# المحتويات

## الفصل الثاني

## أوامر الأدخال والأخراج



## الفصل الثالث

## ايعازات القرار والتكرار

الفصل الرابع

الدوال



الفصل الخامس

المصفوفات

## الفصل السادس

## المؤشرات



## الفصل السابع

## متواليات الرموز- السلاسل الرمزية

الفصل الثامن

التراكيب، الأتحاد، وحقول البتات

الفصل التاسع

الصنوف



الفصل الثاني عشر

عمليات الملف

# المقدمة

**أَمّا بَعْدَ حَمْدِ اللهِ الَّذِي جَعَلَ الحَمْدَ ثَمَناً لِنَعْمَائِهِ، وَ مَعاذاً مِنْ بَلائِهِ، وَوَسِيلاً إلى جِنانِهِ، وَسَبَباً لِزِيادَةِ إحْسانِهِ، وَالصَّلاةِ عَلى رَسُولِهِ نَبِيِّ الرَّحْمَةِ، وإمامِ الأَئِمَةِ، وَسِراجِ الأُمَةِ، المُنْتَخَبِ مِنْ طِينَةِ الكَرَمِ، وَسُلالَةِ المَجْدِ الأَقْدَمِ، وَمَغْرَسِ الفِخارِ المُعْرِقِ، وَفَرْعِ العَلاءِ المُثْمِرِ المُورِقِ، وَعَلى أَهْلِ بَيْتِهِ مَصابِيحِ الظُّلَمِ، وَعِصَمِ الأُمَمِ، وَمَنارِ الدِّينِ الواضِحَةِ، وَمَثاقِيلِ الفَضْلِ الرَّاجِحَةِ، صَلَّى اللهُ عَلَيْهِمْ أَجْمَعِينَ، صَلاةً تَكُونُ إزاءً لِفَضْلِهِمْ، وَمُكافأةً لِعَمَلِهِمْ، وكِفاءً لِطِيبِ فَرْعِهِمْ وَأَصْلِهِمْ، ما أَنارَ فَجْرٌ ساطِعٌ، وَخَوى نَجْمٌ طالِعٌ.**

لغات البرمجة تسمح للمبرمج باستخدام اللغة بشكل مشابهة لتلك التي تكتب بشكل طبيعي وهي تستند على توليد ايعازات تعتمد على الحاسوب لتنفيذ البرنامج. هناك العديد من لغات البرمجة مثل C، Pascal، Fortran، Cobol، Basic وغيرها الكثير وجميع هذه اللغات تهدف الى انجاز مهمة خاصة، تسهيل التعامل مع الحاسوب لحل المشكلات، وتنفيذ العديد من التطبيقات التي نحتاج اليها بشكل يومي ودوري.

لغة البرمجة C++ هي اضافة جديدة لقائمة كبيرة من لغات البرمجة المتوفرة حاليا. فهي لغة قوية ومرنة لها مالانهاية من التطبيقات.

تدعى لغة C++ لغة مترجمة، حيث ليس بمقدورك كتابة برنامج C++ وتنفيذة على حاسبتك مالم يكن لديك مترجم C++، هذا المترجم يستلم ايعازات لغة C++ الخاصة بك ويحولها الى شكل يمكن لحاسبتك قرائتها، مترجم C++ هو الاداة التي يستخدمها حاسوبك لفهم ايعازات لغة C++ في برنامجك.

أمكانية تنظيم ومعالجة البيانات هو مفتاح النجاح في الحياة الحديثة. صمم الحاسوب لحمل ومعالجة كميات كبيرة من المعلومات بسرعة وكفاءه. بشكل عام فان الحاسوب لايمكنة عمل أي شيء مالم يتم أخبارة مايجب أن يقوم به. لذلك وجدت C++. C++ هي لغة برمجة عليا (أي قريبة من لغة الأنسان وفهمة) والتي تسمح لمهندس البرامجيات بالتواصل بكفاءة مع الحاسوب. وتعد لغة C++ من اللغات ذات المرونة العالية والقابلة للتكيف. ومنذ أختراعها في عام 1980 فقد تم استخدامها لبرامج واسعة ومختلفة تضمنت تعليمات مخزنة على الحاسوب للمسيطرات الدقيقة ( micro controller)، أنظمة التشغيل (operating systems)، التطبيقات، (applications) وبرامج الرسوم (graphics programs). وأصبحت C++ بسرعة لغة البرمجة التي يتم أختيارها.

ومن خلال تدريسي لمادة البرمجة والبرمجة الكيانية باستخدام لغة البرمجة C++ شعرت بوجود الحاجة الملحة لكتاب يبسط المفاهيم والافكار التي تساعد الطالب والقاريء على تعلم البرمجة وتطوير مهاراتة وامكانياتة في مجال البرمجة الكيانية، ومن الملاحظ افتقار المكتبة العربية الى مصادر علمية متخصصة مكتوبة باللغة العربية مما يظطر القاريء الى الاستعانة بالمصادر الاجنبية والتى تفقدة الكثير من المهارات والمعارف نظرا للنقص الكبير باللغة الاجنبية التي كتب بها الكتاب.

من هذا شرعت بكتابة هذا الكتاب الذي يركز على لغة البرمجة C++ ومايتعلق بها فضلا عن البرمجة الكيانية، وحاولت جاهدا ان يكون هذا الكتاب بسيط يسهب بشرح المفاهيم وقواعد اللغة فضلا عن احتوائة الى اكثر من 230 مثالا محلولا، وهو يفيد الاشخاص الذين ليس لديهم فكرة عن البرمجة او

هؤلاء الراغبين بتطوير امكانياتهم البرمجية وحتى المختصين ومحترفي البرمجة.

ولابد من الاشارة هنا الى ان غالبية حلول البرامج التي وضعت في هذا الكتاب لم تراعي ان يكون البرنامج برنامج احتراف ومثالي وذلك لان الهدف من الامثلة المحلولة هو توضيح افكار ومفاهيم معينة لذلك تم التركيز على هذا المبدأ مبتعدين بعض الشيء عن المثالية وعن اختصار بعض الشفرات في كتابة البرامج او ان يكون البرنامج ذات وقت اقصر بالتنفيذ.

الكتاب نظم وفقا لفصول عددها اثنا عشرة فصلا وكل فصل ركز على موضوع او مواضيع معينة وكما يأتي:

**الفصل الاول** ركز على اعطاء القاريء فكرة عامة عن البرمجة وبعض المصطلحات كثيرة الاستخدام وهو يعتبر مدخلا للبرمجة ولذلك فلابد لمن يرغب الولوج الى عالم البرمجة ان يفهم ماورد بهذا الفصل قبل ان ينتقل الى الفصول الاخرى.

**الفصل الثاني** يبدأ باولى خطوات البرمجة والتي تعتمد على اوامر الادخال والاخراج ويوضح هذا الفصل كيفية التعامل مع اوامر الادخال والاخراج وتم ايراد عدد من الامثلة التي توضح ذلك.

في **الفصل الثالث** تم الانتقال الى شرح الايعازات التي تتعامل مع القرارات في البرمجة وهي حجر الزاوية في الكثير من البرامج.

اما الاساس الذي تبنى عليه لغة البرمجة C++ الا وهي الدوال فقد تم تخصيص **الفصل الرابع** لها ليتم التعامل معها بشكل موسع، هذا الفصل توسع بشرح كل ماله علاقة بالدوال وكيفية استخدامها والضوابط التي تحكمها وميزات استخدام الدوال.

المصفوفات خصص لها **الفصل الخامس**، والمصفوفات لها الكثير من التطبيقات وهي تساعد بشكل او اخر على تسهيل حل المشكلات. وقد تم خلال هذا الفصل التعامل مع المصفوفات الاحادية والثنائية ويحتوي الفصل على الكثير من الامثلة المحلولة.

بعد هذه الفصول التي تعد اساسية للراغبين بتطوير امكانياتهم في البرمجة يتم التقدم باتجاة المؤشرات التي خصص لها **الفصل السادس** وتم خلال هذا الفصل التركيز على المؤشرات والمرجعية وينتقل الفصل من بيان اهمية المؤشرات، واستخدام المؤشرات مع المصفوفات الى التخصيص الالي للذاكرة وتوضيح الكثير من خواص المؤشرات باستخدام امثلة مختلفة.

ونظرا لاهمية الرموز والتعامل معها فقد افرز لها **الفصل السابع** ولم توضع مع المصفوفات كما هو معتاد وذلك لاهميتها، ولذلك فقد تم التركيز على كيفية التعامل مع الرموز وتوضيح الدوال التي تتعامل مع الرموز وعلاقة الرموز بالمصفوفات وماهية الاعمال التي يمكن ان تطبق على الرموز بشكل عام.

**الفصل الثامن** هو مرحلة انتقالية من البرمجة المهيكلة الى البرمجة الكيانية وقد توسع هذا الفصل بتوضيح التراكيب والاتحادات وكيفية التعامل مع البتات، وكيفية تعامل التراكيب مع المؤشرات.

الفصل الاول في البرمجة الكيانية هو شرح الصنوف والذي كان **الفصل التاسع** مخصص له حيث تم الشرح باسهاب عن مفاهيم الصنوف وماهية الكيانات والبرمجة الكيانية، وفي هذا الفصل تم شرح الكثير من الدوال التي لها اهمية في البرمجة الكيانية. لابد ان اشير الى ان هذا الفصل تم التوسع به بشكل كبير لتوضيح الكثير من مفاهيم البرمجة الكيانية وبما يتناسب مع اهمية هذا الموضوع.

**الفصل العاشر** تطرقنا به الى مفهوم اخر مهم من مفاهيم البرمجة الكيانية وهو الوراثة وحاولنا شرحها بشكل مبسط وكيفية الاستفادة من فكرة الوراثة، وكيفية تاثيرها على البرمجة الكيانية.

ومن صفات البرمجة الكيانية موضوع القوالب والذي افرد له **الفصل الحادي عشر** وتم التطرق للقوالب بشكل عام وقوالب الصنف وكذلك تم التطرق الى الاستثناءات لما لها اهمية كبيرة في البرمجة بشكل عام.

اخيرا كان **الفصل الثاني عشر** الذي ركزنا فيه على التعامل مع الملفات بكل انواعها والتركيز على كيفية استخدام العديد من الدوال الخاصة التي تتعامل مع الملفات.

واذا كان لابد من كلمة اخيرة فاني اقول اني بذلت جهدا كبيرا لاخراج هذا الكتاب بشكل يساعد جميع المهتمين بالبرمجة على الاستفادة منه واذا كان هناك نقص او ملاحظة فانا على استعداد لسماعها عسى ان تنفعنا في وقت لاحق لتنقيح الكتاب وساكون سعيد بكل مايردني من ملاحظات.. فقد اردت من هذا الكتاب مرضاة الله، واسال الله عز وجل ان يحسبه في ميزان حسناتي.

**نضال العبادي**

**النجف الأشرف/ العراق 2011**

**comp_dep_educ@yahoo.com**

# الفصل الأول
# تمهيد للغة
# C++

## 1.1 المقدمة

البرنامج هو سلسلة متتالية من الايعازات، يمكننا تشبيهها بوصفة أعداد وجبة غذائية، النوتة الموسيقية، أو نموذج حياكة. وتتميز عنها برامج الحاسوب بشكل عام بأنها أطول أمتدادا وكتابتها تستدعي دقة وعناية فائقتين. وقبل الشروع والخوض في موضوع البرمجة لابد من تعريف بعض المصطلحات التي تحتاجها لاحقا.

## 1.2 بعض الصفات العامة للبرنامج

- يحتاج البرنامج بصورة عامة الى من يكتبة وهو المبرمج (Programmer)، والى المعالج (Processor) لتفسير وتنفيذ (Execution OR Running) الايعازات أو الأوامر (Instructions OR Commands)، وتسمى عملية تنفيذ كامل البرنامج (المعالجة) (Process).
- أن تنفيذ البرنامج يتم بصورة متتالية (أي أيعاز (instruction) بعد الأخر حسب تسلسلها)، مالم يتم الأخبار خارجيا عن غير ذلك. هذا يعني أن نبدأ بأول أيعاز وينفذ ثم الثاني والثالث وهكذا لحين الوصول الى الأيعاز الأخير. هذا النموذج ممكن أن يتغير بطريقة محددة مسبقا بشكل جيد من قبل المبرمج, كما يمكن أن يتم تكرار جزء من البرنامج وحسب تحديدات المبرمج (مثل عملية تكرار مقطع من نوتة موسيقية).
- كل برنامج يجب أن يكون له تأثير.. مثلا في القطعة الموسيقية يكون هذا التأثير عبارة عن صوت، أما في برامج الحاسوب هذا التأثير يكون على شكل مخرجات، أما مطبوعة أو معروضة على الشاشة.

- كل برنامج يعمل على أشياء محددة (تدعى كيانات) للوصول الى التأثير المطلوب (مثلا في وصفة أعداد الطعام فان هذه الاشياء ممكن أن تكون اللحوم، الخضار، وغيرها)، أما في البرامج فأن هذه الاشياء تكون متغيرات.
- في العديد من البرامج يجب أن يتم الأعلان المسبق عن الكيانات (المتغيرات) التي سيتم استخدامها، وماهية أنواعها (هذا مشابهة لعملية اعداد وجبة طعام حيث يجب أن تحتوي الوصفة ابتداءا تحديد المواد التي ستستخدم وكمياتها).
- في بعض الايعازات ربما تكون هناك حاجة أن يترك أتخاذ قرار تنفيذ الأيعاز الى المعالج وفقا لشرط أو شروط معينة تحدد مسبقا.. فمثلا (عندالقيام بالحياكة يكتب في الوصفة مثلا ما يلي "عند توفر خيوط حياكة بيضاء تستخدم في خلاف ذلك استخدم الخيوط الصفراء").
- ربما تكون هناك حاجة لتنفيذ أيعاز أو مجموعة من الايعازات لأكثر من مرة. عليه طالما هناك أيعاز يراد تكراره فأن عدد مرات التكرار يجب ان تحدد. من ممكن أنجاز ذلك أما بتحديد عدد مرات التكرار بشكل دقيق أو تحديد عدد مرات التكرار اعتمادا على شرط محدد مسبقا (مثلا في الحياكه نقول نستخدم الخيط ذو اللون الأبيض بقدر ثلاثين نفذة) أو بفحص حالة تكون من ضمن العملية (مثلا يستخدم الخيط الأبيض لحين أن تنتهي من رسم دائرة أو شكل معين).

## 1.3 مدخل للبرمجة

الحاسوب هو أداة أو ماكنة لحل المشاكل، حيث يستلم البيانات المدخلة، ويجري عليها عمليات حساب بسرعة كبيرة ليوفر مخرجات كنتائج لعملية الحساب. تتم السيطرة على عمل الحاسوب بواسطة سلسلة من الايعازات أو الأوامر (Instructions OR Commands) تسمى بمجموعها برنامج (Program).

يتعامل الناس مع مهام مختلفة لغرض أنجازها، مثل ضبط الوقت في الساعة أو تشغيل جهاز التلفزيون وهناك أمور أكثر تعقيدا مثل عمل قالب من الكيك، ابدال حنفية ماء، بناء فناء في الدار وهذه الأمور الأكثر تعقيدا تحتاج الى مهارات أكثر

لحل المشاكل. فمثلا أن المشاكل الواجب عليك حلها عند أعداد قالب من الكيك تبدأ من اعداد الوصفة التي تتضمن ماهية المواد التي تدخل في صناعتة وكمياتها، نوع القالب الذي يجب أن يستخدم وكذلك الخطوات الواجب اتباعها لاعداد هذا القالب والتي تتضمن أسبقية المواد التي تضاف وكيفية خلطها ودرجة الحرارة... الخ، اذاً عليك أن تحلل المشكلة وتجد الحلول. لنبدل المطبخ بعمل أكثر تعقيدا وهو معالجة مشكلة في حنفية ماء مثلا، هنا لا توجد وصفة تتبع لأنجاز هذا العمل، حيث لا توجد وصفة تتبع لتحديد الأجزاء الواجب ابدالها والأدوات الواجب استخدامها، ولا يوجد دليل عمل يمثل الخطوات الواجب أتباعها لانجاز مثل هكذا عمل، مثل هذا العمل يحتاج من الشخص الذي يقوم بالعمل (السباك) ببعض التحضيرات المهمة المسبقة وبعدها يقرر ما هي المواد المطلوبة وما هو العمل المطلوب قبل الشروع بالعمل فمثلا هل المطلوب ربط الماء الحار مع البارد او يكونان منفصلين وكيفية السيطرة على درجة حرارة الماء وكيفية الربط بمصادر المياه وغيرها من التفاصيل الواجب معرفتها مسبقا وجميع ذلك يعتبر جزء من **تحليل المشكلة الابتدائي**، بعدها يجب أن يقرر ما هي الأدوات الواجب استخدامها مثل قاطع الأنابيب، مفاتيح الربط والفتح وهل تكون مسننة أم ملساء وهكذا. أما الخطوات الواجب أتباعها فهي تمثل الخطوات اللازمة لفتح الحنفية القديمة وإبدالها بالجديدة.

أن المكونات (components) التي تستخدم في حل المشاكل تسمى (objects) (أشياء أو **كيانات**). وهي تمثل كتل البناء والادوات التي تتفاعل لانتاج المنتج النهائي. نحن نرى الأشياء أو الكيانات بدلالة مواصفاتها التي تبين ماهيتها، وكذلك الأفعال التي تصف ما يمكن أن تقوم به هذه الكيانات. فمثلا لو عدنا الى أمثلتنا السابقة.. أولا أعداد قالب الكيك.. فأن قالب الكيك الذي يستخدم للشواء هو كيان له مواصفات مثل الشكل (دائري، مستطيل...الخ)، عمق القالب ("2, "3، "6)، المادة المصنوع منها القالب (المنيوم، تفلون، زجاج). كذلك الفرن هو كيان مع أفعال للسيطرة على الحرارة ومصدر الحرارة، هذا الكيان له مواصفات مثلا الحجم، مستوى الحرارة، مصدر الحرارة (الأعلى للشوي والأسفل للتسخين)، أما الأفعال

فهي مثلا تشغيل وأطفاء الفرن، أختيار مصدر الحرارة، ضبط درجة الحرارة...الخ.

أما المثال الخاص بمعالجة مشكلة حنفية الماء فهناك كيانات مثل روابط الأنابيب، المفك، الحنفية... وكل منها له خواص وصفات خاصة وكذلك أفعال فمثلا المفك له قياس، مثل طول القبضة، حجم الفكوك وهكذا، أما أفعالها فأن فتحة فكوكها ممكن أن تنظم لتلائم حجوم مختلف الأنابيب.

عندما نحدد الكيانات فأن حل المشكلة يجب أن يعرف الوسيط (agent) الذي ينظم عملية التفاعل بين الكيانات لأنجاز المهمة. فمثلا الطباخ الذي يقوم بأعداد قالب الكيك هو الوسيط فهو يقوم بمزج المواد، دهن القالب، تسخين الفرن، ويحدد الوقت اللازم لبقاء قالب الكيك في الفرن.

كذلك فأن السباك هو الوسيط الذي يزيل الحنفية القديمة، يقطع ويصل الأنابيب، ويركب الجزء الجديد مع الواشرات أو اللحيم أو أي وسيلة أخرى.

أن تكنولوجيا الكيانات تنظر الى حل المشكلة من منظار الكيانات. التحليل الأولي يعرف الكيانات كعناصر لعملية حل المشكلة، أما التحليل النهائي فأنه يخلق خطة رئيسية أو وصفة تسمح للوسيط بترتيب أفعال الكيانات.

دعنا ننظر الى حالات حقيقية تتضمن كيانات وحل لمشكلة:

- نفرض أنك في غرفتك في وقت متأخر من الليل وقررت أن تقرأ كتابا، تتطلب المشكلة مجموعة من الكيانات.. فيجب أن يكون لديك كتاب، وسيلة أنارة، وربما تحتاج الى أوراق وقلم. انت الوسيط الذي ينير ويطفيء الضوء، يفتح الكتاب وينظم كتابة الملاحظات.
- جهاز التحكم عن بعد يحل الكثير من مشاكل مشاهدة برامج التلفزيون. هذا الجهاز يحتوي على لوحة مفاتيح وهو كيان بينما مشاهد التلفزيون هو الوسيط المسؤول عن تشغيل المنظم، أختيار القناة، وينظم الصوت.

## 1.4 الحاسوب وحل المشاكل

الوسيط في عالم حل المشاكل الحقيقي يتعامل ماديا مع الكيانات. ولكن عندما يتدخل الحاسوب فأن العملية تتغير لتلائم طبيعة الماكنة. الحاسوب هو أداة حساب تعمل مع بيانات الأرقام والأحرف، فهو يتصف بوجود ذاكرة لخزن البيانات ونتائج الحسابات، لوحة المفاتيح لأدخال البيانات، أزرار للتعامل مع العمليات، وشاشة لعرض النتائج. الحاسوب لا يشبة الحاسبة الجيبية البسيطة فهو جهاز من الممكن أن ينظم باستخدام الايعازات المصممة للتعامل مع حالات مختلفة. أن عمليات الحاسوب مصممة للتعامل مع سيل من المعلومات حيث أن البيانات تدخل الى الذاكرة، اجراء عمليات الحساب، تجهيز النتائج كمخرجات.

عندما تستخدم الحاسوب لحل المشكلة فأنك تحتاج الى أن تركز أنتباهك على الكيانات (وهي بيانات) والتي لها خواص ولها أفعال تتمثل بعمليات الوصول ومعالجة البيانات.

الشفرة الحقيقية لبرنامجك تتكون من جزئين: المتغيرات (الكيانات) وايعازات التنفيذ، المتغيرات تستخدم للتعامل مع البيانات المستخدمة بواسطة برنامجك. ايعازات التنفيذ تخبر الحاسوب ماذا يعمل بالبيانات.. توضع المتغيرات (الكيانات) في ذاكرة الحاسوب المخصصة للقيم، C++ يحدد هذه المواقع من خلال أسم المتغير ويفضل استخدام الاحرف الصغيرة للمتغيرات بينما الأحرف الكبيرة للثوابت.

## 1.5 نمذجة كيانات العالم الحقيقي

كيانات الحاسوب تمثل ملخص لنماذج العالم الحقيقي. في العالم الحقيقي فأن الطالب يعتبر كيان معقد مع خواص مادية مختلفة مثل (الجنس، لون البشرة، لون العين، لون الشعر..الخ) ومعلومات عن السكن (العنوان الحالي، مسقط الرأس..الخ). وعندما يقبل الطالب في الجامعة فأنه يراجع دائرة التسجيل، الحسابات، القسم المقبول فيه وربما الرابطة الطلابية، وكل اتصال مع الدوائر أعلاه يتضمن التعامل مع بيانات مختلفة ومشاكل مختلفة للحاسوب. كل أتصال يتضمن

بيانات خاصة ويحتاج منا الى خلق نماذج مختلفة للطالب داخل الحاسوب. فمثلا دائرة الحسابات لا تهتم بعمر الطالب، عنوان السكن، الجنس.. لكن هذه المعلومات مهمة مثلا لدائرة الأقسام الداخلية بينما دائرة الحسابات تهتم بالرقم التعريفي للطالب، طريقة دفع الأقساط أن كانت هناك أقساط...الخ.

الكيانات هي قوالب تتضمن الصفات والعمليات المتوفرة لذلك الكيان. برامج الحاسوب هي أدوات قوية لحل المشكلة. تبدأ بتحليل المشكلة، ثم خلق سلسلة من الخطوات التي تقود الى الحل، هذه السلسلة من الخطوات تدعى خوارزمية (Algorithm)، والخوارزمية هي سلسلة من الأفعال والخطوات تقود الى حل للمشكلة في وقت محدد. حل المشكلة بالحاسوب يتم بواسطة الخوارزميات التي تنفذ بواسطة البرامج، ولتصميم برنامج يجب أولا أن تعرف الكيانات التي تخزن وتتعامل مع البيانات، فعندما يتم اختيار الكيان فأنك تحتاج الى تطوير برنامج رئيس، له خوارزميات توفر المدخلات الضرورية، وكذلك ترتب أو تنظم عملية التفاعل بين الكيانات وتكتب المخرجات على الشاشة. هذا البرنامج الرئيس هو الوسيط لانجاز عمليات الحساب للمهام.

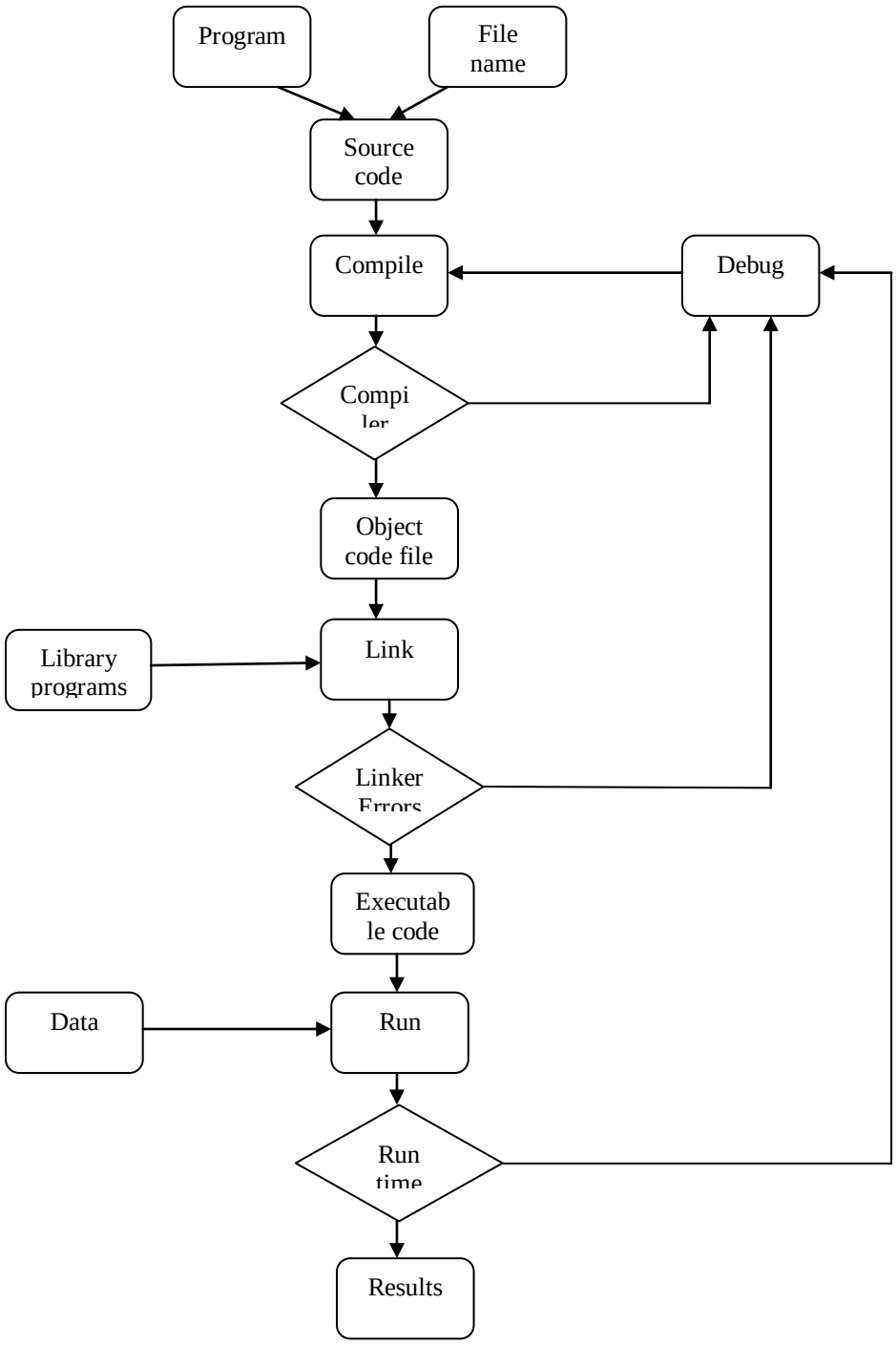
### C++ **1.6**

أمكانية تنظيم ومعالجة البيانات هو مفتاح النجاح في الحياة الحديثة. صمم الحاسوب لحمل ومعالجة كميات كبيرة من المعلومات بسرعة وكفاءة. بشكل عام فان الحاسوب لايمكنة عمل أي شيء مالم يتم أخبارة مايجب أن يقوم به. لذلك وجدت C++. C++ هي لغة برمجة عليا (أي قريبة من لغة الأنسان وفهمة) والتي تسمح لمهندس البرامجيات بالتواصل بكفاءة مع الحاسوب. وتعد لغة C++ من اللغات ذات المرونة العالية والقابلة للتكيف. ومنذ اختراعها في عام 1980 فقد تم استخدامها لبرامج واسعة ومختلفة تضمنت تعليمات مخزنة على الحاسوب للمسيطرات الدقيقة (micro controller)، أنظمة التشغيل (operating systems)، التطبيقات (applications)، وبرامج الرسوم (graphics programs). وأصبحت C++ بسرعة لغة البرمجة التي يتم أختيارها.

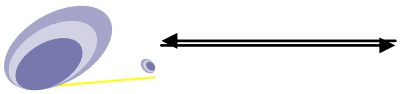
صممت C++ كجسر بين المبرمج والحاسوب. الفكرة بجعل المبرمج ينظم البرنامج بطريقة (هو/ هي) يفهمة بسهولة. بعدها يقوم المترجم (هو برنامج واجبه تحويل اللغة العليا الى اللغة التي يتعامل بها الحاسوب) بنقل اللغة (البرنامج) الى صيغة تستطيع الماكنة استخدامها (التعامل معها). برنامج الحاسوب يتكون من جزئين: هيكل البيانات والايعازات. يفرض الحاسوب او لايفرض القليل من التنظيم على هذين الجزئين. بعد هذا كله فان الحواسيب مصممة لان تكون عامة قدر الأمكان. البيانات في الحاسوب تخزن كسلسلة من البايتات و C++ تنظم هذه البايتات ببيانات مفيدة. الأعلان عن البيانات تستخدم من قبل المبرمج لوصف المعلومات التي (هو/هي) يتعامل معها.

برامج C++ تكتب بلغة عليا باستخدام الأحرف، الأرقام، والرموز الأخرى التي نجدها على لوحة المفاتيح. واقعا فان الحواسيب تنفذ البرامج المكتوبة بلغة دنيا تدعى لغة الماكنة (machine code) (والتي هي سلسلة من الأرقام ممثلة بطريقة الصفر، واحد). لذلك، وقبل ان يتم استخدام البرنامج يجب أن يكون هناك عدد من التحويلات. البرامج تبدأ كفكرة في رأس المبرمج. يقوم المبرمج بكتابة افكارة في ملف، يدعى ملف المصدر (source file or source code) مستخدما محرر اللغة. هذا الملف يحول بواسطة المترجم الى (الملف الهدف) (object file). بعدها يستدعي البرنامج الرابط (linker) حيث ياخذ الملف الهدف ليربطة أو يشركة مع روتينات معرفة مسبقا من المكتبة القياسية (standard library) لينتج برنامج قابل للتنفيذ (والذي هو عبارة عن مجموعة من ايعازات لغة الماكنة). الشكل (1.1) يبين خطوات تحويل البرنامج المكتوب بلغة عليا إلى برنامج قابل للتنفيذ.

في لغة البرمجة C++ فأن البرنامج هو **تجميع للدوال**. والبرامج البسيطة تحتوي على دالة واحدة فقط هي ((main)) وعادة فأن التنفيذ يبدأ عند (main) حيث أن جميع البرامج بلغة C++ يجب أن تحتوي على الدالة ((main)).




**شكل ( 1.1 ): خطوات تنفيذ البرنامج**



***ملاحظة://***

كل عبارة في لغة C++ يجب أن تنتهي بفارزة منقوطة عدا بعض الحالات الأستثنائية التي سيشار اليها في حينها.

***ملاحظة://***

- الايعازات (الأوامر أو العبارات statements): تبدو مختلفة في لغات البرمجة المختلفة، ولكن هناك وظائف أو دوال اساسية قليلة تظهر في كل البرامج تقريبا منها:

الأدخال input وهي عملية الحصول على البيانات من لوحة المفاتيح او الملفات او الأجهزه الاخرى.

الأخراج output عرض البيانات على الشاشة او ارسالها الى ملف او الأجهزة الأخرى.

الرياضيات math أنجاز العمليات الرياضية الاساسية مثل الجمع والضرب.

الاختبار testing أختبار بعض الشروط وتنفيذ بعض العبارات وفقا لذلك.

التكرار repetition أنجاز بعض الاعمال بشكل متكرر، عادة مع بعض التغيرات.

### 1.6.1 لماذا لغة C++

C++ هي اللغة الاكثر استخداما في العالم. هذه اللغة لها صفات وخصائص تميزها عن لغات البرمجة الاخرى، واكثر هذه الصفات هي:

- البرمجة الكيانية

امكانية تنظيم البرنامج على شكل كيانات تسمح للمبرمج تصميم تطبيقاته، لتكون اكثر اتصال بين الكيانات بدلا من هيكل الشفرة المتتالية. فضلا عن انها تسمح بامكانيه كبيرة الى اعادة استخدام الشفرة بطرق اكثر منطقية وانتاجيه.

- النقل

بامكانك عمليا ان تترجم نفس شفرة C++ على الاغلب في اي نوع من الحواسيب وانظمة التشغيل دون اجراء تغييرات صعبة.

- الأيجاز

الشفرة التي تكتب بلغة C++ هي قصيرة جدا بالمقارنة مع اللغات الاخرى، حيث يفضل استخدام الرموز الخاصة للكلمات المفتاحية، وهذه تختزل بعض الجهد المبذول من المبرمج.

- برمجة الاجزاء

من الممكن ان تكون تطبيقات C++ من عدد من الملفات لشفرة المصدر والتي تترجم بشكل منفصل، ثم يتم ربطها مع بعض، هذا يساعد على تقليل الوقت وليس من الضروري اعادة ترجمة كامل التطبيق عندما يتم عمل تغيير مفرد ولكن فقط الملف الذي يحتوية. بالاضافة لذلك، فان هذه الخاصية تسمح لربط شفرة C++ مع الشفرة الناتجة بلغات اخرى مثل المجمع (assembler) او C.

- التوافق مع لغة C

C++ هي البوابة الخلفية للتوافق مع لغة C، اي شفرة تكتب بلغة C سيكون من السهولة تضمينها في برنامج C++ دون الحاجة لاي تغييرات صعبة.

- السرعة

الشفرة الناتجة من تجميع C++ هي كفوءة جدا، وذلك بسبب كونها لغة ثنائية، فهي تعد من اللغات ذات المستوى العالي ومن اللغات ذات المستوى الواطيء فضلا عن صغر حجم اللغة نفسها.

## 1.7 أوامر المعالج الأولي The C++ Preprocessor Commands

### 1.7.1 الموجة #include

تعد هذه التعليمة الأشهر والأوسع أستعمالا بعد التعليمة (#define) في لغة C++، عمل هذا الموجة هو أنه يطلب من المعالج الاولي (preprocessor) اضافة محتويات الملف المطلوب مع هذه التعليمة (يذكر أسم هذا الملف بعد #include مباشرة ويكون محدد بين العلامتين (< >)) وحشرة في الملف المصدر، حيث يتم ضم وأحتواء هذا الملف مع ملف البرنامج عند التنفيذ، هذا الملف يدعى ملف التعليمات، ويعود السبب في ذلك الى ان بعض الايعازات داخل البرنامج تحتاج الى تعاريف ودوال يتضمنها هذا الملف.

## 1.8 المعرفات Identifires

كل البرامج تحتوي على نوعين من الرموز:

**النوع الاول**.. وهي **الرموز التي تعود الى اللغة**.. تستخدم هذه الرموز بطريقتين أما أن تكون على شكل رمز واحد أو أثنين مثل (**:، ;، ()، +، -**) أو على شكل كلمات تسمى الكلمات المحجوزه او الكلمات المفتاحية (KeyWords) مثل: (**if، else، while، do، inline**)

**النوع الثاني**.. هي **المعرفات** وهي عبارة عن رموز تستخدم في البرامج فأما أن تكون معرفات قياسية مثل (**char, int, float… etc**), أو أن تكون معرفات يتم أختيارها من قبل المبرمج، وهذه المعرفات الأخيرة نسميها أيضا المتغيرات (Variables)، و**المتغير** هو رمز أو أكثر يستخدم في البرنامج ليشير الى محتوى موقع في الذاكرة.

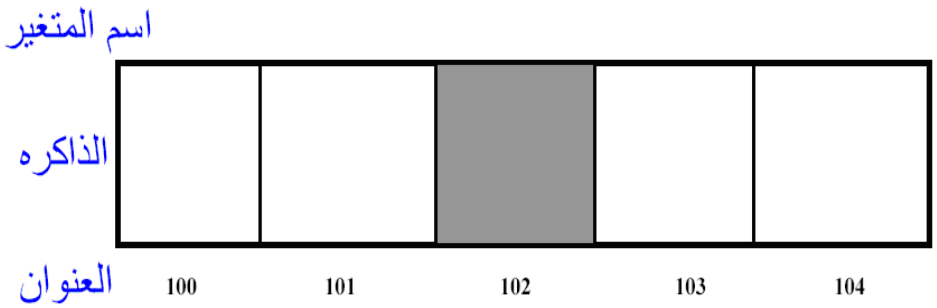
***ملاحظة://***

**المتغير..** في أغلب لغات البرمجة فأن المتغير هو مكان لتخزين المعلومات، المتغير هو مكان أو موقع في ذاكرة الجهاز حيث يمكن تخزين قيمة

بداخلة ثم أمكانية أستعادة هذه القيمة فيما بعد.

والمتغير هو أسم يمثل برقم أو سلسلة حرفية ( وممكن حرف واحد أو تعبير منطقي).

من الممكن تصور ذاكرة الجهاز على أنها مجموعة من المواقع التي تخزن فيها المعلومات، هذه المواقع مرقمة بشكل متسلسل تبدأ من الصفر وتنتهي بحجم الذاكرة، تدعى هذه الأرقام عناوين الذاكرة (Addresses)، يمثل أسم المتغير (بطاقة عنونة) ملصقة على أحد المواقع بحيث تستطيع الوصول اليه سريعا دون الحاجة الى معرفة العناوين الحقيقية في الذاكرة (لذا فان المتغير سيشير الى أحد هذه العناوين، وعند حاجتك وضع قيمة في الموقع الذي يشير له هذا المتغير فان المعالج (processor) سيذهب الى العنوان الذي يشير له المتغير ويضع فيه القيمة وكذلك عندما تريد أن تعرف قيمة المتغير فأن المعالج يذهب الى العنوان الذي يشير له المتغير ويقرأ القيمة التي فيه). يعرض الشكل التالي هذه الفكرة والتي تبين بعض المواقع في الذاكرة والتي من الممكن ان يشير اليها المتغير.




**شكل رقم (1.2).: بعض مواقع الذاكرة المنطقية**

***ملاحظة://***

لغة C++ تعد حساسة لحالة الأحرف ( أي أنها تميز بين الأحرف الكبيرة والصغيرة)، لذلك فأن الحرف الصغير يعد معرفا غير مساوي لشكلة الكبير ( أي أن (a) لا يساوي (A)). علما ان بعض لغات البرمجة لاتميز بين حالات الاحرف.

تتكون أسماء المتغيرات من " حرف واحد، مجموعة حروف، أو حروف وأرقام ومن الممكن استخدام الشارحة ".. على أن يكون دائما **أول رمز باسم المتغير حرف او شارحة** حتما مثل:

(x, ad, _count, endofpoint, end_of_point, Saad6, x345)

هذه جميعا متغيرات مقبولة.

أما المتغيرات التالية فهي متغيرات غير مقبولة:

(first name, next.word, 15may, Ten%)

والسبب هو أن المتغير الأول يحتوي على فراغ، الثاني يحتوي على نقطة، الثالث يبدأ برقم، أما الأخير فيحتوي على رمز لايمكن استخدامة مع المتغيرات.. وهذه جميعها غير مقبولة في البرنامج. أن أختيار المتغير من قبل المبرمج تعد مسألة مهمة ويفضل أن يعكس المتغير المعنى الذي يستخدم لأجلة المتغير فمثلا يفضل استخدام المتغير (sum) مع الجمع وأذا ما استخدم متغير أخر فان ذلك سوف لا يؤدي الى أي خطأ، وكذلك يفضل أن لا يكون المتغير طويل فمثلا يفضل استخدام متغير متكون من حرف واحد عندما نستخدمة في برنامج قصير ولا يتكرر كثيرا، أما استخدام متغير من حرف واحد ويستخدم بشكل متكرر وبأجزاء متكررة في برنامج طويل فأنه يعد أختيارا سيئا بالرغم من أنه لا يعيق عمل البرنامج.

## 1.9 البيانات Data

كل عنصر من البيانات في البرنامج أما أن تكون قيمتة ثابتة أو متغيرة ( قيمة المتغير ربما تتغير خلال تنفيذ البرنامج). كل متغير (والذي هو بيانات) في البرنامج يجب أن يكون له نوع وبموجب هذا النوع سيتم تحديد المساحة الخزنية اللازمة لقيمة هذا المتغير، وكذلك تحدد العمليات التي ممكن أجراؤها على هذا المتغير (تحدد لكل نوع عدد البايتات في الذاكرة التي تحجز لخزن قيم ذلك النوع وعند الكتابة في هذا الموقع فان الكتابة ستحدد بعدد بايتات هذا النوع أي لا يتم تجاوز هذا العدد من البايتات حتى وان كانت القيمة تتجاوز الحدود العليا والدنيا لهذا

النوع، وعند القراءة فانه سيتم قراءة القيم الموجودة في هذه البايتات فقط وبذلك تتجنب الخطأ في القراءة والكتابة). والأنواع القياسية التي تستخدم في لغة C++ هي:

### 1.9.1 الاعداد الصحيحة Integers

الأعداد الصحيحة هي كل الأعداد الموجبة والسالبة التي لا تحتوي على كسر. فالصفر عدد صحيح و 123 هو عدد صحيح و 45- أيضا عدد صحيح. أما (123.345 و 1.45-) فهي ليست أعداد صحيحة.

أن أي محاولة لاستخدام قيم خارج نطاق الحدود العليا والدنيا للأعداد الصحيحة سيؤدي الى حدوث خطأ. وبشكل عام فأن المتغيرات من نوع الأعداد الصحيحة تستخدم اضافة الى العمليات الرياضية في العدادات والفهارس.

العلاقات الرياضية التي تستخدم مع الأعداد الصحيحة هي( + , - , * , / , % ) وهي على التوالي (الجمع، الطرح، الضرب، القسمة، وحساب باقي القسمة).

**أمثله://**

21 / 3 = 7

9 / 2 = 4

2+3*4 = 14

هنا ينفذ داخل القوس أولا

(2+3) * 4 = 20

5 % 2 = 1

7 % 4 = 3

ويصرح عن الأعداد الصحيحة بلغة C++ في أي مكان داخل جسم البرنامج بالمعرف (int) والتي تعني (integer) وهي تكتب قبل المتغيرات، مثال

```
int  x ;
```

***ملاحظة://***

نتيجة قسمة عدد صحيح على عدد صحيح أخر هو عدد صحيح.

أما أذا كان أحد العددين هو حقيقي فأن النتيجة ستكون عددا حقيقيا، مثال

2.0 / 3 = 0.66666667

50 / 2.0 = 25.0

***ملاحظة://***

فضلا عن الارقام العشرية ( وهي التي أساسها عشرة والتي تستخدم بالأعمال الأعتيادية (9.. 0))، فان C++ تسمح لك باستخدام ثوابت من الأرقام وفق النظام الثماني (octal numbers) ( أساسها 8) وكذلك أرقام وفق النظام السادس عشر (hexadecimal) (أساسها 16). ولتنفيذ ذلك فاذا أردت تمثيل رقم بالنظام الثماني فضع ( 0) ( صفر) أمام الرقم للدلالة على أنه بالنظام الثماني، أما أذا وضعت (0x ) ( صفر ثم x) أمام الرقم فذلك يعني أن الرقم ممثل بالنظام السادس عشر. المثال اللاحق يمثل ثوابت بالانظمة الثلاثة وكل منها مكافيء للأخر (جميعا تمثل الرقم 75 خمس وسبعون):

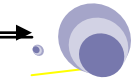
```
نظام عشري      //      75

نظام ثماني     //      0113

نظام سادس عشر  //      0x4b
```

**جدول (1.1): أنواع الأعداد الصحيحة وحجمها بالبتات**

| الحجم بالبتات | المـــــــــــدى | انواع البيانات |
|---|---|---|
| 16 | -32767...32767 | short |



| | | |
|---|---|---|
| int | -32767...32767 | 16 |
| long | -2147483647... 2147483647 | 32 |
| unsigned short | 0...65535 | 16 |
| unsigned | 0...65535 | 16 |
| unsigned long | 0...4294967295 | 32 |

### 1.9.2 الأعداد الحقيقية Real Numbers

وهي الأعداد التي تحتوي على كسور مثل

(10.0 , 356.67890- , 12.5 , 0.03 ). الأعداد الحقيقية ممكن أن تمثل بعدد صحيح وفارزة (تستخدم نقطة لتفصل العدد الصحيح عن الجزء الكسري)، وممكن أن تستخدم الرمز (e) والذي يمثل الرقم عشرة مرفوعا الى أس معين (الأس هو الرقم الذي يلي الحرف (e)) (الرقم الذي يلي الحرف (e) يجب ان يكون عددا صحيحا)، مثال

| | | | |
|---|---|---|---|
| 3.14159 | // | = | 3 .14159 |
| 6.02e23 | // | = | 6.02 x 10^23 |
| 1.6e-19 | // | = | 1.6 x 10^ -19 |
| 3.0 | // | = | 3.0 |

المثال أعلاه يحتوي على أربعة نماذج من الأرقام الحقيقية المقبولة في C++. العدد الاول يمثل (PI) (النسبه الثابتة) اما الثاني فهو يمثل عدد افوكادرو، الثالث يمثل الشحنة الكهربائية للألكترون (وهو عدد صغير جدا) وكل هذه الاعداد هي تقريبية، اما العدد الأخير فهو يمثل الرقم (3) ولكن كعدد حقيقي.

أما العمليات الرياضية التي ممكن أجراؤها على الأعداد الحقيقية فهي ( / , * , - , +) وهي على التوالي (الجمع، الطرح، الضرب، القسمة). ويصرح عن الأعداد الحقيقية في لغة البرمجة C++ في أي مكان داخل جسم البرنامج بالمعرف (float) التي تسبق المتغيرات، **مثال**

```
float  x;
```

***ملاحظة://***

تمثل الأرقام بطريقتين فأما أرقام صحيحة بدون كسر أو أرقام كسرية.
القواعد التالية تطبق عند كتابة أرقام في الحاسوب:

1. الفارزة ( , ) لا يمكن أن تظهر في أي مكان في الرقم.
2. ممكن أن تسبق الأرقام أحد العلامتين ( - ، +) للدلالة على كون الرقم موجب أو سالب ( يعد الرقم موجبا أذا لم تظهر أي من العلامتين على يسارة).
3. يمكن تمثيل الأرقام بطريقة العلامة العلمية (وذلك بأستبدال الرقم ( 10) بالحرف (e)). مثلا الرقم ( $2.7 \times 10^{-6}$ ) يكتب حسب العلامة العلمية كما يلي (2.7 e -6). كذلك فأن العدد ($6 \times 10^{12}$) يمكن ان يمثل حسب العلامة العلمية كما يلي (6 e 12), وكما وضح اعلاه.

***ملاحظة://***

يفضل عند استخدام التعريف (long ) وضع حرف (L ) بعد القيمة فمثلا :

```
long  SunDistance = 93000000 ;
```

هنا ستنتج قيمة مقدارها (12544- ) ويعطي المترجم رسالة تحذير ولتجنب ذلك تكتب كما يلي :

```
long SunDistance = 93000000L ;
```

***ملاحظة://***

أدناه بعض القواعد المهمة التي يجب أن تراعى عند كتابة العلاقات الرياضية :

أن وضع أشارة السالب قبل المتغيرات هي مكافأة لضرب المتغير بالقيمة

(-1). مثلا المتغيرات (x+y ) - من الممكن أن تكتب (( x+y) * (-1).

يجب أن تكتب العلاقات الرياضية وفقا للطريقة التي تحددها لغة البرمجة C++ بحيث تذكر كل العلامات الرياضية دون أختصار. مثال : العلاقه الرياضية الأتية غير مقبولة ((3x2 + x1 ) 2 ) هذه العلاقه لكي تكون مقبولة في لغة البرمجة C++ يجب أن تكتب بالشكل التالي: ( (2 * x1 + 3 * x2) ) العلاقة الأولى هي التي تعودنا على استخدامها في الرياضيات.

العدد المرفوع الى قيمة معينة سيضرب بنفسة عدد من المرات بقدر الأس اذا كان الاس عددا صحيحا ولا يهم فيما اذا كان الأساس سالبا أو موجبا.

لايجوز رفع القيمة السالبة الى أس كسري ( وذلك لأن حساب ناتج الرقم المرفوع الى أس كسري يتم بحساب اللوغاريثم للأساس، ويضرب هذا اللوغاريثم بالأس، وعندها يحسب معكوس اللوغاريثم، وأن اللوغاريثم للرقم السالب غير معرف لذا لايمكن أيجاد النتيجة).

العمليات الرياضية لايمكن أجراؤها على السلاسل الرمزية. مثال
("xyz" + 34 ) هذا غير مقبول وذلك لأن (xyz) هو سلسلة حرفية وليس عددا أو متغيرا رقمي (لاحظ أنه محصور بين علامتي أقتباس (quotation mark ) للدلالة على أنه سلسلة حرفية).

**جدول (1.2): أنواع الأعداد الحقيقية وحجومها بالبتات**

| نوع البيانات | المـــــــــــــــدى | الحجم بالبتات |
|---|---|---|
| float | $3.4x10^{-38}$. $3.4x10^{+38}$ | 32 |
| double | $1.7x10^{-308}$. $1.7x10^{+308}$ | 64 |
| long double | $3.4x10^{-4932}$.. $1.1x10^{+4932}$ | 80 |

### 1.9.3 الرموز Characters

وهي كافة الرموز التي تستخدم في الحاسوب والتي غالبا ما نجدها على لوحة المفاتيح والتي تشمل الحروف الأبجدية سواء كانت حروف كبيرة (**A..Z**) أو حروفا صغيرة (**a..z**)، الأرقام (**0..9**)، الرموز الاخرى التي نراها على لوحة المفاتيح مثل (etc ... ? , & , % , ! , + , / , .) وتستخدم بشكل مفرد. ويصرح عن الرموز بلغة البرمجة C++ في أي مكان داخل جسم البرنامج بالمعرف (char) التي تسبق المتغيرات.

أن أكثر مجاميع الحروف استخداما هما أثنان:

**ASCII**

(American Standard Code for Information International)

**EBCDIC**

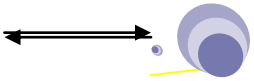
(Extended Binary Coded Decimal Information Code)

وكل منهم له صفاتة الخاصة به (لمزيد من المعلومات راجع الملاحق في نهاية الكتاب).

| |
|---|
| ***ملاحظة://*** |
| تكتب الحروف بين علامتي أقتباس مفردة ( ' ' ). |

- **عمليات الأحرف**

الأحرف تمثل داخل الحاسوب بواسطة أرقام صحيحة وفقا لنظام (ASCII) تسمى الأعداد الترتيبية (ordinal numbers)، لذا فأن المبرمج بأمكانة أن يمزج بين الرموز والأعداد الصحيحة بتعابير رياضية لتؤدي غاية معينة، فمثلا



أذا فرضنا أن المتغير الرمزي (ch) هو متغير من نوع حروف وتم أسناد قيمة له كما يأتي

(ch = 'S')

عليه فأن التعبير التالي ; ch = ch + 1

سيؤدي الى أن تكون قيمة المتغير الرمزي (ch) تساوي الرمز (' T ')، وكذلك فأن التعبير التالي ch = ch – 3

سيؤدي الى أن تكون قيمة المتغير الرمزي (ch) تساوي الرمز (' P ') وهذا يعتمد على القيم الرقمية التي تمثل الاحرف بنظام (ASCII).

**ملاحظة://**

الفرق العددي بين تمثيل الأرقام الكبيرة والأرقام الصغيرة هو (32) (اي ان الحرف الصغير اكبر من الحرف الكبير بالقيمة 32).

فمثلا أن قيمة الرمز (A = 65) حسب نظام (ASCII) بينما قيمة الرمز (a = 97) وفقا لنفس النظام. عليه فأذا كانت

ch = ' E ' ;

أذن ch = ch + 32 ;

ستؤدي الى أن تكون قيمة المتغير الرمزي ( ch = ' e ')

وكذلك أذا كانت قيمة المتغير الرمزي ch = ' d ' ;

فأن ch = ch – 32;

ستؤدي الى أن تكون قيمة المتغير الرمزي ( ch = ' D ')

العدد الترتيبي للصفر هو (48) لذا فأن الاعداد (0..9) تأخذ الأعداد الترتيبية ( 48 – 57 )

**ملاحظة://**

الرموز تحدد بعلامة اقتباس مفردة مثل (‘ 5 ‘ ) او (‘ { ‘ ) اما السلاسل الرمزية فهي تحدد بعلامة اقتباس مزدوجة مثل (“ 51 “ ) او (“ good “ ) بينما الارقام لاتحدد باي علامة مثل (5 ) او (456 ).

### 1.9.3.1 رموز الدلالة Directing Characters

وهي حروف خاصة عادة تستخدم مع الشرطة العكسية (/) لاعطاء تأثير معين يلاحظ ضمن مخرجات البرنامج. الجدول (1.3) يبين هذه الرموز مع التاثير الذي تحدثة.

وهذه تسمى ايضا سلاسل الهروب Escape Sequences. فالشارطة المعكوسة (\) التي تسبق بعض الاحرف تخبر المترجم بان هذا الحرف الذي يلي الشارطة المعكوسة ليس له نفس المعنى كما لو ظهر الحرف بنفسة دون هذه الشارطة المعكوسة (\). هذه السلاسل يتم كتابتها كرمزين دون وجود فراغ بينهما. بعض هذه السلاسل معرفة في C++.

اذا وضعت (\) او (“) في سلسلة حرفية ثابتة، فانك يجب ان تهرب من قدرة (“) على انهاء سلسلة حرفية ثابتة وذلك باستخدام (\”)، او قدرة (\) للهرب باستخدام (\\). ان استخدام (\\) تخبر المترجم بانك تعني شارطة معكوسة حقيقية (\)، وليست شارطة معكوسة لسلسلة هروب، وان (\”) تعني حاصرة حقيقية وليس نهاية سلسلة ثابتة.

لاحظ دائما تستخدم سلاسل الهروب مع حاصرتين مزدوجتين مثل (“\n”).

**جدول (1.3): رموز الدلالة في لغة C++**

| الرمز | الناتج (التأثير على المخرجات) |
|---|---|
| \a | صوت أو صفير (Beep) |

| | |
|---|---|
| \b | (Backspace) الترجيع خطوة واحدة للخلف |
| \f | (form feed) التغذية |
| \n | (new line) سطر جديد |
| \r | (carriage return) الاعادة او الرجوع |
| \t | (horizontal tabulator) الازاحة الأفقية |
| \v | (vertical tabulator) الازاحة العمودية |
| \\ | (Backslash) الشرطة المعكوسة |
| \' | (single quota)حاصرة مفردة |
| \" | (double quota)حاصرة مزدوجة |

### 1.9.4 النوع المنطقي (Boolean)

النوع الاخر هو النوع المنطقي والذي يرمز له (bool). هذا النوع اضيف حديثا الى لغة C++ بواسطة هيئة (ISO\ANSI) (منظمة المقاييس العالمية/ منظمة المقاييس الامريكية الوطنية).

التعابير المنطقية تشير الى واحدة من القيم وهي (صح، او خطأ). التعابير المنطقية تستخدم في التفرع او حلقات التكرار والتي سندرسها لاحقا.

## 1.10 التعابير المنطقية The Boolean Expressions

وهي التعابير التي تمثل نتيجتها بحالة واحدة من أثنتين وهما (صح أو خطأ) **(true OR false)**، وهناك ثلاث عوامل منطقية وهي **(Not، Or، And).**

التعبير المنطقي يعيد القيمة (1) عندما يكون التعبير (TRUE) والقيمة (0) عندما يكون التعبير (FALSE). وهي تستخدم لوصف أي تعبير فيما أذا كان صح أو خطأ. أن أنواع المتغيرات التي تستخدم لهذا الغرض يصرح عنها في حقل المتغيرات بالدالة (bool) (هذه عادة لاتجدها في جميع نسخ C++ وانما النسخ الحديثة فقط).

فمثلا عندما نعرف العبارة التالية على أنها من نوع القيم المنطقية كمايأتي

```
bool  c = (a= =b) ;
```

نلاحظ هنا اننا استخدمنا علامة المساواة للدلالة على ان نتيجة الطرف الايمن ستؤول الى المتغير في الطرف الأيسر بينما استخدمنا العلامة (==) وهي تستخدم لعمليات **فحص المساواة,** فاذا كان (a ، b) متساويان فان (c) ستكون قيمتها تساوي (true) وبخلاف ذلك تكون قيمتها تساوي (false).

### 1.11.1 العمليات المنطقية Logical Operators

هناك ثلاثة أنواع من العمليات المنطقية وهي **(NOT ، OR ، AND)** كل منها يتعامل مع التعابير الشرطية (أي التي تحتوي شرط). كل واحد من هذه التعابير له تأثير مختلف على التعابير الشرطية. أدناه أمثلة تبين كيفية استخدام هذه التعابير والتي من الممكن أن تستخدم بين تعبيرين أو أكثر من التعابير الشرطية.

**AND**

العامل (&&) يستخدم للدلالة على العامل المنطقي (and) في لغة C++ وهو يستخدم لمقارنة تعبيرين لتحصل على نتيجة منطقية مفردة، والنتيجة التي تحصل عليها تحدد بجدول الصدق (1.4) ادناة

**جدول (1.4): جدول الصدق للعامل (و) (&&) (And)**

| *A* | *B* | *A && B* |
|---|---|---|

| *A* | *B* | *A && B* |
|---|---|---|
| true | true | True |
| true | false | false |
| false | true | false |
| false | false | false |

**OR**

العامل ( || ) يستخدم للدلالة على العامل المنطقي (or) في لغة C++ وهو يستخدم لمقارنة تعبيرين لتحصل على نتيجة منطقية مفردة، والنتيجة التي تحصل عليها تحدد بجدول الصدق (1.5) ادناه:

**جدول (1.5): جدول الصدق للعامل (أو) (||) (Or)**

| *A* | *B* | *A \|\| B* |
|---|---|---|
| true | true | True |
| true | false | True |
| false | true | True |
| false | false | False |

( (5 = = 5) && (3 > 6)) // النتيجة خطأ (صح && خطأ)

( (5 = = 5) || (3 > 6)) // النتيجة صح (صح || خطأ)

**NOT**

لاحظ في لغة C++ فان العامل ( ! ) يمثل العامل (لا) (not) وهو يأخذ معامل واحد يتواجد في يمينة والعمل الوحيد الذي يقوم به هو عكس قيمتة (قيمة المعامل الذي على يمينة) فاذا كانت قيمتة (صح) تصبح خطأ واذا كانت خطأ تصبح صح. نتيجة استخدام العامل (لا) موضحه بالجدول (1.6)

**جدول (1.6 ): جدول الصدق للعامل (لا) (!) (Not)**

| A | ! A |
|---|---|
| true | False |
| false | True |

**مثال://**

النتيجة تصبح خطأ لان التعبير (5==5) هو صح // !(5==5)

النتيجة تصبح صح لان (6<=4) هي خطأ // !(6<=4 )

النتيجة تصبح خطأ // !true

النتيجة تصبح صح // !false

***ملاحظة://***

من الممكن ان تستخدم عوامل العلاقات المنطقية للمقارنة بين قيمتين ومن الممكن ان تكون هذه القيم من أي نوع من أنواع البيانات مثل (float, int, char…etc)، او ممكن أن تكون ( كما سنرى لاحقا) اصنافا معرفة من المستخدم.

أن نتيجة المقارنة أما أن تكون ( صح او خطا) (true ،false). فمثلا العبارة التالية

```
cout <<  5  <  23 ;
```

ستطبع القيمة (1) لان العبارة صحيحة.. اما العبارة التالية

```
cout <<  45  > 60 ;
```

ستطبع القيمة (0) لان النتيجة خاطئة

***ملاحظة://***

العامل (NOT ) يختلف عن العاملين السابقين اذ أنه يتقبل مدخلا واحدا ودائما يعكس حالة العبارة التي يدخل عليها فأذا كانت صحيحة يجعلها خاطئه وأن كانت خاطئة يجعلها صحيحة.

***ملاحظة://***

أن أسناد قيمة لمتغير من نوع معين خارج المدى المحدد له سيؤدي الى حدوث خطأ، هذا الخطأ أما أن يوقف التنفيذ أو يؤدي الى ظهور نتائج غير متوقعة.

## 1.11 الأعلان عن المتغيرات Declarations

يتم الاعلان عن المتغير وذلك بان يتم كتابة النوع أولا ثم يتبع ذلك اسم المتغير والذي يجب ان يخضع للقواعد المذكورm انفا فمثلا:

```
int  a;
float  mynum ;
```

وبالأمكان الأعلان عن أكثر من متغير من ذات النوع بنفس الطريقة أعلاه على أن تفصل فارزة بين أسم متغير وأخر، مثال:

```
int  x ،y ،z ;
```

وهذه تكافيء الأعلان التالي

```
int  x ;
int  y ;
int  z ;
```

الطريقتان صحيحتان والفرق هو ان الأولى أكثر اختصارا.

***ملاحظة://***

بالامكان استخدام (signed، unsigned) لوحدهم، وتعني انها من نوع الاعداد الصحيحة مثال

```
unsigned   nextpage ;
unsigned int   nextpage ;
```

العبارتان متكافأتان

## 1.12 الثوابت Constants

في بعض البرامج تحتاج الى استخدام قيم ربما تكون معروفة مسبقا قبل تنفيذ البرنامج ولا يمكن أن تتغير داخل البرنامج مثل النسبة الثابتة (Л) والتي قيمتها (3.1415926585) هذه القيم الثابتة سواء كانت ذات قيمة معروفة مسبقا أو أي قيمة ممكن أن تسند الى متغير، جميعها ممكن أن يعلن عنها في أي مكان من جسم البرنامج وباحدى الطرق التالية، الأعلان عنها (باستخدام الكلمة المفتاحية (const)، استخدام الكلمة المفتاحية (enum)، أو باستخدام الموجة (#define)) والتي تسبق أنواع البيانات للمعرف المراد تعريف قيمتة على انها ثابتة.

***ملاحظة://***

المعرفات التي تعرف على أنها ثوابت لا يمكن ان تتغير قيمها أثناء تنفيذ البرنامج بأي شكل من الأشكال.

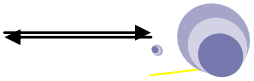
- const

وهي تسبق انواع البيانات لتعرف واحد او أكثر من المتغيرات على أنها ثابتة وفقا للصيغة القواعدية التالية:

```
const   TYPE   variable_name = value ;
```

مثال:

```
const    float    Pi = 3.1413926535 ;
```



```
const    string  Error = ' Run_Time Error ' ;
```

- **Enum**

وهي تستخدم لتعريف قائمة من المتغيرات على أنها ثابتة وفقا للصيغة القواعدية التالية:

**enum** TYPE *{CONSTANT1=value ،CONSTANT2 = value,...}*;

وسناتي عليها لاحقا لتوضيح عملها بشكل اكثر تفصيلا

- **الموجة (التعليمة) #define**

وهي تقوم بتعريف رموز كثوابت، وبالرغم من عدم شيوع استخدام هذا الهيكل في لغة (C++)، ولكن بالامكان استخدامة لتعريف المتغيرات الحسابية أو الرمزية في بداية البرنامج وتعوض قيمتها الحسابية أو الرمزية في أي مكان تذكر فيه هذه الأسماء في البرنامج وتستخدم الحروف الأبجدية الكبيرة عادة لتعريف أسماء هذه المتغيرات. مثال:

```
#define    TRUE  1
#define    PI    3.1415927
#define    EOF   -1
```

***ملاحظة://***

هذا الهيكل شائع في لغة (C)، وان كل ما موجود في لغة (C) ممكن استخدامة في لغة C++ .. العكس ليس صحيح

***ملاحظة://***

من الممكن الأستعاضة عن (#define ) بالكلمة المفتاحية (const ) مثال

```
const    TRUE = 1
const    PI = 3.1415927
```

مع ملاحظة استخدام علامة المساواة

### 1.12.1 أسباب استخدام الثوابت:

- أذا كان هناك عدد يستخدم بشكل متكرر داخل البرنامج فأن المبرمج يفضل أن يصفة بأسم يشار اليه على أنه يحمل قيمة ثابتة.
- من الممكن استخدام الثوابت لتسمية متغيرات من نوع السلاسل الرمزية والتي تستخدم بشكل متكرر في مخرجات البرنامج وهي في جميع الأحوال تستخدم لتسهيل العمل البرمجي.

**مثال:**

نفرض أننا نحتاج الى طباعة أسم جامعة مثلا بشكل متكرر في البرنامج، ممكن أن نقوم بمايأتي:

```
const  string  University  = "Al _ Kufa University " ;
const  string  Underline   = "--------------------------" ;
```

الأن من الممكن استخدام الأسماء المعرفة كثوابت في البرنامج وكما يأتي:

```
cout << University << endl ;
cout <<  Underline  ;
```

***ملاحظة://***

يستخدم تعريف الثابت في أي مكان داخل جسم البرنامج، وان أي محاولة لتغيير قيمتة أثناء تنفيذ البرنامج سيؤدي الى صدور رسالة خطأ.

## 1.13 العوامل Operotors

عند وجود المتغيرات والثوابت، فبامكانك القيام بالعديد من العمليات عليها مستخدما العوامل المناسبة لكل عملية.. منها:

### 1.13.1 عامل التخصيص ( = Assignment)

عامل التخصيص واجبة اسناد قيمة الى متغير مثل

```
A = 7 ;
```

هنا تم أسناد القيمة (7) الى المتغير (A) ودائما تسند القيمة في الجانب الأيمن من عامل التخصيص الى المتغير في الجانب الأيسر من التخصيص.

تختلف C++ عن اللغات الأخرى بأمكانية استخدام علامة التخصيص في الجانب الأيمن أو ان تكون جزء من الجانب الأيمن لعملية تخصيص أخرى مثال

```
A = 8 + (b = 4) ;
```

وهي تكافيء العبارات التالية

```
b = 4 ;
A = 8 + b ;
```

كذلك فان التعبير التالي مقبول أيضا

```
A = b = c = d = 6 ;
```

### 1.13.2 العمليات الرياضية ARITHMETIC OPERATORS

**( + , - , * , / , % )**

وهي العمليات المعروفة لنا في الرياضيات، والتي هي (الجمع، الطرح، الضرب، والقسمة)، يضاف لها عامل أخر وهو أستخراج باقي القسمة باستخدام العلامة (%) الجدول (1.7) يبين هذه العمليات:

**جدول (1.7): يبين العمليات الرياضية التي تدعمها لغة C++**

| العملية الرياضية | العامل |
|---|---|
| الجمعAddition | + |
| الطرحSubtraction | - |
| الضربMultiplication | * |
| القسمةDivision | / |
| أستخراج باقي القسمة Modulo | % |

### 1.13.3 المساواة المركبة Compound Assignment

وهي استخدام المساواة مع عوامل اخرى

**( + = ، - = ، * = ، / = ، % = ، > = ، < = ، & = ، | = )**

عندما نرغب بتحوير قيمة متغير بأنجاز عمليات رياضية على القيمة المخزونة حاليا بالموقع الذي يشير له المتغير فاننا يمكن ان نستخدم عوامل المساواة المركبة، هذه العمليات تستخدم بطريقة مختلفة عن العمليات المتعارف عليها حيث ان العوامل الموجودة مع المساواة هي جميعا عوامل ثنائية أي تستخدم مع أثنين من المتغيرات أو القيم، وجميعها تستخدم وفقا للقاعده التالية:

حيث يستخدم العامل على الجانب الأيسر من المساواة لأجراء العملية الرياضية أو المنطقية بين المتغير في الجانب الأيسر من المساواة مع المتغير أو القيمة على الجانب الأيمن من المساواة، وتسند النتيجة الى المتغير الذي في الجانب الأيسر من المساواة. مثال يوضح ذلك في الجدول (1.8):

**جدول (1.8): أمثله توضح استخدام المساواة المركبة**

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

***ملاحظة://***

لايجوز ان يكون في الطرف الايسر من ( المساواة) تعبير وأنما يكون متغير ومتغير واحد فقط.

### 1.12.4 الفاصلة (,) كأداة The comma (,) operator

وهي أداة ثنائية (binary) وتحتل الاسبقية الأخيرة في سلم أسبقيات الأدوات المختلفة، وتأخذ الصيغة العامة:

Expression1 ، Expression2

وتستخدم لفصل تعبيرين على يمين المساواة، فعند استخدام فاصلة لتفصل بين تعبيرين، فأن تسلسل العمليات يأخذ الترتيب التالي:

1. تستخرج قيمة التعبير الأول الذي على يسار الفاصلة (الفارزة) ثم تسند للتعبير الثاني على يمين الفاصلة (الفارزة).
2. تستخرج قيمة التعبير الثاني الذي على يمين الفاصلة (الفارزة) كقيمة نهائية لكامل التعبير.

**مثال:**

A = (b = 2 ، b+1) ;

في هذا المثال سيعمل المترجم على يمين المساواة كما هو متعارف، اذ سيسند القيمة (2) الى المتغير (b) (يبدأ أولا بالتعبير الذي على يسار الفاصلة)،

المرحلة الثانية, العمل على التعبير الذي موجود على يمين الفاصلة في هذه الحالة فأن قيمة (b) هي (2) ومنها يستخرج القيمة النهائيه للتعبير (b+1) لتكون النتيجة هي (3) وهي تمثل نتيجة التعبيرين على يمين المساواة والتي ستسند الى المتغير (A) على يسار المساواة.

### 1.13.5 عوامل المساواة والعلائق Relation And Equality Opetotors

وتستخدم هذه العوامل لأغراض المقارنة، وهي (==، =!، <، >، =>، =<) والجدول (1.9) يوضح استخدام هذه العوامل.

**جدول (1.9): عوامل المساواة والمقارنة المستخدمة في لغة C++**

| العامل | استخدامة |
|---|---|
| = = | تساوي |
| =! | لا تساوي |
| > | أكبر من |
| < | أصغر من |
| => | أكبر من أو تساوي |
| =< | أصغر من أو تساوي |

لغرض المقارنة بين تعبيرين فانك يمكنك ان تستخدم عوامل العلاقات والمساواة. نتيجة عملية المقارنة هي قيمة منطقية (Boolean) اي (صح او خطأ) وفقا للنتيجة. **مثال**

(7 = = 5) // النتيجة خطأ

(5 > 4) // النتيجة صح

(3 =! 2) // النتيجة صح

(6 > = 6) // النتيجة صح

(5 < 5) // النتيجة خطأ

بالطبع بدلا من استعمال قيمة رقمية ثابتة واحدة فانك بامكانك استعمال اي تعبير مقبول يتضمن متغيرات، كمثال نفرض ان

(a = 2 ،b = 3 ،and  c = 6)

ولنلاحظ العلاقات التالية

(a = = 5)  //  النتيجة خطأ لان (a ) لاتساوي (5)

(a*b >= c)  //  النتيجة صح حيث ان (2*3 >= 6)

(b+4 > a*c)  //  النتيجة خطأ حيث ان (3+4 > 2*6) هي خطأ

((b=2) = = a)  //  النتيجة صحيحة

عند كتابة تعبير معقد يحتوي على عدد من العمليات ربما يحدث لنا بعض الغموض عن كيفية أجراء العمليات الرياضية بمعنى أي من المعاملات يحسب أولا وأيهما لاحقا مثال:

a = 5 + 7 % 2

ربما يكون هناك غموض فهل هذا التعبير يعني التعبير اللاحق الاول ام التعبير اللاحق الثاني

مع نتيجة قدرها (6) او  //  a = 5 + (7 % 2)

مع نتيجة قدرها صفر  //  a = (5 + 7) % 2

النتيجة الصحيحة هي التعبير الاول مع نتيجة قدرها (6)، وذلك لأعتمادنا على ترتيب لأسبقيات حساب العوامل (جدول 1.10 يبين الاسبقيات) وهي ليست للعوامل الحسابية فقط وأنما لكل العوامل التي تظهر في C++.

## 1.14 التعبير Expression

أي ترتيب من المتغيرات والعوامل الرياضية والذي في النهاية يمثل عملية حسابية يسمى **تعبير**، والتعبير عبارة عن أشتراك عناصر البيانات مع العوامل الحسابية وهذه العناصر ممكن ان تكون ثوابت، متغيرات، تعابير، وعند أجراء

العملية الحسابية فان النتيجة تكون قيمة واحدة.. ومن الممكن ان يكون جزء من التعبير تعبير أيضا.. مثل

( a+20) * b/3

هذا كله يسمى تعبير واجزاءه مثل (a+ 20) و (b/3) كل منها يسمى تعبير أيضا.

وتستخدم مع التعبير عادة عبارة الأسناد (assignment statement) وهي علامة او عبارة تستخدم لأسناد قيمة الى متغير وتستخدم علامة المساواة (=) لتحقيق هذا الغرض.. وبالتأكيد فان العملية ستتم باسناد القيمة المستحصلة من الطرف الأيمن من المساواة الى المتغير الموجود في الطرف الأيسر من المساواة.

بالأمكان كتابة تعبير معين يحتوي على متغيرات من أنواع بيانات مختلفة، مثلا تعبير يحتوي على متغيرات من نوع بيانات صحيحة وبيانات من نوع بيانات حقيقية.. في هذه الحالة فان عملية تحويل ألية داخل الحاسوب ستتم دون تدخل المستخدم حيث سيتم تحويل المتغيرات ذات النوع الاقل اسبقية الى النوع الاكثر اسبقية، الجدول (1.10) يبين أسبقيات العوامل:

**جدول (1.10): يبين اسبقيات العوامل**

| قواعـــــــد الأسبـــقيات | | |
|---|---|---|
| The Unary Operators<br>العوامل الاحادية | !، --، ++، -، + | الاسبقية العليا<br>(تنفذ اولا) |
| The Binary Arithmetic Operations<br>العوامل الرياضية الثنائية | %، /، * | |
| The Binary Arithmetic operations<br>العوامل الرياضية الثنائية | -، + | |
| The Boolean operations<br>العوامل المنطقية | >=، =<، <>، < | |
| The Boolean operations<br>العوامل المنطقية | !=، == | |
| The Boolean Operations<br>العوامل المنطقية | && | |
| The Boolean Operations<br>العوامل المنطقية | \|\| | الاسبقية الدنيا<br>(تنفذ اخيرا) |

## 1.15 توليد الأرقام العشوائي Random Numbers Generation

تحتاج بعض التطبيقات الى استخدام أرقام عشوائية، وهذا ممكن في لغة البرمجة C++ وذلك من خلال استخدام الأمر (Random) الذي يعمل على توليد رقم بشكل عشوائي، وهو يعمل وفقا لما يأتي:

* يستخدم الأمر (random) لتوليد أرقام عشوائية من نوع الأعداد الصحيحة تتراوح قيمتها بين الصفر والواحد. والامر (random) هو ماكرو معرف في (stdlib).

randomize : وهي تستخدم لتوليد أساس للأرقام العشوائية التي ستعتمد على الوقت

```
randomize ;
x  = random ;
```

هنا المتغير (x) تكون قيمتة (0 <= x < 1) وفي كل مرة يتم تنفيذ هذا الأمر سنحصل على قيمة جديدة ضمن نفس المدى.

* **الطريقة الثانية:** هي باستعمال الأمر (Randomize), ثم الأمر (Random) على أن يحتوي الأمر (Random) على المدى المطلوب لأيجاد الرقم العشوائي ضمنه (أي أنه سيولد أعداد صحيحة موجبة عشوائيا تتراوح قيمتها بين الصفر والعدد المحدد بين القوسين بعد (Random) ناقص واحد والذي يمثل الحد الأعلى)، **مثال**:

```
Randomize ;
x = random (100) ;
```

هنا تكون قيمة المتغير (x) (0<= x < 100) وفي كل مرة يعاد تنفيذ هذا الأمر ستحصل على قيمة جديدة. أن المدى المحدد يمكن تغيره حسب طبيعة التطبيق المراد تنفيذة.

* الطريقة الثالثة: لاستخدام الأمر (Random) هي بدون استخدام الأمر (Randomize) وبدلا منه استخدم المتغير (Randseed) قبل الأمر (Random) على أن يتم أسناد قيمة للمتغير (Randseed ). من المفروض ان يتم تغيير قيمة (Randseed) عند كل تنفيذ لكي نحصل على عشوائية. **مثال**

```
randseed = 1200 ;
x = random ;
OR
randseed  = 3425 ;
x = random (1000) ;
```

في الحالة الأولى فأن المتغير (randseed) أسند له قيمة وهي (1200) ووفقا لها سيولد أرقام عشوائية حقيقية قيمتها أقل من واحد ولو أعدنا التنفيذ مع أسناد قيمة

مختلفة للمتغير (randseed) فأن أرقام عشوائية مختلفة ستولد (حاول تنفيذ الطريقتين ولاحظ الفرق).

أما المثال الثاني فأنه سيولد أرقام عشوائية صحيحة أكبر من الصفر وأصغر من (1000).

## 1.16 التعليقات Comments

تعد التعليقات من الأمور المهمة في البرنامج، واغلب المبرمجين لايستعملونها بشكل كاف. عليك ان تدرك ان ليس كل الناس بدرجة الذكاء التي يتمتع بها المبرمج.. فضلا عن أنك تحتاج أحيانا الى شرح وتوضيح أكثر لبيان الفكرة او الغاية من كتابة عبارة او أيعاز معين او واجب هذه العبارة ضمن البرنامج. كذلك، فان المبرمج ربما لايتذكر بعد مضي شهر او أكثر التفاصيل الكافية وراء كتابة عبارة او أيعاز معين ضمن البرنامج.. لذلك تستخدم التعليقات التي تكتب على البرنامج وفقا لقاعدة كتابتها التي سناتي عليها لتشرح لمن يقرأ البرنامج ماذا نحن عاملون. ولما كانت التعليقات تكتب أمام عبارات البرنامج لذلك يفضل ان تعطي الصورة العامة وليس التفاصيل الدقيقة جدا والتي تكفي لتوضيح الفكرة. وبشكل عام فان التعليقات لاتعد جزء من البرنامج وسيهملها المترجم عند ترجمة البرنامج.

التعليقات نوعان.. الأول يبدأ بخطين متوازين (//) وهنا المترجم سيعتبر مابعد الخطين تعليق ليس له علاقة بالبرنامج ويبدأ التعليق من الخطين المتوازيين وينتهي بنهاية السطر.مثال

```
int  x ;  // تعليق قصير
```

أما النوع الثاني فهي التعليقات التي من الممكن أن تكون على عدة أسطر فيتم تحديد نص التعليق بواسطة ( /* و */ ) وهي مفيدة مع التعليقات الطويلة، اذ يستعمل الرمز (/*) لبداية التعليق والرمز (*/) للدلالة على نهاية التعليق.مثال

```
int  x ;      /*      هذا هو تعليق على عبارات البرنامج
        وهو تعليق طويل يراد منه توضيح أسباب استعمال نوع البيانات
```

لذلك أضطررنا الى استعمال عدة سطور من التعليق... الخ */

يجب أن تلاحظ مايلي عند كتابة تعليق:

1. عدم ترك فراغ بين الشرطة (/) والنجمه (*) من كل جهات جملة التعليق.
2. يقوم مترجم C++ بأهمال النصوص المستعملة في جملة التعليق (أي لا ينفذها).
3. من الممكن وضع جملة التعليق في أي مكان من البرنامج، ما عدا وسط الأسم التعريفي (identifier) أو الكلمة المحجوزة (keyword). فمثلا الأمثلة أدناه غير مقبولة:

```
* whi /* name = ' saad ' */ le     c = ' Ahmed '
* Sum =  /* xxx  */  0 ;
```

4. لا ينصح بوضع تعليق داخل تعليق أخر، لأن ذلك قد يتسبب بحدوث أخطاء، مثال

```
/* Program /* written by Saad */ card game */
```

هنا المترجم سيعتبر الجمله التعليقية تنتهي عند (Saad)، والباقي سيعتبره خطأ.

5. يهمل المترجم السطر أو بقية السطر الذي يبدأ بخطين مائلين (//).

## 1.17 عامل الزيادة Increment Operator

تستعمل في بعض التطبيقات عدادات لأغراض محددة وهي عادة تبدأ بالرقم (0) أو أي رقم أخر وتزداد بمقدار واحد (او اكثر) في كل مرة وتكتب عادة كما ياتي:

```
count = count + 1 ;
```

ونظرا لأن هذا العامل واسع الاستعمال لذا فأن لغة C++ وفرت عامل مفرد (للأختصار) لهذا الغرض وهو (++) لأغراض الزيادة بمقدار واحد أو (- -)

لأغراض النقصان بمقدار واحد حيث يستخدم هذا العامل بطريقتين أما أن يسبق المتغير مثل (++m) او أن يلي المتغير مثل (m++) وهما ليسا متشابهين فكل منهما له معنى خاص فعندما يسبق المتغير عامل الزيادة فان المتغير تزاد قيمتة بمقدار واحد ثم يستخدم أما أذا جاء عامل الزيادة بعد المتغير فان المتغير يستخدم حسب قيمتة الحالية وبعدها يزاد بمقدار واحد. أما العامل (- -) فتعمل بالطريقة نفسها التي يستخدم فيها عامل الزيادة أي قبل وبعد المتغير مع الأختلاف ان استخدامها يقلل قيمة المتغير بمقدار واحد، مثال

أذا فرضنا ان المتغير (b = 7) والمتغير (a = 2) فان قيمة (C) في التعبير التالي:

```
C = a * ++b ;
```

تكون قيمتها (16)، حيث ان المترجم سيقوم بزيادة قيمة (b) لتكون (8) ثم يعوض عنها في التعبير ويحسب نتيجة التعبير، أما قيمتها في التعبير التالي:

```
C = a* b++ ;
```

فتكون (14)، حيث ان المترجم سيستخدم القيمة الحقيقية للمتغير (b) ثم يقوم بحساب نتيجة التعبير وبعد ذلك تتم زيادة قيمة المتغير (b) لتكون (8)

```
C = a* --b ;
```

هنا قيمة (C) تكون (12) حيث سيقوم المترجم بأنقاص قيمة (b) بواحد لتكون قيمتة (6) ثم تعوض قيمتة في التعبير لايجاد قيمة (C)

أما قيمتها بالتعبير التالي:

```
C = a* b-- ;
```

تكون (14) حيث ستستخدم قيمة (b) الحقيقية (7) لأيجاد قيمة (C) بعدها تقلل قيمة (b) لتكون قيتها (6).

## 1.18 بعض المحددات الخاصة

هذه المحددات ستكون اكثر وضوحا في الفصول الاخرى وسيتم شرحها بالتفصيل, هنا فقط يتم الاشارة لها.

### 1.18.1 المحدد (متطايرة) volatile

بعكس المحدد (const) الذي يؤدي الى جعل قيمة المتغير ثابتة فأن المحدد (volatile) يؤدي الى جعل قيمة المتغير تتغير كلما تطلب الأمر ذلك بدون سيطرة المترجم أو توجيه تحذير الى المبرمج، وهذا المحدد مفيد في العمليات المتعددة التي تأخذ معلوماتها من الذاكرة. وبعبارة أخرى يحتاج المبرمج الى استخدام (volatile) عندما يتعامل البرنامج مع البرامج الفرعية ذات العلاقة المباشرة بالمكونات المادية للحاسوب .. مثال

```
volatile print_register ;
volatile port ;
volatile A[10] ;
```

### 1.18.2 المحدد (المسجل) register

يستعمل هذا المحدد لأعلام المترجم أن يحفظ قيم المتغيرات في مسجلات (registers) وحدة المعالجة المركزية (CPU) مباشرة، وليس في الذاكرة حيث تخزن عادة قيم المتغيرات. وهذا يعني أن العمليات التي تجري على هذا النوع من المتغيرات تكون أسرع من العمليات التي تجري على المتغيرات المخزنة في الذاكرة. ومما تجدر الاشارة له أن المحدد (register) يتعامل مع نوعين من المتغيرات هما الأعداد الصحيحة والرموز (characters) كما أنه يستعمل في حالات المتغير الموضعي أو متغير الدالة اللذان يعتبران من نوع المتغيرات الذاتية (Auto)، ولذا فأن (register) لا تستعمل للمتغير العام، وتستخدم هذه المتغيرات في برامج التكرار (Loops)، مثال

```
register  int  i ;
for (i = 0 ; i < last ; ++ i)
```

أن عدد المتغيرات من هذا النوع يعتمد على نوع المعالج المستعمل وعلى تطبيقات C++ فمثلا في الأنظمه ذات (bit8) يستخدم متغير واحد وفي نظام (bits16 ) يستخدم متغيران.

وكمبرمج بلغة C++ يمكنك استخدام أي عدد من هذه المتغيرات لأن المترجم سيسجل الفائض من هذه المتغيرات كمتغيرات عادية وليس متغيرات (register) بشكل تلقائي.

وينصح باستخدام متغيرات (register) في التطبيقات التي تستخدم حلقات التكرار (Loops) عادة.

## 1.19 الأدوات الدقيقة Bitwise Operators

تتميز لغة C++ عن سائر لغات البرمجة الراقية باستخدامها أدوات دقيقة تعمل على مستوى وحدة التخزين الاولية (bit)، وسميت هذه الأدوات بالدقيقة لأنها تتعامل مع البت بشكل مباشر، فحصا، ضبطا، وازاحة. وتستعمل هذه الأدوات مع البيانات الصحيحة (int) والرمزية (char) فقط. ولا تستعمل مع غيرها من البيانات والجدول (1.11) يوضح الأدوات الدقيقة وعملها:

**جدول (1.11): الأدوات الدقيقة واستخداماتها**

| العوامل الدقيقة | العمليات الرياضية المكافئة | استخدامها (عملها) |
|---|---|---|
| & | AND | تقوم بعملية (و) بين البتات Bitwise AND |
| \| | OR | تقوم بعملية (أو) بين البتات Bitwise Inclusive OR |
| ^ | XOR | تقوم بعمل (xor) بين البتات |

| | | |
|---|---|---|
| | | Bitwise Exclusive Or |
| ~ | NOT | عكس قيمة البت bit inversion |
| << | SHL | أزاحة البتات لليسار Shift Left |
| >> | SHR | أزاحة البتات لليمين Shift Right |

1. النفي يحول كل صفر الى واحد وكل واحد الى صفر.
2. أدوات الازاحة أستعمالها يؤدي الى أزاحة قيمة المتغير الصحيح (الممثل بالنظام الثنائي) يمينا أو شمالا عدد من الخانات (البتات) وحسب الطلب، وتملأ الخانات المفرغة أصفارا أو واحدات حسب أشارة العدد (فالعدد الموجب عند أزاحة تملأ فراغاتة أصفار، بينما العدد السالب تملأ فراغاتة واحدات عند أزاحة)، مثال..

أذا أردنا أزاحة المتغير (X) الى اليمين خانتين فيكتب كمايأتي:

```
X >> 2;
```

**جدول (1.12): جدول يبين أسبقيات العمليات الدقيقة**

| | أسبقيات الأدوات الدقيقة |
|---|---|
| 1 | ~ |
| 2 | <<، >> |
| 3 | & |
| 4 | ^ |
| 5 | \| |

***ملاحظة://***

للتأكد من سلامة نتائج عمليات الأزاحة فمن الممكن استخدام القاعدة التالية:
كل ازاحة الى اليمين بمقدار بت واحد ينتج عنها قسمة القيمة المزاحة على ( 2)
(أي لكل بت ازاحة نقسم العدد على 2)

كل ازاحة الى اليسار بمقدار بت واحد ينتج عنها ضرب القيمة المزاحة بالرقم ( 2) ( أي لكل بت أزاحة نضرب العدد في 2)

## 1.20 تحويل نوع البيانات Type Conversions

عند استخدام أكثر من نوع من البيانات في تعبير معين، فأنه من الممكن أن نحول نوع متغير معين ضمن التعبير الى نوع أخر، وذلك بأجراء التحويل على المتغير الموجود الى يمين المساواة، ليصبح نوعة حسب نوع المتغير في جانبها الأيسر. مثال

```
int   a,b ;
char  name;
float   x ;
name = a ;    b = x ;   x = name ;     x = a ;
```

نلاحظ أن هذه الأشكال من التحولات بين أنواع البيانات غير موجودة في العديد من اللغات الأخرى، وذلك لأن C++ صممت أصلا لتكون لغة وسيطة بين اللغات العليا ولغة التجميع (Assembly).

*** تغيير نوع المتغير:**

ان تغيير نوع المتغير هو اسم معقد لمفهوم بسيط. فعند تغيير نوع المتغير من نوع الى اخر، فان كل الذي تعملة هو اخبار الحاسوب باستعمال نوع مختلف لخزن المتغير. اذن لماذا نحتاج الى عمل ذلك؟ دعنا نقول بانك اعلنت عن متغير من نوع short، في اغلب الاحيان ان هذا يعني ان اكبر قيمة موجبة من الممكن ان تخزنها ستكون 32,767، ولكن في مكان ما في البرنامج، ادركت انك ستقوم بعملية حساب ستؤدي الى زيادة القيمة فوق هذه القيمة العظمى. فمثلا لحساب طول c (وتر المثلث القائم الزاوية)، فانك تحتاج الى حساب الجذر التربيعي لمربع الظلعين الاخرين $a^2 + b^2$، ولكن ماذا يحدث لو كانت قيم كل من a، b كبيرة جدا، عليه سيكون التربيع

كبيرا جدا، فاذا اصبحت القيمة اكبر من 32,767 فان قيمتك ستكون ليس كما تتوقع (اذا استخدمت النوع short لخزن الناتج) ستكون قيمة الناتج غير صحيحة.

عليه فان الحل هو تغيير النوع، فبامكانك ان تغير النوع للارقام الى نوع بيانات اكبر، مثل (long ,int) لاغراض الحساب.. وبعدها من الممكن اعادتها ثانية الى short عند الانتهاء، اذ ان القيمة النهائية للمتغير c ربما ستكون صغيرة بما يكفي ان تخزنها بالنوع short. في الحقيقة هذا مثال بسيط ويمكن حل المشكلة بان تخزن المتغير من البداية بالنوع int ، مثال اكثر فائدة يحدث اذا كان لديك رقم والذي يمثل معدلا مثلا, فانك ربما ترغب ان تمثل الرقم بالنوع float لتكون القيمة اكثر دقة عند حسابها. ويمكن تغيير النوع ليكون int.

**كيف يتم تغيير النوع:**

عملية تغيير النوع في C++ عملية سهلة. لنقل لديك المتغير (average) من النوع float لخزن رقم مثل الرقم (0.314188526) وترغب ان يكون لديك خزن من نوع int لخزن جزء العدد الصحيح من الرقم اعلاه. ادناه كيف تعمل ذلك:

```
int  CastAverage = (int) average ;
```

لاحظ هنا اننا اعلنا عن متغير (CastAverage) من النوع int لنضع فيه القيمة بعد تغيير النوع وهنا اننا غيرنا النوع وذلك بوضع النوع الذي نرغب ان نغير نوع المتغير اليه نضعة بين قوسين قبل اسم المتغير.

***ملاحظة ://***

عند التحويل من البيانات الطويلة الى أخرى أقصر فأن عدد من الخانات ( البتات) ستفقد.

***ملاحظة://***

أن التحويل بين نوع وأخر من أنواع البيانات، يتم بصورة تلقائية ( أوتوماتيكية) داخل التعبير الواحد، اذ يقوم مترجم C++ بتحويل جميع المتغيرات الى النوع ذي الطول الأكبر، فيتحول الصحيح الى حقيقي ويتحول الحقيقي الى مضاعف وهكذا.

### 1.20.1 عامل تحويل النوع الخارجي Explicit Type Casting Operator

عامل تحويل النوع يسمح لك بتحويل نوع معين الى نوع أخر. هناك عدة طرق لعمل ذلك في C++، أبسط طريقة والتي ورثت من لغة C هو بأن تسبق التعبير المراد تحويلها بالنوع الجديد محاط بقوسين ( ()):

```
int i;
float  f = 3.14;
i = (int) f;
```

المثال السابق يحول العدد الحقيقي (3.14) الى عدد صحيح (3)، طبعا الباقي (الكسر) سيفقد. هنا معامل التحويل هو (int). طريقة أخرى لعمل نفس الشيء في C++ وذلك باستخدام النوع الذي سبق التعبير المراد تحويلة بالنوع الجديد وتحديد التعبير بأقواس

```
i = int (f);
```

كلا الطريقتين مقبول في C++

### 1.21 حجم البيانات (sizeof)

هذا العامل يقبل وسيط واحد والذي ممكن ان يكون نوعا او المتغير نفسة ويعيد قيمة تمثل حجم النوع او الكيان **بالبايت**:

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

في المثال اعلاه فان قيمة (a) ستكون (1) وذلك لان النوع (char) هو نوع بطول بايت واحد. القيمة المعادة بواسطة (sizeof) ثابتة، لذلك دائما تحسب قبل

تنفيذ البرنامج.

## 1.22 الأخطاء التي ترافق البرامج Errors

هناك أربع أنواع من الأخطاء التي تحدث في الحاسوب عند تنفيذ برنامج وهي:

**1. أخطاء المترجم Compiler errors**

تحدث هذه الأخطاء أثناء محاولة المترجم ترجمة البرنامج، وهي ناتجة عن خطأ قواعدي في كتابة البرنامج، مثل عدم وضع فارزة منقوطة في نهاية عبارة كاملة.

**2. أخطاء الربط Linker errors**

ان أغلب الأخطاء من هذا النوع تحدث عندما لايتمكن الرابط (Linker) من أيجاد الدوال أو عناصر البرنامج الأخرى والتي يشار اليها في البرنامج.

**3. أخطاء وقت التنفيذ Run-time errors**

في بعض الأحيان لايتم الكشف عن الخطأ الا أثناء تنفيذ البرنامج، مثال القسمة على صفر.

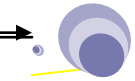
**4. أخطاء مرئية Conceptual errors**

هذه أخطاء يقع بها المبرمج نتيجة لخطأ في الطباعة أو السهو وهي صحيحة للمترجم ولكنها تعطي نتائج خاطئة.

## 1.23 موجهات التضمين وفضاء الاسماء Include Directives and Namespaces

جميع برامجك تبدأ بالسطرين التاليين

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



ولمناقشة وظيفة هذين السطرين سنبدأ بالسطر الاول والذي هو يتضمن جزئين، الجزء الاول هو (#include) وهذا يعني ان المطلوب هو تضمين برنامجك بالملف الموضح اسمة لاحقا، وهذه من الممكن ان تكون اكثر من ملف واحد (اي اكثر من #include) كل واحد منها له وظيفة اضافة ملف معين تحتاج له في تنفيذ برنامجك وهذه الملفات موجودة ضمن المكتبة القياسية للغة، اما الجزء الثاني من السطر الاول والذي سنطلق عليه تسمية الموجة او الملفات الرأسية (السطر الاول) فانه يحتوي على اسم الملف المطلوب اضافتة الى البرنامج ويكون محددا بين العلامتين (< >) كما سبق وان اوضحنا، الملف الموضح في هذا السطر هو باسم (iostream) وهذا الملف هو المسؤول عن توفير وتفعيل اوامر الادخال والاخراج ونظرا الى انك في كل برنامج تكتبة لابد من الاحتياج الى عملية ادخال او اخراج او كليهما لذلك فلا بد من ان تكون مكتبة iostream متوفرة، هذه المكتبة تتضمن تعريف cin، / cout (وهي اوامر الادخال والاخراج وسيتم شرحها في الفصل القادم)، فضلا عن امور اخرى. هناك ملفات اخرى كثيرة ربما تحتاج لها في تنفيذ برنامجك ولكل منها واجب محدد (لمزيد من المعلومات يمكنك الاطلاع على هذه الملفات في الملاحق).

السطر الثاني يتضمن التعبير **using namespace std;**

C++ تقسم الاسماء الى فضاءات اسماء، وفضاء الاسماء هو تجمع للاسماء، مثل الاسماء (cin، cout). العبارة التي تحدد فضاء الاسماء بالطريقة الموضحة ادناة تدعى الموجة using.

```
using namespace std;
```

هذا الموجة الخاص (using) يفيد ان برنامجك يستخدم او يفرض استخدام فضاء الاسماء القياسية (std)، هذا يعني بان الاسماء التي تستخدمها سيكون لها المعاني المحددة لها في فضاء الاسماء القياسية. في هذه الحالة، الشيء المهم هو عندما تكون الاسماء مثل cin، cout معرفة في iostream، تعريفها يفيد انتماءهم

الى فضاء الاسماء القياسية. لذا ولاجل استخدام الاسماء مثل cin، cout فانك تحتاج الى اخبار المترجم بانك تستخدم فضاء الاسماء القياسية.

هذا كل ماتحتاج الى معرفة الان حول فضاء الاسماء، ولكن توضيح مختصر سوف يحل اللغز الذي يحيط استخدام فضاء الاسماء. السبب ان C++ له فضاء اسماء بشكل مطلق وذلك بسبب وجود اشياء كثيرة يجب تسميتها. كنتيجة، احيانا يستلم عنصران او اكثر نفس الاسم، بمعنى اسم مفرد وممكن ان يحصل على تعريفين مختلفين. ولازالة هذا الغموض، C++ يقسم العناصر الى مجاميع، لذا لايوجد عنصران في نفس التجمع (نفس فضاء الاسماء) لهما نفس الاسم.

لاحظ ان فضاء الاسماء هو ليس تجميع بسيط للاسماء. هو جسم لشفرة C++ والتي تحدد المعنى لبعض الاسماء، مثل بعض التعريفات و/او الاعلانات. وظيفة فضاء الاسماء هو تقسيم جميع مواصفات اسماء C++ الى تجمعات (تدعى فضاء الاسماء) اذ ان كل اسم في فضاء الاسماء يملك فقط مواصفة واحدة (تعريف واحد) في فضاء الاسماء.

فضاء الاسماء يقسم الاسماء ولكن ياخذ الكثير من شفرة C++ مع الاسماء.

ماذا لو اردت ان تستخدم عنصرين في فضائي اسماء مختلفين، اذ ان كلا العنصرين له نفس الاسم؟ من الممكن ان تقوم بذلك وهي ليست معقدة، وهذا سنشير اليه لاحقا في هذا الكتاب.

***ملاحظة://***

بعض نسخ C++ تستخدم التالي، والذي هو نسخة قديمة او شكل قديم للموجة include ( دون استخدام فضاء الاسماء):

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

فاذا كانت برامجك لاتترجم او لاتنفذ مع العبارات التالية

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

عليك ان تحاول استخدام السطر التالي بدلا من السطرين السابقين

#include<iostream.h>

فاذا طلب برنامجك iostream.h بدلا من iostream، عليه فان ذلك يعني انك تملك نسخة قديمة من مترجم C++ وعليك ان تحصل على نسخة حديثة.

**جدول (1.13) بعض الدوال المهمة**

| التسلسل | الدالة | وظيفتها |
|---|---|---|
| 1 | abort( ) | دالة أيقاف البرنامج (تنهي تنفيذ البرنامج فورا) |
| 2 | abs() | دالة القيمة المطلقة الصحيحة |
| 3 | ceil() | دالة أيجاد أكبر عدد صحيح للقيمة (x) مثال Ceil (8.7) هـــو 9 |
| 4 | clrscr() | دالة تنظيف الشاشه |
| 5 | exit() | دالة الخروج من البرنامج |
| 6 | floor() | دالة أيجاد أصغر عدد صحيح للقيمة (x) (تستعمل لأيجاد أصغر عدد صحيح للقيمة الحسابية حسب التعريف الرياضي المعروف [ x ] (أذا كانت (x) سالبه يحذف كسرها وتنقص واحد) |
| 7 | log() | دالة اللوغاريتم الطبيعي (تحسب اللوغاريتم الطبيعي (lin (x)) ويجب أن تكون قيمة (x) أكبر من الصفر. |
| 8 | log10() | دالة اللوغاريتم العشري (تحسب اللوغاريتم للأساس 10 ($\log_{10}(x)$) ويجب أن تكون (x) أكبر من الصفر |
| 9 | pow() | تحسب هذه الداله قيمة المقدار ($x^y$) وكمايلي Z = pow (x ،y) ; |
| 10 | sqrt() | دالة أيجاد الجذر التربيعي لعدد موجب (x)، مثال Y = sqrt (x); |

## اسئلة للحل:

1. اي من التعابير التالية هو متغير مقبول:

```
int n = - 10 ;
int x = 2.9 ;
int 2k ;
float y = y * 2 ;
char c = 123 ;
char h = “c” + 23 ;
int  !b ;
float  c ;
```

2. اي من العبارات ادناه يمثل معرف مقبول:

```
Seven_11
_unique
Gross-income
Gross$income
2by2
Averag_ weight_of_a_large_pizaa
Object.oriented
Default
@yahoo
```

# الفصل الثاني
# أوامر الأدخال والأخراج
# INPUT / OUTPUT INSTRUCTIONS

## 2.1 المقدمة

جميع اللغات الطبيعية التي يتعامل بها الأنسان كوسيلة للتخاطب والتواصل لها قواعد وضوابط تحدد ألية استخدامها، ولما كانت لغات البرمجة تصنف على أنها من اللغات العليا (أي اللغات القريبة من لغات البشر) فكان لا بد وأن تكون لها قواعد تحدد ألية استخدامها لتكون واضحة للمتعامل معها وكذلك للمترجم داخل الحاسوب. علية فأن هذا الفصل والفصول اللاحقة ستوضح هذه القواعد وسنبدأ خلال هذا الفصل بمعرفة كيفية تلقيم الحاسوب بالمعلومات وطرق الحصول على النتائج بعد أنجاز عمليات الحساب.

## 2.2 هيكلية البرنامج Program Construction

يتكون برنامج لغة C++ من (الرأس والجسم) (head and block) والرأس هو السطر الأول في البرنامج ويبدأ بكلمة (#include) ويتبع باسم الملف الرأسي (header file) والذي يكون محدد بين علامتي الاكبر والاصغر (< >) وكما يأتي:

```
#include<iostream>
```

اما جسم البرنامج فيبدأ بالدالة (main()) ثم يتبع بالايعازات والأوامر التي تمثل الخطوات الواجب أتباعها أو تنفيذها من قبل الحاسوب للحصول على النتائج المطلوبة من البرنامج، وتكون هذه الايعازات محددة بأشارة البداية والنهاية حيث تستخدم الأقواس المتوسطة لهذا الغرض ( { } ).

```
#include<iostream>
main ( )
        {
```

Set of instructions;

}

## 2.3 المخرجات والمدخلات Input / Output

كل برنامج يجب أن تكون له مخرجات تبين النتائج التي تم الحصول عليها من البرنامج، هذه النتائج سيتم عرضها على شاشة الحاسوب باستخدام عبارة الأخراج (<< cout ) أن الأمر (<< cout)من الممكن ان يترجم على انه أكتب ماموجود بعد العلامة (<<) على السطر الذي يؤشر عليه المسيطر (controller) في شاشة التنفيذ.

عبارة الأخراج لها أثنان من صفات C++ الجديدة وهي (cout) و (<<)، حيث أن المعرف (cout) يلفظ (C out) وهو كيان معرف مسبقا يمثل تدفق المخرجات القياسية في++C، هنا تدفق المخرجات القياسية يمثل طباعتها على الشاشة، ومن الممكن أعادة توجية المخرجات الى أجهزة أخرى.

أما العامل (<<) ويدعى (insertion OR put to operator) (عامل الحشر أو الوضع) وواجبة حشر أو أرسال محتويات المتغير الذي على جانبها الأيمن الى الكيان الذي موجود على جانبها الأيسر.

***ملاحظة://***

(bit_wise) يستخدم أيضا العامل ( <<) كعامل تزحيف الى اليسار( يعمل) على مستوى البتات, كما سبق وان اشرنا في الفصل الاول.

أن ما يوضع بعد العلامة (<<) سيأخذ حالة من أثنتين:

### 2.3.1 الحالة الأولى

ان يكون ما بعدها محدد بعلامات اقتباس مزدوجة ( double quotation mark) (" ") وبهذه الحالة فان ما موجود بين علامتي الأقتباس سيتم طباعتة على الشاشة كما هو دون أدنى تغيير.

برنامج لطباعة عبارة معينة على الشاشة

```
// Example 2.1
#include <iostream>
using namespace std;
```

```
main()
{
  cout <<" Hello World. Prepare to learn C++ !!" ;
}
```

**لاحظ مايلي://**

**اولا**/ان مخرجات هذا البرنامج هي العبارة التي تلي العامل (<<)، وستظهر على الشاشة كما يلي:

***مخرجات البرنامج 2.1:***

```
Hello World. Prepare to learn C++ !!
```

**ثانيا**/ عند تنفيذ هذا البرنامج سوف لا يمكن ملاحظة المخرجات والسبب هو أن الحاسوب سريع جدا بحيث يعرض ويخفي شاشة التنفيذ دون أن تلاحظ ذلك، ولغرض رؤية المخرجات فيمكن بعد ان يتم التنفيذ ضغط الزرين (Alt+ F5) معا وعندها ستظهر شاشة التنفيذ (السوداء).. ويمكن الخروج من شاشة التنفيذ بضغط الزر (Enter )

***ملاحظة://***

لغرض ايقاف شاشة التنفيذ بعد انتهاء التنفيذ لرؤية النتائج، استخدم الامر التالي في نهاية البرنامج:

```
system ( "pause") ;
```

مع ملاحظة ان هذا الامر يعمل مع الموجهة #include<stdlib>

وعند استخدامة سوف لاتختفي شاشة التنفيذ بعد انتهاء التنفيذ مع وجود ملاحظة تخبر المستخدم بالضغط على اي زر لغرض الأستمرار.

### 2.3.2 الحالة الثانية

أما أذا كان ما موجود بعد العلامة (<<) ليس محدد بين علامتي أقتباس فعند ذلك سيعامل ما موجود بعدها على أنه معرف والمعرفات هنا تكون على واحدة من الحالات ادناه:

* أما أن تكون مقادير ثابتة (قيم حسابية) مثل القيم (4567، 123-، 78.456...الخ) فهي تطبع مباشرة على الشاشة دون تغيير، مثلا

```
cout << 3456 ;
```

هنا سيتم طباعة (3456) على الشاشة.

* أو تكون على شكل تعبير حسابي (expression) (اي مقادير تفصل بينها العوامل الرياضية او المنطقية مثل (+، - ، *.. الخ) وبهذه الحالة فسيتم استخراج قيمة العملية الحسابية او المنطقية وطباعتها على الشاشة، مثال

```
cout << 34 + 56 ;
```

في هذه الحالة سيتم طباعة ( 90) على الشاشة.

* أو أن تكون على شكل رموز، وتعد انذاك متغيرات (والمتغيرات لها اسماء) تؤشر الى قيم في الذاكرة (**يجب أن تكون لها قيمة**) (كما سبق ان وضحنا بالفصل الاول فان المتغيرات تشير الى مواقع في الذاكرة وهذه المواقع تحتوي على قيم)، لذا فان الحاسوب سيطبع قيمة المعرف (المتغير) على شاشة التنفيذ (أي تطبع القيمة الموجودة او المخزونة في موقع الذاكرة الذي يشير له المتغير).

هنا عليك أن تلاحظ أن استخدام أي معرف (متغير) داخل البرنامج يحتاج الى شرطين:

**الأول/** أن يتم الأعلان عن المعرف قبل أن يتم استخدامة لأول مرة في البرنامج ويحدد نوعة وفقا للأنواع التي سبق أن نوهنا عنها في الفصل الاول، فأذا كانت قيمة المتغير غير ثابتة ويمكن ان تتغير قيمتة (تتغير قيمتة أثناء تنفيذ البرنامج) فيعلن عنه ويحدد نوعة (ويتم ذلك بكتابة اسم المتغير مسبوقا بنوعة)، فمثلا أذا كان المطلوب استخدام المتغير (x) وهو من نوع الأعداد الصحيحة، فيكون بكتابة النوع أولا ثم يتبع ذلك كتابة أسم المتغير( على أن يكون هناك فراغ بين النوع واسم المتغير) وتنتهي العبارة دائما بفارزة منقوطة، وكما يأتي:

```
int  x ;
```

هذا المتغير هو من نوع الأعداد الصحيحة (integer) أي أن القيمة التي يحملها دائما ستكون عدد صحيح. ويجب ان تلاحظ ان الاعلان عن المعرف يكون لمرة واحدة في البرنامج.

**ثانيا/** يجب أن تكون لهذا المتغير أو الثابت قيمة عند أول استخدام له داخل البرنامج فمثلا أنك عرفت المتغير (x) من نوع الاعداد الصحيحة لكن كم هي قيمة هذا المتغير؟ هو عدد صحيح لكن كم !! فعندما تعطي الأمر (cout << x ;) فكم يجب على المترجم أن يطبع على شاشة التنفيذ ! لذا يجب أن تحدد قيمة المتغير أو الثابت قبل او اثناء أول استخدام.

هذه القيمة التي تحدد وتسند للمتغير تأتي من احدى عمليتين فأما أن تسند القيمة للمتغير اثناء كتابة البرنامج أو تسند القيمة للمتغيراثناء تنفيذ البرنامج... لنناقش الحالتين:

***ملاحظة://***

سبق وان ذكرنا ان بالامكان أسناد الاعداد الصحيحة للمتغيرات من نوع الاعداد الصحيحة، والقيم الحقيقية للمتغيرات من نوع الاعداد الحقيقية، والحروف للمتغيرات من نوع الحروف وهكذا.. ولكن الحقيقة ان هذا القول ليس دقيقا وذلك لان لغة C++ تحول بين الانواع أليا في بعض الحالات، مثال:

```
int number;
```

number = ‘a’;

cout << number <<endl ;

النـــاتج هنـــا ســـيكون (97) وهـــو الـــرقم الـــذي يســـتخدم داخليـــا فـــي لغـــة C++) ASCII) لتمثيل الحرف ( a)، ولكن من المناسب استخدام الاعداد الصحيحة لمتغير الاعداد الصحيحة والحروف لمتغير الحروف ولا تحول بينهما الا اذا كان هناك سبب معقول.

برنامج لتوضيح الحالات اعلاه، يستخدم المتغيرات واوامر الطباعة لطباعة عبارة معينة وعدد يمثل العمر، مع ملاحظة زيادة هذه الارقام وانقاصها.

```
// Example 2.2
#include <iostream>
using namespace std;
 int main()
 {
 int myAge = 39; // initialize two integers
 int yourAge = 39;
 cout << "I am: " << myAge << " years old.\n";
 cout << "You are: " << yourAge << " years old\n";
 myAge++;   // postfix increment
 ++yourAge; // prefix increment
 cout << "One year passes...\n";
 cout << "I am: " << myAge << " years old.\n";
 cout << "You are: " << yourAge << " years old\n";
 cout << "Another year passes\n";
 cout << "I am: " << myAge++ << " years old.\n";
 cout << "You are: " << ++yourAge << " years old\n";
```

```
cout << "Let's print it again.\n";
cout << "I am: " << myAge << " years old.\n";
cout << "You are: " << yourAge << " years old\n";
return 0;
}
```

**مخرجات البرنامج 2.2:**

```
I am 39 years old
You are 39 years old
One year passes
I am 40 years old
You are 40 years old
Another year passes
I am 40 years old
You are 41 years old
Let's print it again
I am 41 years old
You are 41 years old
```

**ملاحظة://**

لغرض اخراج رسالة خطأ فبالامكان استخدام الايعاز (cerr<< ) بدلا من ايعاز الاخراج الاعتيادي (cout<<)، وطبعا عليك ان تكتب ماهي الرسالة التي ترغب ان تظهر عند وجود خطأ، مع ملاحظة ان مايكتب بعد (cerr<<) سيكون محدد بحاصرة مزدوجة. مثال

```
cerr<< " Error, can't divide by zero " ;
```

* اسناد القيم أثناء كتابة البرنامج:

ويتم ذلك من خلال استخدام التعابير (expression)، ويستخدم التعبير مع معادلة (والمعادلة عبارة عن طرفين يفصل بينهما علامة التخصيص (assignment) الطرف الأيمن هو عبارة عن تعبير او قيمة ثابتة بينما الطرف الأيسر يكون متغيرا ومتغير واحد فقط، لذا فان المساواة تستخدم لاسناد قيمة للمتغير)، فمثلا تقول:

```
x  = 5 ;
```

هنا استخدمنا المساواة (=) وبذلك فان قيمة المتغير (x) ستكون مساوية الى العدد الصحيح (5)، أو ممكن أن تكون المعادلة على شكل:

```
x  = 3 * 2 +5 ;
```

هنا قيمة (x) تساوي (11)، وكذلك ممكن أن تحدد قيمة للمتغير بالمساواة ولكن في حقل الأعلان عن الثوابت.

***ملاحظة://***

دائما عند وجود علامة المساواة (=) فان الضوابط التالية ستطبق:

يجب أن يكون هناك طرفين تفصل بينهما علامة المساواة، وبذلك ممكن أن نطلق عليها تسمية المعادلة.

الطرف الأيسر من المعادلة أي الذي يقع على الجانب الأيسر من المساواة يكون متغيرا ومتغير واحد فقط دائما، ولا يجوز أن يكون قيمة ثابتة (مثلا 6، 456، 34.2..الخ)، ولا يجوز أن يكون رمز معرف ومعلن عنه على أنه ثابت، كذلك لا يجوز أن يحتوي على علاقات رياضية مثل ( x + 6).

أما الطرف الأيمن فيمكن أن يكون قيمة رقمية أو عددية واحدة أو علاقة رياضية (تعبير) تحتوي على ( قيم عددية تفصل بينها العلامات الرياضية، أو علاقة رياضية تحتوي متغير واحد، متغيرات، أو متغيرات وقيم عددية). مثلا العلاقات التالية مقبولة

X = 89 ;

X = 34 – 45 +3;

X = y ;

X = 3 * y +90 ;

من الممكن أن يكون في التعبير الواحد أكثر من مساواة واحدة ( سنأتي عليها في موضعها).

عند تنفيذ البرنامج فأن المترجم سيبدأ بالطرف الأيمن من المعادلة دائما ويتم فحص هذا الطرف فأذا كانت فيه متغيرات فسيبحث المترجم في الخطوات السابقة للخطوة التي هو فيها ضمن البرنامج للتأكد من أن المتغير معلن عنه ( له نوع) أولا، ثم يجب أن تكون له قيمة قبل هذه الخطوة، وتجلب هذه القيمة لتعوض عن المتغير في المعادلة ( ممكن أن تتخيل الطرف الأيمن عندها سيصبح عبارة عن مجموعة من القيم الثابتة بعد ان يتم تعويض قيم المتغيرات داخليا في الحاسوب)، بعدها تجرى العمليات الحسابية وتكون من اليسار ألى اليمين وحسب أسبقيات العمليات الرياضية، فالأسبقية الأعلى تنفذ أولا وأذا تساوت عمليتان بالاسبقية فتنفذ العملية التى في اليسار اولا، من ذلك سينتج لنا قيمة واحدة ثابتة، هذه القيمة ستؤول الى المتغير الذي في الطرف الأيسر (دائما القيمة تنتقل من الطرف الأيمن للمعادلة (التعبير) الى المتغير الذي في الطرف الأيسر اي تخزن في الذاكرة في الموقع الذي يشير له المتغير الذي بالطرف الأيسر).

يجب أن يكون المتغير الذي على يسار المساواة والمتغير أو المتغيرات على يمين المساواة من نفس النوع وأذا ما اختلفت الأنواع فهناك عمليات من الممكن أن تجرى أليا لتحويل الأنواع سنأتي عليها لاحقا.

* أسناد القيم أثناء تنفيذ البرنامج:

وتتم عملية اسناد (ادخال) قيمة للمتغير أثناء تنفيذ البرنامج وذلك باستخدام أمر القراءة (cin >>) وهي تعني (أقرأ القيمة المطبوعة على شاشة التنفيذ واخزنها في موقع الذاكرة الذي يشار اليه بواسطة المتغير الموجود بعد العلامة (>>).

* برنامج لادخال قيمتين لمتغيرين اثناء تنفيذ البرنامج وايجاد مجموعهما.

```
// Example 2.3
#include <iostream>
using namespace std;

main()   // no semicolon
{
  int num1 ،num2 ،sum ;
  cout<< "input number 1 :";
  cin>> num1;
  cout<< "input number 2 :";
  cin>> num2;
  sum = num1 + num2; //addition
  cout<<sum ;
  return 0;
  }
```

**مخرجات البرنامج 2.3 :**

```
input number 1:    20       // Press enter
```

***input number 2:  15  // Press enter***
***35***

***ملاحظة://***
في كل تطبيق يجب أن يتأكد المبرمج من أن الكيان أو المتغير الموجود في البرنامج له قيمة قبل أن يتم استخدامة لأول مرة في البرنامج، في خلاف ذلك فأن المترجم سيستخدم متغيرا ليس له قيمة محددة من المبرمج او المستخدم، لذلك فان المترجم سيستخدم القيمة الموجودة في موقع الذاكرة الذي يشير عليه المتغير ودائما تكون قيم من برامج سابقة ليس لها علاقة ببرنامجك وبالتالي فستحصل على نتائج خاطئة او ربما تكون قيمتة صفرا اذا لم يتم استخدامة سابقا ( اي خالي من القيم ).

شرح البرنامج 2.3://

**أولا**:// تم استخدام المتغيرات (num1، num2، sum) وهي جميعا من نوع الأعداد الصحيحة لأن هذا البرنامج صمم للتعامل مع الأعداد الصحيحة (يقوم بجمع عددين صحيحين وأظهار النتيجة).

**ثانيا**:// يمكن الأعلان عن كل متغير بسطر منفصل، ويمكن وضعها جميعا بسطر واحد كما في هذا البرنامج على شرط أن تكون جميع المتغيرات من نفس النوع (هنا جميعها أعداد صحيحة) وذلك لغرض تقليل المساحة التي يكتب عليها البرنامج، على ان يتم الفصل بين متغير وأخر بفارزة. وطبعا العبارة تنتهي بفارزة منقوطة.

**ثالثا**:// بعد الدالة (main()) لاحظ العبارة التالية ({ no semicolon }) وهي تعني لا تستخدم فارزة منقوطة، وبما أنها وضعت بعد العلامة (//) فأن ذلك يعني أنها ملاحظة أو تعليق (Comment) للمستخدم أو القاريء بعدم استخدام الفارزة المنقوطة بعد كلمة ((main)) هذه العبارة التي أعتبرت تعليقا كتبت ووضعت بعد

العلامة (//)، وسوف لا يكون لها تأثير على تنفيذ البرنامج (أي أنها تهمل أثناء تنفيذ البرنامج)، عليه فسيكون عندك قاعدة وهي " أن أي عبارة تستخدم لغرض التوضيح أو التعليق ممكن كتابتها داخل البرنامج وحسب القواعد التي تم التطرق لها في الفصل الأول، وسوف لا تكون جزء من البرنامج أثناء التنفيذ (تهمل)".

***ملاحظة://***

التعليقات أو الملاحظات تستخدم لأيضاح عمل بعض الدوال والأجراءات التي تكون معروفة لدى المبرمج وغير معروفة للمستخدمين، أيضا تستخدم لكتابة بعض المعلومات حول البرنامج ( كوقت انشائة أو تحديثة ) أو معلومات حول المبرمج نفسه ( مثلا الأسم ، العنوان الالكتروني).

التعليقات ممكن أن توضع في أي مكان في برنامج C++، ولكن يفضل أن تكتب في بداية البرنامج ( في حالة كون المعلومات عن وظيفة البرنامج أو معلومات عن المبرمج )، أو تكتب بجانب الأوامر التي تحتاج الى توضيح .

**رابعا**:// كما سبق وأن ذكرنا أن تنفيذ البرنامج يتم بالتسلسل من الأعلى الى الأسفل فيبدأ من الموجهة (#include) ثم العبارة ( () main)، وبعدها أمر بداية البرنامج ({) (والتي تعني أن ما بعدها هي أوامر برمجة مطلوب من الحاسوب تنفيذها، يلي ذلك قراءة المتغيرات، بعدها ينفذ أمر الطباعة (لاحظ الموجود بعد العلامة (<<) في أمر الطباعة هو محصور بين علامتي أقتباس لذا فأنه يطبع كما هو) هذه العبارة ستظهر على شاشة التنفيذ وهي تخبر المستخدم مايلي (أدخل الرقم الأول: input number1) وهي بشكل عام يمكن الأستغناء عنها دون أن يتأثر البرنامج.. ولكنها مفيدة حيث تخبر المستخدم عن الخطوة أو الخطوات الواجب أتباعها لأنجاز تنفيذ البرنامج، (يمكن ملاحظة مثل ذلك في البرامج التي تعملون عليها مثلا في برنامج للعبة (game) معينة فأن هناك ملاحظات ستظهر على الشاشة لأرشاد

المستخدم عن الخطوات الواجب أتباعها لتشغيل اللعبة أو أختيار درجة الصعوبة وغيرها).

**خامسا**:// هنا تبدأ عملية أسناد قيمة للمتغير (num1) وذلك باستخدام الأمر (cin>>)، عند الوصول الى هذه الخطوة فأن شاشة التنفيذ (الشاشة السوداء) ستظهر ويكون هناك مؤشر صغير على شكل شارحة ( _) يظهر ويختفي (ينبض) في موقع على الجانب الأيسر من شاشة التنفيذ، هذا المؤشر يحفز المستخدم على طباعة قيمة على الشاشة (طباعة قيمة معينة باستخدام لوحة المفاتيح)، وبعد أن تطبع هذه القيمة يتم أعلام (المعالج) بأنجاز العمل وذلك من خلال الضغط على الزر (Enter)، في هذه الحالة سيتم قراءة القيمة التي طبعت على الشاشة وخزنها في الموقع الذي يؤشر عليه المتغير الموجود بعد الأمر (cin>>) وبذلك تكون قد أسندت قيمة للمتغير (num1) (خزن قيمة) في الموقع الذي يؤشر عليه المتغير في الذاكرة بعد هذه الخطوة، وهذا ما أسمية الأسناد الذي يتم بواسطة المستخدم أثناء تنفيذ البرنامج.

**سادسا**:// الأمران اللاحقان هما مشابهان للخطوتين الرابعة والخامسة.

**سابعا**:// التعبير (sum = num1 + num2)، عند الوصول الى هذا التعبير فأن المترجم سيبدأ بالطرف الأيمن من التعبير ويعوض عن المتغيرات الموجودة بما يساويها من قيم (هذه القيم تم اسنادها الى المتغيرات من خلال الامر cin>> والذي اشرنا له)، بعدها يتم أجراء عملية الجمع على هذه القيم لينتج عن ذلك قيمة واحدة في الطرف الأيمن، هذه القيمة ستوضع (تخزن) في الموقع الذي يؤشر عليه المتغير الموجود في الطرف الأيسر، وبذلك فان المتغير (sum) ستسند له قيمة (تخزن في الموقع الذي يؤشر عليه في الذاكرة) من خلال المعادلة، وهذا ما أسميه أسناد قيمة اثناء كتابة البرنامج ( أي أن المستخدم لا يتدخل في ذلك أثناء تنفيذ البرنامج).

**ثامنا**:// بعد أنجاز العمل المطلوب من البرنامج فلا بد من أعلام المستخدم بالنتيجة المتحصلة من أنجاز أو تنفيذ هذا البرنامج، ويتم ذلك من خلال طباعة القيمة

المتحصلة والتي هي الأن موجودة في المتغير(sum)، لذا تم استخدام أمر الطباعة ليطبع ما موجود بعد العلامة (>>) ولما كان ما موجود بعد هذا العامل غير محدد بعلامتي أقتباس لذا فان القيمة المخزونة في الذاكرة في الموقع الذي يشير عليه المتغير(sum) هي التي تظهر على شاشة التنفيذ (اي ان المترجم يعوض اولا قيمة المتغير sum في امر الاخراج وبعدها تتم طباعة القيمة).

**تاسعا**:// الأمر الأخير هو (}) الذي يمثل نهاية البرنامج.

***ملاحظة:***//

بشكل عام فان استخدام القوس المتوسط المفتوح ( { ) والذي يشير الى البداية يجب أن يقابلة قوس متوسط مغلق يشير الى النهاية ( } ) ، عليه فأن عدد الأقواس المتوسطة المفتوحة في البرنامج الواحد تساوي عدد الأقواس المتوسطة المغلقة في ذات البرنامج، أما الأستثناءات فسنشير لها في موضعها .

***ملاحظة:***//

في أدناه بعض القواعد التي يجب أن تلاحظ عند أدخال البيانات المطلوبة :

يجب أن يتطابق نوع القيمة المدخلة لمتغير معين مع النوع المعلن لهذا المتغير .

أذا كانت هناك رغبة في أسناد قيم لأكثر من متغير في أيعاز قراءة واحدة فيجب أن يفصل بين متغير وأخر بواسطة العامل ( >> ).

يجب أن يتطابق عدد البيانات التي يتم أدخالها مع عدد المتغيرات المدونة بعد العامل ( >> ) في أيعاز القراءة.

اذا كان أكثر من متغير واحد في ايعاز قراءة واحد فيمكن أدخالها جميعا ثم ضغط الزر ( Enter ) على أن يفصل بين قيمة وأخرى فراغ، أو تدخل القيم واحدة بعد الأخرى على أن تضغط الزر ( Enter ) بعد أدخال كل قيمة .

لا يجوز أن تكون القيم المدخلة صيغ رياضية ( أي قيم بينها علامات رياضية)

***ملاحظة://***

من الممكن استخدام العوامل ( >> )، ( <<) بشكل متكرر مع عبارات الادخال والاخراج ( cout OR cin ) لتفيد تكرار أمر الادخال والاخراج. مثال

```
cout << x << y << z ;
cin >> x >> y >> z ;
```

## 2.4 بعض الصيغ المهمة في عمليات الأدخال والأخراج

**Formatted Consol for I/O Operations**

دعم C++ عدد من الصفات التي من الممكن ان تستخدم لصياغة او تنظيم طريقة ظهور المخرجات والموضحة بالجدول (2.1)، هذه الدوال تستخدم مع الموجة (iostream) او مايكافئها مع (iomanip) وهي تستخدم بالترافق مع الأمر (cout)، والصيغة العامة لها هي:

cout.function

لاحظ هنا تم استخدام *النقطة* ((.) بدلا من <<).

**جدول (2.1): بعض الصفات المهمة التي تستخدم لصياغة او تنظيم المخرجات**

| دوال مع الموجة #include<iostream> | دوال مع الموجة #include<iomanip> | وضيفة الدالة |
|---|---|---|
| width ( ) | setw ( ) | تحدد حجم الحقل المطلوب لعرض قيم المخرجات |
| precision ( ) | setprecision ( ) | تحدد عدد المراتب بعد الفارزة عند عرض القيم الحقيقية |
| fill ( ) | setfill ( ) | تحدد نوع الرمز الذي سيستخدم لملأ الجزء غير المستخدم في الحقل المحدد لعرض قيمة معينة |

| | | |
|---|---|---|
| setf ( ) | setiosflags ( ) | تحدد اشارة للمسيطر لتحدد نوع الصياغة المطلوبة (مثل طباعة القيمة من اليمين او اليسار,ملأ السطور) |
| unsetf ( ) | setiosflags ( ) | تستخدم لألغاء الصياغة المحددة بالأيعاز السابق |

**مثال:**

```
cout.width (5) ;
cout << 345 ;
```

المخرجات ستكون كما يأتي:

| | | | | |
|---|---|---|---|---|
| | | 3 | 4 | 5 |

اي ان المترجم سيحدد خمس مواقع لطباعة القيمة، ويبدأ الطباعة من اليمين، لذلك سيكون هناك فراغين في اليسار.

***ملاحظة://***

تأثير الدالة width () يستمر لأمر طباعة واحد فقط، فاذا كان هناك اكثر من امر طباعة فنستخدم (() width ) مع كل امر طباعة..

***ملاحظة://***

يستخدم الأمر (()fill ) لملا الفراغات، ويجب ان تضع بين قوسي الأمر fill() الرمز المطلوب طباعتة ( بما انه رمز فيجب ان يحدد بحاصرات مفردة ). اما أذا لم يحدد ماهية الرمز المطلوب طباعتة في الحقول الفارغة ( عند تحديد حجم الحقل لطباعة قيمة معينة ) فأن المترجم سيتركها فارغة كما في المثال السابق . مثال

```
cout.fill ( ' * ' ) ;
```

```
cout.width ( 7 ) ;
cout << 345 ;
```

في هذه الحالة فان الحقول الفارغة ستملأ بالعلامة ( * ) وستكون النتيجة:

| | | | | | | |
|---|---|---|---|---|---|---|
| * | * | * | * | 3 | 4 | 5 |

***ملاحظة://***

في حالة تحديد عدد المراتب بعد الفارزة فأن تأثير الدالة سيستمر على كل القيم اللاحقة لحين الغاء أو أعادة التحديد . مثال

```
cout.precision ( 10 ) ;
```

هذا يعني ان كل الأرقام الحقيقية اللاحقة سيحدد لها عشر مراتب بعد الفارزة .

***ملاحظة://***

أذا لم يحدد عدد المراتب التي بعد الفارزة للأرقام الحقيقية فأن المترجم سيفرضها ست مراتب .

***ملاحظة://***

من الملاحظ في جميع الأمثلة أعلاة أن الطباعة تبدأ من اليمين الى اليسار وهي الحالة الأفتراضية ( default ) للحاسوب، أما أذا كان المطلوب غير ذلك فهناك دالة خاصة لهذا الغرض سنأتي عليها (( setf ( )، والتي لها استخدامات مختلفة .

* الدالة (setf()) تعمل مع الأمر (cout) كما بينا ولكنها تختلف بعض الشئ عن الدوال الأخرى المشار اليها أعلاه حيث أنها من الممكن أن تأخذ معامل واحد أو معاملين (وسيط او اثنين)، ووفقا لهذه المعاملات سيحدد واجبها وكمايأتي:

**1. الدالة مع وسيطين وتكون الصيغة العامة لها كما يأتي:**

cout.setf (arg1 ،arg2) ;

ويكون استخدام هذه الدالة وفقا لما موضح في الجدول (2.2).

**جدول (2.2): يبين وظيفة الدالة (setf()) مع استخدام اثنين من الوسائط**

| وظيفة الـــــــدالة | قيمة الوسيط الأول(flagarg1) | قيمة الوسيط الثاني bit-field (arg2) |
|---|---|---|
| ملأ السطور من اليسار | ios::left | ios::adjustifield |
| ملأ السطور من اليمين | ios::right | ios::adjustifield |
| اظهار العلامات الرياضية (الاشارة الموجبة والسالبة) | ios::internal | ios::adjustifield |
| العلامة العلمية | ios::scientific | ios::floatfield |
| علامة النقطة الثابتة | ios::fixed | ios::floatfield |
| الأساس العشري | ios::dec | ios::basefield |
| الأساس الثماني | ios::oct | ios::basefield |
| الأساس السادس عشر | ios::hex | ios::basefield |

**مثال://**

```
cout.fill ('@') ;
cout.precision (3) ;
cout.setf( ios::internal ،ios::adjustifield) ;
cout.setf (ios:: scientific ،ios::floatfield);
cout.width (15) ;
cout << -12.34567 <<"\n" ;
```

نتيجة هذا المثال هي:

| - | @ | @ | @ | @ | @ | 1 | . | 2 | 3 | 5 | e | + | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

تلاحظ ان الأيعاز الأول هو لملأ الفراغات بالرمز(@)، اما الايعاز في السطر الثاني فهو يمثل عدد المراتب بعد الفارزة للرقم الحقيقي وهي هنا (3)، الايعاز الثالث فهو يستخدم معاملين او وسيطين لاظهار العلامة الرياضية، الايعاز في السطر الرابع يستخدم لأظهار العلامة العلمية، ثم تم تحديد عدد المواقع التي ستطبع بها القيمة والتي حددت (15 موقع).. واخيرا تم ادخال القيمة المطلوب طباعتها (لاحظ النتيجة).

**2. استخدام وسيط واحد مع الدالة (() setf) والصيغة العامة لها هي:**

```
cout.setf (arg) ;
```

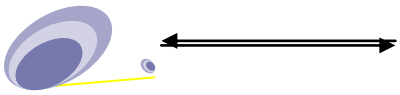
واعتمادا على قيمة الوسيط تقوم الدالة بعملها.

الجدول (2.3) يبين وظيفة الدالة (()setf) عند استخدامها وسيط واحد ووفقا لقيمة الوسيط المقابل لها

**جدول (2.3): وظيفة الدالة (()setf) عند استخدام وسيط واحد**

| وظيفة الـــــــدالة | قيمة المعامل (flag) |
|---|---|
| تستخدم base indicator في المخرجات | ios::showbase |
| تطبع العلامة الموجبة (+) قبل الأرقام الموجبة | ios::showpos |
| تظهر الفارزة والأصفار | ios::showpoint |
| تستخدم الحروف الكبيرة في المخرجات الممثلة بالنظام السادس عشري | ios::uppercase |
| حذف الفراغات (white space) في المخرجات | ios::skipus |
| تدفق كل (stream) بعد الحشر | ios::unitbuf |
| تدفق (stdout and stderr) بعد الحشر | ios::stdio |

* برنامج لايجاد الجذر التربيعي للرقم 5 مع تنظيم المخرجات، وكذلك الجذر التربيعي للرقم 100 باستخدام العلامة العلمية.



```
// Example 2.4
#include <iostream>
using namespace std;
#include <math>
main()
{  cout.fill('*') ;
   cout.setf(ios::left  ‹ios::adjustifield );
   cout.width(10);   cout << "value";
   cout.setf(ios::right  ‹ios::adjustfield);
   cout.width(15);
   cout<<"sqrt of value"<<"\n";    cout.fill('.');
   cout.precision(4);
   cout.setf(ios::showpoint);
   cout.setf(ios::showpos);
   cout.setf(ios::fixed  ‹ios::floatfield);
     cout.setf(ios::internal  ‹ios::adjustfield);
     cout.width(5);
     cout<<5;
     cout.setf(ios::right  ‹ios::adjustfield);
     cout.width(20);
     cout<<sqrt(5)<<"\n";     }
     cout.setf(ios::scientific  ‹ios::floatfield);
     cout<<"\nsqrt(100)="<<sqrt(100)<<"\n";
     return 0;
}
```

***مخرجات البرنامج 2.4://***

***value * * * * * * sqrt of value***

***+ . . . . 5 . . . . . . . . . . . . . +2.2361***

***sqrt ( 100 ) = +10000e+01***

سيتم شرح ايعاز التكرار الوارد في المثال ( 2.4) في الفصل الرابع.

***ملاحظة://***

تستخدم setw() مع الأعداد والسلاسل الرمزية.

***ملاحظة://***

يستخدم الأيعاز ( cin.get(ch) ) لأسناد حرف للمتغير الحرفي ( ch ) أثناء تنفيذ البرنامج حتى وأن كان فراغ أو سطر جديد، مثال

```
cin >> m ;
cin .get (ch) ;
cin >> n ;
```

الأن لتلاحظ ماهي المخرجات لحالات الأدخال المختلفة في أدناة :

Input 1 : 25 w 34 // m is 25 ، ch is w ، n is 34

Input 2 : 33 41 // m is 33 ، ch is blank ، n is 41

Input 3 : 67 ( Enter ) 55 // m is 67 ،ch is newline (\n) ،n is 55

***ملاحظة://***

الأرقام تمثل داخل الذاكرة بالصيغة الثنائية ( binary ) وهي تحدد عدد البتات اللازمة لتمثيل ذلك الرقم، لذلك يجب ملاحظة تعريف المتغير بما يتناسب وحجمة، وفي خلاف ذلك فأن النتائج ستكون خاطئة.

* لغرض أخراج القيم العددية الصحيحة وفقا لأساس يتم اختياره مثل (decimal، octal، hexadecimal) فان بامكانك كتابة المختصرات التالية مع أمر الأخراج لتحصل على قيمة عددية وفقا لذلك الاساس:

dec = decimal

oct = octal

hex = hexadecimal

* برنامج لادخال قيمة عددية وطباعتها بالنظام العشري، السادس عشر، والنظام الثماني.

```
// Example 2.5
# include<iostream>
using namespace std;

main( ) {
int value ;
cout<<" Enter number " << endl;
cin>>value ;
cout<<"Decimal base =" << dec<<value<<endl;
cout << " Hexadecimal base  =" << hex<<value <<endl ;
cout<<" Octal base=" << oct<<value << endl ;
return 0;
}
```

**مخرجات البرنامج 2.5 :**

```
Enter number
```

```
10
Decimal base =10
Hexadecimal base = a
Octal base = 12
```

لنفس الغرض اعلاه بالأمكان استخدام الأيعاز (() setbase) والذي يستخدم لأخراج القيم العددية الصحيحة وفقا للأساس المحدد بين القوسين لهذا الأيعاز (بكلام أخر بالأمكان تحويل الاعداد من اساس الى أخر والمقصود بالاساس هنا هو ان الاعداد العشرية (decimal) اساسها (10)، والثماني (octal) اساسها (8)، والسادس عشر (hexadecimal) اساسها (16)). وهذه الدالة تستخدم مع الموجة (#include<iomanip >)

* سنعيد كتابة المثال (2.5) ولكن باستخدام الأيعاز (setbase())

```
// Example 2.6
# include<iostream>
# include<iomanip>
using namespace std;
main( ) {
int value;
cout<<" Enter number " << endl ;
cin>>value ;
cout<<" Decimal base = " << setbase ( 10 ) ;
cout <<  value << endl ;
cout << " Hexadecimal base  =" << setbase ( 16 ) ;
cout << value <<endl ;
cout<<" Octal base=" << setbase ( 8 )  ;
cout<< value << endl ;
```

```
return 0;
}
```

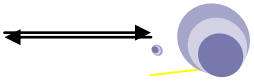
## 2.5 التعامل مع البتات Bit Manipulations

تعلمنا من المواضيع السابقة عندما نعلن عن متغير فان المترجم يحجز مساحة في الذاكرة لهذا المتغير وحسب نوعة. في الحقيقة، وكما تعلمنا من دراسة البايتات والكلمات، المتغير المعلن عنه يشغل مساحة في الذاكرة عبارة عن مجموعة من الصناديق الصغيرة. فحسب فهمنا الانساني، ليس من السهل دائما ان نفهم كيف يتم خزن حرف مثل الحرف B بثمانية صناديق صغيرة عندما نعرف ان الحرف B هو حرف واحد. ان التعامل مع البتات تسمح لك للسيطرة على كيفية خزن القيم بالبتات. هذه ليست عملية تحتاج الى انجازها كل مرة، خصوصا ليس في المراحل المبكرة من رحلتك مع C++. على الرغم من ذلك، عمليات البتات (والعمليات المتطابقة ذات العلاقة) تقدم في كل بيئات البرنامج التطبيقي، لذا فانك يجب ان تهتم بماذا تعمل وماذا تقدم. في ذلك الوقت فانك يجب ان تهتم بماذا يعني البت، البايت، الكلمة. وقد سبق وان وضحنا في الفصل الاول العوامل المنطقية والتي هي تستخدم مع الشرط وسنستخدم هنا مايشبة ذلك قليلا ولكن تتعامل مع البتات.

### 2.5.1 عمليات البتات: العامل ~ Bitwise Not

واحدة من العمليات التي من الممكن ان تنجزها على البت تتمثل بعكس قيمتة. عليه فاذا كانت قيمة البت واحد فانها ستتغير وتكون صفر وبالعكس. هذه العملية سوف يقوم بها العامل Not والذي سيرمز له بالرمز (~). ان العامل Not هو عامل احادي اي يكون معه عامل واحد ويكون هذا العامل على الجانب الايسر كما في المثال:

```
~value
```



| Bit | ~Bit |
|---|---|
| 1 | 0 |
| 0 | 1 |

لنفرض رقم بحجم بايت مثل الرقم 248. بالتاكيد فانك تعلم كيف تحول الارقام من نظام الى اخر، فمثلا ان القيمة الثنائية للرقم 248 هي 10001111 (والقيمة بالنظام السادس عشر هي xF80). فاذا نفذت العامل Not عليه لعكس قيم بتاتة، فانك ستحصل على النتيجة التالية:

| Value | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|
| ~value | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

### 2.5.2 عامل مقارنة البتات (و) The Bitwise AND Operator & Comparing Bits:

| Bit1 | Bit2 | Bit1 & Bit2 |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

العامل And هو عامل ثنائي اي يستخدم مع اثنين من المعاملات ويستخدم وفق الصيغة القواعدية التالية:

Operand1 & Operand2

هذا العامل ياخذ قيمتين ويقارن البت للقيمة الاولى مع البت الذي يقابلة في القيمة الثانية، والنتيجة ستكون وفقا لجدول الصدق المبين ادناه.

تخيل لدينا قيمتان البايت الاولى 187 والثانية 242. استنادا الى دراستنا لانظمة الاعداد فان القيمة الثنائية للعدد العشري 187 هي 1011 1011 (وقيمتة

بالنظام السادس عشر 0xBB))). القيمة الثنائية للرقم العشري 242 هي 00101111 (وقيمتها بالنظام السادس عشر هو(0xF2)، دعنا نقارن هاتين القيميتين بت بت، باستخدام عامل البتات And:

| | ثنـــــــائي | | | | | | | | عشري |
|---|---|---|---|---|---|---|---|---|---|
| N1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 187 |
| N2 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 242 |
| N1 & N2 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 178 |

في كثير من الاحيان تحتاج ان يقوم المترجم بانجاز هذه العملية واستخدام الناتج في البرنامج، هذا يعني امكانية الحصول على النتيجة لهذه العملية وعرضها على شاشة الحاسوب، هذه العملية من الممكن ان نوضحها في المثال التالي.

* برنامج لادخال قيمتين واجراء عملية (و) على بتاتهما.

```
// Example 2.7
#include <iostream>
using namespace std;

main(){
const int N1 = 187;
const int N2 = 242;
cout<<N1<<"&"<<N2<<"="<< (N1 & N2)<<"\n\n";
return 0;
}
```

**مخرجات البرنامج 2.7 :**//

```
187 & 242 = 178
```

### 2.5.3 عامل المقارنة او ( | ) Comparing Bit: Bitwise OR Operator

من الممكن ان تقوم بنوع اخر من المقارنة على البتات باستخدام عامل مقارنة البتات OR والذي يمثل بالعلامة ( | ) والصيغة القواعدية هي:

Value1 | value2

مرة اخرى، فان المترجم يقارن البتات المتقابلة في القيمتين. فاذا كان على الاقل واحد من البتات يساوي 1 فان نتيجة المقارنة ستكون 1. نتيجة المقارنة ستكون صفرا اذا كان البتان المقارنان قيمتيهما صفرا. يمكن ملاحظة ذلك في الجدول ادناه:

| Bit1 | Bit2 | Bit1 \| Bit2 |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

مرة اخرى دعنا نتعامل مع القيمتين 187 و 242 ونقارن بينهم باستخدام عامل مقارنة البتات OR

| | الثنـــــائي | | | | | | | | العشري |
|---|---|---|---|---|---|---|---|---|---|
| N1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 187 |
| N2 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 242 |
| N1 \| N2 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 251 |

وكذلك من الممكن ان تدع المترجم ينجز هذه العملية وتستخدم الناتج في البرنامج.

* برنامج لادخال عددين صحيحين واجراء عملية (او)على بتاتهما وطباعة الناتج.

```
//Example 2.8
#include<iostream>

main(){
const int N1 = 187;
const int N2 = 242;
cout<< N1 << "|" <<N2<<"="<< ( N1 | N2 )<< "\n\n";
return o;
}
```

**مخرجات البرنامج 2.8 ://**

187 | 242 = 251

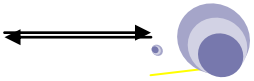
### 2.5.4 مقارنة البتات باستخدام العامل XOR

**Comparing Bits: The Bitwise-Exclusive XOR Operator ( ^ )**

مثل العاملين السابقين فان هذا العامل يقوم بمقارنة كل بتين متقابلين في القيمتين، الصيغة القواعدية هي:

value1 ^ value2

| Bit1 | Bit2 | Bit1 ^ Bit2 |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |



المترجم سيقارن البت لواحدة من القيم مع البت المقابل للقيمة الاخرى. نتيجة المقارنة تعتمد على الجدول اعلاه:

لناخذ مرة ثانية القيمتين 187 و 242، ونقارن بينهما باستخدام العامل XOR ونتيجة هذه المقارنة كما في ادناه:

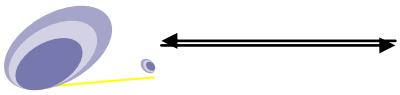
| | الثنـــــائي | | | | | | | | العشري |
|---|---|---|---|---|---|---|---|---|---|
| N1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 187 |
| N2 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 242 |
| N1 ^ N2 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 73 |

اذا ما نفذ المترجم هذه العملية فانه سيولد ناتج من الممكن ان يستخدم ضمن البرنامج.

* برنامج لادخال عددين صحيحين واجراء عملية XOR على بتاتهما وطباعة الناتج.

```
//Example 2.9
#include<iostream>
using namespace std;

main(){
const int N1 = 187;
const int N2 = 242;
cout<< N1<< "^" << N2<< "="<< N1 ^ N2 <<"\n\n";
return 0;
}
```



**مخرجات البرنامج 2.9 ://:**

187 ^ 242 = 73

### 2.5.5 عامل تزحيف البتات لليسار Bit Shift Operators: The Left Shift <<

في المواضيع السابقة، تعلمت ان البتات تنظم بطريقة معينة لخزن البيانات التي تحتاجها. احد العوامل الذي بامكانك استخدامة على البتات يتكون من تحريك البتات باتجاه تختارة. لغة C++ توفر عامل التزحيف لليسار والذي يرمز له (<<) والصيغة القواعدية له هي:

Value << Constant Integer

عامل التزحيف لليسار، هو عامل احادي اي يعمل على قيمة واحدة تكون على يسار العامل ويجب ان تكون القيمة عدد صحيح ثابت. عند تنفيذ هذه العملية، فان المترجم سوف يدفع قيم البتات الى اليسار بعدد محدد مسبقا (Constant Integer) والذي سيكون على يمين العامل <<. البتات التي على اليسار سوف تختفي عند التزحيف وعدد البتات التي ستختفي هي بعدد (Constant Integer) ، بعد تزحيف البتات الى اليسار فان الفراغ المتولد في مواقع البتات في الجانب الايمن سيملأ باصفار.

افرض لديك القيمة 42 حيث ان القيمة الثنائية لها هي 00101010 وترغب بتزحيفها الى اليسار مرتبتين كما يأتي:

```
const int N = 42;
N<<2;
```

هنا ستكون النتيجة كما في ادناه

| | الثنـــــائي | | | | | | | | العشري |
|---|---|---|---|---|---|---|---|---|---|
| قبل التزحيف | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 42 |
| بعد التزحيف: مرتبتين | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 168 |

لاحظ هنا ان البتان على اليسار اختفت واضيف صفران على اليمين. وهذه العملية من الممكن ان تستخدم ناتجها في البرنامج.

* برنامج لاجراء عملية تزحيف بتات الى اليسار (بمقدار بتان) على القيمة 42.

```
//Example 2.10
#include <iostream>
using namespace std;

main(){
const int value = 42;

cout << value<<"<<2="<<(value<<2)<<"\n\n";
return 0;
}
```

**مخرجات البرنامج 2.10 ://:**

42 << 2= 168

### 2.5.6 عامل تزحيف البتات لليمين >> Bit Shift Operators:The Right Shift

وهو يعمل عكس عامل التزحيف لليسار، فهو يزحف بتات القيمة المعطاة الى اليمين وفقا للعدد المحدد للتزحيف. كل شيء يعمل بشكل مشابهة للتزحيف

لليسار ماعدا التزحيف الى الاتجاه المعاكس، لذا لننفذ التزحيف على القيمة 42 الى اليمين بمرتبتين ونلاحظ مايحدث:

| | الثنـــــائي | | | | | | | | العشري |
|---|---|---|---|---|---|---|---|---|---|
| قبل التزحيف | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 42 |
| بعد التزحيف مرتبتين | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 9 |

## 2.6 أمثله محلولة

* برنامج لتحويل (sec42200 ) الى ما يقابلها بالساعات والدقائق والثواني.

```
// Example 2.7
#include<iostream>
using namespace std;
main()
{
  int sec =42200 % 60;
  int temp =42200 / 60;
  int min =temp % 60;
  int hour = temp / 60;
  cout<<"hour="<< hour<<" ,min="<< min<<" ,sec="<< sec;
return 0;
}
```

مخرجات البرنامج 2.7 ://

```
hour= 11, min=43 ,sec=20
```

* برنامج لأيجاد قيمة (y) من المعادلة $y = 4x^2 + 3x - 6$

```
// Example 2.8
#include<iostream>
using namespace std;

main()
{
  int x, y;
cout <<  " Enter number ";
  cin>>x;
  y =4*x*x +3*x-6;
  cout<< y;
return 0;
}
```

**مخرجات البرنامج 2.8 ://**

```
Enter number 10
424
```

* أكتب برنامج لتحويل درجة حرارة مقاسة بالفهرنهايت الى درجة مئوية.

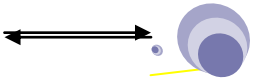
```
// Example 2.9
#include<iostream>
main()
{
  int f;
  cout<<"Enter temperature degree in Fahrenheit "<<endl;
  cin>> f;
  float c =( 5/9)*(f+32);
  cout<< c ;
return 0;
}
```

**مخرجات البرنامج 2.9 ://**

```
Enter temperature degree in Fahrenheit
70
56.6666
```

* برنامج لأيجاد مساحة ومحيط دائرة.

```
// Example 2.10
#include<iostream>
using namespace std;
```



```
main()
{
  const float pi=3.141529 ;
  int r;
  float area,  perimeter;
  cout<<"enter circle radius \n";
  cin>> r;
  area =r*r*pi;
  perimeter =2*r*pi;
  cout<<"area= "<< area<<", perimeter=" <<perimeter;
return 0;
}
```

**مخرجات البرنامج 2.10 ://**

```
enter circle radius
4
area= 50.2644 , perimeter=25.1322
```

* برنامج لأيجاد حاصل ضرب ومعدل ثلاث أرقام.

```
// Example 2.11
#include<iostream>
using namespace std;

main()
{
  int prod, a, b, c;
```

```
 float average;
 cout<<"enter three numbers \n";
 cin>> a >> b >> c;
 prod = a*b*c;
 average =( a + b + c)/3;
 cout<<"prod= "<< prod<< endl;
 cout<<"averge= "<< average;
return 0;
}
```

**مخرجات البرنامج 2.11 ://:**

```
enter three numbers
3 7 9
prod= 189
average= 63
```

## اسئلة للحل ://:

1. اكتب برنامج لايجاد مربع والجذر التربيعي لاي رقم.
2. صحح جزء البرنامج التالي

```
#include<iostream>
Main() {
Char    gap = ‘ ‘ ;
Int       m ; n ;
float   a, b;
```

```
char   c1, c2
int (a + m) = 12 ;
cin >> a >> b >> m >> n ;;
cout << a+b << c1 ;
gap = a + c2 ;
m = a / b ;
cin > c2 ;
cout << n = a * b ;
if ( a = b)     cout << " equal" ;
else        cout << a not equal b ;
}
```

3. اكتب برنامج لايجاد قيمة العلاقة التالية

$$Y = 3x^2 - 2x + 4$$

4. اكتب برنامج لايجاد مساحة مثلث.
5. اكتب برنامج لابدال (swap) رقمين واحد بدل الاخر.

# الفصل الثالث
# ايعازات القرار والتكرار
# DECISION AND REPEAT INSTRUCTIONS

## 3.1 المقدمة

الأن جاء دور دراسة القواعد الأكثر اهمية في البرمجة. وهي ايعازات القرار **(if statement)** وكذلك الأيعاز المرافق لها **(else)** وعبارات التكرار التي هي **(for loop، do.. while loop، while loop)**، غالبا تعد هذه الأوامر من الأوامر الكثيرة الاستخدام في البرمجة لذا ننصح بعد الأنتهاء من دراسة هذا الفصل الشروع بكتابة برامج تستخدم فيها هذه القواعد وزيادة الخبرة العملية قبل الأنتقال الى موضوع جديد.

## 3.2 عبارة اذا (if) Statement

يستخدم هذا الأمر لأتخاذ قرار من قبل المترجم بناءا على بعض المعطيات التي ترد في البرنامج، هناك العديد من الحالات التي لايمكن التنبأ بها من قبل المبرمج أثناء كتابة البرنامج، فعلى سبيل المثال أننا نكتب برنامج لأيجاد الجذر التربيعي لأعداد صحيحة يتم أدخالها من قبل المستخدم أثناء تنفيذ البرنامج، في هذه الحالة وكما معلوم فأن العدد الصحيح يجب أن يكون موجب لأنه لايمكن أيجاد الجذر التربيعي للعدد السالب، السؤال هنا هل يمكن منع المستخدم من أدخال عدد سالب سواء كان بقصد أو سهوا، أن المبرمج سوف لايجد وسيلة أثناء كتابة البرنامج لمعالجة هذا الأشكال البسيط ألا أن يستخدم عبارة القرار (أذا) والتي ممكن أن تكون كما يلي (أذا كان العدد موجب أوجد الجذر التربيعي).. (وبالتأكيد فأن المترجم في الحاسوب لا يفهم عبارة موجب لذا نستبدلها بما يتناسب وقواعد لغة البرمجة C++ فنقول أذا كان العدد أكبرمن أو يساوي صفر فأوجد الجذر التربيعي).

ان استخدام عبارة (if) يكون كما يلي (أذا (شرط)).. (if (condition)) أذا تحقق الشرط الذي يرافق الأمر (if) فيتم تنفيذ العبارة التي بعده أما أذا لم يتحقق

هذا الشرط فيهمل ما بعده (اي تهمل العبارة التي بعده) أذن ستكون طريقة كتابة هذا الأمر كما يأتي:

**if** conditional expression true // لتنفيذ فعل واحد

Statement ;

***ملاحظة ://***

لا توجد بعد الامر ( if ) مباشرة فارزة منقوطة .

عادة يكون تنفيذ البرنامج خطوة بعد الاخرى من الاعلى الى الاسفل حسب ترتيب خطوات البرنامج، عبارة (if) تمكنك من اختيار تنفيذ عمل معين وفقا للشرط المحدد (مثلا، فيما اذا كان متغيران متساويان) والتحول الى جزء مختلف من البرنامج حسب النتيجة، من الممكن اعادة كتابة الصيغة القواعدية للامر (if) كمايأتي:

if (expression)

Statement ;

كل شيء يعوض بقيمة يسمى تعبير(expression) مثل pi، 23+

التعبير بين القوسين ممكن ان يكون اي تعبير ولكن عادة في هذه الحالة يكون احد التعابير العلائقية (اي التعابير التي يكون احد اجزاءها او اكثر متعلق بالاجزاء الاخرى للتعبير، وعادة يتم استخدام العوامل المنطقية)، فاذا كانت قيمة التعبير مساوية للصفر فسوف يعتبر التعبير (false) اما اذا كانت قيمتة لاتساوي الصفر فيعتبر التعبير (true) وتنفذ العبارة (واقعا المترجم هو الذي يحدد القيمة صفر ام لا استنادا الى كونها صحيحة ام لا)، مثال

```
if (bignumber > smallnumber)
bignumber = smallnumber;
```

نلاحظ هنا ان التعبير يقارن بين الرقم الكبير والرقم الصغير فاذا كان الرقم الكبير اكبر من الرقم الصغير فيتم تنفيذ العبارة التي تاتي بعد (if) مباشرة وهي

مساواة العددين في هذا المثال، واذا لم يكن اكبر فلا يتم تنفيذ عبارة المساواة (في هذا المثال هل سيتم تنفيذ المساواة ام لا ؟)، لاحظ هنا ان قيمة الشرط ستكون لاتساوي صفر اذا كانت التعبير صحيح اي ان الرقم الاكبر اكبر من الرقم الاصغر وتكون صفر اذا كان التعبير خاطيء.

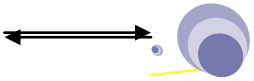
مثال اخر: من الممكن مثلا أن نطلب من أحدهم عملا ونقول له (أذا كان المحل مفتوحا فأجلب لي شراب الببسي)، (get me Pepsi, if shop opening) هذه العبارة ممكن صياغتها برمجيا، كما يأتي:

```
if shop_opening
Drink  = Pepsi ;
```

لاحظ في هذا المثال أن الافعال المطلوب أنجازها هي فعل واحد (أن يجلب لنا شراب الببسي)، أما أذا كان ما مطلوب أنجازة هو أكثر من فعل واحد فأن الصيغة ستختلف حيث ستحدد الاعمال الواجب انجازها عند تحقق الشرط بين قوسي البداية والنهاية لتكون كتلة من العبارات التي تعامل على انها عبارة واحدة:

```
if conditional expression TRUE
{
Statements…
                                    لتنفيذ مجموعة من الأفعال  //          }
```

ماذا يعني ذلك.. ان الأمر (if) ينفذ عبارة واحدة فقط تأتي بعده والتي تمثل الفعل المطلوب انجازة عند تحقق الشرط، أما أذا كان هناك أكثر من فعل واحد مطلوب أنجازة عند تحقق الشرط فيجب أن تحدد هذه الأفعال للمترجم ويكون ذلك بأن تحددها بين الأمرين ({ }) (واللتان تمثلان البداية والنهاية) وبذلك سيكون واضحا أن الأفعال المطلوب تنفيذها عند تحقق الشرط تبدأ بعد الأمر ({) وتنتهي بالعبارة التي قبل (}).



لنعد الى المثال السابق ونطلب من أحدهم عملا ونقول (اذا كان المحل مفتوح فأجلب لي شراب الببسي وقطعة كيك)

(if shop_opening get me Pepsi, and cake)

الفعل المطلوب أنجازة هنا هو أكثر من واحد حيث المطلوب جلب شراب الببسي وقطعة من الكيك، لذا ستكون صياغة هذه العبارة برمجيا كما يأتي:

```
if shop_opening
{
drink  =  Pepsi ;
 food   =  Cake ;
}
```

في حالة عدم وضع ({ }) فان أول عبارة ستأتي بعد الشرط الذي بعد الأمر (if) هي التي ستعامل على أنها تعود الى الأمر (if) وتنفذ في حالة تحقق الشرط وهي هنا ستكون (drink) أما العبارة الاخرى فسوف لاتعامل على انها تابعة للامر (if) والتي هي (food) وتنفذ في جميع الاحوال سواء تحقق الشرط ام لا، اما عند استخدام ({ }) فهي دلالة للمترجم على أن الايعازات المحصورة بين ({ }) جميعا مطلوب تنفيذها أذا ما تحقق الشرط.

اذن بالامكان استخدام عبارة واحدة او كتلة من العبارات (block) حيث ان كتلة العبارات تكون بين قوسي البداية والنهاية وكل عبارة تنتهي بفارزة منقوطة. الكتلة تعامل وكانها عبارة واحدة، فالعبارات الثلاثة التالية تعامل مع الامر (if) على انها مكافئة لعبارة واحدة فأما ان تنفذ جميعا او تهمل جميعا:

```
{ temp = a;     a=b;      b= temp;   }
```

**مثال اخر**

```
if (bignumber > smallnumber)
{
```

```
bignumber = smallnumber ;
cout << " bignumber: " << bignumber <<"\n";
cout<< "smallnumber: " << smallnumber <<" \n";
 }
```

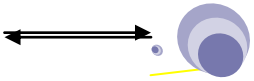
هنا لاحظ ان التعبير بعد (if) يقارن بين رقمين احدهما كبير واخر صغير فاذا كان الرقم الكبير اكبر من الرقم الصغير وهو الحال الطبيعي فيجب ان تنفذ العبارات المحددة بين قوسي البداية والنهاية والتي تمثل كتلة واحدة وهما مساواة العددين ثم طباعة العدد الاكبر بعدها طباعة العدد الاصغر اما في حالة كون التعبير (false) فتهمل الكتلة كلها اي العبارات الثلاث.

***ملاحظة://***

عند الحاجة لاستخدام المساواة في الشرط بعد ( if ) فلا تستخدم المساواة العادية ( = ) ( assignment ) وأنما تستخدم المساواة المزدوجة ( = = ) لأن استخدام الأولى سيؤدي الى عدم اكمال التنفيذ وظهور رسالة خطأ.

هناك حالة أخرى عند استخدام (if)، هو استخدامها لأختيار فعل واحد من أثنين فمثلا في مثالنا السابق ممكن أن يكون الطلب كما يلي ( أذا كان المحل مفتوحا فأجلب لي شراب الببسي وبخلاف ذلك (أي أذا كان المحل مغلقا) فأعمل لي قهوة (if shop_opening, get me pepsi, otherwise get me a coffee ) هذه العبارة تنفذ برمجيا كما يأتي:

```
if shop_opening
  Drink  = Pepsi ;
else
  Drink  = coffee ;
```



لاحظ هنا أن حالة الشرط التي بعد (if) عادة أما أن تكون (صح، أو خطأ) (true OR false) أي أما أن يكون المحل مفتوحا أو مغلقا ولا يوجد أحتمال أخر. فأذا كان المحل مفتوحا فالمطلوب أن يجلب شراب وهو الببسي، في خلاف ذلك (else) أي أذا كان المحل مغلقا فليكن الشراب هو قهوة. الملاحظة المهمة هنا هي أنه لايمكن أن ينفذ العملان سوية أي لا يمكن أن يجلب ببسي وقهوة في نفس الوقت والسبب هو أنه لايمكن أن يكون المحل مفتوحا ومغلقا بذات الوقت. عليه فأذا تحقق الشرط (أي الشرط صح بمعنى أن المحل مفتوح) فأن العبارة التي تأتي بعد الشرط الذي بعد (if) ستنفذ بينما العبارة التي بعد (else) ستهمل، أما أذا كان الشرط غير متحقق (أي أجابة الشرط خطأ بمعنى أن المحل مغلق) فأن العبارة التي بعد (if) ستهمل وتنفذ العبارة التي بعد (else).

المثال التالي مقطع برنامج ممكن أن يكون جزء من لعبة بامكانك ان تضيف اليها أسئله أخرى لتكون لعبة متكاملة:

```
cout<< " Who has discovered the land of America?" ;
cin>> ans ;
if (ans == "Christopher Columbus")
 score = score + 1 ;                    // if condition is true
else              // if condition is false
 cout << "sorry, you've got it wrong! " ;
```

* برنامج لادخال عددين والمقارنة بينهما (التحقق من قيمة العدد المدخل).

```
// Example 3.1
# include<iostream>
using namespace std;
int main()
{
int firstNumber, secondNumber;
```

```
cout <<"Please enter a big number:";
cin>>firstNumber;
cout<<"\n Please enter a smaller number: ";
cin >> secondNumber;
if ( firstNumber > secondNumber)
cout<<"\nThanks!\n";
else
cout<<"\n Oops. The second is bigger!";
return 0;
}
```

***مخرجات البرنامج 3.1***

```
Please enter a big number: 10
Please enter a smaller number: 12
Oops.  The second is bigger!
```

***ملاحظة://***

بالامكان استخدام اكثر من تعبير علائقي في الوقت الواحد بعد ( if ) مستخدمين العوامل المنطقية للفصل بينها وتحسب نتيجتها وفقا لنتيجة العوامل المنطقية

مثال

```
if ( (x == 5 ) && ( y == 5 ))
if ( (x==5) || ( y==5 ))
if ( ! (x==5))
```

هذه العبارة الأخيره صحيحة عندما ( x ) لاتساوي 5 وهي نفس العبارة التالية

```
if ( x !=5)
```

***ملاحظة://***

في لغة C++ فان نتيجة الشرط اذا كانت عبارة خاطئة فسيعيد المترجم القيمة صفر كما بينا واي قيمة لاتساوي الصفر تفسر على ان العبارة صحيحة.

كذلك:

```
if (x)  //  تعني اذا كانت قيمة المتغير لاتساوي صفر اي صح
x=0;
```

هذه العبارة تكون اكثر وضوحا اذا كتبت بالصيغة التالية

```
if (x!=0)
x=0;
```

كذلك فان العبارة التالية

```
if (!x)
```

تعني اذا كانت x تساوي صفر (false) وهي تكافيء

```
if (x==0)
```

والعبارة الاخيرة اكثر وضوح

***ملاحظة://***

يفضل استخدام الاقواس حول الاختبارات المنطقية لجعلها اكثر وضوحا كذلك يفضل استخدام الاقواس مع (if) المتداخلة ( المركبة) لجعل عبارة (else) اوضح ولتجنب الاخطاء.

### 3.2.1 عامل الشرط الثلاثي ( Conditional Ternary Operator :?)

عامل الشرط الثلاثي يفحص تعبير، ويعيد قيمة معينة اذا كان ذلك التعبير صح، ويعيد قيمة مختلفة اذا كان ذلك التعبير خطأ، هذا العامل هو اختصار لعامل الاختيار (if. else)

الصيغة العامة له:

condition ? result1: result2

فاذا كان الشرط (condition) صح فان التعبير سيعيد القيمة (result1) اما اذا كان خطأ فانه سيعيد القيمة (result2)

**مثال:**

7==5 ? 4: 3 // يعيد (3) حيث ان (7) لاتساوي (5)

7==5+2 ? 4: 3 // يعيد (4) لان (7) تساوي (2+5)

5>3 ? a: b // يعيد القيمة ( a) لان (5) اكبر من (3)

a>b ? a: b // يعيد ايهما اكبر( a) او (b)

هذا التعبير الثلاثي يمكن ان نعبر عنه بما يأتي (اذا كان الشرط صحيحا فعلية ستكون النتيجة هي النتيجة الاولى وبخلاف ذلك اي اذا كانت نتيجة الشرط غير صحيحة فستكون النتيجة هي النتيجة الثانية). عادة هذه القيمة المعادة يجب ان تسند الى متغير. مثال

```
{
int min ،i=10 ،j=20;
min =(i < j ? i: j);
cout<<min;
}
```

* برنامج لأدخال عددين وطباعة الاكبر

```
// Example 3.2
```

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

int main()
{
int x, y, z;
cout<<"Enter two numbers.\n";
cout<<"First:";
cin>>x;
cout<<"\n Second: ";
cin>>y;
cout<<"\n";
if (x > y)
z=x;
else
z = y;
cout<<"z:"<<z;
cout<<"\n";
z= (x > y) ? x : y;
cout<<"z:"<<z;
cout<<"\n";
return 0;
}
```

**مخرجات البرنامج 3.2:**

```
Enter two numbers. First: 5
Second: 8
z:8
z:8
```

## 3.3 اذا المركبة Compound if

من الممكن أن تستخدم (if) بشكل متداخل مع (if OR else) أخرى، وبهذه الحالة تسمى مركبة (أي ممكن أن يكون بعد الشرط الذي بعد (if) عبارة (if) أخرى وممكن أيضا بعد عبارة (else) وممكن أن تكون أكثر من عبارة (if) واحدة. **فمثلا** تريد أن تفحص نوعية رمز معين ووفقا لذلك تقرر ماهو الأجراء الواجب أتباعة وكمايأتي:

```
if (expression1)
  {    if (expression2)
          Statment1;
else
     {   if (expression3)
            Statment2;
          else
            Statment3;
       }
   }
else
    Statment4;
```