمة مختلفة للمؤشّر p في حين أنّ numbers سيشير دوماً إلى أول عنصر من العشرين عنصر. لذلك فإنّ p هو مؤشّر عادي إلى المتغيّرات أمّا numbers فهو مؤشّر ثابت (اسم المصفوفة هو مؤشّر ثابت).

أمّا الإسناد numbers = p; فليس صحيحاً لأنّ numbers مصفوفة (مؤشّر ثابت) ولا يمكن إسناد قيمة للمؤشّر الثابت.

مثال 3

```
// more pointers
#include <iostream>
using namespace std;
int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;
}
```

```
10, 20, 30, 40, 50,
```

في فصل المصفوفات استخدمنا الأقواس المربعة للتعامل مع عناصر المصفوفة، هناك طريقة مكافئة باستخدام المؤشّرات كما يلي (إذا كان المؤشّر p يشير إلى المصفوفة a):

*p	a[0]
*(p+1)	a[1]
*(p+2)	a[2]
...	...
*(p+i)	a[i]
.	.
.	.

فإذا أردنا وضع القيمة 0 في العنصر الخامس للمصفوفة a فهناك طريقتان متكافئتان

```
a[4] = 0;
*(p+4) = 0;
```

تبدئة المؤشّرات

يمكن عند التصريح عن المؤشّرات جعلها تشير إلى المتغيّر المراد أي

```
int number;
int * p = &number;
```

وهذا يكافئ

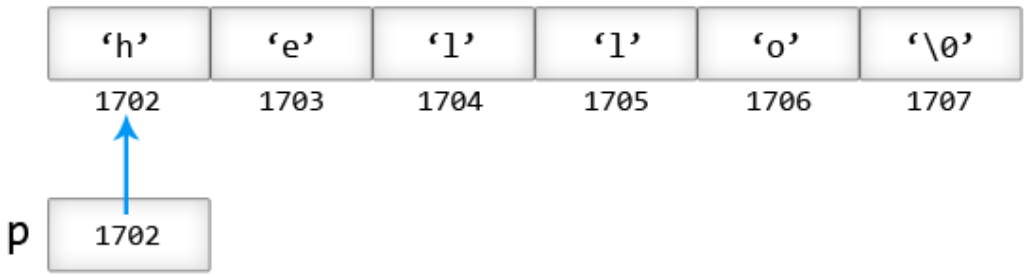
```
int number;
int * p;
 p = &number;
```

في المؤشّرات نسند فقط عناوين المتغيّرات للمؤشّرات وليس إسناد قيم المتغيّرات للمؤشّرات.
فلا يجوز في أي مكان عدا مكان التصريح أن نكتب *p = &number; لأنّ *p هو قيمة مايشير إليه المؤشّر ولا يجوز أن نسند إليها عنوان المتغيّر.

أمّا في حالة المصفوفات فإنّ المترجم يسمح لنا بأن نسند الثابت الذي سيشير إليه المؤشّر أثناء التصريح عن المؤشّر (هنا الثابت هو سلسلة رموز):

```
char * p = "hello";
```

في هذه الحالة تمّ حجز حجم ثابت من الذاكرة لتخزين السلسة "hello" وتمّ جعل المؤشّر p يشير إلى أول رمز (char) لمجموعة الحجرات الذاكريّة التي تمّ حجزها أي أنّ p يشير إلى الحجرة التي تحوي الحرف 'h'
فإذا فرضنا أنّ "hello" خُزّنت في العنوان 1702 وما بعده فإنّ الذي تمّ يُلخصه الشكل التالي:



مصفوفة المؤشّرات

يمكن أن نعرّف مصفوفة مؤشّرات كما يلي :

```
type* name[ELEMENTS];
```

مثال :

```
char* cars[3];       // مصفوفة 3 مؤشّرات على سلاسل

char[0]="Mercedes";
char[1]="Nissan";
char[2]="Ferrari";
```

العمليّات الحسابية على المؤشّرات

العمليّات الحسابية على المؤشّرات تختلف قليلاًعن تلك التي ننفذها على الأعداد الصحيحة إذ يمكن أن ننفذ فقط عمليتي الجمع والطرح أمّا الضرب والقسمة فليس لهما أي معنى فى عالم المؤشّرات. لكن الجمع والطرح لهما سلوك مختلف مع المؤشّرات وذلك بحسب حجم نوع المعطيات التي تشير إليها المؤشّرات.

كما نعلم فإنّ مقدار ما يتمّ حجزه من الذاكرة يعتمد على نوع المعطيات فمثلاً النوع (*char*) يحجز 1 بايت والنوع (*short*) يحجز 2 بايت أما النوع (*long*) فيحجز 4 بايت ولنفرض أنّه لدينا ثلاثة مؤشّرات

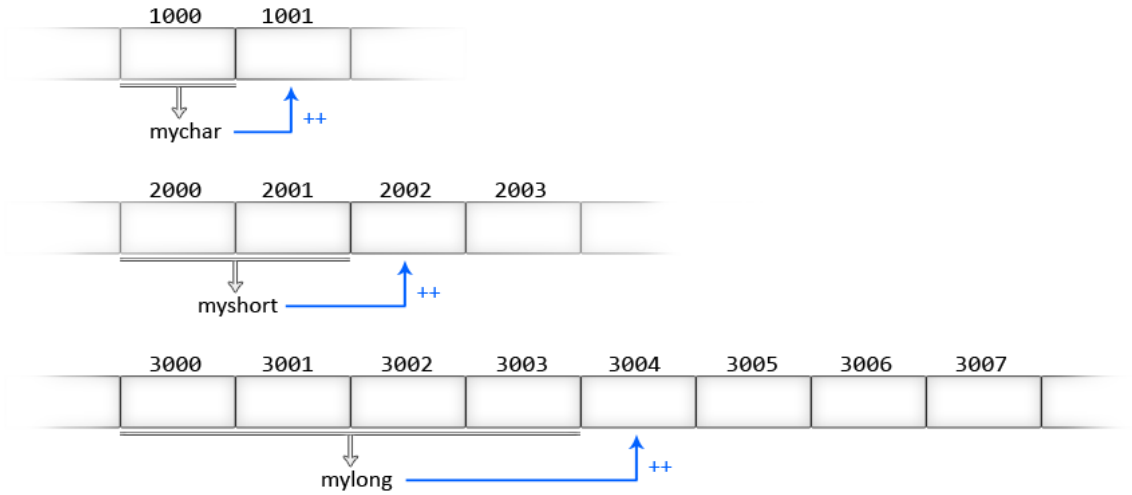
```
char *mychar;
short *myshort;
long  *mylong;
```

وبفرض أنّها تشير إلى حجرات الذاكرة 1000 ، 2000 ، 3000 بالترتيب فإذا كتبنا

```
mychar++;
myshort++;
mylong++;
```

عندئذٍ المؤشّر mychar سيحوي القيمة 1001 و myshort سيحوي القيمة 1002 و mylong سيحوي القيمة 1004 وسبب ذلك أننا عندما نضيف 1 إلى المؤشّر معنى ذلك أننا نجعله يشير إلى العنصر التالي لنفس النوع الذي عرفناه أي يضاف الحجم بالبايت إلى المؤشّر حسب النوع.

حجم النوع (*char*) 1 بايت لذلك تم إضافة 1 إلى المؤشّر mychar
حجم النوع (*short*) 2 بايت لذلك تم إضافة 2 إلى المؤشّر myshort
حجم النوع (*long*) 4 بايت لذلك تم إضافة 4 إلى المؤشّر mylong



سيحدث نفس الأمر تماماً إذا كتبنا

```
mychar = mychar + 1;
myshort = myshort + 1;
mylong = mylong + 1;
```

من المهم التنبيه إلى أنّ معاملا الزيادة (++) والنقصان (--) لهما أولويّة على معامل المرجع (*) لذلك فإنّ التعابير التالية ربما تسبب التباس في فهمها

```
*p++;
*p++ = *q++;
```

التعبير الأول مكافئ لـ (*(p++) وهو يقوم بزيادة 1 إلى العنوان الذي يحويه المؤشّر p وليس القيمة التي يشير إليها وبالتالي فإنّ القيمة الجديدة التي يشير إليها p هي قيمة عشوائية.
في التعبير الثاني بما أنّ المعاملين (++) و (--) بعد المساواة التي يجب أن تتحقّق فإنّ القيمة *q تُسند إلى *p ثمّ يُزاد كلّ من p و q بمقدار 1 أي التعبير الثاني مكافئ لـ

```
*p = *q;
 p++;
 q++;
```

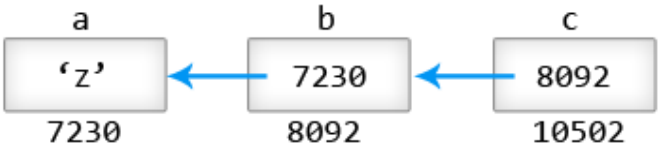
لذلك ينبغي استخدام الأقواس لتجنب الأخطاء.

المؤشّرات على المؤشّرات

يمكن في لغة C++ استخدام مؤشّرات تشير إلى مؤشّرات أخرى والتي تشير بدورها إلى بيانات. لفعل ذلك نحتاج فقط إلى أن نضيف * إلى كلّ مستوى من التأشير فمثلاً:

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```

بفرض حجرات الذاكرة التي حُجزت عشوائياً مبيّنة بالشكل:



الجديد في هذا المثال هو المتغيّر c الذي يمكن أن يدلّ على ثلاث قيم مختلفة

c هو متغيّر من النوع (char **) وقيمته 8092
*c هو متغيّر من النوع (char *) وقيمته 7230
**c هو متغيّر من النوع (char) وقيمته 'z'

ملاحظة

```
*c == b == 7230
**c == *b == a == 'z'
```

المؤشّرات الخالية (*void* pointers)

المؤشّر ذو النوع *void* يمكنه أن يشير إلى أي نوع من المعطيات سواءً كان صحيحاً (*integer*) أو حقيقياً (*float*) أو سلسلة رموز(*string of characters*). في هذا النوع من المؤشّرات لا نستطيع أن نستخدم معامل المرجع * بشكل مباشر لأنّ طول المؤشّر غير معلوم، ولهذا السبب يجب علينا أن نلجأ دائماً إلى تحويل النوع *void* إلى نوع آخر محدَّد من المعطيات.

من فوائد هذا المؤشّر هو إمكانية تمرير قيم عامّة إلى تابع ما كما في المثال التالي :

```
// integer increaser
#include <iostream>
using namespace std;

void increase (void* data, int type)
{
  switch (type)
  {
    case sizeof(char) : (*((char*)data))++; break;
    case sizeof(short): (*((short*)data))++; break;
    case sizeof(long) : (*((long*)data))++; break;
  }
}

int main ()
{
  char a = 5;
  short b = 9;
  long c = 12;
  increase (&a,sizeof(a));
  increase (&b,sizeof(b));
  increase (&c,sizeof(c));
  cout << (int) a << ", " << b << ", " << c;
  return 0;
}
```

```
6, 10, 13
```

sizeof هو تابع مدمج في اللغة يرجع قيمة ثابتة تمثل الحجم بالبايت لما بين القوسين، فمثلاً sizeof(char) هو 1 لأنّ طول النوع *char* هو 1 بايت.

المؤشّرات على التوابع

من خلال المؤشّرات على التوابع يمكن أن نمرّر التابع كمعامل لتابع آخر. نصرّح عن المؤشّر على تابع بنفس الأسلوب الذي نصرّح به عن تابع باستثناء أنّنا نضع الاسم بين قوسين ونسبقه بـ *.

مثال:

```
// pointer to functions
#include <iostream>
using namespace std;

int addition(int a, int b)
{ return (a+b); }

int subtraction(int a, int b)
{ return (a-b); }

int (*minus)(int,int) = subtraction;//pointer to function

int operation(int x, int y, int (*p)(int,int))
{
  int g;
  g = (*p)(x,y);
  return (g);
}
```

```
8
```

```
int main ()
{
  int m,n;
  m = operation(7, 5, addition);
  n = operation(20, m, minus);
  cout <<n;
  return 0;
}
```

minus هو مؤشّر عام على تابع من النوع *int* يقبل معاملين من النوع *int*. أسندنا التابع subtraction إلى المؤشّر minus في سطر التعريف نفسه

int (minus)(int,int) = subtraction;*

الفصل الثاني عشر:

الذاكرة الديناميكية (Dynamic Memory)

حتّى الآن كان حجز الذاكرة في برامجنا يتمّ أثناء كتابة الكود أي أثناء التصريح عن المتغيّرات المختلفة قبل تنفيذ البرنامج. **ولكن ماذا لو احتجنا مقداراً متغيّراً من الذاكرة يتمّ تحديده أثناء تنفيذ البرنامج؟**
مثلاً في حال احتجنا إلى دخل من المستخدم لنحدّد الحجم اللازم حجزه.
الإجابة على ذلك هي *الذاكرة الديناميكية* التي من أجلها تحوي C++ معاملي *new* و *delete*

المعامل new

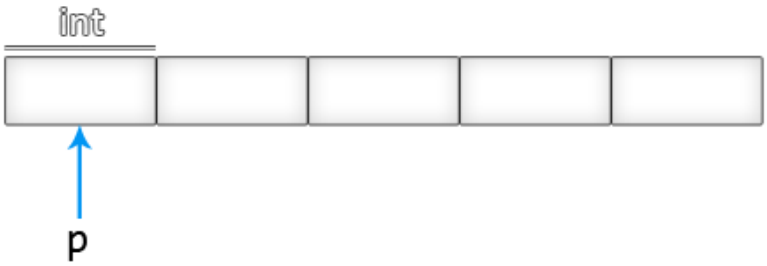
- لكي نطلب ذاكرة ديناميكية نستخدم المعامل *new* متبوعاً بنوع المعطيات أو متبوعاً بنوع المعطيات و قوسين مربعين [ ] يحويان عدد العناصر المطلوبة.
- يعيد المعامل *new* مؤشّراً يشير إلى أول حجرة من حجرات الذاكرة التي تمّ حجزها.

نستخدمه كما يلي
لحجز ذاكرة تحوي عنصر واحد من نوع المعطيات. *pointer* = **new** *type;*
أو
لحجز ذاكرة تحوي مجموعة عناصر(مصفوفة) من نوع المعطيات *pointer* = **new** *type[elements];*
مثال:

```
int * p;
p = new int [5];
```

في هذه الحالة يقوم النظام بحجز مقدار من الذاكرة مؤلّف من 5 حجرات من النوع `int` ويعيد مؤشّراً يشير إلى أول حجرة من الحجرات الخمس وتمّ إسناد قيمة هذا المؤشّر إلى المؤشّر p.

إذاً p يشير إلى أول خانة من 5 خانات من النوع `int` كما في الشكل



سابقاً عند التصريح عن المصفوفة كان يجب أن يكون عدد العناصر قيمة ثابتة أمّا في الذاكرة الديناميكية فيمكن أن يكون عدد العناصر متغيّراً يقوم المستخدم بإدخاله
العديد من البرامج تستخدم الذاكرة الديناميكية و يتمّ أحياناً استهلاك الذاكرة بشكل كامل لذلك يجب التأكد من أنّ الحجز قد تمّ بشكل صحيح عند استخدام *new* كما يلي

```
int * p;
p = new int [5];
if (p == NULL)
   exit(1); // error assigning memory.
```

المعامل *delete*

بعد أن تنتهي حاجتنا إلى الذاكرة التي حجزناها باستخدام *new* ينبغي علينا أن نحرّرها باستخدام المعامل *delete* لكي تصبح متاحةً للحجز لاحقاً، ونستخدمه كما يلي:

لتحرير الذاكرة المحجوزة لعنصر واحد
delete *pointer*;

أو

لتحرير الذاكرة المحجوزة لعدة عناصر (مصفوفة).
delete [] *pointer*;

مثال:

```
//calculate average using dynamic memory
#include <iostream>
#include<stdlib.h>
using namespace std;


int main ()
{
  int n,i,sum=0;
  cout<<"enter the number of courses: ";
  cin>>n;
  int *p=new int[n];

  if(p==NULL)exit(1);

  for(i=0;i<n;i++)
  {
        cout<<"mark "<<i+1<<"= ";
        cin>>p[i];//or cin>>*(p+i)
        sum+=p[i];
  }
  cout<<"average= "<<(float)sum/n<<endl;
  delete [] p;
  return 0;

}
```

```
enter the number of courses: 7
mark 1= 91
mark 2= 78
mark 3= 77
mark 4= 68
mark 5= 83
mark 6= 73
mark 7= 81
average= 78.7143
```

NULL هي قيمة ثابتة معرّفة في مكتبات C++ لتدلّ على المؤشّرات الفارغة (التي لا تشير إلى شيء).
في حال لم تكن معرّفة فيمكن أن تعرّفها بنفسك كما يلي: #define NULL 0
لا يوجد فارق أن تضع 0 أو NULL عندما تفحص المؤشّرات، ولكن NULL مستخدمة بشكل أكبر وهي أكثر وضوحاً لأنّنا نادراً ما نقارن المؤشّرات مع قيم ثابتة أو نسندها إلى أرقام.

الفصل الثالث عشر:

التراكيب (Structures)

تراكيب المعطيات

تركيب المعطيات هو نوع يقوم المستخدم بتعريفه ويتألف من أنواع مختلفة من البيانات ذات الحجوم المختلفة مجموعةً في تصريح واحد بالشكل :

```
struct  name {
  data_type   element1;
  data_type   element2;
.
.
 data_type   elementN;
} object1_name, object2_name, … , objectN_name ;
```

name: اسم التركيب.

object_name: اسم الكائن الذي نريد أن نشتقه من التركيب (تعريفه هنا اختياري).

أنواع البيانات المختلفة التي يحويها التركيب موجودة بين القوسين { }.

مثال:

```
struct products {
  char name [30];
  float price;
} apple, orange, melon;
```

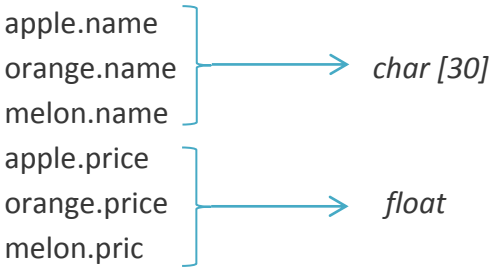
- عرّفنا أولاً التركيب products الذي يتألف من حقلين هما name و price كلّ منهما له نوع مختلف عن نوع الآخر.
- بعد أن عرّفنا التركيب قمنا بتعريف ثلاثة كائنات من نفس النوع products وهي: apple و orange و melon.
- يمكن أن نشتق الكائنات من التركيب بطريقة أخرى وذلك بأن نعرف الكائن المشتق باستخدام اسم التركيب. مثلاً:

```
struct products {
  char name [30];
  float price;
} ;
products apple;
products orange, melon;
```

عرّفنا أولاً التركيب products الذي يتألف من حقلين هما name و price كل منهما له نوع مختلف عن نوع الآخر.
بعد أن عرّفنا التركيب استخدمنا اسم التركيب (products) للتصريح عن ثلاثة كائنات من النوع نفسه هي: apple و orange وmelon.

- بعد أن صرّحنا عن التركيب (products) فإنّ products يصبح اسم نوع جديد مثل الأنواع الأساسية المعرفة مسبقاً كالنوع ***int*** أو ***char*** أو ***short*** ويصبح بإمكاننا أن نصرح عن كائنات (متغيّرات) من هذا النوع.
- بعد أن صرحنا عن الكائنات apple و orange و melon يصبح لكل منها حقلان هما name و price يمكن أن نتعامل مع الحقول بذكر اسم الكائن تليه نقطة (.) ثمّ اسم الحقل.

أمثلة



مثال

```
// example about structures
#include <iostream>
using namespace std;
#include <string>

struct students{
       char name[50];
       char city[50];
       int tel;
}st1,st2;

int main()
{
       //filling information
       cout<<"enter student1 name, city and tel_no: ";
       cin>>st1.name;
    cin>>st1.city;
       cin>>st1.tel;
       cout<<"enter student2 name, city and tel_no: ";
       cin>>st2.name;
    cin>>st2.city;
       cin>>st2.tel;
    //printing information
       cout<<"\nname"<<"\tcity\t"<<"tel_no";
       cout<<"\n------------------------\n";
       cout<<st1.name<<"\t"<<st1.city<<"\t"<<st1.tel<<endl;
       cout<<st2.name<<"\t"<<st2.city<<"\t"<<st2.tel<<endl;
       return 0;
}
```

```
enter student1 name, city and tel_no: Ali
Hama
3245675
enter student2 name, city and tel_no: Ahmad
Homs
2452658

name    city    tel_no
---------------------------
Ali     Hama    3245675
Ahmad   Homs    2452658
```

مصفوفة التراكيب

التراكيب ميزة كثيراً ما تستخدم لبناء قواعد المعطيات خاصةً إذا استخدمناها مع المصفوفات حيث يمكن بناء مصفوفة تراكيب بالإعلان عن التركيب أولاً ثمّ نعلن عن مصفوفة من نوع التركيب كما يلي :

```
struct name {
 type  element1;
 type  element2;
.
.
 type  elementN;
} object_name [ELEMENTS];
```

مثال:

```
// array of structures
#include <iostream>
using namespace std;
#include <stdlib.h>

#define N_BOOKS 5

struct book {
  char title [50];
  int year;
} books [N_BOOKS];

void printBook (book mybook);

int main ()
{
  char buffer [50];
  int n;
  cout<<"please enter the titles and
years for 5 books!.\n";
  for (n=0; n< N_BOOKS; n++)
  {
    cout << "title: ";
    cin.getline (books[n].title,50);
    cout << "year: ";
    cin.getline (buffer,50);
    books[n].year = atoi(buffer);
  }
```

```
Enter title: java how to program
Enter year: 1997
Enter title: eat, pray, love
Enter year: 2006
Enter title: advanced memory
Enter year: 2005
Enter title: being logical
Enter year: 2004
Enter title: c++ how to program
Enter year: 2005

You have entered these movies:
java how to program (1997)
eat, pray, love (2006)
advanced memory (2005)
being logical (2004)
c++ how to program (2005)
```

```
  cout << "\nYou have entered these
books:\n";
  for (n=0; n< N_BOOKS; n++)
    printBook (books[n]);
  return 0;
}

void printBook (book mybook)
{
  cout << mybook.title;
  cout << " (" << mybook.year<< ")\n";
}
```

التراكيب و المؤشّرات

يمكن استخدام للمؤشّرات لتشير إلى التراكيب كما يلي :

```
struct book {
 char title [50];
 int year;
};
book abook;
book * pbook;
```

abook: هو كائن من النوع book

pbook: هو مؤشّر يشير إلى كائن من النوع book

ولجعل المؤشّر pbook يشير إلى الكائن abook نكتب

```
pbook = &abook;
```

مثال

```
// pointers to structures
#include <iostream>
using namespace std;
#include <stdlib.h>
struct book {
  char title [50];
  int year;
} ;
int main ()
{
  char buffer[50];
  book abook;
  book * pbook;
  pbook = &abook;
  cout<<"please enter a book title and
its publish year!.\n";
  cout << "title: ";
  cin.getline (pbook->title,50);
  cout << "Enter year: ";
  cin.getline (buffer,50);
  pbook->year = atoi (buffer);

  cout << "\nYou have entered:\n";
  cout << pbook->title;
  cout << " (" << pbook->year << ")\n";

  return 0;
}
```

```
Enter title: c++ how to program
Enter year: 2005

You have entered:
c++ how to program (2005)
```

- المعامل (<-) هو معامل مرجعي يستخدم خصيصاً مع المؤشّرات على التراكيب أو المؤشّرات على الأصناف للوصول إلى الحقول الموجودة فيها وهو يلغي الحاجة لاستخدام الأقواس في كل مرة نريد أن نصل إلى حقل ما في تركيب ما.

pbook->title ⟺ (*pbook).title

كلا العبارتين صحيحتان وتعنيان أنّنا نتعامل مع الحقل title المشار إليه بالمؤشّر pbook ويجب أن نميزهما عن العبارتين :

*pbook.title ⟺ *(pbook.title)

اللتين تعنيان أنّنا نتعامل مع القيمة المشار إليها بالعنصر title (الذي لا يعتبر مؤشّراً بالأساس) أي أنّ العبارتين الأخيرتان خاطئتان ولا تعنيان شيئاً.

الجدول التالي يلخّص الحالات الممكنة للمؤشّرات والتراكيب

العبارة	الوصف	العبارة المكافئة
pbook.title	العنصر title من التركيب pbook	
pbook->title	العنصر title من التركيب المشار إليه بالمؤشّر pbook	(*pbook).title
***pbook.title**	القيمة المشار إليها بالعنصر title من التركيب pbook	*(pbook.title)

التراكيب المتداخلة

يمكن أن تكون التراكيب حقولاً موجودةً داخل تراكيبَ أخرى كما في المثال التالي:

```
struct books{
  char title [50];
  int year;
}

struct friends{
  char name [50];
  char email [50];
  books favourite_book;
  } ahmad,ali;

friends *p = &ahmad;
```

بعد هذه التصريحات يمكن أن نستخدم العبارات التالية :

```
ahmad.name
ali.favourite_book.title
ahmad.favourite_book.year
p->favourite_book.year
```

تمرين

اكتب برنامج يقوم باستعراض لائحة تتضمن :

1. إدخال اسم جديد لطالب .
2. استعراض كل أسماء الطلاب المدخلة .
3. البحث عن طريق إدخال اسم الطالب .
4. البحث عن طريق إدخال السنة الدراسية للطالب .
5. ترتيب أسماء الطلاب أبجدياً .
6. الخروج من البرنامج .

الحل:

```
#include<iostream>
#include<string>
using namespace std;
#define M 100
int count=0;
struct studentsType
{
	string name;
	int age,year;
}student[M];
void add();
void browse();
void searchByName();
void searchByYear();
void order();
int main()
{
	int choice;
	do
	{
		cout<<"please enter your choice!.\n";
		cout<<"1-add new student\n2-browse\n3-search by name\n4-search
by year"
		   <<"\n5-alphabetical order\n6-exit\n";
		cin>>choice;
		switch(choice)
		{
		case 1: add();break;
		case 2: browse();break;
		case 3: searchByName();break;
		case 4: searchByYear();break;
		case 5:order();break;
		case 6: return 0;
		}
	}while(count<M);
	return 0;
}
```

```
void add()
{

        cout<<"student name: ";cin>>student[count].name;
   cout<<"student age: ";cin>>student[count].age;
        cout<<"student Year: ";cin>>student[count].year;
        count++;
        cout<<"\n";
}
void browse()
{
   cout<<"name\tage\tyaer\n";
        for(int i=0;i<count;i++)
          cout<<student[i].name<<"\t"<<student[i].age<<"\t"<<student[i].year<<"\n";
        cout<<"\n";
}

void searchByName()
{

        string name;
        bool found=false;
        int i;
        cout<<"enter name:";cin>>name;
        for(i=0;i<count;i++)
        {
                if(name==student[i].name)
                {

cout<<student[i].name<<"\t"<<student[i].age<<"\t"<<student[i].year<<"\n";
                 found=true;
                }

        }
        if(!found)
                cout<<name<<" was not found\n";
        cout<<"\n";


}
void searchByYear()
{
        int y;
        bool found=false;
        int i;
        cout<<"enter year:";cin>>y;
        for(i=0;i<count;i++)
        {
                if(student[i].year==y)
                {
```

```
cout<<student[i].name<<"\t"<<student[i].age<<"\t"<<student[i].year<<"\n";
                    found=true;
                    }

          }
          cout<<"\n";
          if(!found)
                    cout<<"no result\n";

}
void order()
{
          int i,j;
          studentsType temp;
          if(count<2)goto out;

          for(i=0;i<count;i++)
          {
                    for(j=0;j<count-1;j++)
                    {
                              if(student[j].name>student[j+1].name)
                              {
                                        temp=student[j];
                                        student[j]=student[j+1];
                                        student[j+1]=temp;
                              }
                    }
          }
 cout<<"\n";
out:browse();
}
```

الفصل الرابع عشر:

الأنواع المعرّفة من قِبل المستخدم

تعرّفنا سابقاً على التراكيب التي تعتبر نوعَ بياناتٍ يعرّفه المستخدم (المبرمج)، بالإضافة إلى التراكيب توجد أنوع أخرى مثل:

تعريف أنواع خاصّة (typedef)

يمكن أن نعرف أنواع خاصّة بنا بالاعتماد على أنواع معرّفة وموجودة مسبقاً باستخدام الكلمة المفتاحية typedef التي تُستخدم كما يلي :

```
Typedef  existing_type  new_type_name;
```

existing_type: هو النوع الأصلي المعرف في C++ أو أي نوع معرّف مسبقاً.
new_type_name: هو اسم النوع الجديد الذي سنعرفه. مثلاً :

```
typedef char C;
typedef unsigned int WORD;
typedef char * string_t;
typedef char field [50];
```

هنا عرّفنا أربعة أنواع جديدة هي C و WORD و string_t و field على أنها char و unsigned int و char* و char[50] بالترتيب، وبالتالي يمكن استخدام هذه الأنواع لتعريف متغيّرات كما يلي :

```
C achar, anotherchar, *ptchar1;
WORD myword;
string_t ptchar2;
field name;
```

typedef مفيدة لتعريف نوع سيُستخدم بكثرة في البرنامج ومن الممكن أن تحتاج لتغييرهذا النوع فيما بعد، كما تستخدم أحياناً عندما يكون اسم النوع طويلاً وتريده أن يكون مختصراً.

الوحدات (Unions)

الـ *union* هو تركيب جميع عناصره يتمّ حجزها في نفس العنوان الذاكري.
طريقة الإعلان عنها واستخدامها مشابهة للطريقة المتبعة في التراكيب لكنّ عملها مختلف تماماً:

```
union  model_name {
  type1 element1;
  type2 element2;
  type3 element3;
  .
  .
} object_name;
```

الحجم الذي يشغله الـ union هو حجم أكبر عنصر فيه.

مثلاً:

```
union mytypes_t {
  char c;
  int i;
  float f;
  } mytypes;
```

تنتج العناصر :

```
mytypes.c
mytypes.i
mytypes.f
```

كل واحد من هذه العناصر له نوع مختلف ولكنها في نفس الموقع من الذاكرة وأي تعديل على قيمة أحد العناصر سيؤثر على قيم كل العناصر.

المرقّمات (*enum*)Enumerations

يسمح هذا النوع بتعريف أنواع بيانات يمكن أن تأخذ قيم مختلفة محدّدة من قبل المستخدم والشكل العام لها هو :

```
enum model_name {
 value1,
 value2,
 value3,
.
.
} object_name;
```

مثال:

```
enum   colors {black, blue, green, cyan, red, purple, yellow, white};
```

لاحظ أنّنا لم نضمّن أي نوع بيانات أساسي أي أنّنا أنشأنا نوع بيانات جديد دون الإعتماد على نوع موجود مسبقاً وهو النوع colors الذي يأخذ أحد القيم الموجودة بين القوسين {} وعندما نعرّف الكائن mycolor من هذا النوع فإنّ العبارات التالية صحيحة:

```
colors  mycolor;
mycolor = blue;
if (mycolor == green) mycolor = red;
```

في الواقع إنّ البيانات الموجودة داخل القوسين {} تستبدل بقيم عددية صحيحة أثناء الترجمة والقيم الافتراضية هي 0 لأول عنصر و1 للثاني و 2 للثالث وهكذا...
أمّا إذا خصّصنا قيمة لأول عنصر فإنّ قيمة العنصر الثاني تزيد عن قيمة الأول بواحد وهكذا..
مثلاً إذا كتبنا black=1 بدلاً من black وتركنا العناصر الباقية كما هي فإنّ white سيأخذ القيمة 8.

الفصل الخامس عشر:

الإدخال والإخراج في الملفات

الملفات تشكّل أساس التعامل مع الحاسب فعندما تعمل على أي حاسب لابدّ لك من أن تفتح ملفّاً ما أو تكتب على ملف سواءً كان هذا الملف نصّاً أو صورةً أو صوتاً.. .إلخ.
في معظم البرامج التي كتبناها سابقاً كانت المعلومات تخزّن في الذاكرة العشوائية ويتمّ فقدانها عند إنهاء البرنامج فإذا كنّا بحاجة لهذه المعلومات فيمكننا تخزينها في ملفات على القرص الصلب لاستعادتها لاحقاً.

تدعم لغة C++ الإدخال والإخراج في الملفات من خلال الصفوف التالية :

- **ofstream:** صف الملفات الخاص بعمليات الكتابة (مشتق من الصف ostream)
- **ifstream:** صف الملفات الخاص بعمليات القراءة (مشتق من الصف istream)
- **fstream:** صف الملفات الخاص بعمليات القراءة والكتابة (مشتق من الصف iostream)

فتح ملف

عموماً العملية الأولى التي تنفّذ على كائن من الصفوف المذكورة هي ربط هذا الكائن بملف حقيقي أي عملية فتح ملف، هذه العملية تُمثّل في البرنامج من خلال الكائن (stream) وأي دخل أو خرج ينفذ على هذا الكائن سيطبق على الملف الحقيقي.

لكي نفتح ملفاً عن طريق الكائن stream نستخدم تابعه العضوي open() المعرّف بالشكل :

```
void open (const char * filename, openmode  mode);
```

***filename*:** سلسلة رموز تمثل اسم الملف المراد فتحه.
***mode*:** وضعية الفتح وهي مزيج من الأوضاع المبينة بالجدول :

ios::in	فتح الملف للقراءة
ios::out	فتح الملف للكتابة
ios::ate	الوضع الابتدائي : نهاية الملف
ios::app	كل خرج يضاف إلى نهاية الملف
ios::trunc	إذا كان الملف موجود سابقاً فيتمّ حذفه
ios::binary	الوضع الثنائي

يمكن الجمع بين هذه الأوضاع بالمعامل "أو" الثنائي |. مثلاً إذا أردنا فتح الملف "example.bin" في الوضع الثنائي لإضافة بيانات يمكن أن نفعل ذلك باستدعاء التابع العضوي open كالتالي:

```
ofstream file;
file.open ("example.bin", ios::out | ios::app | ios::binary);
```

لكل تابع open من التوابع الخاصة بالصفوف ofstream و ifstream و fstream وضع افتراضي خاص يختلف من صف إلى آخر كما في الجدول :

الصف	الوضع الافتراضي للفتح
ofstream	ios::out | ios::trunc
ifstream	ios::in
fstream	ios::in | ios::out

تطبق القيمة الافتراضية فقط في حال تمّ استدعاء التابع بدون تخصيص البارامتر *mode*.

- بما أنّ العملية الأولى التي تنفذ على الكائن من الصفوف السابقة هي عملية فتح ملف فإنّ كلاً من هذه الصفوف الثلاثة يحتوي على مشيد (باني) يقوم باستدعاء التابع open بنفس البارامترات، لذلك يمكننا أن نعرّف الكائن ونستدعي التابع بسطر واحد كما يلي:

```
ofstream   file ("example.bin", ios::out | ios::app | ios::binary);
```

يمكنك التحقق من أنّ الملف قد فُتح بشكل صحيح باستدعاء التابع العضوي is_open()

```
bool   is_open();
```

هذا التابع يُعيد true إذا تمّ ربط الكائن بالملف بشكل صحيح وإلّا فإنّه يُعيد false.

إغلاق ملف

عندما تنتهي عمليات القراءة أو الكتابة أو في حال اكتمال الملف فيجب إغلاقه ليصبح متاحاً للاستخدام ثانيةً.
لفعل ذلك نستدعي التابع العضوي close() المعرّف كما يلي:

```
void  close ();
```

بعد استدعاء هذا التابع يصبح الكائن stream جاهزاً لفتح ملف آخر

- في حالة هدم الكائن وهو مايزال مرتبط بالملف فإنّ الهادم يستدعي التابع close تلقائياً.

الملفات النصّية

الصفوف ofstream و ifstream و fstream مشتقة من الصفوف ostream و istream و iostream لذلك فإنّ كائنات الـ *fstream* يمكنها أن تستخدم عناصر الصفوف الأصلية لتصل إلى المعطيات، أي يمكننا أن نستخدم cin و cout مع الكائنات التي نعرّفها كما في المثال التالي حيث نستخدم معامل الإدخال >> الذي تمّ إعادة تحميله

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;
int main () {
  ofstream examplefile ("example.txt");
  if (examplefile.is_open())
  {
    examplefile << "This is a line.\n";
    cout<< "This is a line.\n";
    examplefile << "This is another line.\n";
    cout<< "This is another line.\n";
    examplefile.close();
  }
  else cout<<"Unable to open the file";
  return 0;
}
```

```
file example.txt

This is a line.
This is another line.
```

كما أنّ إدخال البيانات من ملف ينفذ بنفس الطريقة التي نستخدم بها cin:

```
// reading a text file
#include <iostream>
#include <fstream>
#include <stdlib.h>
using namespace std;

int main () {
  char buffer[256];
  ifstream examplefile ("example.txt");
  if (! examplefile.is_open())
  { cout << "Error opening file"; exit (1); }

  while (! examplefile.eof() )
  {
    examplefile.getline (buffer,100);
    cout << buffer << endl;
  }
  return 0;
}
```

```
This is a line.
This is another line.
```

- هذا البرنامج يقوم بقراءة ملف نصّي ويطبع محتواه على الشاشة.
- التابع العضوي eof() يعيد true عند الوصول إلى نهاية الملف.

التحقق من الحالة

بالإضافة إلى التابع eof() توجد توابع عضوية أخرى للتحقق من حالة الكائن (الملف) تعيد كلها قيم منطقية (bool)

bad():
يعيد true إذا حدث خطأ في عملية القراءة أو الكتابة. مثلاً إذا حاولنا الكتابة على ملف غير مفتوح لأجل الكتابة.

fail():
يعيد true كما في حالة التابع bad() كما يعيد true في حال حدوث خطأ في صيغة الملف كما في حالة قراءة عدد صحيح وتمّ إدخال حرف.

good():
يعيد false في نفس الحالات التي تعيد فيها التوابع السابقة true.

لتصفير علامات الحالة التي تمّ فحصها بالتوابع السابقة نستخدم التابع clear() بدون وسطاء.

مؤشّرا التدفق (*get* و *put*)

جميع كائنات التدفق الخاصة بالدخل والخرج تحوي مؤشّراً واحداً على الأقل:

- **ifstream** مثل istream يحوي المؤشّر get الذي يشير إلى العنصر التالي المراد قراءته.
- **ofstream** مثل ostream يحوي المؤشّر put الذي يشير إلى المكان الذي سيكتب فيه العنصر التالي.
- وأخيراً **fstream** مثل iostream يرث كلّاً من get و put.

يمكن التعامل مع هذه المؤشّرات من خلال التوابع التالية:

tellg() و tellp()

هذان التابعان لا يقبلان وسطاء ويعيدان قيمة من النوع pos_type وهي قيمة من النوع الصحيح (integer) تمثل المكان الحالي للمؤشّر get (في حالة التابع tellg()) أو مكان المؤشّر put (في حالة التابع tellp()).

seekg() و seekp()

يقوم هذان التابعان بتغيير مكاني المؤشّرين get و put وتمّ إعادة تحميلهما كما يلي :

seekg (**pos_type** *position*);
seekp (**pos_type** *position*);

باستخدام هذا الشكل يتمّ تحويل المؤشّر إلى مكان يبعد بعداً ثابتاً عن بداية الملف والوسيط المطلوب من نفس النوع الذي يعيدانه التابعان tellg() و tellp().

seekg (**off_type** *offset*, **seekdir** *direction*);
seekp (**off_type** *offset*, **seekdir** *direction*);

باستخدام هذا الشكل يتمّ تحويل المؤشّر إلى مكان يبعد بعداً ثابتاً عن نقطة معينة بحيث:
off_type: نوع مكافئ للنوع **long**.
direction: الوجهة التي سيتم نقل المؤشّر إليها ويجب أن تأخذ إحدى الحالات التالية:

`ios::beg`	**إلى بداية الملف**
`ios::cur`	من المكان الحالي للمؤشّر
`ios::end`	إلى نهاية الملف

المثال التالي يستخدم التوابع المذكورة لحساب حجم ملف ثنائي:

```
// obtaining file size
#include <iostream>
#include <fstream>
using namespace std;
const char * filename = "example.txt";

int main () {
  long l,m;
  ifstream file (filename,
ios::in|ios::binary);
  l = file.tellg();
  file.seekg (0, ios::end);
  m = file.tellg();
  file.close();
  cout << "size of " << filename;
  cout << " is " << (m-l) << " bytes.\n";
  return 0;
}
```

```
size of example.txt is 40 bytes.
```

الفصل السادس عشر:

معالجة الاستثناءات (Exceptions handling)

ملاحظة: معالجة الاستثناءات من الميزات الحديثة التي تمّ إنتاجها مع لغة ANSI-C++ القياسية.
لذلك إذا كنت تستعمل مترجماً لا يدعم هذه اللغة القياسيّة فمن الممكن ألّا تكون قادراً على استخدامها.

أثناء تطوير البرنامج قد نتعرض لمواقف لا نستطيع فيها أن نحدّد فيما إذا كانت هذه الشيفرة ستعمل بالشكل الصحيح أو لا وذلك لأنّه قد تكون هناك بعض الحالات التي لا نتوقّع حدوثها والتي قد تتسبب بخطأ في البرنامج. مثل هذه المواقف هو ما ندعوه بالاستثناءات.

وأخيراً قدمت C++ ثلاث معاملات جديدة (try, throw, catch) لتساعدنا في معالجة هذه الحالات وهي تأخذ الشكل التالي:

```
try {
  الشيفرة التي ستعمل أوّلاً //
  throw exception; //يرسل متغيّراً يستخدم لمعرفة الاستثناء الحاصل
}
catch (type  exception)
{
  الشيفرة التي سيتم تنفيذها حين يحدث الاستثناء //
}
```

وهي تعمل بالطريقة التالية:

- الشيفرة داخل التعليمة try تُنفَّذ بشكل طبيعي، وبمجرّد حصول استثناء ما يتمّ استدعاء التعليمة throw التي تعيد وسيطاً قيمته وصفٌ للاستثناء الحاصل و يأخذ (الوسيط) أي نوع مناسب للاستثناء.
- إذا حصل الاستثناء فإنّ التعليمة catch ستعمل على استقبال الوسيط الممرّر من التعليمة throw وإلّا فإنّ التعليمة catch لن تنفّذ.

مثال :

```
// exceptions
#include <iostream>
using namespace std;
int main () {
  char myarray[10];
  try
  {
    for (int n=0; n<=10; n++)
    { if (n>9) throw "Out of range";
      myarray[9]='z';
    }
  }
  catch (char * str)
  { cout << "Exception: " << str << endl; }
  return 0;
}
```

```
Exception: Out of range
```

يمكننا أن نعيد تحميل المعامل catch ليقبل أكثر من نوع من الوسطاء. عندها فإنّ التعليمة التي ستنفّذ هي تلك التي تقابل الحالة الحاصلة ويتعرّف المترجم عليها من خلال الوسيط المرسل من التعيمة throw.

مثال :

```
// exceptions: multiple catch blocks
#include <iostream>
using namespace std;
int main () {
  try
  {
    char * mystring;
    mystring = new char [10];
    if (mystring == NULL)
      throw "Allocation failure";
    for (int n=0; n<=100; n++)
    {
      if (n>9) throw n;
      mystring[n]='z';
    }
  }
  catch (int i)
  {
    cout << "Exception: ";
    cout << "index " << i
         << " is out of range"
         << endl;
  }
  catch (char * str)
  {
    cout << "Exception: " << str;
  }
  return 0;
}
```

```
Exception: index 10 is out of range
```

في هذا المثال يوجد احتمالان لحدوث استثنائين مختلفين :

- **الأوّل** هو ألّا يتمّ إسناد أيّ حرف من الحروف العشر المشكلة للمصفوفة mystring عندها سيتمّ استدعاء التعليمة {…}catch(char * str).
- **الثاني** عندما يأخذ العدّاد i القيمة أكبر من 9 عندها تُستدعى التعليمة {…} catch (int i).

القسم الثالث

إذا غامَرْتَ في شَرَفٍ مَرُومِ ... فَلا تَقنَعْ بما دونَ النّجومِ
فطَعْمُ المَوْتِ في أمْرٍ حَقِيرٍ ... كطَعْمِ المَوْتِ في أمْرٍ عَظيمِ
يرَى الجُبَناءُ أنّ العَجزَ عَقْلٌ ... وتِـــلكَ خَديعَةُ الطّبعِ اللّئيمِ
وكلّ شَجاعَةٍ في المَرْءِ تُغني ... ولا مِثلَ الشّجاعَةِ في الحَكيمِ
وكَمْ من عائِبٍ قَوْلاً صَحيحاً ... وآفَتُهُ مِنَ الفَهْمِ السّقيمِ
ولكِنْ تأخُذُ الآذانُ مِنْهُ ... على قَدَرِ القَرائِحِ والعُزُومِ

للمتنبّي

الفصل السابع عشر:

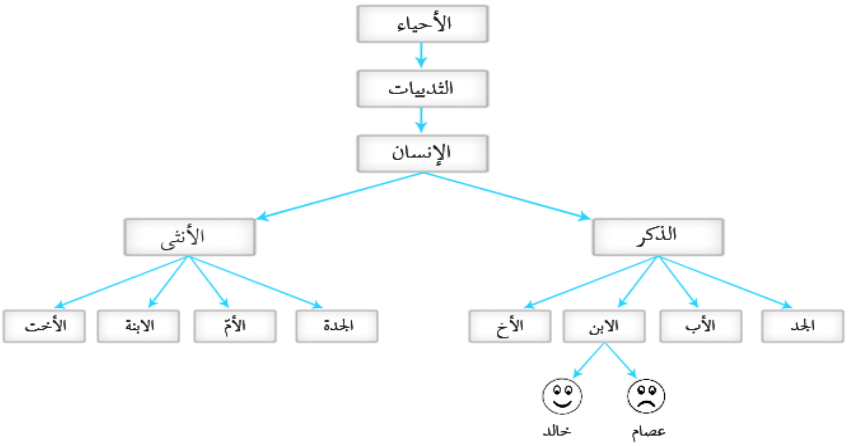
البرمجة كائنيّة التوجّه

(Object-Oriented programming)

مقدّمة:

حتّى هذه اللحظة كلّ البرامج التي كتبناها كانت من نمط البرمجة الهيكليّة أي أنّ البرنامج له مسار محدّد لا يمكن أن يتغيّر، ويواجه المبرمجون مشاكل عديدة في هذا النمط من البرمجة حيث تصبح عمليّة تتبُّع الأخطاء و التعديل على البرنامج صعبةً جدّاً حتّى مع استخدام التوابع ففي البرامج الكبيرة والتي يعمل على إنجازها عشرات المبرمجين سيكون من الصعب تقسيم العمل بحيث نضمن فصلاً تامّا لما سيقوم به كلّ مبرمج لذلك تمّ تصميم هذا المبدأ الجديد ليحلّ محلّ سابقه وقد اعتَمَدَ هذا المبدأ على التفكير بالبرامج على أنّها مجموعة من الكائنات تربط بينها علاقات منطقيّة معيّنة تماماً كما يحدث في حياتنا اليوميّة.

ما هي الكائنات؟ الكائنات هي الأشياء!. أجل انظر من حولك فكلّ الأشياء التي تراها هي كائنات، وكلّ الكائنات التي تصادفها تمتلك صفات خاصة بها تميزها عن غيرها وقد تمتلك بعض الصفات المشتركة مع كائنات أخرى. لنأخذ المثال التالي:



إنّ كلّ مربع من هذه المربعات هو عبارة عن صفّ مستقلٍّ بحدّ ذاته عن الآخرين في الصفات الخاصّة به (كالاسم والطول ولون البشرة...) وهو يشبه البقيّة في صفاته العامّة الأخرى (كل الأحياء تتنفّس وتأكل وتنتقل وتتكاثر...). وقياساً على ذلك يمكننا أن نحوّل كل الأشياء التي من حولنا إلى صفوف ونقومَ ببرمجتها لتؤدّي الوظائف المطلوبة منها ولتسهيل هذه العمليّة سنقوم بكتابة الشيفرة (التي تحتوي على بيانات وصفات هذا الصفّ) ثمّ نستطيع أن ننشأ كائنات عديدة من هذا الصفّ الذي يمثّل مرجعاً لكلّ الكائنات من نوعه ومثال ذلك كلٌّ من الجدّ والأب والابن في مثالنا السابق هي صفوف مستقلّة لكنّ عصام وخالد هما كائنين من صفّ الابن وكل كائن منهما يختلف عن الكائن الآخر في بعض الصفات.
الخلاصة إنّ الصفّ يمثّل المرجع العام للنوع. أمّا الكائن فهو نسخة من هذا الصفّ يمتلك بياناته الخاصّة والمختلفة عن باقي الكائنات المنشأة من هذا النوع.

ملاحظة: البنية الأساسيّة في البرامج الهيكليّة هي التوابع.
أمّا في البرمجة كائنيّة التوجّه فالبنية الأساسيّة هي الصفوف.

الصفّ: هو طريقة منطقيّة لترتيب البيانات والتوابع في بنية واحدة.

يصرّح عن الصفّ باستخدام الكلمة المحجوزة class والتي تقابل من الناحية الوظيفيّة الكلمة المحجوزة struct في لغة C مع قدرتها على احتواء التوابع كأعضاء، بدلاً من احتوائها على بيانات فقط كما في struct ويمكننا القول: "إنّ struct هو نوع خاص من الصفوف".

الصفوف هي أنواع جديدة من تعريف المستخدم.

يمكننا أن نعرّف الصفّ بالشكل :

```
class class_name {
   member0;
 permission_label_1:
   member1;
 permission_label_2:
   member2;
. . .
} object_name;
```

class: هي كلمة محجوزة تُستخدم للتصريح عن صفّ جديد.

class_name: هو اسم الصفّ (النوع من تعريف المستخدم).

object_name: اسم الكائن المُشتق من الصفّ وهو اختياري، و يمكن أن يكون هناك أكثر من كائن مشتقٍّ عندها نفصل بين أسمائها بفاصلة عاديّة تماماً كما فعلنا في struct.

جسم الصفّ يمكن أن يحتوي على أعضاء والأعضاء إمّا أن تكون تصريحات عن بيانات (متغيّرات) أو عن توابع، وهناك مستوى الوصول (السماح) وهو اختياريّ ويكون إحدى الكلمات المحجوزة الثلاث التالية: private, public, protectd والتي توضع قبل الأعضاء لإكسابها مستوى وصولٍ معيّنٍ وهي تختلف عن بعضها كما سنبيّن في الفقرة التالية.

معرّفات الوصول Access modifieres

تقدّم C++ ما يُدعى بمعرّفات الوصول أو Access modifieres وهي ثلاثُ معرّفاتٍ تحدّد المدى الذي سيظهر فيه العضو وهي :

- **private** : الأعضاء من هذا النوع في صفّ ما يمكن الوصول إليها فقط من أعضاء الصفّ نفسه أو من أي صفّ من النوع friend الذي يمكن لأعضائه أن يصِلوا إليها (الأعضاء private).
- **protected** : الأعضاء من هذا النوع في صفّ ما يمكن الوصول إليها من الصفّ نفسه أو من أيّ صفّ من النوع friend وكذلك من أعضاء الصفوف التي يتمّ اشتقاقها من هذا الصفّ.
- **public** : الأعضاء في هذا المستوى يمكن الوصول إليها من أيّ مكان يظهر فيه الصفّ الذي يحويها.

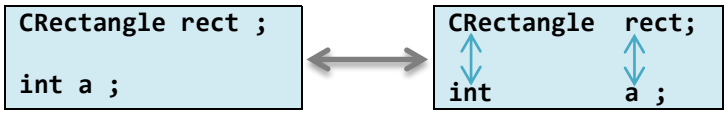
ملاحظة: لو صرّحنا عن أعضاء في صفّ ما قبل أن نضع أيّ معرّف وصول فإنّ هذه الأعضاء ستُعتبر من المستوى private لأنّه يعتبر مستوى الوصول الافتراضي للأعضاء التي يُصرّح عنها في أيّ صفّ ما لم نحدّد لها مستوىً آخرَ.

مثال:

```
class  CRectangle {
   int x, y;
 public:
   void set_values (int,int);
   int area (void);
 } rect;
```

في هذا المثال قمنا بالتصريح عن صفّ (نوع) أسميناه CRectangle ثمّ أنشأنا منه كائناً دعوناه rect. هذا الصفّ يحتوي أربعة أعضاء، متغيّرين (x, y) من النوع int وهما من المستوى private لأنّه المستوى الافتراضي للأعضاء إن لم نحدِّد لها أيّ مستوى وصولٍ كما قلنا، و يحتوي أيضاً على تابعين set_values() و area() في المستوى public ، و قد صرّحنا عنهما بالشكل prototype كما تلاحظ.

لاحظ الفرق بين اسم الصفّ و اسم الكائن المشتقّ منه، في المثال السابق اسم الصفّ هو Crectangle الذي يمثل اسم النوع الجديد الذي عرفّناه، بينما اسم الكائن هو rect وهو من نوع الصفّ (لأنّه اشتُقَّ منه). يمكن توضيح هذه المسألة من خلال المثال التالي:



int هو اسم الصفّ (النوع)، بينما a هو اسم الكائن المُشتقّ من الصفّ (النوع) int.
أثناء كتابة التعليمات داخل البرنامج يمكننا أن نشير إلى أيّ عضو عام (من المستوى public) للكائن rect كما لو كان تابعاً أو متغيّراً عاديّاً، وذلك فقط بوضع اسم الكائن متبوعاً بنقطة ثمّ اسمِ العضوِ (كما فعلنا مع struct في C). مثال:

```
rect.set_value(3,4) ;
myarea = rect.area();
```

لكن لا يمكننا أن نفعل ذلك مع العضوين x, y لأنّهما من المستوى الخاص (private) أي أنّه لا يمكن الوصول إليهما إلّا من داخل الصفّ CRectangle الذي عُرِّفا داخله. إليك الآن المثال كاملاً:

```
// classes example
#include <iostream>
using namespace std;
class CRectangle {
    int x, y;
  public: // permission_label_1:
    void set_values (int,int);
    int area (void) {return (x*y);}
};
void CRectangle::set_values (int a, int b) {
  x = a;
  y = b;
}
int main () {
  CRectangle rect; // like int a ;
  rect.set_values (3,4);
  cout << "area: " << rect.area() << endl;
return 0;
}
```

```
area: 12
```

الشيء الجديد في هذا المثال هو معامل المدى :: الذي ظهر في تعريف التابع set_values() حيث استُخدِم من أجل التصريح عن العضو set_values() خارج الصفّ الذي عُرِّف فيه.

لاحظ أنّنا قمنا بكتابة جسم (سلوك) التابع العضو area() داخل الصفّ CRectangle في حين صرّحنا عن التابع set_values() بالتعريف المبدئي للتوابع prototype كي نستطيع كتابة جسمه خارج الصفّ. في هذه الحالة عندما نرغب بكتابة جسم (سلوك) تابعٍ عضوٍ ما خارج الصفّ الذي يحويه يجب علينا أن نستخدم المعامل :: .

معامل المدى ::

يحدّد هذا المعامل إلى أيّ صفّ ينتمي هذا العضو، وهذا المعامل يمنح التصريح عن العضو خارج الصفّ نفس صفات المدى للعضو عندما يُعرّف داخل الصفّ.
في المثال السابق استطعنا الوصول إلى المتغيّرين x, y في التابع set_values() علماً أنّ جسم التابع كُتِبَ خارج الصفّ وأنّهما عضوين في الصفّ CRectangle وهما من المستوى الخاص(private) أي أنّه لا يمكن الوصول إليهما إلّا من داخل الصفّ كما أسلفنا سابقاً.
مثال يوضح كيف يمكننا باستخدام هذا المعامل الوصول إلى المتغيّرات الشاملة في برنامجنا.

```
// example on Scope Operator ::
#include <iostream>
using namespace std;
int x ,y ; //Auto Initializing to 0 (global var's)
int main()
{
   int x = 150;
   cout<< x << "\t"<< y << endl;
   cout<< ::x << "\t" << y << endl;
   return 0;
}
```

```
150        0
0          0
```

الفرق الوحيد بين أن تقومَ بكتابة التابع العضو داخل الصفّ بشكل كامل وألّا تفعلَ ذلك هو أنّ المترجم في الحالة الأولى سيعتبر التابع من النوع inline بشكل تلقائي. أمّا في الحالة الثانية (التعريف المبدئي) فإنّ المترجم سيعتبر التابع من النوع الطبيعي not-inline.

التوابع الأعضاء inline

كل التوابع الأعضاء المعرفة بشكل كامل داخل الصف هي من النوع inline. لنفترض أن لدينا المثال التالي:

```
class Account
{
private:
   double balance;
public:
   Account(double initial_balance) { balance = initial_balance; }
   double getBalance();
   double deposit( double amount );
   double withdraw( double amount );
};
```

التابع Account والذي يدعى مشيّداً (انظر الفقرة القادمة) في هذا المثال هو تابع من النوع inline أمّا التوابع الأخرى مثل GetBalance(), Deposite(), Withdraw() فهي من النوع الطبيعي not-inline إلّا أنّه يمكننا أن نحولها إلى توابع من النوع inline بالشكل التالي :

```
inline double Account::getBalance()
{
  return balance;
}

inline double Account::deposit( double amount )
{
  return ( balance += amount );
}

inline double Account::withdraw( double Amount )
{
  return ( balance -= amount );
}
```

ملاحظة: كل التوابع المعرفة بشكل كامل داخل الصف هي من النوع inline سواء أستخدمنا الكلمة inline في التصريح عنها أم لم نستخدمها.

إنّ التصريح عن التوابع على أنّها من النوع inline يجعلها تعمل بأسلوب مختلف فبدلاً من استدعائها مثل التوابع العادية يتمّ نسخ جسم التابع مرّة لكل استدعاء وينفّذ كما لو كان جزءاً من الشيفرة التي استُدعي فيها.

ربّما تسأل نفسك **لماذا جعلنا العضوين x, y من المستوى الخاص private؟** (تذكّر أنّه إذا لم نحدد مستوى الوصول للأعضاء فهي من المستوى private). **الجواب** لأنّنا عرّفنا تابعاً set_values() ليقوم بقراءة قيمِهما في محاولة لحمايتهما من استقبال قيم خاطئة (نستطيع أن نكتب شيفرة داخل التابع للتحقق من صحة البيانات المدخلة) ولذا فليس من حاجة لجعل البرنامج قادراً على التعامل معهما مباشرة. ربّما في البرامج الصغيرة والبسيطة كالمثال السابق لن تجد منفعةً كبيرة من ذلك، ولكن في المشاريع الضخمة قد يكون من الضروري جداً أن تبقى قيم المتغيّرات بعيدة عن التحوير أو التغيير(قد ينتج التغيير عن خطأ غير متوقع أثناء كتابة الشيفرة. انظر فقرة التغليف encapsulation).

من أهمّ ميزات الصفوف أنّه يمكننا أن نصرّح عن العديد من الكائنات منها. في المثال السابق كان بإمكاننا أن نصرِّح عن الكائن (المتغيّر) rectb بالإضافة إلى الكائن rect.

```
// class example
#include <iostream>
using namespace std;
class CRectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area (void) {return (x*y);}
};
void CRectangle::set_values (int a, int b) {
  x = a;
  y = b;
}
```

```
rect area: 12
rectb area: 30
```

```
int main () {
  CRectangle rect, rectb;
  rect.set_values (3,4);
  rectb.set_values (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
}
```

لاحظ أنّ الاستدعاء rect.area() لا يعطي نفس نتيجة الاستدعاء rectb.area().
وذلك لأنّ كل كائن يُشْتَقُّ من الصفّ CRectangle يمتلك أعضاء الصفّ كلها من جديد. أي أنّ كل كائن منهما يحتوي على متغيّرين x, y خاصّين به، وتابعين area() , set_value() خاصين به أيضاً.
من هنا نشأت فكرة الكائنات والبرمجة كائنيّة التوجّه. حيث تُعتبر البيانات والتوابع خصائص (صفات) للكائن.
سنناقش في فقرات قادمة ميزات هذا الأسلوب.

في حالتنا السابقة الصفّ (نوع البيانات للكائنات المشتقّة منه) هو CRectangle حيث أنشأنا منه نسختين أو كائنين هما rect , rectb وكلّ منهما له أعضاؤه الخاصة (صفاته من متغيّرات و توابع).

تمرين

1- اكتب صفّاً جديداً اسمه Circle بحيث يحتوي على الأعضاء التالية :
- ✓ متغيّر نصف القطر r.
- ✓ تابع لقراءة نصف القطر set_radius() .
- ✓ تابع لحساب مساحة الدئرة calc_space حيث أنّ المساحة هي $S = \pi r^2$.
- ✓ تابع حساب محيط الدائرة calc_perimeter حيث أن المحيط هو $B = 2\pi r$.

ثمّ استخدمه في برنامجك حيث يدخل المستخدم نصف القطر و يطبع البرنامج المساحة و المحيط.

2- اكتب صفّاً اسمه Point حيث يحتوي على الأعضاء التالية:
- ✓ متغيّرين يمثلان احداثيات النقطة x , y.
- ✓ تابع لاستقبال قيم المتغيّرين السابقين set_point.
- ✓ تابع لحساب المسافة بين نقطتين distance حيث

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}\ .$$

ثمّ استخدمه في برنامج يحسب المسافة بين نقطتين يحددهما المستخدم

المشيّدات Constructors

تحتوي الصفوف على تابع خاصّ يدعى المشيّد constructor الذي يصرّح عنه على أنّه تابعٌ عضو في الصفّ وهو يأخذ نفس اسم الصفّ. **يتمّ استدعاء المشيّد مرّة واحدة فقط وبشكل تلقائي عندما تقوم بإنشاء (اشتقاق) نسخة (كائن) جديدة من الصفّ** وهو أوّل تابع يتمّ تنفيذه في الكائن.

تحتاج الكائنات عموماً إلى تبدئة متغيّراتها أو حجز ذاكرة ديناميكية أثناء عمليّات بنائها لتصبح فعّالة بشكلٍ كاملٍ ولنتجنّب القيم غير المتوقّعة أو العشوائيّة لأعضائها خلال فترة تنفيذ الكائن والناتجة عن عدم فعلِ ذلك (التبدئة).
مثال: **ماذا سيحدث في المثال السابق لو أنّنا استدعينا التابع area() قبل أن نستدعي التابع set_values()؟**
قد تكون النتيجة غير معروفة (عشوائية) لأنّ المتغيّريّن x, y لم يُسْند إليهما أيّة قيم، وفي محاولة للتخلّص من هذه المشكلة سنكتب الصفّ CRectangle وسنضيف إليه مشيّداً constructor

```
// classes example
#include <iostream>
using namespace std;

class CRectangle {
    int width, height;
  public:
    CRectangle (int,int); //Constructor
    int area (void) {return (width*height);}
};
CRectangle::CRectangle (int a, int b) {
  width = a;
  height = b;
}
int main () {
  CRectangle rect (3,4); //try to remove (3,4) & run
  CRectangle rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
}
```

```
rect area: 12
rectb area: 30
```

كما ترى نتيجة هذا المثال هي نفس نتيجة المثال السابق. حيث قمنا باستبدال التابع set_values() -الذي لم يعد موجوداً- بالتابع المشيّد constructor.

لاحظ كيف مرّرنا وسيطين إلى المشيّد في نفس اللحظة التي أنشأنا فيها نسخة من الصفّ. كالآتي:

CRectangle rect (3,4);
CRectangle rectb (5,6);

وترى أيضاً أنّ التابع المشيّد لا يمتلك قيمة معادة كما أنّه ليس من النوع void.

***يجب** أن تضع هذا في حسبانك دائماً المشيّد لايعيد قيمة وليس من النوع void.*

تمرين 1:

أضف مشيّد اً إلى التمرين السابق بحيث يمكنه أن يقرأ نصف القطر كوسيط مباشرة دون الحاجة إلى التابع set_radius.

الهادمات Destructors

الهادم: هو تابع له وظيفة تعاكس بشكل كامل وظيفة المشيّد حيث **يتمّ استدعاؤه بشكل تلقائي مرّة واحدةً عندما يتحرّر الكائن من الذاكرة** (إمّا لأنّه أنهى كل تعليماته كأن نعرّف كائناً على أنّه متغيّر موضعي ضمن تابع ما و ينهي التابع تعليماته، أو لأنّه كائن تمّ إسناد القيم المطلوبة إليه ثمّ تمّ تحريره من الذاكرة باستعمال معامل الحذف delet).

الهادم يجب أن يأخذ نفس اسم الصفّ مسبوقاً بالرمز ~ كما ويجب ألّا يعيد الهادم قيمةً.

يستخدم الهادم عادةً لتحرير الذاكرة -التي يحجزها الكائن أثناء تنفيذه- في اللحظة التي ينهي فيها الكائن عمله.

```
// example on constructors and destructors
#include <iostream>
using namespace std;

class CRectangle {
    int *width, *height;
  public:
    CRectangle (int,int);
    ~CRectangle ();
    int area (void) {return (*width * *height);}
};
CRectangle::CRectangle (int a, int b) {
  width = new int;
  height = new int;
  *width = a;
  *height = b;
}
CRectangle::~CRectangle () {
  delete width;
  delete height;
}

int main () {
  CRectangle rect (3,4), rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

```
rect area: 12
rectb area: 30
```

إعادة تحميل المشيّدات Overloading Constructors

كأيّ تابع آخر يمكن تطبيق هذا المبدأ على المشيّد وذلك بإنشاء عدّة مشيّدات لها نفس الاسم و تختلف في عدد أو أنواع الوسطاء. (راجع فقرة إعادة التحميل فصل التوابع) ويستطيع المترجم أن يعرف أيّ هذه التوابع سيستخدمها الكائن عندما تصرّح عنه.
في الواقع، عندما ننشئُ صفّاً ولا نعرّف فيه أيّ مشيّد فإنّ المترجم يفترض بشكل تلقائي وجود مشيّدين هما المشيّد العادي (دون وسطاء أو nop) والمشيّد النسخة، مثال :

```
class  CExample {
 public:
  int a,b,c;
  void multiply (int n, int m) { a=n; b=m; c=a*b; };
 };
```

بما أنّ الصفّ السابق لا يمتلك مشيّداً فإنّ المترجم سيفترض أنّ هناك مشيّدين افتراضيين هما:

المشيّد العادي : وهو مشيّد لا يمتلك وسطاء ويُعْرَفُ بـ nop (no parameters) وهو لا يفعل أي شيء عادةً.

```
CExample::CExample () { };
```

المشيّد النسخة : يحتوي على وسيط من نوع الصفّ ممرّر بالمرجع، وهو يقوم بنسخ قيم المتغيّرات الأعضاء غير الثابتة non-static من الكائن الوسيط ويسندها إلى المتغيّرات الأعضاء المقابلة لها (non-static) في الكائن a المشتقّ من صفّ من النوع non-static أيضاً.

```
CExample::CExample (const CExample& rv) {
 a=rv.a;  b=rv.b;  c=rv.c;
 }
```

يجب أن تعلم أنّ المشيّدين الافتراضين يكونان موجودين عندما لا تقوم بالتصريح عن أيٍّ مشيّد في الصفّ و لذلك عندما تضيف مشيّداً ما إلى الصفّ فإنّ هذين المشيّدين لن يكونا موجودين وفي حال رغبت بالإبقاء على أحدهما فعليك أن تصرّح عنه بنفسك.

```
// overloading class constructors
#include <iostream>
using namespace std;
class CRectangle {
    int width, height;
  public:
    CRectangle ();
    CRectangle (int,int);
    int area (void) {return (width*height);}
};
CRectangle::CRectangle () {
  width = 5;
  height = 5;
}
CRectangle::CRectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  CRectangle rect (3,4);
  CRectangle rectb;
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
}
```

```
rect area: 12
rectb area: 25
```

في هذه الحالة الكائن rectb صُرِّح عنه بدون وسطاء، لذلك فقد تمّت تبدئة متغيّراته باستخدام المشيّد العادي nop والذي يسند القيمة 5 إلى كلّ من المتغيّرين width و height

ملاحظة: إذا أردنا أن ننشأ كائناً جديداً من صفّ ما واستخدامنا المشيّد العادي فعلينا أن نذكر اسم الصفّ و بعده اسم الكائن الجديد دون أن نضع أقواساً من الشكل (). أمّا إذا كان في صفّ ما مشيّد ات غير المشيّد العادي (nop) الذي لا يمتلك وسطاء لتمرّر إليه وأردنا استخدام أحدها عندها يجب علينا أن نذكر القوسين () بعد اسم الكائن الجديد وبداخلهما الوسطاء المطلوبة.

مثال: في المثال السابق:

CRectangle rectb; // تصريح صحيح لأنّنا استخدمنا المشيّد العادي

CRectangle rectb(); // تصريح خاطئ! لأنّ المشيّد العادي لا يحتاج إلى القوسين ()

تمرين 2:

1- في التمرين 1 (الصفّ Circle) الذي أضفت إليه مشيّد اً لقراءة نصف القطر هل العبارة التالية صحيحة :

```
Circle circle ;
```

2- في التمرين 2 (الصفّ Point) أضف مشيّد ين الأول هو مشيّد يقبل وسيطين من خلاله تصبح قادراً على أن تسند قيمتي المتغيّرين x , y ، و المشيّد العادي.

3- تأمّل البرنامج التالي:

```
#include<iostream>
using namespace std;
int main()
{
  cout<<"Sweat plus sacrifice equals success.";
  return 0;
}
```

اجعل البرنامج يطبع الشكل التالي[10]:

```
Give the world the best you have, and the best will come to you.
Sweat plus sacrifice equals success.
If man hasn’t discovered that he will die for, he isn’t fit to live.
```

وذلك دون أن تعدّل في التابع main أبداً.

(تذكر أنّ المتغيّرات الشاملة تتمّ تبدئتها في بداية البرنامج قبل التابع main و يتمّ هدمها بعد التابع main).

[10] العبارة الأولى لـ Madeline Bridge والثانية لـ Charles O. Finley ، والثالثة لـ Martin Luther King, JR

المؤشّرات على الصفوف Pointers to classes

يمكننا أن ننشئ مؤشّراً ليشير على صفّ ما بشكلٍ حقيقيٍ، ولفعل ذلك علينا ببساطة أن نُعرِّف مؤشّراً على كائن من نوع الصفّ كما يلي:

CRectangle * prect;

يمثل المؤشّر السابق مؤشّراً على كائن من النوع CRectangle، ولكي نشير بشكل مباشر إلى عضو ما من أعضاء هذا الكائن ينبغي أن نستخدم المعامل **->** والمثال التالي يوضّح بعض العبارات الممكنة.

```
// pointer to classes example
#include <iostream>
using namespace std;

class CRectangle {
    int width, height;
  public:
    void set_values (int, int);
    int area (void) {return (width * height);}
};

void CRectangle::set_values (int a, int b) {
  width = a;
  height = b;
}

int main () {
  CRectangle a, *b, *c;
  CRectangle * d = new CRectangle[2];
  b= new CRectangle;
  c= &a;
  a.set_values (1,2);
  b->set_values (3,4);
  d->set_values (5,6);
  d[1].set_values (7,8);
  cout << "a area: " << a.area() << endl;
  cout << "*b area: " << b->area() << endl;
  cout << "*c area: " << c->area() << endl;
  cout << "d[0] area: " << d[0].area() << endl;
  cout << "d[1] area: " << d[1].area() << endl;
  return 0;
}
```

```
a area: 2
*b area: 12
*c area: 2
d[0] area: 30
d[1] area: 56
```

وهذه قائمة مختصرة بطرق قراءة المؤشّرات والمعاملات (*, &, . , ->, [ ]) :

المثال	**يقرأ كالآتي**
***x**	القيمة التي يشير إليها المؤشّر x
&x	عنوان الكائن x
x.y	العضو y من المؤشّر x
(*x).y	العضو y من الكائن المشار إليه بالمؤشّر x
x->y	العضو y من الكائن المشار إليه بالمؤشّر x
x[0]	الكائن الأول الذي يشير إليه المؤشّر
x[1]	الكائن الثاني الذي يشير إليه المؤشّر
x[n]	الكائن رقم n الذي يشير إليه المؤشّر

تأكّد من أنّك تمكّنت من فهم كل هذه العمليات بشكل تام، وإذا كان لديك أيّة شكوك حولها فننصحك بقراءة فصلي المؤشّرات pointers والسجلّات struct.

الكلمة المفتاحية this

هذه الكلمة داخل الصفّ تشير إلى العنوان في الذاكرة للكائن من هذا الصفّ الذي يُنفّذ حالياً. وهي تمثّل مؤشّراً قيمته دائماً عنوان هذا الكائن.

يمكن أن تستخدم هذه الكلمة لمعرفة فيما إذا كان *الوسيط الممرر* إلى تابعٍ عضوٍ ما من الكائن هو *الكائن نفسه*.
مثال:

```
// this
#include <iostream>
using namespace std;
class CDummy {
  public:
    int isitme (CDummy& param);
};
int CDummy::isitme (CDummy& param)
{
  if (&param == this) return 1;
  else return 0;
}
int main () {
  CDummy a;
  CDummy* b = &a;
  if ( b->isitme(a) )
    cout << "yes, &a is b\n";
  return 0;
}
```

```
yes, &a is b
```

عادةً تستخدم هذه الكلمة في المعامل = الذي يقبل وسيطاً بالمرجع من نفس نوع الصفّ ويعيد كائناً منه. تتبّع المثال التالي ولاحظ كيف أنّنا استخدمنا الكلمة this للتخلّص من المتحول الموضعي كما فعلنا في المثال الأخير من الفقرة السابقة :

```
CVector&  CVector::operator= (const CVector& param)
{
  x = param.x ;
  y = param.y ;
  return *this ;
}
```

وهذه الشيفرة هي الشيفرة الافتراضية للمعامل = من أجل أي صفّ إذا لم نقم نحن بإعادة تحميل المعامل = وتغيير الشيفرة.

ملاحظة: الأعضاء من النوع static لا يمكن أن نشير إليها بالمؤشّر this.

عندما يُستدعى عضو ما - ليس من النوع static- من كائن فإنّ عنوان الكائن يمرّر كوسيط خفي إلى هذا العضو كما يلي:

```
myDate.setMonth( 3 );
```

و الذي يعني :

```
setMonth( &myDate, 3 );
```

التعبير (this*) يستخدم عادة من أجل إعادة الكائن الحالي كقيمة للتابع العضو كما فعلنا سابقاً.

ملاحظة: الكلمة this ليست موجودة في نسخ C++ القديمة.
مثال يوضح الطرق الممكنة لاستخدام this :

```
// Example of the this pointer
class  Date{itn month ; void setMonth(int);}
void   Date::setMonth( int mn )
{
  month = mn;          // هذه العبارات الثلاث متكافئة
  this->month = mn;
  (*this).month = mn;
}
```

لنفترض أن لدينا المثال التالي :

```
// this keyword example
#include <iostream>
using namespace std;
class Example
{
       int x;
public:
       void setX(int x){ x = x; }
    void printX(){cout<< x << endl; }
};
int main()
{
  Example ex ;
  ex.setX(5);
  ex.printX();
  return 0;
}
```

```
-858993460
```

ما هي نتيجة هذا البرنامج ؟ في حقيقة الأمر سيطبع هذا البرنامج **قيمةً عشوائيّةً** هي قيمة المتغيّر العضو x الذي لم تُسند إليه أيّة قيمة. و**لكن كيف؟ و مالذي فعله التابع set_x(int x) ؟** هذا التابع قام بإسناد قيمة المتغيّر x إلى المتغيّر x لكن أيّ المتغيّرين هو المتغيّر العضو وأيّهما هو المتغيّر الوسيط؟ هذه هي المشكلة لذلك فإنّ المترجم يعتبر هذه التعليمة هي إسناد قيمة المتغيّرالعضو x إلى نفسه وبما أنّ قيمته **عشوائيّة** لذلك سيطبع التابع print قيمة المتغيّر العضو x العشوائيّة. **لكن ما هو الحل؟** راقب الفرق في المثال التالي:

```
// this keyword example
#include <iostream>
using namespace std;
class Example
{
       int x ;
public:
       void set_x(int x) { this->x = x; }
        void printX(){cout<< x << endl ;}
};

int main()
{
  Example ex ;
  ex.set_x(5);
  ex.printX();
  return 0;
}
```

```
5
```

الصفوف المعرّفة باستخدام الكلمة المحجوزة struct

وسّعت C++ مفهوم الكلمة المحجوزة struct إلى مفهوم الصفّ class ماعدا أنّ أعضاء السجل تكون من المستوى public بشكل افتراضي بدلاً من أن تكون من المستوى private كما في الصفّ.

على أية حال كلٌّ من الصفّ و السجل لهما نفس الوظائف تقريباً في C++ ، لكن struct يستخدم عادة من أجل البيانات فقط أمّا class فهي تستخدم من أجل الصفوف التي تحتوي على أعضاء من البيانات والتوابع و الإجراءات.

إعادة تحميل المعاملات Overloading operators

أتاحت لنا C++ الخيار في استخدام معاملاتها القياسية بين الصفوف التي تنشئها أنت بالإضافة إلى إمكانيّة استخدامها بين الأنواع الأساسية. مثال:

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

يمكنك بالتأكيد استخدام معامل الجمع + بين متغيّرين (كائنين) من النوع الأساسي int. و لكن ماذا عن المثال التالي:

```
struct { char product [50]; float price; } a, b, c;
a = b + c;
```

في الواقع لا يجوز أن نفعل ذلك على الرغم من أنّ إسناد كائن من صفّ ما إلى كائن آخر من نفس الصفّ يعدّ أمراً ممكناً (ينسخ المشيّد بشكل افتراضي) ، لكنّ عمليّة الجمع السابقة هي التي ستُنتِج الخطأ لأنّ معاملَ الجمعِ غيرُ متاحٍ إلّا بين الأنواع الأساسية.

إلّا أنّنا نودّ هنا أن نقدّم الشكر إلى C++ لأنّها تمكّننا من إعادة تحميل معاملاتها القياسية الأمر الذي يجعلنا قادرينَ على أن نكتب المثال السابق، فالكائنات المشتقّة من أنواع مركّبة كما في المثال السابق تستطيع أن تقبل المعاملات بعد إعادة تحميلها، كما سنكون قادرين بذلك على أن نغيّر في آلية عمل المعاملات.

هذه هي المعاملات التي يمكننا أن نعمل عليها تحميلاً زائداً:

```
+     -     *     /     =     <     >     +=    -=    *=    /=    <<    >>    <<=
>>=   ==    !=    <=    >=    ++    --    %     &     ^     !     |     ~     &=
^=    |=    &&    ||    %=    []    ()    new   delete
```

كيف نقوم بعمل إعادة التحميل على المعاملات؟ للقيام بذلك نحتاج فقط إلى اتّباع القاعدة التالية:

type operator sign(parameters);

وكما تلاحظ فقد التعريف المبدئي prototype كما يمكننا أن نقوم بكتابة التابع بشكله العادي مباشرة داخل الصفّ.

و الآن لنقل إنّنا نريد أن نكتب صفّاً جديداً من أجل الأشعّة ثنائيّة البعد والتي تأخذ الشكل:

$$\vec{v} = v(x, y) \quad : x, y \in Z$$

و نريد أن نقوم بعمل تحميل زائد على المعامل + ليقوم بجمع شعاعين. حيث يخضع جمع شعاعين إلى القاعدة التالية: $\vec{v} = v(x_1, y_1), \vec{w} = w(x_2, y_2) \Rightarrow \vec{e} = \vec{v} + \vec{w} : \vec{e} = e(x_1 + x_2, y_1 + y_2)$

مثال: $\vec{v} = v(3,1), \vec{w} = w(1,2) \Rightarrow \vec{e} = \vec{v} + \vec{w} : \vec{e} = e(3 + 1,1 + 2) = e(4,3)$

```
// vectors: overloading operators example
#include <iostream>
using namespace std;
class CVector {
  public:
    int x,y;
    CVector () {};
    CVector (int,int);
    CVector operator + (CVector);
};
CVector::CVector (int a, int b) {
  x = a;
  y = b; }
CVector CVector::operator+ (CVector param) {
  CVector temp;
  temp.x = x + param.x;
  temp.y = y + param.y;
  return (temp); }
int main () {
  CVector a (3,1), b (1,2), c;
  c = a + b;
  cout<<"a(3,1) + b(1,2) = ";
  cout<<"c("<< c.x << "," << c.y <<")\n";
  return 0;
}
```

```
a(3,1) + b(1,2) = c(4,3)
```

حيث أنّ :

CVector (int, int); : اسم التابع المشيّد للصفّ CVector

CVector operator+ (CVector); : تابع معامل الجمع الذي يقوم بإعادة التحميل على المعامل الرياضي + و هو يعيد النوع CVector.

يمكن أن نقوم باستدعاء هذا التابع بأحد الشكلين التاليين:

```
c = a + b;
```

أو

```
c = a.operator+ (b);
```

لاحظ أيضاً أننا أضفنا المشيّد العادي CVector(){ }; يعدّ هذا أمراً ضرورياً جداً لأنّنا قمنا بتعريف المشيّد CVector(int, int); ما يعني أنّه لن يكون هناك أيّاً من المشيّدين الافتراضيين (راجع فقرة المشيّد ات في الدرس السابق) ولذلك لابّد لنا من إضافته بأنفسنا حتّى يعمل البرنامج، فعدم إضافته تعني أنّ التصريح التالي CVecotr c; غير ممكنٍ داخل التابع main() أو غيره. (بينما يكون التصريحان الآخران ممكنين بسبب وجود مشيّد يأخذ وسيطين)

على أيّة حال ، يجب عليّ أن أُعلِمك أنّه من الأفضل عدم ترك المشيّد العادي أو nop Constructor بدون تعليمات لأنّه وكما في مثالنا لو أنّنا صرحنا عن كائن ما باستخدام المشيّد العاديّ كالآتي :

```
CVector c;
```

فإنّ قيم x, y لن تكون معروفة (عشوائية)، وبالتالي يفضّل أن نقوم بوضع بعض التعليمات التي تساعدنا على التحكم بالأعضاء وتبدنتها. كما يلي:

```
CVector () { x=0; y=0; };
```

لم نقم بذلك في المثال السابق للسهولة فقط.

كما أنّ الصفوف تحتوي بشكل افتراضي على مشيّدين (كما مر معنا سابقاً) فهي تحتوي أيضاً على تعريف لمعامل الإسناد = بين كائنين من نفس الصفّ. حيث يعني: إنسخ كل الأعضاء غير الثابتة(non-static) من الكائن الذي على يمين المعامل إلى ما يقابلها من أعضاء الكائن الذي على يساره.

و يمكنك التحكم بذلك حيث يمكنك أن تحدد الأعضاء التي تود نسخها.

لا تفرض عليك إعادة تحميل المعاملات أن تستخدمها وفقاً لمعنىً عامٍّ أو لعلاقةٍ رياضيةٍ، و على الرغم من ذلك فإنّ استخدام المعاملات وفقاً لوظائفها يعدّ أفضل، إذ ليس من المنطقي أن تعيد تحميل المعامل + ليقوم بطرح كائنين من نفس النوع على الرغم من أنّ هذا ممكنٌ برمجيّاً.

تمرين 3

في تمرين الصفّ `point` أعد تحميل معامل الجمع ليقوم بجمع نقطتين. حيث

$$c(x,y) = a(x1,y1) + b(x2,y2) : \quad x = x1 + x2 \quad , \quad y = y1 + y2$$

الأعضاء الثابتة من النوع static

يمكن أن يحتوي الصفّ على أعضاء من النوع static ، والتي تُعرَف أيضاً بـ "متغيّرات الصفّ"، لأنها المحتوى الذي لا يعتمد على أي كائن. وهي تمثل فقط قِيَماً فريدة و موحّدة لكل الكائنات من نفس النوع.

كمثال، سنقوم بالتصريح عن متغيّر من النوع static لنستخدمه في إحصاء عدد الكائنات التي سيتم إنشاؤها (هدمها) خلال البرنامج:

```
// static members in classes
#include <iostream>
using namespace std;
class CDummy {
  public:
    static int n;
    CDummy () { n++; };
    ~CDummy () { n--; };
};
int CDummy::n=0;
int main () {
  CDummy a;  // create 1th  instance
  CDummy b[5]; // create 2ed to 6th instances
  CDummy * c = new CDummy; //create 7th instance
  cout << a.n << endl; //n = 7
  delete c; //delet 7th instance
  cout << CDummy::n << endl; //n = 6
  return 0;
}
```

```
7
6
```

في الواقع تتمتّع الأعضاء من هذا النوع بخصائص المتغيّرات الشاملة على كامل الصفّ. ولهذا السبب و لتجنّب التصريح عن أيّ منها عدّة مرّات في الصفّ الواحد، ووفقاً لقواعد ANSI-C++ القياسية يمكننا أن نصرّح عنها باستخدام التعريف المبدئي prototype داخل الصفّ ونترك جسم التابع العضو أو إسناد القيم إلى المتغيّرات خارج الصفّ كما فعلنا في المثال السابق.

الأعضاء الثابتة توجد منها نسخة واحدة فقط في الصفّ الأساسي ولا يتمّ نسخها إلى الكائنات المشتقّة لأنّ هذه الأعضاء فريدة و مشتركة بين جميع الكائنات من نفس النوع، فيمكننا أن نشير إلى أي منها كعضو من أي كائن أو مباشرة من الصفّ نفسه وذلك بذكر اسم الصفّ ثمّ المعامل :: ثمّ اسم العضو من النوع static (طبعاً هذا الشيء يصلح فقط من أجل الأعضاء من النوع static) كما يلي:

```
cout<<a.n;
cout<< CDummy::n;
```

في حال كان العضو تابعاً من النوع static عندئذ فإنّ هذا التابع لا يمكنه أن يتعامل إلّا مع المتغيّرات من النوع static. كما نذكّر بأنّ الأعضاء static لا تمتلك المؤشّر this لأنّ هذه الأعضاء لا تمثّل أعضاء خاصة بالكائن وإنّما هي أعضاء مشتركة بين جميع الكائنات من هذا النوع.

تمرين 4 : أوجد الأخطاء في البرنامج التالي:

```
class Example
 {
 public:
   Example  ( int y = 10 )
     : data( y )
   {
     // empty body
   } // end Example constructor

   int getIncrementedData() const
   {
     return data++;
   } // end function getIncrementedData

   static int getCount()
   {
     cout << "Data is " << data << endl;
     return count;
   } // end function getCount

 private:
   int data;
   static int count;
 }; // end class Example
```

الفصل الثامن عشر:

مبادئ البرمجة كائنية التوجه

(Object-Oriented programming principles)

تعتمد البرمجة الكائنية على ثلاثة مبادئ أساسية هي :

1- مبدأ التغليف Encapsulation.
2- الوراثة Inheritance.
3- تعدد الأشكال Polymorphism.

1- مبدأ التغليف Encapsulation

يهدف هذا المبدأ إلى تجميع البيانات الخاصّة بصفّ ما وإخفائها حتّى لا تكونَ ظاهرةً للآخرين.

وللقيام بذلك فإنّنا نمنع بيانات الصفّ (متغيّراته) من الظهور للمستخدم بشكل مباشر ونقدّم له عوضاً عن ذلك توابع (أو ما يعرف في لغات أخرى مثل java و C# باسم الطرق mehods) لتقوم بالتواصل مع بيانات الصفّ. لنفترض أن لدينا المثال التالي :

```
#include<iostream>
using namespace std;
class Person
{
public :
       int age;
};
void main()
{
  Person my_person;
  cout<<"Enter Your age(1 to 110): ";
  cin >> my_person.age ;
  cout<<"\nYour age is: " << my_person.age
      << endl;
}
```

```
Enter Your age(1 to 110):21
Your age is: 21
```

تستطيع أن ترى أنّنا تمكنّا من *الوصول إلى المتغيّر age في الكائن my_person و تخزين القيمة 21 فيه* *مباشرةً* وهذه هي المشكلة تصوّر أنّ المستخدم أدخل رقماً من خارج مجال الأعداد الصحيحة ماذا لو أدخل المستخدم رمزاً أو حتّى حرفاً ؟ في هذه الحالة سيحدث خطأ وقد يتسبب بتوقف البرنامج عن العمل. يبدو مثل هذا الخطأ مستبعداً في حالتنا ولكنّه من المحتمل أن يحدث ومهمّتك عزيزي المبرمج أن تمنعه من أن يحدث.
ولكن كيف ؟ يبدو هذا السؤال ذكياً والإجابة عليه ستكون سهلة لأنّ ما سنقوم به هو أنّنا سنعمل تغليفاً لأعضاء هذا الصفّ بحيث نمنع بياناته (متغيّراته) من الظهور للمستخدم بشكل مباشر وسنقدّم له بدلاً من ذلك إحدى الطريقتين التاليتين:

a- استخدام توابع public للوصول إلى أعضاء private :

```
#include<iostream>
using namespace std;
class Person
{
 int m_age ;
public :
 void SetAge(int p_age)
 {
   if (p_age > 100 || p_age < 1)
   {
     cout<< "you can't edit age like that";
     m_age =1 ;
   }
   else
   {
     m_age = p_age;
     cout<< "done ";
   }
 }
 int GetAge()
 {
   return  m_age;
 }
};
void main()
{
  Person my_person;
  int my_age =0;
  cout<<"Enter Your age(1 to 110): ";
  cin>> my_age;
  my_person.SetAge(my_age) ;
  cout<<"\nYourage is: "<<my_person.GetAge()
      << endl;
}
```

```
Enter Your age(1 to 110):21
done
Your age is: 21
```

b- استخدام المشيّد ات : أضف المشيّد التالي إلى البرنامج السابق (لا تنس أنّ المشيّد العادي لن يكون موجوداً بعدها حتّى تضيفه بنفسك)

```
Person(int p_age)
{ if (p_age > 100 || p_age < 1)
   {  cout<< "you can't edit age like that";
      m_age =1 ;
   }
   else
   {  m_age = p_age;
      cout<< "done";
  }
}
```

قبل الحديث عن الوراثة لنتعرّف على التوابع الصديقة لأنها مهمّة لنا في دراسة الوراثة

التوابع friend (الكلمة المفتاحية friend)

نعلم أنّ هناك ثلاثة معرّفات وصول هي: private, public, protected وقلنا في الحالة التي يكون فيها عضو ما من أحد المستويين private أو protected فلا يمكن الوصول إليه من خارج الصفّ. وعلى الرّغم من ذلك يمكنك أن تتعدّى هذه القاعدة باستخدام الكلمة المحجوزة friend في الصفّ، عندها يمكن الوصول إلى هذا العضو من خارج الصفّ. ولفعل ذلك عليك فقط أن تصرّح عن العضو الذي تريد الوصول إليه من خارج الصفّ باستخدام التعريف المبدئي prototype ثمّ تذكر هذه الكلمة أمامه. مثال:

```
// friend functions
#include <iostream>
using namespace std;

class CRectangle {
    int width, height;
  public:
    void set_values (int, int);
    int area (void) {return (width * height);}
    friend CRectangle duplicate (CRectangle);
};

void CRectangle::set_values (int a, int b) {
  width = a;
  height = b;
}

CRectangle duplicate (CRectangle rectparam)
{
  CRectangle rectres;
  rectres.width = rectparam.width*2;
  rectres.height = rectparam.height*2;
  return (rectres);
}

int main () {
  CRectangle rect, rectb;
  rect.set_values (2,3);
  rectb = duplicate (rect);
  cout << rectb.area() << endl;
}
```

```
24
```

من تعريف التابع duplicate نجد أنه من النوع friend في الصفّ CRectangle، ما يجعلنا قادرين على الوصول إلى المتغيّرين width و height علماً أنّهما من المستوى private وذلك من أي كائن من النوع CRectangle.

لاحظ أنّنا لم نعتبر التابع duplicate عضواً من الصفّ CRectangle لا في التصريح عنه ولا في الاستخدام الأخير في التابع main(). كما يمكن له أن يأخذ المستوى العام public أو الخاص private فهذا لن يؤثر على استخدامنا له.

يمكن استخدام التوابع من هذا النوع من أجل القيام بعمليات على صفّين من نوعين مختلفين. عموماً هذا الاستخدام ليس من مبادئ البرمجة كائنية التوجه OOP لذلك من الأفضل استخدام أعضاء من ضمن الصفّ فمثلاً سيكون من المفيد (بغرض اختصار الشيفرة) في المثال السابق لو قمنا بكتابة التابع duplicate بشكل كامل على أنه عضو من الصفّ CRectangle.

الصفوف من النوع friend

يمكننا أن نصرّح عن صفّ من النوع friend كصديق لصفّ آخر كما فعلنا في التوابع من هذا النوع في الفقرة السابقة ممّا يمكّن الصفّ الصديق friend من الوصول إلى كافّة أعضاء الصفّ الآخر حتّى التي من المستوى private أو protected. مثال:

```
// friend class
#include <iostream>
using namespace std;
class CSquare;
class CRectangle {
    int width, height;
  public:
    int area (void)
      {return (width * height);}
    void convert (CSquare a);
};
class CSquare {
  private:
    int side;
  public:
    void set_side (int a)
      {side=a;}
    friend class CRectangle;
};
void CRectangle::convert (CSquare a) {
  width = a.side;
  height = a.side;
}
int main () {
  CSquare sqr;
  CRectangle rect;
  sqr.set_side(4);
  rect.convert(sqr);
  cout << rect.area() << endl;
  return 0;
}
```

```
16
```

صرّحنا في هذا المثال عن الصفّ CRectangle على أنّه صديق friend للصفّ CSquare لذلك فإنّ الصفّ CRectangle يمكنه الوصول إلى كل أعضاء الصفّ CSquare.

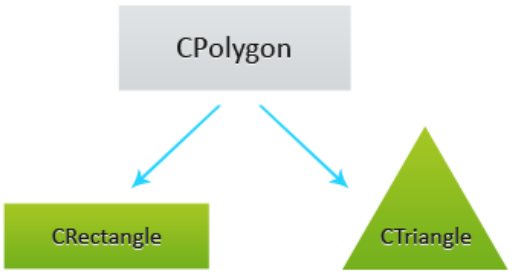
ربما تجد التعليمة class CSquare ; غريبةً بعض الشيء في الواقع تمثّل هذه التعليمة التصريح المبدئي prototype عن الصفّ CSquare لأنّنا استخدمنا هذا الصفّ كوسيط في التابع convert() ولو لم نقم بالتصريح عنه بهذا الشكل لكان من غير الممكن استدعاؤه كوسيط في هذا التابع من الصفّ CRectangle.

في حالتنا هذه اعتبرنا أنّ CRectangle صديق friend للصفّ CSquare ما يعني أنّ بإمكانه أن يصل إلى أيّ عضو موجود في الصفّ CSquare حتّى ولو كان من المستوى private أو protected و لكنّ العكسَ غيرُ ممكنٍ علماً أنّه لا شيء يمنعنا من أن نجعل الصفّ SCquare صديقاً friend للصفّ CRectangle.

2- الوراثة بين الصفوف Inheritance between classes

تعدّ الوراثة صفّة مهمّة من صفات الصفوف. فهي تسمح لنا بإنشاء كائناً (ابناً) مشتقّاً من كائنٍ آخرَ (أبٍ) حيث يصبح الكائنُ الابنُ حاوياً على بعض صفات (خصائص) الكائن الأبّ بالإضافة إلى صفاته الأساسية. (الصفات هي الأعضاء العالمّة المعرّفة في الصفّ والتي تسمح بالتعامل معه). لنأخذ مثالاً على ذلك:
لنفترض أنّنا أردنا أن ننشئ مجموعة من الصفوف تصف الأشكال الهندسيّة ذوات الأضلاع polygons كالتي أنشأناها سابقاً مثل CRectangle أو CTriangle. هذه الأشكال تمتلك بعض الصفات الأساسيّة المشتركة بينها جميعاً، فكلٌّ هذه الأشكال تمتلك قاعدة (عرض) وارتفاع.

لذلك سنقوم بالتصريح عن صفّ يحوي الصفات المشتركة (عرض و ارتفاع) لكلّ الأشكال المضلّعة وسنسميه CPolygon حيث أنّ كل شكل مضلّع يستطيع أن يرث صفاته الأساسيّة من الصفّ CPolygon. كما في الشكل:



الصفوف التي تُشتَقّ من الصفوف الأساسيّة ترث جميع أعضائها الظاهرة. وهذا يعني أنّه لو كان الصفّ الأساسيّ يمتلك عضواً S ولو صنعنا صفّاً جديداً فيه العضو B ويرث الصفّ الأساسيّ فإنّ الصفّ الجديد سيمتلك العضوين S و B معاً.

كيف يمكننا أن نجري وراثةً بين صفّين؟ يتمّ ذلك باستخدام المعامل (**:**) الذي يُخبر المترجم بأنّ الصفّ الذي قبل (على يسار) هذا المعامل سيرث الصفّ الذي بعده (على يمينه). بالطريقة التالية:

class derived_class_name **: public** base_class_name;

حيث أن :
derived_class_name: اسم الصفّ المشتق.
base_class_name: اسم الصفّ الأساسي.
public: معرّف الوصول يمكن أن يُستبدَل بأيٍّ من المعرّفين الآخرين (private, protected). وسنتعرّف على فائدته بعد المثال:

```
// derived classes
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b;}
  };
class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };
```

```
20
10
```

```
class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

كما ترى فإنّ كلّاً من الصفّين CRectangle و CTriangle يحتوي على الأعضاء width, height, set_values() الموجودين أساساً في الصفّ CPolygon.

الكلمة protected مشابهة للكلمة private والفرق الوحيد يحدث عندما نشتقّ (نرث) الصفوف من بعضها. عندما نشتقّ (نرث) صفّاً من صفّ ما آخر فإنّ الأعضاء من النوع protected الموجودة في الصفّ الأساسي يمكن الوصول إليها من قبل أعضاء الصفّ الجديد بينما تلك التي من النوع private فلا يمكن الوصول إليها إلّا من خلال الصفّ الأساسيّ نفسه.

و لأننا نريد أن نتعامل مع كلٍّ من العضوين width, height من خلال الصفوف الجديدة المشتقّة من الصفّ CPolygon لذلك صرّحنا عنهما على أنّهما من النوع protected.

لاحظ الجدول التالي الذي يحدِّد الفرق بين معرّفات الوصول.

إمكانية الوصول إلى	public	protected	private
الأعضاء من نفس الصفّ	✓	✓	✓
الأعضاء من الصفوف المشتقة	✓	✓	×

في مثالنا ، الأعضاء الموروثة في كلّ من الصفّين CTriangle, CRectangle تتبع نفس مستويات الوصول في الصفّ الأساسي CPolygon.

```
CPolygon::width          // protected access
CRectangle::width        // protected access
CPolygon::set_values()   // public access
CRectangle::set_values() // public access
```

وذلك لأنّنا استخدمنا المعرف public لاشتقاقهما (ليرثا) من الصفّ CPolygon كما يلي :

```
class CRectangle : public CPolygon;
```

إنّ الكلمة المفتاحية public تمثّل أدنى مستوى من الحماية للأعضاء الموروثة من الصفّ الأساسيّ ويمكن تغيير هذا المستوى ليكون أحد المستويين الآخرين. كمثال لنفترض أنّنا عرّفنا الصفّ الجديد daughter المشتقّ من الصفّ الأساسيّ mother وفق المستوى protected كالآتي:

```
class daughter : protected mother;
```

هذا ما سيجعل المستوى protected هو المستوى الأدنى لحماية أعضاء الصفّ الأساسي لأنّ كلّ أعضائه التي كانت من النوع public في الصفّ mother ستصبح من النوع protected في الصفّ daughter. بالطبع هذا لا يمنع من أن يحوي الصفّ daughter على أعضاء خاصّة به من النوع public لأنّ التغيير في مستوى الوراثة سيطبق على أعضاء الصفّ الأساسيّ mother فقط.

أمّا استخدام الكلمة private فيُعتبر الأكثر انتشاراً بعد الكلمة public وذلك لأنّها تؤمّن تغليفاً كاملاً لأعضاء الصفّ الأساسي حيث لا يمكن الوصول إلى أعضاء الصفّ الأساسي إلّا من داخله.

على كلٍّ إذا لم تذكر معرّف الوصول بشكل صريح أثناء تعليمة الوراثة فإنّ المعرّف الافتراضي هو private وذلك من أجل الصفوف المعرّفة باستخدام الكلمة class، أمّا تلك المعرفة باستخدام الكلمة struct فإنّ المعرّف الافتراضي لها هو public.

ما هي الأشياء التي يتمّ وراثتها من الصفّ الأساسي؟ بحسب مبادئ الوراثة فإنّ كلّ أعضاء الصفّ الأساسي ستورَّثُ إلى الصفّ الابن (المشتقّ) ما عدا الأعضاء التالية:

- **Constructor and destructor**
- **operator=() member**
- **friends**

على الرغم من أنّ المشيّد والهادم لا تتمّ وراثته إلّا أنّه سيتمّ استدعاء المشيّد الافتراضي (كالمشيّد العادي مثلاً) أو الهادم من الصفّ لأساسي عند إنشاء أو هدم أيّ صفّ مشتقّ من الصفّ الأساسيّ. أمّا إذا لم يكن في الصفّ الأساسيّ مشيّد افتراضي (لأنّك قمت بإعادة تحميل المشيّد بطريقة ما) و أردت أن يتمّ استدعاء المشيّد الجديد في كلّ مرّة يتمّ فيها إنشاء صفّ جديد فعليك أن تصرّح عن ذلك بالشكل :

derived_class_name (parameters) : base_class_name (parameters) {…}

```
// constructors and derivated classes
#include <iostream>
using namespace std;
class mother {
  public:
    mother ()
      { cout << "mother: no parameters\n"; }
    mother (int a)
      { cout << "mother: int parameter\n"; }
};
class daughter : public mother {
  public:
    daughter (int a)
      {cout<< "daughter: int parameter\n\n";}
};


class son : public mother {
  public:
    son (int a) : mother (a)
      { cout << "son: int parameter\n\n"; }
};
int main () {
  daughter x (1);
  son y (1);

  return 0;
}
```

```
mother:   no   parameters
daughter:  int  parameter

mother:   int   parameter
son: int parameter
```

راقب الفرق بين المشيّد من الصفّ الأساسي الذي تمّ استدعاؤه عندما أنشأنا الكائن x من الصفّ daughter و المشيّد الذي تمّ استدعاؤه عندما أنشأنا الكائن y من الصفّ son هذا الفرق سببه أنّنا عرّفنا المشيّد في الصفّ daughter بالشكل التالي:

daughter (int a)

في هذه الحالة سيتمّ استدعاء المشيّد الافتراضي من الصفّ الأساسي وهو mother() وذلك لأنّنا لم نخبر المترجم أيّ المشيّدات عليه أن يستدعيه فهو يقوم باستدعاء المشيّد الافتراضي.

أمّا في الصفّ son فإنّ المشيّد كان بالشكل التالي:

son (int a) : mother (a)

في هذه الحالة أخبرنا المترجم أنّ عليه أن يستدعي المشيّد الذي يقبل وسيطاً من النوع int وذلك باستخدام المعامل : عند التصريح عن المشيّد في الصفّ الجديد.

الوراثة المتعددة Multiple inheritance

يمكنك في C++ أن تصنع صفّاً يرث صفاته من أكثر من صفّ في نفس الوقت وهذا يشبه تماماً ما يحدث للإنسان فكلنا يعلم أنّ الصفات التي تكون في المولود الصغير بعضُها قادمٌ من الأب و بعضُها الآخر من الأم أي أنّه لو افترضنا أنّ لدينا صفّاً يمثل المولود وآخر يمثّل الأب وثالث يمثل الأم فإنّ بإمكان الصفّ المولود أن يرث من الأب والأم معاً. تتمّ هذه العملية بأن نضع فاصلة (,) بين أسماء الصفوف التي نريد من الصفّ الجديد أن يرثها كما يلي:

class derived_class_name : **public** base_class_name1 , base_class_name2 ;

مثال آخر، لو عرّفنا صفّاً جديداً أسميناه COutput و أردنا من الصفّين CTriangle, CRectangle أن يرثا الصفات الموجودة في هذ الصفّ بالإضافة إلى تلك التي ورثاها سابقاً من الصفّ CPolygon فبإمكاننا أن نكتب عندها :

class CRectangle : **public** CPolygon, public COutput {
class CTriangle : **public** CPolygon, public COutput {

إليك المثال كاملاً:

```
// multiple inheritance
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b;}
  };
class COutput {
  public:
    void output (int i);
  };
```

```
20
10
```

```
void COutput::output (int i) {
  cout << i << endl;
  }
class CRectangle: public CPolygon, public
COutput {
  public:
    int area (void)
      { return (width * height); }
 };
class CTriangle: public CPolygon, public
COutput {
  public:
    int area (void)
      { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  rect.output (rect.area());
  trgl.output (trgl.area());
  return 0;
}
```

3- تعدد الأشكال Polymorphism

يسمح هذا المبدأ بكتابة صفّ يحتوي على الهيكل العام للصفّ من دون أيّة تعليمات (تضمينات) لكي يجبر المبرمجين الذين سيشتقّون هذا الصفّ على تضمين الكائنات التي ينشؤنها من هذا الصفّ من جديد.
من أجل فهم أفضل للفقرات التالية ننصح بالتمكّن من المفاهيم التالية (المؤشّرات – الوراثة بين الصفوف). إذا كانت إحدى العبارات التالية غير مفهومة عندك ننصحك بالتمرّن على المفاهيم السابقة الذكر بشكل أفضل :

int a::b(c) {}; // Classes

a->b // pointers and objects

class a: public b; // Relationships between classes

المؤشّرات على الصفّ الأساسي Pointers to base class

واحدة من أهمّ الميزات التي نحصل عليها باشتقاق أصناف جديدة من أصناف أساسيّة هي أنّ المؤشّر على أيٍّ من الأصناف المشتقّة متوافق مع مؤشّر على الصفّ الأساسي الذي اشتُقّ منه هذا الصف.
هذه الفقرة مخصّصة بالكامل للحديث عن قوّة هذه المزايا التي تقدمها C++. كمثال، سنقوم بكتابة برنامج الأشكال الهندسية السابق باستخدام هذه المزيّة :

```
// pointers to base class
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
  };
class CRectangle: public CPolygon {
  public:
    int area (void)
```

```
20
10
```

```
      { return (width * height); }
  };
class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

في التابع الرئيسي main() أنشأنا مؤشّرين ppoly2 * , ppoly1 * على كائنين من الصفّ CPolygon.
و أسندنا أليهما عنواني الكائنين rect, trgl ولأنّهما كائنين من صفّين مشتقّين من الصفّ الأساسيّ CPolygon فإنّ هذا الإسناد ممكن. نستطيع أن نستخدم المؤشّرات من نوع الصفّ الأساسي لنشير بها على الأعضاء الموروثة في الصفوف المشتقّة منه فقط لذلك لم نقم باستخدامهما في استدعاء التابع area() وقمنا باستدعائه بالطريقة المعتادة.

الأعضاء من النوع virtual

في المثال السابق كي نستطيع التأشير على التابع area() يجب أن نجعله عضواً في الصفّ الأساسي ولكن هذا سيتسبب بمشكلة جديدة هي أنّ كلاً من الصفّين CRectangle, CTriangle يرثان صفاتهما من الصفّ الأساسي و التابع area() هو واحد من هذه الصفات إلّا أنّ مساحة المستطيل ناتجة عن ضرب الطول بالعرض، ومساحة المثلث ناتجة عن ضرب الطول بالعرض وتقسيم الناتج على 2، **فما هو الحل؟**.
الحل يكمن في أن نقوم بكتابة التابع area() في كل من الصفّين CRectangle, CTriangle ولكي نصبح قادرين على التأشير عليهما باستخدام مؤشّر من نفس نوع الصفّ الأساسي سنقوم بتعريفه في الصفّ الأساسي أيضاً لكن سيكون تعريفه من نوع خاصّ هو النوع virtual، ثمّ نقوم بتعريفه بنفس الاسم في كلّ من الصفّين CRectangle, CTriangle ونقوم بكتابة التعليمات المناسبة لكلّ منهما من جديد وهذا ما يُعرف بإعادة القيادة **Overriding**.

إذن الأعضاء من النوع virtual هي الأعضاء المشتركة التي نرغب بأن ترثها كل الصفوف المشتقة من الصفّ الأساسي والتي سنعيد كتابة تعليماتها (overriding) داخل بعض أو كل الصفوف التي ترث الصفّ الأساسي.

و بعد التعديل يصبح المثال بالشكل:

```
// virtual members
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void)
      { return (0); }
```

```
20
10
0
```

```
  };
class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };
class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon poly;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  CPolygon * ppoly3 = &poly;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly3->set_values (4,5);
  cout << ppoly1->area() << endl;
  cout << ppoly2->area() << endl;
  cout << ppoly3->area() << endl;
  return 0;
}
```

تستطيع أن ترى الآن أنّ كلّاً من الصفوف (CPolygon, CTriangle, CRectangle) تمتلك نفس الأعضاء (width, height, set_value(), area()).

و الآن جرّب أن تزيل الكلمة المحجوزة virtual ثمّ شغّل البرنامج، **ماذا ترى؟** النتيجة ستكون **0** في الحالات الثلاث بدلاً من **0 ,10 ,20**، **ولكن لماذا؟** لأنّك عندما أزلت الكلمة virtual من أمام التصريح عن التابع area() في الصفّ CPolygon ثمّ استخدمت مؤشّراً من نوع هذا الصفّ لتشير إلى التابع area() في كلّ من الكائنين trgl , rect فإنّ المؤشّر سيشير إلى التابع في الصفّ الأساسي والذي يعيد القيمة ّ كما هو واضح من تعريفه أمّا الكائن poly فهو أصلاً من النوع CPolygon والتابع المعرّف فيه يعيد 0 دائماً.

إنّ ما تفعله هذه الكلمة هو أنّها تسمح باستدعاء العضو من الصفّ المشتق والذي يمتلك نفس اسم العضو في الصفّ الأساسي عندما يستخدم المؤشّر، كما في المثال السابق.

لا يقتصر الأمر فقط على استخدام المؤشّرات ففي هذه الحالة التابع area() هو أصلاً من الصفات المشتركة في كل الأشكال الهندسية المضلّعة المغلقة لذلك علينا أن نقوم بوضعه في الصفّ الأساسي CPolygon ولأنّ طريقة حساب مساحة كل شكل تختلف عن الطرق الأخرى فإنّنا نصرّح عن التابع area() في الصفّ الأساسيّ على أنّه من النوع virtual ثمّ نعيد قيادته في الصفوف المشتقّة.

ومثال ذلك لو افترضنا أنّ لدينا صفّاً أساسيّاً للمركبات ثمّ اشتقّينا منه صفّاً للسيارة وآخر للدراجة النارية، معلومٌ أنّ كلّ المركبات تستخدم الوقود لكي تسير لذلك سنعرّف التابع fuel() في الصفّ الأساسي ليحسب كمية الوقود اللازمة لقطع مسافة معيّنة لكن **هل تستهلك السيارة نفس الكمية من الوقود التي تستهلكها الدراجة لتقطعا نفس المسافة؟** بالطبع لا و هذا الاختلاف يجعل من الأفضل أن نستخدم الكلمة virtual لنعيد قيادة التابع fuel() ضمن الصفّ المشتق كلّما دَعَت الحاجة لذلك.

```
// virtual members
#include <iostream >
using namespace std;
class Vehicle
{
protected:
 float distance ;
public:
 Vehicle()
 {
  distance = 0.0;
 }
 void SetDistance(float d)
 {
  distance = d;
 }
 virtual float fuel()
 {
  return distance/5;
 }
};

class Car: public Vehicle
{
public:
 float fuel()
 {
  return distance/10;
 }
};

class Cycle:public Vehicle
{
public:
 float fuel()
 {
  return distance/15;
 }
};

int main()
{
 Vehicle myvehicle;
 Car mycar;
 Cycle mycycle;
 myvehicle.SetDistance(10);
 mycar.SetDistance(10);
 mycycle.SetDistance(10);
 cout<<"Vehikle need: "<<myvehicle.fuel()
     <<" leters of fuel in 10 KM\n";
 cout<<"Car need: "<<mycar.fuel()
     <<" leters of fuel in 10 KM\n";
 cout<<"Cycle need: "<<mycycle.fuel()
     <<" leters of fuel in 10 KM\n";
return 0;
}
```

```
Vehikle need: 2 leters of fuel
in 10 KM
Car need: 1 leters of fuel in 10
KM
MotoCicle need: 0.666667 leters
of fuel in 10 KM
```

الصفوف الأساسيّة المجردة Abstract base classes

الصفوف المجرّدة البسيطة هي صفّوف تشبه إلى حدّ كبير الصفّ CPolygon الذي أنشأناه سابقاً والفرق البسيط يكمن في التابع area() المعرّف في الصفّ CPolygon حيث لا يمكن للصفّوف المجرّدة أن تحوي على تعليمات في توابعها التي من النوع virtual وبدلاً من كتابة التابع بكلّ تعليماته في الصفّ الأساسي سنكتفي فقط بالتصريح عن التابع دون كتابة أيّ تعليمة فيه ولن نضيف له الأقواس {} أيضاً لكن يمكننا في حالتنا هذه أن نضيف إلى التعريف قيمة ابتدائية ليعيدها التابع وذلك بإضافة الإسناد التالي:

```
virtual int area (void) = 0 ;
```

ليصير عندها شكل الصفّ CPolygon كالآتي:

```
// abstract class CPolygon
class CPolygon {
 protected:
   int width, height;
 public:
   void set_values (int a, int b)
     { width=a; height=b; }
   virtual int area (void) =0;
 };
```

لاحظ كيف أنّنا أضفنا المساواة **0** = إلى التعريف عن التابع عوضاً عن كتابة جسمه كاملاً. هذا النوع من التوابع يدعى pure virtual function، وكلّ الصفوف التي تحتوي على مثل هذا النوع من التوابع تدعى صفوفاً مجرّدة abstract classes.

الاختلاف الأكبر لهذه الصفوف عن غيرها أنّه لا يمكننا أن ننشئ منها نسخاً جديدة فهي غير قابلة للاشتقاق، لكنّه يمكننا أن نشير إليها باستخدام مؤشّر.

عند ذلك فإن التعليمة :

```
CPolygon poly ;
```

غير صحيحة لأنّ الصفّ CPolygon أصبح الآن من النوع المجرّد abstract class.

لكن يمكننا أن نقوم بالتصريحين الآتيين:

```
CPolygon * ppoly1;
CPolygon * ppoly2;
```

وهذا لأنّ التوابع من النوع pure virtual function لا تحتوي على تعليمات بداخلها وهذا يجعل من المستحيل إنشاء نُسخ من الصفّ المجرّد إذ أنّ بعض أعضائه لا تمتلك تعريفات كاملة لوظائفها، إلّا أنّه يمكننا أن نشير إلى الصفّ الذي يرث الصفّ الأساسي المجرّد باستخدام مؤشّر كما في المثال التالي:

```
// virtual members
#include <iostream>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
```

```
20
10
```

```
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
  };
class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };
class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << ppoly1->area() << endl;
  cout << ppoly2->area() << endl;
  return 0;
}
```

لو أعدت النظر في المثال السابق لرأيت أنّنا استطعنا أن نشير إلى كائنات من أنواع (صفوف) مختلفة باستخدام نوع واحد من المؤشّرات CPolygon *. هذا الشيء مفيدٌ للغاية. تصوّر أنّنا نريد الآن إنشاء تابعٍ عضوٍ في الصفّ الأساسي CPolygon ليطبع المساحة على الشاشة دون أن نعرّف نوع الصفّ الذي ورث هذا التابع من الصفّ الأساسي.

```
// virtual members
#include <iostream>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
    void printarea (void)
      { cout << this->area() << endl; }
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
```

```
20
10
```

```
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  return 0;
}
```

تذكر: تشير الكلمة(المؤشّر) this إلى الكائن الذي يتمّ تنفيذه في الذاكرة حالياً.

إنّ الصفوف المجرّدة و الأعضاء virtual هي التي تمنح لغة C++ خاصيّة تعدد الأشكال Polymorphism التي تعدّ إحدى أهمّ مبادئ البرمجة كائنيّة التوجّه OOP.

بالطبع التطبيقات التي أنشأناها بسيطة للغاية و لكنّها فقط لتوضيح هذه المبادئ و الميزات ، و لكن تخيل لو طبّقناها على مصفوفة من الكائنات بالطبع سيكون الأمرُ أكثرَ إثارةً و أكثرَ نفعاً.

الفصل التاسع عشر:

القوالب (Templates)

ملاحظة: القوالب من الميزات الحديثة التي تمّ إنتاجها مع لغة ANSI-C++ القياسيّة. لذلك إذا كنت تستعمل مترجماً لا يدعم هذه اللغة القياسيّة فمن الممكن ألّا تكون قادراً على استخدامها.

تابع القوالب

تسمح القوالب بإنشاء توابع عموميّة تستطيع أن تقبل أي نوع من البيانات كوسطاء وتعيد القيمة المطلوبة من دون إعادة تحميل التابع أبداً. وهذه التوابع العمومية تخضع للقاعدة التالية:

```
template < typename  identifier > function_declaration ;
```

typename : هو نوع البيانات الممرّرة و قد يكون نوعاً أساسيّاً أو نوعاً معرّفاً (مثل الصفوف).

identifier : اسم الوسيط.

مثال: هذا التابع يعيد أكبر كائن من الكائنين الممررين إليه (من النوع(الصفّ) GenericType).

```
template < class GenericType >
GenericType GetMax (GenericType a, GenericType b) {
 return ( a > b ? a : b ) ;
}
```

كما يبيّن السطر الأوّل فقد أنشأنا قالباً لنوع (صفّ) عامّ من البيانات دعوناه GenericType. لذلك فالتابع الذي تلا التصريح الأوّل سيأخذ النوع المذكور GenericType كما استُخدِم هذا النوعُ لتمرير وسيطين منه إلى التابع GetMax() و الذي سيعيد أكبرهما.

حتى الآن لم يمثل النوع المعرف GenericType أي نوع منفصل من البيانات، ولكن عندما سيستدعى التابع سنكون قادرين على استبدال هذا النوع بأي نوع آخر والذي سندعوه نموذجاً pattern. للقيام بذلك يمكننا أن نستدعي التابع العموميّ بالطريقة التالية:

```
function <pattern> (parameters) ;
```

pattern : نوع البيانات الذي سيتم استبدال النوع GenericType به.

فمثلاً: لاستدعاء هذا التابع ليقارن بين عددين صحيحين نكتب:

```
int  x,y ;
GetMax <int> (x,y) ;
```

والآن سيُستدعى التابع GetMax() وكأنّه أصلاً من النوع int. لنكتب المثال بشكله النهائي:

```
// function template
#include <iostream>
using namespace std;
template <class S>
S GetMax (S a, S b) {
  S result;
  return ( result = (a>b)? a : b );
}
int main () {
  int i=5, j=6, k;
  float l=10.0, m=5.0, n;
  k=GetMax<int>(i,j);
  n=GetMax<float>(l,m);
  cout << k << "\n" << n << endl;
  return 0;
}
```

```
6
10
```

في هذه الحالة دعونا النوع العام بالاسم S بدلاً من الاسم GenericType كما ويمكننا أن نستخدم أي اسم آخر علماً أنّ الاسم T هو الأكثر شهرةً مع القوالب (لأنّه أوّل حرف من الكلمة template).

تستطيع أن ترى كيف استخدمنا التابع GetMax() مع نوعين مختلفين من البيانات (int , float) مع أنّنا لم نقم بإعادة تحميل التابع ليستقبل أكثر من نوع.

تستطيع أن ترى أيضاً أنّنا تمكنّا من تعريف متغيّر من النوع S داخل التابع GetMax()

S result;

سيتمّ استبدال المتغيّر من النوع **S** بآخر من النوع المذكور بين القوسين <> عند استدعاء التابع GetMax().

ملاحظة: في هذه الحالة الوسطاء الممرّرة إلى التابع هي من النوع **S** وهو نفس نوع البيانات التي سيعيدها التابع لذلك فإنّ المترجم يستطيع أن يعرف نوع البيانات من خلال الوسطاء الممرّرة مباشرة دون أن نذكره بين القوسين <>. كما يلي:

```
int i, j ;
GetMax ( i , j );
```

```
// function template
#include <iostream>
using namespace std;
template <class S>
S GetMax (S a, S b) {
  S result;
  return (result = ((a>b)? a : b));
}
int main () {
  int i=5, j=6, k;
  float l=10.0, m=5.0, n;
  k=GetMax(i,j);
  n=GetMax(l,m);
  cout << k << "\n" << n << endl;
  return 0;
}
```

```
6
10
```

بينما لا يمكننا كتابة التعليمات التالية بسبب الاختلاف بين الأنواع الممرّرة إلى التابع :

```
int i;
long l;
k = GetMax (i,l);
```

حيث أنّ التابع ينتظر منّا أن نمرر إليه متغيّرين (كائنين) من نفس النوع (الصفّ).

يمكننا أن نصنع تابعاً عموميّاً يقبل أكثر من نوع من البيانات كما في المثال التالي :

```
template <class T, class U>
T  GetMin (T a, U b) {
  return ( a < b ? a : b ) ;
}
```

في هذه الحالة التابع GetMin() يقبل نوعين مختلفين من الوسطاء ويعيد كائناً من نوع الوسيط الأول (T).

نستطيع استدعاء هذا التابع بأحد الشكلين التاليين :

```
int  i, j ;
long  l;
```

1- الشكل العام :

```
i = GetMin<int,long> (j,l);
```

2- الشكل المختصر :

```
i = GetMin (j,l);
```

صفّ القوالب

يمكننا أن نكتب صفّاً ليقبل أعضاءً من أنواع عامّة، و كمثال على ذلك:

```
template <class T>
class pair {
   T values [2];
 public:
   pair (T first, T second)
   {
      values[0]=first;
      values[1]=second;
   }
};
```

هذا الصفّ يستطيع أن يخزّن عنصرين من أي نوع مقبول، فلو أردنا أن ننشأ كائناً لنخزّن قيمتين صحيحتين مثل 115 و 36 فعندها سنكتب:

```
pair<int> myobject (115, 36);
```

و باستخدام نفس الصفّ السابق يمكننا أن نخزّن عنصرين من أي نوع آخر كالآتي:

```
pair<float> myfloats (3.0, 2.18);
```

```
// class templates
#include <iostream>
using namespace std;
template <class T>
class mypair {
T a, b;
public:
mypair (T first, T second)
{a=first; b=second;}
T getmax ();
};
template <class T>
T mypair<T>::getmax ()
{
T retval;
retval = a>b? a : b;
return retval;
}
int main () {
mypair <int> myobject (100, 75);
cout << myobject.getmax()<<endl;
return 0;
}
```

```
100
```

لا حظ كيف سبقنا سطر تضمين (كتابة تعليمات) التابع getmax() بالتعليمة template <class T>:

```
template <class T>
T pair<T>::getmax () {...}
```

وهذا لأنّنا استخدمنا التصريح المبدئي prototype فوجب علينا أن نسبقه بالسابقة template<...> أمّا الأعضاء المصرّح عنها كلّيا داخل الصفّ inline فلا داعي من ذكر هذه السابقة أمامها.

تخصيص القالب Template specialization

تسمح لنا هذه الخاصيّة بعمل مجموعة من الأشياء مع نوع معيّن دون الأنواع الأخرى من البيانات، لنفترض أننا أردنا أن نكتب المثال السابق ليحتوي على تابعٍ عضوٍ يعيد باقي قسمة كائنين، لكن لن يقوم بهذه العمليّة إلّا إذا كان الكائنان من النوع الصحيح وإلّا سيعيد 0. نستطيع فعل ذلك كما يلي:

```
// Template specialization
#include <iostream>
using namespace std;
template <class T>
class mypair {
T a, b;
public:
mypair (T first, T second)
{a=first; b=second;}
T module () {return 0;}
};
template <>
int mypair<int>::module() {
  return a%b;
}
```

```
25
0
```

```
int main () {
  mypair <int> myints (100,75);
  mypair <float> myfloats (100.0,75.0);
  cout << myints.module() << '\n';
  cout << myfloats.module() << '\n';
  return 0;
}
```

كما ترى فإنّ شيفرة التخصيص من الشكل:

template <> class class_name <type>

يعتبر التخصيص جزءاً من القالب لذلك علينا أن نبدأ التصريح بالكلمة template <> ولا يمكننا أن نضع النوع العام في أول قوسين من الشكل <> ويجب أن نبقيهما فارغين وبعد اسم الصفّ علينا أن نضع النوع الذي نرغب بتخصيصه بين قوسين من نفس النوع <>.

بعض الأشكال الممكنة للقوالب :

```
template <class T>              // The most usual: one class parameter.
template <class T, class U>     // Two class parameters.
template <class T, int N>       // A class and an integer.
template <class T = char>       // With a default value.
template <int Tfunc (int)>      // A function as parameter.
```

فضاءات الأسماء (Namespaces)

تمكّننا فضاءات الأسماء من تجميع عدّة صفوف عامّة في مجموعة واحدة، من الكائنات و(أو) التوابع تحت اسم واحد لكلّ هذه المجموعة حيث تصبح كلُّها أعضاءً داخل المدى الذي يشكّل فضاء الاسم، ويكون كلّ عضو منها قادراً على الوصول إلى الأعضاء الأخرى.
الشكل الذي يمكننا من خلاله التصريح عن فضاء اسم جديد هو :

```
namespace  identifier
{
 namespace-body
}
```

حيث أنّ :
identifier: هو أي اسم تختاره أنت شريطة أن يكون صحيحاً (يخضع لقواعد تسمية المتغيّرات).
namespace-body: مجموعة من الصفوف و(أو) الكائنات و(أو) التوابع.
مثال :

```
namespace general
{
 int a, b;
 class S;
 void B(){…};
}
```

و لكي نصل إلى أحد أعضاء فضاء الاسم general من خارج الفضاء علينا أن نستخدم معامل المدى **::**

مثال:

```
general::a;
general::B();
```

الوظيفة التي يشغلها فضاء الاسم مفيدة خصوصاً في الحالات التي يُحتَمل فيها وجود أعضاء أو كائنات شاملة تمتلك نفس الاسم داخل نفس البرنامج ما يعني أنّ علينا أن نعيد تسميتها لكي نستطيع أن نتعامل معها كلّها إلّا أنّ فضاءات الأسماء تقدّم حلّاً مناسباً لهذه المشكلة. راقب المثال التالي :

```
// namespaces
#include <iostream>
using namespace std;
namespace first {
  int var = 5;
}
namespace second {
  double var = 3.1416;
}
int main () {
  cout << first::var << endl;
  cout << second::var << endl;
  return 0;
}
```

```
5
3.1416
```

في هذه الحالة لدينا متغيّران شاملان لهما نفس الاسم var لكنّ الأوّل موجودٌ في الفضاء first أمّا الثاني فهو موجودٌ في الفضاء second لذلك لن يكون من الخطأ وجود متغيّرين لهما نفس الاسم كما كان سيحصل لو أزلنا الفضائين السابقين وأبقينا على المتغيّرين فقط.

التعليمة using namespace

تأخذ هذه التعليمة الشكل التالي :

using namespace *identifier*;

مثال :

```
// using namespace example
#include <iostream>
using namespace std;
namespace first
{
  int var = 5;
}
namespace second
{
  double var = 3.1416;
}
int main () {
  using namespace second;
  cout << var << endl;
  cout << (var*2) << endl;
  return 0;
}
```

```
3.1416
6.2832
```

تستطيع أن ترى كيف أنّنا لم نقم بكتابة اسم الفضاء الذي ينتمي إليه المتغيّر var في المرّتين، وذلك لأنّ التعليمة using تقوم بتوجيه المترجم كي يستخدم الأعضاء الموجودِين في فضاء الاسم المذكور بعدها وبالتالي فإنّ المترجم يعلم الآن أنّك تقصد المتغيّر var الموجود في الفضاء second.

ملاحظة: إنّ التعليمة using namespace تخضع لقاعدة المدى المعروفة من قبل.
حيث أنّ مدى هذه التعليمة هو الأقواس {} التي كتبت فيها. أمّا إن كُتِبَتْ في مكان التصريح عن المتغيّرات الشاملة فإنّها ستكون ذات مدى شامل لكلّ البرنامج.

مثال :

```
// using namespace example
#include <iostream>
using namespace std;
namespace first
{ int var = 5; }
namespace second
{ double var = 3.1416; }
int main () {
  {
    using namespace first;   // scope of
    cout << var << endl;     // namespace first
  }
  {
    using namespace second;  // scope of
    cout << var << endl;     // namespace second
  }
  return 0; }
```

```
5
3.1416
```

الفضاء std

واحدٌ من أفضل الأمثلة حول فضاءات الأسماء، وهو مكتبة C++ القياسية نفسها. حيث تمّ تضمين كل صفوف و كائنات وتوابع مكتبة C++ القياسية في الفضاء std.

مثال :

```
// ANSI-C++ compliant hello world
#include <iostream>
using namespace std;
int main ()
{
  std::cout << "Hello world in ANSI-C++\n";
  return 0;
}
```

```
Hello world in ANSI-C++
```

ملاحظة: يجب أن تستخدم اسم الفضاء ثمّ معامل المدى لكي تتمكن من التعامل مع أعضاء هذا الفضاء.
إلّا أنّه يمكننا أن نستخدم التعليمة ; using namespace std لنتمكن من الاستغناء عن ذكر اسم الفضاء و معامل المدى من أجل كل مرّة سنتعامل فيها مع أعضاء هذا الفضاء كما يلي :

```
// ANSI-C++ compliant hello world (II)
#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello world in ANSI-C++\n";
  return 0;
}
```

```
Hello world in ANSI-C++
```

جرّب أن تزيل التعليمة ; using namespace std ثمّ شغّل البرنامج. **ماذا تجد؟ ما هو السبب؟**

حلول التمارين غير المحلولة

بنية البرنامج في لغة الـ C++

1.

- main()
- بفاصلة منقوطة (;)
- تعليق.
- التعليق هو قطعة من الشيفرة مهملة من قبل المترجم.
- إنهاء البرنامج بطريقة سليمة.
- القوسين {}
- تضمين الملف المصدري <iostream> لاستخدام التعليمات الموجودة فيه.

2.

```
cout<<"In the name of Allah. The peace upon you, Hello world!";
```

المتغيّرات

1. أجب عن الأسئلة التالية :
   - لا ، لأنّ C++ حسّاسة لحالة الأحرف.
   - نعم. لأن التصريح يوجب ذكر نوع المتغيّر.
   - لا.
   - نعم.
   - signed يعني أنّ المتغيّر ذو إشارة سالبة أو موجبة ، بينما unsigned فالمتغيّر لا يمكن أن يكون سالباً.
2. املأ الفراغات التالية :
   - أماكن ، المؤقتة ، اسم ، يميزه ، الذاكرة.
3.
   a) خاطئة لأن المتغيّر f لا يقبل إشارة سالبة.
   b) خاطئة لأن اسم المتغيّر يجب ألا يبدأ برقم.
   c) خاطئة المتغيّر d من النوع العشري وليس محرفاً.
   d) صحيحة.
   e) خاطئة اسم المتغيّر يجب ألا يحوي نقطة أو رموز خاصة.
   f) صحيحة.
4. معظم هذه الأخطاء منطقيّة لذلك قد يعمل البرنامج مع وجود بعضها:

- نوع المتغيّرين a, b عشري وتم اسنادهما لمتغيّر من لنوع الصحيح لذلك يجب تحويل نوع المتغيّر result إلى عشري (سيتجاهل الأرقام العشرية بعد الفاصلة ويخزن القسم الصحيح فقط).
- المتغيّر result لا يقبل أرقاماً سالبة في حين أن القيمة المسندة إليه سالبة فيجب تعديله ليكون signed.
- **التعليمة result = A + b; غير صحيحة لأنه لا يوجد متغيّر اسمه A**
  فيجب أن نعدلها لتصبح result = a + b ;.
- التابع main() من النوع int لذلك يجب أن يعيد قيمة من النوع الصحيح إلا أنه لا يمتك التعليمة return 0; فيجب أن نضيفها إليه (لا يسبب خطأ إلا أنه يعطي تحذيراً فقط).

.1

```
*
* *
* * *
* * * *
* * * * *
```

.2

```
#include <iostream>
using namespace std;
int main ()
{
  float f = 0.0 , c = 0.0 ;
  cout<<"Enter celecius degree: ";
  cin>>c;
  f = (9/5)*c + 32 ;
  cout<<'\n'<<f<<"\n";
  return 0;
}
```

.3

```
#include <iostream>
using namespace std;
int main ()
{
  float f = 0.0;
  cout<<"Enter fehrnhite degree: ";
  cin>>f;
  float c = 5*(f - 32)/9 ;
  cout<<'\n'<< c <<"\n";
  return 0;
}
```

.4

```
#include <iostream>
using namespace std;
int main ()
{
  float a = 0.0 , d = 0.0,v =0.0 ;
  int t = 0;
  cout<<"Enter accelration of body by m.s^-2: ";
  cin>>a;
  cout<<"Enter time of movement by seconds: ";
  cin>> t;
  d = 0.5*a*t*t;
```

```
  v = a*t;
  cout<<"distance = "<<d<<" meters\n";
  cout<<"velocity = "<<v<<" m.s^-1\n";
  return 0;
}
```

.5

```
#include <iostream>
using namespace std;
int main ()
{
  float f = 0.0 , x =0.0;
  cout<<"Enter x: ";
  cin>>x;
  f = 5 - x*x;
  cout<<"\nf("<<x<<") = "<<f<<endl;
  return 0;
}
```

.6

```
#include <iostream>
using namespace std;
int main ()
{
  int n = 0;
  cout<<"Enter n: ";
  cin>>n;
  cout<<"\nsum of numbers from (1 to "<<n<<") = "<<n*(n+1)/2<<endl;
  cout<<"\navg = "<<(float)(n*(n+1)/2)/n<<endl;//avg = sum / n
  return 0;
}
```

.7

```
#include <iostream>
using namespace std;
int main ()
{
 cout<<"Just remember \"Allah created You\'.\n";
 return 0;
}
```

.8

```
#include <iostream>
using namespace std;
int main ()
{
        int x1,x2,x3;
        cout<<"Enter three numbers plz: ";
        cin>>x1>>x2>>x3;
   cout<<"\nAverage of ["<<x1<<','<<x2<<','<<x3<<"] = "
                << float(x1+x2+x3)/3<<endl;
 return 0;
}
```

.1

int x = 0, a = 0, b = 1, d = 3, c = 2;

bool e = true , r = true, t = false;

العبارة	الجواب
a += (b = c = d - 5) ;	-2
d %= 2 * c / ++b ;	3
a = d - (b += (c * (1-a)));	0
e = !e	0
r = e || t	0
t = e && r	0
e = !((e && r) || (e && t)) ;	1

.2

العبارة	الجواب
x = 7 + 3 * 6 / 2 - 1 ;	15
x = 2 % 2 + 2 * 2 -2 / 2 ;	3
x = (3 * 9 * (3 + (9 * 3 / (3)))) ;	324
c = 2*b - 3/d + a++;	1

.3

cout<< ++a ;	1
cout<< a++ ;	0
cout<< a++ / 2 ;	0
cout<< ++a / 2 ;	0

بنى التحكم

.1

```
#include<iostream >
using namespace std;
float avg =0;
int main()
{
   cout<<"Enter Average: ";
   cin>>avg;
   if(avg > 89 && avg <= 100 )
      cout<<"Excellent!\n";
   else if(avg > 79 && avg <= 89)
            cout<<"Very Good\n";
   else if(avg > 69 && avg <= 79)
            cout<<"Good\n";
   else if(avg > 59 && avg <= 69)
            cout<<"Passed\n";
   else
       cout<<"Not passed\n";
    return 0;
}
```

.2

البرنامج	التصحيح

`while ( c <= 5 )`
`{`
`product *= c;`
`++c;`
	`while ( c <= 5 )`
	`{ product *= c;`
	`++c;`
	`}`

`if ( gender == 1 )`
`cout<< "Woman" ;`
`else;`
`cout<<"man";`
	`if ( gender == 1 )`
	`cout<< "Woman" ;`
	`else // لا يوجد فاصلة منقوطة هنا`
	`cout<<"man";`

`while ( z >= 0 )`
`sum += z;`
	`while ( z >= 0 )`
	`{`
	`sum += z;`
	`z--;`
	`}`

`// nested loop`
`#include <iostream>`
`using namespace std;`
`int main ()`
`{`
`int j = 1;`
`for( int i = 1 ; i<=3 ;`
`i++)`
`while( j<i )`
`{`
`cout <<"* " ;`
`j++;`
`}`
`return 0;`
`}`
	`// nested loop`
	`#include <iostream>`
	`using namespace std;`
	`int main ()`
	`{`
	`int j = 1;`
	`for( int i = 1 ; i<=5 ; i++)`
	`{while( j<=i )`
	`{cout <<"* " ;`
	`j++;`
	`}`
	`cout<<endl;`
	`j = 1;``}`
	`return 0;`
	`}`

.3

```
#include <iostream>
using namespace std;
int main ()
{
  int x,y,z,max,med,min;
  cout<<"Enter three numbers: ";
  cin>>x>>y>>z;
  if (x > y && x > z && y > z)
      cout<<z<<"\t"<<y<<"\t"<<x<<endl;
   else if (x > y && x > z &&  z>y )
            cout<<y<<"\t"<<z<<"\t"<<x<<endl;

   if (y > x && y > z && x > z)
       cout<<z<<"\t"<<x<<"\t"<<y<<endl;
   else if (y > x && y > z &&  z>x )
             cout<<x<<"\t"<<z<<"\t"<<y<<endl;

    if (z > y && z > x && y > x)
        cout<<x<<"\t"<<y<<"\t"<<z<<endl;
     else if (z > y && z > x &&  x>y )
               cout<<y<<"\t"<<x<<"\t"<<z<<endl;
return 0;
}
```

.4

البرنامج	الشكل
#include <iostream> using namespace std; int main () { for(int i = 3;i<=6;i++) { for(int j = i;j<i+4;j++) cout<<j<<" "; cout<<"\n"; } return 0; }	3 4 5 6 4 5 6 7 5 6 7 8 6 7 8 9

#include <iostream>
using namespace std;
int main ()
{ for(int i = 0;i<=3;i++)
{
for(int j = 0;j<=3;j++)
if(i>=j) cout<<j+3+i<<" ";
cout<<"\n";
}
return 0;
}
	3
	4 5
	5 6 7
	6 7 8 9

```
#include <iostream>
using namespace std;
int main ()
{
for(int i = 0;i<=3;i++)
{
for(int j = 0;j<=3;j++)
 if(i<=j)
    cout<<j+3+i<<" ";
   else cout<<"   "; // 2 spaces
 cout<<"\n";
}
return 0;
}
```

```
3 4 5 6
  5 6 7
    7 8
      9
```

```
#include <iostream>
using namespace std;
int main ()
{
 for(int i = 0;i<=3;i++)
 {
   for(int j = 0;j<=3;j++)
     if(i+j>=4-1)
        cout<<j+3+i<<" ";
     else cout<<"   ";//2 spaces
   cout<<"\n";
}
return 0;
}
```

```
      6
    6 7
  6 7 8
6 7 8 9
```

```
#include <iostream>
using namespace std;
int main ()
{
for(int i = 0;i<=3;i++)
{
          for(int j = 0;j<=3;j++)
               if(i<=j)  cout<<j+3<<" ";
          cout<<"\n";
}
return 0;
}
```

```
3 4 5 6
4 5 6
5 6
6
```

```
#include <iostream>
using namespace std;
int main ()
{
for(int i = 0;i<=3;i++)
{
for(int j = 0;j<=3;j++)
if(i == 0 || i == 3)
cout<<j+3+i<<"  ";//2 spaces
else
{
```

```
3 4 5 6
4     7
5     8
6 7 8 9
```

```
            if(j == 0 || j == 3)
cout<<j+3+i<<" ";
else cout<<"  ";// 2 spaces
}
cout<<"\n";
}
return 0;
}
```

.5

```
#include <iostream>
using namespace std;
int main ()
{
cout<<"Enter n: ";
int n;
cin>>n;
if(n%10 == 0)
cout << "yes\n";
else
cout<<"no\n";
return 0;
}
```

.6

```
#include <iostream>
#include<cmath >
using namespace std;
int main ()
{
double x, fx =0.0 ;
err:
cout<<"Enter x ecept (1 or 3): ";
cin>>x;
if(x == 1 || x == 3)
{cout << "Infinite Error You can't Enter 1 or 3\n";
goto err;}
else{
fx = (pow(x,3) - 7)/(pow(x,2)-4*x+3);
cout<<"f("<<x<<") = "<<fx<<endl;
}
return 0;
}
```

.7

```
#include <iostream>
using namespace std;
int main()
{
	int i,sum=0;

	for(i=1;i<=99;i++)
		if(i%2==0 && i%3==0)
			sum+=i;

	cout<<"sum="<<sum<<endl;
	return 0;
}
```

.8

```
#include<iostream>
using namespace std;
void main()
{
int a,b,lcm ;
cout << "enter a and b" << endl;
cin>>a>>b;
if (a == 0 || b == 0) lcm = 0;
else if ( a>b && a%b==0 )
cout <<"lcm="<<a<<endl ;
else if(b>a && b%a==0)
cout <<"lcm="<<b<<endl;
else
lcm=a*b;

cout <<"lcm="<<lcm<<endl;
}
```

.9

```
#include<iostream >
using namespace std;
void main()
{
  double x,p=1.0,y;
   cout <<"Enter x, y: ";
   cin >> x >> y;
   if(y>=0){
      for(int i = 1; i<=y;i++)
           p *=x;
      cout<< p << endl;
    }
}
```

```
#include<iostream >
using namespace std;
void main()
{  double x,p=1.0,y;
   cout<<"Enter x, y: ";
    cin>>x>>y;
    if(y>=0){
      int i = 1;
      while( i<=y )
       {   p *=x;
           i++;
       }
        cout<< p  << endl;
     }
}
```

.10

```
#include<iostream >
using namespace std;
void main()
{          int n;  char c;
           cout<<"Enter charactor then number: ";
           cin >> c >> n;
           for(int i = 0; i<n;i++){
                      for(int j = 0; j<n;j++)
                         if(i>=j) cout<<c<<" ";
                      cout<<endl;}
}
```

.11

```
#include<iostream.h>
void main()
{
float x,y;
char c = 'n';
while(c == 'n' )
{
cout<<"Enter two numbers: ";
cin>> x >> y;
ZeroErr:
if(y != 0)
{
cout<< x/y << endl;
}
else
{
cout<<"Error You can't enter 0";
cout<<"ReEnter second number without 0 plz: ";
cin>> y;
goto ZeroErr;
}
inputErr:
cout<<"press n for new operation,(e to exit): ";
cin>> c;
if(c == 'n' || c == 'N') continue;
else if(c == 'e' || c == 'E') break;
else goto inputErr;
}
}
```

.12

```
#include<iostream >
using namespace std;
void main()
```

```
{
        int x;
        cout<<"Enter x: ";
        cin>>x;
        int i = 1;
        while(i<=x)
        {
                if(x%i == 0)cout<<i<<endl;
                i++;
        }
}
```

النوابع

.1

التابع	الأخطاء

`int Sum(int a , int b)`
`{`
`  int sum = a + b;`
`}`
	هذ التابع من النوع int ويجب أن يعيد قيمة إلا أنه لا يحتوي على التعليمة **return statement;**
	return sum; // return a+b;

`int divide(int a , int b)`
`{`
`return (float)a/b;`
`}`
	هذا التابع من النوع int و هو يعيد قيمة من النوع float فعلينا أن نغير نوع التابع إلى float.
	float divide(int a, int b){…}

`void subtract(int a , int b)`
`{`
`return a-b;`
`}`
	هذ التابع من النوع void ويجب ألّا يعيد قيمة لكنه يحتوي على التعليمة **return a-b;** لذلك سنغير نوعه إلى int
	int subtract(int a, int b){…}

.2

البرنامج	الناتج

`#include<iostream>`
`using namespace std;`
`int x = 0;`
`float b;`
`float d(float a , float& b)`
`{`
`a--;`
`b++;`
`x = a/b;`
`return (x+1)/2.0;`
`}`
`int main()`
`{`
`float a;`
`cout<<"x\tb\ta\n";`
`cout<<x<<"\t"<<b<<"\t"<<a<<endl;`
`a = b = ++x;`
`cout<<d(a , b)<<endl;`
`cout<<x<<"\t"<<b<<"\t"<<a<<endl;`
`return 0;`
`}`
	x b a
	0 0 -1.07374e+008
	0.5
	0 2 1
	أمّا بعد التعديل :
	x b a
	0 0 -1.07374e+008
	0.5
	1 2 1
	
	ملاحظة: القيمة -1.07374e+008 هي قيمة عشوائية تختلف دوماً وهي عشوائية لأن المتغيّر a متغيّر موضعي ولقد قلنا إنّ المتغيّرات الموضعية تأخذ قيماً عشوائية إذا لم تتمّ تبدئتها أمّا المتغيّرات الشاملة فهي تأخذ القيم الصفرية للنوع

.3

```
#include<iostream>
using namespace std;
float sum(float a,float b);
float sub(float a,float b);
float mult(float a,float b);
float divide(float a,float b);
void main()
{
float x,y;
char op ;
cout<<"Enter x: ";
cin>>x;
cout<<"Enter operator from{+,-,*,/}: ";
cin>>op;
cout<<"Enter y: ";
cin>>y;
switch(op)
{
case '+': cout<<sum(x,y)<<endl; break;
case '-': cout<<sub(x,y)<<endl; break;
case '*': cout<<mult(x,y)<<endl; break;
case '/':
        if(y!=0) cout<<divide(x,y);
        else cout<<"Error divide by 0\n";
        break;
}
}
float sum(float a,float b)
{
        return a+b;
}
float sub(float a,float b)
{
        return a-b;
}
float mult(float a,float b)
{
        return a*b;
}
float divide(float a,float b)
{
        return a/b;
}
```

.4

```
#include<iostream >
using namespace std;
float rec_power(float a,int b);
float frec_power(float a,int b);
```

```
void main()
{
	float x;
	int y;
	cout<<"Enter x, y: ";
	cin>>x>>y;
	cout<<"Recutrivity power = "<<rec_power(x,y)<<endl;
	cout<<"Frecuently power = "<<frec_power(x,y)<<endl;
}
float rec_power(float a,int b)
{
	if(b == 0)
		return 1;
	else if(b>0)
		return a*rec_power(a,b-1);
	else
		return 1/(a*rec_power(a,-b-1));
}
float frec_power(float a,int b)
{
	float res = 1;
	if(b==0)
	  res = 1;
	else if(b>0)
		for(int i=1;i<=b;i++)
			res*=a;
	else{
			for(int i=1;i<=-b;i++)
				res*=a;
			res = 1/res;
	}
return res;
}
```

.5

```
#include<iostream >
using namespace std;
float frec_power(float a,int b);
float pi(long n);
void main()
{
	long n ;
	cout<< "enter n: ";
	cin>>n;
	cout<<"\n"<<pi(n)<<endl;
}
float pi(long n)
{
	float p = 0;
	for(long i = 0;i<n;i++)
```

```
        p += 4*frec_power(-1,i)/(2*i+1);
        return p;
}
float frec_power(float a,int b)
{
        float res = 1;
        if(b==0)
          res = 1;
        else if(b>0)
        for(int i=1;i<=b;i++)
                res*=a;
        else
        {
                for(int i=1;i<=-b;i++)
                        res*=a;
                res = 1/res;
        }
return res;
}
```

.6

```
#include<iostream >
using namespace std;
int frec_gcd(int a, int b); // frecuently
int rec_gcd(int x, int y);  // recursivity
void main()
{
        int a , b;
        cout<<"Enter a , b: ";
        cin>>a>>b;
        cout<<"\nfrequently result: "<< frec_gcd(a,b);
        cout<<"\nrecursivity result: "<< rec_gcd(a,b)<< endl;
}
int frec_gcd(int a, int b)
{
        while(a != b)
        {
        if(a>b)
                a-=b;
        else
                b-=a;
        }
        return a; // or return b : a = b
}
int rec_gcd(int x, int y)
{
        if(x == y)
                        return x;
                else if (x>y)
                        return rec_gcd(y,x-y);
```

```
                else
                        return rec_gcd(x,y-x);
        //second way to finding gcd
        /*if(y == 0)
                return x;
        else
                return rec_gcd(y,x%y); */
}
```

.7

```
#include<iostream >
#include<math.h>
using namespace std;
long fact(int a);
double Compination(int n, int k);
double Permitation(int n, int r);
void main()
{
        int n , k , r ;
        cout<<"Enter n: ";
        cin>>n;
        cout<<"\n"<<n<<"! = "<<fact(n);
        cout<<"\nEnter k < " << n << " : ";
        cin>>k;
        cout<<"\nc("<<n<<","<<k<<") = "<<Compination(n,k);
        cout<<"\nEnter r < " << n << " : ";
        cin>>r;
        cout<<"\np("<<n<<","<<r<<") = "<<Permitation(n,r)<<endl;
}
long fact(int a)
{
        long f = 1;
        for(int i =1; i<=a ; i++)
                f*= i;
        return f;
}
double Compination(int n, int k)
{
        double c = fact(n)/(fact(n-k)*fact(k));
        return c;
}
double Permitation(int n, int r)
{
        double p =Compination(n,r)*fact(r);
        return p;
```

```
}
```

.8

```
#include<iostream >
using namespace std;
void multiple(int a, int b);
bool multiple1(int a, int b);
void main()
{
	int n , k  ;
	cout<<"Enter n, k: ";
	cin>>n>>k;
	if(multiple1(n,k))
		cout<<"true\n";
	else
		cout<<"false\n";
	multiple(n,k);
}
void multiple(int a, int b)
{
	if(b>a && b%a ==0)
		cout<<"true\n";
	else
		cout<<"false\n";
}
bool multiple1(int a, int b)
{	if(b>a && b%a ==0)
		return true;
	return false; // no need to else statement. why?
}
```

.9

```
#include<iostream >
using namespace std;
void print_primes(int a);
bool is_prime(int a);
void main()
{
	int n;
	cout<<"Enter n: ";
	cin>>n;
	if(is_prime(n))
		cout<<"true\n";
	else
		cout<<"false\n";
	print_primes(n);

}
bool is_prime(int a)
{
```

```
        if(a>0 && a != 1)//positive and not 1 (1 isn't prime)
        {
         int sum = 0;
         for(int i = 1; i <= a ; i++)
         {
                 if(a%i == 0)
                         sum++;
         }
         if(sum == 2)
                 return true;
        }
        return false;
}
void print_primes(int a)
{
        for(int i = 1; i <= a ; i++)
                if(is_prime(i))
                        cout<<i<<"\n";
}
```

.10

```
#include<iostream>
using namespace std;
double abs_val(double x)
{
if(x>0) return x;
else return -x;
}
void main()
{
double n;
cout<<"Enter n: ";
cin>>n;
cout<<"| "<<n<<" | = "<<abs_val(n)<<endl;
}
```

.11

```
#include<iostream>
using namespace std;
const double pi = 3.141592;
double Circl_space(double r)
{ return (pi*r*r); }
double Rect_space(double a , double b)
{ return (a*b); }
double Trgl_space(double b , double h)
{ return (0.5*b*h); }
void main()
{
        cout<<"1- Calculate circle spsace\n";
```

```
        cout<<"2- Calculate rectangle spsace\n";
        cout<<"3- Calculate triangle spsace";
        cout<<"\nEnter number from list above to select: ";
        int ch ;
        cin>>ch;
        cout<<"\n";
        switch(ch)
        {
        case 1 :
                cout<<"Calculating circle space.. .\n";
                double r ;
                cout<<"Enter r: ";
                cin>>r;
                cout<<"S ="<<Circl_space(r)<<endl;  break;
        case 2 :
                cout<<"Calculating rectangle space.. .\n";
                double a, b ;
                cout<<"Enter width: ";
                cin>>a;
                cout<<"Enter height: ";
                cin>>b;
                cout<<"S ="<<Rect_space(a,b)<<endl;
                break;
        case 3 :
                cout<<"Calculating triangle space.. .\n";
                double h;
                cout<<"Enter base: ";
                cin>>b;
                cout<<"Enter height: ";
                cin>>h;
                cout<<"S ="<< Trgl_space(b,h)<<endl;
                break;
        default:
                cout<<"program will terminate now!.";
                break;
        }
```

.12

```
#include<iostream>
using namespace std;
int print(int h);
int main()
{ int h;//h is hieght
 cin>>h;
 print(h);
 return 0;  }
int print(int h)
{ int i,j,k;
  for(i=1;i<=h;i++)//number of rows
  {
    for(j=1;j<=h-i;j++)//from line beginning to the first star pos print space
       cout<<"  ";// 2 spaces
    for(k=1;k<=2*i-1 ;k++)// from last space in line to (2*line number -1)
       cout<<"* ";            // print * where (2*line number -1) is number of
                              // * in each line (i)
    cout<<"\n";              // print new line
  }
 return 0;
}
```

المصفوفات

-1

(a) for (int i = 0; i<= 10; i++)

سيأخذ العداد القيمة 10 والعنصر b[10] غير موجود إذاً يجب أن يكون شرط التوقف i<10

أو يمكن أن نعطي i القيمة 1 أي int i = 1

(b) cout<<a[1]<<" "<<a[2]<<" "<<a[3]<<endl;

العنصر a[3] غير موجود في المصفوفة a لأن عناصرها هي a[0] و a[1] و a[2].

(c) double f[3] = { 1.1, 10.01, 100.001, 1000.0001 };

المصفوفة f تتسع لـ 3 عناصر فقط لذلك يجب حذف أحد العناصر على الأقل

(d) d[1, 9] = 2.345;

يجب أن يكون كل دليل ضمن قوسين أي d[1][9] = 2.345;

-2

(a) p[0] و p[1] و p[2] و p[3].

(b) تعريف.

(c) d[2][4].

(d) 20.

-3

(a) const int arraySize = 10;

(b) double fractions[arraySize] = { 0.0 };

(c) fractions[9] = 1.667; fractions[6] = 3.333;

(d)

```
for(int i=0; i<arraySize;i++)
 cout<<"fractions["<i<<"]="<<fractions[i]<<endl;
```

-4

```
#include<iostream>
using namespace std;
const int n=8;
void print(int a[])
{
	for(int i=0;i<n;i++)
		cout<<a[i]<<" ";
}
int main()
{
	int A[n]={1,5,4,3,7,2,9,0},i,j,temp;
	for(i=0;i<n;i++)//ascending
		for(j=0;j<n-1;j++)
			if(A[j]>A[j+1])
			{temp=A[j];A[j]=A[j+1];A[j+1]=temp;}
	cout<<"ascending:  ";
	print(A);
```

```
       for(i=0;i<n;i++)//descending
              for(j=0;j<n-1;j++)
                     if(A[j]<A[j+1])
                     {temp=A[j];A[j]=A[j+1];A[j+1]=temp;}
       cout<<"\ndescending: ";
        print(A);
       cout<<endl;
       return 0;
}
```

-5

```
#include<iostream>
using namespace std;
const int n=10;
int a[n],b[n],c[2*n],i,j;

void read(int arr[])
{
       for(i=0;i<n;i++)
              cin>>arr[i];
}
void merge(int a[],int b[])
{
       for(i=0;i<n;i++)
       {
              c[i]=a[i];
              c[i+n]=b[i];
       }
}
void print(int arr[])
{
       for(int i=0;i<2*n;i++)
              cout<<a[i]<<" ";
}
int main()
{
       cout<<"enter A members:\n";
       read(a);
       cout<<"enter B members:\n";
       read(b);
       merge(a,b);
    cout<<"C=[ ";
       print(c);
       cout<<"]\n";
       return 0;
}
```

-6

```
#include<iostream>
using namespace std;
const int n=10;
int main()
{
       int a[n],frq[4]={0},i;
       cout<<"enter array of 10 integers:\n";
       for(i=0;i<n;i++)
```

```
		cin>>a[i];
	for(i=0;i<n-1;i++)
	{
		if(a[i]<a[i+1])
			frq[0]++;
		else if(a[i]>a[i+1])
			frq[1]++;
		else if(a[i]==a[i+1])
			frq[2]++;
		else
			frq[3]++;
	}

	if(frq[0]==9)
		cout<<"the array is growing.\n";
	else if(frq[1]==9)
		cout<<"the array is decreasing.\n";
	else if(frq[2]==9)
		cout<<"the array is constant.\n";
	else
		cout<<"the array is growing and decreasing.\n";
	return 0;
}
```

-7

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
	int n,mark[100],i,j,temp;
	string name[100],strTemp;
	cout<<"enter students number ";
	cin>>n;
	for(i=0;i<n;i++)
	{
		cout<<"student "<<i+1<<" name:";
		cin>>name[i];
		cout<<"student "<<i+1<<" mark:";
		cin>>mark[i];
	}
	for(i=0;i<n;i++)
		for(j=0;j<n-1;j++)
			if(mark[j]<mark[j+1])
			{

	temp=mark[j];mark[j]=mark[j+1];mark[j+1]=temp;

	strTemp=name[j];name[j]=name[j+1];name[j+1]=strTemp;
			}

	cout<<"students ranks by marks:\n";
	for(i=0;i<n;i++)
		cout<<name[i]<<endl;
	return 0;
}
```

-8

```
// program to add matrix to another one
//using functions
#include <iostream>
using namespace std;
#define MAX 10
int a[MAX][MAX],b[MAX][MAX],c[MAX][MAX];

void add(int [][MAX],int [][MAX],int [][MAX],int ,int );
void print(int [][MAX],int,int);

int main()
{
       int m,n;
       cout<<"number of rows=";cin>>m;
       cout<<"number of columns=";cin>>n;
       add(a,b,c,m,n);
       print(a,m,n);
       cout<<"\n  +\n\n";
       print(b,m,n);
       cout<<"\n  =\n\n";
       print(c,m,n);
       return 0;
}
void print(int arr[][MAX],int m,int n)
{
       int i,j;
   for(i=0;i<m;i++)
       {
              for(j=0;j<n;j++)
               cout<<arr[i][j]<<"  ";
        cout<<"\n";
    }

}
void add(int a[][MAX],int b[][MAX],int c[][MAX],int m,int n)
{
       int i,j;
  for(i=0;i<m;i++)
       for(j=0;j<n;j++)
       {
          cout<<"a["<<i+1<<"]["<<j+1<<"]=";cin>>a[i][j];
          cout<<"b["<<i+1<<"]["<<j+1<<"]=";cin>>b[i][j];
          c[i][j]=a[i][j]+b[i][j];
       }
}
```

-9

```
#include<iostream>
#include<string>
#include<cstdlib>
#include<ctime>
using namespace std;
```

```
int main()
{
          cout<<"enter your Choice :\n";
          cout<<"R\tfor Rock\n";
          cout<<"P\tfor Paper\n";
          cout<<"S\tfor Scissors\n";
          cout<<"The computer chose\n";
  int cmpCounter = 0;
  int YourCounter = 0;
 lbb:
  cout<<"\nYour points:\t\t" << YourCounter;
         cout<<"\nComputer points:\t" << cmpCounter;
  string mesg ;
  srand(time(0));//seed rand
  int r = rand() % 3 ;
       switch (r)
       {
          case 0:
             mesg = "Computer choice's Rock";
             break;
          case 1:
             mesg = "Computer choice's Paper";
             break;
          case 2:
             mesg = "Computer choice's Scissors";
             break;
       }

       cout<<"\nChose please:";
       string PlayerChosen ;
                          cin>>PlayerChosen ;
       if (PlayerChosen == "R" || PlayerChosen == "r")
       {
          cout<<mesg;
          switch (r)
          {
             case 0:
                cout<<"\nNo Winner";
                cmpCounter = cmpCounter;
                YourCounter = YourCounter;
                goto lbb;
                break;
             case 1:
                cout<<"\nComputer Won...";
                cmpCounter+=1;
                goto lbb;
                break;
             case 2:
                cout<<"\nYou Won...";
                YourCounter+=1;
                goto lbb;
```

```
            break;
        }
    }
    else if (PlayerChosen == "P" || PlayerChosen == "p")
    {
        cout<<mesg;
        switch (r)
        {
            case 0:
                cout<<"\nYou Won...";
                YourCounter+=1;
                goto lbb;
                break;
            case 1:
                cout<<"\nNo Winner";
                cmpCounter = cmpCounter;
                YourCounter = YourCounter;
                goto lbb;
                break;
            case 2:
                cout<<"\nComputer Won...";
                cmpCounter+=1;
                goto lbb;
                break;
        }
    }
    else if (PlayerChosen == "S" || PlayerChosen == "s")
    {
        cout<<mesg;
        switch (r)
        {
            case 0:
                cout<<"\nComputer Won...";
                cmpCounter+=1;
                goto lbb;
                break;
            case 1:
                cout<<"\nYou Winner";
                YourCounter+=1;
                goto lbb;
                break;
            case 2:
                cout<<"\nNo Winner...";
                cmpCounter = cmpCounter;
                YourCounter = YourCounter;
                goto lbb;
                break;
        }
    }
    else if (PlayerChosen == "E" || PlayerChosen == "e")
    {
```

```
            cout<<"Do you want to exit the game? (y/n)";
            string anser;
                                        cin>>anser;
            if (anser == "Y" || anser == "y")
               return 0;//exit main func
            else
               goto lbb;
         }
         else
         {
            cout<<"\nBe sure from your inserting...";
            goto lbb;
         }
return 0;
}
```

.1

```
#include<iostream>
using namespace std;
const double pi = 3.141592;
class Circle
{
private :
	double radius;
public:
	void set_radius(double r);
	double calc_space();
	double calc_perimeter();
};
void Circle::set_radius(double r)
{
	radius = r;
}
double Circle::calc_space()
{
	return pi*radius*radius;
}
double Circle::calc_perimeter()
{
	return 2*pi*radius;
}
void main()
{
	Circle c1;
	double r;
	cout<<"Enter radius: ";
	cin>>r;
	c1.set_radius(r);
	cout<<"c1 Space = "<<c1.calc_space()<<endl;
	cout<<"c1 perimeter = "<<c1.calc_perimeter()<<endl;
}
```

.2

```
#include<iostream>
#include<math.h>
using namespace std;
class Point
{
public:
	double x, y;
	void set_point(double p_x, double p_y);
	double distance(Point p1, Point p2);
};
```

```
void Point::set_point(double p_x, double p_y)
{
        x = p_x;  y = p_y;
}
double Point::distance(Point p1, Point p2)
{
        return( sqrt( pow( (p2.x-p1.x),2) + (pow((p2.y-p1.y),2) ) ) );
}
void main()
{
        Point p1,p2;
        double x1,y1,x2,y2;
        cout<<"Enter x1, y1: ";
        cin>>x1>>y1;
        cout<<"Enter x2, y2: ";
        cin>>x2>>y2;
        p1.set_point(x1,y1);
        p2.set_point(x2,y2);
        cout<<"Distance between p1 and p2 is :"
            <<p1.distance(p1,p2)<<endl;
}
```

.3

```
#include<iostream>
using namespace std;
const double pi = 3.141592;
class Circle
{
private :
        double radius;
public:
  Circle(){}
        Circle(double r){radius =r;}
        void set_radius(double r);
        double calc_space();
        double calc_perimeter();
};
void Circle::set_radius(double r)
{
        radius = r;
}
double Circle::calc_space()
{
        return pi*radius*radius;
}
double Circle::calc_perimeter()
{
        return 2*pi*radius;
}
void main()
```

```
{
        Circle c1;
        double r;
        cout<<"Enter radius: ";
        cin>>r;
        Circle c2(r);
        c1.set_radius(r);
        cout<<"c1 Space\t"<<"c2 Space\t"<<"c1 perimeter\t"<<"c2
perimeter\n";
        cout<<c1.calc_space()<<"\t\t"<<c2.calc_space()

        <<"\t\t"<<c1.calc_perimeter()<<"\t\t"<<c2.calc_perimeter()<<endl;
}
```

4. **العبارة Circle circle; ليست صحيحة لأنّنا لم نضف المشيد العادي.**
5.

```
#include<iostream>
#include<math.h>
using namespace std;
class Point
{
public:
        double x, y;
Point(){}
Point(double p_x, double p_y) { x = p_x ; y = p_y ; }
void set_point(double p_x, double p_y);
double distance(Point p);
};
void Point::set_point(double p_x, double p_y)
{
        x = p_x;
        y = p_y;
}
double Point::distance(Point p)
{ return sqrt( pow( (p.x-this->x),2) + (pow((p.y-this->y),2) ) ); }
int main()
{
        Point p1,p2;
        p1.set_point(1,2);
        p2.set_point(2,1);
        cout<<p1.distance(p2)<<endl;
        return 0;
}
```

.6

```
#include<iostream.h>
class Printing
{public:
  Printing(){
  cout<<
  "Give the world the best you have, and the best will come to you.\n"; }
          ~Printing(){
          cout<<
    "\nIf man hasn't discovered that he will die for, he isn't fit to live.\n"; }
}print_that;
int main()
{
  cout<<"Sweat plus sacrifice equals success.";
  return 0;
}
```

ملاحظة: لا تغيّر التعليمة #include<iostream.h> (لا تستخدم using namespace std;) قد لا يعمل البرنامج بشكل كامل (قد لا يطبع العبارة الثالثة الموجودة في الهادم)

.7

```
#include<iostream>
#include<math.h>
using namespace std;
class Point
{
public:
          double x, y;
          void set_point(double p_x, double p_y);
          double distance(Point p1, Point p2);
          Point operator+ (Point p);
};
void Point::set_point(double p_x, double p_y)
{
          x = p_x;
          y = p_y;
}
double Point::distance(Point p1, Point p2)
{
          return sqrt( pow( (p2.x-p1.x),2) + (pow((p2.y-p1.y),2) ) );
}
Point Point::operator+(Point p)
{
          Point temp;
          temp.x = this->x + p.x;
          temp.y = this->y + p.y;
          return temp;
}
void main()
{
          Point p1,p2;
```

```
        double x1,y1,x2,y2;
        cout<<"Enter x1, y1: ";
        cin>>x1>>y1;
        cout<<"Enter x2, y2: ";
        cin>>x2>>y2;
        p1.set_point(x1,y1);
        p2.set_point(x2,y2);
        cout<<"Distance between p1 and p2 is :
"<<p1.distance(p1,p2)<<endl;
        cout<<"c(x,y) = ("<<(p1+p2).x<<", "<<(p1+p2).y<<")\n";
}
```

.8

- أزل الكلمة const من أمام التصريح عن التابع **.**getIncrementedData
- التابع getcount من النوع static لذلك عليك أن تزيل التعليمة الخاصة بالطباعة منه.

الخاتمة

أخيراً لا يحوي هذا الكتاب على كلّ ما عليك أن تتعلّمه في البرمجة، وليس هذا هو آخر الكتب التي أُلِّفت في هذا المجال؛ لذلك إن كنت ترى أنّ البرمجة هي الشيء الذي كنت تبحث عنه فعليك أن تبحث عن كتاب آخر لتكمل معه رحلتك.

اللّهمّ اجعله في ميزان حسناتنا ولا تحرِمنا أجره.

اللّهمّ من كان سبباً في إنجاز هذا الكتاب فحرِّمه على النار وارزقه الجنّة برحمتك يا أرحم الراحمين.

اللّهمّ ما كان من صوابٍ فمنك وحدك لا شريك لك، وما كان من زللٍ أو نسيانٍ فمن أنفسِنا والشيطان.

وصلّى الله على سيدّنا محمّد وعلى آله وصحبه أجمعين.

وآخر دعوانا أن الحمد لله ربّ العالمين.

***

يرجى زيارة الرابط التالي لإبداء رأيكم بالكتاب

https://docs.google.com/spreadsheet/viewform?formkey=dDVxYmV5UFgwaHBFZkQ0U3RiOTNNSUE6MQ&ifq

المراجع

1- The cplusplus.com tutorial (http://www.cplusplus.com/doc/)
Auther: Juan Soulie
2- C++ How to Program
Fifth Edition
3- The C+ + Programming Language
Third Edition, Auther: Bjarne Stroustrup(the creator of c++)
4- Java How to Program
5- خطوة بخطوة مع فيجوال ستوديو C# And VB.net 2008
أحمد جمال خليفة