الجامعة الافتراضية السورية

SYRIAN VIRTUAL UNIVERSITY

برمجة 2

الدكتورة غيداء الربداوي

Books

البرمجة 2

الدكتورة غيداء ربداوي

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

Computer Programming 2

Ghaidaa Ribdawi

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

الفهرس

البرمجة 2

يهدف هذا المقرر إلى التعريف بالبرمجة الغرضية التوجه وتقاناتها الأساسية، ومساعدة الطالب للوصول به إلى كتابة خوارزميات بلغة غرضية التوجه، وسنستخدم لغة C++ كأداة لهذا الغرض.

يعتمد هذا المقرر على المعارف الأساسية الواردة في مقرر البرمجة1.

المفردات الأساسية

1- مدخل إلى مفهوم الغرض

2- مبادئ البرمجة الغرضية التوجه

3- أسس لغة C++

4- بناء الصفوف

5- البواني والهوادم

6- المؤشرات والحجز الديناميكي للذاكرة

7- تركيب الأغراض

8- زيادة تحميل المعاملات

9- الوراثة

10- تعدد الأشكال

11- مفاهيم متقدمة في C++

مدخل

تعلمت في مقرر البرمجة1 أن عالم البرمجة عالم الأرقام والأحرف وهو يتعامل معها وفق مجموعة من التعليمات الأساسية التي لا تخرج عن خمسة أنواع من التعليمات:

1- القراءة 2- الكتابة 3- الإسناد 4- الاختيار 5- التكرار

وبالطبع تترجم هذه التعليمات إلى العديد من الأشكال في لغات البرمجة المختلفة وتشتق منها تعليمات أخرى تساعد على كتابة أفضل للرماز البرمجي.

ولكن أتعتقد أن عالم البرمجة يعج بالأرقام والأحرف فقط؟

لو طلب منك أن تكتب برنامجاً لإدارة المرور لتخزين المعلومات حول السيارات التي يملكها المواطنون، فماذا ستفعل؟ كيف تخزن معلوماتك عن "سيارة"؟

يجب أن تسأل نفسك أولاً: ما هي المعلومات التي تهمني في "سيارة" في نظام إدارة المرور؟ وستقرر أنك تهتم باسم مالك السيارة، نوع السيارة، نموذج السيارة، عام الصنع، اللون، رقم اللوحة،.....) فهل تعرف متحولاً منفرداً لكلٍ من هذه المعلومات؟

وكيف تربطها معاً بحيث تستحضرها جميعاً لكل سيارة في برنامجك؟

لو أنك تستطيع أن تعرف متحولاً واحداً يحتوي جميع المعلومات الخاصة بكل سيارة لتحقق لك ما تريد.

ألا تتمنى عندما تكتب برنامجاً يتعامل مع السيارات في إدارة المرور لو أنك تستطيع أن تعرف متحولاً من النمط "سيارة" على غرار تعريف متحول من النمط عدد صحيح؟

```
int i;
car c;
```

ألا تتمنى عندما تكتب برنامجاً يرسم الدوائر والخطوط لو أنك تستطيع أن تعرف متحولات من النمط "دائرة" و"خط" على غرار تعريف متحول من النمط عدد صحيح؟

```
int i;
Circle c;
Line l;
```

ولكن كيف يمكن للغة برمجة أن تحتوي على جميع الأنماط التي يمكن أن تخطر في بال المبرمجين في العالم في كل زمان ومكان، وأن يكون تعريف النمط مطابقاً لحاجة المبرمج في مسألة ما؟

لابد أنك تدرك أن هذه المهمة مستحيلة، ولكن لا توجد مشكلة دون حل.

زودت بعض لغات البرمجة بآلية تسمح للمبرمج أن يعرف نمطاً جديداً لم يسبق له وجود في اللغة بحيث يصف عند التعريف خصائص هذا النمط الجديد والمعلومات التي يرغب بالاحتفاظ بها عنه، ويعطي المبرمج لهذا النمط الجديد اسماً يختاره، فيصبح لديه في هذا البرنامج بالذات نمطاً جديداً يستخدمه على نحو شبيه بالأنماط مسبقة التعريف في اللغة.

وهكذا يمكن للمبرمج تعريف أنماط عدة في برامجه دون أن تحد اللغة من إمكاناته، ويستخدمها استخداماً مشابهاً للأنماط العددية والمحرفية التي يعرفها، فلا تقف اللغة عائقاً أمام إبداعه وحاجاته.

ولعل أشهر آلية زودت بها لغات البرمجة هي آلية الصفوف والأغراض، والتي فتحت الباب على مصراعيه أمام منحى جديد من البرمجة سمي "البرمجة الغرضية التوجه" وهي ماسنتناوله بالتفصيل في هذا المقرر.

مقدمة في تقانة الأغراض

هدف الفصل

يهدف هذا الفصل إلى تحصيل المعارف التالية:

- تعرف مبادئ تقنيات البرمجة الغرضية التوجه التي تمكنك من التعامل مع معطيات العالم الحقيقي.
- فهم مفهوم الغرض.
- تكوين حدس لديك فيما يتعلق بتحديد الأغراض واستخراجها وتوصيفها عند بناء الحلول البرمجية.
- تعرف أسس التصميم الغرضي التوجه.

عناصر حل المسائل في العالم الحقيقي

تأمل الحياة من حولك وآليات العمل التي يستخدمها الناس لإنتاج أشياء مفيدة ولحل المسائل المطروحة، تجد أن صنع أي منتج وحل أي مسألة يبنى على استخدام المكونات الموجودة في الواقع والتي يؤدي تركيبها بطريقة محددة إلى الحصول على منتج جديد.

لنأخذ مثلاً صناعة السيارات. تطرح على المهندس مسألة وضع تصميم لسيارة جديدة.

لا يعقل أن ينطلق المهندس لوضع تصميم سيارته من المواد الخام اللازمة لصنع جميع القطع والمكونات في السيارة، أي من الحديد والزجاج والمطاط وغيرها من المواد الأولية، لأنه سيكون عليه في هذه الحالة أن يصنع القطع الأساسية من المواد الأولية، ثم يصنع السيارة من هذه القطع. وسيرتب عليه هذا الوضع أن يحل عدة مسائل تتعلق كل منها بكيفية صنع قطعة ما من المواد الأولية المؤلفة لها. فيضع مثلاً تصميماً للأبواب وطريقة صنعها انطلاقاً من الخلائط المعدنية اللازمة، و تصميماً لأطر العجلات وطريقة صنعها انطلاقاً من المطاط، وتصميماً للمحرك وطريقة صنعه انطلاقاً من المعادن المستخدمة، .. وهكذا.

ونلاحظ هنا أن كل مسألة جزئية هي بحد ذاتها مسألة تحتاج خبيراً في المجال، ولا يمكن لمهندس في مجال تصميم السيارات الإحاطة به.

في الواقع، ينطلق المهندس لحل المسألة من مجموعة من المكونات (محرك، أطر عجلات، أبواب، نوافذ، علبة سرعة، إلخ...). تتواجد هذه المكونات في السوق بأنواع عديدة، وعلى المهندس تحديد نوع كل مكون، ووصفه وصفاً دقيقاً لتمييزه عن المكونات التي تشبهه، وتحديد الطريقة التي سيرتبط بها بالمكونات الأخرى التي تشترك معه لبناء السيارة، وتعريف الخطوات اللازمة لتجميع المكونات الواحد تلو الآخر لبناء كامل التصميم المطلوب.

نسمي المكونات والعناصر الجاهزة أغراضاً.

ولكن كيف نحدد الأغراض التي سنستخدمها لحل المسألة؟

لنعد إلى مثالنا عن تصميم نموذج جديد من السيارات. لكي يتمكن المهندس من تحديد نوع كل مكون، ومواصفاته، لابد من أن يكون مطلعاً على الأنواع المعروضة ومواصفاتها، واعتماداً على المواصفات التي يسعى إلى تحقيقها في السيارة التي سيقوم بتصميمها، سيقوم باختيار الأنواع المناسبة. تسمى هذه العملية التي تسبق التصميم "التحليل الأولي للمسألة". تلي هذه المرحلة مرحلة تصميم الحل.

الأغراض وحل المسائل

ذكرنا أن المكونات التي تستعمل لحل المسألة تسمى أغراضاً. فما الذي يهمنا معرفته من الأغراض؟

نهتم بالطبع بمعرفة خصائص الغرض أو ما نسميه واصفات الغرض **attributes** التي تصف ماهية الغرض، وبالأفعال **actions** التي يقوم بها.

مثلاً الغرض سيارة يمكن تعريفه بعدد من المعلومات وهي: النموذج، اللون، الاستطاعة، الموضع، وبالأفعال التي تقوم بها وهي السير إلى الأمام، الرجوع إلى الخلف، الالتفاف يميناً، الالتفاف يساراً.

الغرض : سيارة

الواصفات:
النموذج، اللون، الاستطاعة، الموضع

الأفعال:
السير إلى الأمام، الرجوع إلى الخلف،
الالتفاف يميناً، الالتفاف يساراً.

بعد تحديد الأغراض، ننهي مرحلة التحليل بتعيين فاعل **agent** يقوم بتنظيم التفاعل بين الأغراض لإنجاز المهمة المطلوبة. وفي مثالنا آنف الذكر يقوم سائق السيارة بتحريكها إلى الأمام والخلف وذات اليمين وذات الشمال.

تنظر تقانة الأغراض إلى حل المسائل من منظور الأغراض. يعين التحليل الأولي الأغراض وهي العناصر التي ستستخدم في عملية حل المسألة، ثم يحدد التحليل النهائي الخطوات الواجب اتباعها والتي تتيح لفاعل محدد أن ينسق أفعال الأغراض.

تقويم:

1. الدراسة ليلاً:

أنت تجلس في غرفتك في وقت متأخر ليلاً وأمامك مهمة قراءة الفصل الأول من كتاب البرمجة واستخلاص أهم الأفكار فيه.

- ما هي الأغراض التي نحتاجها لإنجاز هذه المهمة؟

الإجابة:

- كتاب
- ورقة
- قلم
- ضوء

- من هو الفاعل؟

الإجابة:

أنت الفاعل الذي يضيء الضوء ويفتح الكتاب ويمسك بالقلم ليضع الأفكار على الورقة.

2. جهاز التحكم عن بعد بالتلفاز:

يستخدم جهاز التحكم عن بعد بالتلفاز لإراحة المتفرج.

- ما هي الأغراض التي يتألف منها الجهاز ؟

الإجابة:

- لوحة أرقام
- زر التحكم بالصوت

- زر التشغيل/الإيقاف

- من هو الفاعل؟

الإجابة:

الفاعل هو المتفرج الذي يشغل جهاز التلفاز ويختار القناة ويضبط ارتفاع الصوت.

حل المسائل حاسوبياً

في العالم الحقيقي يتدخل الفاعل بنفسه ليتعامل مع الأغراض. أما في الحاسوب، فتختلف الإجرائية **process** المتبعة لتناسب طبيعة الآلة.

لقد جرى تصميم إجرائيات الحاسوب لتتعامل مع المعلومات بحيث ترد المعطيات إلى الذاكرة، فتخضع للمعالجة والحساب ثم تصدر النتائج كمعطيات خرج.

وعند استخدام الحاسوب في حل المسائل يجب التفكير بمنطق الأغراض كذلك، حيث تكون المعطيات هي واصفات الأغراض، وتكون العمليات التي تنفذ إلى هذه المعطيات وتعالجها هي الأفعال، ثم يبنى الحل البرمجي الذي يحدد طريقة تفاعل هذه الأغراض مع بعضها والخطوات التي يجري هذا التفاعل وفقها.

نمذجة أغراض العالم الحقيقي

لكي نمكّن الحاسوب من حل المسائل، يجب أن نضع في برامجه نماذج تمثّل أغراض المسألة، وبهذا نكون قد بنينا أغراضاً حاسوبية تمثّل نماذج مجردة لأغراض حقيقية. تحتفظ هذه الأغراض المجردة بالمعطيات التي تتعلق بالمسألة.

لنأخذ مثلاً سجلات الطلاب في جامعة. في عالم الواقع، يعتبر الطالب كائناً معقّداً يتميز بمجموعة من الصفات الشخصية (العمر، الجنس، لون الشعر،...)، وبمجموعة من الصفات الديموغرافية (العنوان المحلي، مكان الولادة)، وبمعلومات دراسية (الصف، المعدل)، وهكذا....

وعندما يقدم الطالب للتسجيل في الجامعة، فإنه يقابل المسؤول عن التسجيل والمحاسب والمسؤول عن السكن الجامعي وغيرهم لإنجاز عدد من الأعمال التي تتعلق بتسجيله ودفع الرسوم وحجز مكان في السكن.

ويقوم كل موظف من موظفي الجامعة آنفي الذكر بالتعامل مع جزء من المعطيات التي تمثّل الطالب، ويعالج مسألة مختلفة حسب اختصاصه، أي أن كل موظف على حده يحتاج إلى معلومات

مختلفة عن الطالب وكأن كل موظف يحتاج إلى نموذج للطالب يمثّله من وجهة النظر التي تهم هذا الموظف حسب اختصاصه.

فمثلاً، لا يهتم المحاسب بعمر الطالب وجنسه ولون شعره وصفه، وإنما يهمه معرفة رقم تسجيله ووضع ذمته المالية تجاه الجامعة. أما المعلومات المتعلقة بالعمر والجنس فهي تهم المسؤول عن السكن الجامعي، في حين يهتم كل من المسؤول عن التسجيل والامتحانات بصف الطالب ومعدله. وننوه هنا إلى أن الطالب يعطى رقم تسجيل يسمح بتحديده في جميع الأنظمة.

نلاحظ إذاً أن كل مسؤول عن نمط من الأعمال يهتم بجزء من المعلومات المميزة للكائن "طالب"، وكأننا نحتاج إلى عدة نماذج للطالب: نموذج الطالب في نظام المحاسبة، ونموذج الطالب في نظام السكن الجامعي، و نموذج الطالب في نظام الامتحانات، وسنرى لاحقاً أن كل نموذج يمثل غرضاً في النظام الذي سيتعامل معه.

الطالب: (الغرض طالب في نظام المحاسبة)

الواصفات: رقم التسجيل، الاسم، الذمة المالية

الطالب: (الغرض طالب في نظام الامتحانات)

الواصفات: رقم التسجيل، الاسم، الاختصاص، المعدل

الطالب: (الغرض طالب في نظام السكن الجامعي)

الواصفات: رقم التسجيل، الاسم، الجنس، العمر، رقم الغرفة

Grand mother Dr.Jako

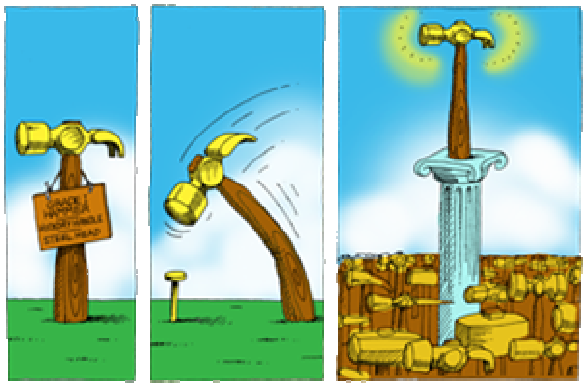
مثال

ليكن لدينا طالب محدد. نصنع له غرضاً لمعالجته في نظام السكن الجامعي. هذا الغرض هو:

وكما رأينا سابقاً فإن الأغراض لا تنمذج الواصفات وحسب بل تنمذج كذلك العمليات التي تصف التفاعلات بين الطالب والموظفين على اختلاف أنواعهم.

ففي نظام السكن الجامعي مثلاً، يجب أن يتضمن النظام عمليات تسمح بالحصول على جنس طالب أو عمره من خلال رقم تسجيله، كما يجب أن يتضمن النظام عمليات لتحديد رقم الغرفة الموافق له، أو تبديل الغرفة. وعليه تكون العمليات التي تطبق على الغرض طالب في نظام السكن الجامعي هي:

الحصول على العمر- الحصول على الجنس- الحصول على رقم التسجيل- الحصول على الاسم- تحديد غرفة- تبديل غرفة



attribute behavior identity

تدريبات:

- نريد بناء نموذج للسيارة في نظام لإدارة المرور. حدد خصائص الغرض سيارة الموافق للمسألة (واصفات وعمليات)
- نريد بناء نموذج للموظف في نظام إداري في مؤسسة. حدد خصائص الغرض موظف الموافق للمسألة (واصفات وعمليات)
- نريد بناء نموذج لحساب مصرفي يخص المتعامل في مصرف. حدد خصائص الغرض حساب مصرفي الموافق للمسألة (واصفات وعمليات)

إذاً عملية النمذجة هي الآلية الذهنية التي يقوم بها الإنسان ليستخلص الصفات والخصائص التي تميز غرضاً ما في الواقع الحقيقي، والتي تعتبر ذات علاقة بالمسألة التي يطلب حلها، لنصنع من هذه الخصائص غرضاً برمجياً يمكننا التعامل معه حاسوبياً.

كيف نصف الأغراض في البرامج الحاسوبية

ذكرنا أن عملية حل المسائل تستخدم الأغراض. وإذا عدنا إلى نظام السكن الجامعي، فإن النظام يهتم بإيجاد غرفة لكل طالب. يمثل كل طالب داخل النظام بغرض يتألف من رقم تسجيل الطالب، اسمه، جنسه، عمره، رقم غرفته، ويطبق عليه المجموعة نفسها من العمليات وهي:

- الحصول على العمر
- الحصول على الجنس
- الحصول على رقم التسجيل
- الحصول على الاسم
- تحديد غرفة
- تبديل غرفة

وكلما جاء طالب جديد فسيبنى له غرض يتصف بنفس الخصائص التي يتصف بها كل طالب في النظام. ويعود ذلك إلى أن هذه الخصائص تعرف قالباً أو بنية عامة لكل طالب في نظام السكن الجامعي. يشبه هذا القالب قالب صنع الحلوى الذي يستخدم لصنع قطع من الحلوى متماثلة الشكل ولها نفس شكل القالب. يسمى هذا القالب في التقانة الغرضية التوجه نمط الغرض **object type**، في حين يسمى غرض محدد مستنسخاً **instance** من نمط الغرض.

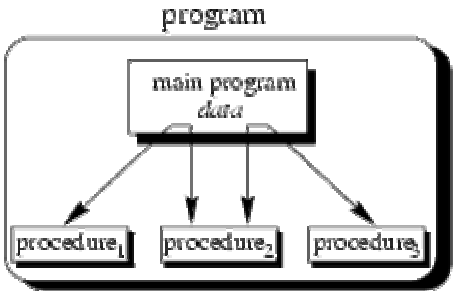
تصميم الحل

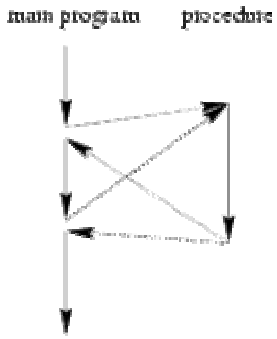
الآن، وقد عرفنا الأغراض، وعلمنا من أين تأتي وفي أية مرحلة، يجب أن نهتم بمرحلة تصميم الحل استناداً إلى الأغراض.

تهدف مرحلة تصميم الحل إلى وضع الخطوات التي يجب أن يجري العمل وفقها لإنجاز المهمة المطلوبة. وبتعبير أدق، تهدف مرحلة تصميم الحل إلى وضع الخوارزمية التي تتوالى وفقها العمليات. هذه العمليات لا توجد منفصلة وحدها بل هي تابعة للأغراض كما رأينا، وعليه، فإن الخوارزمية تصبح تعبيراً عن تفاعل الأغراض بعضها مع بعض، ومن ثم تطبيقاً للعمليات الموجودة في هذه الأغراض وفق مخطط واضح محدد.

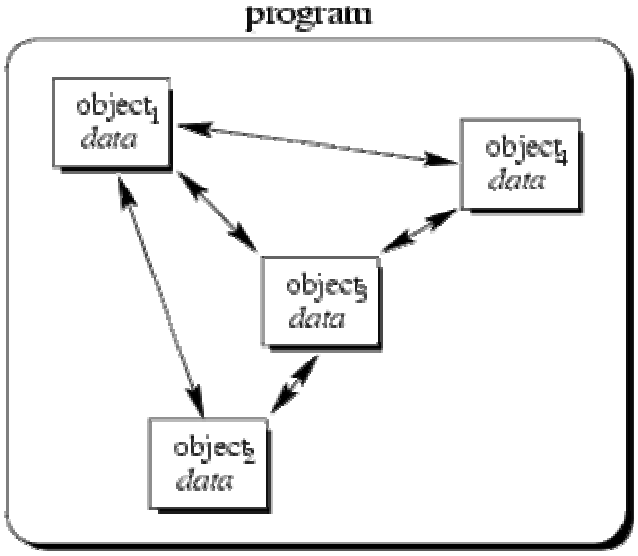
ويمكننا أن نميز الآن بين نوعين من البرمجة:

1. البرمجة الإجرائية: التي تعتمد على كتابة العديد من البرامج الجزئية، بحيث يصبح البرنامج الرئيسي مخطط استدعاء لهذه البرامج الجزئية كما في الشكل:





2. البرمجة الغرضية التوجه: حيث نبني الأغراض التي نحتاجها لحل المسألة، ويصبح دور البرنامج هو تنظيم التفاعل بين هذه الأغراض، كما في الشكل:



سنتناول في الفصول اللاحقة كيفية صنع الأغراض في **C++**.

الفصل الثاني

لغة C++

هدف الفصل

يهدف هذا الفصل إلى تحصيل المعارف التالية:

- لمحة عن لغة **C++** وعلاقتها بلغة **C**
- البنية العامة لبرنامج بلغة **C++**
- البرمجة البنيوية بلغة **C++**
  - الأنماط في **C++**
  - التعليمات في **C++**
  - الأقنية في **C++**

مدخل

++C هي لغة برمجة حديثة، لذا، فهي تعكس معظم الإضافات التي ظهرت في نظريات البرمجة. وهي من ثَم لغة غنية تعطي المبرمج إمكانات هائلة. يربط الكثير من الناس بين البرمجة الغرضية التوجه ولغة **++C**، و هذا خطأ. إذ يمكننا البرمجة بـ **++C** دون أي ذكر للأغراض، على عكس بعض اللغات الغرضية التوجه الصرفة، مثل جافا . وبهذا فإن **++C** لغة هجينة بما لذلك من حسنات وسيئات.

لمحة تاريخية

طور **Bjarne Stroustrup** لغة البرمجة **++C** في مختبرات **AT&T Bell** في بداية الثمانينيات. وكما يقول مبتكر اللغة، فإن **++C** تأخذ الكثير من **C**، إذ احتُفظ بلغة **C** كمجموعة جزئية من **++C** للتسهيلات التي تقدمها في المستويات القريبة من الآلة، وهذا مما يسمح ببرمجة نظم التشغيل. وهكذا، فقد قام **Stroustrup** عام **1979** بتطوير لغة "**C** المزودة بالصفوف". وقد أعيد تشكيل هذه اللغة ما بين **1983/1984** فنتجت لغة **++C** التي خضعت لبعض التحسينات الطفيفة فأصبحت **++C** النسخة **1.2** عام **1987** . انتشرت هذه اللغة انتشاراً واسعاً وأصبحت متوفرة على جميع بيئات التطوير، ثم طورت اللغة من النسخة **1.2** إلى النسخ **2.0** و **2.1** و **3.0**.

في عام **1990** نشر **Stroustrup** مؤلفه (**Annotated C++ Reference Manual " ARM**") أي مرجع لغة **++C** الذي يعتبر الأساس في تقييس اللغة.

ولغة **++C** هي تطور للغة **C** في ثلاثة مسارات أساسية جديدة وهي:

- احتواؤها على تحسينات واضحة مقارنة بلغة **C** التقليدية.
- دعمها لبناء واستكمال أنماط جديدة.
- احتواؤها على تقنيات البرمجة الغرضية التوجه.

ولعل أفضل طريقة لتعلم اللغة هي البدء بكتابة البرنامج الأول بـ **++C**، ثم سنتناول بشيء من التفصيل أهم النقاط التي تخص لغة **++C** معتمدين على أن هذا المقرر ليس المقرر الأول في

البرمجة، وبالتالي فقد سبق لك أن تعرفت المفاهيم الأساسية (متحول، ثابت، تعليمة، ....) في مقرر سابق (راجع مقرر البرمجة1).

البرنامج الأول بلغة C++

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

```
Hello World!
```

يتضمن المستطيل الموجود إلى يسارك البرنامج، في حين يتضمن المستطيل الموجود إلى يمينك نتيجة تنفيذ البرنامج بعدما تمت ترجمته. تختلف الطريقة المستخدمة لترجمة البرنامج باختلاف المترجم المستخدم.

إيضاح:

المترجم هو البرنامج الذي يتحقق من صحة برنامجك قواعدياً بلغة البرمجة التي كتب بها. يوجد عدة مترجمات للغة **C++** ، فهناك المترجم **Microsoft visual C++** وهناك **Visual Studio.Net** وهناك المترجم **GNU C++** الذي يعمل في البيئة يونكس.

إن هذا البرنامج مثال جيد لبرنامج يكتبه طالب مبتدئ بلغة **C++**، وينتج عنه ظهور العبارة "**Hello World!** " على الشاشة، وبالرغم من بساطة البرنامج، فهو يتضمن المكونات الأساسية التي نجدها في جميع البرامج المكتوبة بلغة **C++**. وسنتفحص هذا البرنامج سطراً سطراً.

```
// my first program in C++

#include <iostream>

using namespace std;

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

هذا سطر لكتابة تعليق أو ملاحظة.

جميع الأسطر التي تبدأ بالرمز // هي أسطر ملاحظات، ولا يكون لها أي تأثير على تصرف البرنامج. يستخدم المبرمج أسطر الملاحظات ليضيف شرحاً أو ملاحظة إلى أسطر الرماز في البرنامج.

```
// my first program in C++

#include <iostream>

using namespace std;

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

جميع الأسطر التي تبدأ بالرمز # تتضمن توجيهات للمترجم. وهي لا تعتبر تعليمات بالمعنى المعروف للتعليمة. في برنامجنا هذا، يطلب هذا التوجيه

#include <iostream>

من المترجم أن يضمن الملف القياسي iostream الذي يحتوي على التصريحات الخاصة بمكتبة الدخل والخرج القياسية في C++. وقد جرى تضمينها لأننا سنستخدمها لاحقاً في البرنامج

```
// my first program in C++

#include <iostream>

using namespace std;

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

إن جميع العناصر التابعة للمكتبة القياسية في C++ يصرح عنها فيما يسمى فضاء الأسماء الذي يحمل الاسم std. وعليه، فإن هذه العبارة تدل على أننا سنستخدم هذه العناصر. غالباً ما نرى هذا السطر في برامج C++ التي تستخدم المكتبة القياسية.

```
// my first program in C++

#include <iostream>

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

يدل هذا السطر على بدء تعريف التابع الرئيسي (البرنامج). ويعتبر التابع الرئيسي نقطة البدء التي يجري تنفيذ البرنامج اعتباراً منها بقطع النظر عن مكان وجودها في الرماز، فقد توجد توابع أخرى بأسماء أخرى قبل أو بعد التابع الرئيسي، لكن التنفيذ يبدأ من الأسطر الموجودة فيه. وهكذا يجب أن يحتوي كل برنامج C++ على تابع رئيسي.

نلاحظ هنا وجود زوج من الأقواس ( ) بعد كلمة main ، والسبب في ذلك يعود إلى أننا نصرح عن تابع وليس عن متحول، والقوسان يفيدان في عملية التمييز هذه. أحياناً توضع لائحة من الموسطات بين القوسين. وفوراً بعد هذين القوسين نجد جسم التابع الرئيسي محصوراً بين زوج من الأقواس الكبيرة { } أي التعليمات التي تنفذ عند تنفيذ هذا التابع.

```
// my first program in C++

#include <iostream>
using namespace std;

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

يمثل هذا السطر تعليمة من تعليمات لغة C++ . والتعليمة هي صيغة بسيطة أو مركبة يؤدي تنفيذها إلى الحصول على ناتج ما. في الواقع، تقوم هذه التعليمة بالعملية الوحيدة في برنامجنا وهي الحصول على إظهار على الشاشة.

تعتبر cout قناة الخرج القياسية في C++، ويعني هذا السطر أننا نضع متتالية من المحارف (في برنامجنا الجملة Hello World!) في قناة الخرج القياسية والتي تكون عادة هي الشاشة. يحتوي الملف iostream الموجود في فضاء الأسماء std على تصريح كلمة cout، ولهذا السبب وجدت الأسطر السابقة.

لاحظ أن التعليمة تنتهي بفاصلة منقوطة، وهنا نشير إلى ضرورة وضع هذه الفاصلة المنقوطة في نهاية كل تعليمة في C++.

```
// my first program in C++

#include <iostream>
using namespace std;

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

تؤدي تعليمة return إلى انتهاء البرنامج ، وقد نجد بعد هذه الكلمة رمازاً. في مثالنا، لدينا 0 وهو يفسر بأنه قد تم تنفيذ البرنامج دون أخطاء. وغالباً ما نجد هذا السطر بهذا الشكل في جميع برامج C++.

لعلك لاحظت في هذا البرنامج أن هناك بعض الأسطر التي لا تقوم بأفعال عند التنفيذ. فهناك أسطر للتعليقات (وهي الأسطر التي تبدأ بـ //)، وأسطر لتوجيه المترجم (وهي الأسطر التي تبدأ بـ #)، وأسطر للتصريح عن التوابع (في حالتنا هذه عن التابع الرئيسي). أما أسطر التعليمات (مثل تعليمة **cout**) فتكون موجودة داخل كتلة التابع الرئيسي بين القوسين الكبيرين { }.

أسلوب كتابة البرنامج

نلاحظ أنه قد تم تنظيم البرنامج السابق في أسطر متتالية بحيث تكون قراءته سهلة وواضحة، ولكن لا يوجد في **C++** قواعد صارمة لفصل التعليمات في عدة أسطر. فمثلاً، بدلاً من كتابة البرنامج على النحو التالي:

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

يمكن أن نكتب:

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

ويرجع ذلك إلى أن التعليمات تنتهي بفاصلة منقوطة. وعليه يمكننا أن نكتب عدة تعليمات في سطر واحد أو نكتب تعليمة واحدة في عدة أسطر.

لنضيف سطراً جديداً إلى البرنامج:

```
// 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
```

في هذا البرنامج كتبنا تعليمتي إظهار. وكما ذكرنا آنفاً فإن كتابة كل تعليمة على سطر تهدف إلى توضيح القراءة فقط.

غير أنه من الممكن أن نكتب البرنامج على النحو التالي دون أن يؤثر ذلك على صحته:

```
int main () { cout << " Hello World! "; cout << " I'm a C++ program "; return 0; }
```

كما أننا يمكن أن نكتبه على النحو التالي إذا اعتقدنا أن ذلك أوضح للقارئ:

```
int main ()
{
  cout <<
    "Hello World!";
  cout
    << "I'm a C++ program";
  return 0;
}
```

لابد من الإشارة إلى أننا نستثني التوجيهات التي تخص المترجم، أي تلك التي تبدأ بالرمز#، من إمكان كتابتها في أكثر من سطر، لأنها ليست تعليمات بل هي توجيهات للمترجم يعالجها بطريقة مختلفة ولا تولد بذاتها أي رماز، كما أنها لا تنتهي بفاصلة منقوطة.

ملاحظة حول التعليقات

توجد طريقتان لكتابة التعليقات:

```
// line comment
/* block comment */
```

الطريقة الأولى تعرف بالتعليقات السطرية **line comment** حيث يتجاهل المترجم جميع الكلمات الواردة بعد الرمز // وحتى نهاية السطر.

الطريقة الثانية تعرف بالتعليقات الكتابة **line comment** حيث يتجاهل المترجم جميع الكلمات الواردة بعد الرمز/* وحتى الظهور الأول للرمز */ ولا يوجد ما يمنع كون هذه التعليقات في عدة أسطر.

ولإيضاح الفكرة سنضيف تعليق كتلي لبرنامجنا الثاني:

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

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

الكلمات المحجوزة في لغة C++

تستخدم لغة **C++** مجموعة من الكلمات المحجوزة **reserved words** نوردها في الجدول التالي:

and	and_eq	asm	auto	bitand
bitor	bool	break	case	catch
char	class	const	const_cast	continue
default	delete	do	double	dynamic_cast
else	enum	explicit	export	extern
false	float	for	friend	goto
if	inline	int	long	mutable
namespace	new	not	not_eq	operator
or	or_eq	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
while	xor	xor_eq		

المتحولات والأنماط في C++

كما رأيت في مقرر سابق للبرمجة (راجع مقرر البرمجة1)، فإن البرامج تتضمن متحولات يجري تداولها والتعامل معها في البرنامج لتخزين المعطيات ونتائج الحسابات والعمليات. وكما في الكثير من لغات البرمجة، فإنه يجب التصريح عن أي متحول قبل استخدامه.

توجد في **C++** أربعة أنماط معرفة في أصل اللغة هي: **int**، **float** ، **bool** ، **char** ، وقد اشتقت منها مجموعة من الأنماط الأخرى .

Type	Description	Size	Domain
char	Signed character/byte. Characters are enclosed in single quotes.	1	-128..127
double	Double precision number	8	ca.10^{-308}..10^{308}
int	Signed integer	4	$-2^{31}..2^{31} - 1$
float	Floating point number	4	Ca. $10^{-38}..10^{38}$
long (int)	Signed long integer	4	$-2^{31}..2^{31} - 1$
long long (int)	Signed very long integer	8	$-2^{63}..2^{63} - 1$
short (int)	Short integer	2	$-2^{15}..2^{15} - 1$
unsigned char	Unsigned character/byte	1	0..255
unsigned (int)	Unsigned integer	4	$0..2^{32} - 1$
unsigned long (int)	Unsigned long integer	4	$0..2^{32} - 1$
unsigned long long (int)	Unsigned very long integer	8	$0..2^{64} - 1$
unsigned short (int)	Unsigned short integer	2	$0..2^{16} - 1$

وهكذا يمكن أن نكتب التصريحات التالية:

```
int i, j, count;
float sum, product;
char ch;
bool passed_exam;
double wave_length;
unsigned char color;
long seconds;
```

ويمكن إعطاء قيمة ابتدائية للمتحول عند التصريح:

```
int i, j, count = 0;
float sum = 0.0, product;
char ch = '7';
bool passed_exam = false;
double wave_length=0.00000879;
unsigned char color=120;
long seconds=54087996;
```

التصريح عن الثوابت

في البرامج، نحتاج أحياناً إلى التصريح عن الثوابت. نصرح عن الثوابت في **C++** على النحو التالي:

```
    const type constant_identifier = value ;
```

مثال

```
const int days_in_year = 365;
const float VatRate = 17.5;
const int days_in_year = 365,
          days_in_leap_year = 366;
```

ولا يسمح بمحاولة أي تعليمة في البرنامج أن تغير قيمة الثابت.

الإدخال والإخراج في C++

تستخدم لغة **C++** قناة الدخل (**input stream**) **cin** للإدخال وقناة الخرج (**output stream**) **cout** للإخراج.

- **الإدخال باستخدام cin:**

مثال

```
cin >> number;
cin >> n1 >> n2;

      n2 و n1          لإدخال قيمتي المتحولين
```

ويستدل على نهاية الإدخال عندما يضغط المستخدم الزر **enter**، وفي حال وجود عدة متحولات يطلب إدخالها، يمكن الفصل بين القيم المدخلة بالفراغ أو **enter**.

```
int count, n;
float value;
cin >> count >> value >> n;
```

قد يدخل المستثمر المعطيات على النحو التالي:

```
     23  -65.1 3
```

أو على النحو التالي:

```
23
-65.1        3
```

تستخدم لغة **C++** قناة الدخل (**input stream**) **cin** للإدخال وقناة الخرج (**output stream**) **cout** للإخراج.

- **الإخراج باستخدام cout:**

مثال

```
cout << count;
cout <<"Hello there" << endl;     لإخراج الجملة الموضوعة بين علامتي اقتباس
```

لاحظ هنا استخدام **endl**. نستخدم **endl** عندما نريد أن ننتقل للسطر جديد بعد تعليمة الإخراج تلك.

تقويم :

ما هو خرج التعليمات التالية على الشاشة إذا كان دخل المستخدم هو **6.51** ثم **3.24**:

```
float length, breadth;
cout << "Enter the length and breadth: ";
cin >> length >> breadth;
cout << endl << "The length is " << length;
cout << endl << "The breadth is " << breadth << endl;
```

الإجابة:

```
The length is 6.51
The breadth is 3.24
```

تدريب:

اكتب برنامجاً يقرأ أربعة قيم يدخلها المستخدم، ثم يطبعها كل واحدة على سطر بين علامتي اقتباس بسيطة(مثلاً ' **7** ').

التعليمات في لغة C++

إيضاح

يفترض هذا المقرر أنك قد تعرضت لمعاني ودلالات أنواع التعليمات المختلفة في مقرر سابق. وهدف هذا المقرر تعريفك شكل التعليمة في لغة **C++** وليس إعادة شرح المفاهيم الأساسية. يمكنك العودة إلى مقرر البرمجة**1** إن احتجت إلى مراجعة المفاهيم.

1. تعليمات الإسناد (راجع مقرر البرمجة**1**)

يستخدم الرمز= للدلالة على الإسناد. والشكل العام لتعليمة الإسناد البسيطة هو:

```
result = expression ;
```

مثال

```
average = (a + b)/2;
```

يمكن في الصيغ الحسابية الموجودة إلى يمين الرمز = أن نجد الرموز التالية:

+ للجمع

- للطرح

* للضرب

/ للقسمة

% باقي القسمة (modulus)

مثال

```
i = 3;
sum = 0.0;
perimeter = 2.0 * (length + breadth);
ratio = (a + b)/(c + d);
```

إن نمط المعاملات المستخدمة داخل الصيغة هام جداً لتحديد النتيجة، وهنا تراعى القواعد التالية:

- إذا كان المعاملان من النمط **int** تكون النتيجة من النمط **int** كذلك.

- إذا كان أحد المعاملين أوكلاهما من النمط **float** تكون النتيجة من النمط **float** كذلك.
- إذا كانت نتيجة الصيغة من النمط **int** وكان المتحول إلى يسار معامل الإسناد من النمط **float**، عندها يجري تحويل القيمة الناتجة من **int** إلى **float** قبل خزنها في المتحول إلى يسار معامل الإسناد.
- إذا كانت نتيجة الصيغة من النمط **float** وكان المتحول إلى يسار معامل الإسناد من النمط **int**، عندها يجري تحويل القيمة الناتجة من **float** إلى **int** قبل خزنها في المتحول إلى يسار معامل الإسناد، وهذا يعني ضياع الجزء العشري الموجود في القيمة الأصلية. ويجري هذا التحويل بقطع الجزء العشري دون إجراء أي عملية تدوير.

حسب القاعدة الأخيرة، فإن الإسناد قد يؤدي إلى فقد الدقة ما لم تراعى الأنماط على نحو دقيق.

تقويم:

ما محتوى **i** بعد تنفيذ التعليمات التالية:

```
int i;
i = 3.5 ;
```

الجواب:

3

تواجهنا المشكلة ذاتها عند استعمال معامل القسمة، حيث تقول القاعدة:

- ناتج قسمة عددين من النمط **int** يكون من النمط **int** كذلك. ويجري القطع عندما يكون حاصل القسمة عدداً عشرياً موجباً. أما في حالة الناتج السالب، فلا تحدد اللغة ما الذي يحدث. ومن هنا يجب عليك الانتباه لئلا تفقد الدقة!

مثال

```
i = 1/7 ;
```

تؤدي إلى وضع القيمة 0 في **i**.

أما معامل % لباقي القسمة فهو يعمل على نحو صحيح عندما يكون العددان موجبان، أما عندما يكون أحدهما سالباً فلا يمكن التكهن بسلوك المترجم.

مثال

```
int i;
i=34 % 10;
cout<<i;     //i=4
```

في حين لا ندري ما قيمة نتيجة التعليمة التالية:

```
int i;
i=10 % -7;
cout<<i;     //i=3 or -4
```

وكل ما يمكننا أن نضمنه أن المترجم يراعي المساواة التالية:

```
i % j = i - (i / j) * j
```

التعليمات في لغة C++، أولويات المعاملات

تراعى الأولويات في حساب الصيغ على النحو التالي:

( ) * / % + =

التعليمات في لغة C++، التحويل بين الأنماط

رأينا في الفقرة السابقة (راجع الفقرة تعليمة الإسناد) أن ناتج قسمة عددين من النمط **int** يكون من النمط **int** كذلك، فكيف نفعل للحصول على الجواب الصحيح؟ يجب أن نجبر المترجم على حساب الناتج كقيمة من النمط **float**. فإذا كان أحد المعاملين ثابتاً، يمكننا تحويله إلى عدد ثابت من نمط **float** بإضافة 0. إلى اليمين.

مثال

```
int i;
float x=1.0/i;
```

أما إذا كان المعاملان متحولين صحيحين، فلابد من تحويل أحدهما على الأقل إلى النمط **float** وذلك باستخدام عملية القسر **cast** التالية:

```
f = float(i)/float(n);
```

هنا نستخدم الكلمة **float** الدالة على النمط كمعامل لإعطاء تمثيل بالفاصلة العائمة للمتحول أو للصيغة الموضوعة بين قوسين.

على غرار هذا المعامل لدينا المعامل **int(x)** الذي يعيد قيمة **x** كقيمة صحيحة و**char(y)** الذي يعيد المحرف الموافق للقيمة **y** في ترميز **ASCII**.

تقويم:

تحويل درجة الحرارة:

في التقويم التالي يطلب منك كتابة برنامج يقرأ عدداً صحيحاً يدخله المستخدم، ويمثل درجة الحرارة درجة بالفهرنهايت، ثم يقوم بتحويلها إلى قيمتها بالسنتيغراد.

الإجابة:

```
// Convert Fahrenheit to Centigrade
// Enters a Fahrenheit value from the user,
// converts it to centigrade and outputs
// the result.

#include <iostream.h>

void main()
{
  const float mult = 5.0/9.0;  // 5/9 returns zero
                               // integer division
  const int sub = 32;
  float fahr, cent;
  cout << "Enter Fahrenheit temperature: ";
  cin >> fahr;
  cent = (fahr - sub) * mult;
  cout << "Centigrade equivalent of " << fahr
       << " is " << cent << endl;
}
```

ملاحظة: تستخدم بعض المترجمات ترويسة المكتبة التي تضمن في الملف بوضع iostream.h بدلاً من iostream

التعليمات في لغة C++، معاملا الزيادة increment والإنقاص decrement

يكثر في جميع البرامج ورود شكلين من التعليمات هما:

```
n = n + 1;
n = n – 1;
```

يوجد في **C++** معاملان هما: ++ للزيادة و-- للإنقاص، بحيث يمكننا أن نكتب :

n++; بدلاً من **n = n + 1;** و

n--; بدلاً من **n = n - 1;**

يسمى هذان المعاملان في هذه الحالة معاملي الزيادة اللحقية **postincrement** والإنقاص اللحقي **postdecrement** لأنهما وردا بعد المتحول.

يوجد كذلك معاملان مماثلان هما معاملا الزيادة السبقية **preincrement** والإنقاص السبقي **predecrement** ويستعملان على النحو التالي:

++n; بدلاً من **n = n + 1** ; و

--n; بدلاً من **n = n - 1** ;

لا فرق بين استعمال أي من معاملي الزيادة السبقي أو اللحقي إذا كان الهدف هو زيادة 1 للمتحول **n**. وكذلك الحال بالنسبة لمعاملي الإنقاص، غير أن لكل منهما أثره الجانبي الذي يميزه عن الآخر. لتوضيح هذه الفكرة لنأخذ المثال التالي:

```
i = n++;
```

ما قيمة **i** بعد التنفيذ؟ هل هي قيمة **n** قبل زيادة **1** أم القيمة الجديدة بعد الزيادة؟

تقول القاعدة أن معاملي الزيادة اللحقية **postincrement** والإنقاص اللحقي **postdecrement** يعطيان قيمتهما قبل زيادة أو إنقاص المتحول، في حين أن معاملي الزيادة السبقية **preincrement** والإنقاص السبقي **predecrement** ينفذان زيادة أو إنقاص المتحول أولاً ثم يعطيان القيمة الناتجة عن ذلك. على سبيل المثال، إذا كانت قيمة **n** هي **5** يكون لدينا بعد تنفيذ التعليمة التالية:

```
i = n++;
```

قيمة **i** هي **5** وقيمة **n** هي **6**. في حين يكون لدينا بعد تنفيذ التعليمة التالية:

```
i = ++n;
```

قيمة **i** هي **5** وقيمة **n** هي **6**.

التعليمات في لغة C++، تعليمات مختصرة للإسناد

توجد في **C++** تعليما ت تنفذ التعليمة الحسابية المطلوبة والإسناد معاً. فمثلاً في حالة الجمع، وبدلاً من كتابة

```
sum = sum + x;
```

يمكن أن نكتب:

```
sum += x;
```

يبين المثال التالي استعمال التعليمات المختصرة للعمليات الحسابية +، *، /، %:

```
total += value; or total = total + value;
prod *= 10;     or prod = prod * 10;
x /= y + 1;     or x = x/(y + 1);
n %= 2;         or n = n % 2;
```

التعليمات في لغة C++، الصيغ المنطقية

إيضاح:

الصيغة المنطقية هي صيغة تعطي صح أو خطأ كنتيجة عند تقويمها

تستعمل المعاملات التالية في الصيغ المنطقية:

> أصغر من

< أكبر من

=> أصغر من أو يساوي

=< أكبر من أو يساوي

== يساوي

=! لا يساوي

2. التعليمات الشرطية

- التعليمة الشرطية **if statement**:

تأخذ التعليمة الشرطية في **C++** الشكل :

```
if (condition)
  statement
```

حيث يكون الشرط **condition** إما متحولاً منطقياً أو صيغة منطقية (راجع الفقرة السابقة).

أما **statement** فهو إما تعليمة وحيدة تنتهي بفاصلة منقوطة، أو مجموعة من التعليمات. وفي حالة مجموعة من التعليمات نضع هذه التعليمات بين قوسين كبيرين.

مثال

```
if (x > 0.0)
   cout << "The value of x is positive;"
```

- التعليمة الشرطية الاختيارية **if_else statement**:

تأخذ التعليمة الشرطية الاختيارية في **C++** الشكل :

```
if ( condition )
statementT
else
statementF
```

حيث يكون الشرط **condition** إما متحولاً منطقياً أو صيغة منطقية.

أما **statementT** و **statementF** فتدل كل منهما إما على تعليمة وحيدة تنتهي بفاصلة منقوطة، أو مجموعة من التعليمات. وفي حالة مجموعة من التعليمات نضع هذه التعليمات بين قوسين كبيرين.

مثال

```
if (disc >= 0.0)
  cout << "Roots are real";
if (disc < 0.0 )
  cout << "Roots are complex";
```

تدريب:

1. اكتب برنامجاً بلغة **C++** يقرأ عددين صحيحين **x** و **y** ويظهر على الشاشة القيمة المطلقة للفرق بينهما. اختبر البرنامج بقيم مختلفة تغطي جميع الحالات الممكنة (الأول<الثاني، الأول>الثاني، حالة المساواة)

2. اكتب برنامجاً بلغة **C++** يقرأ ثلاثة أعداد صحيحة تمثل علامات الطالب في ثلاثة مقررات، ويظهر كخرج الكلمة **Pass** أو **Fail**.

يعتبر الطالب ناجحاً (**Pass**) إذا كان ناجحاً في جميع الامتحانات (أي حائزاً على علامة **40** وما فوق في كل مقرر)، أو كان حائزاً على علامة أدنى من **40** في مقرر واحد على الأكثر ولكن وسطي العلامات الثلاث **50** وما فوق.

- تعليمة **switch**

تأخذ التعليمة **switch** في **C++** الشكل :

```
 switch (   selector )
{
case label1: statement1;
    break;
case label2: statement2;
    break;
    ...
case labeln: statementn;
    break;
default: statementd; // optional
    break;
}
```

حيث يكون **selector** متحولاً من نمط **int** أو **char** أو تعبيراً نتيجته من أحد هذين النمطين.

و **labeli** من نفس نمط **selector**.

مثال

```
switch (i)
  {
    case 1 :  grade = 'A';
              break;
    case 2 :  grade = 'B';
              break;
    case 3 :  grade = 'c';
              break;
    default : cout << i
                   << " not in range";
              break;
  }
```

3. التعليمات التكرارية

- تعليمة **while**

تأخذ التعليمة **while** في **C++** الشكل :

```
while (  condition  )
        statement
```

حيث يكون الشرط **condition** إما متحولاً منطقياً أو صيغة منطقية.

أما **statement** فهو إما تعليمة وحيدة تنتهي بفاصلة منقوطة، أو مجموعة من التعليمات. وفي حالة مجموعة من التعليمات نضع هذه التعليمات بين قوسين كبيرين.

مثال

```
sum = 0.0;
cin >> x;
while (x > 0.0)
   {
    sum += x;
    cin >> x;
   }
```

◂ تعليمة for

تأخذ التعليمة **for** في **C++** الشكل:

```
for ( initialise ; test ; update )
statement
```

حيث يمثل **initialise** عملية إعطاء قيمة ابتدائية لمتحول التحكم بالحلقة

حيث يمثل **test** عملية تقويم الصيغة المنطقية

حيث يمثل **update** عملية تعديل قيمة متحول التحكم بالحلقة قبل إعادة تنفيذها ثانية

وعادة نستعيض عن كتابة مجموعة التعليمات التالية:

```
i = 1;
while (i <= 10)
  {
    cout << i << endl;
    i++;
  }
```

بالتعليمة:

```
for (i = 1; i <= 10; i++)
  {
    cout << i << endl;
  }
```

تدريب:

1. اكتب برنامجاً يقرأالعدد الصحيح n و يحسب مجموع السلسلة:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}$$

عدل البرنامج بحيث يحسب مجموع السلسلة:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots \frac{1}{n}$$

لاحظ أنه كلما كانت n أكبر كان المجموع أقرب إلى **0.7854** أي (**π/4**)

2. اكتب برنامجاً يقرأ مجموعة من الأعداد الصحيحة التي يبلغ عددها **n** (**n** عدد صحيح يدخله المستخدم) ويحسب عدد الأعداد الموجبة والسالبة الواردة في هذه المجموعة.

3. اكتب برنامجاً يقوم بإظهار جدول الضرب من الرتبة **n** حيث **n** قيمة يدخلها المستخدم وتكون أصغر من **10**. على سبيل المثال إذا أعطى المستخدم لـ **n** القيمة **4** يجب أن يظهر الجدول التالي:

```
   1   2   3   4
1  1   2   3   4
2  2   4   6   8
3  3   6   9  12
4  4   8  12  16
```

استخدم لهذا الغرض الحلقات المتداخلة وانتبه للإظهار بحيث يظهر الجدول واضحاً.

القنوات Streams

في **C++** تقرأ المعطيات من القنوات وتكتب في القنوات. والقناة هي:

- سلسلة من المحارف
- متصلة بجهاز أو بملف (على القرص)
- تحمل اسماً محدداً

ولقد تعاملنا سابقاً مع القناتين **cin** و **cout** الموصولتين إلى لوحة المفاتيح والشاشة على الترتيب. وعند استخدام هاتين القناتين يجب تضمين المكتبة **iostream** في البرنامج.

إيضاح

تستخدم بعض المترجمات ترويسة المكتبة التي تضمن في الملف بوضع **iostream.h** بدلاً من **iostream**.

تسمح **C++** بالتعامل مع قنوات أخرى، ويصرح عن هذه القنوات كما يصرح عن أي متحول آخر. ونجد أنه للتصريح عن قنوات لإدخال المعطيات فإننا يجب أن نستخدم النمط **ifstream**، في حين نستخدم النمط **ofstream** للقنوات التي نستخدمها لإخراج المعطيات.

بعد التصريح عن القناة يجب أن نصلها بملف خارجي.

وجب ألا ننسى أن نضمن المكتبة **fstream** للبرنامج.

إيضاح

تستخدم بعض المترجمات ترويسة المكتبة التي تضمن في الملف بوضع **fstream.h** بدلاً من **fstream**.

وصل القنوات بالملفات الخارجية

إن الشكل العام لتعليمة وصل قناة بملف هو:

```
streamname.open(filename);
```

ملاحظة:

Streamname: اسم القناة

Filename: اسم الملف الخارجي

مثال

نبدأ أولاً بالتصريح عن القناة كما في التعليمات التالية:

```
ifstream ins;  // input stream
ofstream outs; // output stream
```

لوصل الأقنية بالملفات الخارجية الموافقة لها نستخدم التعليمة **open**، وهي في الواقع تابعاً يستدعى لهذا الغرض. فمثلاً لوصل القناة **ins** بالملف **indata.dat** نكتب:

```
ins.open("indata.dat");
```

إيضاح :

استدعاء التوابع الموجودة ضمن الأغراض يتم بالرمز . كما سنرى لاحقاً

إذا كان الملف **indata.dat** غير موجود فستفشل عملية الربط. نختبر ما إذا فشلت هذه العملية باستخدام التابع **streamname.fail ()** الذي يعيد القيمة **true** إذا فشل التابع **open**. يجب إجراء هذا الاختبار بعد كل عملية **open**، كما في المثال التالي:

```
ifstream ins;
ins.open("indata.dat");
if (ins.fail())
  {
    cout << "Error opening file indata.dat"
         << endl;
    return 1;
  }
```

في البرنامج الرئيسي الذي كتبت داخله هذه التعليمات، يتم الخروج من البرنامج عند فشل عملية الفتح لأننا استخدمنا التعليمة **return**. تعيد هذه التعليمة **1** مما يدل نظام التشغيل على توقف البرنامج بحالة خطأ.

بعد أن يتم الوصل فإن جميع عمليات القراءة من القناة تعني القراءة من الملف والكتابة في القناة تعني الكتابة في الملف.

مثال

```
int main()
{
 ifstream ins;
 ofstream outs;
 ins.open("indata.dat");
 if (ins.fail())
   {
     cout << "Error opening indata.dat"
          << endl;
     return 1;
   }
 outs.open("results.dat");
 if (outs.fail())
   {
     cout << "Error opening results.dat"
          << endl;
     return 1;
   }
      .
      .
```

الآن أصبح تعاملنا مع الملفين يتم من خلال المتحولين **ins** و **outs** باستخدام المعاملين >> و<<.

وهكذا نقرأ من القناة **ins** قيمة المتحول **x**:

```
ins >> x;
```

ونخرج النتائج على القناة **outs** :

```
outs << "Result is " << count << endl;
```

اختبار بلوغ نهاية ملف end-of-file

نستخدم لهذا الغرض التابع **eof**. تكون نتيجة الاختبار **true** إذا حاولنا القراءة من ملف بلغ نهايته، وإلا تكون **false**.

مثال

لقراءة القيم الموجودة في ملف وحساب وسطي هذه القيم نكتب البرنامج التالي:

```
#include <iostream.h>
#include <fstream.h>
int main()
{
  int n;
  float x, sum, average;
  ifstream ins;         // input stream
  ofstream outs;        // output stream
  ins.open("indata.dat");

     // open files, exit program if fail
  if (ins.fail())
    {
      cout << "Can't open indata.dat" <<endl;
      return 1; //exit with code 1 for failure
    }
  outs.open("results.dat");
  if (outs.fail())
    {
      cout << "Can't open results.dat" << endl;
      return 1; //exit with code 1 for failure
    }

  // Initialise and let user know something is happening
  sum = 0.0;
  n = 0;
  cout << "Reading input file " << endl;
      // read from file, accumulate sum and output average
      // to output file.
  ins >> x;  // if file was empty then eof would now be true
  while (!ins.eof())
    {
      sum += x;
      n++;
      ins >> x;
    }
  average = sum / n;
  cout << "Writing results to file " << endl;
  outs << "The average of " << n << " numbers is "
       << average << endl;
  ins.close();     // Close all files - GOOD PRACTICE
  outs.close();
  return 0;        // indicate success
}
```

لاحظ أننا قمنا بإغلاق الملفين **ins** و **outs**. يفضل في البرامج إجراء الإغلاق قبل الخروج من البرنامج، وإلا فقد يحصل أحياناً أن نفقد بعض المعطيات التي أخرجناها على الملف.

الفصل الثالث

البرامج الجزئية في C++

التوابع التي يعرفها المبرمج

تتيح لغة **C++** للمبرمج إمكان تعريف توابعه الخاصة. الشكل العام للتابع هو:

```
Function_type function_name( parameter_list)
{
Local_definitions;
Function_implementation;
}
```

- إذا كان التابع يعيد قيمة، يجب تحديد نمط هذه القيمة في **function_type**. وفي حال عدم وجود
  قيمة معادة نضع **void** في **function_type**.
- تسرد **parameter_list** موسطات التابع كل مع نمطه.
- إن **local_definitions** هي تعريفات للمتحولات المحلية المستخدمة داخل التابع، ولا يكون لهذه
  المتحولات أي معنى خارج التابع.
- تتألف **function_implementation** من التعليمات التي تقوم بالوظيفة المطلوبة من التابع.

مثال

التابع **distance** يأخذ كموسطي دخل إحداثيي نقطة **x,y** ويعيد بعد هذه النقطة عن مركز الإحداثيات.

```
float distance(float x, float y)
//Returns the distance of (x, y) from origin
    {
float dist;  //local variable
dist = sqrt(x * x + y * y) ;
return dist;
}
```

في هذا التابع قمنا باستخدام **متحول محلي local variable** يسمى **dist**.

التوابع التي ليس لها موسطات

يستخدم هذا النوع من التوابع استخداماً محدوداً، وهي تقوم عادة بوظيفة معينة دون أن تعيد أية نتيجة.

مثال

يقوم التابع التالي بإظهار انزياح مقداره ثلاثة أسطر، أي أنه يترك ثلاثة أسطر فارغة على الخرج ليبدأ الإظهار على السطر الرابع :

```
void skipthree(void)
//skips three lines on output
    {
cout << endl << endl << endl;
}
```

لاحظ نمط الخرج **void**، وعدم وجود موسطات دخل للتابع لذلك نضع **void** في **parameter_list**. كما أنه لا توجد **local_definitions**. ينصح بشدة اتباع القاعدة التي تقضي بكتابة تعليق في التابع يشرح وظيفة التابع كما في هذا التابع. ونظراً لعدم وجود نمط عودة، يجري استدعاء **call** التابع على النحو التالي:

```
Skipthree();
```

وبرغم عدم وجود موسطات دخل، فإنه **يجب** وضع القوسين ( ).

ينصح بالعودة إلى مقرر سابق للبرمجة لفهم كيفية عمل المترجم عند استدعاء البرنامج الرئيسي تابعاً ما.

ويجب أن يعرف أي تابع قبل التابع الرئيسي **main** الذي يقوم باستدعائه، لأن البرنامج الرئيسي يجب أن يعلم بوجوده وتعريفه وأنماط موسطاته عندما يصادف تعليمة استدعاء هذا التابع.

نرى في الجزء التالي من برنامج استدعاء التابع skipthree ترتيب تعريف التابع الرئيسي والتابع skipthree.

```
#include <iostream.h>
void skipthree(void)
//Function to skip three lines
{
cout << endl << endl << endl;
}

void main()
{
int …;
float …;
cout << “Title Line 1”;
skipthree();
cout << “Title Line 2”;
.
.
}
```

غير أن هذا الترتيب يقلل من فهم البرنامج، إذ أن البرنامج الرئيسي يحتوي على معلومات ذات أثر أكبر في إفهام القارئ وظيفة البرنامج من التوابع المختلفة، وعليه، يجدر به أن يأتي تعريفه أولاً. إلا أن هذا الحل يسبب عدم تعرف المترجم على التوابع عند استدعائها في البرنامج الرئيسي.

جرت معالجة هذه النقطة باستخدام ما يسمى نموذج التابع function prototypes. فنموذج التابع يحتوي على نمط العودة وأنماط موسطات الدخل. يوضع هذا النموذج قبل البرنامج الرئيسي الذي يستدعي هذا التابع. أما التعريف الكامل للتابع فيوضع بعد البرنامج الرئيسي، وقد يكون موجوداً في ملف منفصل سبقت ترجمته ويجري ربطه link مع البرنامج الرئيسي لاحقاً. يتطابق نموذج التابع مع ترويسته.

```
#include <iostream.h>

void skipthree(void);  // function prototype

void main()
{
int…..;
float…..;
cout << “Title Line 1”;
skipthree();
cout << “Title Line 2”;
.
.
}

// Now the function definition
void skipthree(void)
// Function to skip three lines
  {
cout << endl << endl << endl;
}
```

نموذج التابع skipthree

استدعاء التابع skipthree

مر معنا في الفصل السابق (راجع الفقرة: البرنامج الأول بلغة **C++** في الفصل الثاني تعرف **C++**) أن استخدام توابع من المكتبات يقتضي تضمين تعريفات هذه المكتبات في الملف عبر استخدام **include**. في الحقيقة تقوم عملية التضمين بوضع نماذج (ترويسات) التوابع في الملف

قبل البرنامج الرئيسي، بحيث يتعرف عليها المترجم عندما يجد استدعاءها، لأن هذه الملفات المضمنة تحتوي على الترويسات التي تقدمها المكتبة الموافقة.

للتعرف على مكتبة iostream.h راجع الملحق الأول في نهاية هذا الملف.

للتعرف على مكتبة iomanip.h راجع الملحق الثاني في نهاية هذا الملف.

التوابع التي تعيد نتيجة

ترد التوابع على الأغلب في شكل تأخذ فيه موسطات دخل وتعيد نتيجة. لنأخذ على سبيل المثال النقطة التي نريد حساب بعدها عن مركز الإحداثيات انطلاقاً من إحداثييها.

```
float distance(float x, float y)
//Returns the distance of (x, y) from origin
{
float dist;  //local variable
dist = sqrt(x * x + y * y);
return dist;
}
```

إن نموذج التابع distance هو:

```
float distance(float, float); // function prototype
```

يمكن لنا كتابة ما يلي في البرنامج الرئيسي:

```
float a, b, c, d, x, y;
a = 3.0;
b = 4.4;
c = 5.1;
d = 2.6;


x = distance(a, b);
y = distance(c, d);


if (distance(4.1, 6.7) > distance(x, y))
cout << " Message 1 " << endl;
```

قد نجد في التابع أكثر من تعليمة return كما في المثال التالي الذي نحسب فيه الجذر التربيعي لعدد حقيقي:

```
float mysqrt(float x)
//  Function returns square root of x.
//If x is negative it returns zero.
 {
const float tol = 1.0e-7;  // 7 significant figures
float xold, xnew;          // local variables
if (x <= 0.0)
return 0.0;              // covers -ve and zero case
else
{
xold = x;                // x as first approx
xnew = 0.5 * (xold + x / xold); // better approx
while (fabs((xold-xnew)/xnew) > tol)
{
xold = xnew;
xnew = 0.5 * (xold + x / xold);
}
return xnew;  // must return float value
}
}  //  end mysqrt
```

ولكن إذا كان التابع من النمط void، أي أنه لا يوجد نمط معاد، فيجب ألا يكون هناك تعليمة return في جسم التابع، فإن وجدت، فسيصدر المترجم تحذيراً بهذا الشأن.

مثال: تابع الرفع إلى قوة

يعيد هذا التابع نتيجة رفع الموسط الأول إلى القوة الموافقة للموسط الثاني. يكون الموسط الثاني عدداً صحيحاً، ويمكن أن تكون هذه القوة 0 أو سالبة.

```
float power(float x, int n)
   {
float product = 1.0;
int absn;
int i;
if ( n == 0)
return 1.0;
else
    {
absn = int(fabs(n));
for (i = 1; i <= absn; i++)
product *= x;
if (n < 0)
return 1.0 // product;
else
return product;
        }
}    //end of power
```

يمكن استخدام التابع كما يلي:

```
float x, y, z;
int p;
cout << “Enter a float and an integer:”;
cin >> x >> p;
y = power(x, p);
z = power(x + y, 3);
```

الموسطات التي تمرر بالقيمة Call-by-value parameters

مر معك في مقرر سابق للبرمجة نوعان من تمرير الموسطات: تمرير بالقيمة وتمرير بالمرجع.

(راجع البرمجة1 الفقرة "تمرير الوسطاء (المحددات) parameter Passing").

في التمرير بالقيمة لا تتغير قيمة المتحول الفعلي الذي يحل محل الموسط الشكلي عند الاستدعاء.
وهكذا ، فلو أضفنا التعليمة:

```
n++;
```

إلى تعليمات التابع power مباشرة قبل قوس نهاية التابع، ثم نفذنا التعليمات:

```
p = 4;
y = power(x, p);
cout << p;
```

لوجدنا النتيجة هي 4 وليس 5، وذلك لأنه جرى تمرير الموسط بالقيمة passed by value.

ويعني هذا التمرير أنه عند الاستدعاء نأخذ نسخة من قيمة المتحول الفعلي ونتعامل معها داخل التابع، في حين تبقى قيمة المتحول الفعلي بعيدة عن التغيير.

نستخدم هذا النمط من التمرير عندما نرغب بتمرير المعلومات للتابع من خلال متحول معين دون إجراء أي تعديل عليه، وهذا ما يجنّب إحداث آثار جانبية.

الموسطات التي تمرر بالمرجع Call-by- reference parameters

لا يمكن إعادة قيمة جرى حسابها في تابع عبر الموسطات الممررة بالقيمة للأسباب التي ذكرناها في الفقرة السابقة، ولكن لو رغبنا بذلك فيجب أن نمرر للتابع عنوان الموسط لا قيمته. يستخدم التابع هذا العنوان للنفاذ إلى الموسط الفعلي وإجراء التعديل اللازم على قيمته بحيث يبقى هذا التعديل جاري المفعول بعد انتهاء استدعاء التابع. وبما أننا نمرر مرجعاً للموسط، فإنني نسمي هذا النمط بالتمرير بالمرجع **call-by-reference**. ولتمييز الموسط الذي نمرره بالمرجع عن ذلك الذي نمرره بالقيمة، نستخدم الرمز & الذي نضعه بعد نمط الموسط.

مثال

ليكن لدينا التابع التالي الذي يحل معادلة من الدرجة الثانية:

```
//  solves the quadratic equation a*x*x+b*x+c = 0.
//  If the roots are real then the roots are
//returned in two parameters root1 and root2 and
//  the function returns true, if they are complex
//then the function returns false.

bool quadsolve(float a,        // IN coefficient
                    float b,        // IN coefficient
                    float c,        // IN coefficient
                    float& root1,   // OUT root
                    float& root2)   // OUT root
 {
float disc;      // local variable
disc = b * b - 4 * a * c;
if (disc < 0.0)
return false;
else
        {
root1 = (-b + sqrt(disc))/(2 * a);
root2 = (-b - sqrt(disc))/(2 * a);
return true;
        }
  }
```

لاحظ أنه جرى التصريح عن الجذرين (وهما موسطي الخرج) كموسطين ممررين بالمرجع، في حين أنه جرى تمرير أمثال المعادلة بالقيمة لأنها موسطات دخل. ويكون شكل نموذج التابع كمايلي :

```
int quadsolve(float, float, float, float&, float&);
```

يجري استدعاء التابع في البرنامج الرئيسي على النحو التالي:

```
float c1, c2, c3;
float r1, r2;
.
.
if (quadsolve(c1, c2, c3, r1, r2))
cout << “Roots are “ << r1 << “ and “ << r2 << endl;
else
cout << “Complex Roots” << endl;
```

تدريب:

يطلب كتابة برنامج يقوم بمعالجة فاتورة تحتوي المعلومات التالية:

n عدد البنود الموجودة في الفاتورة

يتميز كل بند بالمعلومات التالية: رمز للبند(على ست خانات)، الكمية، سعر القطعة الواحدة(بالليرات وأجزائها العشرية)

وهكذا يمكن على سبيل المثال إعطاء الدخل التالي للبرنامج :

```
3
161432 5 6 50
543289 10 2 25
876234 2 10 75
```

للدلالة على وجود 3 بنود في الفاتورة، أولها رمزه هو 161432 كميته 5 وسعر الواحدة منه 6 ليرات و50 قرشاً.

والمطلوب كتابة برنامج بلغة **C++** يقرأ تاريخ اليوم، ثم يقرأ المعلومات الخاصة بفاتورة كما في المثال أعلاه، ويقرأ تاريخ اليوم، ثم يظهر السعر الكلي لكل بند، وسعر المجموع.

يكون شكل الخرج كما يلي:

```
Invoice date: 10/6/96
 Item    quantity  unit price  total price

161432       5         6.50        32.50
543289      10         2.25        22.50
876234       2        10.75        21.50

                          Total    76.50
```

يمكن أن نكتب خوارزمية الحل على النحو التالي:

إعطاء القيم الابتدائية initialise

قراءة التاريخ

قراءة عدد البنود n

إظهار الترويسة

من أجل n بند كرر

{

قراءة المعلومات المتعلقة بالبند

حساب السعر الإجمالي للبند

الإضافة للسعر الإجمالي للفاتورة

كتابة سطر في الفاتورة خاص بالبند

إظهار السعر الإجمالي للفاتورة

}

لو افترضنا أنه يوجد لدينا أربعة مبرمجين لكتابة الحل، فيمكن أن نقسم الحل إلى أربعة مهام منفصلة نراها واضحة داخل الحلقة، ويمكن إعطاء كل مهمة منها لمبرمج. ولعل أفضل طريقة لإنجاح تقسيم المهمات هو الطلب من كل مبرمج أن يكتب تابعاً للمهمة الموكلة إليه.

وتحدد مهمة كل مبرمج بتوصيف دقيق كما يلي:

لنأخذ مثلاً مهمة قراءة المعلومات المتعلقة بالبند. توصف المهمة كما يلي:

```
Function name: dataentry
Operation:     Enter a record
Description:   Enters four integers from the current
               input line and returns their values.
Parameters:    Output parameter  int  itemno
               Output parameter  int  quantity
               Output parameter  int  unitpounds
               Output parameter  int  unitpence
```

وبالمثل توصف المهمة حساب السعر الإجمالي للبند كما يلي:

```
Function name : calccost
Operation     : Calculates the cost for a single item.
Description   : Given the unit price of an item in
                pounds and pence and the quantity of
                the item calculates the total cost in
                pounds and pence.
Parameters    : Input parameter int quantity
                input parameter int unitpounds
                input parameter int unitpence
                output parameter int totalpound
                output parameter int totalpence
```

وبالمثل توصف المهمة الإضافة للسعر الإجمالي للفاتورة كما يلي:

```
Function name : acctotal
Operation     : Accumulates the total cost of invoice
Description   : Given current total invoice cost and
                the total cost of an invoice item
                calculates the new total invoice cost.
Parameters    : input parameter int totalpound
                input parameter int totalpence
```

input & output parameter int invpound
input & output parameter int invpence

وبالمثل توصف المهمة كتابة سطر في الفاتورة خاص بالبند كما يلي:

Function name : writeline

Operation : Writes a line of the invoice.

Description : Given the item reference number, the quantity, the unit price and total price of an item outputs a line of the invoice.

Parameters : input parameter int itemno
input parameter int quantity
input parameter int unitpounds
input parameter int unitpence
input parameter int totalpound
input parameter int totalpence

وهكذا نحصل على البرنامج التالي الذي يقوم باستدعاء التوابع التي تنفذ هذه المهمات على النحو التالي:

```
void main()
{
    int i,          // control variable
        n,          // number of items
        itemno,     // item reference number
        quantity,   // quantity of item
unitpounds,
unitpence,  // unit item price
totalpound,
totalpence, // total item price
invpound,
invpence;   // total invoice price
//  initialise
invpound = 0; // total value of invoice has to be
invpence = 0; // set to zero initially
//  Enter number of items
cout << “Enter number of items on invoice:”;
cin  >> n;
//  Headings
cout << “ Item   quantity  unit price  total price”
<<endl << endl;
//For n items
for (i=1; i<=n; i++)
     {
dataentry(itemno, quantity, unitpounds, unitpence);
calccost(quantity, unitpounds, unitpence, totalpound,
totalpence);
acctotal(totalpound, totalpence, invpound, invpence);
writeline(itemno, quantity, unitpounds, unitpence,
totalpound, totalpence);
   }
//write total line
cout << “                          Total     “
<< invpound
 << “.”
<<invpence << endl;
}
```

لاحظ هنا أن الفائدة العظمى من التوابع هي أننا يمكن أن نأخذ توصيف أي من المهمات الواردة سابقاً، فننفذها بكتابة تابع يحقق هذه المهمة، ونختبره على حده. فمثلاً، لو أخذنا التابع calccost فإننا ننفذه على النحو التالي:

```
void calccost(int q, int ul, int up, int& totl, int& totp)
//  Calculates the quantity q times the unit cost in
//  pounds and pence in ul and up and places the
//result in pounds and pence in totl and totp
{
int p;
p = q * up;
totp = p % 100;
totl = q * ul + p/100;
}
```

لاختبار هذا التابع على نحو مستقل عن البرنامج، يجب أن نكتب **برنامج سواق driver program.**

إيضاح:

البرنامج السواق هو برنامج يسمح بأن يقوم المبرمج بإدخال قيم لموسطات التابع المراد اختباره ، ثم يظهر النتيجة الموافقة.

فعلى سبيل المثال يمكن أن نكتب البرنامج السواق التالي:

```
// Driver program to test calccost
#include <iostream.h>
// function prototype
void calccost (int, int, int, int&, int&);


void main()
{
int quant, unitl, unitp, totall, totalp;
// stop on negative quantity
cout << “Enter quantity:”;
cin >> quant;
while (quant >= 0)
     {
cout << “Enter unit cost (pounds pence:) “;
cin >> unitl >> unitp;
calccost(quant, unitl, unitp, totall, totalp);
cout << endl
<< quant << “ times”
<< unitl << “ pounds”
<<unitp << “ pence”
<<”is “
<< totall << “ pounds”
<< totalp << “ pence “;
cout << endl << “ Enter quantity:”;
cin >> quant;
}
}
// function definition here
```

عند الاختبار يجب أن تحاول عطاء قيم دخل توافق جميع الحالات. فمثلاً إذا أدخلنا معطيات دخل لهذا التابع بحيث نسحب السعر الإجمالي لـ 7 قطع من بند سعر القطعة منه هو 1.10 لا يختبر حالات وجود حمل في الجزء العشري من المبلغ بحيث يضاف إلى الجزء الصحيح. ينصح باختبار السعر الإجمالي لـ 6 قطع يبلغ سعر الواحدة منها 2.74 مثلاً. نقصد هنا أن نقول أن اختيار معطيات الدخل يستلزم عناية خاصة.

التحميل الزائد للتوابع

يسمح في البرنامج نفسه في C++ أن يجري تعريف تابعين أو أكثر لهما الاسم نفسه. يفترض هذا ضمناً أن هذه التوابع تختلف فيما يتعلق بقائمة موسطاتها الشكلية formal parameters. وتقع على عاتق المترجم مهمة تحديد التابع الواجب استدعاؤه بناء على الموسطات الفعلية effective parameters المستخدمة عند استدعاء التابع.

يسمح هذا الإمكان بتسهيل قراءة البرامج، إذ لا يطالَب المبرمج في كل مرة باختراع اسم جديد لتابع يقوم بعمل مطابق لتابع آخر، لكن على معطيات من نمط مختلف. إن إمكان زيادة تحميل التوابع يسمح بإعطاء صفة تعددية الأشكال للعمل، يحدد الشكل حسب نمط الموسطات وعددها عند الاستدعاء.

لنأخذ مثلاً الحالة التي نريد فيها صنع تابع لإظهار القيمة الممررة في الموسط. لو كانت اللغة لا تسمح باستخدام الاسم ذاته للتوابع كما في العديد من لغات البرمجة، لوجب أن نكتب التوابع التالية:

```
void displayint (int i  )
{
cout<<”integer “<<i<<endl;
}


void displaydouble (double i  )
{
cout<<”double “ <<i<<endl;
}
```

تقدم لغة C++ إمكان استخدام الاسم نفسه، فنكتب البرنامج التالي:

```
#include <iostream.h>
void display (int i)
{
cout<<"integer "<<i<<endl;
}

void display (double i  )
{
cout<<"double " <<i<<endl;
}

int main()
{
display(7);
display(3.5);
return 0;
}
```

يعطي تنفيذ البرنامج:

```
integer 7
double 3.5
```

يتبين لنا هنا أن المترجم قام في كلتا الحالتين بالاختيار الصحيح للتابع الواجب استدعاؤه حسب نمط الموسطات الممررة للتابع. ولكن يجب الانتباه طبعاً إلى عدم التسبب في حالات التباس لا يستطيع فيها المترجم الاختيار بدلاً عن المبرمج.

مثال

```
#include <iostream.h>
void display (int i )
{cout<<"integer "<<i<<endl;}

void display (double i )
{cout<<"double " <<i<<endl;}

int main(){
long int i = 1;
display(7);
display(3.5);
display(i);  // Error … compilation is ambiguous!!
return 0;
}
```

في هذا المثال، يقع المترجم في حيرة، إذ أنه لا يعرف: أيحول المتحول i ذو القيمة 1 والنمط long ضمنياً إلى int أم إلى double؟ علماً بأن كلتي الحالتين ممكنتان. يجب إذن استعمال زيادة التحميل مع الحذر من الوقوع في مثل هذه الحالات.

المعاملات الافتراضية

عندما يكتب المبرمج توابع تستخدم موسطات، يمكن أن يكون لبعض هذه الموسطات قيم اعتيادية. في هذه الحالة، يمكن تعريف تابع بمحددات ذات قيم افتراضية. يسرع هذا الأمر تنفيذ استدعاء التابع لأن هذه المحددات لا تحتاج إلى أن تُنسخ. لهذا السبب، جعلت المحددات التي تأخذ قيماً افتراضية على اليمين في نموذج التابع.

فعلى سبيل المثال:

```
void init (int a , int b  = 0) ;      // 2nd argument = 0 by default
```

في هذه الحالة، init (3) و init (2,4) هي أشكال صحيحة لاستدعاء التابع المصرح عنه.

إن المثال التالي خاطئ لأن المحددات الافتراضية يجب أن تكون متوالية اعتباراً من أقصى اليمين نحو اليسار.

```
void incorrect (int a =3, int b, int c = 0) ; //error, b has no value
```

إذا جرى التصريح عن تابع مع محددات افتراضية، فلا يصح عند تعريف التابع تحديد هذه القيم الافتراضية مرة أخرى

```
void init (int a = 0) ;
//declaration of init
    .
    .
main()
{
    .
    .
}

void init (int a = 0) //definition of init ERROR
{/* body   */ }
```

```
void init (int a = 0) ;
// declaration of init
    .
    .
main()
{
    .
    .
}

void init (int a ) //OK
    { / * body * / }
```

بالطبع، يسمح بالتعريف الزائد للتابع ذي المحددات الافتراضية. في هذه الحالة يجب التنبه إلى عدم تحديد القيم الافتراضية مرة أخرى عند إعادة التعريف.

في المثال التالي نعرف التابع init الذي يأخذ عددين صحيحين كدخل، غير أنه يمكن أن يأخذ العدد الأول، وتكون القيمة الافتراضية للثاني 3. نقوم بعد ذلك بتعريف ثان للتابع نعطي فيه قيمة افتراضية للموسط الأول ولتكن 7.

```
#include <iostream.h>
void init(int,int=3);
void init(int a,int b) { cout<<a <<",”<<b;}
void init(int=7,int);  // overdefinition

int main(){
init(2,1); // displays 2,1
init(4);   // displays 4,3
init();    // displays 7,3
return 0;
}
```

عند التعريف الزائد للتابع init() هنا لم يكن يسمح بإعادة تعريف القيمة الافتراضية للمحدد الثاني. نلاحظ هنا أنه يجب ألا يذكر اسم المحدد الذي يأخذ قيمة افتراضية.

المصفوفات Arrays في C++

لا بد أنك قد سبق لك أن تعاملت مع بنية المعطيات المسماة مصفوفة في مقرر سابق للبرمجة (راجع فقرة "بنية المعطيات مصفوفة" في المقرر برمجة1)، وهي بنية تسمح بإعطاء اسم واحد لمجموعة من العناصر التي لها نفس النمط، ويمكن الوصول إلى كل عنصر منه باستخدام الدليل subscript /index الذي يدل على ترتيبه في المصفوفة، ويجب أن يكون هذا الدليل عدداً صحيحاً.

للتصريح عن المصفوفة في C++ نحدد نمط العناصر التي سنخزنها في المصفوفة، كما نعطي اسماً للمصفوفة، ونحدد بين قوسين متوسطين عدد العناصر فيها ويجب أن يكون هذا العدد صحيحاً.

مثال

يمكن تخزين المعطيات التي تمثل وسطي درجة الحرارة في العام في سوريا على مدى الأعوام الخمسين الماضية في مصفوفة على النحو التالي:

```
float annual_temp[50];
```

يجب أن يكون عدد العناصر محدداً في زمن الترجمة، لذا يفضل وضع القيمة الموافقة لطول المصفوفة في ثابت، بحيث نقوم في حال رغبنا بتغيير طول الشعاع بتعديل قيمة هذا الثابت فقط بدلاً من البحث داخل أسطر الرماز عن كل موقع تظهر فيه هذه القيمة واستبدالها.

```
const int NE = 50;
float annual_temp[NE];
```

ويجب عدم استخدام متحول لطول الشعاع لأن المترجم لا يعرف قيمة المتحول عند الترجمة كما في المثال:

```
int i;
cin>>i;
float annual_temp[i];
//Error
```

وبالمثل يعتبر التصريح التالي غير صحيح:

```
int i=50;
float
annual_temp[i];//Error
```

دليل المصفوفة

يأخذ العنصر الأول في المصفوفة في C++ الدليل ذا القيمة 0، في حين يأخذ العنصر الأخير القيمة الموافقة للطول -1، ويمكن وضع قيمة الدليل في متحول وإجراء عمليات عليه.

مثال

```
const int NE = 100;
N = 50;
int i, j, count[N];
float annual_temp[NE];
float sum, av1, av2;
.
for (i = 0; i < NE; i++)
cin >> annual_temp[i];
.
.
cin >> count[i];
count[i] = count[i] + 5;
count[i] += 12;
.
.
if (annual_temp[j] < 10.0)
cout << “It was cold this year”<<endl;
```

ولإيجاد وسطي درجات الحرارة لـ k عنصر مخزن في المصفوفة annual_temp اعتباراً من بدايتها (k<=NE) نكتب:

```
sum = 0.0;
for (i = NE - k; i < NE; i++)
  sum += annual_temp[i];
av2 = sum / k;
```

ملاحظة هامة:

لا تتحقق لغة C++ من كون قيمة الدليل التي يجري التعامل معها في تعليمة ما هي بين القيمة الدنيا والعظمى المصرح عنها. وعليه يمكن أن تخطئ وتكتب annual_temp[200]=10.8; (تعرف المصفوفة annual_temp كما يلي:

```
const int NE = 100;
float annual_temp[NE];
```

(

دون أن يعطي المترجم خطأ ولكن لا يمكن توقع نتيجة ذلك الأمر. فقد يحدث أن يحاول البرنامج الكتابة على موضع في الذاكرة يقع خارج فضاء الذاكرة المخصصة له، وعندها يتدخل برنامج التشغيل لإيقاف عمل البرنامج.

وقد يحاول الكتابة على موضع في الذاكرة يقع داخل فضاء الذاكرة المخصصة له فيعدل قيمة متحول في البرنامج أعطي له هذا الموضع، أو يكتب فوق الرماز المولد لبرنامجك. ويحصل أمر مماثل في حالة قراءة المعطيات المخزنة خارج المجال.

تقع مسؤولية حسن إدارة الدليل بحيث لا يتجاوز المجال المصرح عنه على المبرمج. ويسمى هذا التجاوز **طوفان الدليل Subscript Overflow.**

إعطاء المصفوفة قيمة ابتدائية

كما في حالة المتحولات البسيطة، يمكننا عند التصريح عن المصفوفة إعطاء عناصرها قيماً ابتدائية على النحو التالي:

```
int primes[] = {1, 2, 3, 5, 7, 11, 13};
```

لاحظ أننا هنا لم نحدد طول المصفوفة، إذ أن المترجم يمكنه استنتاج الطول من عدد القيم الابتدائية المعطاة وهو هنا 7.

وبافتراض أننا حددنا طول المصفوفة، فيمكن أن تحتوي قائمة القيم الابتدائية عدداً من العناصر مساوياً أو أقل من الطول. ففي المثال:

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

يحجز المترجم عشرة مواضع ويضع في الخمسة الأولى منها القيم الواردة في القائمة، ويكمل ما تبقى بـ 0.

سلاسل المحارف strings في C++

يمكن في C++ بناء سلسلة محارف والتعامل معها وكأنها عنصر واحد، وذلك باعتبارها مصفوفة عناصرها من النمط char. ويوضع عادة علامة في نهاية هذا النوع من المصفوفات للدلالة على نهايتها، وهذه العلامة هي محرف خاص يسمى **المحرف الصفري** **null character** وقيمته هي '\0' . وهكذا يكون طول كل سلسلة محارف مساوياً لعدد محارفها+1.

ونصرح عن متحول s1 كسلسلة محارف على النحو التالي:

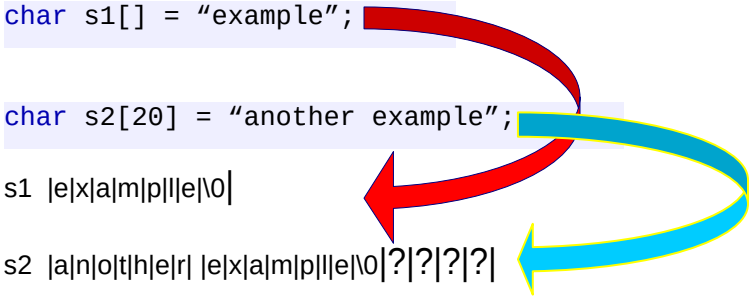
```
char s1[10];
```

تتسع s1 لتسعة محارف (+ المحرف الصفري).

مثال

```
char s1[] = "example";

char s2[20] = "another example";
```



سلاسل المحارف strings في C++، إخراج المحارف

كما هي الحال في جميع أنواع المتحولات، يكون الإخراج كما في المثال التالي:

```
cout << "The string s1 is " << s1 << endl;
```

فيظهر على الشاشة:

The string s1 is example

سلاسل المحارف strings في C++، إدخال المحارف

كما هي الحال في جميع أنواع المتحولات، يكون الإدخال كما في المثال التالي:

```
cin >> s1;
```

تنتهي قراءة متحول من نمط سلسلة محارف عندما يدخل المستخدم فراغاً أو enter أو tab.
ويضاف المحرف الصفري إلى آخر المتحول s1 .

بافتراض أن المتحول s1 كان يحتوي القيمة "example" قبل عملية القراءة

ثم أدخل المستخدم الكلمة first لوجدنا داخل المتحول s1 :

|f|i|r|s|t|\0|e|\0|

إيضاح:

بما أن طول الكلمة المدخلة first أقصر من طول السلسلة s1 تخزن الكلمة، ثم يوضع 0\ ويبقى باقي السلسلة على حاله قبل تنفيذ هذه التعليمة أي أنه يبقى فيه المحتوى السابق.

تقويم:

ماذا يحصل لو قرأنا كلمة طولها أكبر من طول المتحول الذي يجب أن نخزنها فيه؟

لقراءة عدة كلمات من الدخل باستخدام تعيمة cin يجب تخصيص عدة متحولات،كما في المثال التالي:

```
char first[12], last[12];
cout << “Enter your name (first last)";
cin >> first;
cin >> last;
cout << “The name entered was “
<<first “  “<<
<< last;
```

تدريب:

نريد كتابة برنامج ينسخ ملفاً يحتوي على أعداد صحيحة في ملف آخر. يدخل المستخدم اسم الملف الأصلي (ملف الدخل) واسم الملف الناتج (ملف الخرج) ثم ينسخ محتويات الملف الأصلي على الملف الناتج.

```
// Example program which copies a specified
// input file to a specified output file.
// It is assumed that the input file holds a
// sequence of integer values.

#include <iostream.h>
#include <fstream.h>

int main()
{
  ifstream ins;                  // declare input and output
  ofstream outs;                 // file streams
char infile[20], outfile[20];  // strings for file names
int I;
// ask user for file names
cout << “Enter input file name:”;
cin >> infile;
cout << “Enter output file name:”;
cin >> outfile;

//Associate file names with streams
ins.open(infile);
if (ins.fail())
{
cout << “Could not open file “ << infile <<” for input” << endl;
return 1; // exit with code 1 for failure
}
outs.open(outfile);
if (outs.fail())
{
cout << “Could not open file “ << outfile
<<” for output” << endl;
return 1; // exit with code 1 for failure
   }

//input from input file and copy to output file
ins >> I;
while (!ins.eof())
{
outs << i<<”  “;
```

```
ins >> i;
}
outs << endl;
//close files
ins.close();
outs.close();
return 0; //return success indication.
}
```

النمط string في C++

يوجد في C++ نمط خاص بالسلاسل المحرفية يسمى string يمكّنك من التعامل مع أية سلسلة محرفية كما لو كنت تتعامل مع متحول بسيط، مما يسهل تداول السلاسل المحرفية. يتميز هذا النمط عن تعريف السلسلة المحرفية كمصفوفة من المحارف بأنه لا يلزم المبرمج بتحديد الطول عند التصريح، كما يسمح له بالتعامل مع المتحولات من هذا النمط بعمليات سهلة وبسيطة.

التصريح عن متحول من هذا النمط يجب كتابة: #include<string> ثم:

```
string s,s1="hi";
```

تجري قراءة المتحول من النمط string باستخدام cin،

!=	==	<	>

يمكن مقارنة متحولين من هذا النمط مباشرة باستخدام الرموز

والترتيب الموجود على هذا النمط هو الترتيب الأبجدي على مستوى الكلمة كاملة.

توجد توابع وعمليات خاصة للتعامل مع هذا النمط منها:

- التابع substr(b,l): وهو تابع يعطي كنتيجة السلسلة الجزئية الموجودة في السلسلة التي يستدعى معها هذا التابع اعتباراً من الدليل b وبطول مقداره L.

مثال

```
string s1="hello";
string s2=s1.substr(0,2);
cout<<s2;   //result:  he
```

ملاحظة

لاحظ هنا أن المتحول s1 الذي كنا نتوقع تمريره للتابع substr بين قوسين قد قام هو باستدعاء التابع substr باستخدام المؤثر النقطة . ذلك لأن s1 هو غرض وهذه هي طريقة استدعاء التوابع في الأغراض. سنفصل في فصل "الصفوف والأغراض" هذه الطريقة.

- التابع length: للحصول على طول سلسلة.

مثال

```
string s1=”hello”;
int le=s1.length();
cout<<le;  //result     5
```

- المؤثر [ ] : يمكن استخدام المؤثر [ ] للوصول إلى محرف محدد.

مثال

```
string s1=”hello”;
char c=s1[0];
cout<<c;  //result      h
```

- المؤثر + : يمكن استخدام المؤثر + لضم سلسلتين إلى بعضها.

مثال

```
string s1=”hello”;
string s2=” world”;
string s=s1+s2;
cout<<s;  //result      hello world
```

مثال

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

int main()
{
string s1,s2;
s1="hello";  //assignment
cout<<"Guess the hidden word, Enter a word"<<endl;
cin>>s2;
if (s1==s2)
   cout<<"You guess!"<<endl;
else cout<<"You miss it, I will give you the first two characters"<<endl;
string s=s1.substr(0,2);
cout<<s;
return 0
};
```

الملحق 1

تحتوي هذه المكتبة تعريفات لمجموعة من المؤثرات نذكر منها على سبيل المثال:

setfill(char) , setprecision(int), setw(int).

- setw : يسمح setw(int) بتحديد حجم الحقل المراد إدراج القيمة ضمنه. وفي هذه الحالة يكون التنسيق إلى اليمين وهو أمر عملي في حالة تمثيل الأعداد.

```
#include <iostream.h>
#include <iomanip.h>
int main() {
  int n = 64;
  cout << "In hexadecimal : " << hex << n << endl;
  cout << " In octal : " << oct << n << endl;
  cout << " In decimal : " << dec << n << endl;
  //  The same display right justified
  cout << setw(20) << "Hexadecimal : " << hex << setw(6) << n << endl;
  cout << setw(20) << "Octal : " << oct << setw(6) << n << endl;
  cout << setw(20) << "Decimal : " << dec << setw(6) << n << endl;

  return 0;
}
```

يعطي التنفيذ النتيجة التالية:

```
In hexadecimal :  40
 In octal : 100
 In decimal : 64
     Hexadecimal :     40
                  Octal :   100
                 Decimal  :     64
```

يبدو هذا المؤثر مفيداً جداً عند تحصيل القيم. يقوم البرنامج التالي بإظهار النص المدخل على شكل كتل طولها LL-1 ينتهي كل منها بالمحرف "\0".

```
#include <iostream.h>
#include <iomanip.h>
const int LL = 10;  // maximum size of a line
int main() {
  char line[LL];
  while (cin >> setw(LL) >> line)
       cout << line << endl;
  return 0;
}
```

يعطي التنفيذ ما يلي:

```
abcdefghijklmnopqrstuvwxyz
abcdefghi
jklmnopqr
stuvwxyz
```

^Z

ملاحظة : ^Z هو المحرف الذي ينهي ملف الدخل في الأجهزة التي تعمل بـ Ms_Dos.

- setprecision : في الحالة الافتراضية، يجري إظهار أعداد الفاصلة العائمة على 6 أرقام معنوية. يمكن تعديل هذه الدقة باستعمال المعالج setprecision (int).

```
#include <iostream.>
#include <iomanip.h>

const double pi = 3.141592654;

int main() {
  cout << pi << endl;                    // 6 digit by default
  cout << setprecision(9) << pi << endl; // 9 digit
  cout << pi/2.0 << endl;                // we are still on  9 digit
  cout << setprecision(2) << pi << endl; // 2 digit
  return 0;
}
```

وعند التنفيذ نجد:

```
 3.14159
 3.14159265
 1.57079633
 3.1
```

setfill: يستعمل المؤثر setfill(char) عادةً مرتبطاً بـ setw(int)، ويملأ المساحة غير المستعملة بالمحرف المحدد.

```
#include <iostream.h>
#include <iomanip.h>
int main() {
  int n = 64;
  cout << "In hexadecimal : " << hex << n << endl;
  cout << "In octal : " << oct << n << endl;
  cout << "In decimal : " << dec << n << endl;
  // The same display right justified
  cout << setw(20) << "Octal : " << oct << setw(6) << n << endl;
  cout << setw(20) << "Decimal : " << dec << setw(6) << n << endl;

  cout << setw(20) << "In hexadecimal : ";
  cout << hex << setfill('.') << setw(6) << n << endl;
  cout << setfill(' ') << setw(20) << "In octal : ";
  cout << oct << setfill('.') << setw(6) << n << endl;
  cout << setfill(' ') << setw(20) << "In decimal : ";
  cout << dec << setfill('.') << setw(6) << n << endl;

  return 0;
}
```

يعطي التنفيذ النتيجة التالية:

```
 In hexadecimal :  40
  In octal : 100
 In decimal : 64
       In hexadecimal : ….40
                 In  Octal : ...100
       In Decimal  : .…64
```

يمكن للقارئ المهتم أن يعود لدليل المستخدم للمترجم **C++** الذي يعمل عليه، للاطلاع على القائمة الشاملة للمؤثرات الموجودة.

الملحق 2

تعرف مكتبة iostream.h التي توجد مع كل نظام C++ نمطين من الدخل والخرج يسمى الأول istream (input stream) (أي قناة الدخل) والثاني ostream (output stream) (أي قناة الخرج).

تُنفّذ عملية الإخراج من قبل المبرمج باستعمال مؤثر الإدراج ">>" إذ يدرج قيمة في قناة الخرج.

أما عملية الإدخال فهي تنفّذ باستعمال مؤثر الاستخراج "<<". أي أننا نستخرج قيمة من قناة الدخل.

وهنالك ثلاثة متحولات قَبلِيةُ التعريف من النمط "قناة" وهي:

- cin، متحول من النمط istream مرتبط بملف الدخل المِقيس.
- cout، متحول من النمط ostream مرتبط بملف الخرج المِقيس.
- Cerr متحول من النمط ostream مرتبط بملف الأخطاء المِقيس ولا ترتبط به ذاكرة مؤقتة.

كما يوجد في هذه المكتبة تعريفات للمؤثرات التالية:

- المؤثر endl الذي يتسبب في إدراج المحرف "\n"
- المؤثرات dec, flush, hex, oct :

1. flush :يتسبب في كتابة الذاكرة المؤقتة المرتبطة بالقناة، ومن ثَم بمسحها وإفراغها. قد يكون هذا الأمر ضرورياً في عمليات التحصيل، إذ يجب أن يظهر النص الذي يوضح المطلوب قبل أن يقوم المؤثر بإدخال القيمة المطلوبة. فإن لم يستعمل flush، فلا نضمن هذا الأمر.

```
#include <iostream.h>
int main() {
  int numbe;
  cout << "Enter a number between 0 and 12 " << flush;

  cin >> nombre;
}
```

2. Hex يجعل النظام ينتقل للنمط الست عشري أثناء استخراج وإدراج الأعداد الصحيحة. يبقى هذا النظام فعالاً إلى أن يطلب المبرمج تغييره. وهناك

مؤثرات أخرى مشابهة هي oct, dec، غير أن الأول يعتمد الأساس الثماني، والثاني يعتمد الأساس العشري. في الحالة العامة الافتراضية يعمل النظام في النمط العشري.

```
#include <iostream.h>

int main() {
  int n;
  cout << "Enter an integer number " << flush;
  cin >> n;  // equivalent to cin >> dec >> n;
  cout << "This is the  number in hexadecimal : " << hex << n << endl;
  cout << " This is the  number in octal : " << oct << n << endl;

  cout << (++n) << endl; // we are still in octal mode!

  cout << " This is the  number in hexadecimal : " << hex << n << endl;
  // now we move to decimal mode
  cout << " This is the  number in decimal : " << dec << n << endl;

  cout << " Enter an integer number " << flush;
  cin >> hex >> n;  // input in en hexadecimal mode
  cout << " This is the  number in hexadecimal: " << hex << n << endl;
  cout << " This is the  number in decimal: " << dec << n << endl;

  cout << " Enter an integer number " << flush;
  cin >> n;  // still in hexadecimal mode
 cout << "" This is the  number in hexadecimal: " << hex << n << endl;
  cout << "" This is the  number in decimal: " << dec << n << endl;

  return 0;
}
```

يعطي تنفيذ البرنامج ما يلي:

```
Enter an integer number 45
This is the  number in hexadecimal : 2d
This is the  number in octal : 55
56
This is the  number in hexadecimal : 2
This is the  number in decimal : 46
Enter an integer number AB
This is the  number in hexadecimal: ab
This is the  number in decimal: 171
Enter an integer number D2
This is the  number in hexadecimal: d2
This is the  number in decimal: 210
```

- يسمح المؤثر ws (white space) أثناء القراءة من القناة باستخراج واستبعاد جميع المحارف "الفارغة" أي محارف الفراغ والجدولة والقفز إلى سطر جديد والقفز إلى صفحة جديدة، وذلك اعتباراً من الموقع الحالي وحتى أول محرف غير فارغ

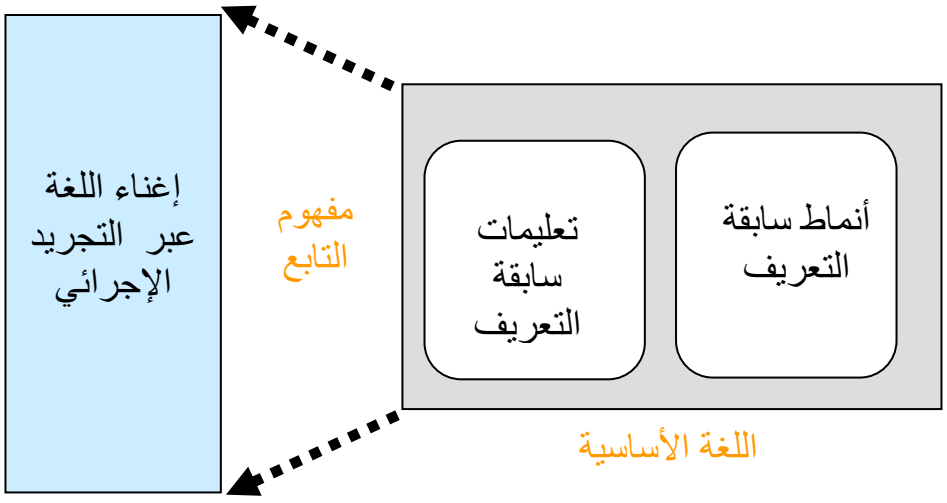
الفصل الرابع
الصفوف والأغراض

مقدمة

تقدم لغة **C++** للمبرمج وسائل تعريف أنماط جديدة، مع إمكان دمج هذه الأنماط على وجهٍ أفضل في مجموعة الأنماط المعرفة سابقاً، يرافق ذلك كله قيام المترجم بجميع عمليات التحكم والمراقبة.

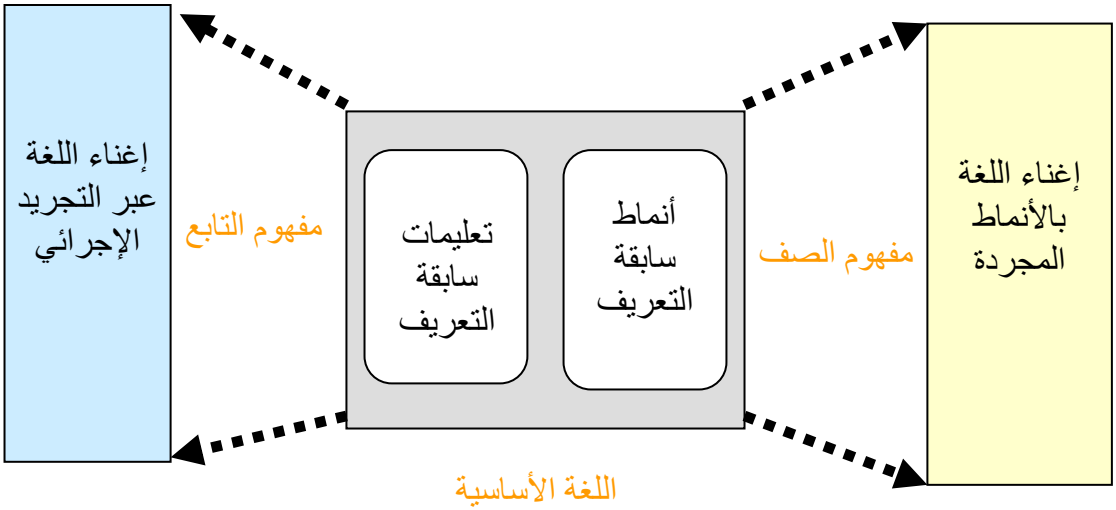
إن بناء أنماط جديدة قابلة للتكامل التام مع ما هو موجود في اللغة، هو خطوةٌ نحو بناء برمجيات أفضل جودةً، ونحو قابلية إعادة استعمال الترميز الموجود، باستغلال أنماط المعطيات التي يبنيها المبرمج أثناء عمله.

وقد مر معك أنه يمكنك في **C++** بناء عمليات جديدة إضافة إلى العمليات السابقة التعريف في اللغة باستعمال مفهوم التابع. تسمى هذه الآلية **التجريد الإجرائي procedural abstraction** وهي آلية تسمح بها غالبية لغات البرمجة الإجرائية.



غير أن التجريد الإجرائي لا يشمل إلا جانباً واحداً من جوانب السيطرة على التطوير البرمجي.

في الحقيقة، من المفيد غالباً أن نتمكن من زيادة قوة تعبير اللغة ببناء أنماط جديدة، وذلك اعتماداً على ما يسمى بنى المعطيات المجردة **abstract data types**.



وكما سنرى لاحقاً، فإن المفهوم الذي يسمح لنا بتنفيذ **أنماط المعطيات المجردة** في C++ هو مفهوم **الصف class**.

ملاحظة:

يعود السبب في تسمية هذه الأنماط بالأنماط المجردة إلى أن مستعملها لا يعرف بالضرورة تمثيلها الداخلي. وهذه مِيْزة هامة تفيد أثناء كتابة البرامج التي تستعمل هذه الأنماط. فعندما يتعامل المبرمج مع الأنماط السابقة التعريف في لغة البرمجة، فإنه يستعمل وضوحاً أنماطاً مجردة. ذلك أنه عندما يعطي المبرمج نمطاً ما لمتحول، يصبح للمتحول بنية داخلية معرفة جيداً، وتكون العمليات المسموح بها على هذه البنية مشروحة في دليل اللغة (int, char,...) دون الإشارة عموماً إلى تمثيلها الداخلي.

وهنالك عدة أسباب تدعم العمل مع أنماط المعطيات منها:

- يعكس البرنامج العمل الذهني للمبرمج على وجهٍ أفضل، عندما نسمح لهذا الأخير بالعمل في مستوى إدراكي أعلى.
- يعزز تكرار استعمال الأنماط الوقايةَ من الأخطاء، إذ يعلن المبرمج نواياه بصراحة،

بتحديد عدد القيم والعمليات.

- يسهل استعمال أنماط المعطيات صيانة البرامج.

يتيح استعمال الأنماط المجردة للمعطيات التفكير بعمق أكبر بوظائف النمط بدلاً من بنية المعطيات الواجب تنفيذها لتحقيق هذا النمط المجرد.

وقبل الدخول في التفاصيل سنعرف كل من الغرض والصف.

الغرض

الغرض هو المفهوم الأساسي في البرمجة الغرضية التوجه. فالأغراض هي الكيانات الأساسية عند تنفيذ البرنامج. وكما رأينا في الفصل الأول، يحتل الغرض مكاناً في الذاكرة الرئيسية وتُحدد المعطياتُ المضمنَة فيه حالته **state**. ولكن الغرض يتضمن إضافة إلى حالته جميع التوابع التي تحدد سلوكه **behavior**، وهذا ما نسميه عادة الكبسلة **encapsulation.**

تسمح الكبسلةُ بجمع المعطيات والتوابع التي تنمذج كياناً ما بعضها إلى بعض، وبذلك يمكن رؤيتها ككل. تُقلص الكبسلةُ إذن تشتت المعطيات والتوابع، وتمكن المبرمج من فهم المسألة فهماً أفضل، وهذا مما يؤدي إلى تقليص تعقيدها.

الغرض = معطيات + عمليات

ومع أن جميع المعلومات تُخزن ضمن الغرض، فإن ذلك لا يعني أن بالإمكان الوصول إليها جميعها. إذ تقسم معلومات غرض ما إلى قسمين:

قسم عام Public، مرئي ويمكن الوصول إليه من خارج الغرض.

قسم خاص Private، غير مرئي ولا يمكن الوصول إليه من خارج الغرض. يمكن، بالطبع، الوصول إلى معلومات هذا القسم واستخدامها من قبل التوابع الأعضاء (**members**) في الغرض.

الصف Class

يعرفُ الصفُ مجموعةً ممكنةً من الأغراض. ينْتُج كلُّ غرضٍ، عند تنفيذ البرنامج، من استنساخ **instantiation** صف موجود. أما الصفُ فهو تمثيلٌ ساكن، جامد (في البرنامج) لمجموعة ممكنة من الأغراض المستَنسخَة.

لتوضيح الفرق بين الغرض والصف، يستَعملُ عادةً مجاز المصنع (الصف) الذي ينتج منتجات (الأغراض). وبإمكان المبرمج تحديد الطريقة التي يجب اتباعها لإنتاج الغرض (البواني في **C++**).

لا نرى في البرنامج إلا الصفوف، في حين لا يوجد عند التنفيذ إلا الأغراض. يمكن مقارنة هذا السلوك بذلك الموافق للنمط، فتعريف صف يكافئ تعريفَ نمطٍ جديد.

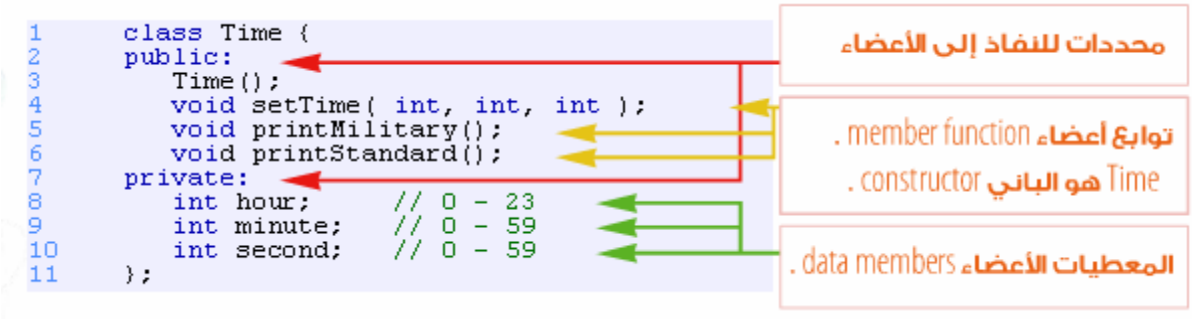
الصف هو نمط يتضمن وصفاً موحداً:

- للواصفات وهي مجموعة من المعطيات التي ترتبط فيما بينها وتكون غالباً من أنماط مختلفة (وتسمى "المعطيات الأعضاء")
- للسلوك وهو مجموعة من الطرائق (وتسمى "التوابع الأعضاء" التي تسمح بالتعامل مع المعطيات الأعضاء).

يعرف الصف باستخدام الكلمة المفتاحية **class** ، ويقع جسم الصف بين العلامتين } و {، وينتهي تعريف الصف بفاصلة منقوطة.

مثال: تنجيز بنية المعطيات المجردة Time في صف

يبين المثال التالي تعريفاً للصف **Time** :



يتوفر عموماً في الصف أربعة أنواع مختلفة من التوابع الأعضاء العامة:

البواني Constructors: تفيد في بناء أغراض من الصف وإعطائها قيماً ابتدائية على وجهٍ صحيح

الهادم Destructor: يسمح الهادم بهدم الغرض في نهاية حياته

النوافذ Accessors: تسمح بالنفاذ إلى المعطيات الأعضاء الخاصة المحتواة في الغرض

المعدلات Modificators: تسمح بتغيير حالة المعطيات الأعضاء الداخلية في الغرض

داخل تعريف كل صف، تكون ضمناً جميع الأعضاء (المعطيات والتوابع) خاصة **private**، ما لم يصرح خلاف ذلك. في حال رغبتنا في جعل المعطيات و/أو التوابع قابلة للنفاذ من الخارج، يجب أن نضعها في قسم مسبوق بالكلمة المحجوزة **public** التي تتبع بالرمز " : ". وهكذا يمكن النفاذ إلى جميع الحقول والتوابع المصرح عنها في القسم العام من قبل جميع مستخدمي الصف.

هناك كلمة أخرى محجوزة وهي **private** للدلالة على أن المعطيات والتوابع الأعضاء التي تظهر في هذا القسم لا يمكن النفاذ إليها إلا من خلال التوابع الأعضاء في الصف نفسه. من ثَم لا يحق للمستخدم النهائي للصف أن ينفذ إلى المعطيات المعرفة في القسم الخاص.

بوجهٍ عام، إذا لم ترد كلمة **private** أو **public**، فهذا يعني أن الحقول ستعامل كما لو كانت معرفة في القسم الخاص.

في تعريف الصف الواحد، يمكن أن ترد الكلمات **private** و **public** عدة مرات وبأي ترتيب. ويمتد كل قسم من الكلمة المحجوزة حتى الكلمة المحجوزة التالية أو حتى نهاية الصف.
من وجهة نظر منهجية يفضل أثناء تعريف الصف أن نُظهر أولاً التوابع والمعطيات الأعضاء العامة، ثم تُسرد المعطيات والتوابع الأعضاء الخاصة.
يجري التصريح عن **المعطيات الأعضاء** بالطريقة ذاتها التي يصرح بها عن المتحولات، عدا أن المعطيات الأعضاء **لا يمكن إعطاؤها** قيماً ابتدائية.
يمكن أن يحوي الصف **تصريحات و/أو تعريفات لتوابع**. ويجري التصريح عن هذه التوابع وتعريفها وفقاً للقواعد المتّبعة في الفصل السابق.
تستطيع التوابع الأعضاء النفاذ إلى جميع المعطيات في الصف سواء أكانت خاصة أم لا. ولا يمكن النفاذ إلى الأعضاء الخاصة **private** إلا من خلال التوابع الأعضاء العامة **public**.

تعريف التوابع الاعضاء

يغلب علينا أن نعرف التوابع الأعضاء خارج الصف أي بعد القوس الكبيرة ;{ التي تنهي تعريف الصف. وإذا عرف تابع عضو **خارج الصف**، يجب أن يسْبق معرف التابع باسم الصف الذي يصرح عن التابع. ويستعمل المؤثر " :: " لهذه الغاية. غير أنه يمكن أن يعرف التابع العضو **داخل الصف**، عندها لا نحتاج لمعامل تمييز المدى :: ولا لاسم الصف قبل اسم التابع.
إن تعريف التابع خارج الصف الذي تنتمي إليه لا يعني أي تعديل في كونه **public** أو **private**

الشكل العام لتعريف التوابع الأعضاء التي تنتمي إلى صف خارج الصف:

```
returnType ClassName::MemberFunctionName(){
    …
    }
```

معامل تمييز المدى scope resolution operator

يسمى المعامل (::) معامل تمييز المدى الثنائي **binary scope resolution operator**، وهو يحدد الصف الذي يمتلك التابع العضو (ذلك أنه من الممكن للتوابع الأعضاء في الصفوف المختلفة أن يكون لها نفس الاسم).

استخدام الأغراض في البرنامج

بعد تعريف الصف، يصبح اسم الصف اسم نمط جديد يمكن أن يستخدم كنمط في تعريف الأغراض، والمصفوفات، على النحو التالي:

```
Time sunset,              // object of type Time
arrayOfTimes[5];          // array of Time objects
```

معامل النفاذ إلى أعضاء الصف

بالطبع **لا يحق** لنا **في البرنامج** النفاذ إلى الأعضاء الخاصة أو استدعاء التوابع الخاصة. أما التوابع العامة، فيحق لنا استدعاءها ولكن باستخدام **معامل النقطة (.)**. يميز هذا المعامل طريقة استدعاء التوابع الأعضاء في صف عن طريقة استدعاء التوابع التي لاتنتمي إلى أي صف (مثل pow(x,n)، sin(x)، ...).

أما **التوابع الأعضاء في الصف**، فهي **تستطيع** النفاذ إلى جميع المعطيات في الصف سواءً أكانت خاصة أم لا، دون استخدام النقطة.

تطبيق

فيما يلي التصريح الكامل عن الصف **Time** وتوابعه:

```cpp
// Time class.
#include <iostream>

using std::cout;
using std::endl;

// Time abstract data type (ADT) definition
class Time {
public:
  Time();                          // constructor
  void setTime( int, int, int ); // set hour, minute,
  void printMilitary();            // print military time
  void printStandard();            // print standard time format
private:
  int hour;     // 0 - 23
  int minute;   // 0 - 59
  int second;   // 0 - 59
};

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time() { hour = minute = second = 0; }

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
 {
   hour = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
 }

// Print Time in military format
void Time::printMilitary()
 {
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
 }

// Print Time in standard format
void Time::printStandard()
 {
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12)
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
 }

```

لاحظ وجود :: قبل أسماء التوابع

```
1 // Driver to test simple class Time
2 int main()
3  {
4    Time t;  // instantiate object t of class Time
5
6    cout << "The initial military time is ";
7    t.printMilitary();
8    cout << "\nThe initial standard time is ";
9    t.printStandard();
10
11   t.setTime( 13, 27, 6 );
12   cout << "\n\nMilitary time after setTime is ";
13   t.printMilitary();
14   cout << "\nStandard time after setTime is ";
15   t.printStandard();
16
17   t.setTime( 99, 99, 99 ); //attempt invalid settings
18   cout << "\n\nAfter attempting invalid settings:"
19        << "\nMilitary time: ";
20   t.printMilitary();
21   cout << "\nStandard time: ";
22   t.printStandard();
23   cout << endl;
24   return 0;
25 }
```

لاحظ استدعاء التوابع باستخدام المعامل (.)

خرج البرنامج:

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

ملاحظة1: \n في الرماز السابق هي نفس **endl**

ملاحظة2: التعليمة التالية:

```
hour = ( h >= 0 && h < 24 ) ? h : 0;
```

هي طريقة ثانية لكتابة:

```
if ( h >= 0 && h < 24 ) hour=h; else hour=0;
```

ملاحظة3: تستخدم بعض المترجمات سطراً واحداً بدلاً من تكرار هذه الأسطر

```
using std::cout;
using std::endl;
```

وذلك بكتابة :

```
using namespace std;
```

فصل الواجهات عن التنجيز

عادة، يصرح عن **واجهة الصف** في ملف، ويعرف **جسم التوابع الأعضاء** تعريفاً منفصلاً في ملف آخر.

إيضاح: واجهة الصف تتضمن اسم الصف، والتعريف بالمعطيات الأعضاء والتوابع الأعضاء دون كتابة الرماز التفصيلي للتوابع.

تفيد عملية الفصل فيما يلي:

- إن فصل الواجهات عن التنجيز في الصف يسهل عمليات تعديل الصفوف ومن ثم البرامج.
- عند التصريح عن واجهة الصف **class interface** نخبر المترجم **compiler** عن واصفات الصف والعمليات المتاحة عليه.
- يصف تنجيز الصف **class implementation** كيفية الترميز الداخلي لكل عملية من عمليات الصف.
- تعرف واجهة الصف زبائن الصف بالخدمات التي يقدمها هذا الصف.
- يخفي تنجيز الصف التفاصيل عن كيفية إنجاز الصف لمهامه (التنجيز).

من الناحية العملية، يمكن تقسيم البرنامج في C++ إلى :

- **ملفات الترويسات Header files:** وتتضمن تعريف الصفوف وترويسات التوابع **prototypes function**
- **ملفات الرماز المصدري Source-code files:** وتتضمن تعريف التوابع الأعضاء

مخطط شبه الرماز Pseudocode لإنشاء ملفي الواجهة والتنجيز لكل صف

1- ننشئ ملف ترويسة للصف

2- ننشئ ملف رماز مصدري على النحو التالي:

1. نحمل ملف الترويسة لنحصل على تعريف الصف
2. نعرف التوابع الأعضاء في الصف

تطبيق على الصف Time

```
1     // time1.h
2     // Declaration of the Time class.
3     // Member functions are defined in time1.cpp
4
5     // prevent multiple inclusions of header file
6     #ifndef TIME1_H
7     #define TIME1_H
8
9     // Time abstract data type definition
10    class Time {
11    public:
12       Time();                          // constructor
13       void setTime( int, int, int ); // set hour, minute
14       void printMilitary();            // print military time
15       void printStandard();            // print standard time
16    private:
17       int hour;     // 0 – 23
18       int minute;   // 0 – 59
19       int second;   // 0 – 59
20    };
21
22    #endif
```

نستبدل (.) بالعلامة (_) في اسم الملف

```
1     // time1.cpp
2     // Member function definitions for Time class
3     #include <iostream>
4
5     using std::cout;
6
7     #include "time1.h"
8
9     // Time constructor initializes each data member to zero.
10    // Ensures all Time objects start in a consistent state
11    Time::Time() { hour = minute = second = 0; }
12
13    // Set a new Time value using military time. Perform validity
14    // checks on the data values. Set invalid values to zero.
15    void Time::setTime( int h, int m, int s )
16    {
17       hour   = ( h >= 0 && h < 24 ) ? h : 0;
18       minute = ( m >= 0 && m < 60 ) ? m : 0;
19       second = ( s >= 0 && s < 60 ) ? s : 0;
20    }
21
22    // Print Time in military format
23    void Time::printMilitary()
24    {
25       cout << ( hour < 10 ? "0" : "" ) << hour << ":"
26            << ( minute < 10 ? "0" : "" ) << minute;
27    }
28
29    // Print time in standard format
30    void Time::printStandard()
31    {
32       cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
33            << ":" << ( minute < 10 ? "0" : "" ) << minute
34            << ":" << ( second < 10 ? "0" : "" ) << second
35            << ( hour < 12 ? " AM" : " PM" );
36    }
```

يستخدم الملف المصدري #include لتحميل ملف الترويسة

يتضمن الملف المصدري تعريفات التوابع

```
1     //prog using the class Time
2     // Demonstrate errors resulting from attempts
3     // to access private class members.
4     #include <iostream>
5
6     using std::cout;
7
8     #include "time1.h"
9
10    int main()
11    {
12       Time t;
13
14       // Error: 'Time::hour' is not accessible
15       t.hour = 7;
16
17       // Error: 'Time::minute' is not accessible
18       cout << "minute = " << t.minute;
19
20       return 0;
21    }
```

محاولة تعديل hour وهو عضو من أعضاء المعطيات الخاصة private في المتحول t.

محاولة النفاذ إلى minute وهو عضو من أعضاء المعطيات الخاصة private في المتحول t.

خرج البرنامج:

```
Compiling...
Fig06_06.cpp
D:\Fig06_06.cpp(15) : error C2248: 'hour' : cannot
access private
member declared in class 'Time'
D:\Fig6_06\time1.h(18) : see declaration of 'hour'
D:\Fig06_06.cpp(18) : error C2248: 'minute' : cannot
access private
member declared in class 'Time'
D:\time1.h(19) : see declaration of 'minute'
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

البواني والهوادم

1- البواني

ما الذي يحدث عندما نصرح في البرنامج الرئيسي عن متحول (غرض) من النمط (الصف) **Time**؟

في الحقيقة، يستدعي المترجم آلياً تابعاً خاصاً من التوابع الأعضاء يسمى الباني **constructor**، فيقوم هذا التابع ببناء الغرض وإعطائه قيمة ابتدائية صحيحة.

يمكن تمييز باني الغرض عن غيره من التوابع بكونه يحمل **اسم الصف** نفسه. **لا يقبل الباني أي نمط عودة** (ولا حتى من النوع **void**)، ولا يسمح بالعمل مع عنوان الباني.

إذاً نعرف الباني كما يلي:

الباني constructor : تابع عضو خاص يقوم بإنشاء غرض من الصف وإعطاء قيمة ابتدائية للمعطيات الأعضاء في هذا الغرض

- يسمى بنفس اسم الصف
- لا يمكن أن يعيد الباني قيمة **return value**

في مثال الصف **Time**، الباني هو التابع **()Time** الذي ليس له موسطات. يقوم الباني **Time** بوضع القيمة الابتدائية **0** في المعطيات الأعضاء مما يضمن أن يكون الغرض في حالة سليمة عند إنشائه.

يمكن أن يتوفر عدة بوانٍ في الصف، وبالطبع، يختلف نموذج هذه البواني رغم كونها جميعاً **تشترك بالاسم نفسه** (راجع فقرة التحميل الزائد للتوابع في الفصل الثالث) ، كما يسمح بتعريف بوانٍ تقبل قيماً افتراضية لمحدداتها.

مثال

لديك فيما يلي واجهة الصف **Point** حيث جرى تعريف ثلاثة بواني أحدها افتراضي والآخران أحدهما يأخذ الإحداثيين **x** و **y** والآخر يأخذ الإحداثيين القطبيين للنقطة:

```
lass Point{
    public:              //  public methods
        Point () { x = 0; y = 0;}  //  a default constructor
        Point (int x0, int y0);    //  a constructor
        Point(double alpha, double r);  //  a constructor
        void move (int dx, int dy);
        void rotate (double alpha);
        int distance (Point p);
    private:
        double x, y;    // private data members
};
```

عند التصريح عن المتحولات يمكننا أن نكتب:

```
int main() {
             Point p2;        // call default constructor
             Point p1 (20,10),
                   p3 (3.14 / 4, 2.5);
```

1- البواني، الباني الافتراضي default constructor

نطلق اسم الباني الافتراضي **default constructor** في الصف على الباني الذي لا يتضمن أية محددات، أو الباني الذي تُعطى قيم افتراضية لجميع محدداته.

إذا لم يحدد المبرمج في تعريفه للصف أي بانٍ، يولّد المترجم بانياً افتراضياً. تكمن فائدة هذه العملية في كون الباني الافتراضي يقوم بانتظام باستدعاء جميع بواني المعطيات الأعضاء إن وجدت. ويعتبر هذا الأمر مفيداً عندما تكون المعطيات الأعضاء في الصف من نمط صفوف أخرى.

عندما ترغب باستدعاء الباني الافتراضي، تكتفي بالتصريح عن الغرض كما يلي:

```
Point p2;
```

إيضاح: راجع الباني الافتراضي للصف **Point** .

2- البواني، البواني ذوات المحددات constructors with arguments

يمكن أن يرغب المبرمج عند استعمال غرض من صف ما أن ينشئ الغرض بقيم ابتدائية يحددها هو. توجد بوانٍ أخرى غير الباني الافتراضي يمكنها تمرير هذه القيم كموسطات لباني الصف.

فمثلاً عند التصريح عن غرض كما يلي:

```
Point p1 (20,10),
      p3 (pi / 4, 2.5);
```

فإن هذا يعني استدعاء بان غير الباني الافتراضي.

تقويم:

إذا وجد عدة بوان تأخذ موسطات، كيف يميز المترجم أيها هو الباني المقصود؟ حدد الإجابة بما يخص تعريف النقطتين **p1** و **p3**.

3- البواني، استخدام المحددات الافتراضية Default argument في البواني

تعامل التوابع الأعضاء في الصفوف -ومن ضمنها البواني- كما يعامل أي تابع في **C++**. وعليه، يمكننا أن نعطي لبعض موسطات الباني قيماً افتراضية. وهنا يجب الانتباه للنقاط التالية:

- يعطى المحدد **argument** قيمة في ترويسة التابع الباني (في تعريف الصف).
- قد يكون عدد القيم الممررة للباني أقل من عدد المتحولات الأعضاء، عندها تعطى هذه القيم للمتحولات الموافقة من اليسار إلى اليمين، وتعطى المتحولات التي لم يبق لها قيم والواقعة على اليمين قيماً افتراضية .

تطبيق

```
1     // time2.h
2     // Declaration of the Time class.
3     // Member functions are defined in time2.cpp
4
5     // preprocessor directives that
6     // prevent multiple inclusions of header file
7     #ifndef TIME2_H
8     #define TIME2_H
9
10    // Time abstract data type definition
11    class Time {
12    public:
13       Time( int = 0, int = 0, int = 0 );  // default
14       void setTime( int, int, int ); // set hour, minute,
15       void printMilitary();          // print military time
16       void printStandard();          // print standard time
17    private:
18       int hour;     // 0 - 23
19       int minute;   // 0 - 59
20       int second;   // 0 - 59
21    };
22
23    #endif
```

لاحظ هنا أن القيم الافتراضية للمتحولات الأعضاء الثلاثة قد وضعت في ترويسة الباني. لاحاجة بنا لأسماء المتحولات الأعضاء. تطبق القيم الافتراضية بترتيب التصريح عن المتحولات الأعضاء في الصف.

```
24
25    // Demonstrating a default constructor
26    // function for class Time.
27    #include <iostream>
28
29    using std::cout;
30    using std::endl;
31
32    #include "time2.h"
33
34    int main()
35    {
36       Time t1,            // all arguments defaulted
37            t2(2),         // minute and second defaulted
38            t3(21, 34),    // second defaulted
39            t4(12, 25, 42), // all values specified
40            t5(27, 74, 99); // all bad values specified
41
42       cout << "Constructed with:\n"
43            << "all arguments defaulted:\n   ";
44       t1.printMilitary();
45       cout << "\n   ";
46       t1.printStandard();
47
48       cout << "\nhour specified; minute and second defaulted:"
49            << "\n   ";
50       t2.printMilitary();
51       cout << "\n   ";
52       t2.printStandard();
53
54       cout << "\nhour and minute specified; second defaulted:"
55            << "\n   ";
56       t3.printMilitary();
57       cout << "\n   ";
58       t3.printStandard();
59
60       cout << "\nhour, minute, and second specified:"
61            << "\n   ";
62       t4.printMilitary();
63       cout << "\n   ";
64       t4.printStandard();
65
66       cout << "\nall invalid values specified:"
67            << "\n   ";
68       t5.printMilitary();
69       cout << "\n   ";
70       t5.printStandard();
71       cout << endl;
72
73       return 0;
74    }
```

لاحظ كيفية إعطاء قيم ابتدائية للأغراض:

Constructor ObjectName (value1,value2…) ;

عند وجود نقص في عدد القيم، يعتبر المترجم كأن القيم الافتراضية موجودة في أقصى اليمين لتكمل العدد.

خرج البرنامج:

```
 OUTPUT
Constructed with:
all arguments defaulted:
   00:00
   12:00:00 AM
hour specified; minute and second defaulted:
   02:00
   2:00:00 AM
hour and minute specified; second defaulted:
   21:34
   9:34:00 PM
hour, minute, and second specified:
   12:25
   12:25:42 PM
all invalid values specified:
   00:00
   12:00:00 AM
```

عندما تخصص قيمة للساعة hour، توضع القيم الافتراضية لـ minute و second وهي 0.

4- البواني، تنجيز الباني

عند تنجيز الباني تسمح لغة C++ للمبرمج باستخدام قائمة قيم ابتدائية initilization list لإعطاء القيم الابتدائية للمعطيات الأعضاء. ويجري ذلك بكتابة الباني (الترويسة والجسم) وفق الشكل التالي:

```
ClassName::ClassName(T1 arg1…,Tm argm,…,Tn argn):data1(arg1),…, datan(argn)
{……
……}
```

حيث data1,...,datan هي أسماء الحقول في المعطيات الأعضاء وهي من الأنماط T1,...Tm,...Tn على الترتيب.

في حال استخدام قائمة القيم الابتدائية يبدأ العمل بتنفيذ العمليات الواردة في قائمة القيم الابتدائية initilization list ، قبل التعليمات الواردة في جسم التابع.

مثال: باني الصف Rectangle:

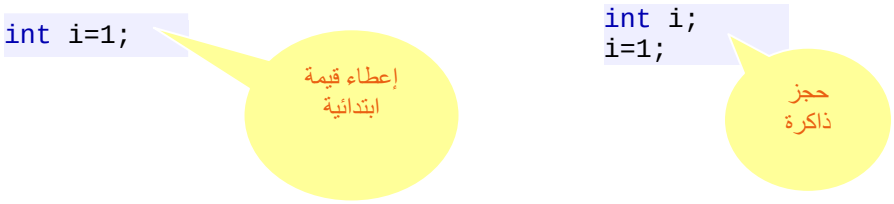
```
Rectangle:: Rectangle(double l, double w): length(l), width(w)
{ }
```

ولكن ما الهدف من إعطاء قيم ابتدائية للباني؟ وما الفرق بين إعطاء قيم ابتدائية للباني واستخدام التعليمات في جسم الباني لإعطاء القيم المرغوب بها للمعطيات الأعضاء؟

قبل الإجابة عليك أن تتذكر الحالة البسيطة التي تصرح فيها عن متحول **i** من نمط بسيط وليكن **int**.

ما الفرق بين أن نكتب: **int i=1;** أو نكتب: **int i; i=1;** ؟

بمعنى آخر ما الفرق بين حجز ذاكرة لمتحول **allocation** وإعطاء قيمة ابتدائية **initialization** ؟



إعطاء قيمة ابتدائية **initialization** : إنشاء متحول وإعطاؤه قيمة خاصة لحظة الإنشاء .

حجز ذاكرة لمتحول **allocation**: تخصيص ذاكرة للمتحول وإعطائه قيمة لاحقاً.

```
int num=0, round, position=2;
.
.
round=7;
```

إن الفرق بين استخدام قائمة القيم الابتدائية في ترويسة الباني أو التعليمات الواردة في جسم التابع يشبه تماماً الفرق بين إعطاء قيمة ابتدائية للمتحول لحظة الإنشاء و إسناد قيمة له لاحقاً.

مثال

```
class Rectangle{

public:
//default constructor
 Rectangle(double = 0.0, double = 0.0);

//compute rectangle measurements
double perimeter();
double area();

//data access function
double getLength();
double getWidth();

//data update function
void setSides(double l, double w);

private:
   double length,width;
};
Rectangle:: Rectangle(double l, double w): length(l), width(w)
 { }
```

يمكن أن يكون تنجيز هذا الباني الأخير على النحو التالي:

```
Rectangle:: Rectangle(double l, double w)
{  length=l;
 width=w };
```

تقويم:

هل هذا الباني هو بان افتراضي؟

نعم لأن موسطاته تأخذ قيماً افتراضية في تعريف الصف، كما في السطر التالي:

```
Rectangle(double = 0.0, double = 0.0);
```

في بعض الحالات كما في مثال المستطيل لا يوجد فرق بين الطريقتين، لكن هنالك حالات يجب فيها استخدام قائمة القيم الابتدائية **initilization list** عند تنجيز البواني كما هي الحال في الوراثة، والثوابت، والمرجع.

2- الهوادم

هادم الغرض هو تابع يحمل اسم الصف ذاته، مسبوقاً بالرمز "~". وكما في البواني، لا يقبل الهادم نمط عودة. إن الهادم لا يقبل محددات، ومن ثَم لا يمكن زيادة تحميله.

إن دور الهادم هو أن يقوم بجميع أعمال الصيانة اللازمة بعد أن ينتهي "وجود" الغرض في ذاكرة الحاسوب. وهذا العمل يهدف تحديداً إلى استرداد الذاكرة **المحجوزة ديناميكياً** للغرض الذي سوف يختفي.

إيضاح: سنتكلم لاحقاً عن الحجز الديناميكي للذاكرة في فصل :

"المؤشرات **pointers** والإدارة الديناميكية للذاكرة **dynamic memory management**"

عند نهاية تنفيذ كتلة، يستدعي نظام البرمجة C++ هادم الغرض آلياً، وذلك لكل متحول من نمط صف معرف في هذه الكتلة، كما يستدعي النظام الهادم للتخلص من جميع الأغراض المؤقتة التي أنشأها دون علم المبرمج. ويسمح باستدعاء هادم الغرض صراحةً (في حالات نادرة) وذلك على نقيض البواني.

أمثلة عن صفوف

الصف Person:

```
class Person{
   char name[20];
   int yearOfBirth;

public:
   void displayDetails()
{
           cout << name << " born in   "
              <<yearOfBirth << endl;
}
     //…..
};
```

إن عدم وجود الكلمة **private** أو **public**

يعني ضمنياً **private**

الصف Creature:

لاحظ وجود طريقتين لتعريف توابع الصف:

1- تعرف التوابع كاملة داخل الصف ويستخدم هذا الأسلوب إذا كانت تعريف التوابع قصيراً.

2- تعرف التوابع خارج تعريف الصف ويستخدم المعامل :: لهذا الغرض

الطريقة الأولى:

```
class Creature
   {
 private:
   int yearOfBirth;
 public:
   Creature()
    {
      yearOfBirth = 1970;
      cout << “Hello.”;
    }
Creature(int year){
      yearOfBirth = year;
      }
Creature(Creature & otherCreature){
      yearOfBirth=
          otherCreature.getYearOfBirth();
}

void setYearOfBirth(int year)
{
            yearOfBirth = year;
            }
   int getYearOfBirth()
{
            return yearOfBirth;
            }
};
```

باني افتراضي.

باني النسخ لانه يستخدم للحصول على غرض ثان من نفس النمط يحتوي نفس المعطيات.

الطريقة الثانية:

```
class Creature {
private:
   int yearOfBirth;
public:
   Creature();
   Creature(int year);
   Creature(Creature & otherCreature);
   void setYearOfBirth(int year);
   int getYearOfBirth();
};
Creature:: Creature()
   {
      yearOfBirth = 1970;
      cout << “Hello.”;
}
 Creature::  Creature(int year){
      yearOfBirth = year;
      }
  Creature:: Creature(Creature& otherCreature){
      yearOfBirth= otherCreature.getYearOfBirth();
      }

void Creature::setYearOfBirth(int year){
            yearOfBirth = year;
            }
int Creature::getYearOfBirth(){
            return yearOfBirth;
            }
```

باني افتراضي.

باني النسخ لانه يستخدم للحصول على غرض ثان من نفس النمط يحتوي نفس المعطيات.

باني افتراضي.

يسمى هذا الباني باني النسخ لأنه يستخدم للحصول على غرض ثان من نفس النمط يحتوي نفس المعطيات. فمثلاً لو بنينا الغرض MyDog كما يلي: Creature myDog(1995); يمكننا باستخدام باني النسخ الحصول على غرض له نفس قيم المعطيات (أي نفس القيمة للعضو yearOfBirth) كما يلي:
Creature myCat(myDog);

الأغرض الثابتة

سبق لك أن تعاملت مع المتحولات الثابتة في **C++** كما في التعريف:

```
const double pi=3.14;
```

على غرار هذا التعريف يمكنك بناء أغراض ثابتة باستخدام الكلمة المفتاحية **const**. إن استخدام **const** يحدد أن الغرض غير قابل للتعديل، بحيث أن أية محاولة لتعديل الغرض تؤدي إلى خطأ نحوي.

مثال

```
const Time noon( 12, 0, 0);
```

تصريح عن الغرض الثابت **noon** من الصف **Time** وإعطاؤه القيمة 12 ظهراً.

التوابع الأعضاء الثابتة

إن وجود أغراض ثابتة **const** يتطلب تعريف توابع أعضاء ثابتة **const** ، إذ أنه لايمكن لتابع عضو أن يتعامل مع غرض ثابت ما لم يكن هذا التابع ثابتاً.

- لاتستطيع التوابع الثابتة تعديل الغرض
- تضاف كلمة **const** في ترويسة التابع وفي تنجيزه

في الترويسة:

```
ReturnType FunctionName(param1,param2…) const;
```

في التنجيز:

```
ReturnType FunctionName(param1,param2…) const {….};
```

مثال

```
int A::getValue( ) const
              {return privateDataMember};
```

تطبيق على الصف Time

```
1     // time5.h
2     // Declaration of the class Time.
3     // Member functions defined in time5.cpp
4     #ifndef TIME5_H
5     #define TIME5_H
6
7     class Time {
8     public:
9        Time( int = 0, int = 0, int = 0 );  // default
10
11       // set functions
12       void setTime( int, int, int );  // set time
13       void setHour( int );     // set hour
14       void setMinute( int );   // set minute
15       void setSecond( int );   // set second
16
17       // get functions (normally declared const)
18       int getHour() const;     // return hour
19       int getMinute() const;   // return minute
20       int getSecond() const;   // return second
21
22       // print functions (normally declared const)
23       void printMilitary() const;  // print military time
24       void printStandard();        // print standard time
25    private:
26       int hour;              // 0 - 23
27       int minute;            // 0 - 59
28       int second;            // 0 - 59
29    };
30
31    #endif
```

```
32    // time5.cpp
33    // Member function definitions for Time class.
34    #include <iostream>
35
36    using std::cout;
37
38    #include "time5.h"
39
40    // Constructor function to initialize private data.
41    // Default values are 0 (see class definition).
42    Time::Time( int hr, int min, int sec )
43       { setTime( hr, min, sec ); }
44
45    // Set the values of hour, minute, and second.
46    void Time::setTime( int h, int m, int s )
47    {
48       setHour( h );
49       setMinute( m );
50       setSecond( s );
51    }
52
53    // Set the hour value
54    void Time::setHour( int h )
55       { hour = ( h >= 0 && h < 24 ) ? h : 0; }
56
57    // Set the minute value
58    void Time::setMinute( int m )
59       { minute = ( m >= 0 && m < 60 ) ? m : 0; }
60
61    // Set the second value
62    void Time::setSecond( int s )
63       { second = ( s >= 0 && s < 60 ) ? s : 0; }
64
65    // Get the hour value
66    int Time::getHour() const { return hour; }
67
68    // Get the minute value
69    int Time::getMinute() const { return minute; }
70
71    // Get the second value
72    int Time::getSecond() const { return second; }
73
```

إن الباني ليس تابعاً ثابتاً لكنه يمكن أن يستدعى للحصول على أغراض ثابتة.

لاحظ استعمال الكلمة المفتاحية const في تعريف التابع وترويسته

لايمكن للتوابع غير الثابتة functions non-const

استخدام أغراض ثابتة وإن كانت لاتقوم بتعديلها مثل printStandard.

```
74    // Display military format time: HH:MM
75    void Time::printMilitary() const
76    {
77       cout << ( hour < 10 ? "0" : "" ) << hour << ":"
78            << ( minute < 10 ? "0" : "" ) << minute;
79    }
80
81    // Display standard format time: HH:MM:SS AM (or PM)
82    void Time::printStandard()  // should be const
83    {
84       cout << ( ( hour == 12 ) ? 12 : hour % 12 ) << ":"
85            << ( minute < 10 ? "0" : "" ) << minute << ":"
86            << ( second < 10 ? "0" : "" ) << second
87            << ( hour < 12 ? " AM" : " PM" );
88    }
89
90    // Attempting to access a const object with
91    // non-const member functions.
92    #include "time5.h"
93
94    int main()
95    {
96       Time wakeUp( 6, 45, 0 );       // non-constant object
97       const Time noon( 12, 0, 0 );   // constant object
98
99                                 // MEMBER FUNCTION   OBJECT
100      wakeUp.setHour( 18 );     // non-const         non-const
101
102      noon.setHour( 12 );       // non-const         const
103
104      wakeUp.getHour();         // const             non-const
105
106      noon.getMinute();         // const             const
107      noon.printMilitary();     // const             const
108      noon.printStandard();     // non-const         const
109      return 0;
110   }
```

أخطاء سيولدها المترجم.

خرج البرنامج:

```
Compiling...
Fig07_01.cpp
d:fig07_01.cpp(14) : error C2662: 'setHour' : cannot
convert 'this' pointer from 'const class Time' to
'class Time &'
Conversion loses qualifiers
d:\fig07_01.cpp(20) : error C2662: 'printStandard' :
cannot convert 'this' pointer from 'const class Time'
to 'class Time &'
Conversion loses qualifiers
Time5.cpp
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

الصفوف المركبة

رأيت حتى الآن الصفوف التي تكون أنماط معطياتها الأعضاء أنماطاً بسيطة. ولما كنا نستخدم الأغراض للتوسع في الأنماط، فيمكننا بناء صفوف تكون أنماط معطياتها الأعضاء صفوفاً أيضاً. في هذه الحالة:

- يمتلك الصف أعضاء هي أغراض من صفوف أخرى.
- يسمى الصف الذي ينتمي إليه الغرض العضو صفاً مزوداً **supplier**.
- يسمى الصف الذي يحتوي الأغراض الأعضاء صفاً زبوناً **client** أو صفاً مضيفاً **host**.

بناء الأغراض المركبة

ينجز الباني في الصفوف المركبة باستخدام قائمة القيم الابتدائية حصراً. ولكن عند استدعاء الباني للحصول على غرض مركب، كيف تبنى الأغراض المركبة؟ ما الذي يبدأ بناؤه الغرض المركب أم أعضاؤه الأغراض التي تحتاج بدورها إلى بان؟ في الواقع ، تبنى الأغراض الأعضاء قبل الغرض المضيف، وذلك بترتيب التصريح عنها في الصف لا بترتيب قائمة القيم الابتدائية للأعضاء في الباني. أما عند الهدم، فتهدم الأغراض المضيفة، ثم تهدم الأغراض الأعضاء:

- تستدعى البواني من الداخل للخارج.

تستدعى الهوادم من الخارج للداخل.

مثال

الصف **Employee** يتضمن حقول من النمط **Date**

```
class Employee    // maintain employee information
{
      private:
        // String (ID) and Date are supplier classes
        String name;                // employee ID (Supplier)
        Date birthDate;             // date of birth (Supplier)
         Date hireDate;             // date of hire (Supplier)
      public:
            . . .
};
```

```
Employee::Employee( char *fname ,
                    int bmonth, int bday, int byear,
                    int hmonth, int hday, int hyear)
                  : birthDate( bmonth, bday, byear),
                    hireDate( hmonth, hday, hyear)
```

مثال

```
1     // date1.h
2     // Declaration of the Date class.
3     // Member functions defined in date1.cpp
4     #ifndef DATE1_H
5     #define DATE1_H
6
7     class Date {
8     public:
9        Date( int = 1, int = 1, int = 1900 ); // default constructor
10       void print() const;  // print date in month/day/year format
11       ~Date();  // provided to confirm destruction order
12    private:
13       int month;  // 1-12
14       int day;    // 1-31 based on month
15       int year;   // any year
16
17       // utility function to test proper day for month and year
18       int checkDay( int );
19    };
20
21    #endif
```

```
22    // date1.cpp
23    // Member function definitions for Date class.
24    #include <iostream>
25
26    using std::cout;
27    using std::endl;
28
29    #include "date1.h"
30
31    // Constructor: Confirm proper value for month;
32    // call utility function checkDay to confirm proper
33    // value for day.
34    Date::Date( int mn, int dy, int yr )
35    {
36       if ( mn > 0 && mn <= 12 )       // validate the month
37          month = mn;
38       else {
39          month = 1;
40          cout << "Month " << mn << " invalid. Set to month 1.\n";
41       }
42
43       year = yr;                      // should validate yr
44       day = checkDay( dy );           // validate the day
45
46       cout << "Date object constructor for date ";
47       print();         // interesting: a print with no
48       cout << endl;
49    }
50
51    // Print Date object in form  month/day/year
52    void Date::print() const
53       { cout << month << '/' << day << '/' << year; }
54
55    // Destructor: provided to confirm destruction order
56    Date::~Date()
57    {
58       cout << "Date object destructor for date ";
59       print();
60       cout << endl;
61    }
62
63    // Utility function to confirm proper day value
64    // based on month and year.
65    // Is the year 2000 a leap year?
66    int Date::checkDay( int testDay )
67    {
68       static const int daysPerMonth[ 13 ] =
69          {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
70
71       if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
72          return testDay;
73
74       if ( month == 2 &&      // February: Check for leap
75            testDay == 29 &&
76            ( year % 400 == 0 ||
77             ( year % 4 == 0 && year % 100 != 0 ) ) )
78          return testDay;
```

```
79
80       cout << "Day " << testDay << " invalid. Set to day 1.\n";
81
82       return 1;  // leave object in consistent state if bad
83    }
84    // emply1.h
85    // Declaration of the Employee class.
86    // Member functions defined in emply1.cpp
87    #ifndef EMPLY1_H
88    #define EMPLY1_H
89
90    #include "date1.h"
91
92    class Employee {
93    public:
94       Employee( char *, char *, int, int, int, int, int, int);
95       void print() const;
96       ~Employee();  // provided to confirm destruction order
97    private:
98       char firstName[ 25 ];
99       char lastName[ 25 ];
100      const Date birthDate;
101      const Date hireDate;
102   };
103
104   #endif
105   // emply1.cpp
106   // Member function definitions for Employee class.
107   #include <iostream>
108
109   using std::cout;
110   using std::endl;
111
112   #include <cstring>
113   #include "emply1.h"
114   #include "date1.h"
115
116   Employee::Employee( char *fname, char *lname,
117                       int bmonth, int bday, int byear,
118                       int hmonth, int hday, int hyear )
119      : birthDate( bmonth, bday, byear ),
120        hireDate( hmonth, hday, hyear )
121   {
122      // copy fname into firstName and be sure that it fits
123      int length = strlen( fname );
124      length = ( length < 25 ? length : 24 );
125      strncpy( firstName, fname, length );
126      firstName[ length ] = '\0';
127
128      // copy lname into lastName and be sure that it fits
129      length = strlen( lname );
130      length = ( length < 25 ? length : 24 );
131      strncpy( lastName, lname, length );
132      lastName[ length ] = '\0';
133
134      cout << "Employee object constructor: "
135           << firstName << ' ' << lastName << endl;
```

```
136   }
137
138   void Employee::print() const
139   {
140      cout << lastName << ", " << firstName << "\nHired: ";
141      hireDate.print();
142      cout << "  Birth date: ";
143      birthDate.print();
144      cout << endl;
145   }
146
147   // Destructor: provided to confirm destruction order
148   Employee::~Employee()
149   {
150      cout << "Employee object destructor: "
151           << lastName << ", " << firstName << endl;
152   }
153
154   // Demonstrating composition: an object with member objects.
155   #include <iostream>
156
157   using std::cout;
158   using std::endl;
159
160   #include "emply1.h"
161
162   int main()
163   {
164      Employee e( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );
165
166      cout << '\n';
167      e.print();
168
169      cout << "\nTest Date constructor with invalid values:\n";
170      Date d( 14, 35, 1994 );  // invalid Date values
171      cout << endl;
172      return 0;
173   }
```

التابع print هو تابع ثابت const وسيطبع كلما جرى بناء غرض Date أو هدمه. يستطيع طباعة غرض ثابت لأنه تابع ثابت.

نحمل فقط emply.h لأن هذا الملف يحمل بدوره date.h .

خرج البرنامج:

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Jones

Jones, Bob
Hired: 3/12/1988  Birth date: 7/24/1949

Test Date constructor with invalid values:
Month 14 invalid. Set to month 1.
Day 35 invalid. Set to day 1.
Date object constructor for date 1/1/1994

Date object destructor for date 1/1/1994
Employee object destructor: Jones, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
```

تدريب:

ورد في المثال السابق استخدام بعض التوابع الخاصة بالتعامل مع مصفوفات من النمط **char** مثل:**strlen** لتحديد طول المصفوفة و **strncpy** لنسخ مصفوفة على مصفوفة أخرى. أعد كتابة المثال والصفوف باستخدام النمط **string**.

التوابع الصديقة friend function والصفوف الصديقة friend class

مر معك أن التوابع الأعضاء هي فقط التي يحق لها النفاذ إلى المعطيات الأعضاء. وأن كبسلتها معاً ضمن الصف هو الذي يؤمن خاصية إخفاء المعلومات هذه.

غير أنه في بعض الحالات نجد حاجة لخرق هذه القاعدة وبشروط خاصة جداً. نستخدم لهذا الغرض التوابع والصفوف الصديقة.

يمكن أن تنفذ التوابع الصديقة والصفوف الصديقة إلى الأعضاء الخاصة **private** و المحمية **protected** في صف آخر. ولا تكون التوابع الصديقة توابع أعضاء في الصف، كما أنها تعرف خارج مدى الصف.

خصائص الصداقة

- الصداقة تمنح ولا تؤخذ
- الصداقة ليست علاقة متناظرة (إذا كان B صديق A فليس بالضرورة أن يكون A صديق B)
- ليست علاقة متعدية (إذا كان A صديق B، B صديق C فليس بالضرورة أن يكون A صديق C)

التصريح عن صديق friend، التوابع الصديقة

- تضاف الكلمة **friend** قبل ترويسة التابع في الصف الذي يمنح الصداقة.

أي أننا نكتب في الصف الذي يمنح الصداقة لتابع ما تصريحاً عن هذه الصداقة على النحو التالي:

```
friend int myFunction(int x);
```

- تضاف الكلمة **friend** قبل كلمة **class** في الصف الذي يمنح الصداقة.
- إذا كان الصف **ClassOne** يمنح الصداقة لـ **ClassTwo** يكون لدينا مايلي:

```
friend class ClassTwo;
```

في تعريف الصف **ClassOne**

مثال

```
1
2     // Friends can access private members of
3     #include <iostream>
4
5     using std::cout;
6     using std::endl;
7
8     // Modified Count class
9     class Count {
10       friend void setX( Count &, int ); // friend declaration
11    public:
12       Count() { x = 0; }                // constructor
13       void print() const { cout << x << endl; }  // output
14    private:
15       int x;  // data member
16    };
17
18    // Can modify private data of Count because
19    // setX is declared as a friend function of Count
20    void setX( Count &c, int val )
21    {
22       c.x = val;  // legal: setX is a friend of Count
23    }
24
25    int main()
26    {
27       Count counter;
28
29       cout << "counter.x after instantiation: ";
30       counter.print();
31       cout << "counter.x after call to setX friend function:
";
32       setX( counter, 8 );  // set x with a friend
33       counter.print();
34       return 0;
35    }
```

setX هو friend للصف Count (يمكنه النفاذ للأعضاء الخاصة في Count)

setX يعرف بشكل عادي وهو ليس عضواً في Count)

يسمح بتعديل متحول private.

خرج البرنامج:

```
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```

مثال

```
1
2     // Non-friend/non-member functions cannot access
3     // private data of a class.
4     #include <iostream>
5
6     using std::cout;
7     using std::endl;
8
9     // Modified Count class
10    class Count {
11    public:
12       Count() { x = 0; }                    // constructor
13       void print() const { cout << x << endl; }  // output
14    private:
15       int x;  // data member
16    };
17
18    // Function tries to modify private data of Count,
19    // but cannot because it is not a friend of Count.
20    void cannotSetX( Count &c, int val )
21    {
22       c.x = val;  // ERROR: 'Count::x' is not accessible
23    }
24
25    int main()
26    {
27       Count counter;
28
29       cannotSetX( counter, 3 ); // cannotSetX is not a friend
30       return 0;
```

cannotSetX ليس تابعاً صديقاً. لا يمكنه النفاذ للأعضاء الخاصة في Count

cannotSetX يحاول تعديل متحول خاص...

(private:
int x;)

خطأ متوقع للمترجم

cannot access private data

```
Compiling...
Fig07_06.cpp
D:\books\2000\cpphtp3\examples\Ch07\Fig07_06\Fig07_06.c
pp(22) :
   error C2248: 'x' : cannot access private member
declared in
   class 'Count'
        D:\books\2000\cpphtp3\examples\Ch07\Fig07_06\
        Fig07_06.cpp(15) : see declaration of 'x'
Error executing cl.exe.

test.exe - 1 error(s), 0 warning(s)
```

زيادة تحميل المؤثرات

قد يكون استدعاء التوابع مزعجاً في بعض أنواع الصفوف لاسيما الرياضية. فلو فرضنا وجود تابع لجمع عددين وتابع لإسناد قيمة إلى متحول، فلا بد أنك تلاحظ الفرق في وضوح العملية المطلوبة بين السطرين التاليين:

```
a = 2 + 3;
assign(a, add(2,3));
```

وفي **C++** مع تعريف أنماط جديدة، نرغب لو أننا نستطيع
استخدام الرموز المعروفة في اللغة لتحقيق بعض العمليات بمعناها المألوف على الأغراض التي تنتمي للصفوف الجديدة، كجمع عددين عقديين (بعد التصريح عن الصف الذي يعرف الأعداد العقدية) و جمع مصفوفتين من الأعداد الصحيحة (بعد التصريح عن الصف الذي يعرف مصفوفات الأعداد الصحيحة).

سنتناول المؤثرات الأصيلة في **C++** إذاً (**C++'s built-in operators**) للتعامل مع هذا النوع من الصفوف.

تتوفر في **C++** إمكانات تسمح للمبرمج بزيادة تحميل بعض المؤثرات المعرفة سابقاً. وفيما يلي قائمة المؤثرات التي يمكن زيادة تحميلها في **C++**.

+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	<-	[ ]	( )	new	Delete
new []	Delete[]						

يمكن زيادة تحميل المؤثرات "+" و "-" و "*" و "&" بشكلها الأحادي أو الثنائي. أما المؤثرات التي لا يمكن زيادة تحميلها فهي:

::	.*	.	?:	sizeof

ولئن كانت زيادة تحميل المؤثرات ممكنة، فإنه لا يمكن تغيير قواعد الأولوية الخاصة بها. كذلك الأمر فيما يتعلق بالخاصة التجميعية للمؤثرات، وبعدد المعاملات الخاصة بها إذ لا يمكن تعديلها. ثُم إنه لا يمكن بناء مؤثرات جديدة.

كيفية زيادة تحميل المؤثرات وشروطها

- اكتب تعريف التابع بشكل عادي
- يتألف اسم التابع من الكلمة المفتاحية **operator** يليها رمز العملية التي يجري زيادة تحميلها فمثلاً لزيادة تحميل مؤثر الجمع (+) نعرف التابع **operator+**
- لايمكن اختراع مؤثرات جديدة بل تُستخدم المؤثرات المتوفرة فقط
- إن محاولة زيادة تحميل المؤثرات للأنماط الأصيلة في اللغة يؤدي إلى خطأ نحوي **syntax error** (لايمكنك أن تعيد تعريف عملية جمع عددين صحيحين).
- لايمكن تغيير عدد الأطراف التي يؤثر فيها المؤثر
- يمكن أن تكون توابع المؤثرات توابع أعضاء أو توابع غير أعضاء في الصف

توابع المؤثرات أعضاء في الصف أم توابع صديقة؟

- توابع المؤثرات كتوابع أعضاء
  - يجب أن يكون العنصر الأيسر غرضاً (أو مرجعاً على غرض) من الصف.
  - إذا كان العنصر الأيسر من نمط آخر، فيجب أن لا يكون تابع المؤثر تابعاً عضواً
  - في حال لم يكن تابع المؤثر عضواً، فيجب أن يكون صديقاً **friend** وذلك في حال النفاذ المباشر إلى معطيات خاصة أو محمية في هذا الصف.

- تسمح توابع المؤثرات التي يجري زيادة تحميلها والتي لم تعتبر توابع أعضاء بجعل المؤثرات تبديلية.

```
HugeInteger bigInteger;
int integer;
bigInteger = integer + bigInteger; //or
bigInteger = biginteger + integer;
```

زيادة تحميل المؤثرين الخاصين بالإقحام والاستخراج في الفئوات

يجب أن يمتلك التابعان << و>> عند زيادة تحميلهما طرفاً أيسر من نمط **istream &** و **ostream &** على الترتيب .

- يجب أن لايكون هذين التابعين عضوين في الصف (لأن الطرف الأيسر ليس غرضاً من الصف).
- يجب أن يكون هذان التابعان صديقين للصف إذا كانا يتضمنان نفاذاً إلى المعطيات الخاصة.

تطبيق

```
1
2     // Overloading the stream-insertion and
3     // stream-extraction operators.
4     #include <iostream>
5
6     using std::cout;
7     using std::cin;
8     using std::endl;
9     using std::ostream;
10    using std::istream;
11
12    #include <iomanip>
13
14    using std::setw;
15
16    class PhoneNumber {
17       friend ostream &operator<<( ostream&, const PhoneNumber &);
18       friend istream &operator>>( istream&, PhoneNumber & );
19
20    private:
21       char areaCode[ 4 ];  // 3-digit area code and null
22       char exchange[ 4 ];  // 3-digit exchange and null
23       char line[ 5 ];      // 4-digit line and null
24    };
25
26    // Overloaded stream-insertion operator (cannot be
27    // a member function if we would like to invoke it with
28    // cout << somePhoneNumber;).
29    ostream &operator<<( ostream &output, const PhoneNumber &num)
30    {
31       output << "(" << num.areaCode << ") "
32              << num.exchange << "-" << num.line;
33       return output;     // enables cout << a << b << c;
34    }
35
36    istream &operator>>( istream &input, PhoneNumber &num )
37    {
```

```
38       input.ignore();                     // skip (
39       input >> setw( 4 ) >> num.areaCode; // input area code
40       input.ignore( 2 );                  // skip ) and space
41       input >> setw( 4 ) >> num.exchange; // input exchange
42       input.ignore();                     // skip dash (-)
43       input >> setw( 5 ) >> num.line;     // input line
44       return input;      // enables cin >> a >> b >> c;
45    }
46
47    int main()
48    {
49       PhoneNumber phone; // create object phone
50
51       cout << "Enter phone number in the form (123) 456-7890:\n”;
52
53       // cin >> phone invokes operator>> function by
54       // issuing the call operator>>( cin, phone ).
55       cin >> phone;
56
57       // cout << phone invokes operator<< function by
58       // issuing the call operator<<( cout, phone ).
59       cout << "The phone number entered was: " << phone <<endl;
60       return 0;
61    }
```

خرج البرنامج:

```
Enter phone number in the form (123) 456-7890:
(800) 555-1212
The phone number entered was: (800) 555-1212
```

تدريب:

1. يطلب بناء صف يمثل النمط عدد كسري **Rational** لإجراء حسابات على الأعداد الكسرية. اكتب تعريف الصف وقم بزيادة مؤثري الجمع والضرب و المؤثرين << و >>. اكتب برنامجاً لاختبار الصف.
2. يطلب بناء صف يمثل النمط عدد عقدي **Complex** لإجراء حسابات على الأعداد العقدية. اكتب تعريف الصف وقم بزيادة مؤثري الجمع والضرب و المؤثرين << و >>. اكتب برنامجاً لاختبار الصف.
3. يطلب بناء صف يمثل النمط مستطيل **Rectangle** الذي مر ذكره في المثال الوارد في الفقرة تنجيز الباني بحيث يمكن القيام بالعمليات الواردة في المثال: حساب المحيط، حساب المساحة، الحصول على الطول الحصول على العرض، تعديل البعدين. كما يطلب تزويد الصف بتوابع << و >> . يطلب كتابة التعريف والتنجيز في ملفين منفصلين.

الفصل الخامس

المؤشرات pointers والإدارة الديناميكية للذاكرة dynamic memory management

هدف الفصل

1. التعرف إلى مفهوم المؤشر
2. التعامل مع المتحولات في الذاكرة من خلال عناوينها
3. فهم علاقة المصفوفات بالمؤشرات
4. الحجز الديناميكي للذاكرة
5. تحرير الذاكرة المحجوزة ديناميكياً

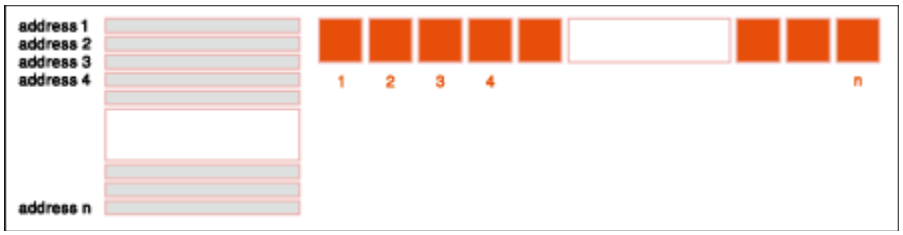
1- مقدمة

سنتناول في هذا الفصل أحد المواضيع الهامة في C++ وهو المؤشرات. وقد تطرقنا حتى الآن إلى كيفية التصريح عن متحولات من الأنماط البسيطة وعن أغراض من الصفوف التي يعرفها المبرمج دون أن نعير أي اهتمام لمواضع هذه المتحولات والأغراض في ذاكرة الحاسوب . سنركز اهتمامنا في هذا الفصل على عمليات حجز المتحولات في الذاكرة وكيفية التعامل معها.

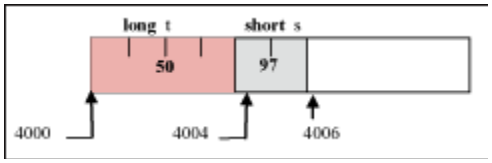
2- المؤشرات

لاستيعاب مفهوم المؤشر، لابد لنا من عرض كيفية التخزين الداخلي للمعطيات في الذاكرة. تتألف الذاكرة من تتالي خانات يتسع كل منها لثمانية بتّات، وتسمى كل خانة من هذه الخانات بايت byte. يعرف كل بايت بالعنوان address الذي يوجد فيه في الذاكرة وبمحتواه من المعطيات.

يمكن التعامل مع الذاكرة وكأنها مصفوفة من البايتات. ويمكن رسمها عمودياً أو أفقياً كما يلي:



يشغل كل متحول أو غرض في C++ بايتاً أو أكثر حسب الحجم المحدد للنمط الذي ينتمي إليه . فعلى سبيل المثال، يشغل متحول من النمط long أربعة بايتات، في حين يشغل متحول من النمط char بايتاً واحداً فقط. نسمي عنوان البايت الأول الذي يخزن فيه المتحول أو الغرض عنوان المتحول. فمثلاً لدينا متحول t من النمط long في العنوان 4000 ومتحول s من النمط short في العنوان 4004.



تمتلك لغة C++ نمطاً يسمى المؤشر pointer ، وتخزن متحولات هذا النمط عناوين المعطيات في الذاكرة. وقد سمي المؤشر مؤشراً لأنه "يشير إلى" العنصر. وحسبما ذكرنا فإن الذاكرة تحتوي جميع أنواع المتحولات وهي بالتالي تحتوي على مواضع من النمط مؤشر، أي أن كل متحول من هذا النمط سيحتوي على عنوان الذاكرة التي يشير إليها وعلى عناوين لذواكر أخرى. (في مثالنا سنجد في الذاكرة القيمة 4000 أو 4004).

3- التصريح عن متحول من النمط مؤشر

يجري التصريح عن مؤشر بوضع الرمز "*" قبل اسم المتحول في سطر تصريح اعتيادي.

مثال

```
long *ptr;              //pointer to a long integer
```

الشكل العام للتصريح عن متحول من نمط مؤشر:

```
Type *ptr;            //pointer to a variable from the type Type
```

أمثلة

```
int *intPtr;
char *charPtr;
```

4- إسناد القيم إلى المؤشرات

جرى تأمين النمط مؤشر في اللغة لكي نتمكن من خزن عناوين الذواكر في متحولات هذا النمط. غير أن التصريح عن متحول بالشكل التالي:

```
Type *ptr;
```

لايضع أية قيمة في المتحول ptr وبالتالي فإن هذا المتحول من نمط مؤشر لايشير إلى أية خانة في الذاكرة. ولإعطاء قيمة لهذا المتحول، يجب أن نسند إليه عنوان ذاكرة موجودة. ولكن كيف نحصل على عنوان ذاكرة موجودة؟

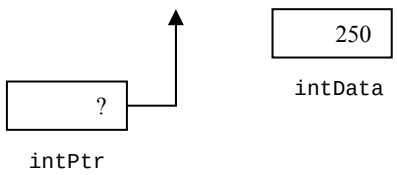
يوجد في C++ مؤثر يسمى مؤثر العنوان ورمزه "&" يعطي عند استخدامه مع اسم المتحول عنوان هذا المتحول.

لنأخذ المثال التالي:

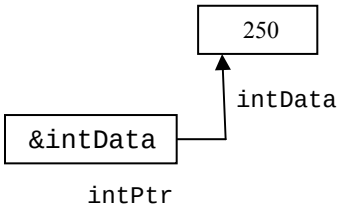
ليكن المتحول intData المصرح عنه كعدد صحيح بقيمة ابتدائية قدرها 250، وليكن المتحول intPtr المصرح عنه كمؤشر على عدد صحيح. كما رأينا لايشير المؤشر عند تصريحه إلى أي متحول، باستخدام المؤثر "&" نحصل على عنوان intData ، فنسند هذا العنوان إلى intPtr. بعد تعليمة الإسناد هذه يكون لدينا المؤشر intPtr الذي يشير إلى المتحول intData.

```
int intData=250, *intPtr;  //pointer is uninitialized
```

بعد تنفيذ هذه التعليمة يكون الوضع كالتالي:



```
intPtr= &intData;  //&intData is the address of intData
```



تدريب:

ماعلاقة & بـ * ؟ نفذ البرنامج التالي وتحقق من العلاقة:

```
// Using the & and * operators.
#include <iostream>
using namespace std;

int main()
{
   int a; // a is an integer
   int *aPtr; // aPtr is an int * -- pointer to an integer

   a = 7; // assigned 7 to a
   aPtr = &a; // assign the address of a to aPtr

   cout << "The address of a is " << &a
        <<”\nThe value of aPtr is " << aPtr;
   cout << "\n\nThe value of a is " << a
        <<”\nThe value of *aPtr is " << *aPtr;
   cout << "\n\nShowing that * and & are inverses of"
        <<”each other.\n&*aPtr = " << &*aPtr
        << “\n*&aPtr = " << *&aPtr << endl;
   return 0;
}
```

4-1- النفاذ إلى المعطيات باستخدام المؤشرات

لاحظ هنا أنه يمكن الوصول إلى المعطيات المخزنة في المتحول intData من طريقين:

1. باستخدام اسم المتحول intData:

```
cout<< intData;
```

2. باستخدام المؤشر intPtr:

للنفاذ إلى المعطيات المخزنة في ذاكرة باستخدام عنوان هذه الذاكرة نستخدم المؤشر "*". إن وضع هذا المؤثر قبل اسم المؤشر في تعليمة إسناد أو قراءة أو إظهار أو أية صيغة حسابية أو منطقية يعني أننا نريد الحصول على محتوى الذاكرة التي يشير إليها المؤشر، وهكذا فـإن intPtr* هو محتوى الذاكرة التي يشير إليها المؤشر intPtr:

```
cout<< *intPtr;  //output: 250
*intPtr=300;     //assign 300 to the memory pointed to by intPtr
cout<< intData;  //output: 300
```

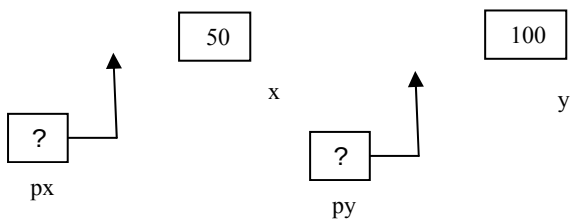
يسمى المؤشر * مؤثر النفاذ غير المباشر لانه يتيح لنا النفاذ إلى المتحول نفاذاً غير مباشر وإنما باستخدام المؤشر الذي يشير إليه.

تمارين:

ارسم الأشكال الموافقة لتنفيذ التعليمات التالية:

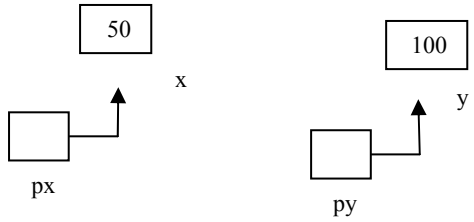
```
int x=50, y=100,*px, *py;
```

الحل:



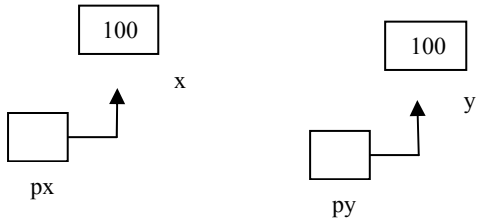
```
px=&x;
py=&y;
```

الحل:



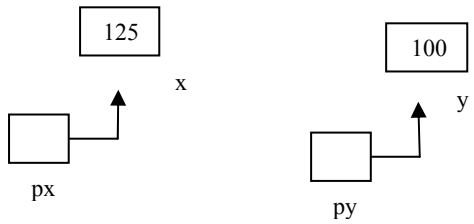

```
x=*py;
```

الحل:



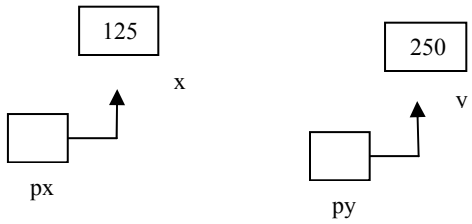

```
*px=y+25;
```

الحل:



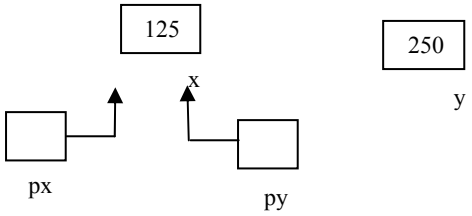
```
*py=*px*2;
```

الحل:




```
py=px;
```

الحل:




```
cout<<*px<<"  "<<*py;   // output : 125   125
cout<<*py<<"  "<<y;     // output : 125   250
```

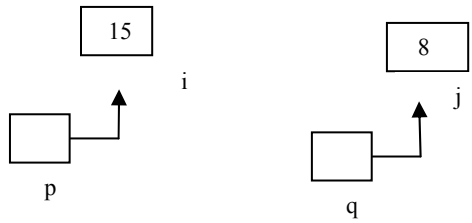
تقويم:

ليكن i وj متحولين صحيحين، و p وq مؤشرين على عددين صحيحين.

ارسم المخططات الموافقة للذواكر بعد تنفيذ كل سطر مما يلي:

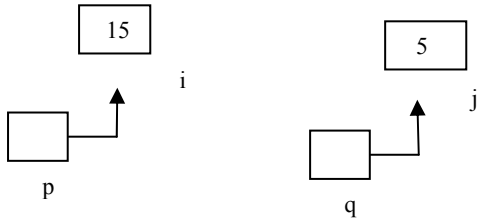
```
int i=15, j=8,*p=&i, *q=&j;
```

الحل:



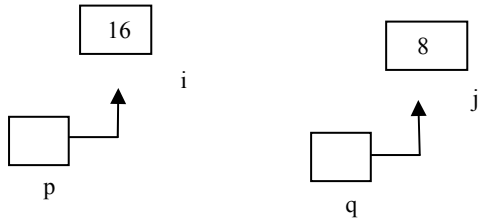

```
*q=5;
```

الحل:



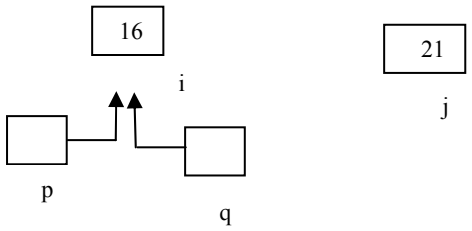

```
(*p)++;
*q+=3;
```

الحل:



```
q=p;
j=*q+5;
```

الحل:



5- المصفوفات والمؤشرات

سبق وتعرضنا إلى تعريف مفهوم المصفوفة وذكرنا أنها تعبر عن مجموعة من المعطيات التي تشغل مواضع متجاورة ومتتالية في الذاكرة. وعند التصريح عن مصفوفة، يكون اسم المصفوفة بمثابة عنوان العنصر الأول فيها. في C++ يعتبر اسم المصفوفة مؤشراً ثابتاً.

مثال

لو صرحنا عن المتحولين التاليين:

```
int arr[10],*p;
```

فإن arr يكون مؤشراً على العدد الصحيح الذي يقع في الموضع الأول في المصفوفة، وp مؤشر على متحول من نمط int. يقوم المترجم عند ترجمة التعليمات بحجز 10 مواضع في الذاكرة لأعداد صحيحة ويسند عنوان أولها إلى arr. يمكننا إذاً استخدام هذا العنوان لإعطاء قيمة ابتدائية لمتحول من نمط مؤشر على عدد صحيح.

يمكننا مثلاً كتابة:

```
p=arr;  //p points to arr[0]
```

5-1- الحسابات على العناوين

رأينا في الفقرات الخاصة بالمصفوفات أنه يمكن النفاذ إلى عنصر في المصفوفة باستخدام الدليل index. يوجد طريقة أخرى تستخدم المؤشرات. لنفرض أن p مؤشر على مصفوفة عناصر من النمط T. إن الصيغة p+1 تمثل عنوان العنصر التالي في المصفوفة أي أن p[1]= *(p+1) . تعني إضافة 1 للعنوان إجراء انزياح بمقدار عنصر واحد إلى يمين p. عموماً يمكـن أن نكتب : p+n للدلالة على عنوان العنصر الذي ترتيبه هو n+1 أي أن p[n]= *(p+n). بمعنى لآخر، يحسب المترجم العنوان الموافق للصيغة p+n بإجراء انزياح معتمد على حجم معطيات النمط T بالبايتات.

```
(p+n) =p+ n*sizeof(T)
```

مثال

ليكن لدينا التصريحات التالية:

```
long arr[5]={200,-60,50,5,90};
long *p=arr;
```

لنفرض أن المصفوفة arr تبدأ عند العنوان 8500. يبين الجدول التالي عناوين العناصر الخمسة في المصفوفة. كما يتضمن الجدول القيم الناتجة عن حساب الصيغة p+n حيث 4≤ n ≤ 0 ، وقيم عناصر المصفوفة الموافقة p[n]=*(p+n)

العنوان		عنصر المصفوفة	
p	8500	*p	200
P+1	8504	*(p+1)	-60
P+2	8508	*(p+2)	50
P+3	8512	*(p+3)	5
P+4	8516	*(p+4)	90

مثال

تستخدم العمليات الحسابية على العنوان لتغيير قيمة المؤشر، فمثلاً يمكننا أن نكتب:

```
p=p+3;
p=p-2;
p++;
p--;
```

مثال

سنستخدم هنا مثالاً عن سلاسل المحارف في C++ (راجع فقرة سلاسل المحارف strings في C++ في فصل " البرامج الجزئية في C++")

```
// Converting lowercase letters to uppercase letters
// using a pointer to data.
#include <iostream>
#include <cctype> // prototypes for islower and toupper
using namespace std;

void convertToUppercase( char *);

int main()
{
   char phrase[] = "characters and $32.98”;

   cout << "The phrase before conversion is: " << phrase;
   convertToUppercase( phrase );
   cout << "\nThe phrase after conversion is:  " << phrase << endl;
   return 0;
}
// convert string to uppercase letters
void convertToUppercase( char *sPtr )
{
   while ( *sPtr != '\0' ) // loop while current character is not
'\0
 {
      if ( islower( *sPtr ) ) // if character is lowercase
      *sPtr = toupper( *sPtr ); // convert to uppercase

      sPtr++; // move sPtr to next character in string
   } // end while
}      //end function convertToUppercase
```

تقويم:

قارن الاستدعاء الوارد في المثال السابق:

```
convertToUppercase( phrase);
```

مع نموذج التابع:

```
void convertToUppercase(* char);
```

ماذا تستنتج؟

مثال

في المثال التالي نكتب خوارزمية فرز (ترتيب) لشعاع من الأعداد الصحيحة وهي selectionSort ثم نستدعيها في برنامج.

تعتمد هذه الخوارزمية على البحث عن أصغر عدد في الشعاع، وتبديل موضعه مع العدد الموجود في أول الشعاع، ثم نكرر العملية على ما تبقى من الشعاع بالبحث عن أصغر عنصر في الشعاع الناتج عن عملية التبديل، ثم وضعه في الدليل التالي وهكذا حتى بلوغ نهاية الشعاع. يبين البرنامج التالي التابع selectionSort واستدعاءه.

```
//This program puts values into an array, sorts the values into
// ascending order and prints the resulting array.
#include <iostream>
#include <iomanip>
using namespace std;

void selectionSort( int a[] ,  int ); // prototype
void swap( int & , int &  ); // prototype
int main()
{
   const int arraySize = 10;
   int a[ arraySize ] = {37, 6, 4, 8, 68, 12, 89, 10, 45, 2};
   cout << "Data items in original order\n";
   for ( int i = 0; i < arraySize; i++)
      cout << setw( 4 ) << a[ i ];
   cout<<endl;
   cout<<"begin sorting"<<endl;
   selectionSort( a, arraySize ); // sort the array
   cout << "\nData items in ascending order\n";
   for ( int j = 0; j < arraySize; j++)
      cout << setw( 4 ) << a[ j ];
   cout << endl;
   return 0;
}   //end main
```

```
//function to sort an array
void selectionSort( int   array[],  int size)
{
   int smallest; // index of smallest element

// loop over size - 1 elements
   for ( int i = 0; i < size - 1; i++)
{
      smallest = i; // first index of remaining array

      // loop to find index of smallest element
      for ( int index = i + 1; index < size; index++)
         if ( array[ index ] < array[ smallest])
            smallest = index;

      swap( array[ i ], array[ smallest]);

      for ( int j = 0; j < size; j++)
          cout << setw( 4 ) << array[ j];
      cout<<endl;

   }// end if
} // end function selectionSort
//swap values in element1 and element2
void swap( int  &element1, int &element2)
{
   int hold = element1;
   element1 = element2;
   element2 = hold;
} // end function swap
```

مثال

نريد الآن تعديل أنماط الموسطات الممررة للتابعين selectionSort و swap بحيث تصبحان كما يلي:

```
void selectionSort( int * ,  int);
void swap( int * , int *);
```

مع المحافظة على التصريح عن الشعاع كما في البرنامج السابق:

```
const int arraySize = 10;
int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37};
```

عدل البرنامج السابق ليتوافق مع المطلوب.

المثال المعدل

```
//This program puts values into an array, sorts the values into
// ascending order and prints the resulting array.
#include <iostream>
#include <iomanip>
using namespace std;

void selectionSort( int * ,  int ); // prototype
void swap( int * , int *  ); // prototype
int main()
{
   const int arraySize = 10;
   int a[ arraySize ] = { 37, 6, 4, 8, 68, 12, 89, 10, 45, 2};

   cout << "Data items in original order\n";

   for ( int i = 0; i < arraySize; i++)
      cout << setw( 4 ) << a[ i ];

   selectionSort( a, arraySize ); // sort the array

   cout << "\nData items in ascending order\n";

   for ( int j = 0; j < arraySize; j++)
      cout << setw( 4 ) << a[ j ];

   cout << endl;
   return 0; // indicates successful termination
}     //end main
// function to sort an array
void selectionSort( int *  array,  int size )
{
   int smallest; // index of smallest element
   // loop over size - 1 elements
   for ( int i = 0; i < size - 1; i++)
{
      smallest = i; // first index of remaining array
      // loop to find index of smallest element
      for ( int index = i + 1; index < size; index++)
         if ( array[ index ] < array[ smallest])
            smallest = index;
      swap( &array[ i ], &array[ smallest]);
      for ( int j = 0; j < size; j++)
          cout << setw( 4 ) << array[ j ];
      cout<<endl;
   } //end if
} //end function selectionSort
// swap values at memory locations to which
// element1Ptr and element2Ptr point
void swap( int *  element1Ptr, int *  element2Ptr )
```

```
{
   int hold = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = hold;
}  //end function swap
```

6- المؤشرات والصفوف

رأيت حتى الآن أنه يمكن التصريح عن متحولات من نمط مؤشر على أنماط بسيطة مثل int, long, char. في الحقيقة، لا يوجد ما يمنع التصريح عن مؤشر على غرض. فمثلاً لنفرض أننا قد عرفنا الصف Time كما في الفصل الرابع. (راجع الفصل الرابع فقرة التصريح الكامل عن الصف Timeوتوابعه) يمكننا أن نعرف مايلي:

```
Time morning(7,0,0);
Time *timePtr;
timePtr=&morning;
```

ولاستدعاء التوابع الأعضاء التابعة للغرض يمكننا أن نكتب:

```
(*timePtr).setTime(6,30,0);
```

وهو يكافئ morning.setTime(6,30,0);.

يجب الانتباه هنا إلى ضرورة استخدام الأقواس لأن المؤثر * له أولوية أضعف من المؤثر . ، وبدون القوسين يصبح لدينا:

```
*timePtr.setTime(6,30,0);
```

الذي يفهمه المترجم كما يلي:

```
*(timePtr.setTime(6,30,0));
```

ولتفادي الوقوع في خطأ من هذا النوع، وتبسيط شكل الاستدعاء، زودت لغة C++ بمؤثر هو -> يسمح للنفاذ إلى أعضاء الصف، وعليه يصبح استدعاء التابع setTime في الغرض المشار إليه بـ timePtr على النحو التالي:

```
timePtr->setTime(6,30,0);
```

7- الذاكرة الديناميكية

كثيراً ما نجد في التطبيقات أن الاحتياجات المتعلقة بحجز الذواكر للبرنامج لا تتحدد إلا عند التنفيذ (run time)، أي أنه يجري حجز الذاكرة عند الطلب بدلاً من حجزها عند الترجمة (compile time).

كمثال على ما سبق، لنفرض أننا بصدد بناء تطبيق لمعالجة علامات الطلاب في جامعة ما . إذا كان عدد الطلاب كبيراً (500 طالب مثلاً)، يجب أن نستخدم مصفوفة كبيرة لخزن علامات الطلاب، في حين أنه في حالة صف فيه 10 طلاب مثلاً لاحاجة لنا بحجز كامل الذاكرة السابقة (500) في حين أننا لن نستخدم منها سوى 10. وهنا يتضح لنا أن تثبيت الحجم المطلوب عند الترجمة قد يتسبب بهدر كبير في موارد الذاكرة، ويجب أن يتوفر إمكان حجز حجم الذاكرة اللازم ديناميكياً اعتماداً على عدد الطلاب في الصف. يسمى هذا المفهوم "الحجز الديناميكي للذاكرة " "dynamic memory allocation" ، وهو مفهوم يقع ضمن مجال "الإدارة الديناميكية للذاكرة " "dynamic memory management".

سنبدأ بعرض مؤثرين جديدين في C++ هما new و delete اللذان يسمحان بحجز الذاكرة وتحريرها ديناميكياً.

7-1- المؤثر new لحجز الذاكرة ديناميكياً

عند تنفيذ أي برنامج، يقوم نظام التشغيل بتهيئة جزء خاص من الذاكرة يستخدمه البرنامج عند حجز الذواكر التي يتعامل معها في البرنامج. يسمى هذا الجزء من الذاكرة " الكومة " "heap". ولكي تستطيع النفاذ إلى هذه الكومة وحجز ذاكرة ما فيها، يجب أن تستخدم المؤثر new مع المؤشرات. يحتاج المؤثر new إلى نمط (يستخدمه كمحدد أي وسيط) وعندها يحجز موضعاً في الكومة يتسع لعنصر من النمط المرفق، ويعيد عنوان الذاكرة المحجوزة الذي نستطيع فيما بعد إسناده إلى مؤشر.

لنأخذ كمثال عملية الحجز الديناميكي لذاكرتين أحدهما من النمط int والأخرى من النمط long. نبدأ بالتصريح عن مؤشر لهذين النمطين:

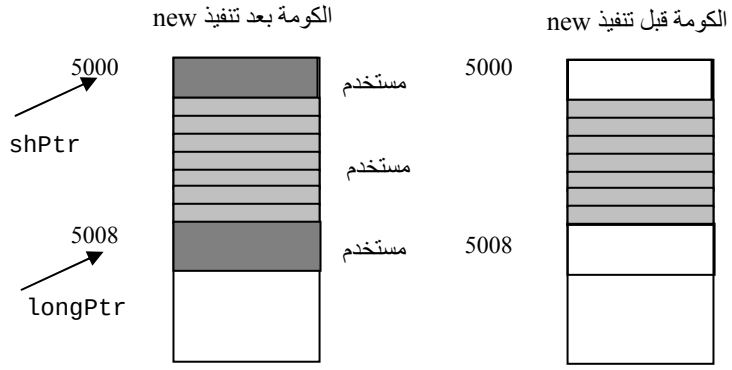
```
int *shPtr;      //assume the size of int is 2 bytes
long *longPtr;   //assume the size of long is 4 bytes
```

يطلب المؤثر new الحصول على ذاكرة من الكومة. إن توفرت الذاكرة، يجري حجز عدد من البايتات يوافق الحجم اللازم للنمط، ويعيد عنوان أول بايت يبدأ عنده الحجز. إن لم تتوفر الذاكرة المطلوبة من الكومة يعيد المؤثر new القيمة 0. يطلق على هذه القيمة في معرض حديثنا عن المؤشرات اسم المؤشر NULL أي المؤشر الصفري ويقصد بذلك أنه لا يشير إلى أي موضع في الذاكرة. إن الثابت NULL الذي يساوي 0 معرف في مكتبة iostream ويمكن أن نضعه قيمة ابتدائية لأي مؤشر من أي نمط. يعتبر NULL العدد الصحيح الوحيد الذي يسند إلى مؤشر، ويستخدم في صيغ المقارنة.

وفيما يلي التعليمتان اللتان تحجزان الذاكرتين المذكورتين:

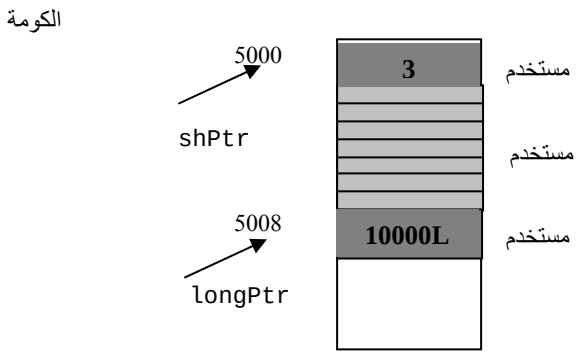
```
shPtr=new int;    //a pointer to an integer
longPtr=new long;    //a pointer to a long
```

نرى في الشكل التالي عملية حجز الذاكرتين shPtr و longPtr. لنفرض أن العناوين في الكومة تبدأ عند العنوان 5000. يطلب المؤثر new حجز ذاكرة من النمط int أي بحجم قدره 2 بايت فتحجز عند الموضع 5000، ويعاد العنوان 5000 ليخزن في المتحول shPtr. عند ورود طلب الحجز التالي لذاكرة من النمط long أي بحجم يعادل 4 بايتات يجري الحجز في موضع آخر يحوي ذواكر لم يجر استخدامها بعد. يمتلك نظام التشغيل لائحة بالذواكر المستخدمة، ومن هنا يمكنه تحديد الذواكر المستخدمة من تلك التي لم تستخدم. بفرض أن الذاكرتين اللتان تبدآن بالعناوين 5002 و 5007 كانتا مستخدمتين، عندها يحجز النظام الذاكرة المطلوبة اعتباراً من العنوان 5008، ويعيد هذا العنوان الذي يخزن في الذاكرة longPtr.



لاتعطى الذواكر في الكومة قيماً افتراضية. وفي حال رغبة المبرمج إعطاء قيمة محددة ابتدائية، عليه أن يمررها بعد اسم النمط بين قوسين صغيرين( ). فمثلاً لإعطاء القيمتين 3 و 10000L على الترتيب للذاكرتين المحجوزتين بالتعليمتين السابقتين يمكننا أن نكتب:

```
shPtr=new int(3);
longPtr=new long(10000L);
```



يمكننا الآن التعامل مع هاتين الذاكرتين عبر المؤشر الذي يشير إلى كل منهما، ونستخدم لهذا الغرض المعامل *.

```
*shPtr=2;      //*shPtr becomes 6
cout<<*longPtr;    //output:10000
```

مثال

لحجز ذاكرتين من نمط string و double نكتب مايلي:

```
double *d=new double;
string *str=new string(”heap”);
//test if memory  is allocated
if (str==NULL)
{cerr<<”memory allocation failure”<<endl;
exit(1);
}
```

7-1-1- الحجز الديناميكي لمصفوفة

تظهر قوة الحجز الديناميكي عندما نرغب بحجز مصفوفات لا تتحدد أطوالها إلا لحظة التنفيذ . نستخدم لهذا الغرض المؤثر new نفسه لكن نستخدم بعده المؤثر [ ] مع تمرير الطول بين هذين القوسين المتوسطين. فمثلاً لحجز مصفوفة طولها 50 نكتب بما يلي:

```
const int ARRSIZE=50;
int *arr;
arr=new int[ARRSIZE];    //dynamically allocate the array
```

تحجز كتلة من الذاكرة بطول ARRSIZE ويوضع عنوان بداية الكتلة في المتحول arr. يمكن الآن التعامل مع المصفوفة باستخدام الدليل كالعادة، حيث arr[0] هو العنصر الأول و arr[ARRSIZE-1] هو العنصر الأخير فيها.

7-1-2- الحجز الديناميكي لأغراض من صفوف

يمكن استخدام المؤثر new لحجز ذواكر لأغراض من صفوف أنشأها المبرمج. يجري استدعاء الباني عندما يحجز ديناميكياً ذاكرة لغرض وحيد من هذا الصف. وإن لم يكن هناك بان افتراضي للصف، يجب عندها تمرير المحددات.

مثال:

```
Time *midNight,*noon;
midNight =new Time;
noon=new Time(12,0,0);
```

لحجز مصفوفة من الأغراض ديناميكياً، يجب أن يكون الصف مزوداً ببان افتراضي، وعندها يحجز المؤثر new مصفوفة جميع عناصرها تأخذ القيمة الابتدائية التي يعطيها الباني الافتراضي.

مثال

```
Time *t;
t=new Time[100];
```

7-2- المؤشر delete لتحرير الذاكرة ديناميكياً

تقع مسؤولية الإدارة الفعّالة للذاكرة في التطبيقات على عاتق المبرمج. يجب أن يدرك المبرمج أن الكومة هي مصدر قابل للنفاد، وعليه إن عملية حجز الذواكر بالمؤشر new يجب أن يعقبها تحرير للذواكر المحجوزة عندما تنتهي الحاجة إليها. تقدم C++ مؤشراً هو delete مهمته تحرير الذاكرة التي سبق حجزها باستخدام new. يعتمد استخدام delete على أن النظام يحتفظ بلائحة تتضمن عناوين الذواكر التي حجزت باستخدام المؤشر new، والحجوم المحجوزة لها. يقوم المؤشر delete بإبطال مفعول new وذلك بتحرير الذاكرة الموافقة.

مثال

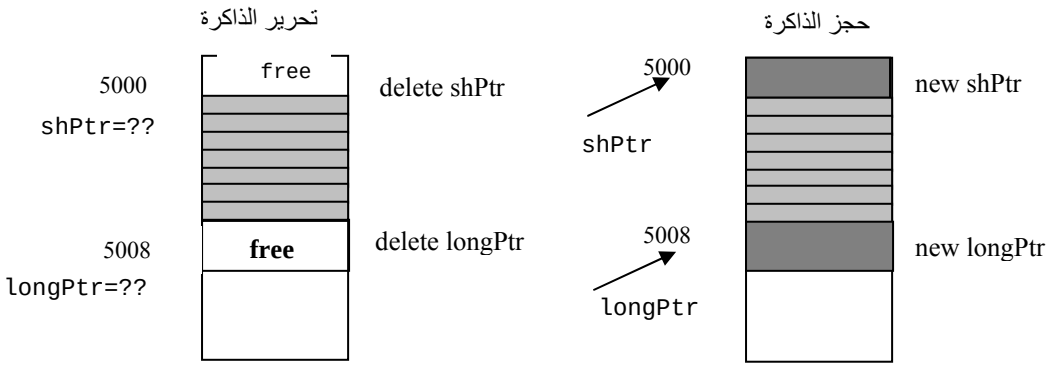
لنعد إلى مثالنا الذي قمنا فيه بحجز ذاكرتين ديناميكياً على النحو التالي:

```
int *shPtr=new int;
long *longPtr=new long;
```

لتحرير الذاكرتين المشار إليهما بـ shPtr و longPtr نستخدم delete يليها اسم المؤشر:

```
delete shPtr;   //deallocates 2 bytes starting at 5000
delete longPtr;    //deallocates 4 bytes starting at 5008
```

يوضح الشكل التالي وضع الكومة بعد عمليات new و delete.



لتحرير ذاكرة حجزت لمصفوفة ديناميكياً، نستعمل المؤثر delete متبوعاً بالقوسين المتوسطتين: [ ] .

مثال

```
int *arr=new int[ARRSIZE];  //allocate the array arr
delete [] arr;              //deallocate the array memory
```

مثال

```
  int *p;
  long *q;

  p=new int(5);
  q=new long[20];

  delete p;
  delete [] q;
```

الفصل السادس

بناء بنى المعطيات باستخدام الصفوف والمؤشرات

هدف الفصل

- دمج مفهومي الصفوف والمؤشرات في صنع مفاهيم متقدمة
- السيطرة على مفهوم المؤشرات والحجز الديناميكي للذاكرة
- إتقان بناء الصفوف المركبة
- تعلم بناء بنى معطيات مفيدة جداً (كاللوائح المترابطة)
- تعلم إدارة بنى المعطيات

1. مقدمة

إذا عدت إلى الأنماط التي تعرفت إليها في اللغة قبل أن تتعرف إلى المؤشرات، لوجدت أنها لا تسمح بالإدارة الديناميكية للذاكرة، لاسيما عندما يتعلق الأمر بحجز كتل في الذاكرة سواء باستخدام المصفوفات أو الصفوف.

وإذا تأملت في المصفوفات التي تعاملت معها سابقاً، لوجدت أنك يجب أن تحدد طول المصفوفة منذ التصريح عنها وذلك إما بوضع الطول بين القوسين [ ] عند التصريح، أو من خلال عدد القيم الابتدائية المستخدمة. وعند الحاجة إلى التعامل مع المصفوفات في البرامج، وإلى كتابة توابع تمرر لها المصفوفات كموسطات يجب تمرير طول المصفوفة للتابع مع تمرير المصفوفة كما في مثال selectionSort الذي مر معك في فصل

" المؤشرات pointers والإدارة الديناميكية للذاكرة dynamic memory management"

سترى كيف ننتقل بالتدريج وبخطوات متتالية من النمط المعرف في أصل C++ للمصفوفات إلى صفوف تعرف أنماطاً تمثل المصفوفات بخصائص معينة ومنها إلى صفوف تمثل بنى معطيات أكثر شمولية من المصفوفات.

2. الصف الأول vect

بما أن طول المصفوفة هو عدد يلازم عناصر المصفوفة عند كل عملية عليها، فلماذا لانعرف صفاً يحتوي في قسمه الخاص على أعضاء المعطيات التالية:

- حقل يمثل عناصر المصفوفة

```
int arr[];
```

- حقل يمثل طول المصفوفة

```
int size;
```

غير أنه لابد في هذه الحال من تحديد الطول الأعظمي للمصفوفة arr. سنختار هنا على سبيل المثال الطول 100.

وفيما يلي تصريحنا عن الصف vect:

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

class vect
{ public:
Vect();
vect(int s);
int getsize();
int &operator[](int);

private:
int arr[100];
int size;
};
vect::vect()
{
 size=10;
  for(int i=0;i<size;i++) arr[i]=0;
}

vect::vect(int s)
{
if( s>0 && s<=100)
     {size=s;
      for(int i=0;i<=size-1;i++) arr[i]=0};
```

```
 else { cerr << "\nError: size " << s
        <<  "out of range" << endl;
        exit(1);}
}
int vect::getsize()
{return size};

int &vect::operator[](int i)
{
     if(i<size && i>=0)
           return arr[i];
     else
     { cerr << "\nError: Subscript " << i
<<"out of range" << endl;
        exit(1);}
}
int main()
{vect ;int I;
v[0]=5;
for( i=0;i<10;i++)
cout<<v[i]<<"  ";
vect v2(4);
v2[0]=1; v2[1]=4;
v2[2]=7; v2[3]=9;

for( i=0;i<v2.getsize();i++)
cout<<v2[i]<<"  ";

for( i=0;i<v2.getsize();i++)
cin>>v2[i];

for( i=0;i<v2.getsize();i++)
cout<<v2[i]<<"  ";
return 0;
}
```

لاحظ أنك لو صرحت عن متحول vect v(200); أو vect v(-3); لظهرت الرسالة:

Error: size 200 out of range

أو Error: size -3 out of range

وكذلك الأمر لو حاولت الوصول إلى الخانة v2[6] أو v2[300]:

```
cout<<v2[6];  //outputs the message:  Error: Subscript 6 out of range
v2[300]=9; //outputs the message:  Error: Subscript 300 out of range
```

ماميزة هذا النمط مقارنة بنمط المصفوفة المعرف سلفاً في C++؟

1- إن عملية التدقيق التي تجري في الباني vect(int s) لعدم تجاوز الطول الأعظمي للمصفوفة تحمي من حصول التجاوز الذي كان يقع في النمط المعرف في أصل C++ للمصفوفات.

2- إن عملية التحقق من وقوع الدليل بين 0 وطول المصفوفة في التابع int &operator[](int) الذي يسمح بزيادة تحميل المؤثر [ ] تحمي من حصول التجاوز الذي كان يقع في النمط المعرف في أصل C++ للمصفوفات.

3- لقد أصبحت المصفوفة الآن ممثلة بغرض واحد، وعند تمرير المصفوفة كموسط نمرر غرضاً واحداً يتضمن في معطياته الأعضاء عناصر المصفوفة وطولها.

وهكذا يمكننا أن نكتب التابع selectionSort كما يلي:

```
void selectionSort( vect &v )
{
   int smallest; // index of smallest element
   for ( int i = 0; i < v.getsize() - 1; i++)
{
      smallest = i; // first index of remaining array
      // loop to find index of smallest element
      for ( int index = i + 1; index < v.getsize(); index++)
         if ( v[ index ] < v[ smallest])
            smallest = index;
      swap( v[ i ], v[ smallest]);
      for ( int j = 0; j < v.getsize(); j++)
          cout << setw( 4 ) << v[ j ];
      cout<<endl;
   }  // end if
}
```

هل حللنا بتعريف نمط المصفوفة بهذا الشكل جميع نقاط الضعف التي تؤخذ على النمط المعرّف في أصل C++ للمصفوفات؟

لا، فإن تحديد الطول الأعظمي للمصفوفة يعني أننا في حال احتجنا إلى مصفوفة أطول فعلينا تعديل الصف، وإعادة ترجمته.

لحل هذه المشكلة سنلجأ إلى تعريف نمط ثان للمصفوفات Array يسمح بإجراء حجز ديناميكي للمصفوفة عند التنفيذ.

3. الصف الثاني Array

لنصرح عن الصف Array بحيث تتألف معطياته الأعضاء من مؤشر على عدد صحيح ومن عدد يمثل طول المصفوفة.

- حقل يمثل مؤشر على عناصر المصفوفة

```
int *ptr;
```

- حقل يمثل طول المصفوفة

```
int size;
```

إن ptr هو مؤشر يشير إلى كتلة في الذاكرة تتضمن عناصر المصفوفة. يجري حجز هذه الكتلة ديناميكياً عند التنفيذ حيث يحجز في الذاكرة كتلة تتناسب مع عدد العناصر الممرر كمحدد للباني الافتراضي. ويجري تحرير هذه الكتلة كذلك دفعة واحدة عندما يستدعى الهادم.

مثال:

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

class Array
{
public:
   Array( int = 10 ); // default constructor
~  Array();
   int getSize();
   int &operator[](int);

private:
   int *ptr;
   int size;
};
Array::Array( int s)
{
   if (s>0)
        size = s ;
   else size= 10 ;
   ptr = new int[ size];

   for ( int i = 0; i < size; i++)
      ptr[ i ] = 0;
}

// destructor for class Array
Array::~Array()
{
 delete [] ptr;
}
// return number of elements of Array
int Array::getSize ()
{
   return size;
}
int &Array::operator[]( int subscript )
{
   // check for subscript out-of-range error
   if ( subscript < 0 || subscript >= size)
  {
      cerr << "\nError: Subscript " << subscript
<<"out of range" << endl;
```

```
      exit( 1 );
 }
   return ptr[ subscript ]; // reference return
 }
int main()
{
   Array integers1( 7 ); // seven-element Array
   Array integers2; // 10-element Array by default
   int I;
   // print integers1 size and contents
   cout << "Size of Array integers1 is "
      <<integers1.getSize()
         <<"\nArray after initialization:\n";
   for (i=0;i< integers1.getSize();i++)
         cout<<integers1[i];

   // print integers2 size and contents
   cout << "\nSize of Array integers2 is"
        <<integers2.getSize()
        <<"\nArray after initialization:\n";
   for (i=0;i< integers2.getSize();i++)
         cout<<integers2[i];
   // input and print integers1 and integers2
   cout << "\nEnter 17 integers:" << endl;
   for (i=0;i< integers1.getSize();i++)
         cin>>integers1[i];
   for (i=0;i< integers2.getSize();i++)
         cin>>integers2[i];

   cout << "\nAfter input, the Arrays contain:\n"
      <<"integers1:\n";
   for (i=0;i< integers1.getSize();i++)
         cout<<integers1[i];
   cout<< "\nintegers2:\n";
   for (i=0;i< integers2.getSize();i++)
         cout<<integers2[i];
  cout << "\nintegers1[5] is " << integers1[ 5 ];

  cout << "\n\nAssigning 1000 to integers1[5]" << endl;
  integers1[ 5 ] = 1000;
  cout << "integers1:\n";
  for (i=0;i< integers1.getSize();i++)
         cout<<integers1[i];
  // attempt to use out-of-range subscript
   cout << "\nAttempt to assign 1000 to integers1[15]"
<< endl;

   integers1[ 15 ] = 1000; // ERROR: out of range
```

```
  return 0;
}
```

ماميزة النمط Array مقارنة بالنمط vect ؟

يجري تحديد طول المصفوفة ديناميكياً، ولايوجد حد أعظمي لطول المصفوفات التي يمكننا التصريح عنها، وأياً كان الطول المرغوب فيكفي تمريره للباني الافتراضي دون إعادة ترجمة الصف.

هل حللنا بتعريف النمط Array بهذا الشكل جميع نقاط الضعف التي تؤخذ على vect؟

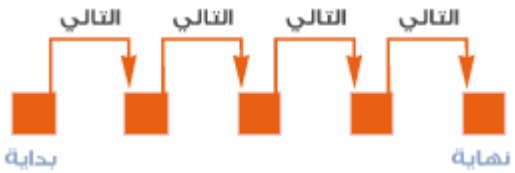
الجواب: لا، فإن تحديد طول المصفوفة ديناميكياً يعني حجز حجم من الذاكرة يوافق الطول المطلوب، ولكن إن احتجنا بعد ذلك إلى إطالة هذه المصفوفة أو تقصيرها حسب عدد العناصر التي تضاف إلى المصفوفة أو تحذف منها أثناء تنفيذ العمليات على المصفوفة، أي لو احتجنا إلى حجز جزء إضافي يلحق بنفس الكتلة أو إعادة جزء منها فإننا لن نتمكن من ذلك لأن عمليتي new[ ] و delete [ ] تحجزان كتلة كاملة أو تعيدانها كاملة.

لحل هذه المشكلة سنلجأ إلى تعريف نمط جديد List يسمح بالحجز الديناميكي عند التنفيذ مع إمكان إضافة أو حذف على مستوى العنصر.

4. الصف الثالث List

<u>مفهوم اللائحة</u>

يعتمد هذا الصف مفهوم اللائحة List. تعرف اللائحة بأنها بنية تحوي عناصر من نفس النمط. وتكون اللائحة متتالية sequential list إذا كان يوجد ترتيب للعناصر فيها، بحيث يوضع العنصر الأول في بداية اللائحة، والعنصر الأخير في نهايتها، ويكون لكل عنصر فيها -عدا العنصر الأخير - عنصر تالٍ له. يمكن الوصول إلى أي عنصر في اللائحة انطلاقاً من بدايتها بالعبور من عنصر إلى العنصر التالي له باتجاه النهاية.



لتمثيل اللائحة المتتالية العناصر سنصرح عن الصف List الذي يتضمن عمليات إقحام عناصر جديدة insert ونزع عناصر من اللائحة remove. لتعريف الصف، نصرح عن الواصفات attributes والتوابع العامة لإدارة اللائحة، أي توابع الإقحام والحذف التي تتعامل مع طرفي اللائحة أي مع بدايتها ونهايته وهما التابعان insertAtFront و insertAtBack للإقحام والتابعان RemoveFromFront و RemoveFromBack للحذف. كما سنعرف التابع isEmpty لفحص إذا كانت اللائحة فارغة، وprint لطباعة اللائحة .

قبل تفصيل كيفية بناء اللائحة، لنتعرف على توابع الصف List لفهم كيفية عملها:

- **الباني:** ينشئ لائحة فارغة

```
List l;
```

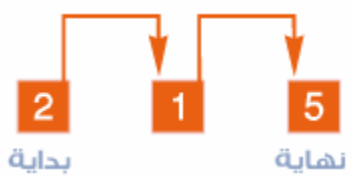
- **الإقحام في البداية:** يقحم عنصراً في بداية اللائحة:

```
l.insertAtFront(1 );
```


```
l.insertAtFront(2 );
```

- **الإقحام في النهاية:** يقحم عنصراً في نهاية اللائحة:

```
l.insertAtBack(5 );
```



- **الحذف من البداية:** يحذف عنصراً من بداية اللائحة:

I. removeFromFront (v);

- **الحذف من النهاية:** يحذف عنصراً من نهاية اللائحة:

I. removeFromBack (v);

سندخل الآن في تفاصيل بناء اللائحة.

الصف Node

يتضح من المثال أن اللائحة تتكون من عناصر منفصلة يتضمن كل منها قيمة ما، غير أن وجود الترتيب الذي ذكرناه سابقاً بين العناصر، وعمليات الإقحام الديناميكية التي تجرى في البداية والنهاية يعني وجود تتالٍ معين للعناصر، وهذا ما أشرنا إليه بقولنا لائحة متتالية. لتحقيق هذا المفهوم، وجب أن ينتهي كل عنصر بمؤشر على العنصر الذي يليه، بالطبع عدا العنصر الأخير الذي لاينتهي بمؤشر للعنصر التالي لعدم وجود عنصر تال.

نستنتج إذاً أن كل عنصر من اللائحة يشبه عقدة node في سلسلة، ويتألف من واصفتين:

- واصفة لقيمة العنصر
- واصفة للمؤشر على العنصر التالي

نحتاج إذاً إلى بناء الصف Node ، وهو يمثل الصف الذي تنتمي إليه كل عقدة في اللائحة. يمكن تمثيل عناصر الصف Node كما في الشكل التالي:

وفيما يلي التصريح عن الصف Node :

```
#ifndef Node_H
#define Node_H

//forward declaration of class List required to
announce that class
// List exists so it can be used in the friend
declaration at line 13
class List;

class Node
{
   friend class List; // make List a friend

public:
   Node( const int & ); // constructor
```

```
   int getData() const; // return data in node
private:
   int data; // data
   Node *nextPtr; // next node in list
};// end class Node
 //constructor
 Node::Node( const int &info)
    : data( info ), nextPtr( 0 )
 {
    // empty body
 }   // end Node constructor
 // return copy of data in node
 int Node::getData() const
 {
    return data;
 } // end function getData

 #endif
```

ملاحظات على الصف Node

1- تأمل الأعضاء الخاصة المصرح عنها في الصف Node:

```
   private:
      int data; // data
      Node *nextPtr; // next node in list
```

ماذا تلاحظ؟

إن العضو الثاني nextPtr هو مؤشر إلى عنصر من النمط Node !!

وهكذا أثناء التصريح عن الصف Node نستخدم عضواً هو مؤشر إلى عنصر من النمط Node نفسه !!

يسمى هذا النوع من الصفوف الصف ذاتي المرجعية Self-Referential class، ويعتبر المؤشر الذي يشير إلى عنصر من نمط الصف (وهو هنا nextPtr) بمثابة رابط link بين عناصر الصف.

يستخدم هذا النوع من الصفوف للتمكن من ربط أغراض هذا الصف بعضها مع بعض للحصول على بنى معطيات متقدمة مثل اللوائح والأرتال والمكدسات والأشجار.

2- يتضمن الصف Node تصريحاً عن الصداقة (راجع الفقرة " التوابع الصديقة friend function والصفوف الصديقة friend class" في الفصل الرابع) بينه وبين الصف List . مافائدة هذه الصداقة؟

بالطبع تستخدم الصداقة لتمكين أغراض الصف List من النفاذ مباشرة إلى الأعضاء الخاصة للصف Node دون استخدام توابع النفاذ، أي لخرق خاصة الكبسلة التي يتمتع بها كل غرض من الصف Node ولكن هذا الخرق مسموح فقط لأغراض الصف الصديق List. سنرى مسببات هذا الخرق عند تعريف الصف List.

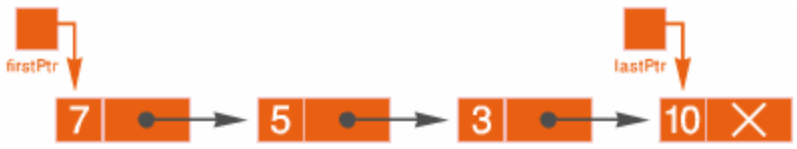
الآن وقد صرحنا عن الصف Node سنستخدمه لنصرح عن الصف List.

الصف List

يمكننا النظر إلى اللائحة على أنها تتألف من سلسلة من العقد، ونظراً لكون عقدتي البداية front والنهاية back تتمتعان بوضع خاص، فإن السلسلة تتضمن مؤشرين إليهما هما firstPtr و lastPtr. تجدر الإشارة إلى أنه لا يوجد عقدة تالية لعقدة النهاية، ومن ثم، فإن قيمة مؤشرها هي NULL. تسمى هذه البنية لائحة مترابطة linked list.

إذاً تتألف اللائحة من واصفتين:

- واصفة للمؤشر على عقدة البداية
- واصفة للمؤشر على عقدة النهاية



لائحة مترابطة

وفيما يلي التصريح عن الصف List :

```
#ifndef LIST_H
#define LIST_H

#include "Listnode.h" // ListNode class definition
#include <iostream>
using namespace std;
class List
{
public:
   List(); // constructor
  ~List(); // destructor
   void insertAtFront( const int  & );
   void insertAtBack( const int & );

   bool removeFromFront( int& );
   bool removeFromBack( int & );
   bool isEmpty() const;
   void print() const;
private:
   Node *firstPtr; // pointer to first node
   Node *lastPtr; // pointer to last node

   //utility function to allocate new node
   Node *getNewNode( const int & );

};   // end class List
//default constructor

List::List ()
   : firstPtr( 0 ), lastPtr( 0 )
{
   // empty body
 }  // end List constructor
// destructor

List::~List()
{
   if ( !isEmpty() ) // List is not empty
     {
      cout << "Destroying nodes ...\n";

      Node *currentPtr = firstPtr;
      Node *tempPtr;
```

```
      while ( currentPtr != 0 ) // delete remaining nodes
      {
         tempPtr = currentPtr;
         cout << tempPtr->data << '\n';
         currentPtr = currentPtr->nextPtr;
         delete tempPtr;
        }//end while
  }// end if

   cout << "All nodes destroyed\n\n";
 }// end List destructor
 // insert node at front of list
 void List::insertAtFront( const int &value)
 {
    Node *newPtr = getNewNode( value ); // new node

    if ( isEmpty() ) // List is empty
       firstPtr = lastPtr = newPtr; // new list has only one node
    else // List is not empty
 {
       newPtr->nextPtr = firstPtr; // point new node to previous
 1st node
       firstPtr = newPtr; // aim firstPtr at new node
    }  //end else
 }// end function insertAtFront
// insert node at back of list
void List::insertAtBack( const int &value )
{
   Node *newPtr = getNewNode( value ); // new node

   if ( isEmpty() ) // List is empty
      firstPtr = lastPtr = newPtr; // new list has only one node
   else // List is not empty
}
      lastPtr->nextPtr = newPtr; // update previous last node
      lastPtr = newPtr; // new last node
   }// end else
 }// end function insertAtBack
 // delete node from front of list
 bool List::removeFromFront( int &value)
 {
    if ( isEmpty() ) // List is empty
       return false; // delete unsuccessful
    else
    {
       Node *tempPtr = firstPtr; // hold tempPtr to delete

       if ( firstPtr == lastPtr )
```

```
         firstPtr = lastPtr = 0; // no nodes remain after removal
      else
         firstPtr = firstPtr->nextPtr; // point to previous 2nd
node

      value = tempPtr->data; // return data being removed
      delete tempPtr; // reclaim previous front node
      return true; // delete successful
}// end else
}// end function removeFromFront
// delete node from back of list
bool List::removeFromBack( int &value )
{
   if ( isEmpty() ) // List is empty
      return false; // delete unsuccessful
   else
{
      Node *tempPtr = lastPtr; // hold tempPtr to delete
      if ( firstPtr == lastPtr ) // List has one element
         firstPtr = lastPtr = 0; // no nodes remain after removal
      else
      {
         Node *currentPtr = firstPtr;
         // locate second-to-last element
         while ( currentPtr->nextPtr != lastPtr )
            currentPtr = currentPtr->nextPtr; // move to next node
         lastPtr = currentPtr; // remove last node
         currentPtr->nextPtr = 0; // this is now the last node
      }// end else
      value = tempPtr->data; // return value from old last node
      delete tempPtr; // reclaim former last node
      return true; // delete successful
   }// end else
}//end function removeFromBack
// is List empty?
bool List::isEmpty() const
{
   return firstPtr == 0;
}//end function isEmpty

//return pointer to newly allocated node
Node *List::getNewNode( const int &value )
{
   return new Node( value);
}//end function getNewNode
// display contents of List
void List::print() const
{
```

```
    if ( isEmpty() ) // List is empty
    {
       cout << "The list is empty\n\n";
       return;
    }//end if
    Node *currentPtr = firstPtr;
    cout << "The list is:";
    while ( currentPtr != 0 ) // get element data
   {
       cout << currentPtr->data <<'  ';
       currentPtr = currentPtr->nextPtr;
   }// end while
    cout << "\n\n";
 }//end function print
#endif
```

شرح التوابع الأعضاء

1- الباني:

يبدأ الباني بإنشاء لائحة فارغة أي ليس فيها أية عقدة، ومعنى هذا أن كلا المؤشرين firstPtr و lastPtr فيهما 0 (NULL)

2- الهادم:

إن المهمة الأساسية للهادم هي إعادة الذاكرة الديناميكية التي جرى حجزها لعقد اللائحة.

تبدأ الإجرائية بالتحقق من وجود عناصر في اللائحة أي من أنها ليست فارغة (!isEmpty()، وفي حال كانت غير فارغة، نستخدم مؤشر يسمى currentPtr يشير إلى عقد اللائحة الواحدة تلو الأخرى اعتباراً من عقدة البداية، وكلما نقلناه إلى العقدة التالية حذفنا العقدة السابقة باستخدام delete.

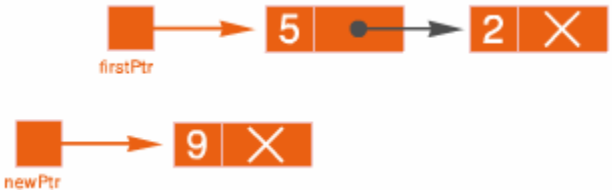
3- الإقحام في بداية اللائحة insertAtFront:

إن المهمة الأساسية لهذا الإجراء هي إنشاء عقدة جديدة ووضعها في بداية اللائحة. ينفذ الإجراء مهمته على مراحل هي كالتالي:

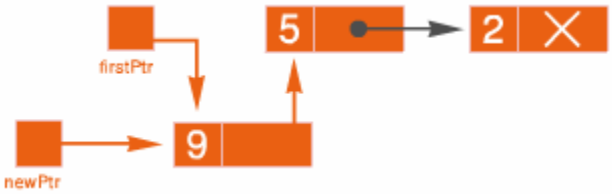
- استدعاء التابع getNewNode مع تمرير القيمة value له وهي القيمة التي نريد خزنها في العقدة الجديدة.

- يستخدم التابع getNewNode المؤثر new لإنشاء عقدة جديدة ويعيد المؤشر الذي يشير إلى هذه العقدة، فيسند هذا المؤشر إلى المتحول newPtr.
- إذا كانت اللائحة فارغة سابقاً فهذا يعني أن العقدة الجديدة هي الوحيدة الموجودة فيها الآن، وعليه يشير كلا المؤشرين firstPtr و lastPtr إلى هذه العقدة.
- إذا لم تكن اللائحة فارغة سابقاً، فهذا يعني أن العقدة الجديدة يجب أن توضع في البداية، وعليه نضع قيمة المؤشر firstPtr في مؤشر العقدة الجديدة، ثم نجعل firstPtr يشير إليها.

(A)



(B)



4- الإقحام في نهاية اللائحة insertAtBack:

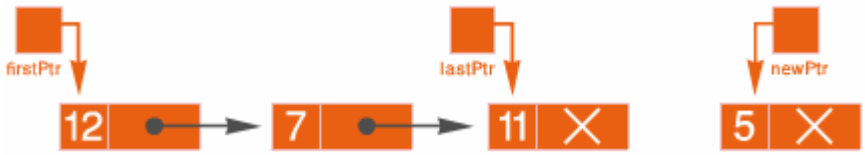
إن المهمة الأساسية لهذا الإجراء هي إنشاء عقدة جديدة ووضعها في نهاية اللائحة.

ينفذ الإجراء مهمته على مراحل هي كالتالي:

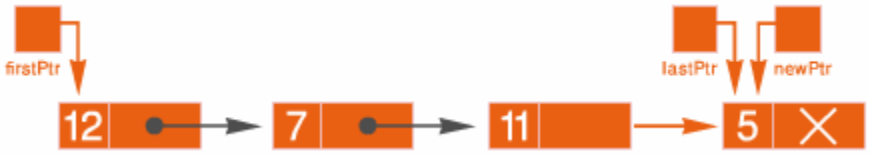
- استدعاء التابع getNewNode مع تمرير القيمة value له وهي القيمة التي نريد خزنها في العقدة الجديدة.

- يستخدم التابع getNewNode المؤثر new لإنشاء عقدة جديدة ويعيد المؤشر الذي يشير إلى هذه العقدة، فيسند هذا المؤشر إلى المتحول newPtr.
- إذا كانت اللائحة فارغة سابقاً فهذا يعني أن العقدة الجديدة هي الوحيدة الموجودة فيها الآن، وعليه يشير كلا المؤشرين firstPtr و lastPtr إلى هذه العقدة.
- إذا لم تكن اللائحة فارغة سابقاً، فهذا يعني أن العقدة الجديدة يجب أن توضع في النهاية، وعليه نضع قيمة المؤشر newPtr في مؤشر عقدة النهاية، ثم نجعل lastPtr يشير إلى هذه العقدة الجديدة التي أصبحت عقدة النهاية..

(A)



(B)



5- الحذف من بداية اللائحة removeFromFront:

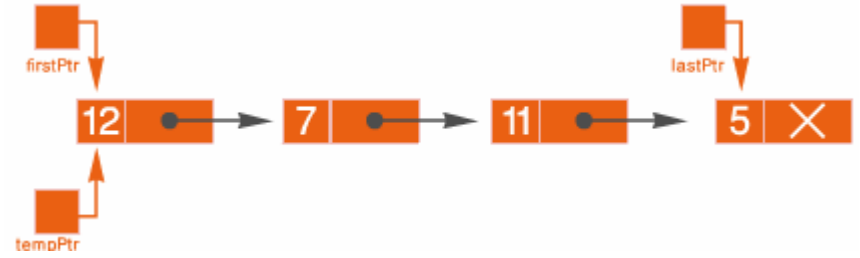
إن المهمة الأساسية لهذا الإجراء هي حذف العقدة الموجودة في البداية، وإسناد قيمة العنصر المحذوف إلى الموسط الممرر بالمرجع لهذا الإجراء. يعيد الإجراء القيمة false في حال جرت محاولة لحذف عنصر البداية من لائحة فارغة، في حين يعيد true عندما تنجح عملية الحذف.

ينفذ الإجراء مهمته على مراحل هي كالتالي:

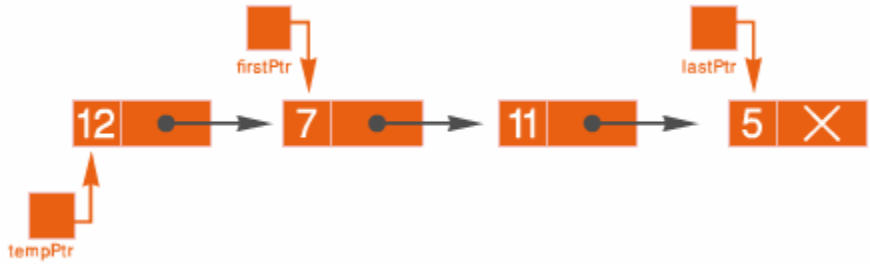
- يضع في المتحول tempPtr عنوان الذاكرة التي يشير إليها المؤشر firstPtr. إن الذاكرة التي يشير إليها tempPtr هي التي سيجري حذفها.

- في حال كان المؤشرين firstPtr و lastPtr متساويين أي أن اللائحة تتألف من عنصر وحيد، يجب وضع 0 في كل من firstPtr و lastPtr لإزالة العقدة من اللائحة.
- في حال كانت اللائحة تحتوي على عدد من العقد يزيد على 1 فإن الإجراء لا يقوم بأي تعديل على lastPtr في حين يجعل firstPtr يشير إلى العقدة التالية وذلك بوضع قيمة firstPtr->nextPtr في firstPtr وهكذا تصبح العقدة الثانية في اللائحة هي العقدة الأولى.
- يجري نسخ قيمة العقدة التي نعمل على حذفها في المتحول value.
- ننفذ تعليمة delete على المؤشر tempPtr الذي يشير إلى العقدة التي نعمل على حذفها، وذلك لإعادة الذاكرة المحجوزة.
- يعيد الإجراء القيمة true للدلالة على نجاح عملية الحذف.

(A)



(B)



6- الحذف من نهاية اللائحة removeFromBack:

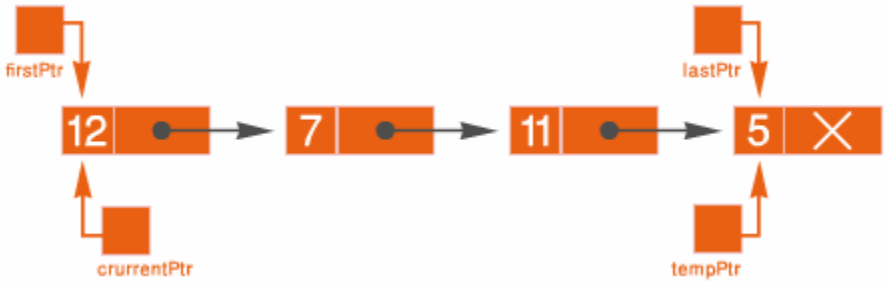
إن المهمة الأساسية لهذا الإجراء هي حذف العقدة الموجودة في النهاية، وإسناد قيمة العنصر المحذوف إلى الموسط الممرر بالمرجع لهذا الإجراء. يعيد الإجراء القيمة false في حال جرت محاولة لحذف عنصر النهاية من لائحة فارغة، في حين يعيد true عندما تنجح عملية الحذف.

ينفذ الإجراء مهمته على مراحل هي كالتالي:

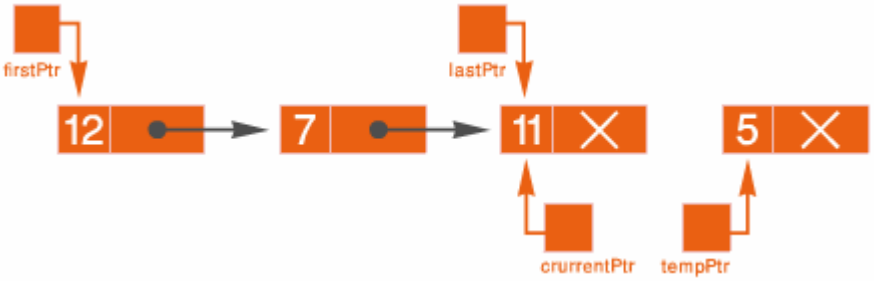
- يضع في المتحول tempPtr عنوان الذاكرة التي يشير إليها المؤشر lastPtr. إن الذاكرة التي يشير إليها tempPtr هي التي سيجري حذفها.

- في حال كان المؤشرين firstPtr و lastPtr متساويين أي أن اللائحة تتألف من عنصر وحيد، يجب وضع 0 في كل من firstPtr و lastPtr لإزالة العقدة من اللائحة.
- في حال كانت اللائحة تحتوي على عدد من العقد يزيد على 1 فإن الإجراء لا يقوم بأي تعديل على firstPtr في حين يجب أن يجعل lastPtr يشير إلى العقدة السابقة. وللوصول إلى العقدة السابقة عليه أن يستخدم مؤشراً (currentPtr) يبدأ بالتأشير على أول عقدة أي فيه قيمة firstPtr.
- يمر currentPtr على جميع عقد اللائحة، حتى يصل إلى العقدة قبل الأخيرة، وذلك بتنفيذ حلقة تضع currentPtr في currentPtr->nextPtr مادام currentPtr->nextPtr لايساوي lastPtr.
- نضع قيمة currentPtr في lastPtr لحذف العقدة الأخيرة من اللائحة.
- نضع القيمة 0 في currentPtr->nextPtr في العقدة الأخيرة الجديدة.
- يجري نسخ قيمة العقدة التي نعمل على حذفها في المتحول value.
- ننفذ تعليمة delete على المؤشر tempPtr الذي يشير إلى العقدة التي نعمل على حذفها.
- يعيد الإجراء القيمة true للدلالة على نجاح عملية الحذف.

(A)



(B)



7- طباعة اللائحة

يفحص هذا الإجراء اللائحة لمعرفة إذا كانت فارغة أم لا. فإن كانت فارغة يظهر على الشاشة جملة موافقة ويعود، وإلا يمر على جميع العقد ويطبع حقل القيمة في كل عقدة. للمرور على كافة العقد يستخدم مؤشر مساعد هو currentPtr الذي يشير بدايةً إلى العقدة الأولى (قيمته الابتدائية = firstPtr)، ومادام currentPtr لايساوي 0 يطبع currentPtr->data ثم ينتقل للعقدة التالية بوضع currentPtr->nextPtr في currentPtr.

البرنامج الرئيسي

يقوم البرنامج التالي باستدعاء التابع testList الذي يأخذ كوسيط لائحة ويظهر على الشاشة مجموعة خيارات (الإقحام والحذف في بداية اللائحة ونهايتها، والخروج)، وبعد كل خيار يطبع قيم عقد اللائحة:

```
//List class test program.
#include <iostream>
#include <string>
#include "List.H" // List class definition
using namespace std;
void instructions();
// function to test a List
void testList( List &listObject)
{
   cout << "Testing a List  \n";
   instructions(); // display instructions
   int choice; // store user choice
   int value; // store input value
   do // perform user-selected actions
{
      Cout<<"? ";
      cin >> choice;
      switch ( choice )
     {
         case 1: // insert at beginning
            cout << "Enter an integer: ";
            cin >> value;
            listObject.insertAtFront( value );
            listObject.print();
```

```
            break;
         case 2: // insert at end
            cout << "Enter an integer :";
            cin >> value;
            listObject.insertAtBack( value );
            listObject.print();
            break;
         case 3: // remove from beginning
            if ( listObject.removeFromFront( value ))
               cout << value << " removed from list\n";
            listObject.print();
            break;
         case 4: // remove from end
            if ( listObject.removeFromBack( value ))
               cout << value << " removed from list\n";
            listObject.print();
            break;
      } //end switch
   } while ( choice != 5 ); // end do...while
   cout << "End list test\n\n";
}  // end function testList
// display program instructions to user
void instructions()
{
   cout << "Enter one of the following:\n"
<<"  1 to insert at beginning of list\n "
<<"  2  to insert at end of list\n "
<<"  3  to delete from beginning of list\n "
<<"  4 to delete from end of list\n "
<<"  5 to end list processing\n”;
}  // end function instructions
int main()
{
   // test List of int values
   List integerList;
   testList( integerList );

   return 0;
}// end main
```

التقويم:

لابد أنك لاحظت في تعريف النمط List أننا استخدمنا مؤشرين firstPtr و lastPtr. وإذا أمعنت النظر يمكن أن يتبين لك أننا نستطيع تنفيذ جميع الوظائف السابقة دون وجود المؤشر lastPtr . والمطلوب:

- إعادة تعريف الصف List الذي يتألف من عضو معطيات واحد هو front الذي يشير إلى البداية.
- تزويد الصف بتابع بحث search عن وجود قيمة يعطيها المستخدم ضمن عقد اللائحة، ويعيد true إذا كانت موجودة و false إن لم تكن موجودة.
- تزويد الصف بتابع حذف remove عقدة في اللائحة إذا كانت تحتوي على قيمة يعطيها المستخدم، أياً كان موقعها (بداية اللائحة أو نهايتها أو داخل اللائحة)

الإجابة:

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

class Node{
  friend class List;
public:
     Node(int, Node* n = 0);
 private ;
  int data;
  Node* nextPtr;
};

class List{
public :
  List();
~ List();
  void insertAtFront(const int &);
```

```
  bool remove(int );
  bool removeFromFront(int &);
  void print();
  bool isEmpty() const;
  bool search(int);
private:
  Node* front;
};
Node::Node(int x ,Node* n){
  data = x;
  nextPtr = n;
}

 List::List(){
  front = 0;
};

bool List::isEmpty() const {
  return front == 0 ? true : false;
}
List::~List()
{   Node* tmp = front;
  while (tmp)
     {      front = tmp->nextPtr;
           delete tmp;
              tmp = front;
     }
 }

void List::insertAtFront(const int &x)
{   Node* p = new Node(x,front);
    front = p};
bool List::search(int x)
{
  if (front == 0) return false;
  Node* p = front;
  bool found = false;
  while ((p) && (!found))
    if (p->data == x)
        found = true;
    else
        p = p->nextPtr;
  return found;
}

void List::print()
 {
  cout <<"( ";
```

```
  for (Node* p = front; p; p = p->nextPtr) cout << p->data<<" ";
  cout << ")" << endl;
}
bool List::removeFromFront(int &x)
{
  if (front == 0)
     return false;
  else {
        x=front->data;
           Node* p = front;
        front = front->nextPtr;
        delete p;
           return true;
          }
}
  bool List::remove(int x)
 {
  if (front == 0){
    cout << "empty List !" << endl;
    return false;
 }
 // research of the element to remove
 Node* p = front;
 Node* pred = 0;
 bool found = false;
 while ((p) && (!found))
   if (p->data == x)
       found = true;
   else {
       pred = p;
       p = p->nextPtr;
  }
 if (!found){
   cout <<  x << " does not exist in the list !" << endl;
   return false;
}
 else { // remove the found element, pointed by p
   if (pred) { // the element to remove has a predecessor
(pred->nextPtr) = (p->nextPtr);
     delete p;
       return true;
  }
   else {      // the element to remove is the first in the list
     front = (p->nextPtr);
     delete p;
       return true;
  }
 }
```

```
}
void main()
{
   List l;
   int i;

   cout << "Enter integers, 0 to finish : " << flush;
   cin >> i;
   while (i)
{
     l.insertAtFront(i);
     cin >> i;
   }
   if (!(l.isEmpty()))
      l.print();
   else
     cout << "empty list !" << endl;
   cout << "Research in the list, 0 to finish : " << flush;
   cin >> i;
   while (i){
     if (l.search(i))
     cout << i << " exist in the list" << endl;
     else
     cout << i << " does not exist in the list" << endl;
     cout << "Research in the list, 0 to finish : " << flush;
     cin >> i;
   }

   cout << "Elimination from the list, 0 to finish : " << flush;
   cin >> i;
   while (i){
     l.remove(i);
     l.print();
     cout << "Elimination from the list, 0 to finish : " << flush;
     cin >> i;
   }
}
```

ملاحظة:

لابد أنك لاحظت أن باني العقدة في هذا البرنامج يضع قيمة في المؤشر nextPtr هي عنوان لذاكرة تمرر أثناء البناء في حين كان الباني في البرنامج السابق يضع 0 في المؤشر.

تقويم:

لتكن لدينا قائمة بأسماء الطلاب في الصف ومعدلاتهم مرتبة أبجدياً. نريد إدخال هذه القائمة إلى الحاسوب في بنية ديناميكية هي اللائحة المترابطة. تقرأ المعلومات بالترتيب الأبجدي لأسماء الطلاب ولكن إدخال المعلومات إلى اللائحة يتم بحيث يراعى الترتيب التنازلي للمعدلات.

والمطلوب: التصريح عن الصف StudentList الذي يتألف من عقد تحتوي كل منها على اسم الطالب، ومعدله ومؤشر على العقدة التالية، وكتابة التوابع الأعضاء التالية:

- إقحام عقدة الطالب في الموضع المناسب لترتيب المعدل
- حذف عقدة طالب يعطى اسمه
- البحث عن معدل طالب يعطى اسمه
- طباعة اللائحة

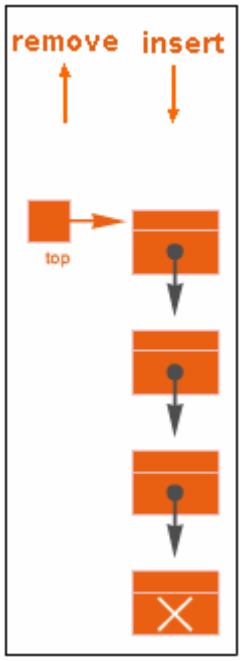
كما يطلب كتابة برنامج رئيسي يبين استدعاء هذه التوابع.

5. بنى المعطيات المتقدمة

بعد أن تعرفت إلى مفهوم اللائحة، فإنه يمكن صنع بنى معطيات عديدة تستند في بنيتها التخزينية على اللائحة، ويختلف بعضها عن بعض بطريقة إدارتها، ومنها المكدسات والأرتال والأشجار.

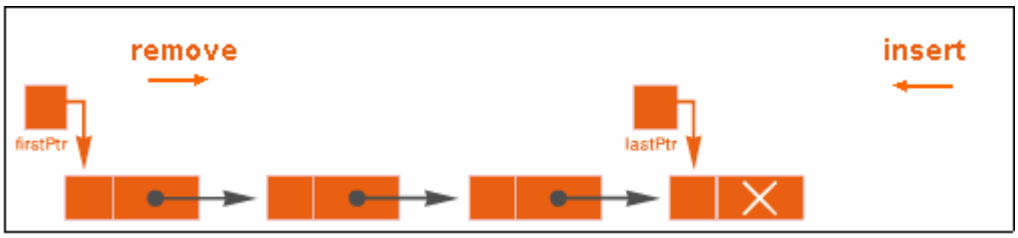
المكدس

بنية معطيات نقحم العناصر فيها في بدايتها وننزعها من بدايتها فقط. لفهم هذه البنية تذكر كيف نكدس الأطباق بعضها فوق بعض، وعندما نضع طبقاً نضعه في أعلى المكدس وعندما نحتاج إلى سحب طبق نسحبه من الأعلى كذلك. نستخدم بنية معطيات هي لائحة لكن التوابع الأعضاء لاتتضمن سوى توابع تتعامل مع الأعلى (البداية) فقط (لا يوجد مؤشر نهاية، ولا توابع إقحام أو حذف في أي مكان سوى أعلى المكدس). يسمى أسلوب الإدارة هذا (الداخل آخراً يخرج أولاً (LIFO) last in first out)



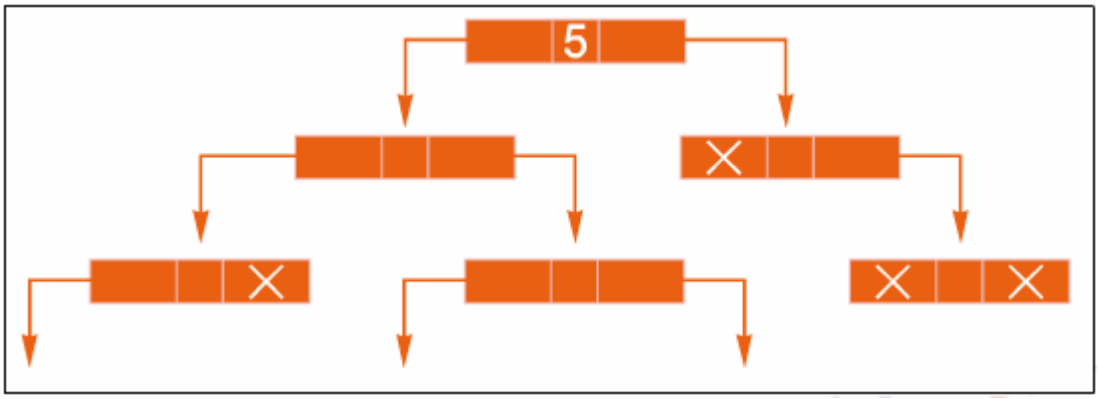
الرتل

بنية معطيات نقحم العناصر فيها في نهايتها وننزعها من بدايتها. لفهم هذه البنية تذكر كيف يقف الزبائن أمام صندوق الدفع في محل تجاري، وكلما أتى زبون يقف في آخر الرتل، في حين يخدم الزبون الذي يقف في أول الرتل. نستخدم بنية معطيات هي لائحة لكن التوابع الأعضاء لاتتضمن سوى توابع الإقحام في النهاية والحذف من البداية (يوجد مؤشران للبداية والنهاية. يسمى أسلوب الإدارة هذا (الداخل أولاً يخرج أولاً (FIFO) first in first out)



الشجرة

بنية معطيات غير خطية، تتألف من عقد، في كل عقدة توجد مؤشرات للعقد الفروع. للتبسيط، ولفهم المبدأ العام، لنأخذ بنية ثنائية البعد. تمثل هذه البنية شجرة ثنائية تتألف من عقد، تحتوي كل عقدة على مؤشرين أحدهما يشير إلى عقدة الفرع الأيمن والثاني يشير إلى عقدة الفرع الأيسر. لهذه البنية استخدامات عديدة ستراها لاحقاً في مقرر الخوارزميات وبنى المعطيات المتقدمة.



إن الدخول في تفاصيل بناء هذه البنى (المكدس، الرتل، الشجرة) واستخداماتها يحتاج إلى فصول. وسيخصص لهذا الغرض مقررات لاحقة تعنى ببنى المعطيات المتقدمة والخوارزميات المبنية عليها، كما يحتاج الدخول في بعض الحالات إلى مفاهيم برمجية غرضية التوجه لم تسنح الفرصة لعرضها هنا كالوراثة والربط الديناميكي. وسيفرد مقرر كامل لهذه المفاهيم البرمجية الهامة في البرمجة الغرضية التوجه.

Type	Description	Size	Domain
char	Signed character/byte. Characters are enclosed in single quotes.	1	-128..127
double	Double precision number	8	ca.10^{-308}..10^{308}
int	Signed integer	4	$-2^{31}..2^{31} - 1$
float	Floating point number	4	Ca. $10^{-38}..10^{38}$
long (int)	Signed long integer	4	$-2^{31}..2^{31} - 1$
long long (int)	Signed very long integer	8	$-2^{63}..2^{63} - 1$
short (int)	Short integer	2	$-2^{15}..2^{15} - 1$
unsigned char	Unsigned character/byte	1	0..255
unsigned (int)	Unsigned integer	4	$0..2^{32} - 1$
unsigned long (int)	Unsigned long integer	4	$0..2^{32} - 1$
unsigned long long (int)	Unsigned very long integer	8	$0..2^{64} - 1$
unsigned short (int)	Unsigned short integer	2	$0..2^{16} - 1$

تحتوي هذه المكتبة تعريفات لمجموعة من المؤثرات نذكر منها على سبيل المثال:

setfill(char) , setprecision(int), setw(int).

- setw : يسمح setw(int) بتحديد حجم الحقل المراد إدراج القيمة ضمنه. وفي هذه الحالة يكون التنسيق إلى اليمين وهو أمر عملي في حالة تمثيل الأعداد.

```
#include <iostream.h>
#include <iomanip.h>
int main() {
  int n = 64;
  cout << "In hexadecimal : " << hex << n << endl;
  cout << " In octal : " << oct << n << endl;
  cout << " In decimal : " << dec << n << endl;
  //  The same display right justified
  cout << setw(20) << "Hexadecimal : " << hex << setw(6) << n << endl;
  cout << setw(20) << "Octal : " << oct << setw(6) << n << endl;
  cout << setw(20) << "Decimal : " << dec << setw(6) << n << endl;

  return 0;
}
```

يعطي التنفيذ النتيجة التالية:

```
In hexadecimal :  40
 In octal : 100
In decimal : 64
          Hexadecimal :    40
                Octal :   100
              Decimal :    64
```

يبدو هذا المؤثر مفيداً جداً عند تحصيل القيم. يقوم البرنامج التالي بإظهار النص المدخل على شكل كتل طولها LL-1 ينتهي كل منها بالمحرف "\0".

```
#include <iostream.h>
#include <iomanip.h>
const int LL = 10;  // maximum size of a line
int main() {
  char line[LL];
  while (cin >> setw(LL) >> line)
      cout << line << endl;
  return 0;
}
```

يعطي التنفيذ ما يلي:

```
abcdefghijklmnopqrstuvwxyz
abcdefghi
jklmnopqr
stuvwxyz
^Z
```

ملاحظة : ^Z هو المحرف الذي ينهي ملف الدخل في الأجهزة التي تعمل بـ Ms_Dos.

- setprecision : في الحالة الافتراضية، يجري إظهار أعداد الفاصلة العائمة على 6 أرقام معنوية.

يمكن تعديل هذه الدقة باستعمال المعالج setprecision (int).

```
#include <iostream.>
#include <iomanip.h>

const double pi = 3.141592654;

int main() {
  cout << pi << endl;                  // 6 digit by default
  cout << setprecision(9) << pi << endl; // 9 digit
  cout << pi/2.0 << endl;              // we are still on  9 digit
  cout << setprecision(2) << pi << endl; // 2 digit
  return 0;
}
```

وعند التنفيذ نجد:

```
3.14159
3.14159265
1.57079633
3.1
```

setfill: يستعمل المؤثر setfill(char) عادةً مرتبطاً بـ setw(int)، ويملأ المساحة غير المستعملة بالمحرف المحدد.

```
#include <iostream.h>
#include <iomanip.h>
int main() {
  int n = 64;
  cout << "In hexadecimal : " << hex << n << endl;
  cout << "In octal : " << oct << n << endl;
  cout << "In decimal : " << dec << n << endl;
  // The same display right justified
  cout << setw(20) << "Octal : " << oct << setw(6) << n << endl;
  cout << setw(20) << "Decimal : " << dec << setw(6) << n << endl;

  cout << setw(20) << "In hexadecimal : ";
  cout << hex << setfill('.') << setw(6) << n << endl;
  cout << setfill(' ') << setw(20) << "In octal : ";
  cout << oct << setfill('.') << setw(6) << n << endl;
  cout << setfill(' ') << setw(20) << "In decimal : ";
  cout << dec << setfill('.') << setw(6) << n << endl;

  return 0;
}
```

يعطي التنفيذ النتيجة التالية:

```
In hexadecimal :  40
 In octal : 100
In decimal : 64
          In hexadecimal : ….40
                In  Octal : ...100
    In Decimal  : ….64
```

يمكن للقارئ المهتم أن يعود لدليل المستخدم للمترجم C++ الذي يعمل عليه، للاطلاع على القائمة الشاملة للمؤثرات الموجودة.

تعرف مكتبة iostream.h التي توجد مع كل نظام C++ نمطين من الدخل والخرج يسمى الأول istream (input stream) (أي قناة الدخل) والثاني ostream (output stream), (أي قناة الخرج).

تنفّذ عملية الإخراج من قبل المبرمج باستعمال مؤثر الإدراج "<<" إذ يدرج قيمة في قناة الخرج.

أما عملية الإدخال فهي تنفّذ باستعمال مؤثر الاستخراج ">>". أي أننا نستخرج قيمة من قناة الدخل.

وهنالك ثلاثة متحولات قَبلِيةُ التعريف من النمط "قناة" وهي:

- cin، متحول من النمط istream مرتبط بملف الدخل المِقيَس.
- cout، متحول من النمط ostream مرتبط بملف الخرج المِقيَس.
- Cerr متحول من النمط ostream مرتبط بملف الأخطاء المِقيَس ولا ترتبط به ذاكرة مؤقتة.

كما يوجد في هذه المكتبة تعريفات للمؤثرات التالية:

- المؤثر *endl* الذي يتسبب في إدراج المحرف "\n"
- المؤثرات dec, flush, hex, oct :

1. *flush* :يتسبب في كتابة الذاكرة المؤقتة المرتبطة بالقناة، ومن ثَم بمسحها وإفراغها. قد يكون هذا الأمر ضرورياً في عمليات التحصيل، إذ يجب أن يظهر النص الذي يوضح المطلوب قبل أن يقوم المؤثر بإدخال القيمة المطلوبة. فإن لم يستعمل flush، فلا نضمن هذا الأمر.

```
#include <iostream.h>
int main() {
  int numbe;
  cout << "Enter a number between 0 and 12 " << flush;

  cin >> nombre;
}
```

2. *Hex* يجعل النظام ينتقل للنمط الست عشري أثناء استخراج وإدراج الأعداد الصحيحة. يبقى هذا النظام فعالاً إلى أن يطلب المبرمج تغييره. وهناك مؤثرات أخرى مشابهة هي *oct, dec*، غير أن الأول يعتمد الأساس الثماني، والثاني يعتمد الأساس العشري. في الحالة العامة الافتراضية يعمل النظام في النمط العشري.

```
#include <iostream.h>

int main() {
  int n;
  cout << "Enter an integer number " << flush;
  cin >> n;  // equivalent to cin >> dec >> n;
  cout << "This is the  number in hexadecimal : " << hex << n << endl;
  cout << " This is the  number in octal : " << oct << n << endl;

  cout << (++n) << endl; // we are still in octal mode!

  cout << " This is the  number in hexadecimal : " << hex << n << endl;
  // now we move to decimal mode
  cout << " This is the  number in decimal : " << dec << n << endl;
```

```
 cout << " Enter an integer number " << flush;
 cin >> hex >> n;  // input in en hexadecimal mode
 cout << " This is the  number in hexadecimal: " << hex << n << endl;
 cout << " This is the  number in decimal: " << dec << n << endl;

 cout << " Enter an integer number " << flush;
 cin >> n;  // still in hexadecimal mode
cout << "" This is the  number in hexadecimal: " << hex << n << endl;
 cout << "" This is the  number in decimal: " << dec << n << endl;

 return 0;
}
```

يعطي تنفيذ البرنامج ما يلي:

```
Enter an integer number 45
This is the  number in hexadecimal : 2d
This is the  number in octal : 55
56
This is the  number in hexadecimal : 2
This is the  number in decimal : 46
Enter an integer number AB
This is the  number in hexadecimal: ab
This is the  number in decimal: 171
Enter an integer number D2
This is the  number in hexadecimal: d2
This is the  number in decimal: 210
```

- يسمح المؤثر ws (white space) أثناء القراءة من القناة باستخراج واستبعاد جميع المحارف "الفارغة" أي محارف الفراغ والجدولة والقفز إلى سطر جديد والقفز إلى صفحة جديدة، وذلك اعتباراً من الموقع الحالي وحتى أول محرف غير فارغ.