الكافي في لغتي السي و السي++

C++ Language

تاليف :
أحمد محمد (المتألق)
تصميم :
منتديات العاصفة

بسم الله الرحمن الرحيم

الحمد لله الذي بحمده يُستفتح كل كتاب و بذكره يَصْدُر كل خطاب وبفضله يتنعم أهل النعيم في دار الجزاء و الثواب والصلاة و السلام على سيد المرسلين و إمام المتقين المبعوث رحمة للعالمين محمد ابن عبد الله الصادق الأمين و على صحابته الأخيار و من تبعهم بإحسان إلى يوم الدين أما بعد :

قمتُ بتأليف هذا الكتاب كهدية متواضعة مني إلى منتديات العاصفة و منتديات الفريق العربي للبرمجة, وقد حاولت أن يكون هذا الكتاب دليلا شاملا لكل من أراد دراسة لغة C أو لغة $C++$, بحيث يكون بداية انطلاق المبرمجين المبتدئين الذين يريدون الدخول في عالم البرمجة من خلال تعلم هاتين اللغتين.

إذا قرأت كتابي وانتفعت به *** فاحذر وقيت الردى من أن تغيره

ودعه لي سالما إني شغفت به *** لولا مخافة كتم العلم لم تره

شكر خاص

أشكر المصمم المحترف "المخترق" الذي قام بتصميم غلاف الكتاب و بالمناسبة فهو أحد أبرز أعضاء إدارة منتديات العاصفة كما يشرف حاليا على منتدى التصميم و الجرافيكس في العاصفة.

حول الكتاب:

لقد اعتمدتُ في هذا الكتاب على طريقة مُبسطة للشرح لكي يفهم القارئ بشكل جيد .. مُحاولا الجمع ما بين الجانب النظري و التطبيقي ففي نهاية كل فصل أضع *10* تمارين محلولة (مع الشرح المفصل) لكي تتضح الأمور بشكل أفضل بالإضافة إلى *20* تمرين (للتدريب) في نهاية الكتاب من أجل أن يتمرن القارئ على التعامل مع المسائل بصفة برمجية.

الجزء الأول من الكتاب يحتوي على أساسيات اللغتين مثل:

1-لمحة تاريخية عن *C* و *C++*.

2-المتغيرات و الثوابت و المؤثرات.

3-الجمل الشرطية و القرار *switch*.

4-الحلقات التكرارية.

5-الدوال.

6-المصفوفات.

7-تمارين مُختارة.

9-نصائح لمن يريد الإحتراف.

هذا الجزء يُعالج المفاهيم الأساسية و الفصول الأولى من هذه اللغة, أما بالنسبة للجزء الثاني (مازال قيد الكتابة) فيحتوي على المؤشرات و التراكيب و الملفات و مواضيع أخرى متقدمة.

نظرا لوجود بعض الأخطاء في النسخة الأولى من هذا الكتاب فقد قمت بنشر هذه النسخة (الإصدار الثاني) التي ركزت فيها على تصحيح الأخطاء و تكملة بعض الدروس و إضافة المزيد من التمارين.

لا زلت أنتظر باقي الملاحظات من قراء هذا الكتاب, لذا أرجوا من كل من قرأ هذا الكتاب و وجد به أخطاء سواء كانت إملائية أو معنوية أن يقوم بتنبيهي على بريدي الإلكتروني و له جزيل الشكر.

و الكتاب مفتوح لكل من يريد إضافة حرف, كلمة, جملة, جزء أو أي شيء مفيد,كما أرحب بأي تعليقات أو ملاحظات.

عن المؤلف:

الإسم : أحمد ولد محمد.

سنة الميلاد: 1992

الدولة: موريتانيا.

الهواية: البرمجة و التصميم و الرياضة.

المستوى: طالب جامعي بكلية العلوم و التقنيات.

للاستفسار و تبادل الخبرات: *elmoute2eli9@yahoo.fr*

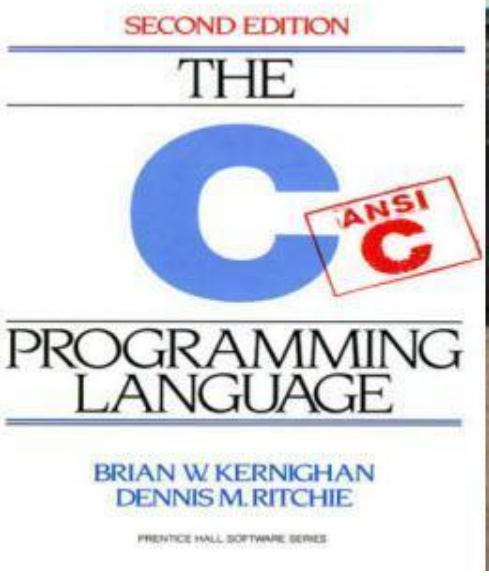
الفصل الأول: البداية مع *C* و *C++* والأدوات اللازمة.

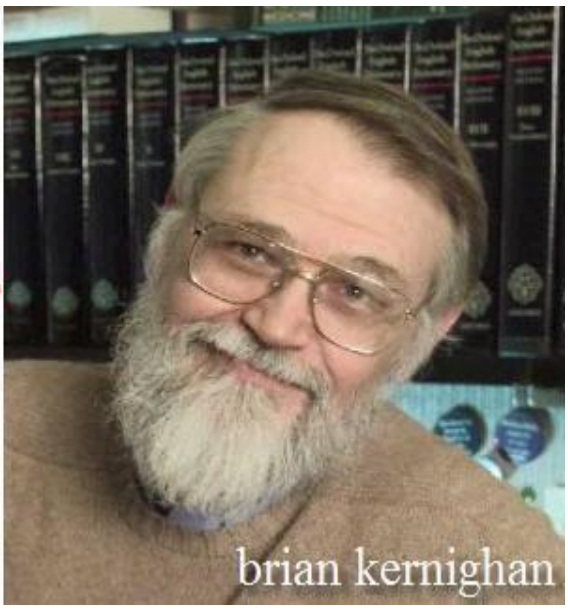
I- مُقدمة:

في الخمسينيات من القرن الماضي كانت توجد مجموعة من لغات البرمجة منها لغة الـ *Fortran* و الـ *Cobol* و الـ *Basic* و *Pascal* و ...., و كان لكل من هذه اللغات مجالها الخاص !, حيث كانت تُستخدم لغة الفورتران في التطبيقات الهندسية و العلمية .. بينما تُستخدم لغة الكوبول في التطبيقات التجارية .. و هكذا بالنسبة للبقية.

بدأ مطوروا لغات البرمجة يسعون لدمج هذه اللغات في لغة واحدة تتميز برامجها بسرعة التنفيذ و سهولة الإستخدام وكانت المحاولة الأولى من نصيب لغة *CPL* و هي اختصار لـ *Combined Programming Language* حيث قامت بتطويرها جامعة كامبريدج (*Cambridge*) في المملكة المتحدة البريطانية, أما المحاولة الثانية فقد كانت على يد الخبير مارتين ريتشارد (*Martin Richards*) و هي لغة *BCPL*,كان ذلك في سنة 1967 في جامعة كامبريدج أيضا, بينما كانت المحاولة الثالثة من طرف السيد كين تومسون (*Ken Thompson*) وهي لغة *B* حيث كانت هذه اللغة أحسن من سابقاتها إلا أن ذلك لم يمنع من كونها بطيئة إلى حد ما ! و في سنة 1972 ولدت لغة *C* على يد خبير يُدعى دنيس ريتشي (*Dennis Ritchie*) في الولايات المتحدة الأمريكية و بالتحديد في مختبرات *Bell* .

في عام 1978 قام دنيس ريتشي و براين كارنيغان (*brian kernighan*) بتأليف أول كتاب لهذه اللغة تحت عنوان : *The C Programming Language*
و الذي يعتبر المرجع الأساسي لهذه اللغة.

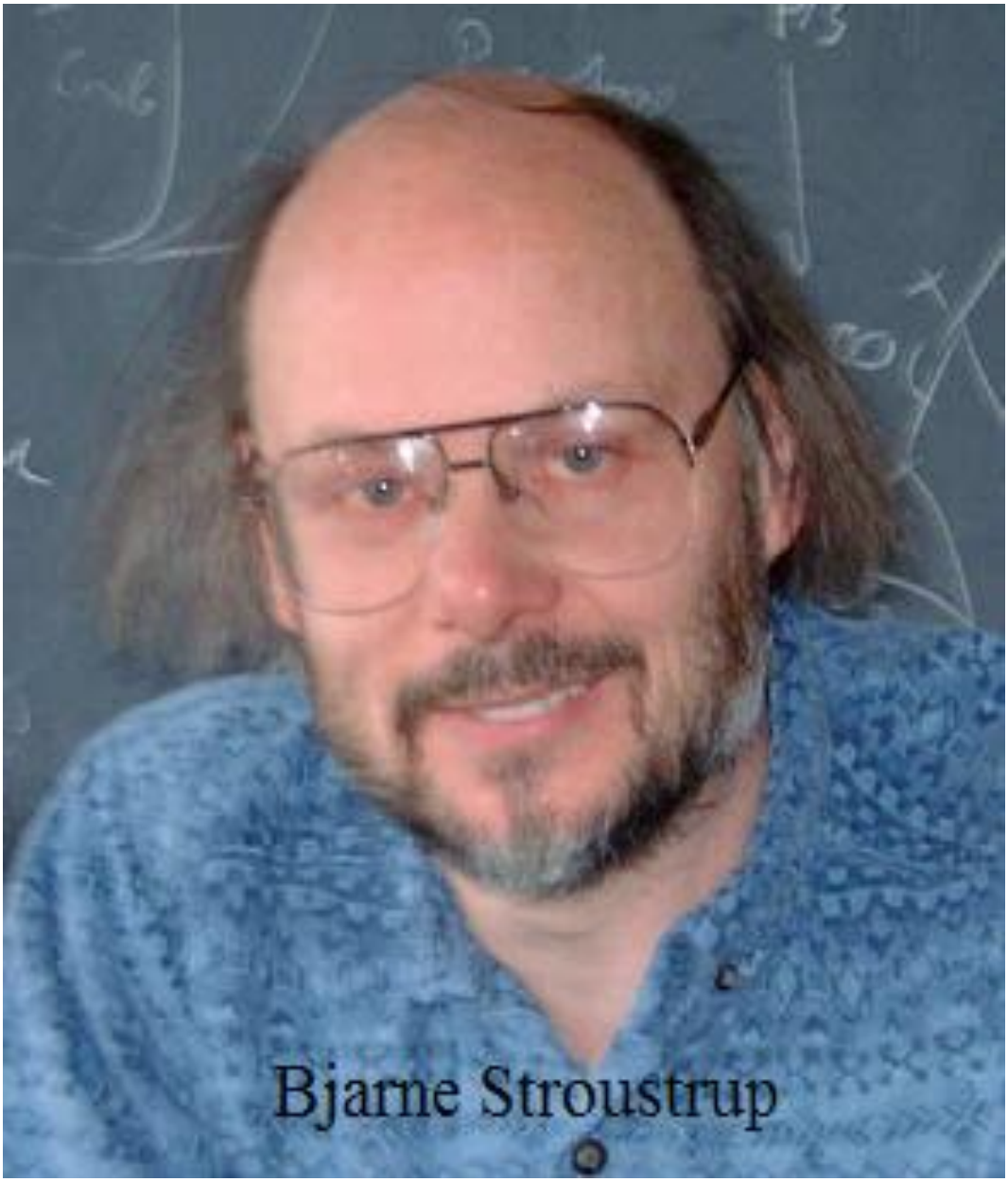




كانت لغة *C* تحتوي على كثير من المميزات التي يرغب بها المبرمجون مثل السرعة و كونها متنقلة (حيث يمكن استعمالها تحت أكثر من نظام تشغيل) و هي لغة قياسية أيضا ففي عام 1989 تم تعريف نسخة قياسية من لغة *C* و سميت بـ *ANSI C* و هي مختصرة من *American National Standards Institue C* أي اللجنة الوطنية الأمريكية للمعايير, و بتعاون بين اللجنة الوطنية الأمريكية للمعايير و المنظمة العالمية للمعايير تم إطلاق لغة *C* القياسية سنة 1990 و سميت بـ *ISO C* و هي اختصار لـ:
International Organization For Standarizatoin

أما بالنسبة للغة السي++ فقد كانت تسمى *C with classes* وتغير اسمها إلى *C++* في سنة 1983 و قد وُلدت على يد الخبير (*Bjarne Stroustrup*) في الثمانينات.
كان ذالك تزامنا مع ظهور البرمجة الكائنية الموجهة و التي تدعى *OOP* و هي اختصار لـ *Object Oriented Programming*, و هي عبارة عن طريقة تفكير جديدة للغات

البرمجة (جيل جديد), لذلك كان لابد من ظهور لغة تكون على المستوى المطلوب بحيث يمكنها منافسة باقي لغات البرمجة, و السي++ هي نسخة مطورة من السي.. بمعنى أن كل ما نستطيع القيام به عن طريق لغة السي يمكننا فعله بلغة السي++ و العكس ليس صحيح دائما.



و لا يزال تطوير لغات البرمجة مُتواصلا ... فهناك نسخة جديدة من *C++* قيد التطوير وتسمى *C++0X*, وقد صدرت منها نسخة *Draft* (قابلة للتغيير) في مارس 2009.

II-الأدوات اللازمة:

لكي نقوم بتنفيذ أي كود برمجي في لغة C أو C++ لا بد من وجود برنامج يُقال له الـ compiler أو المترجم, و لكن .. ما هو المترجم !؟ و لماذا نحتاج إليه!؟و ما هي أشهر المترجمات المتداولة حاليا !؟و ما هي المراحل التي يمر بها الكود أثناء إنشاءه !؟

1-ما هو المترجم !؟

المترجم (بشكل عام) هو برنامج يقوم بتحويل كود مكتوب بلغة برمجة عالية المستوى إلى لغة برمجة أخرى منخفضة المستوى, و بشكل خاص فإن المترجم هو برنامج يقوم بتحويل كود (C أو C++) إلى لغة منخفضة المستوى (قد تشبه لغة الأسمبلي) تسمى *Intermediate Representation*.

2-لماذا نحتاجه !؟

لنفرض أنك لا تعرف اللغة اليابانية و ذهبت إلى اليابان للقاء أحد الأشخاص هناك !!! الآن لديك مشكلة .. فأنت تريد التحدث مع شخص لا يعرف سوى اليابانية و أنت لا تعرف لغته !, يعني باختصار: كلاكما لا يعرف لغة الآخر!.

إذا أردتُما التحدث إلى بعضكما البعض فلابد من وجود شخص ثالث يعمل كمترجم فيقوم بتحويل كلامك إلى اللغة اليابانية (لكي يفهمه الياباني) و يحول كلام الياباني إلى اللغة العربية (لكي تفهمه أنت), إذن فهو يعمل كوسيط بينكما لكي يفهم كل منكما تعليمات الآخر.

التعريف السابق بنفس الشكل والصورة هو الذي يحدث في عالم البرمجة, فجميعنا نعرف أن الحاسب (المعالج والذاكرة) لا يتعاملان إلا مع الأرقام الثنائية 0 و 1 (سريان تيار أو عدم سريانه), فلكي يفهم الحاسوب برامجنا لا بد أن يتم تحويلها إلى لغة الآلة , وهذا بالضبط ما يفعله المترجم حيث يقوم بالتحويل من لغة إلى أخرى (قد تكون لغة الآلة أو لغة أخرى على حسب نوع المترجم والغرض المصمم من أجله.)

عوداً على بدء .. و لكي يكون الكلام أدق فإن الشخص الثالث (المترجم) يجب أن يقوم بالترجمة بشكل صحيح لذلك عليه أن يفهم مالذي يقوله الشخص الأول (**تحليل الكلام**) ومن ثم يقوم بترجمة ما فهمه إلى الشخص الثاني (**توليد الكلام**).

نفس الأمر يحدث لل *compiler* حيث يقوم بتحليل النص أولا في مرحلة يمكن أن نطلق عليها إجمالا بـ*Analysis Phases*, بعدها يقوم بتحويل الكود إلى اللغة الجديدة المراد الوصول إليها وهي مرحلة التوليد أو ما يعرف بـ *Sysntheses Phases*.

ولكن .. لماذا يحلل المترجم وبعدها يبدأ بالتوليد ؟

سنوضح هذا السؤال بسؤال آخر !, لنفرض أنك تتكلم مع الشخص الثالث (ليترجم ما قلتَه إلى الياباني) و قلت له كلام ليس بالعربية !! (المترجم يحول من العربة إلى اليابانية فقط), الآن سيعترض المترجم .. لأن الكلام الذي قلتَه لا يملك معنى في العربية حتى يتم تحويله إلى اليابانية, نفس الشيء بالنسبة لل *compiler* , فالمترجم يحلل الكلام للتأكد من أن البرنامج صحيح وموافق تماما لقواعد اللغة, فإذا كان هناك خطأ ما في هذه المرحلة سيتوقف المترجم عن العمل وسيخبرك بمكان هذا الخطأ, إذا كان الخطأ نحوي (يتعلق بقواعد اللغة *Syntax Error*) سيتوقف المترجم مباشرة عن العمل, أما في الأخطاء الأخرى فلن يتوقف عن العمل وسيكمل مرحلة تحليله واكتشاف باقي الأخطاء, طبعا هذا ما يعرف ب *Error Recovery*.

3- ما هي أشهر المترجمات المتداولة حاليا !؟ يوجد العديد من المترجمات .. منها المترجم **العملاق** *Visual C++* المقدم من شركة *Microsoft* و المترجم **الرائع** *Dev C++* المقدم من شركة *Bloodshed* و المترجم **العجوز** *Borland C++* و المقدم من شركة *Borland*, كما يوجد أيضا المترجم **الجبار** *Code::Blocks* و إذا أردنا سرد أسماء جميع المترجمات المتميزة فالمقام يطول ...!

4- ما هي المراحل التي يمر بها الكود أثناء إنشاءه !؟

تمر عملية إنشاء أي برنامج مكتوب بلغة *C* أو *C++* بست مراحل هي:

1–Editing
2–Preprocessing
3–Compiling
4–Linking
5–Loading
6–executing

شرح هذه المراحل:

في عملية الـ *Editing* نقوم بكتابة كود البرنامج في محرر نصوص ونقوم بعمل التعديلات عليه والتصحيح إذا لزم الأمر, ثم نقوم بتخزين ملف الكود على وحدة تخزين ثانوية مثل الـ *Hard disk*, عندما نقوم بحفظ الملف يجب أن يكون بأحد الامتدادات التالية : *(.cpp) or (.cxx) or (.c)*, يوجد في نظام التشغيل يونيكس محررات مشهورة لهذا الغرض مثل *(vi & emacs)*وفي الويندوز يوجد *Notepad* وفي الدوس يوجد *edit* *programme*, كما توجد أيضا حزم برمجيات سي++ لأجهزة الكمبيوتر الشخصية مثل :

Turbo C++,Borland C++,C++ Builder and Microsoft VC++

ويوجد بداخلهم محرر نصوص تستطيع من خلاله كتابة الكود.

ملاحظة:

- الكود الذي كتبناه داخل محرر النصوص يدعى *Source code*
- الملف الذي يحتوي على *Source code* يدعى *Source file*

بعد مرحلة التحرير (*Editing*) نقوم بإصدار أمر *Compile* فتأتي مرحلة الترجمة أو الـ *Compiling*, ومن المعروف أن الكومبايلر يقوم بتحويل الكود إلى لغة الآلة ولكن قبل ذلك يتم تنفيذ خطوة الـ *Preprocessor* **أوتوماتيكيا**.

برنامج الـ *Preprocessor* يكون مدمج مع الـ *Compiler* في معظم المترجمات الحديثة, حيث يقوم هذا البرنامج بالبحث عن الجمل التي تبدأ بـ #, و التي تدعى *Compiler directives*, يقوم برنامج الـ *Preprocessing* بتوجيه هذه الجمل والتعديل في ملف الـ *Source code*الأصلي, ثم يقوم الـ *Compiler* بأخذ الملف الناتج من عملية الـ *Preprocessing*ويقوم بإنتاج ملف آخر يُدعى

Object file, هذا الملف يحتوي بداخله على الـ *object code* أو *machine code* الذي يفهمه الـ *CPU* الموجود بالكمبيوتر.

حتى الآن لا يستطيع الكمبيوتر تنفيذ الـ *object code* ! و ذلك لاحتوائه على أجزاء لم تحول إلى لغة الآلة بعد, فلغة الآلة تتكون من 0 و 1 فقط.

في هذا الموقف نقول أن الـ *object code* يحتوي على *Undefined references* و هي أجزاء في الـ *object code* لم يتم تعويضها من أي مكان لإتمام البرنامج .. على سبيل المثال ممكن أن يحتوي الـ *object code* على نداءات لدوال ولكنه إلى الآن لا يحتوي على الهيئة التي يمكن تنفيذها بواسطة وحدة *CPU* لتنفيذ تلك الدوال.

هذه الهيئة التي نحتاجها من الممكن أن تكون متوفرة في مكان خارجي كملف مكتبة (أو ما يعرف بـ *Library file*) أو متوفرة في *object file* آخر ولكن البحث عن هذه الأجزاء المفقودة ليس من اختصاص الـ *Compiler* فيظهر هنا دور الـ *Linker* .

يقوم برنامج الـ *linker* بتعويض الـ *undefined references* من *object files* و المكتبات المتضمنة مع اللغة, أي يقوم بالبحث عن الكود المكون من (الصفر و الواحد) والمقابل لـ *undefined refernces* حتى يتم تعويضه في *object file* وهذا الكود يكون موجود في مكتبه من مكتبات اللغة أو في *object file* آخر.

وبعد عملية *Linking* يكون الملف الناتج هو ملف يدعى *Executable file* ويحتوي على *Executable code* , الملف الأخير كله عبارة عن (0 و 1) و هو جاهز للتنفيذ ولكي يتم تنفيذ البرنامج يجب أولا تحميله على ذاكرة الكمبيوتر ولذلك هناك جزء من *Executable code* يقوم بتحميل البرنامج إلى الذاكرة ويدعى هذا الجزء بـ*loader*

ملاحظة: سأعتمد في الشرح على البرنامج *Dev C++* , لتحميل *Dev* اضغط على الرابط أدناه:

http://prdownloads.sourceforge.net/dev-cpp/devcpp-4.9.9.2_setup.exe

جميع المترجمات الحديثة متوفرة لها عدة *IDE*, ماذا يعني هذا المصطلح؟

أولا الكلمة ***IDE*** مختصرة من ***Integrated Development Environment*** أي **بيئة تطوير متكاملة**, حيث تساعد هذه المترجمات المبرمج في كل من التحرير,الترجمة و الربط ففي السابق كانت الترجمة و الربط يتمان على شكل أوامر ! أما في الـ *IDE* فأصبحت عملية الربط و الترجمة تتم عن طريق الضغط على زر واحد من لوحة المفاتيح

بمعنى أدق : بيئة التطوير هي من يقوم بتنفيذ هذه الأوامر بدلا منك.

الفصل الخامس: الدوال.

تعريف : الدوال هي مجموعة من الأوامر و البيانات مُسجلة تحت اسم واحد حيث يمكن استدعائها في أماكن مختلفة من البرنامج.

I-أنواع الدوال

a-دوال اللغة.

b-دوال المبرمج المبتكرة.

بالنسبة لدوال اللغة فهي دوال مُعَرَّفَةٌ مُسبقا مثل الدالة *printf* و *scanf* و *system* أي أنها جاهزة للإستخدام و هي بدورها أيضا تنقسم إلى قسمين :

a-1) الدوال القياسية :

عندما أرادت الشركات المُطورة للغات البرمجة تصميم كومبايلر للغة السي قامت بعض الشركات بتطوير مكتبات و دوال في هذه اللغة .. مما أدى إلى اختلاف في نسخ لغة السي لأن كل شركة تُصمم دوال و مكتبات على حسب رغبتها !!! و أصبحت نسخ هذه اللغة تكاد تكون غير مُتشابهة و هذا ما أدى إلى تعريف نسخة قياسية للغة السي في عام 1989.

الدوال القياسية هي التي يمكن استخدامها في أي مترجم فهي عبارة عن عامل مشترك بين جميع المترجمات (متفق عليها).

b-1)الدوال الغير قياسية : وهي الدوال الغير مُتفق عليها دوليا أي أنها من صنع شركات معينة مثل :

clrscr تستخدم لمسح الشاشة.

gotoxy تستخدم لوضع المؤشر في العمود x من الصف y.

textcolor و *textbackground* هاتان الدالتان تستخدمان على التوالي لتغيير لون الشاشة ولون الشاشة الخلفية.

الدوال السابقة من صنع شركة *borland* بمعنى أنه لا يُمكن استخدامها إلا في المترجمات التابعة لهذه الشركة.

b-دوال المبرمج المبتكرة :

عند تكرار مجموعة من الأوامر أكثر من مرة في مواضع مختلفة من البرنامج فإن أوامر التكرار لن تكون ذات منفعة .. ولذلك يتم كتابة هذه السطور منفصلة عن البرنامج الأساسي كدوال .. أي أننا نُسَلِّم هذه المهمة إلى دالة (نحن من يُنشأها) ثم نقوم باستدعائها كل ما احتجنا لها.

سنقوم بتوضيح ما سبق بمثال :

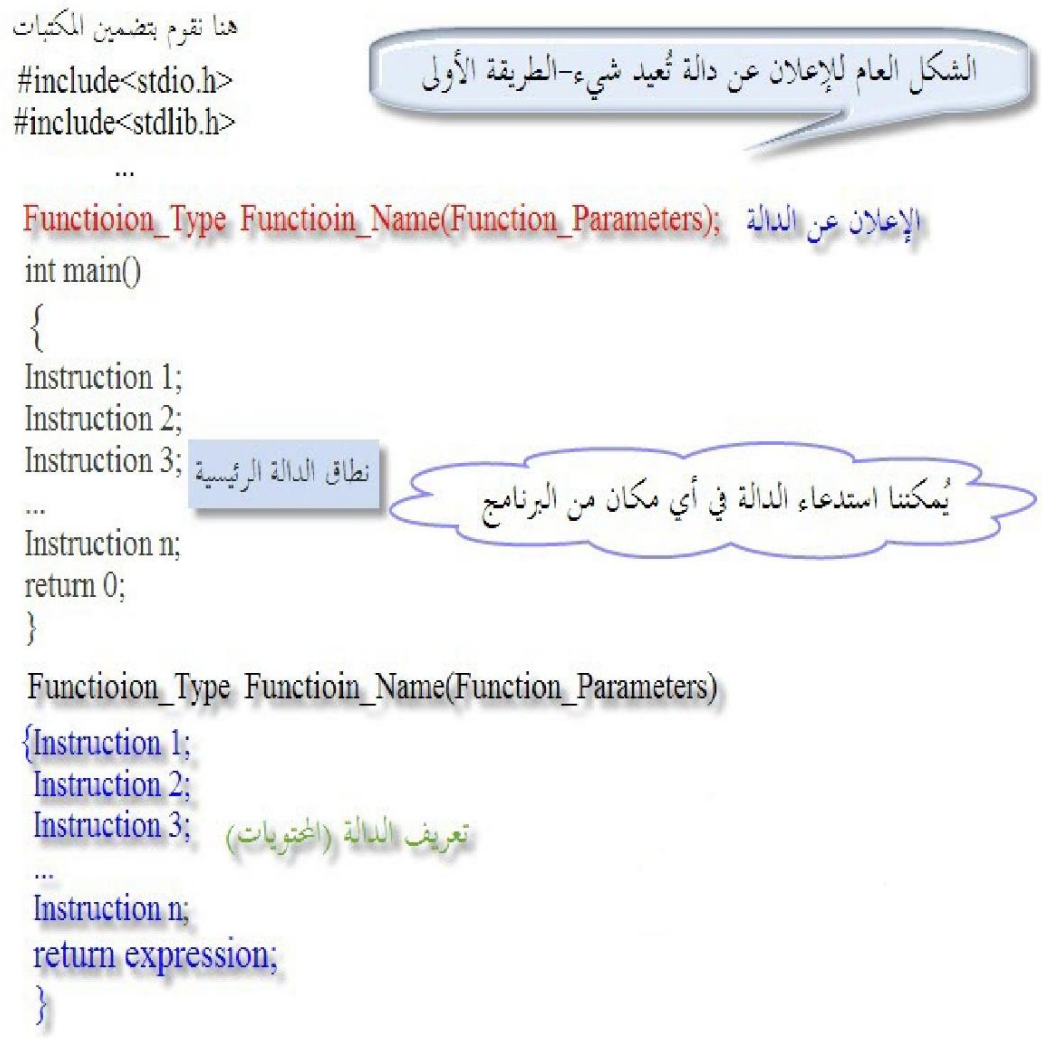
لنفرض أنك تريد كتابة برنامج يقوم بحساب معدل مجموعة من الأعداد يدخلها المستخدم و تنتهي بصفر ,لكنك ستقوم بحساب المعدل في أماكن مختلفة من البرنامج .. بإمكانك كتابة الأوامر المتعلقة بحساب المعدل كلما أردت حسابه .. لكن بهذه الطريقة سيكون الكود طويل وممل أيضا ... هنا تظهر إحدى فوائد الدوال ..كيف؟ .. سنقوم بإنشاء دالة تقوم بحساب معدل مجموعة من الأعداد تنتهي بصفر و سنستدعي هذه الدالة كلما احتجنا إليها .. وبهذه الطريقة سيكون الكود قصير جدا و مفهوم أيضا.

II-طريقة الإعلان عن الدوال

توجد طريقتين للإعلان عن الدوال هما :

a-الطريقة الأولى : نقوم بالإعلان عن الدالة قبل الدالة الرئيسية الـ *main* ثم نستخدم الدالة داخل نطاق الدالة الرئيسية وبعد الإنتهاء من الدالة الرئيسية نكتب تعريف الدالة (محتوياتها) ,هذه هي الطريقة المفضلة لأنها أسهل من ناحية التنظيم.

صورة توضيحية :



مثال :

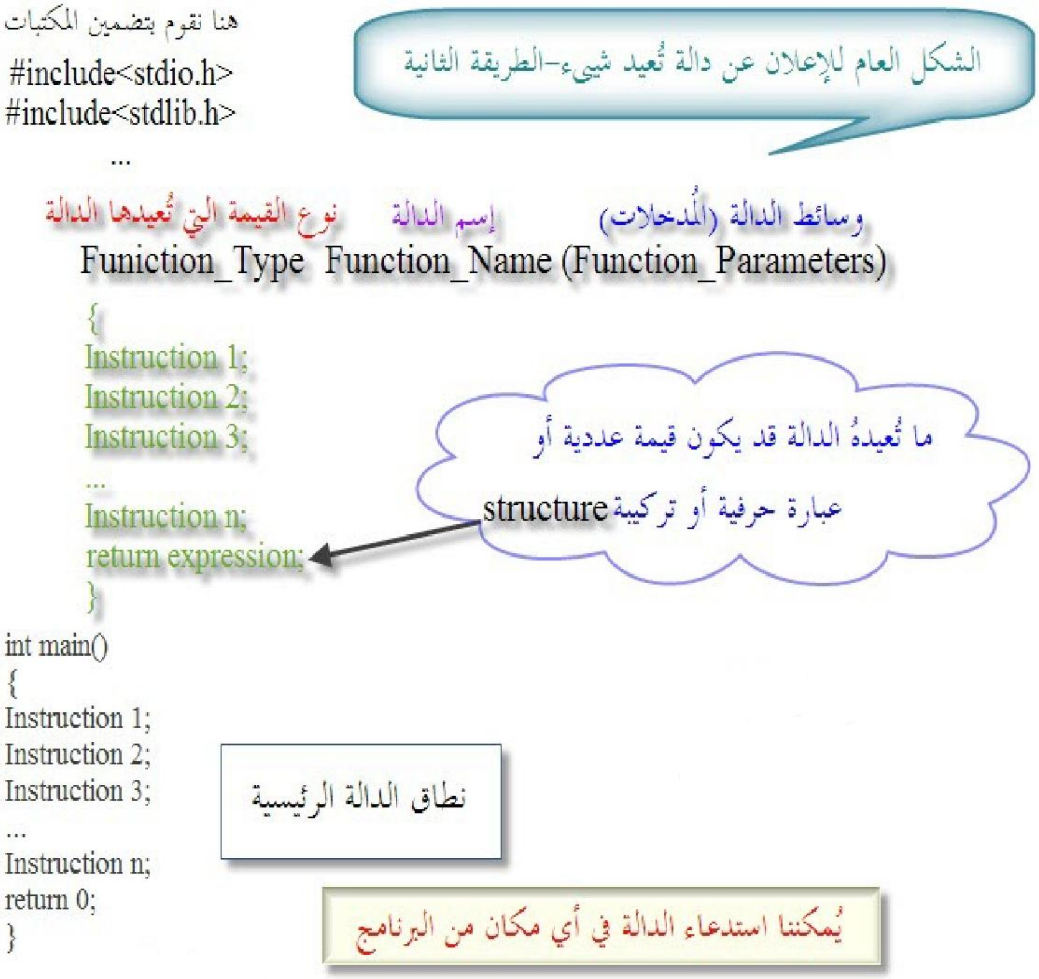
```
#include<iostream>
using namespace std;
float addition(float a,float b);//الإعلان عن الدالة
int main()
{
    float n,m;
    cout<<"n=";
    cin>>n;
    cout<<"m=";
    cin>>m;
    cout<<n<<"+"<<m<<"="<<addition(n,m)<<endl;//استدعاء الدالة
    system("pause");
    return 0;
}
//تعريف الدالة
float addition(float a,float b)
 {
  return a+b;
 }
```

في هذا المثال قمنا بالإعلان عن دالة اسمها *addition* و تستقبل عددين حقيقيين و تعيد عددا حقيقيا,هذا هو الإعلان عن الدالة وقد كتبناه قبل الدالة الرئيسية,لاحظ كتابة الفاصلة المنقوطة ; بعد تعريف الدالة.

في السطر الحادي عشر قمنا باستخدام الدالة بالنسبة للعددين n و m وبعد انتهاء الدالة الرئيسية قمنا بكتابة تعريف الدالة أي الأوامر التابعة لها وهي *return a+b* أي أن هذه الدالة تستقبل عددين حقيقيين و تُعيد عددا حقيقيا عبارة عن جمع العدد المدخلين.

b-الطريقة الثانية :

نقوم بالإعلان عن الدالة (مع كتابة محتوياتها) قبل الدالة الرئيسية الـ *main* ثم نستخدم الدالة داخل نطاق الدالة الرئيسية.



مثال :

```
#include<iostream>
using namespace std;
float addition(float a,float b)
 {
  return a+b;
 }
int main()
{
    float n,m;
    cout<<"n=";
    cin>>n;
    cout<<"m=";
    cin>>m;
    cout<<n<<"+"<<m<<"="<<addition(n,m)<<endl;
    system("pause");
    return 0;
}
```

الدوال التي لا تعيد شيئا

هذا النوع من الدوال لا يقوم بإرجاع أي قيمة ولذا فإن نوعه يكون *void* أي فارغ و أحد استخداماته أنه إذا كانت هناك عدة جمل نُريد كتابتها في الدالة الرئيسية في أكثر من مكان فإنه بدل كتابتها عدة مرات يتم كتابتها في دالة فارغة ثم يتم استدعاء الدالة بكتابة اسمها فقط في المكان الذي نريد كتابة الجمل فيه.

مثال:

```
#include<iostream>
using namespace std;
void affiche(void);
int main()
{
    affiche();
    system("pause");
    return 0;
}
void affiche(void)
{
 cout<<"by elmoute2eli9"<<endl;
}
```

لو أردنا تعديل الكود السابق ليكون كالتالي:

تقوم الدالة *affiche* بإظهار الجملة *n* مرة ,حيث *n* عدد صحيح يُدخله المستخدم.

الكود بعد التعديل:

```
#include<iostream>
using namespace std;
void affiche(int n);
int main()
{
    int a;
    cout<<"a=";
    cin>>a;
    affiche(a);
    system("pause");
    return 0;
}
void affiche(int n)
{
 for(int i=n;i!=0;i--)
 cout<<"by elmoute2eli9"<<endl;
}
```

لاحظ أن العدد a عندما يكون سالبا فإن الدوارة ستكون لا نهائية .. لذا يجب التحقق من أن قيمة a موجبة أو معدومة.

```
#include<iostream>
using namespace std;
void affiche(int n);
int main()
{
    int a;
    debut:
    cout<<"a=";
    cin>>a;
    if(a<0)
    {
      cout<<"erreur !\a entrez un nombre positif ou null"<<endl;
      goto debut;
    }
    else
    affiche(a);
    system("pause");
    return 0;
}
void affiche(int n)
{
 for(int i=n;i!=0;i--)
 cout<<"by elmoute2eli9"<<endl;
}
```

III-بعض الدوال الكثيرة الإستخدام

1-الدالتين putchar و getchar :

الدالة *putchar* اختصار لـ *put charcter* أما الدالة *getchar* فهي اختصار لـ *get charcter* ,هاتان الدالتان تابعتان للمكتبة *stdio.h* حيث تُستعمل الدالة الأولى لإظهار حرف على الشاشة هكذا :

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    putchar('b');
    putchar('y');
    putchar(' ');
    putchar('e');
    putchar('l');
    putchar('m');
    putchar('o');
    putchar('u');
    putchar('t');
    putchar('e');
    putchar('2');
    putchar('e');
    putchar('l');
    putchar('i');
    putchar('9');
    putchar('\n');
    system("pause");
    return 0;
}
```

ربما لا توجد فائدة في استخدامها في هذا المثال لأنه يمكننا استخدام الدالة *printf* لتنفذ الأمر في سطر واحد .. ولكن تظهر إحدى فوائد هذه الدالة عند التعامل مع الحروف أو عند طباعة مصفوفة حرفية حرف حرف باستخدام دوارة.

أما بالنسبة للدالة *getchar* فتقوم بتخزين حرف معين في متغير حرفي,هكذا:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  char ch;
  printf("tapez un caracter:");
  ch=getchar();
  printf("le caracter que vou avez tapez est :%c\n",ch);
  system("pause");
  return 0;
}
```

ويمكننا القول أن الدالتين *putchar* و *getchar* عبارة عن حالة خاصة من الدالتين *printf* و *scanf* و لكنهما قد يكونان أحيانا أفضل منهما وذلك عند التعامل مع الحروف و المصفوفات الحرفية.

2-الدالتان *puts* و *gets*:

هاتان الدالتان تابعتان أيضا للمكتبة *stdio.h*, الدالة الأولى مختصرة من *put string* أما الثانية فمختصرة من *get string*, الدالة *puts* مشابهة جدا للدالة *printf* مع وجود بعض الإختلافات و هي:

a-يمكنك تجاهل وضع رمز السطر الجديد *\n*, لأن الدالة *puts* تضع كل نص في سطر.
b-الدالة *puts* لا يمكنها التعامل مع متغيرات من نوع أرقام أو أحرف أي أنها لا تدعم رموز مثل *%d,%f,%c* .

مثال على الدالة *puts*:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  puts("Nom:ahmed/mohamed");
  puts("prenom:by elmoute2eli9");
  system("pause");
  return 0;
}
```

أما عن الدالة *gets* فتقوم بإدخال النصوص فقط أي أنه لا يمكن التعامل معها بالأحرف و الأرقام و سنتطرق إليها في لاحقا لأننا لم نتعامل مع السلاسل الحرفية بعد.

3-الدالتان *getche* و *getch*:

هاتان الدالتان تابعتان للمكتبة *conio.h* ,الدالة الأولى تستقبل حرف من المستخدم و تقوم بتخزينه في متغير حرفي و لكنها لا تنتظر حتى يضغط المستخدم على *enter* !!.

مثال:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
{
  char ch;
  printf("tapez un caracter:");
  ch=getche();
  printf("\nle caracter que vous etes taper est :%c\n",ch);
  system("pause");
  return 0;
}
```

أما بالنسبة للدالة الثانية فهي مشابهة جدا للدالة الأولى و الفرق الوحيد بينهما هو أن التالية لا تُظهر الحرف المدخل على عكس الأولى.

مثال:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
{
  char ch;
  printf("tapez un caracter:");
  ch=getch();
  printf("\nle caracter que vous etes taper est :%c\n",ch);
  system("pause");
  return 0;
}
```

4-الدوال الرياضية و دوال التعامل مع الحروف:

مكتبة الدوال الرياضية

المكتبة math.h في لغة السي تُقابلها المكتبة cmath في لغة السي++

هاتان المكتبتان تشتركان في دوال كثيرة منها :

الدالة	الوصف
sin(x)	الجيب
cos(x)	جيب تمام
tan(x)	الظل
asin(x)	معكوس الجيب
acos(x)	معكوس الجيب تمام
atan(x)	معكوس الظل
ceil(x)	تقريب x لأصغر قيمة صحيحة أكبر من x
floor(x)	تقريب x لأكبر قيمة صحيحة أصغر من x
fabs(x)	القيمة المطلقة
log10(x)	اللوغاريتم ذو الأساس 10
log(x)	اللوغاريتم الطبيعي
exp(x)	رفع x للأساس e
sqrt(x)	الجذر التربيعي
pow(x,y)	x^y
ldexp(x,y)	$x*2^y$

مكتبة التعامل مع الحروف

المكتبة ctype.h موجودة في لغة السي و يُمكن استخدامها في لغة السي++

الدالة	الوصف
isalnum(x)	إذا كانت القيمة المُعطاة للدالة تُقابل حرفا أو رمزا أو رقما في جدول آسكي فستعيد الدالة 1 و إلا فستعيد 0
isalpha(x)	إذا كانت القيمة المُعطاة للدالة تُقابل حرفا أبجديا في جدول آسكي فستعيد الدالة 1 و إلا فستعيد الدالة 0
isdigit(x)	إذا كانت القيمة المُعطاة لهذه الدالة لا تُقابل رقما في جدول آسكي فستعيد الدالة 0 و إلا فستعيد عدد يختلف عن الصفر
isgraph(x)	إذا كانت القيمة المعطاة لهذه الدالة تقابل رمزا مرئيا في جدول آسكي فستعيد الدالة 0 و إلا فستعيد قيمة تختلف عن الصفر
islower(x)	إذا كانت القيمة المعطاة لهذه الدالة تقابل حرفا صغيرا في جدوا آسكي فستعيد نتيجة تختلف عن الصفر و إلا فستعيد 0
isupper(x)	إذا كانت القيمة المعطاة لهذه الدالة تقابل حرفا كبيرا في جدوا آسكي فستعيد نتيجة

كيفية إنشاء الملفات الرأسية

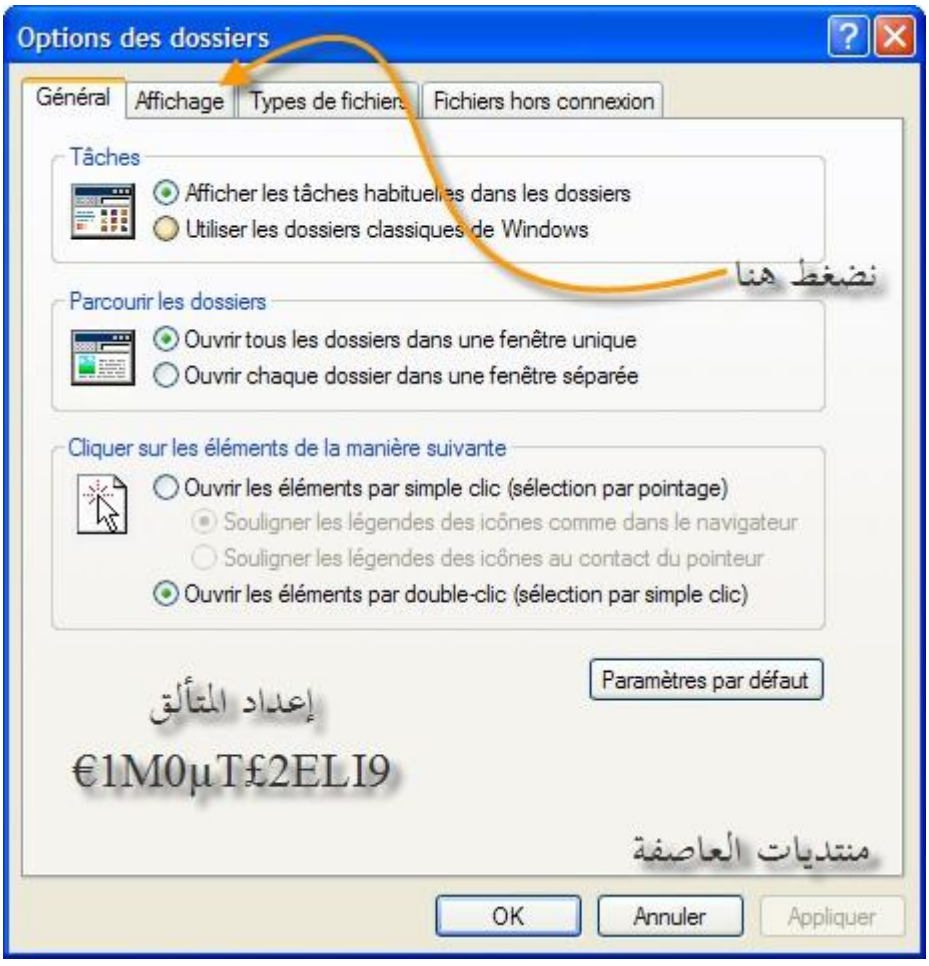
تعريف : الملف الرأسي هو ملف نصي بسيط ينتهي بالإمتداد ***h*** و يحتوي على دوال و ثوابت.

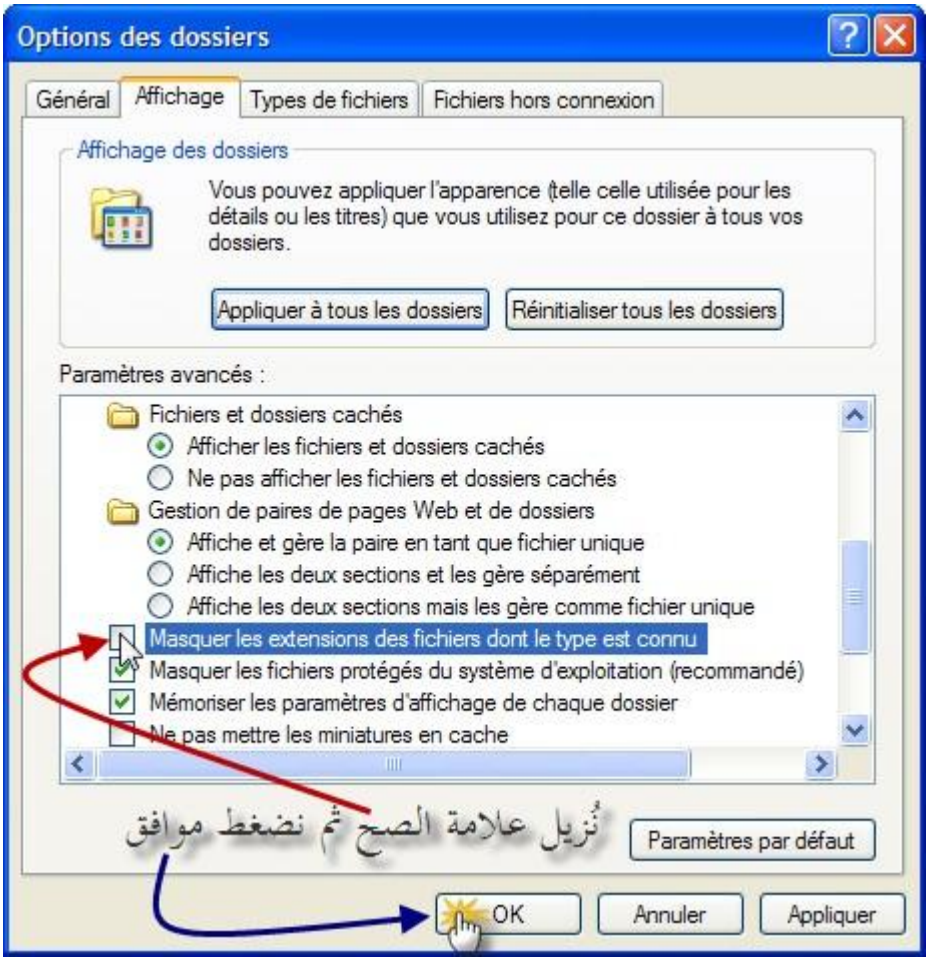
ما هي فائدة الملفات الرأسية : تُستعمل الملفات الرأسية أو الـ ***header file*** عند كتابة برامج كبيرة .. فمثلا : إذا قمت بكتابة الكثير من الدوال في برنامج معين فيستحسن أن تقوم بكتابة ملف رأسي تضع به جميع الدوال التي تود استخدامها في برنامجك و هكذا تكون لديك مكتبة كبيرة خاصة بك.

طريقة إنشاء الملفات الرأسية:

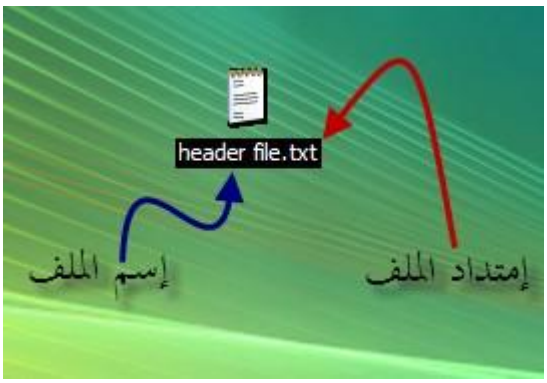
أولا و قبل كل شيء يجب عليك إظهار امتداد الملفات لكي تتمكن من تغيير الإمتداد,إليك الطريقة :


Poste de travail
Fichier
Edition
Affichage
Favoris
Outils
?
Précédente
Adresse
Poste de travail
OK
Connecter un lecteur réseau...
Déconnecter un lecteur réseau...
Synchroniser...
Options des dossiers...
نذهب إلى poste de travail
ثم نضغط هنا
إعداد المتألق
€1M0µT£2ELI9


Options des dossiers
Général
Affichage
Types de fichiers
Fichiers hors connexion
Tâches
Afficher les tâches habituelles dans les dossiers
Utiliser les dossiers classiques de Windows
نضغط هنا
Parcourir les dossiers
Ouvrir tous les dossiers dans une fenêtre unique
Ouvrir chaque dossier dans une fenêtre séparée
Cliquer sur les éléments de la manière suivante
Ouvrir les éléments par simple clic (sélection par pointage)
Souligner les légendes des icônes comme dans le navigateur
Souligner les légendes des icônes au contact du pointeur
Ouvrir les éléments par double-clic (sélection par simple clic)
Paramètres par défaut
إعداد المتألق
€1M0µT£2ELI9
منتديات العاصفة
OK
Annuler
Appliquer



الآن قم بإنشاء مستند جديد و ليكن اسمه *header_file* :



قم بتغيير الإمتداد من *txt* إلى *h* :

الآن أصبح الملف الرأسي جاهز للكتابة .. اكتب فيه الدالة التالية:

```
float abs(float x)
{
return x>0 ?x:-x;
}
```

قم بتخزينه تحت مجلد وليكن اسمه *header file*, الآن بقي شيء واحد وهو كتابة البرنامج المصدر .. قم بإنشاء ملف مصدر و قم بتخزينه في المجلد السابق كما في الصورة:

قم بكتابة ما يلي في الملف المصدر :

```c
#include<stdio.h>
#include<stdlib.h>
#include"header_file.h"
int main()
{
  float y;
  printf("y=");
  scanf("%f",&y);
  printf("abs(%f)= %f\n",y,abs(y));
  system("pause");
  return 0;
}
```

في السطر الثالث من البرنامج قمنا بإضافة الملف الرأسي *header_file.h* ولكن بطريقة مختلفة عن الطريقة المعتادة حيث قمنا بوضع اسم الملف الرأسي بين علامتي تنصيص " "

والسبب في ذلك هو وجود الملف الرأسي في نفس المكان الذي يوجد به الملف المصدر أما إذا أردنا كتابة :

```c
#include<header_file.h>
```

فيجب علينا وضع الملف الرأسي في نفس المكان الذي توجد به المكتبات التابعة للمترجم أي في مجلد *include* ,وفي *DEV* فإن المجلد عادة ما يأخذ المسار التالي :

```
C:\Dev-Cpp\include
```

قلنا سابقا أن الملف الرأسي يجب أن يكون في نفس المجلد الذي يوجد فيه الملف المصدر لكن ماذا لو كان الملف الرأسي في مجلد و الملف المصدر في مجلد آخر ؟؟؟ هنا سنقوم بإتباع نفس الخطوات السابقة مع كتابة المسار الذي يوجد فيه الملف الرأسي عند تضمينه هكذا :

```c
#include<stdio.h>
#include<stdlib.h>
#include"C:\Dev-Cpp\include\c++\3.4.2\mingw32\bits\header_file.h"//مسار الملف
int main()
{
  float y;
  printf("y=");
  scanf("%f",&y);
  printf("abs(%f)= %f\n",y,abs(y));
  system("pause");
  return 0;
}
```

ملاحظات:

1-الدالة الرئيسية *main* تُكتب كالتالي:

```
int main()
```

أي أن الدالة الرئيسية تعيد قيمة من النوع *int* لذلك نكتب في نهاية البرنامج :

```
return 0;
```

ويمكننا كتابة :

```
return 1;
```

أو :

```
return -2;
```

أو أي عدد آخر .. المهم يكون عدد صحيح ,لكن ماذا لو كتبنا :

```
return 7.09;
```

البرنامج سيعمل بدون خطأ !!! والسبب في ذلك هو أنه سيتم أخذ القيمة الصحيحة للعدد ,أي حذف ما بعد الفاصلة .. و لكن المترجم سيظهر تحذير بالأسفل يفيد بأن نوع المخرجات لا يتطابق مع نوع القيمة المعادة.

2-الكتابة الصحيحة للدالة الرئيسية هي :

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

ولكن يمكننا كتابة العبارة التالية اختصارا :

```
int main()
```

3-يمكن إدخال أكثر من حرف في الدالة *getchar* حيث سيتم أخذ الحرف الأول من الحروف المدخلة ,أما الدالتان *getche* و *getch* فلا يمكنك إدخال أكثر من حرف.

4-يمكن كتابة أرقام في بداية الملف الرأسي كما يمكن استعمال مؤثرات الجمع و الطرح أما بالنسبة للرموز التي لا يمكن استعمالها في اسم الملف الرأسي هي :

```
% # | < > " * / & : ? \
```

و أقصى حد يمكن إعطاءه لإسم الملف الرأسي هو 256 رمز.

مجالات الرؤية

تنقسم المتغيرات حسب مجال الرؤية إلى نوعين:

1-المتغيرات الخارجية (العامة):

يتم الإعلان عنها خارج جميع الدوال, ويمكن لأي دالة في البرنامج استعمال هذا النوع المتغيرات.

مثال:

```
#include<iostream>
using namespace std;
int c;
void fonc(int c)
{
 cout<<"c="<<2*c<<endl;
}
int main()
{
 c=3;
 fonc(3);
 system("pause");
 return 0;
}
```

لاحظ أننا استخدمنا المتغير *c* في الدالة *fonc* و في الدالة الرئيسية أيضا, لأن جميع دوال البرنامج يمكنها رؤية (استخدام) المتغيرات العامة.

2-المتغيرات المحلية (الخاصة):

ينم الإعلان عنها داخل دالة معينة ,و لا يمكن استخدامها إلا داخل الدالة أو الإطار التي أُعلنت بداخله.

```
#include<iostream>
using namespace std;
void fonc()
{
 int a;
}
int main()
{
 int b;
 return 0;
}
```

المتغير *a* هو متغير محلي (خاص بالدالة *fonc*) و لا يمكن استخدامه في الدالة الرئيسية ,كما أن المتغير *b* متغير محلي أيضا ,و لا يمكن استخدامه في الدالة *fonc* بل هو خاص بالدالة الرئيسية الـ *main* (التي أُعلن بداخلها).

عمر المتغيرات:

بالنسبة للمتغيرات العامة فتنتهي أعمارها عند انتهاء تنفيذ البرنامج بينما ينتهي عمر المتغيرات الخاصة عند الإنتهاء من تنفيذ الدالة الموجودة بها.

3-المتغيرات الساكنة:

طريقة الإعلان عن متغير من النوع الساكن تكون بكتابة كلمة *static* و نوع المتغير ثم إسمه ,هكذا:

```
static int nombre;
```

في هذا المثال قمنا بالإعلان عن متغير صحيح باسم *nombre* و أضفنا إليه خاصية السكون.

كلمه *static* تعني السكون, و عند إضافتها إلى أي متغير .. فهذا يعني أنك قد أضفت له سمتين رئيسيتين هما:

a-أصبح عمر المتغير مثل عمر المتغيرات العامة التي لا تُمسح من الذاكرة إلا إذا طلبت ذلك برمجيا, أو بانتهاء تنفيذ البرنامج .. لأن المتغيرات العامة مُعرضة إلى أن تتم قراءتها في أي وقت ومن أي دالة.

b-لاُ يمكن رؤيتها إلا داخل الإطار التي أُعلنت بداخله.

لاحظ أن هذا النوع من المتغيرات (أقصد النوع *static*) يأخذ مزايا النوعين السابقين .. فهي مثل المتغيرات الخاصة لأن هناك دالة وحيدة تستطيع رؤيتها وهي الدالة التي تم الإعلان عن المتغير بداخلها ,و من ناحية أُخرى فالمتغيرات الساكنة مثل المتغيرات العامة لأنها لا تنتهي أو تُمسح من الذاكرة عندما ينتهي تنفيذ الدالة

التابعة لها .. بل تظل مخزنة في الذاكرة (جاهزة للإستدعاء) حتى ينتهي تنفيذ البرنامج.

مثال:

لنفرض أننا نريد كتابة برنامج يقوم بعد الطلبة الذين يدخلون في الفصل أو يخرجون منه, ليعطي في النهاية عدد الطلاب المتواجدين في الفصل.

سنقوم بكتابة البرنامج باستخدام الدوال, يعني دالة تقوم بزيادة عدد التلاميذ بـ 1 عندما يدخل تلميذ في الفصل, و دالة أُخرى تنقص عدد التلاميذ بـ 1 عندما يخرج تلميذ من الفصل, لاحظ أن هاتان الدالتان تشتركان في متغير واحد وهو عدد التلاميذ .. بعبارة أُخرى:هناك متغير في البرنامج يجب أن تراه كلتا الدالتين.

```
#include<iostream>
using namespace std;
static int Count;
void CounterInc()
{
    Count++;
    cout<<Count<<"\n" ;
}
void CounterDec()
{
    Count--;
    cout<<Count<<"\n" ;
}
int main()
{
     CounterInc();//نستدعي هذه الدالة عندما يدخل تلميذ في الفصل
     CounterDec();//نستدعي هذه الدالة عندما يخرج تلميذ من الفصل
     system("pause");
    return 0;
}
```

نفس الفكرة السابقة يمكننا تعميمها لتشمل المتواجدين حاليا في المنتدى ولكن كيف نعرف بدخول أو خروج العضو ؟

العملية بسيطة, سيكون لدينا داله تزيد عدد الحضور بـ 1 ودالة أخرى للإنقاص, عندما تتم عملية تسجيل الدخول لعضو ما سنستدعي الدالة التي تقوم بزيادة عدد الحضور بفرد.

في حالة الخروج يحدث هذا بطريقتين: إما الضغط على تسجيل الخروج وفي تلك اللحظة نستدعي دالة الإنقاص, الحالة الأخرى هي إغلاق الصفحة أو انقطاع النت و في هذه الحالة سيظهر مفهوم الجلسة التي تمتد لفترة معينه قد يختارها مدير المنتدى 10 دقائق مثلا ,بعد انتهاء الجلسة الخاصة بالعضو الذي أغلق الصفحة نستدعي آليا دالة الإنقاص.

الآن ماذا لو استبدلنا السطر الثالث من البرنامج بهذا السطر:

```
int Count;
```

لاحظ أن هذا المتغير هو متغير عام ,قمنا الآن بحذف كلمة *static* إلا انه لو تم تنفيذ البرنامج مره أخرى سيعطي نفس النتيجة !!! وهذا لا يعني إلا شيء واحد أن المتغيرات العامة هي بطبيعتها متغيرات ثابتة أو ضمنيا كما يقال.

إذا ما فائدة المتغيرات الساكنة؟

في بداية كلامي قلت أن المتغير الساكن لا يُرى إلا في المجال الذي أعلن فيه وقد يكون هذا مهم إذا أردت أن لا يتم قراءة هذا المتغير إلا داخل دالة واحدة ولا يتأثر بأي تغير من أي مكان آخر في الكود الخاص بك.

يميل أكثر المبتدئين إلى استخدام المتغيرات العامة فهي تبعدك كثيراً عن مشاكل القيم المعادة وتبادل المعلومات بين الدوال .. إلا أن هناك مشاكل كثيرة لها مثلا:أنها ستبقى مرئية في المناطق التي لا تريدها, وفي هذه الحالة فإنه يفضل استخدام المتغيرات الساكنة.

التمارين

التمرين 1:

باستخدام الدوال اكتب برنامج يقوم بإظهار الأعداد من 100 و حتى 0 (شكل تنازلي) ثم يظهر الأعداد من 0 و حتى 100 (شكل تصاعدي),لا يُسمح باستخدام الحلقات.

التمرين2:

باستخدام الدوال اكتب برنامج يطلب من المستخدم إدخال عددين صحيحين ثم يُظهر نتيجة الجداء,لا يُسمح باستخدام الأوامر الشرطية أو الحلقات و لا يسمح أيضا باستخدام الرمز *.

التمرين 3:

باستخدام الدوال قم بكتابة برنامج يطلب من المستخدم إدخال عدد صحيح ثم يظهر له مضروب العدد المدخل,لا يُسمح باستخدام الحلقات.

التمرين 4:

باستخدام الدوال اكتب برنامج يطلب من المستخدم إدخال عدد حقيقي x و آخر صحيح y ثم يظهر له نتيجة x أُسْ y,لا يُسمح باستخدام الحلقات.

التمرين 5:

باستخدام الدوال اكتب برنامج يقوم بحساب قيمة الحد رقم n من متتالية "فيبوناسي",بدون استخدام الحلقات.

حل التمارين:

حل التمرين 1:

```
#include<iostream>
using namespace std ;
void Increment(int x)
{
  if(x>=0){cout<<x<<endl;Increment(--x);}
}
void Decrement(int y)
{
  if(y<=100){cout<<y<<endl;Decrement(++y);}
}
int main()
{
    cout<<endl<<"faire apparaître les nombres de façon croissante de 0 à 100 :"<<endl<<endl;
    Increment(100);
    cout<<endl<<"faire apparaître les nombres de façon décroissante de 0 à 100 :"<<endl<<endl;
    Decrement(0);
    system("PAUSE") ;
    return 0 ;
}
```

في هذا الكود قمنا بإنشاء دالتين *Increment* و *Decrement* الأولى مسئولة عن إظهار الأعداد بشكل تصاعدي أما الثانية فهي مسئولة عن إظهار الأعداد بشكل تنازلي.

بالنسبة للدالة الأولى :

المدخلات : عدد صحيح

المخرجات : لا شيء (*void*) و السبب في ذلك أنها لا تُعيد قيمة و إنما تُظهرها.

الأوامر التابعة لها :

إذا كان الوسيط المدخل موجب أو معدوم فقم بإظهاره ثم نفذ الأمر *Increment(--x)*

وهذا يعني : قم بإظهار العدد *x-1* و بعد ذلك سينفذ الأمر *Increment(--(x-1))* أي قم بإظهار العدد *x-2* ثم ينفذ الأمر *Increment(--(x-2))* ليظهر العدد *x-2* وبعد ذلك يُظهر العدد *x-3* و هكذا .. حتى تصل قيمة *x* إلى الصفر .. عندها يكون الشرط غير محقق و يتم الخروج من الدالة.

نفس الشيء بالنسبة للدالة الثانية *Decrement*.

ملاحظة : هذا النوع من الدوال يُسمى بالدوال التراجعية وهو كثير الإستخدام .. و نظرا لأهميته فقد وضعت مُعظم التمارين حول هذا النمط من الدوال وذلك لكي يتم فهمه بشكل أفضل .

حل التمرين 2:

```
#include <iostream>
using namespace std;
int Multiplication (int x,int y);
int main()
{
    cout<<"entrez le premier nombre:";
    int a;
    cin>>a;
    cout<<"entrez le deuxieme nombre:";
    int b;
    cin>>b;
    cout<<a<<"*"<<b<<"="<<Multiplication(a,b)<<endl;
    system("pause");
    return 0;
}
int Multiplication(int x,int y)
{
    if(y==1)
       return x ;
    else
       return x+Multiplication(x,--y);
}
```

في السطر السادس عشر من البرنامج قمنا بتعريف الدالة *Multiplication* بأنها تستقبل وسيطين صحيحين و تُعيد عددا صحيحا,أما بالنسبة لأوامرها فهي كالتالي:

إذا كان العدد الثاني يساوي 1 فإن نتيجة الجداء تُساوي 1 أما إذا كان العدد الثاني يختلف عن الواحد فإن نتيجة الجداء تُساوي *x+Multiplication(x,--y)* و السبب في ذلك هو أن جداء عددين x و y يساوي جمع x (y مرة).

مثال :

لنأخذ $x=-4$ و $y=3$, الآن سيظهر البرنامج النتيجة *-12* مرورا بالخطوات التالية:

```
(-4)*3=(-4)+Multiplication(-4,2)
(-4)*3=(-4)+(-4)+Multiplication(-4,1)
(-4)*3=(-4)+(-4)+(-4)
```

لاحظ أن قيمة x ثابتة و قيمة y تتناقص بدرجة واحدة في كل مرة.

حل التمرين 3:

```c
#include<stdio.h>
#include<stdlib.h>
long factorielle(int n);
int main()
{
int n;
long fact;
printf("n=");
scanf("%d",&n);
if(n<0)
printf("erreur! le nombre n doit être supérieur ou egal zero.\n\n");
else
{
fact=factorielle(n);
printf("%d!=%ld\n\n",n,fact);
}
system("pause");
return 0;
}
long factorielle(int n)
{
if(n==0) return 1;
else return n*factorielle(n-1);
}
```

في السطر العشرين من البرنامج قمنا بتعريف دالة المضروب هكذا:

إسم الدالة : *factorielle*

مدخلات الدالة : عدد صحيح.

مخرجات الدالة : عدد صحيح طويل.

الأوامر التابعة للدالة :

إذا كان الوسيط المدخل للدالة يساوي 0 أو 1 فإن المضروب=1.

أما إذا كان العكس (الوسيط يختلف عن 0 و 1) فإن المضروب يساوي قيمة العدد *n* ضرب مضروب(*n-1*) و مضروب *n-1* يساوي *n-1* ضرب مضروب(*n-2*) وهكذا حتى يكون *n* يساوي صفر وهذا مطابق تماما لقاعدة المضروب إذا أن:

المضروب=(العدد)*(العدد-1)*(العدد-2)*(العدد-3)*...*1

مثال :

لنأخذ *n*=3 عندما نقوم بتنفيذ الكود السابق سيُظهر لنا النتيجة 6 مرورا بالخطوات التالية:

مضروب(3)=3*مضروب(2).

مضروب(2)=2*مضروب(1).

مضروب(1)=1*مضروب(0).

مضروب(0)=1.

وبالتعويض نجد :

مضروب(3) =3*2*1=6

حل التمرين 4:

```
#include<stdio.h>
#include<stdlib.h>
float puissance(float x,int n)
{
if(n==0) return 1;
if(n>0) return x*puissance(x,n-1);
if(n<0) return (1/x)*puissance(x,n+1);
}
int main()
{
float x;
int n;
printf("x=");
scanf("%f",&x);
printf("n=");
scanf("%d",&n);
if(x==0)
{
if(n<=0)
printf("erreur\n\n");
else
printf("0^%d=0\n\n",n);
}
if(x!=0)
printf("%f^%d=%f\n\n",x,n,puissance(x,n));
system("pause");
return 0;
}
```

في هذا البرنامج قمنا بتعريف الدالة *puissance* بأنها تأخذ عدد حقيقي *x* و آخر صحيح *n* أما بالنسبة لأوامرها فهي كالتالي:

*إذا كان الأُسْ يساوي 0 فإن النتيجة هي 1 ,وهذا شيء بديهي لأن أي عدد أُسْ صفر يساوي 1.

*إذا كان الأُسْ أكبر تماما من الصفر فإن النتيجة تساوي قيمة *x* ضرب *puissance(x,n-1)* و قيمة *puissance(x,n-1)* تُساوي قيمة *x* ضرب *puissance(x,n-2)* وهكذا دواليك .. يعني نحافظ على قيمة *x* و ننقص قيمة الأُس بواحد حتى نصل إلى الصفر.

مثال :لنأخذ $x=2$ و $n=3$,البرنامج سيُظهر النتيجة 8 مرورا بالخطوات التالية :

```
2^3=2*puissance(2,2)
2*puissance(2,2)=2*puissance(2,1)
2*puissance(2,1)=2*puissance(2,0)
2*puissance(2,0)=1
```

بالتعويض نجد :

```
2^3=2*2*2*1=8
```

نفس الشيء بالنسبة للحالة الثالثة.

حل التمرين5:

```
#include<iostream>
using namespace std;
long suite(int n);
int main()
{
int n;
long Un;
cout<<"entrez la valeur de n:";
cin>>n;
if(n>=0)
{
Un=suite(n);
cout<<"U"<<n<<"="<<Un<<endl;
}
else
cout<<"erreur! le nombre n doit être supérieur ou egale zero (n>=0)"<<endl<<endl;
system("pause");
return 0;
}
long suite(int n)
{
if(n==0||n==1||n==2) return 1;
if(n>=3) return (suite(n-1)+suite(n-2)+suite(n-3));
}
```

رأينا سابقا أن :

$$Un=Un-1+Un-2\ ,\ U0=U1=1 \quad \text{حيث} \quad n>=0$$

أي أن كل حد من حدود متتالية "فيبوناسي" عبارة عن جمع الحدين اللذين قبله ,هذا في حالة رقم الحد أكبر أو يساوي 3 أما إذا كان يساوي 0 أو 1 أو 2 فإن قيمة الحد تساوي 1 و هذا بالضبط ما كتبناه في تعريف الدالة.