موقع فيجوال سي للعرب

سلسلة الشامل لعلوم الحاسوب و الإلكترونيات

كتاب لغة C الشامل

تأليف: خليل أونيس

ANSI C

تعلم بأبسط طريقة

كتاب تطبيقي مئة بالمئة

بسم الله الرحمن الرحيم

محتويات الكتاب بنظرة سريعة

جدول المحتويات

حول الكاتب و الكتاب

الحمد الله رب العالمين و الصلاة و السلام على سيد المرسلين نبينا محمد *صلى الله عليه و سلم*، و على آله و صحبه أجمعين...أما بعد:

بختصار، الإسم خليل أونيس، مولود بالجزائر عام *01/01/1989*، أدرس في أحد المدارس الخاصة بعلوم الحاسوب و الإلكترونيات تدعى بمدرسة الحرية للإعلام الآلي و اللغات، رقم الهاتف النقال الخاص بي: من خارج الجزائر *(0021364576618)*، من داخل الجزائر *(064576618)*، العنوان: نهج الاخوة العمراني رقم *10* - باتنة، البريد الإلكتروني: *khalil_ounis@yahoo.com*.

يعتبر الكتاب مرجع شامل للغة C، و هو مقسم إلى ثلاثة فصول متكاملة:

- في الفصل الأول مفاهيم و مبادئ أساسية في لغة C: الإدخال و الإخراج، التعليقات و المؤثرات، القرارات و عناصر لغة C، مع ملخص للفصل الأول.

- الفصل الثاني مكمل للفصل الأول في كل من القرار *Switch*، حلقات التكرار، المصفوفات و المؤشرات، الدوال، الملفات الرأسية، الإدخال و الإخراج في الملفات و التراكيب، و أخيرا ملخص للفصل الثاني.

- الفصل الثالث مكمل للكتاب، مع إضافة أهم ثوابت، مختصرات و دوال المكتبة القياسية للغة C.

في حالة أي إستفسار يمكنك الإتصال بموقع فيجوال سي للعرب على الرابط: *www.vc4arab.com*، أو مراسلتي على البريد الإلكتروني.

خليل أونيس

المقدمة

في أيام بداية الحاسوب كانت البرمجة تتم على لغة بدائية منخفضة المستوى *low-level language* تدعى بلغة الآلة *Machine language*، حيث كانت تفهمها الآلة مباشرة، و يتم البرمجة عليها بأوامر تمثل بخيوط طويلة مكونة من الواحد و الصفر (الصفر تعني *off* و هي تساوي صفر فولت، و الواحد يعني *on* و هو يساوي خمسة فولت) أي بما يسمى بالنظام الثنائي، و كانت البرمجة عليها صعبة و معقدة حتى تم تطوير لغة التجميع *assembly language*، و هي من اللغات المنخفضة المستوى *low-level languages* أيضا، حيث كانت سهلة كثيرا بالنسبة للغة الآلة، فبدل استعمال سلاسل من الصفر و الواحد نستعمل أوامر ذات أسماء واضحة مثل *ADD* و *MOV*، و معا ذلك لا تزال البرمجة صعبة.

مع مرور الوقت تم تطوير لغات برمجة أخرى مثل *COBOL*، *BASIC* و *C*، و كان التعامل معها بالكلمات و النصوص، و هذا ما سهل على المبرمجين في تطوير برامجهم.

تعتبر لغة *C* من أقوى لغات البرمجة رغما أنها من أقدمها، تم تطويرها في السبعينات من طرف كين تومسن *Ken Thompson* و دنيس ريتشي *Dennis Ritchie* في مختبرات *Bell*.

لغة *C* من اللغات المنخفضة المستوى *low-level languages* حيث أنها قريبة من الأجهزة و شبيها بلغة التجميع *assembly language* في عملها، و لكن البعض يعتبرها لغة متوسطة المستوى *mid-level language* لأنها لغة تحاكي لغة الإنسان بعض الشيء، و هي لغة مستقلة عن البنية الصلبة للحاسوب.

قام كين تومسن و دنيس ريتشي بتطوير لغة *C* لبرمجة نظام يونيكس *Unix*، حيث ركزا مطوري هذه اللغة على أن تكون لغتهم سهلة الاستعمال حيث يمكن كتابة برامج كبيرة مع قلة الأخطاء حتى يستطيع المبرمجين تطوير برامجهم بسرعة.

في عام *1972* تم إطلاق لغة *C* بشكل رسمي، و سميت بلغة *C* لأنها كانت مشتقة من لغة الـ *B* (و كانت لغة الـ *B* مشتقة من لغة *BCPL* و التي قام بتطويرها مارتن ريتشاردز *Martin Richards* في عام *1967* و هي مختصرة من *Basic Combined Programming Language*، حيث كان الفرق بين اللغتين هو نوع البيانات) التي قام بتطويرها كين تومسن في عام *1969* حيث أخذ الحرف *B* من اسم المختبر *Bell* الذي يعمل به كين تومسن، و الذي يلي الحرف *B* في الأبجدية هو *C*، و ذلك هو سبب تسميتها بلغة *C*.

في عام *1978* قام دنيس ريتشي و براين كارنيغان *Brian Kernighan* بـــتأليف أول كتاب لهذه اللغة و سمي بـــ *The C Programming Language* و الذي يعتبر المرجع الأساسي لهذه اللغة، و كان الكتاب معروف بنسخة *K&R C* (*Kernighan & Ritchie C*) و السبب في تسميته بـــ *K&R C* هو كثرة اســـتعمال لغـــة *C* بشكل كبير و الذي أدى إلى تطوير مكتبات و دوال في نسخ مختلفة من لغة *C* حتى أصبح كل من تلـــك النسخ غير متوافقة مع بعضها و كادت أن تكون غير متشابها، و هذا ما أدى إلى تعريف نسخة قياسية للغة *C*.

في عام *1989* تم إطلاق النسخة القياسية للغة *C* و سميت بـــ *ANSI C* و هـــي مختـــصرة مـــن *American National Standards Institute* أي اللجنة الوطنية الأميركية للمعاير، و بتعاون بـــين اللجنـــة الوطنيـــة الأميركية للمعاير و المنظمة العالمية للمعاير تم إطلاق لغة *C* القياسية في مختلف أنحاء العالم و سميت بـــ *ISO C* و هي مختصرة من*International Organization for Standardization*.

(في عام *1988* قام دنيس ريتشي و براين كارنيغان بكتابة الطبعة الثانية من كتـــاب *The C Programming Language* لنسخة القياسية للغة *C* أي *ANSI C*.)

1

الفصل الأول – أساسيات في لغة C

1.1 الأدوات اللازمة

1.2 البدء مع لغة *C*

1.3 المتغيرات و الثوابت *Variables and Constants*

1.4 التعليقات *Comments*

1.5 الإدخال باستخدام *scanf*

1.6 المؤثرات *Operators*

1.7 القرارات *if, else, else...if*

1.8 عناصر لغة *C*

1.9 ملخص للفصل الأول،مع إضافات

1.1 الأدوات اللازمة

قبل البدء في البرمجة على لغة C يلزمك أدوات و التي تتمثل في كل من محرر نصوص*texts editor*، و مترجم *compiler*، و المربط *linker*، المترجمات الحديثة توفر جميع الأدوات اللازمة.
هذا شرح مختصر لكل من تلك الأدوات:

1.1.1 **محرر نصوص** texts editor:

محرر نصوص هو الذي نقوم بالكتابة فيه مصادر شفرة برامجنا و حفظها على صيغة *c.*، لا يهم نوع المحرر، المهم أن يكون محرر بسيط مثل *KWrite* في أنظمة *Linux* أو *Notepad* في أنظمة *Windows*.

و لا يمكن استعمال محررات نصوص متقدمة مثل *Word* في *Windows* أو *Kword* في *Linux*.

1.1.2 **مترجم** compiler:

تقوم المترجمات بترجمة أو تحويل الملفات المصدرية إلى لغة منخفضة المستوى إن لم تكون هناك أخطاء في قواعد اللغة، يمكن أن تترجم إلى لغة التجميع *Language Assembly* أو إلى لغة الآلة *Machine Language* مباشرة، حيث بعد الترجمة يتم إنشاء ملفات بصيغة *obj*..

يوجد العديد من المترجمات في أغلب الأنظمة، مثلا في أنظمة *Windows* يوجد المترجم المعروف و المستعمل بكثرة *Visual C++* حيث يقوم بترجمة كلا اللغتين *C* و *C++*، و هو مقدم من طرف شركة *MicroSoft*، و يوجد كذلك المترجم *Dev-C++* و المقدم من شركة *Bloodshed*، و مترجمات أخرى مثل *Turbo C*، *Quick C*، *Pelles C*، ....

بالنسبة للمترجم *Visual C++* فهو غير مجاني.

المترجم *Dev-C++* من المترجمات المجانية و يمكن تحميله من الرابط التالي:
http://www.bloodshed.net/devcpp.html

المترجم *Turbo C* أيضا من المترجمات المجانية، و هو من أقدمها، حيث يمكن تحميله من الرابط التالي:
http://www.pitt.edu/~stephenp/misc/downloadTC.html

المترجم *Pelles C* أيضا من المترجمات المجانية و يعتبر من أفضلها و يمكن تحميله من الرابط التالي:
http://www.smorgasbordet.com/pellesc/download.htm

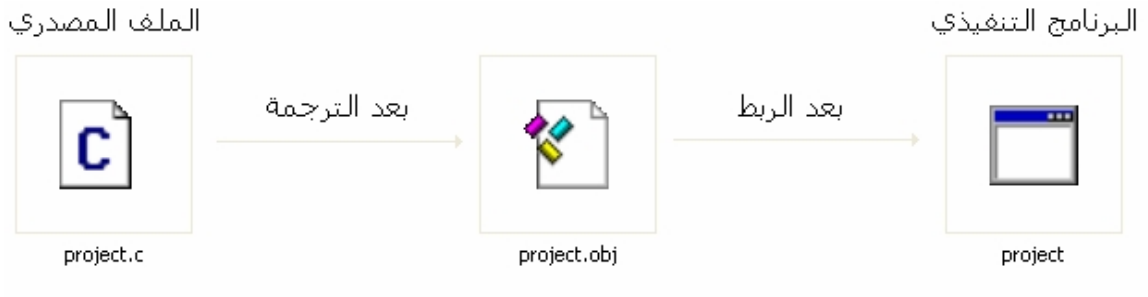
أما في أنظمة *Unix* و *Linux*، فلا تحتاج إلى مترجمات لأنها مدمجة مع أي نسخة من نسخ *Unix* و *Linux*، كل ما تحتاجه هو محرر نصوص. و هذا لا يعني أنه لا يوجد مترجمات لتلك الأنظمة، بل يوجد و ربما عددها أكثر من التي هي موجودة على نظام *Windows*.

أدخل على الرابط التالي حيث توجد مترجمات مجانية عديدة في كل من أنظمة *Windows* و أنظمة *Linux*:
http://www.thefreecountry.com/compilers/cpp.shtml

1.1.3 المربط linker:

يقوم المربط بجمع الملفات ذات الصيغة *obj*. ثم يعطينا البرامج التنفيذية و التي تكون غالبا بامتداد *exe*.، أو ملفات مكتبات الربط الديناميكية و التي تكون بإمتداد *dll*.

و هذه صورة توضح طريقة عمل الترجمة و الربط على نظام *Windows*:



1.2 البدء مع لغة C

قم بتحميل و تثبيت أحد المترجمات السابقة و قم بتشغيلها كأي برنامج، ثم قم بإنشاء مشروع جديد للغة C في بيئة الـــ *Console* مع إنشاء ملف نصي جديد و الحرص على أن يتم حفظه بامتداد *.c*، يمكن كتابـــة *main.c* كاسم للملف النصي الرئيسي و الذي سنقوم بالكتابة عليه البرنامج الأول و هو:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("Hello, World!");
6:}
```

ملاحظة: قم بكتابة البرنامج بدون الترقيمات و النقاط التي بعدها، أي نكتب البرنامج على الشكل التالي:

```
#include<stdio.h>

main()
{
      printf("Hello, World!");
}
```

هذا من أبسط البرنامج التي يمكن كتابتها، يقوم هذا البرنامج عند ترجمته و تنفيذه بطباعة الجملـــة *Hello, World* على الشاشة في بيئة الـــ *Console*.

السطر الأول من البرنامج به الشفرة *(Code)* `#include<stdio.h>` و هي مقسمة إلى قسمين هما:

الأول هو `#include<>`، و غالبا ما تكون الكلمة `#include` أزرقة اللون.

و القسم الثاني هو ما بين الرمزين أكبر من و أصغر من < >، حيث يوجد الملف `stdio.h`، في هذا القـــسم نقوم بكتابة أسماء لملفات تسمى بالملفات الرأسية، و هي عديدة و كل ملف منها له مجاله الخـــاص حيـــث يحتوي على ثوابت و دوال تسهل علينا البرمجة.

الملف الرأسي `stdio.h` مختصر من *Standard Input Output*، أما `h.` فهو امتداد الملف الرأسي و هو مختصر من *Header File*.

فائدة الكلمة `#include` هو ضم الملف الرأسي الموجود بين الرمزين أكـــبر مـــن و أصـــغر مـــن < > إلى مشروعنا. يوجد العديد من الملفات الرأسية، سنتطرق إليها فيما بعد.

في السطر الثالث يوجد اسم دالة و هي main() و هي الدالة الرئيسية لأي مشروع و لا يمكن الاستغناء عنها، و لا يمكن التغير في اسمها إلا في حالات. و هذه الدالة يبدأ البرنامج بالتنفيذ بشكل مترتب، أما القوسين بعد اسم الدالة فهما اللذان يبينان على أنها دالة و ليست متغير أو ثابت.

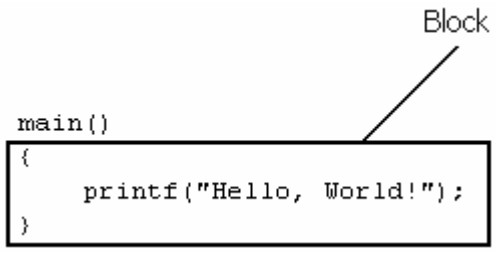
في السطر الرابع توجد الحاضنة { و التي تعني بداية الدالة main.

في السطر الخامس توجد الكلمة printf و هي عبارة عن دالة تم الإعلان عنها في الملف الرأسي stdio.h، و هي مختصرة من *print format* أي صيغة الطبع، و هي تقوم بطبع (إخراج) ما هو بداخل أقواس الدالة إلى الشاشة، و في مثالنا هذا يوجد النص *Hello, World !* و هي الجملة التي سيتم طبعها على الشاشة، و تكون الجمل دائما داخل اقتباسيين " "، و في نهاية السطر نكتب الفاصلة المنقوطة و هي تعني نهاية السطر الخاص بالدالة.

تستعمل الدالة printf بصفة عامة في عرض أو إخراج معلومات إلى أداة الإخراج و هي الشاشة *Screen* الخاصة بالحاسوب.

و أخيرا السطر السادس حيث موجود به الحاضنة { و التي تعني نهاية الدالة الرئيسية main.

و تسمى حاضنة البداية { و حاضنة النهاية { و ما بينهما بالـ *block*، صورة توضيحية:



و يمكن كتابة البرامج السابقة بطرق مختلفة، حيث يمكن تقسيم الجملة *Hello, World !* إلى قسمين مثل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("Hello, ");
6:    printf("World!");
7:}
```

و هنا سيتم طبع الجملة كاملة في سطر واحد، و تقسيمها لا يعني أن كل كلمة في سطر.

و يمكن أيضا كتابة الجملة حرفيا، كل حرف بدالة من printf.

و توجد طريقة لا يمكن استعمالها و هي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("Hello,
6:          World!");
7:}
```

عند ترجمة هذا المثال سينبهك المترجم عن وجود أخطاء، منها نسيان قوس النهاية لدالة printf، و لتفادي هذه الأخطاء نقوم بوضع *anti-slash* في نهاية السطر الأول من الدالة printf، و تصبح الدالة كالآتي:

```
      printf("Hello, \
            World!");
```

هنا سيعمل البرنامج بدون أخطاء.

في المثال السابق إن كتبنا السطر الأول (الذي يتمثل في ضم الملف الرأسي stdio.h) في نهاية البرنامج فإن المترجم لن يجد الدالة printf، و ستنجم أخطاء عن ذلك، لذا يجب دائما أن يكون ضم الملفات الرأسية قبل الدوال المراد استعمالها و يستحسن دائما أن يتم ضم الملفات في بداية كل مشروع.

و يمكن كتابة الكلمة *Hello,* في سطر و الكلمة *World !* في سطر آخر و ذلك بإضافة الرمز \n بين الكلمتين، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("Hello, \nWorld!");
6:}
```

أو كتابة كل من الكلمات في دالة مثل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("Hello, ");
6:    printf("\n");
7:    printf("World!");
8:}
```

عند ترجمة البرنامج و تنفيذه فلن تجد الرمز \n و ستجد أن كل من الكلمتين في سطر، يتم استبدال الرمز \n بسطر جديد حيث لا يتم طباعة الرمز، و الحرف ***n*** يعني ***New line***.

يمكن كتابة المثال الأول في ثلاثة أسطر كما في يلي:

```
1:#include<stdio.h>
2:
3:main(){printf("Hello, World!");}
```

تم جمع جميع الأسطر في السطر الثالث، و البرنامج يعمل مثل السابق بدون أخطاء، حيث ستلاحظ أنه يمكن استعمال الحاضنة } (بداية الدالة) و الحاضنة { (نهاية الدالة) في نفس السطر، و يمكن استعمال أكثر من ذلك مثل:

```
1:#include<stdio.h>
2:
3:main(){
4:    printf("Hello, World!");}
```

و طرق أخرى، و لكن يجب أن تكون الأوامر و الوظائف و الدوال المراد إستعمالها داخل الحاضنتين { } لدالة الرئيسية. و مثل هذه الطرق لا يفضل استعمالها و خاصة إذا كان البرنامج كبير.

و توجد طريقة لا يمكن إستعمالها و هي موضحة في المثال التالي:

```
1:#include<stdio.h> main(){
2:    printf("Hello, World!");}
```

إذا ترجمة هذا المثال فسينبهك المترجم عن وجود خطأ لأن الكلمة #include تتطلب سطرا كاملا لها، و لا يمكن استعمال ما ليس له علاقة معها.

تدعى الكلمة #include بالتوجيه ***directive*** أو قبل المعالج ***preprocessor*** و سميت بقبل المعالج لأنه يتم تنفيذها قبل الترجمة، و هي تقوم بضم محتويات الملف الرأسي المطلوب إلى المشروع، حيث يحتوي ذلك الملف الرأسي على مجموعة من ثوابت، بنيات و دوال تساعدنا في برامجنا.

توجد الكثير من التوجيهات ***directive*** و يمكن تميزها بالرمز #، سنعرفها في الدروس القادمة.

و يمكن أيضا وضع ***block*** داخل الدالة الرئيسية main، مثال:

```
1:#include<stdio.h>
2:
```

```
3:main()
4:{
5:    printf("Hello, World!\n");
6:    {
7:          printf("Hello, World!\n");
8:    }
9:    printf("Hello, World!\n");
10:}
```

و يتم التعامل معها كالتعامل مع *block* الدالة الرئيسية، و يمكن إنشاء أكثر من *block* داخل الدالة الرئيسية، أو استعمال *block* داخل *block* آخر.

1.2.1 التعامل مع الأعداد:

التعامل مع الأعداد هو طباعة الأعداد على الشاشة و استعمال مؤثرات عليها مثل الجمع، الطرح، القسمة و الضرب، و سنتعامل مع الأعداد باستخدام الدالة `printf`، و ربما تقول أن الأمر سهل فقط نقوم بكتابة الأعداد التي نريدها داخل الاقتباسات في الدالة `printf`، صحيح يمكن استعمال تلك الطريقة و لكن المترجم هنا سيتعامل مع الأعداد على أنها نص ثابت و ليست أعداد، و أيضا في هذه الحالة لا يمكن استعمال عمليات تقوم بالجمع، الطرح، القسمة و الضرب.

في لغة C لكل نوع من الأعداد رمز لتعامل معه، مثلا الأعداد الصحيحة يتم التعامل معها بالاستعمال الرمز `%d` أو `%i`، الرمز الأول مختصر من *Decimal* و الرمز الثاني مختصر من *Integer*، هذا بالنسبة للأعداد الصحيح، أما الأعداد الحقيقية فيتم التعامل معها باستخدام الرمز `%f`، و الحرف *f* مختصر من *float*، و أيضا توجد رموز أخرى خاصة بالتعامل مع كل من الحروف و النصوص، سنتطرق إليها فيما بعد.

نذهب إلى التعامل مع الأعداد الصحيحة، لكتابة عدد من نوع الأعداد الصحيحة نكتب كما يلي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%d", 20);
6:}
```

هنا وضعنا رمز الأعداد الصحيحة داخل الدالة `printf` و بين الاقتباسيين، و بعد الاقتباسيين نقوم بكتابة العدد المراد طبعه، و الحرص على أن يكون بين الاقتباسيين و العدد فاصلة، و بهذه الطريقة يمكن استعمال عمليات مثل الجمع مثلا و ذلك بإضافة مؤثر الجمع مع العدد المراد الجمع معه مثل ما هو موضح في المثال التالي:

```
1:#include<stdio.h>
```

```
2:
3:main()
4:{
5:    printf("%d", 20+5);
6:}
```

و يمكن استعمال باقي المؤثرات مثل الطرح، القسمة و الضرب بنفس الطريقة.

و يمكن إظهار أكثر من رقم و ذلك بزيادة الرمز %d مع فصله من الرمز السابق حتى تكون الأرقام واضحة مثل ما يلي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%d %d", 20+5, 87);
6:}
```

كلما نظيف رمز الأعداد الصحيحة نقوم بكتابة الرقم الإضافي بعد الرقم السابق و نفصلهما بفاصلة، يمكن استعمال أكثر من عددين و بطريقة منظمة فمثلا إذا أردنا أن نقوم بكتابة عملية الجمع فسنكتب كما يلي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%d + %d = %d\n", 20, 5, 20+5);
6:}
```

و نفس الطريقة مع الأعداد الحقيقية فقط نستعمل الرمز %f بدل الرمز %d.

و هذا مثال لكيفية استعمالها:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%f + %f = %f\n", 1.0, 2.14, 1.0+2.14);
6:}
```

ملاحظة:

في المثال السابقة وضعنا النقطة في مكان الفاصلة، هكذا كي يميز المترجم على أنها قيم للأعداد حقيقيـــة أي أنها أعداد لها فواصل، أما إذا وضعت الفاصلة في مكان النقطة فسيعتبرها المترجم منفصلة عن الأخـــرى و هكذا ستنجم أخطاء كثيرة.

أما رموز طبع الأحرف و النصوص فطريقة استعمالها مثل الطرق السابقة فقط بدل الرمزين %d و %f نضع %c للأحرف، حيث حرف *c* مختصر من *character* و نضع الرمز %s للنصوص، و الحرف s مختصر من *String* أي سلسلة حروف.

بالنسبة للحروف فهذا مثال يوضح طريقة استعمالها:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%c", "a");
6:}
```

و يمكن استعمال هذا المثال أيضا:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%c", 'a');
6:}
```

هنا سيطبع البرنامج الحرف *a*، و في حالة أردنا طبع نص نكتب كما يلي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%s\n", "Hello, World!");
6:}
```

و يمكن أيضا كتابة كل كلمة أو حرف في إقتباسين مثل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%s", "Hello, " "\n" "World");
6:}
```

و توجد رموز أخرى منها من هي خاصة بأرقام النظام السداسي عشر و التي يكون التعامل معها باستخدام الرمز %x أو %X حيث تبدأ من 0x أو 0X، مثلا الرقم 0x000F، و الرمز %o لأعداد النظام الثماني، و رموز أخرى سنعرفها في الدروس القادمة.

1.2.2 الأخطاء المحتملة:

1- في لغة C المترجمات تفرق بين الحروف الكبيرة و الحروف الصغيرة، مثلا الدالة main لا يمكن كتابتها Main أو MAIN.

2- لا يمكن استعمال الدالة printf أو دوال أخرى خرج الدالة الرئيسية main، مثلا لا يمكن كتابة:

```
1:#include<stdio.h>
2:
3:printf("Hello, World!\n");
4:
5:main()
6:{
7:
8:}
```

إلا في حالة استعمال دوال بها دوال أخرى ثم ربطتها بالدالة الرئيسية، سنتعرف على ذلك في الدروس القادمة.

3- كثيرا ما يتم نسيان الفاصلة المنقوطة، و إن تم نسيانها لا يمكن إنشاء الملف التنفيذي للبرنامج حتى يتم تصحيح الخطأ.

4- لا يمكن استعمال الفاصلة المنقوطة في نهاية سطر الدالة الرئيسية ()main، و سبب ذلك هو عندما يتم الإعلان عن دالة و إعطاءها أوامر لا يجب أن نكتب الفاصلة المنقوطة، ليست مثل دوال معرفة سابقا مثل الدالة printf.

5- في دالة الطبع printf، إن كتبنا النص المراد طبعه بدون رموز الاقتباس " " فإن البرنامج لن يعمل، و سينبهك المترجم عن وجود خطأ.

1.2.3 تمارين:

1- أكتب برنامج يقوم بطباعة الجملة *Hello, World !* مرتين، الأولى في سطر و الثانية في سطر آخر.
2- أكتب برنامج يقوم بطباعة الجملة *Hello, World !*، كل حرف في سطر.
3- هل يمكن تغير اسم الدالة الرئيسية main إلى اسم من اختيارنا؟
4- هل يمكن الاستغناء عن الدالة الرئيسية main؟
5- هل يمكن استعمال الدالة الرئيسية أكثر من مرة؟
6- أكتب برنامج يقوم بطبع نتيجة طرح العدد *2* من *3.5*.

7- أكتب برنامج يقوم بطباعة الكلمة *Hello* باستخدام رموز الأحرف (%c).

8- أكتب برنامج يقوم بكتابة نصيين باستعمال الرمز الخاص بطبع النصوص مرتين (%s%s).

1.3 المتغيرات و الثوابت *Variables and Constants*

المتغيرات هي حجز موقع في ذاكرة الحاسوب لتخزين فيها قيم كي نستطيع استرجاعها في وقت آخـــر، و لكي نفهم المتغيرات جيدا يستحسن فهم طريقة عمل ذاكرة الحاسوب الـــ ***RAM***.

الـــ ***RAM*** هي ذاكرة إليكترونية، مختصر من ***Random Access Memory*** أي ذاكرة الـــدخول(الوصـــول) العشوائية حيث تستعمل عند تنفيذ البرامج، هي ليست ذاكرة دائما مثل القرص الصلب، تفقد جميع بياناتها عند إيقاف تشغيل الجهاز.

في لغة *C* يوجد عدت أنواع من المتغيرات و الثوابت، منها متغيرات خاص بالأعداد الـــصحيح و أخـــرى بالأعداد الحقيقية و أخرى بالأحرف و ...، و دائما نقوم بالإعلان عن المتغيرات و الثوابت قبل استعمالها.

سنتحدث الآن عن المتغيرات.
طريقة الإعلان عن متغير هي كتابة نوع المتغير ثم اسم المتغير ثم القيمة التي ســـيحتويها المـــتغير، صـــورة توضيحية:

```
Variable_Type Variable_Name = Variable_Value
```

نوع المتغير — اسم المتغير — قيمة المتغير

1.3.1 نوع المتغير Variable Type:

توجد عدة أنواع و هي:

1.3.1.1 متغير الأعداد الصحيحة `int`:

نقوم بالإعلان عن متغير من نوع الأعداد الصحيحة بكتابة الكلمة `int` في مكان ***Variable_Type***، حيـــث يحتاج المتغير من نوع ***Integer*** إلى مساحة قدرها *2* بايت و التي تساوي *16* بت و تساوي *65536* احتمال، أي أن أقصى قيمة يمكن أن يحملها المتغير هي *65535*، ابتداء من الصفر، أو ابتداء من *32,768-* إلى *32,767* في حالة ضم الأعداد السالبة.

و يمكن أن يكون حجمها *4* بايت أي تساوي *32* بت، حيث أقصى قيمة يمكن أن تحملها هي *4294967296*، ابتداء من الصفر(في حالة أن المتغير لا يحتوي إلا على قيم موجبة).

مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int Variable_Name = 0;
6:}
```

1.3.1.2 **متغير الأعداد الحقيقية `float`**:

الأعداد الحقيقية هي الأعداد التي لها فواصل، و يتم الإعلان عنها باستخدام الكلمة `float`، حجمها *4* بايت، حيث تبدأ من $1.2^{E}-38$ إلى $3.4^{E}+38$.

مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    float Variable_Name = 0.0;
6:}
```

1.3.1.3 **متغير الأعداد الحقيقية `double`**:

`double` هي ضعف `float`، و يتم الإعلان عنها باستخدام الكلمة `double`، حيث حجمها *8* بايت و تبدأ من $2.3^{E}-308$ إلى $1.7^{E}+308$.

مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    double Variable_Name = 0.0;
6:}
```

1.3.1.4 **متغير الأعداد الصحيحة `short`**:

هو أيضا من متغيرات الأعداد الصحيحة حيث نقوم بالإعلان عنه بكتابة الكلمة `short` في مكان *Variable_Type*، حجمه *2* بايت و التي تساوي *16* بت و تساوي *65536* احتمال، أي أن أقصى قيمة يمكن أن يحملها المتغير هي *65535* ابتداء من الصفر، أو ابتداء من *32,768-* إلى *32,767* في حالة ضم الأعداد السالبة.

مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    short Variable_Name = 0;
6:}
```

1.3.1.5 **متغير الأعداد الصحيحة long**:

هو أيضا من متغيرات الأعداد الصحيحة حيث نقوم بالإعلان عنه بكتابة الكلمة long في مكان *Variable_Type*، حجمه *4* بايت أي يساوي *32* بت، حيث أقصى قيمة يمكن أن يحملها هي *4294967296*، ابتداء من الصفر(في حالة أن المتغير لا يحتوي إلا على قيم موجبة).

مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    long Variable_Name = 0;
6:}
```

1.3.1.5 **متغير الرموز char**:

من أصغر المتغيرات، يتم الإعلان عنه بكتابة الكلمة char في مكان *Variable_Type*، حجمه *1* بايت أي *8* بت حيث يحمل *256* احتمال ابتداء من *0* إلى *255* أو من *128-* إلى *127*، حيث كل رقم يمثل برمز في جدول *ASCII*.

مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char Variable_Name = 'A';
6:}
```

1.3.2 **اسم المتغير** Variable Name:

لاسم المتغير حدود لا يجب تجاوزها و هي:

- أن لا يتجاوز اسم المتغير أكثر من *31* حرف.
- أن لا يبدأ اسم المتغير بأرقام.
- أن لا يكون اسم المتغير يحتوي على مؤثرات مثل الجمع، الطرح، ....

- أن لا يكون اسم المتغير يحتوي على رموز مثل % و # و } و...(باستثناء الرمز _).
- أن لا يكون اسم المتغير مستعمل سابقا في دالة أو متغير آخر.
- أن لا يكون اسم المتغير من أسماء الكلمات المحجوزة.

1.3.3 قيمة المتغير Variable Value:

يجب مراعاة قيمة المتغير حسب نوعه، فمثلا لا يمكن أن نعطي للمتغير int قيمة عدد حقيقي float.

1.3.4 أمثلة حول المتغيرات:

سأقدم أمثلة مختلفة حول طريقة استعمال المتغيرات، و نبدأ بمتغيرات أعداد صحيحة حيث نقوم بإعلان عن متغير باسم Var و به القيمة 5، ثم نقوم بطباعة القيمة الموجودة في المتغير Var على الشاشة، المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int Var = 5;
6:
7:    printf("%d\n", Var);
8:}
```

في هذا المثال، في السطر الخامس تم الإعلان عن متغير باسم Var و من نوع int (عدد صحيح) و به القيمة 5، و في السطر السابع استعملنا الدالة printf لطباعة قيمة المتغير Var، و توجد طرق أخرى لإعطاء للمتغيرات قيم، سأعطي طريقتين:

الأولى هي الإعلان عن المتغير في سطر ثم إعطاءه قيمة في سطر آخر مثل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int Var;
6:    Var = 5;
7:
8:    printf("%d\n", Var);
9:}
```

و الطريقة الثانية هي الإعلان عن متغيرين، الأول به القيمة 5 و الثاني به قيمة المتغير الأول، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int Var_1 = 5;
6:    int Var_2 = Var_1;
```

```
7:
8:    printf("%d\n", Var_1);
9:}
```

مثال آخر:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int Num1, Num2, Num3;
6:    Num1 = 5;
7:    Num2 = 7;
8:    Num3 = Num1 + Num2;
9:
10:   printf("%d + %d = %d\n", Num1, Num2, Num3);
11:}
```

في السطر الخامس تم الإعلان عن ثلاثة متغيرات في نفس السطر حيث نقوم بفصل بين اسم متغير و آخر بفاصلة، و هنا ستكون جميع المتغيرات من نوع أعداد صحيحة (int)، و في السطر السادس و السطر السابع أعطينا للمتغير Num1 القيمة 5 و المتغير Num2 القيمة 7، و في السطر الثامن أعطينا للمتغير Num3 نتيجة الجمع بين المتغير Num1 و المتغير Num2، و أخيرا السطر العاشر و الذي يقوم بطباعة نتائج البرنامج.

و نفس الطرق السابقة يمكن إستعمالها مع متغيرات من نوع float و short و long، أما المتغير char فطريقة استعماله ليست مختلفة كثير، حيث يمكننا أيضا أن نعطيه عدد بدل الحرف حيث عند طباعته لا يطبع عددا، إنما الحرف الذي يحمل ذلك الرقم في جدول *ASCII*، و لكي تفهم طريقة استعمال متغيرات من نوع char إليك المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char ch = 'a';
6:
7:    printf("%c\n", ch);
8:}
```

في السطر الخامس أعطين للمتغير ch الحرف الذي *a* مع مراعاة أن يكون داخل ' '، أما إذا أردنا أن نعطيه عددا يطبع لنا الحرف *a* فهو الرقم *97* في جدول *ASCII*، و سيصبح المثال السابقة كما يلي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char ch = 97;
6:
7:    printf("%c\n", ch);
```

```
8:}
```

و كما قلنا سابقا أن أقصى قيمة يمكن أن يحملها متغير من نوع char هي 255 ابتداء من الصفر، و كل رقم يمثل برمز.

1.3.5 الأعداد الموجبة و الأعداد السالبة:

في حالة أنك أردت استعمال أعداد موجبة و أعداد سالبة لمتغير فتوجد طريقتين لذلك الأولى تكون افتراضية عند كتابة نوع و اسم المتغير، أي أنه عندما نقوم بالإعلان عن متغير مثلا int Num يمكنه أن يحمل كلا من الأعداد السالبة و الموجبة، أو يمكن كتابة signed Num و هي مثل int Num من ناحية الحجم و الاستعمال، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    signed Num = 5;
6:
7:    printf("%d\n", Num);
8:}
```

هذا بالنسبة للمتغيرات التي تحتوي على أعداد موجبة و أعداد سالبة، أما في حالة أردنا أعداد موجبة فقط فسنستعمل الكلمة unsigned قبل اسم المتغير، مثلما هو موضح في المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    unsigned Num = 5;
6:
7:    printf("%u\n", Num);
8:}
```

تم الإعلان عن المتغير الذي لا يحتوي على أعداد سالبة في السطر الخامس و كما أن طريقة إستعمالها كاستعمال متغير طبيعي، و هنا نتعرف على الرمز الجديد %u الذي يخبر المترجم أنا القيمة التي سيتم طبعها من نوع unisgned أي أعداد بدون إشارة.

ملاحظة: عند الإعلان عن متغير طبيعي مثلا char فإنه سيحتوي 256 احتمال كما قلنا سابقا،حيث يمكن أن يبدأ من 0 إلى 255 في حالة عدم الحاجة إلى أعداد سالبة، و يمكن أيضا أن يبدأ من -128 إلى 127 في حالة استعمال أعداد سالبة، و لكي تعرف السبب في ذلك إليك الشرح:

لكل متغير حجمه، مثلا متغير من نوع char حجمه *1* بايت أما int فحجمه هو 2بايـت، و كما نعرف أن *1* بايت يساوي *8* بت، أي أن متغير من نوع char حجمه *8* بت، و *1* بت يساوي إما *1* أو *0* حيث هـــذا يعني أنا *1* بت لديه احتمالين *(0* أو *1)*، أما *8* بت فلديها *256* احتمال تبدأ من *0* و تنتهي عنـــد *255* إن لم تكن تحتوي على أعداد سالبة، أما في حالة أردنا أعداد سالب فسيتم سحب *1* بت مـــن ،*8* أي سيـــصبح حجم متغير من نوع char *7* بت أما البت الثامن سنتركه للإشارة، و سيحمل إما الإشارة + أو الإشارة - ، و *7* بت تساوي *128* احتمال.

و هذا جدول به القيم المحتملة لكل نوع من المتغيرات:

النوع	الحجم	القيم
char و signed char	*1* بايت	من *128-* إلى *127*
char و char unsigned	*1* بايت	من *0* إلى *255*
int و signed int	*2* بايت	من *32,768-* إلى *32,767*
int و unsigned int	*2* بايت	من *0* إلى *65535*
short و signed short	*2* بايت	من *32,768-* إلى *32,767*
short و unsigned short	*2* بايت	من *0* إلى *65535*
long و signed long	*4* بايت	من *2,147,483,648-* إلى *2,147,483,647*
long و unsigned long	*4* بايت	من *0* إلى *4,294,967,295*

الثوابت، هي عكس المتغيرات، يمكن أن تكون عدد، حرف، أو نص، حيث لا يمكن الـــتغير قيمتـــها أي تصبح قابلة للقراءة فقط،، سأعطي مثال، حيث هذا مثال به متغير من نوع أعداد صحيحة، و نعطيه القيمة *5* ثم نقوم بطبع المتغير على الشاشة ثم نقوم بتحديث المتغير إلى القيمة *8* ثم نعيد طباعة قيمة المتغير و ها هو المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num;
6:
7:    num = 5;
8:
9:    printf("%d\n", num);
10:
11:   num = 8;
12:
13:   printf("%d\n", num);
14:}
```

هنا سيتم تغير قيمة المتغير num من 5 إلى 8، و هذه الطريقة صحيح. و الآن سنكتب نفس البرنامج السابق مع إضافة بسيطة، المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    const int num;
6:
7:    num = 5;
8:
9:    printf("%d\n", num);
10:
11:   num = 8;
12:
13:   printf("%d\n", num);
14:}
```

الإضافة موجودة في السطر الخامس، و هي إضافة الكلمة const إلى المتغير int num و التي تعني أن المتغير num ثابت، و هنا البرنامج لن يعمل، و السبب هو أنه لا يمكن تحديث القيمة الأولى لثوابت، و في مثالنا السابقة لا توجد قيمة للمتغير num بعدما اتم الإعلان عنه، و يجب دائما إعطاء قيم لثوابت مباشرة بعد الإعلان عنها و إلا ستكون عبارة عن ثوابت ذات أعداد عشوائية ثابتة لا يمكن التحديث فيها، و هذا المثال السابق بعد التصحيح:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    const int num = 5;
6:
7:    printf("%d\n", num);
8:}
```

و يمكن أيضا كتابة الكلمة const مباشرة بعد نوع المتغير مثل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int const num = 5;
6:
7:    printf("%d\n", num);
8:}
```

و يمكن استعمال نفس أنواع المتغيرات على الثوابت.

و يوجد نوع آخر من الثوابت، و هي باستعمال الكلمة #define و طريقة إستعمالها موضحة كما في الصورة التالية:

```
#define Constant_Name Constant_Value
```

حيث سيصبح المثال السابقة كما يلي:

```
1:#include<stdio.h>
2:
3:#define num 5
4:
5:main()
6:{
7:    printf("%d\n", num);
8:}
```

تم الإعلان عن الثابت num في السطر الثالث، ثم طباعته في السطر السابع.

و كما قلت سابقا، أن *preprocessor* تبدأ بالرمز #، و هذا يعني أن #define من الـــ *preprocessors* و يمكن تسميتها أيضا بـــ *directive*، و جميع *directives* لا تنتهي بفواصل منقوطة.

1.3.6 الأخطاء المحتملة:

1- لا يمكن وضع قيمة أكثر من قيمة المتغير القصوى.

2- لا يمكن الإعلان عن المتغيرات إلا في بداية كل *block*.

3- في حالة لم يتم تعين قيمة لمتغير، و أردت طبع قيمة ذلك المتغير على الشاشة فستأتي أعداد عشوائية تختلف من جهاز لآخر.

4- يجب الحرص على أن بين نوع المتغير و اسم المتغير مسافة واحدة على الأقل.

5- لا يمكن كتابة الكلمة const بعد اسم المتغير أو بعد الإعلان عنه.

6- لا يمكن تغير اسم متغير أو ثابت.

7- لا يمكن الإعلان عن متغيرين بنفس الاسم.

1.3.7 تمارين:

1- أكتب برنامج يقوم بطباعة العديدين *3.14* و *15*، باستخدام الرموز الخاصة بطباعتها.
2- ماذا يحدث إن أعطين لمتغير من نوع `int` قيمة أكثر من *65535* ؟
3- أكتب برنامج يقوم بطباعة الحرف *A* بدل الرقم *65*، بدون استخدام متغيرات أو ثوابت.
4- أكتب برنامج يقوم بطباعة الحرف *A* بدل الرقم *65*، باستخدام `char`.
5- أكتب برنامج به ثلاثة متغيرات، المتغير الأول به القيمة *18* و الثاني *89*، أما الثالث يكون الناتج الحاصل بين المتغير الأول و الثاني في كل من الجمع، الطرح، القسمة و الضرب.
6- أكتب برنامج به ثلاثة من `#define` *preprocessor* حيث الثالثة هي نتيجة الجمع بين الأولى و الثاني، الأولى بها القيمة *5* و الثاني بها القيمة *15*.

1.4 التعليقات *comments*

التعليقات هي مجموعة من سلاسل نصية نستعملها لتوضيح أوامر في مصادر برامجنا، و يمكن أن تحتوي تلك النصوص على أرقام، أحرف، أو رموز يقوم المترجم بتجاهلها.

1.4.1 فائدة التعليقات:

فائدة التعليقات يمكنك أن تلاحظها في الكثير من الأمثل المفتوحة المصدر الموجودة على الإنترنت، مثلا تجد مثال لبرنامج ما كبير و غير واضح، و هنا يلجئ المبرمج إلى استعمال التعليقات لجعلها أكثر وضوح.

1.4.2 أنواع التعليقات:

يوجد نوعين من التعليقات هما:

1.4.2.1 التعليقات بالنصوص الطويلة:

التعليقات بالنصوص الطويلة هي نصوص بها أكثر من سطر، و طريقة استخدامها هي تحديد بداية التعليق و التي تبدأ بـــ /* و نضع */ في نهاية التعليق، مثال:

```
1:/*
2:My First Program:
3:Hello, World!
4:*/
5:
6:#include<stdio.h> /* Standart Input Output Header File*/
7:
8:main()                  /*main function*/
9:{/*Start of main function*/
10:   printf("Hello, World!");/*to print Hello, World!*/
11:}/*End of main function*/
```

1.4.2.2 التعليقات بالأسطر:

التعليقات بالأسطر هي تجاهل سطر تعليقي يبدأ بـــ //، مثال:

```
1://My First Program:
2://Hello, World!
3:
4:#include<stdio.h> //Standart Input Output Header File
5:
6:main()                  //main function
7:{//Start of main function
8:    printf("Hello, World!");//to print Hello, World!
9:}//End of main function
```

و هذه الطريقة ليست من طرق لغة C القياسية في التعليقات، و لكن الكثير من المترجمات تدعمها.

1.4.3 الأخطاء المحتملة:

1- في التعليقات بالنصوص الطويلة إن لم يتم تحديد نهاية التعليق فإن كل ما هو بعد بداية التعليق يعتبر تعليق، و هذا مثال توضيحي:

```
1:/*comment
2:#include<stdio.h>
3:
4:main()
5:{
6:    printf("Hello, World!");
7:}
```

هنا البرنامج كله عبارة عن تعليق، أي أن المشروع فارغ.

2- في التعليقات السطرية يجب الانتباه إلى ما نضعه تعليق فمثلا:

```
1://#include<stdio.h>
2:
3:main()
4:{
5:    printf("Hello, World!");
6:}
```

هنا سيخبرك المترجم على أن الدالة `printf` غير معرفة سابقا، و هذا الخطأ سببه هو جعل الكلمة المحجوزة `#include` الخاصة بضم الملف الرأسي `stdio.h` عبارة عن تعليق، و هنا سيتم تجاهل ضم الملف الرأسي `stdio.h`.

1.4.4 تمارين:

لا توجد تمارين في هذا الدرس.

1.5 الإدخال باستخدام *scanf*

في الدروس السابقة لم ندرس إلا الإخراج باستخدام الدالة printf، الآن سنعرف كيفية الإدخال بواسطة الدالة scanf، التشابه كبير جدا بين الداليتين printf و scanf، فقط الأولى خاصة بالإخراج و الثانية خاصة بالإدخال.

تستعمل الدالة scanf لقراءة أو استقبال المعلومات من أداة الإدخال لوحة المفاتيح ***keyboard***.

الآن سنقوم بكتابة برنامج يطلب من المستخدم إدخال قيمة ثم نعطيه القيمة التي قام بإدخالها:

```
1:/*الإدخال*/
2:#include<stdio.h>
3:
4:main()
5:{
6:    int usr_val;           /*هنا سنضع القيمة التي سيدخلها المستخدم*/
7:
8:    printf("Enter a value: ");/* هنا نطلب من المستخدم إدخال قيمة */
9:    scanf("%d", &usr_val); /* يقوم الجهاز بانتظار المستخدم لإدخال قيمة */
10:   printf("Your value is: %d\n", usr_val);/*  هنا نطبع القيمة التي أدخلها
المستخدم*/
11:}
```

البرنامج موضح بالتعليقات، في السطر التاسع قمنا باستخدام الدالة scanf و داخلها ستجد وسيطين، الأول نقوم فيه بتحديد نوع القيمة التي سيدخلها المستخدم حيث هنا وضعنا الرمز %d و الذي درسنه سابقا في الدالة printf حيث قلنا أنها خاص بالأعداد الصحيحة، و في الوسيط الثاني يوجد &usr_val، و الرمز & يعني وضع القيمة التي أدخلها المستخدم في عنوان المتغير usr_val، و ستفهم السبب إضافة الرمز & في الدروس القادمة. ، إلا هنا تكون قيمة usr_val قد أصبحت القيمة التي أدخلها المستخدم ثم نقوم بطباعتها على الشاشة.

المثال السابق خاص بإدخال الأعداد الصحيحة، أما بالنسبة لباقي أنواع المتغيرات فسنستعمل نفس الطريقة فقط نقوم بتغير الرمز %d إلى نوع المتغير الذي نريد إستقباله، فمثلا إذا أردنا من المستخدم أن يقوم بإدخال رمز بدل رقم نضع الرمز %c في الدالة scanf، و هذا مثال يوضح ذلك:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char usr_char;
6:
7:    printf("Enter a character: ");
```

```
8:    scanf("%c", &usr_char);
9:    printf("Your character is: %c\n", usr_char);
10}
```

الآن سنقوم بكتابة برنامج يطلب من المستخدم إدخال قيمة، ثم يطلب منه إدخال قيمة ثانيـــة، و نعطيـــه النتائج بجميع المؤثرات الأساسية:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int val1, val2;
6:
7:    printf("1)Enter a value: ");
8:    scanf("%d", &val1);
9:
10:   printf("2)Enter a value: ");
11:   scanf("%d", &val2);
12:
13:   printf("%d + %d = %d\n", val1, val2, val1+val2);
14:   printf("%d - %d = %d\n", val1, val2, val1-val2);
15:   printf("%d * %d = %d\n", val1, val2, val1*val2);
16:   printf("%d / %d = %d\n", val1, val2, val1/val2);
17}
```

المثال واضح، قمنا بالإعلان عن متغيرين في السطر الخامس، ثم طلبنا من المستخدم إدخال قيمة في الـــسطر السابع و قمنا بأخذ القيمة في السطر الثامن، و بعدها طلبنا من المستخدم إدخال القيمة الثانيـــة ثم أخـــذنا القيمة الثانية في السطر الحادي عشر، ثم طبعنا النتائج في كل من السطر *13* و *14* و *15* و *16*.

و يمكن استعمال إدخال متعدد في الدالة scanf، و هذا المثال السابقة باستخدام الإدخال المتعدد:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int val1, val2;
6:
7:    printf("Enter two value: ");
8:    scanf("%d%d", &val1, &val2);
9:
10:   printf("%d + %d = %d\n", val1, val2, val1+val2);
11:   printf("%d - %d = %d\n", val1, val2, val1-val2);
12:   printf("%d * %d = %d\n", val1, val2, val1*val2);
13:   printf("%d / %d = %d\n", val1, val2, val1/val2);
14:}
```

التعدد هنا موجود في السطر الثامن، في الدالة scanf، و يمكن أن نزيد أكثر من ذلك، فقط نضيف الرمـــز الخاص بنوع المتغير ثم إضافة اسم المتغير في آخر الدالة.

1.5.1 الأخطاء المحتملة:

1- لا يمكن استخدام الوسيط الأول من الدالة scanf لطباعة(الإخراج)، الوسيط الأول من هـــذه الدالـــة خاص بنوع الرموز التي سيستقبلها البرنامج من المستخدم.

2- في حالة الاستغناء عن الرمز & فستكون النتائج غير صحيحة.

3- يجب الانتباه إلى عدد المتغيرات المراد فحصها، يجب أن يكون عدد رموز أنواع المتغيرات نفسه عـــدد المتغيرات المراد فحصها.

1.5.2 تمارين:

1- أكتب برنامج يطلب من المستخدم إدخال الحرف الأول و الأخير من اسمه ثم يقوم البرنامج بإخبار المستخدم أن إسم يبدأ بـــ "الحرف الأول الذي أدخله المستخدم" و ينتهي بالحرف "الحرف الأخير الذي أدخله المستخدم".

1.6 المؤثرات operators

للمؤثرات أنواع أهمهما ثلاثة و هي:

1.6.1 المؤثرات الحسابية (arithmetic operators):

هي المؤثرات الحسابية الأساسية و التي تتمثل في كل من الجمع(+)، الطرح(-)، القسمة(//) و الضرب(*)،و توجد مؤثرات أخرى خاص بلغة C و هي: الزيادة(++)، النقصان(--) و باقي القسمة(%).

بالنسبة للجمع، الطرح، القسمة و الضرب فقد أخذنا أمثلة عنها سابقا، سنأخذ أمثلة عن كل من مؤثرات الزيادة و النقصان و باقي القسمة:

1.6.1.1 مؤثر الزيادة increment (++):

تعتبر مؤثرات الزيادة و النقصان من أهم المؤثرات، تستعمل في الكثير من البرامج و خاصة في حلقات التكرار(سندرسها في الدروس القادمة).

مؤثر الزيادة يعني زيادة رقم واحد إلى المتغير الذي نريد الزيادة إليه، و هذا مثال يوضح ذلك:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int Inc;
6:    Inc = 0;
7:
8:    printf("Inc = %d\n", Inc);
9:
10:   Inc++;
11:
12:   printf("Inc = %d\n", Inc);
13:
14:   ++Inc;
15:
16:   printf("Inc = %d\n", Inc);
17:}
```

قمنا باستعمال مؤثر الزيادة في كلا من السطر العاشر و السطر الرابع عشر، و نتائج البرنامج تكون *0* ثم *1* ثم *2*.

و يمكن استعمال طرق أخرى مثل:

```
1:    printf("Inc = %d\n", Inc);
2:
```

```
3:    Inc = Inc+1;
4:
5:    printf("Inc = %d\n", Inc);
6:
7:    Inc += 1;
8:
9:    printf("Inc = %d\n", Inc);
```

الطريقة الأولى هي Inc = Inc+1 و التي موجود في السطر الثالث، هي مطابقة تماما لـــ Inc++ و لكـــن في السابقة يمكن أن نقوم بزيادة أكثر من *1* يعني إن كتبنا Inc = Inc+3 فسيتم الإضافة إلى المتغير Inc ثلاثـــة أرقام، كذلك الطريقة الثانية Inc += 1، هي مثل Inc = Inc+1 تمام.

و لكن يستحسن دائما استعمال ++ عند الزيادة بالواحد، و في حالة أن زيادة ستكون أكثر مـــن واحـــد فسنستعمل الطرق الأخرى.

و الفرق بين أن يكون المؤثرين ++ في بداية اسم المتغير أو نهايته هو موضح في المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int Inc;
6:    Inc = 0;
7:
8:    printf("Inc = %d\n", Inc);
9:    printf("Inc = %d\n", Inc++);
10:   printf("Inc = %d\n", Inc);
11:   printf("Inc = %d\n", ++Inc);
12:   printf("Inc = %d\n", Inc);
13:}
```

في السطر الثامن سيتم طباعة العدد *0*، و في السطر التاسع أيضا و معنا ذلك عند كتابة متغير قم المؤثر ++ يعني طباعته ثم تنفيذ مؤثر التزايد.

و في السطر العاشر سيتم طبع العدد *1* لأننا قمنا بالزيادة في السطر التاسع.

و في السطر الحادي عشر سيتم تنفذ مؤثر التزايد أولا يم طباعة النتيجة التي هي *2*، و كـــذلك في الـــسطر الثاني عشر سيتم طباعة العدد *2*.

1.6.1.2 **مؤثر النقصان** decrement *(--)*:

سنتعامل مع المؤثر النقصان مثلما تعاملنا مع مؤثر الزيادة فقط بدل ++ نضع -- و هذا المثـــال الـــسابق باستخدام مؤثر النقصان:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int Dec;
6:    Dec = 2;
7:
8:    printf("Dec = %d\n", Dec);
9:
10:   Dec--;
11:
12:   printf("Dec = %d\n", Dec);
13:
14:   --Dec;
15:
16:   printf("Dec = %d\n", Dec);
17:}
```

و يمكن أيضا استعمال نفس الطرق السابقة في النقصان:

```
1:    printf("Dec = %d\n", Dec);
2:
3:    Dec = Dec-1;
4:
5:    printf("Dec = %d\n", Dec);
6:
7:    Dec -= 1;
8:
9:    printf("Dec = %d\n", Dec);
```

1.6.1.3 مؤثر باقي القسمة (%):

أيضا يمكن اعتباره من المؤثرات المهمة، و طريقة استعماله سهل فمثلا لو أردنا أن نجد باقي القسمة بين العدد *5* و العدد *3* نكتب *5%3*، و هذا مثال توضيحي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%d\n", 5%3);
6:}
```

1.6.2 المؤثرات العلاقية (relational operators):

هي مؤثرات تعتمد على المقارنة بين قيمة و قيمة أخرى، حيث تكون النتيجة إما صحيحة (*true*) أو خاطئة (*false*)، و هذا مثال يوضح ذلك:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%d\n", 5<3);
6:    printf("%d\n", 5==3);
7:    printf("%d\n", 5>3);
```

```
8:}
```

هنا ستجد نتائج البرنامج:

0: لأن العدد *5* ليس أقل من *3*.

0: لأن العدد *5* لا يساوي العدد *3*.

1: لأن العدد *5* أكبير من *3*.

حيث *0* تعني خطأ *(false)* و *1* تعني صحيح *(true)*، و أيضا ستلاحظ أنه كتبنا في السطر السادس *5==3* و ليس *5=3* و ذلك لأننا نقارن و عند المقارنة نكتب ==، و إذا وضعت = مكان == فسيخبرك المتـــرجم عن وجود خطأ.

و يوجد كذلك المؤثر أكبر من أو يساوي و يمثل في لغة C بـــ =>، و مؤثر أصغر من أو يساوي بـــ =<، و المؤثر لا يساوي بـــ =!.

1.6.3 **المؤثرات المنطقية** (logical operators):

و هي مؤثرات تعتمد على المؤثرات العلاقية في نتيجتها و لكن لها رموزها الخاصة و هي:

&& و التي تعني "و"

|| و التي تعني "أو"

! و التي يعني "لا"

و هذا مثال يوضح ذلك:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%d\n", 5<3 && 5>3);
6:    printf("%d\n", 5==3 && 3==5);
7:    printf("%d\n", 5>3 && 5<3);
8:
9:    printf("%d\n", 5<3 || 5>3);
10:   printf("%d\n", 5==3 || 3==5);
11:   printf("%d\n", 5>3 || 5<3);
12:
13:   printf("%d\n", !(5<3));
14:   printf("%d\n", !(5==3));
15:   printf("%d\n", !(5>3));
16:}
```

نتائج البرنامج هي:

0: لأنه توجد علاقة خاطئة و هي *3>5*.

0: لأن كلا العلاقتين خطاءتين.

0: لأنه توجد علاقة خاطئة و هي *3>5*.

1: لأنه توجد علاقة صحيحة و هي *3<5*.

0: لأن كلا العلاقتين خطاءتين.

1: لأنه توجد العلاقة صحيحة و هي *3<5*.

1: لأن *5* ليست أصغير من *3*.

1: لأن *5* لا تساوي *3*.

0: لأن *5* أكبر من *3*.

1.6.4 مؤثرات أخرى:

توجد مؤثرات أخرى خاص بلغة *C*، المؤثر >> و يسمى بالتغيير اليساري،مثال توضيحي لطريقة استعماله:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%d\n", 2<<8);/* = (2E8)*2 */
6:}
```

حيث المثال السابق يعني *2* أس *8* و نتيجة نضربها في *2*.

و المؤثر المعاكس لسابق هو <<، يسمى بالتغيير الأيمن، و هذا مثال توضيحي لطريقة استعماله:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("%d\n", 256>>8);/* = printf("%d\n", (256/8)/2); */
6:}
```

و المثال واضح عبر التعليق، قسمة العدد *256* على *8* ثم تقسيم النتيجة على *2*.

المؤثر #، و هو يستعمل مع التوجيه #define، مثال:

```
1:#include<stdio.h>
2:
3:#define printer(str) printf(#str "\n")
4:
5:
```

```
6:main()
7:{
8:    printer(Hello);
9:}
```

المؤشرين ## معا، أيضا يتم التعامل معهما مع التوجيه #define، و يمكن تسميتهما بالـــ *Merge* أي الدمج، مثال:

```
1:#include<stdio.h>
2:
3:#define MergingName(str, strplus) str##strplus
4:
5:main()
6:{
7:    char MergingName(ch, 1) = 'H';
8:    char MergingName(ch, 2) = 'e';
9:    char MergingName(ch, 3) = 'l';
10:   char MergingName(ch, 4) = 'l';
11:   char MergingName(ch, 5) = 'o';
12:
13:   printf("%c%c%c%c%c\n", ch1, ch2, ch3, ch4, ch5);
14:
15:}
```

1.6.5 **الأخطاء المحتملة:**

لا توجد أخطاء محتملة في هذا الدرس.

1.6.6 **تمارين:**

1- هل يمكن أن نقوم بالتزايد و التناقص داخل الدالة printf ؟
2- ماذا تعني القيمة *0* و القيمة *1* في المؤثرات العلاقية ؟
3- ماذا تعني كل من && و || و *!* ؟

1.7 القرارات if, else, else...if

تستعمل القرارات (و يمكن تسميتها بالجمل الشرطية) للمقارنة بين علاقة حيث تكون إما صحيحة أو خاطئة، في كل من حالة لها أوامر خاص تقوم بتحديدها، مثلا إذا كانت المقارنة صحيح فسيتم تنفيذ الأوامر التي حددناها في حالة أن المقارنة صحيحة، أما إذا كانت المقارنة خاطئة فسيتم تنفيذ الأوامر المعاكسة.

1.7.1 استعمال if:

لكي تفهم طريقة استعمال if ألقي نظرة على الصورة التالية:

```
if ( المقارنة، أو الشرط )
{
      الأمر;
}
```

أو

```
if ( المقارنة، أو الشرط )
      الأمر;
```

الصورة السابقة يوجد بها مثال بدون الرمز } و { لأنه لا يوجد سوى أمر واحد لتنفيذه، و هذه صورة في حالة وجود أكثر من أمر:

```
if ( المقارنة، أو الشرط )
{
      الأمر 1;
      الأمر 2;
      .....;
      الأمر س;
}
```

مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int usr_val;
6:
7:    printf("Enter a value: ");
8:    scanf("%d", &usr_val);
9:
10:   if(usr_val <100)
11:   {
12:         printf("%d are lesser than 100\n", usr_val);
13:   }
14:
15:   if(usr_val >100)
16:   {
```

```
17:         printf("%d are greater than 100\n", usr_val);
18:    }
19:}
```

في هذا البرنامج طلبنا من المستخدم إدخال قيمة، ثم استعملنا الشرط أو المقارنة في السطر العاشر و السطر الخامس عشر و هو المقارنة بين القيمة التي أدخلها المستخدم و *100*، إذا كانت المقارنة صحيحة فسيقوم البرنامج بتنفيذ ما هو داخل الـ *block* الخاص بـ `if`، و في حالة أن المقارنة خاطئة فسيتم تجاهل ما هو داخل الـ *block* الذي يلي `if`.

و يمكن كتابة البرنامج بهذه الطريقة:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int usr_val;
6:
7:    printf("Enter a value: ");
8:    scanf("%d", &usr_val);
9:
10:   if(usr_val <100)
11:         printf("%d are lesser than 100\n", usr_val);
12:
13:   if(usr_val >100)
14:         printf("%d are greater than 100\n", usr_val);
15:}
```

لأنه في كلا الشرطين لا يوجد بهما إلا أمر واحد.

1.7.2 استعمال `else`:

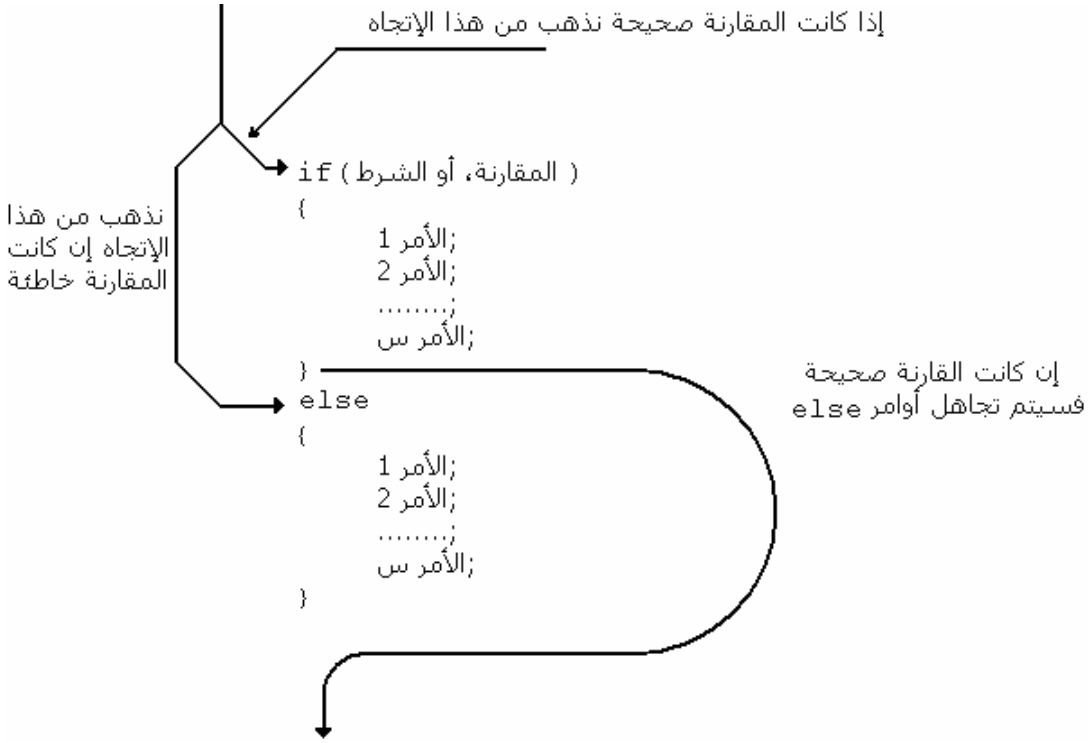
تستعمل الكلمة `else` مع الكلمة `if` دائما حيث لا يمكن استعمالها لوحدها، سنكتب الآن البرنامج السابق باستخدام `else`:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int usr_val;
6:
7:    printf("Enter a value: ");
8:    scanf("%d", &usr_val);
9:
10:   if(usr_val<100)
11:         printf("%d are lesser than 100\n", usr_val);
12:   else
13:         printf("%d are greater than 100\n", usr_val);
14:}
```

استعملنا `else` في السطر الثاني عشر و سترى أن المثال سيعمل مثل المثال السابق و هنا تجد فائدة `else`.

إن لم تكن المقارنة صحيح في if فسيتم تنفيذ else.

و هذه صورة توضح لطريقة عملهما:



1.7.3 استعمال else...if:

في المثال السابق لطريقة استعمال else إذا قمت بتنفيذ البرنامج و أعطيته القيمة *100* فسيقول لك أن النتيجة التي أدخلتها أكبر من *100*، و لتفادي هذا المشكلة نستعمل الكلمتين else...if معا، و هذا مثال يوضح ذلك:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int usr_val;
6:
7:    printf("Enter a value: ");
8:    scanf("%d", &usr_val);
9:
10:   if(usr_val<100)
11:         printf("%d are lesser than 100\n", usr_val);
12:   else if(usr_val==100)
13:         printf("%d are equal 100\n", usr_val);
14:   else
15:         printf("%d are greater than 100\n", usr_val);
16:}
```

و طريقة عمل هذا المثال هو المقارنة بين القيمة التي أدخلها المستخدم و *100* فإذا كانت أقل من *100* فسنطبع ما هو موجود في *block* الخاص بالمقارنة، و في حالة أن القيمة أكبر من *100* فسيتم الانتقال إلى المقارنة الثانية و هي إذا كان العدد الذي أدخله المستخدم يساوي *100*، في حالة أنها صحيح يقوم البرنامج بطباعة ما هو موجود في *block* الخاص بالمقارنة الثانية، في حالة أن كلا المقارنتين خاطئة فسيتم تنفيذ ما هو موجود في الـ *block* الخاص بـ `else`.

1.7.4 **الأخطاء المحتملة:**

1- لا يمكن استعمال `else` مرتين لمقارنة واحدة.

2- لا يمكن استخدام `else` بدون `if`.

1.7.5 **تمارين:**

1- أكتب برنامج يقوم بالمقارنة بين عمر المستخدم و العدد *40*، إذا كان عمر المستخدم أقل من *40* يخبره البرنامج بأن عمرها أقل من الأربعين، و في حالة العكس يخبره البرنامج بأن عمره أكبر من *40*، و إذا كان عمره أربعين يخبره البرنامج بأن عمره *40* سنة.
2- أكتب نفس البرنامج السابقة و لكن بدون استعمال `else`.
3- أكتب برنامج يطلب من المستخدم إدخال قيمة لا تتعدى العدد *9*، و عند إدخال أي عدد يخبره البرنامج أنه قد أدخل ذلك العدد (باستعمال `if...else`).

1.8 عناصر لغة C

في هذا الدرس سنتعرف على عناصر لغة C و التي هي:

1.8.1 **التعليقات** Comments:

هي من عناصر لغة C و قد درسنها سابقا.

التعليقات هي سلسلة من الرموز تبدأ بالرمز *slash* / و النجمة * و تنتهي بنجمة و *slash* */، و يمكن أن يحمل التعليق أي رموز أو أرقام.

الطريقة السابقة هي الطريقة الوحيدة في لغة C، و لكن الكثير من المترجمات تدعم طريقة أخرى من التعليق و هي التعليق السطري حيث يبدأ بالرمزين //.

1.8.2 **الكلمات المحجوزة** Keywords:

الكلمات المحجوزة هي كلمات أساسية في اللغة و التي يكون لونها في أغلب المترجمات أزرق، و سميت بالكلمات المحجوزة لأنها محجوزة سابقا، كل من `int` و `char` و `double` و `if` و `else` تعتبر كلمات محجوزة و الكثير منها، و هذا جدول لجميع الكلمات المحجوزة الأساسية في لغة C:

int	char	else	volatile	return	void	struct	float
short	signed	register	for	continue	typedef	case	static
long	unsigned	auto	while	break	union	switch	default
double	if	const	do	sizeof	enum	extern	goto

و الكلمة المحجوزة `asm`، عددها *33*، و سندرس جميع الكلمات المحجوزة في الدروس القادمة.

1.8.3 **المعرفات** Identifiers:

المعرفات هي سلسلة من حروف أو أرقام، أول حرف منها يجب أن يكون حرفا، و لا يمكن أن يبدأ اسم المعرف برقم، الرمز _ و الذي يدعى *underscore* يعتبر حرفا، و لا يمكن أن يتعدى اسم المعرف أكثر من *31* حرفا، كل من الأحرف الصغير و الأخر الكبيرة مختلفة، و لا يمكن استعمال معرف مرتين.

جدول للأحرف التي يمكن أن تستعمل في أسماء المعرفات:

الأحرف	الأرقام
a b c d e f g h i j k l m n o p q r s t u v w x y z _	9 8 7 6 5 4 3 2 1 0

	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

1.8.3.1 **سلاسل** Trigraphs:

هي سلاسل مكونة من ثلاثة رموز يقوم المترجم باستبدالها إلى رموز أخرى، و هذا جدول توضيحي لهـــذه الرموز:

Trigraphs	الرمز الذي يقابله
??=	#
??<	{
??>	}
??(	[
??)	]
??/	\
??′	^
??!	|
??-	~

و هذا مثال يوضح طريقة استعمال تلك الرموز:

```
1:??=include<stdio.h>
2:
3:main()
4:??<
5:    printf("Hello, World!\n");
6:??>
```

و البرنامج يعمل بدون أخطاء، و يمكن أيضا استعمال تلك الرموز داخل النصوص مثل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("??=\n");
6:    printf("??<\n");
7:    printf("??>\n");
8:    printf("??(\n");
9:    printf("??)\n");
10:   printf("??//\n");
11:   printf("??'\n");
12:   printf("??!\n");
13:   printf("??-\n");
14:}
```

1.8.4 **الثوابت** Constants:

الثوابت تكون إما أرقام أو أحرف أو سلاسل حرفية، لا يمكن التغير فيها أثناء البرنامج، و توجد نوعـــان منها و هي:

1.8.4.1 الثوابت النصية:

الثوابت النصية هي تكون إما بالاستعمال الموجه `#define` أو الكلمة المحجوزة `const`، بالنسبة للموجه `#define` فهذا مثال له:

```
1:#include<stdio.h>
2:
3:#define Const_Str "Hello, World!\n"
4:
5:main()
6:{
7:    printf(Const_Str);
8:
9:    /*can't write Const_Str = "Hello, World2!\n"*/
10:}
```

و أي نصوص أو أحرف معرفة باستخدام الموجه `#define` فهي ثابتة و لا يمكن التغير فيها.

أما الثوابت باستخدام الكلمة المحجوزة `const` فسأعطي مثال لحرف ثابت، المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    const char ch = 'a';
6:
7:    printf("%c\n", ch);
8:
9:    /*you can't use this:*/
10:   /*ch = 'b';*/
11:   /*printf("%c\n", ch);*/
12:   /*because it's a constant*/
13:}
```

أما الثوابت النصية باستخدام الكلمة المحجوزة `const` فسندرسها فيما بعد.

و توجد رموز ثابتة أخرى خاصة بلغة C و هي موضحة على الجدول التالي:

التأثير	الرمز	التأثير	الرمز
طباعة الرمز '	\'	الإنذار، تقوم بإصدار صوت من اللوحة الأم، *Alert*	\a
طباعة الرمز "	\"	الرجوع بمؤشر الكتاب إلى الخلف مع حذف الحرف الذي به، *Backspace*	\b
طباعة الرمز ?	\?	صفحة جديدة، *Form feed*	\f
طباعة الرمز \	\\	سطر جديد، *Newline*	\n

\r	العودة بمؤشر الكتابة إلى بداية السطر، *Carriage return*		
\t	*8* فراغات في الاتجاه الأفقي		
\v	*8* فراغات في الاتجاه العمودي		

1.8.4.2 **الثوابت الرقمية**:

هي الإعلان عن ثوابت بها قيم ثابتة لا يمكن التحديث فيها أثناء البرنامج، و يمكن أن تكون أي نوع من الأعداد `short, long, int, unsigned, signed, float, double`، و يمكن أيضا الإعلان عنها في كل من الموجه `#define` و الكلمة المحجوزة `const`، مثال حول الموجه `#define`:

```
1:#include<stdio.h>
2:
3:#define Const_Num 5
4:
5:main()
6:{
7:    printf("%d\n", Const_Num);
8:
9:    /*can't write Const_Num = 6;*/
10:}
```

عند الإعلان عن ثوابت باستخدام الموجه `#define` فإننا لا نقوم بتحديد نوعه، الموجه نفسه يقوم بتحديد نوع القيمة المدخلة.

و هذا مثال باستخدام الكلمة المحجوزة `const`:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int  Const_Num = 5;
6:    printf("%d\n", Const_Num);
7:
8:    /*can't write Const_Num = 6;*/
9:}
```

1.8.5 **الرموز** Tokens:

هي ستة فئات و هي: المعرفات، الكلمات المحجوزة، الثوابت، السلاسل النصية، المؤثرات و الفواصل.

1.8.6 **السلاسل النصية** String literals:

السلاسل النصية أي الثوابت النصية، هي سلاسل من حروف محاطة باقتباسين `"..."`، ستفهم هذا الجزء من الدروس القادمة.

1.8.7 **الأخطاء المحتملة:**

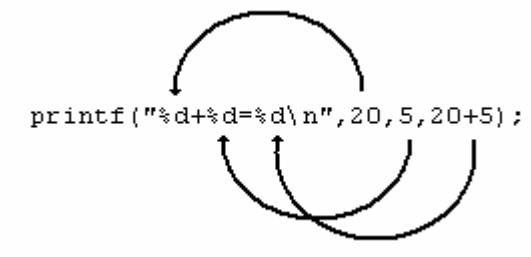
لا توجد أخطاء محتملة في هذا الدرس.

1.8.8 **تمارين:**

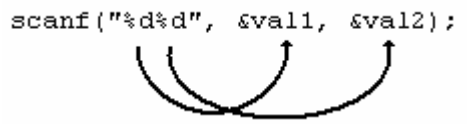
لا توجد تمارين في هذا الدرس.

1.9 ملخص للفصل الأول، مع إضافات

درسنا من قبل الدالة printf، و قلنا أنها خاصة بالإخراج، و هنا توجد صورة توضيحية لطريقة تعامل هذه الدالة مع رموز مثل %d، الصورة:



من الصورة نفهم أنه عند تشغيل البرنامج يتم استبدال القيم المدخل أو القيم المعطاة في مكان الرمز %d، و أيضا نفس الفكرة مع الدالة scanf، صورة توضيحية:



1.9.1 برامج تدريبية:

في هذا الجزء من الدرس سنرى بعض البرامج التي يستعملها كثيرا المبتدأين في لغة C، مع شرح سريع لكل برنامج:

1.9.1.1 البرنامج الأول، عمر المستخدم:

يقوم هذا البرنامج بإعطاء المستخدم عمره، وذلك عبر إدخال العام الحالي و العام الذي ولد به المستخدم، المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num1, num2;
6:
7:    printf("The year now: ");
8:    scanf("%d", &num1);
9:
10:   printf("Your year: ");
11:   scanf("%d", &num2);
12:
13:   printf("You have %d years!\n", num1-num2);
14:}
```

تم الإعلان عن متغير في السطر الخامس، و في السطر السابع طلبنا من المستخدم إدخال السنة الحالية، و في السطر الثامن أخذنا السنة على شكل عدد حقيقي.

في السطر العاشر طلبنا من المستخدم إدخال سنته، و في السطر الحادي عشر أخذنا السنة أيضا على شكل عدد صحيح.

و أخيرا في السطر الثالث عشر قمنا بطباعة النتيجة و التي هي طرح السنة الحالية على سنة المستخدم.

1.9.1.2 **البرنامج الثاني، آلة حاسبة بسيطة:**

هذا البرنامج عبارة عن آلة حاسبة بسيطة ذات حدود لا يمكن تجاوزها، يطلب من المستخدم اختيار إشارة للعملية التي سيقوم بها، ثم إدخال قيمتين اللذان سيجري عليهما العملية، المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num1, num2;
6:    char Char;
7:
8:    printf("1:(+)\n2:(-)\n3:(/)\n4:(*)\nEnter a choice: ");
9:    scanf("%c", &Char);
10:
11:   printf("Enter the first number: ");
12:   scanf ("%d", &num1);
13:
14:   printf("Enter the second number: ");
15:   scanf("%d", &num2);
16:
17:   if(Char == '+' || Char == '1')
18:         printf("%d + %d = %d\n", num1, num2, num1+num2);
19:
20:   else if(Char == '-' || Char == '2')
21:         printf("%d - %d = %d\n", num1, num2, num1-num2);
22:
23:   else if(Char == '/' || Char == '3')
24:         printf("%d / %d = %d\n", num1, num2, num1/num2);
25:
26:   else if(Char == '*' || Char == '4')
27:         printf("%d * %d = %d\n", num1, num2, num1*num2);
28:
29:   else
30:         printf("Error in choice!\nExiting...\n");
31:}
```

في السطر الخامس قمنا بالإعلان عن متغيرين من نوع أعداد صحيحة، و في السطر السادس قمنا بالإعلان عن متغير حرفي و الذي سيكون الإشارة التي سنقوم باستعمالها في العمليات.

في السطر الثامن طلبنا من المستخدم بإختيار الإشارة أو المؤثر الذي يريد استعماله، و أخذنا المؤشر الـــذي إختاره المستخدم في السطر التاسع على شكل رمز.

من السطر الحادي عشر إلى السطر الخامس عشر، قمنا بأخذ الأعداد التي أدخلها المستخدم.

في السطر السابع عشر قمنا باستعمال شرط و هو إذا كان الرمز الذي أدخله المـــستخدم في المـــتغير Char يساوي المؤثر + أو الرقم *1*، في حالة أن الشرط صحيح يتم تطبيق عملية الجمع، و هكذا بالنسبة لكل من الأسطر العشرين، الثالث و العشرين، السادس و العشرين، أما السطر التاسع و العشرين فسيتم تنفيـــذه إذا كان الرمز الذي أدخله المستخدم غير متوافق مع الرموز المطلوبة.

1.9.1.3 البرنامج الثالث، استخراج القيمة المطلقة:

هذا البرنامج يقوم باستخراج قيمة مطلقة لعدد يقوم بإدخاله المستخدم، المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num, x;
6:
7:    printf("Enter a number: ");
8:    scanf ("%d", &num);
9:
10:   if(num<0)
11:         x = -num;
12:   else
13:         x = num;
14:
15:   printf("|%d|=%d\n", num, x);
16:}
```

في السطر الخامس قمنا بالإعلان عن متغيرين، الأول للعدد الذي سيقوم بإدخاله المستخدم، و المتغير الثـــاني هو الذي سيحمل القيمة المطلقة للعدد الذي أدخله المستخدم.

في السطر السابع طلبنا من المستخدم إدخال عدد، و أخذنا العدد الذي أدخله المستخدم على شكل عـــدد صحيح في السطر الثامن.

في السطر العاشر قمنا باستعمال شرط و هو إذا كان العدد الذي أدخله المستخدم أقل من الصفر أي العدد سالب فإن المتغير x يساوي قيمة المتغير num مع عكس الإشارة، و في حالة أن الشرط كان خاطئ فهذا يعني أن العدد الذي أدخله المستخدم عدد أكبر من الصفر أي عدد موجب.

في السطر الخامس عشر قمنا بطباعة النتائج.

1.9.1.4 البرنامج الرابع، أخذ العدد الكبير:

يقوم هذا البرنامج بأخذ العدد الأكبر بين عددين يقوم بإدخالهما المستخدم، المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num1, num2, max;
6:
7:    printf("Enter a number: ");
8:    scanf("%d", &num1);
9:
10:   printf("Enter another number: ");
11:   scanf("%d", &num2);
12:
13:   if(num1>num2)
14:         max = num1;
15:   else
16:         max = num2;
17:
18:   printf("max = %d\n", max);
19:}
```

في السطر الخامس قمنا بالإعلان عن ثلاثة متغيرات، الأول و الثاني لأعداد التي سيقوم بإدخالها المستخدم، و المتغير الثالث لعدد الأكبر بين العددين الذي أدخلهما المستخدم.

من السطر السابع إلى السطر الحادي عشر، طلبنا من المستخدم إدخال عددين.

في السطر الثالث عشر قمنا باستعمال شرط و هو إذا كان العدد الأول الذي أدخله المستخدم أكبر من العدد الثاني فإن المتغير max يساوي قيمة المتغير num1 لأنها الأكبر، و في حالة أن الشرط خاطئ فهذا يعني أن العدد الثاني الذي أدخله المستخدم أكبر من الأول و هنا ستكون قيمة المتغير max هي قيمة المتغير num2 لأنه الأكبر.

و أخيرا السطر الثامن عشر و هو يقوم بطباعة النتائج.

1.9.2 الدالتين putchar و getchar:

الآن سنرى دالتين جديدتين و هما putchar و هي مختصرة من *put character* و الدالة getchar و هي مختصرة من *get character*، الدالتين من دوال الملف الرأسي stdio.h.

الدالة putchar تستعمل لإخراج الأحرف، مثلا لو أردنا أن نكتب الجملة *Hello, World !* سنكتب:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    putchar('H');
6:    putchar('e');
7:    putchar('l');
8:    putchar('l');
9:    putchar('o');
10:   putchar(',');
11:   putchar(' ');
12:   putchar('w');
13:   putchar('o');
14:   putchar('r');
15:   putchar('l');
16:   putchar('d');
17:   putchar('\n');
18:}
```

و لا يمكن استعمالها لطباعة النصوص، و في هذه الحالة ستفضل استعمال printf.

أما الدالة getchar فهي تأخذ من المستخدم حرف و تضعه في متغير، و هذا مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char ch;
6:
7:    putchar(':');
8:
9:    ch = getchar();
10:
11:   putchar(ch);
12:}
```

و هنا كتبنا ch = getchar() لأن الدالة getchar لا تحتوي على وسائط، و هنا الطريقة صحيحة لأن الدالة getchar تقوم بإرجاع قيمة و هي الحرف الذي أدخله المستخدم.

1.9.3 الدالتين puts و gets:

الدالتين puts و gets أيضا من دوال الملف stdio.h، و الأولى مختصرة من *put string* و الثانية *get string*.

الدالة puts مشابه لدالة printf مع بضع الاختلافات، و هي تستعمل لطباعة النصوص، و تختلف عن الدالة printf في:

1- يمكنك تجاهل وضع رمز السطر الجديد \n، لأن الدالة puts تضع كل نص في سطر.

2- الدالة puts لا يمكنها التعامل مع متغيرات من نوع أرقام أو أحرف، أي أنها لا تتعامل مع رموز مثل %d.

و هذا مثال لدالة puts:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    puts("Hello, World!");
6:    puts("Goodbay, World!");
7:}
```

أما عن الدالة gets فهي تقوم باستقبال سلسلة حرفية من المستخدم، و لا يمكن التعامل معها بالأرقـــام أو الأحرف، سنتطرق إليها في الدروس القادمة.

1.9.4 الدالتين wprintf و wscanf:

هما نفس الدالتين printf و scanf و لكنها عريضة و هي مختصرة من *wide printf format* و *wide scan format*، هذا مثال يوضح طريقة استعمالهما:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num, x;
6:
7:    wprintf(L"Enter a number: ");
8:    wscanf(L"%d", &num);
9:
10:   if(num<0)
11:         x = -num;
12:   else
13:         x = num;
14:
15:   wprintf(L"|%d|=%d\n", num, x);
16:}
```

و الحرف L يعني *long*.

1.9.5 الدالتين putch و getch و الدالة getche:

الدالتين putche و getche هما من الدوال القياسية في لغة *C*، في أنظمة *linux* تجد الدالتين في الملف الرأسي stdio.h، أما في أنظمة *Windows* فستجدها في الملف الرأسي conio.h (مختـــصر مـــن *Console Input*

Output)، و الدالة getch ليس من الدوال القياسية للغة *C* و لكنها متوفر في جميع المترجمات تقريبا على أنظمة *Windows*، و تستعمل بكثرة.

إذا كانت النسخة التي لديك من لغة *C* هي النسخة القياسية *ANSI C* فيمكنك الإعتماد على الملف الرأسي stdio.h، و في النسخة القياسية لا توجد الدالة getch.

الدالة putch هي نفس الدالة putchar، و الدالة getch متشابها لـــدالة getchar مع بعض الاختلافات:

1- الدالة getchar تسمح لك برؤية ما قمت بإدخاله، أما getch فلا تسمح لك بذلك.
2- يمكن إدخال أكثر من حرف في الدالة getchar حيث سيتم أخذ الحـــرف الأول مـــن الحـــروف المدخلة، أما getch فلا تستطيع إدخال أكثر من حرف.

مثال حول putch و getch:

```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
5:{
6:    char ch;
7:
8:    ch = getch();
9:    putch(c);
10:   putch('\n');
11:}
```

قمنا بضم الملف الرأسي conio.h في السطر الثاني، ثم استعملنا الدالة getch في الـــسطر الثـــامن حيـــث ستلاحظ أن طريقة استعمالها مثل الدالة getchar، ثم استعملنا الدالة في كلا السطرين التاسع و العاشر، و بالنسبة لرمز \n فهو يعتبر رمز واحد.

و في الكثير من البرامج ستجد أن الدالة getch مستعملة بكثرة و خاصة في نهاية البرنامج و ذلـــك لأنهـــا تتوقف بانتظار المستخدم بالضغط على زر من الأزرار و طريقة هي:

```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
5:{
6:    char ch;
7:
8:    ch = getch();
9:    putch(ch);
10:   putch('\n');
```

```
11:
12:   printf("Press any key to exit\n");
13:   getch(); /*pause*/
13:}
```

حيث هنا لن يخرج البرنامج مباشرة، سينتظر أن يقوم المستخدم بالضغط على أي زر كي يقوم البرنـــامج بالخروج.

أما الدالة getche فهي نفسها getch فقط عند استعمالها تظهر للمستخدم الرمز الذي قام بإدخاله، مثال:

```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
5:{
6:    char c;
7:
8:    c = getche();
9:    putch('\n');
10:   putch(c);
11:   putch('\n');
12:}
```

1.9.6 الكلمة المحجوزة wchar_t:

طريقة استعمالها مثل طريقة استعمال الكلمة المحجوزة char، حيث الكلمة wchar_t مختصرة مـــن *wide-character type* و حجمها *2* بايت، مثال لطريقة استعمالها:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    wchar_t wch = 'a';
6:
7:    printf("%c\n", wch);
8:}
```

1.9.7 الدالة الرئيسية main و wmain:

الدالة main هي دالة رئيسية حيث يبدأ البرنامج بتنفيذ الأوامر منها، باتجاه واحد و بطريقة منظمة، الدالـــة wmain (ليست من الدوال القياسية للغة *C*) هي نفسها main و لكنها عريضة، و كل الدوال و الكلمـــات المحجوزة العريضة تستعمل في النظام الحروف الدولي الموحد أو بما يسمى بـــ ***Unicode***.

و طرقة استعمال wmain مثل main و هذا مثال بسيط حولها:

```
1:#include<stdio.h>
2:
3:wmain()
4:{
```

```
5:    wprintf(L"Hello, World!\n");
6:}
```

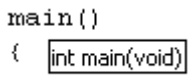
في بعض المترجمات إن كتبت البرنامج التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:
6:}
```

يحذرك بأن الدالة main يجب أن ترجع قيمة، و لإرجاع قيمة لدالة الرئيسية نكتب الكلمة المحجوزة return مع القيمة صفر و التي تعني الخروج من البرنامج بدون أخطاء، و هذا لطريقة استعمالها:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    return 0;
6:}
```

و سبب إرجاع قيمة لدالة الرئيسية هو الوضع الافتراضي لها و هو int، و جميع الدوال من نوع int يجب إرجاع قيمة لها، و إن كنت تستعمل مترجم *MicroSoft* فأشر بالفأرة إلى الدالة و سترى مثلما في الصورة التالية:



و هذا يعني أنه يمكن كتابة:

```
1:#include<stdio.h>
2:
3:int main()
4:{
5:    return 0;
6:}
```

أما إذا أردت عدم إرجاع قيمة لدالة الرئيسية فيمكن كتابة void بدل int، حيث تستعمل void للقيـــام بإجراءات و لا يمكن إرجاع لها قيمة، و هذا مثال:

```
1:#include<stdio.h>
2:
3:void main()
4:{
5:
6:}
```

1.9.8 رموز الإخراج و الإدخال:

رموز الإخراج الخاصة بالدالة printf موضحة على الجدول التالي:

الرمز	الشرح
%d, %i	للأعداد الصحيحة و يمكن استعمالهما لكل من الأنواع int, short, long
%f, %e, %g	للأعداد الحقيقية و يمكن استعمالهم لكل من الأنواع float, double
%u	للأعداد بدون إشارة و يمكن استعماله لكل من الأنواع unsigned, int, short, long
%c	للرموز و الأحرف و يستعمل مع المتغيرات الحرفية char
%s	للنصوص أو السلاسل الحرفية و يمكن استعماله مع char[] و char*
%o	لأعداد النظام الثماني
%x	لأعداد النظام السداسي
%p	للمؤشرات
%ld	لأعداد صحيحة طويلة *Long Decimal* و تستعمل في long int
%lu	لأعداد طويلة بدون إشارة long unsigned

و رموز الإدخال الخاصة بالدالة scanf على الجدول التالي:

الرمز	الشرح
%d, %i	للأعداد الصحيحة و يمكن استعمالهما لكل من الأنواع int, short, long
%f, %e, %g	للأعداد الحقيقية و يمكن استعمالهم لكل من الأنواع float, double
%u	للأعداد بدون إشارة و يمكن استعماله لكل من الأنواع unsigned, int, short, long
%c	للرموز و الأحرف و يستعمل مع المتغيرات الحرفية char
%s	للنصوص أو السلاسل الحرفية و يمكن استعماله مع char[] و char*
%o	لأعداد النظام الثماني
%x	لأعداد النظام السداسي
%ld	لأعداد صحيحة طويلة *Long Decimal* و تستعمل في long int
%lu	لأعداد طويلة بدون إشارة long unsigned

1.9.9 الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

1.9.10 تمارين:

1- أكتب برنامج يطلب من المستخدم إدخال عدد، ثم يقوم بالبرنامج بإخباره إن كان العـــدد الـــذي أدخله فردي أو زوجي.
2- أيهما أفضل في الإخراج الدالة `printf` أو الدالة `puts`؟ و لماذا؟
3- ماذا تعني الكلمة `conio`، و ما هي الدوال التي درستها من الملف الرأسي `conio.h` ؟
4- فيما تستعمل الكلمات المحجوزة و الدوال العريضة؟
5- لماذا نقوم بإرجاع قيمة لدالة الرئيسية في حالة أنها في الوضع الافتراضي، و ماذا تعين تلك القيمة في الدالة الرئيسية؟

الفصل الثاني – أساسيات في لغة C (2)

2.1 القرار *switch*

2.2 حلقات التكرار *repeated loops*

2.3 المصفوفات *arrays*

2.4 المؤشرات *pointers*

2.5 الدوال *Functions*

2.6 الملفات الرأسية *Header files*

2.7 الإدخال و الإخراج في الملفات *Files I/O*

2.8 التراكيب *structures*

2.9 ملخص للفصل الثاني، معا إضافات

2.1 القرار switch

درسنا سابقا القرارات if و else و else...if، و الآن سندرس القرار switch و الذي سيغنينا عن بـــاقي القرارات (if, else, else…if) في الكثير من الحالات منها:

هذا مثال كتبناه سابقا و هو عبارة عن آلة حاسبة بسيطة باستخدام القرارات:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num1, num2;
6:    char Char;
7:
8:    printf("1:(+)\n2:(-)\n3:(/)\n4:(*)\nEnter a choice: ");
9:    scanf("%c", &Char);
10:
11:   printf("Enter the first number: ");
12:   scanf ("%d", &num1);
13:
14:   printf("Enter the second number: ");
15:   scanf("%d", &num2);
16:
17:   if(Char == '+' || Char == '1')
18:         printf("%d + %d = %d\n", num1, num2, num1+num2);
19:
20:   else if(Char == '-' || Char == '2')
21:         printf("%d - %d = %d\n", num1, num2, num1-num2);
22:
23:   else if(Char == '/' || Char == '3')
24:         printf("%d / %d = %d\n", num1, num2, num1/num2);
25:
26:   else if(Char == '*' || Char == '4')
27:         printf("%d * %d = %d\n", num1, num2, num1*num2);
28:
29:   else
30:         printf("Error in choice!\nExiting...\n");
31:}
```

هنا يمكننا استعمال القرار switch أفضل، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num1, num2;
6:    char Char;
7:
8:    printf("1:(+)\n2:(-)\n3:(/)\n4:(*)\nEnter a choice: ");
9:    scanf("%c", &Char);
10:
11:   printf("Enter the first number: ");
12:   scanf ("%d", &num1);
13:
14:   printf("Enter the second number: ");
```

```
15:   scanf("%d", &num2);
16:
17:   switch(Char)
18:   {
19:   case '+':
20:   case '1':
21:         printf("%d + %d = %d\n", num1, num2, num1+num2);
22:         break;
23:
24:   case '-':
25:   case '2':
26:         printf("%d - %d = %d\n", num1, num2, num1-num2);
27:         break;
28:
29:   case '/':
30:   case '3':
31:         printf("%d / %d = %d\n", num1, num2, num1/num2);
32:         break;
33:   case '*':
34:   case '4':
35:         printf("%d * %d = %d\n", num1, num2, num1*num2);
36:         break;
37:
38:   default:
39:         printf("Error in choice!\nExiting...\n");
40:   }
41:
42:}
```

هنا سنلاحظ أن عدد أسطر البرنامج أكثر من السابق، و هذا لا يهم، الذي يهم هو طريقة كتابة البرنـــامج حيث تكون منظمة و مفهومة.

للقرار switch وسيط، و الذي يقوم بتتبع نتيجة المتغير الموجود داخل الوسيط، في مثالنا السابقة اســـتعمالنا المتغير Char، و للقرار switch حالات، و يتم تنفيذها حسب نتيجة المتغير الموجود في الوسيط.

الآن سأشرح المثال السابق بالتفصيل:

في السطر الثامن طلبنا من المستخدم إدخال نوع العملة التي يريد استخدامها، و في السطر التاســـع يأخـــذ البرنامج نوع العملية، و في السطر الحادي عشر و الرابع عشر طلبنا من المستخدم إدخال العـــدد الأول ثم العدد الثاني لإجراء العملية بينهما، و في السطر السابع عشر يأتي القرار switch و به وسيط في داخله المتغير Char حيث هنا ستتم المقارنة بين القيمة الموجد داخل المتغير Char و الحالات الموجودة، في السطر التاســـع عشر و العشرين سيقوم البرنامج بالمقارنة بين الإشارة + و الرقم *1* بالمتغير Char فإذا كانت مقارنة واحـــدة صحيحة فسيتم تنفيذ السطر الواحد و العشرين قم الخروج من القرار switch في السطر الثاني و العـــشرين بدون متابعة المقارنة مع الحالات الأخرى، و في حالة أن المقارنة أو الشرط لم يتحقق فسيتم الانتقـــال إلى

الحالة التي بعدها و تأتي المقارنة معها و معا المتغير Char و هكذا...، و في السطر الثـــامن و ثلاثـــين تجـــد الكلمة المحجوزة default و سيتم تنفيذ ما هو بعدها إن لم يتحقق أي شرط من الشروط السابقة.

سأجعل المثال السابق مفهوم أكثر، لذا سأقوم بالمقارنة بينه و بين المثال الـــذي قبلـــه في المقارنـــة الأولى و الأخيرة.

```
19:   case '+':
20:   case '1':
21:         printf("%d + %d = %d\n", num1, num2, num1+num2);
22:         break;
```

هذا في القرار switch، أما في القرارات if, else, else…if فهو:

```
17:   if(Char == '+' || Char == '1')
18:         printf("%d + %d = %d\n", num1, num2, num1+num2);
```

و هنا ستلاحظ أن السطرين:

```
19:   case '+':
20:   case '1':
```

هو نفسه:

```
Char == '+' || Char == '1'
```

أما الكلمة المحجوزة default:

```
38:   default:
39:         printf("Error in choice!\nExiting...\n");
```

فهي else في المثال السابق:

```
29:   else
30:         printf("Error in choice!\nExiting...\n");
```

و في القرار switch لا يمكن استعمال متغير من نوع أعداد حقيقية مثل float و double.

2.1.1 الكلمة المحجوزة case:

هي من الكلمات المحجوزة و التي تعلمنها مؤخر، و هي لا تستعمل إلى في القرار switch، و هي تعني كلمة حالة، بعد اسم الحالة يأتي الشرط، فمثلا '1' case هنا سيتم المقارنة بين الرقم *1* و المـــتغير الموجـــود في

وسيط القرار switch، فإذا كانت المقارنة صحيحة فسيتم تنفيذ ما هو بعد الحالة من أوامر، و في حالة أنها خاطئة فسيتم الانتقال إلى الحالات الأخرى.

لا يمكن استعمال نص للمقارنة في الكلمة المحجوزة case، لأنها تتعامل مع الأرقام و الأحرف فقط، حتى القرار switch لا يمكن إعطائه متغير منت نوع سلسلة حروف، هو أيضا لا يقبل إلا بمغيرات الأحرف و الأرقام.

2.1.2 الكلمة المحجوزة break:

يمكن استعمالها داخل القرار switch و هي تعني الانقطاع، و هي تستعمل مع الكلمة المحجوزة case، في حالة الاستغناء عنها فستأتي نتائج غير مرغوب بها، و عملها هو الخروج من الحلقة القرار switch.

2.1.3 الكلمة المجوزة default:

أيضا تستعمل مع القرار switch و هي الوضع الافتراضي، أي أنه إن لم تتحقق أي حالة من الحالات السابقة فسيتم تنفيذ ما هو بعدها.

2.8.4 الأخطاء المحتملة:

1- في حالة أنك أردت استعمال الحالة مباشرة باستخدام الرقم بعد case فيجب عليك استعمال متغير من نوع أعداد صحيحة، و ليس متغير أحرف و المثال السابق سيصبح على هذا الشكل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int num1, num2;
6:    int Char;
7:
8:    printf("1:(+)\n2:(-)\n3:(/)\n4:(*)\nEnter a choice: ");
9:    scanf("%d", &Char);
10:
11:   printf("Enter the first number: ");
12:   scanf ("%d", &num1);
13:
14:   printf("Enter the second number: ");
15:   scanf("%d", &num2);
16:
17:   switch(Char)
18:   {
19:   case 1:
20:         printf("%d + %d = %d\n", num1, num2, num1+num2);
21:         break;
22:
23:   case 2:
```

```
24:         printf("%d - %d = %d\n", num1, num2, num1-num2);
25:         break;
26:
27:   case 3:
28:         printf("%d / %d = %d\n", num1, num2, num1/num2);
29:         break;
30:
31:   case 4:
32:         printf("%d * %d = %d\n", num1, num2, num1*num2);
33:         break;
34:
35:   default:
36:         printf("Error in choice!\nExiting...\n");
37:   }
38:
39:}
```

2- يجب دائما الانتباه إلى الكلمة المحجوزة break و مكان استعمالها.

2.8.5 تمارين:

1- أكتب برنامج به قائمة رئيسية للعبة، حيث سيكون لها أربع خيارات و هي:

الخيار الأول هو *Start Game* و عند استعمال هذا الخيار مخبره بأن اللعبة قد بدأ.

و الخيار الثاني هو *Option* و التي به هي أيضا أربع خيارات لكل من تعديل الصوت، الصورة، الفأرة و لوحة التحكم.

الخيار الثالث هو *About Game* و التي ستضع فيها معلومات عنك.

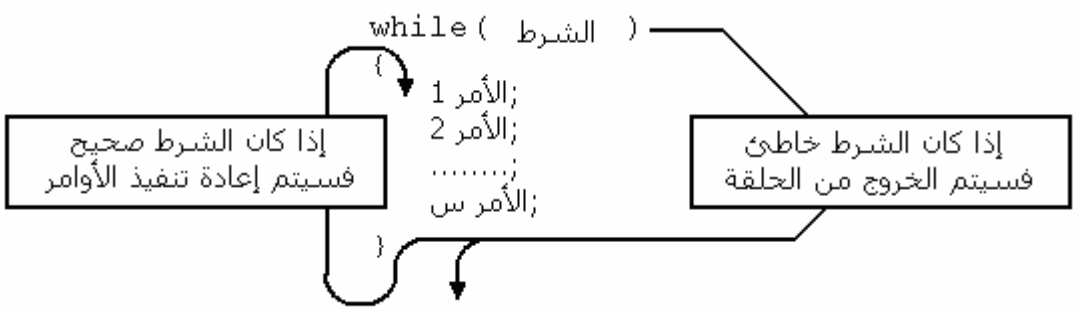
و الخيار الأخير هو الخروج من اللعبة *Exit Game*.

2.2 حلقات التكرار repeated loops

معنى حلقات التكرار بصفة عامة هي تكرار شيء عدت مرات، و في البرمجة التكرار يكون لحلقة يتم تكرارها لمرات يقوم بتحديدها المبرمج، توجد ثلاثة طرقة أساسية في التكرار، و توجد طريقة رابعة و لكنها لا تعتبر من طرق التكرار، الطرق هي:

2.2.1 التكرار بواسطة while:

طريقة استعمالها مشابه قليلا لطريقة استعمال الكلمة المحجوزة if، و هذه صورة بها شرح لطريقة استعمالها:



سيتم تنفيذ الأوامر بدون توقف حتى يصبح الشرط خاطئ، و إن كان شرط ثابت أو أن قيمته لا تتغير فسيتم تنفيذ الأوامر مرات لا حدود لها، أي سيبقى تنفيذ الأوامر بدون توقف لذا يجب أن نستعمل مؤثرات الزيادة أو النقصان أو نعطي الحرية للمستخدم بإيقاف الحلقة متى أراد.

الآن سأعطي مثال بسيط حول طريقة استعمال التكرار بواسطة الكلمة المحجوزة while حيث يقوم بطباعة عد تصاعدي من *1* إلى *3*:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int i;
6:    i = 1;
7:
8:    while(i!=4)
9:    {
10:         printf("\a%d ,", i);
11:         i++;
12:   }
13:
14:   printf(" Go!\n");
15:}
```

في حلقات التكرار دائما نستعمل متغيرات و إلى ستكون الحلقة بلا نهاية، و هنا استعملنا المتغير i و الـــذي سيكون العداد للحلقة، و يجب أيضا إعطاء للمتغير قيمة يبدأ بها، في مثالنا أعطيناه القيمة *1* حيث ســـيبدأ العد من الرقم *1*، و في السطر الثامن وضعنا الكلمة المحجوزة while مع شرط (مقارنة) أن لا يكون المتغير i يساوي *4* حيث سيتم تنفيذ ما هو داخل الـــ *block* الخاص بـــ while إلى أن تصبح المقارنة خاطئة (حيث سيتم الخروج من الحلقة إن كان المتغير i يساوي *4* و هذا يعني أن الشرط i !=4 ســـيكون خـــاطئ)، و ستجد أيضا أنه استعملنا مؤثر الزيادة في السطر الحادي عشر، و عدم استعماله يعطينا نتائج ثابت و غـــير منتهية.

المثال السابق لمتغير من نوع عدد صحيح، الآن سأعطي مثال بسيط لطريقة استعمال التكرار لمتغير حرفي:

```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
5:{
6:    char ch;
7:
8:    printf("Press any key(q=exit): ");
9:    ch = getche();
10:
11:   while(ch != 'q')
12:   {
13:         printf("\nPress any key(q=exit): ");
14:         ch = getche();
15:   }
16:
17:   printf("\nExiting...\n");
18:}
```

استعملنا متغير حرفي في السطر السادس ثم طلبنا من المستخدم إدخال أي حرف، و استعمالنا الدالة getche لكي لا نسمح للمستخدم بإدخال أكثر من حرف، و في السطر الحادي عشر استعمالنا الكلمة المحجـــوزة while مع شرط أن لا يكون المتغير ch يساوي الحرف q، حيث هنا سيبقى البرنامج يعمل بلا نهاية إلا إذا ضغط المستخدم على الحرف q.

يمكننا الاستغناء عن الشرط في الكلمة المحجوزة while و كتابته داخل الـــ *block* الخاص بهـــا باســـتعمال المقارنة عبر الكلمة المحجوزة if حيث سيكون داخل الـــ *block* الخاص بها الكلمة المحجوزة break و الـــتي قلنا عليها سابقا أنها تنهي الحلقات، هذا المثال السابق بعد التعديل:

```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
```

```
5:{
6:    char ch;
7:
8:    printf("Press any key(q=exit): ");
9:    ch = getche();
10:
11:   while(1)
12:   {
13:         if(ch == 'q')
14:               break;
15:
16:         printf("\nPress any key(q=exit): ");
17:         ch = getche();
18:   }
19:
20:   printf("\nExiting!\n");
21:}
```

و هنا استعملنا الرقم *1*(و الذي يعني `true` في لغة *C*) في وسيط الكلمة المحجوزة `while` كي تكون الحلقة بلا نهاية، و هنا ستلاحظ أن الكلمة المحجوزة `break` لا تستعمل فقط في الكلمة المحجوزة `switch`.

في المثال السابق يمكننا الاستغناء عن السطر الثامن و التاسع و نجعل الشرط الموجود في السطر الثالث عشر في آخر الحلقة حيث سيصبح المثال كما يلي:

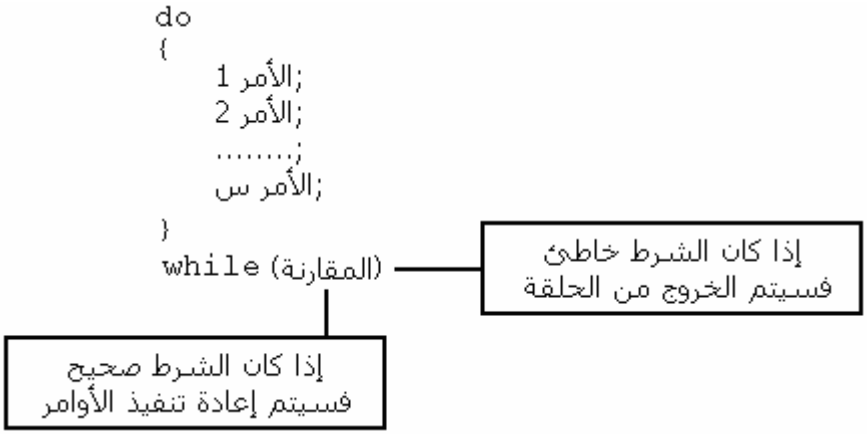
```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
5:{
6:    char ch;
7:
8:    while(1)
9:    {
10:         printf("\nPress any key(q=exit): ");
11:         ch = getche();
12:
13:         if(ch == 'q')
14:               break;
15:   }
16:
17:   printf("\nExiting!\n");
18:}
```

في هذا المثال سيتم تنفيذ الأوامر الموجود في الـ *block* الخاص بـ `while` مرة واحدة على الأقل، حتى و إن تحقق الشرط، و لكنها ليست من ميزات الحلقة `while`، حيث تستعمل هذه طريقة غالبا أو ربما لا تستعمل أصلا.

2.2.2 التكرار بواسطة `do`...`while`:

طريقة استعمالها مثل الطريقة السابقة لـــ while مع بعض الإضافات، و الفرق بين do...while و while هو أن التكرار باستخدام while لا يقوم بتنفيذ الأوامر الموجود فيه و لا مرة واحدة إلا إذا كانـــت الـــشرط صحيح مرة واحدة، أما التكرار بواسطة do...while فسيتم تنفيذ الأوامر الموجود بالـــ *block* الخاص به مرة واحدة على الأقل حتى و إن كان الشرط خاطئ.

طريقة استعمالها هي كتابة الكلمة المحجوزة do ثم نقوم بوضع *block* خاص بها حيث ستكون بـــه الأوامـــر المراد تنفيذها، و في رمز نهاية الـــ block و الذي هو } نكتب while مع الشرط، و هذه صورة توضيحية:



و هذه الأمثلة السابق باستخدام do...while:

مثال العد التصاعدي:

```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
5:{
6:    int i;
7:    i = 1;
8:
9:    do
10:   {
11:         printf("\a%d ,", i);
12:         i++;
13:   }while(i != 4);
14:
15:   printf(" Go!\n");
16:}
```

و هذا مثال المتغير الحرفي:

```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
```

```
5:{
6:    char ch;
7:
8:    do
9:    {
10:         printf("\nPress any key(q=exit): ");
11:         ch = getche();
12:   }while(ch != 'q');
13:
14:   printf("\nExiting...\n");
15:}
```

هنا ستجد أن عدد الأسطر قد قل، و لكن لا توجد طريقة أحسن من الأخرى لأن كل طريقة و مكـــان استعمالها.

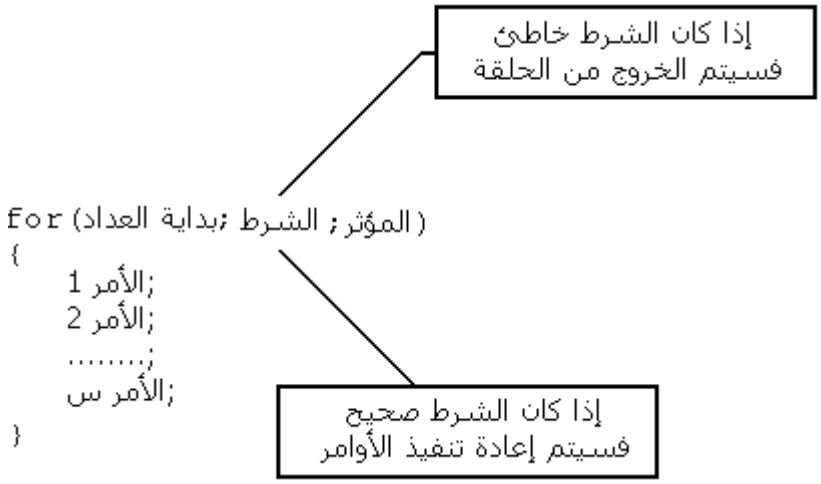
2.2.3 التكرار بواسطة for:

تعتبر هذه الطريقة من أسهل الطرق السابقة في الاستعمال، هي الطريقة الأكثـــر اســـتعمالا لبـــساطتها و سهولتها.

للكلمة المحجوزة for ثلاثة وسطاء و يتم الفصل بينها بفاصلة منقوطة بحيث:

1- الوسيط الأول نضع به قيمة التي سيبدأ بها العداد.
2- الوسيط الثاني هو الشرط، في حالة أن الشرط كان خاطئ يتم الخروج من الحلقة.
3- الوسيط الثالث هو المؤثر(غالبا ما يكون مؤثر الزيادة أو مؤثر النقصان).

صورة توضيحية:



مثال العد التصاعدي سيكون على هذه الطريقة:

```
1:#include<stdio.h>
```

```
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
5:{
6:    int i;
7:
8:    for(i=1;i<=3;i++)
9:    {
10:         printf("\a%d ,", i);
11:   }
12:
13:   printf(" Go!\n");
14:}
```

هنا نقوم أولا بالإعلان عن المتغير في بداية البرنامج ثم نعطي المتغير قيمة داخل الكلمة المحجـــوزة for أي في أول وسيط لها حيث ستكون تلك القيمة هي بداية العداد، ثم المقارنة في الوسيط الثاني حيث يجـــب أن لا يتعدى المتغير i العدد 3 (إذا تعدى المتغير i العدد 3 فسيصبح الشرط خاطئ و سيتم الخروج من الحلقة)، و في الوسيط الثالث استعملنا مؤثر الزيادة.

أما بالنسبة لمثال المتغير الحرفي فالطريقة تختلف قليلا، حيث سنقوم بالاستغناء عن الوسيط الأول و الأخير، و سنستعمل الوسيط الثاني لوضع الشرط حيث إن كان الشرط خاطئ يتم الخروج من الحلقة، و الـــشرط هنا هو أن لا يساوي المتغير ch الحرف q (سيصبح الشرط خاطئ إذا كان المتغير ch يساوي الحرف q)، و البرنامج سيكون على هذا الشكل:

```
1:#include<stdio.h>
2:#include<conio.h>   /* Just in Windows and DOS OS compilers */
3:
4:main()
5:{
6:    char ch;
7:
8:    printf("Press any key(q=exit): ");
9:    ch = getche();
10:
11:   for(;ch!='q';)
12:   {
13:         printf("\nPress any key(q=exit): ");
14:         ch = getche();
15:   }
16:
17:   printf("\nExiting...\n");
18:}
```

في السطر الحادي عشر استعملنا التكرار مع تجاهل الوسيطين الأول و الأخير و لكن يجب أن نترك مكانـــا لهما، و استعملنا الوسيط الثاني و هو الشرط الذي في حالة كان خاطئ يتم الخروج من الحلقة.

2.2.4 التكرار بواسطة goto:

هذه الطريقة إضافية، الكلمة المحجوزة goto تستعمل لـــ التنقل من مكان لآخـــر في مـــصدر البرنـــامج، سأشرح أولا طريقة استعمالها ثم نذهب إلى التكرار بواسطتها.

طريقة استعمالها بسيطة جدا، نكتب الكلمة المحجوزة goto ثم اسم المكان الذي نريد الذهاب إليه، هـــذه صورة توضيحية:

```
goto the_name_of_place;
```

و لكن هذا الكود لا يكفي، يجب أن نضع اسم المكان حيث سيكون على الشكل التالي:

```
the_name_of_place:
```

الآن سأضع مثال بسيط مفهوم:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    goto fin;
6:    printf("Begin!\n");
7:
8:fin:
9:    printf("Fin!\n");
10:}
```

هنا عند تنفيذ البرنامج سيتم تجاهل كل ما بين goto the_name_of_place و اسم المكان، و هنـــا اســـم المكان الذي نريد الانتقال إليه هو fin و هو موجود في السطر الثامن، و أردنا الذهاب إلى المكـــان fin في بداية البرنامج.

إذا أردنا الرجوع إلى بداية البرنامج فالطريقة سهلة، نكتب كما يلي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:begin:
6:    goto fin;
7:    printf("Begin!\n");
8:
9:fin:
10:   printf("Fin!\n");
11:   goto begin;
12:}
```

هنا ستجد تأثر التكرار، حيث هنا حلقتنا لن تنتهي، و إذا أردنا أن نجعل البرنامج يقوم بتكرار نحدده نحن فسنقوم باستعمال متغير و الذي سيكون العداد ثم مؤثر ثم الشرط، و سيصبح المثال على هذا الشكل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int i;
6:    i = 0;
7:
8:    printf("Begin!\n");
9:
10:begin:
11:   printf("%d ,", i);
12:   i++;
13:   if(i==10)
14:         goto fin;
15:   else
16:         goto begin;
17:
18:fin:
19:    printf("\nFin!\n");
20:}
```

و هنا نكون قد وضعنا تكرار بواسطة الكلمة المحجوزة goto.

2.2.5 المفهوم العام لحلقات التكرار:

لكي تفهم التكرار بصفة عامة فكل ما عليك معرفته هو:

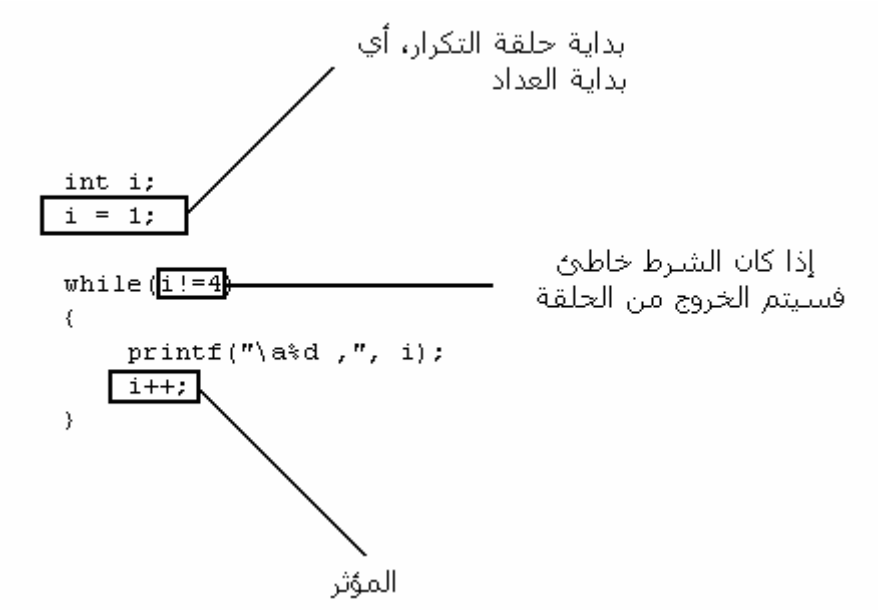
1- لكل حلقة تكرار بداية و التي تسمى ببداية العداد، هنا توجد حالات يمكن فيها تجاهل وضع بداية التكرار، مثل استعمال متغيرات حرفية.
2- لكل حلقة تكرار شرط، في حالة كان خاطئ يتم الخروج من الحلقة، حيث تجاهل وضع الشرط ينتج حلقة غير منتهية.
3- لكل حلقة مؤثر سواء مؤثر الزيادة أو النقصان، هنا أيضا توجد حالات يمكن فيها تجاهل وضع مؤثر، مثل استعمال متغيرات حرفية.

مثالنا الأول في هذا الدرس هو:

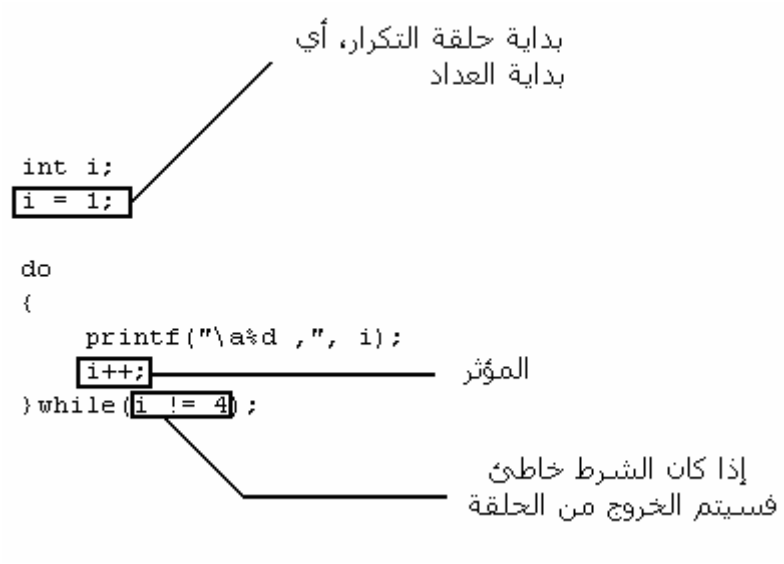
```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int i;
6:    i = 1;
7:
8:    while(i!=4)
```

```
9:    {
10:         printf("\a%d ,", i);
11:         i++;
12:   }
13:
14:   printf(" Go!\n");
15:}
```

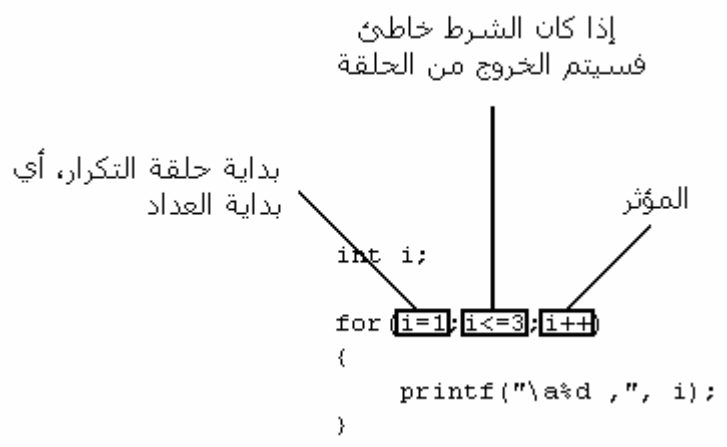
و شرحه بصفة عامة موجود على الصورة التالية:



هذه الصورة لتكرار بواسطة `while`، و هذه صورة أخرى عن التكرار بواسطة `do...while`:



و أخيرا صورة توضيحية للتكرار بواسطة `for`:



و تبقى طريقة التكرار goto، تلك الطريقة يمكننا تجاهلها لأنها طريقة إضافية و هي غالبا ما تستعمل في التكرار.

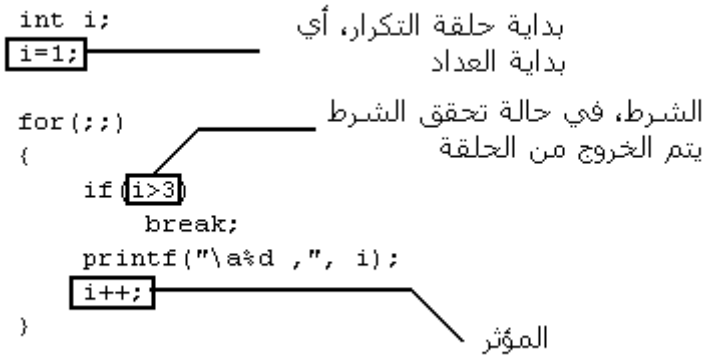
جميع الطرق السابقة يمكن استعمالها بطريقة أخرى، حيث يمكن أن نقوم بكتابة كل من بداية العداد و الشرط و المؤثر خارج وسائط كل من while, do...while, for، و هذا مثل طبيعي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int i;
6:
7:    for(i=1;i<=3;i++)
8:    {
9:          printf("\a%d ,", i);
10:   }
11:
12:   printf(" Go!\n");
14:}
```

و هنا نفس المثال السابق و لكن بطريقة أخرى:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int i;
6:    i=1;
7:
8:    for(;;)
9:    {
10:         if(i>3)
11:               break;
12:         printf("\a%d ,", i);
13:         i++;
14:   }
15:
16:   printf(" Go!\n");
17:}
```

و لكي تفهما هذا المثال إليك صورة توضحه:



و هنا ستجد أن النتيجة ستكون نفسها، و المثال هنا باستخدام for، يمكن استعمال while و do...while، بنفس الفكرة.

2.2.7 الكلمة المحجوزة continue:

تستعمل الكلمة المحجوزة continue داخل حلقات التكرار، حيث تقوم بالرجوع إلى بداية الحلقة، و كي تعرف أهميتها جرب المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int i;
6:    i = 1;
7:
8:    while(1)
9:    {
10:         if(i<=3)
11:         {
12:               printf("\a%d ,", i);
13:               ++i;
14:         }
15:
16:         printf(" Go!\n");
17:         break;
18:   }
19:
20:}
```

هذا المثال لن يعمل كما ينبغي، نريد من البرنامج أن يقوم بعد تصاعدي من *1* إلى *3* أما هذا البرنامج لن يطبع إلى العدد *1*، و كي يعمل البرنامج بطريقة صحيحة نظيف الكلمة المحجوزة continue إلى في نهاية المقارنة و سيصبح المثال السابق كما يلي:

```
1:#include<stdio.h>
2:
3:main()
```

```
4:{
5:    int i;
6:    i = 1;
7:
8:    while(1)
9:    {
10:         if(i<=3)
11:         {
12:               printf("\a%d ,", i);
13:               ++i;
14:               continue;
15:         }
16:
17:         printf(" Go!\n");
18:         break;
19:   }
20:
21:}
```

و يمكن استعمال المثال السابق على كل من do...while و for.

2.2.8 جدول ASCII:

باستخدام حلقات التكرار يمكننا معرفة كل رمز في جدول *ASCII* و رقمه، و ذلك باستخدام المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int i;
6:
7:    for(i=0;i<=255;i++)
8:          printf("%d: %c\n", i, i);
9:
10:}
```

و ستجد عند تشغيل البرنامج أن بعض الرموز لم تطبع منها الرقم *0* و الرقم *7* و *8* و *9* و *13* و غيرهـــا، و سبب في ذلك أنها لا تقوم بأعمال ظاهرة فمثلا الرقم *0* هو الرمز \0 و الرقم *13* هو زر الدخول في لوحـــة المفاتيح.

2.2.9 الأخطاء المحتملة:

1- يجب دائما الانتباه إلى الشرط و طريقة استعمالها.

2- تذكر دائما أن التكرار بواسطة do...while يتم تنفيذ أوامره مرة واحدة على الأقل، حتى و إن كـــان الشرط خاطئ.

3- استعمال حلقة التكرار for فارغة البرامترات تنتج حلقة غير منتهية، حيث ستكون على الشكل for(;;).

4- استعمال الطريقة while(1) يعني حلقة بلا نهاية.

5- استعمال الكلمة المحجوزة continue بدون شرط يعني حلقة بلا نهاية.

2.2.10 تمارين:

1- أكتب برنامج يقوم بالعد التنازلي من *100* إلى *0*، بالطرق الثلاثة.
2- أكتب برنامج يقوم بطباعة الأعداد الثنائية من *0* إلى *100*، بالطرق الثلاثة.

2.3 المصفوفات arrays

تستعمل المصفوفات لإدارة مجموعة كبيرة من البيانات لها نفس النوع، و باستخدام اسم واحد، و يمكن أن تكون المصفوفة من أي نوع من أنواع المتغيرات، و لا يمكن استعمالها معا الدوال.

فائدة المصفوفات كبيرة، و طرق استعمالها كثيرة و متنوعة، و لكي ترى فائدتها بشكل كبير فتخيل أنه طلب منك بناء برنامج به أكثر من 20 متغير و كل متغير به قيمة ربما نقوم بتحديدها نحن أو يقوم بتحديدها المستخدم، و إليك المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr1, arr2, arr3, arr4, arr5, ...,arr25;
6:
7:    printf("Arr1: ");
8:    scanf("%d", arr1);
9:
10:   printf("Arr2: ");
11:   scanf("%d", arr2);
12:
13:   printf("Arr3: ");
14:   scanf("%d", arr3);
15:
16:   printf("Arr4: ");
17:   scanf("%d", arr4);
18:
19:   printf("Arr5: ");
20:   scanf("%d", arr5);
21:
22:   ...
23:
24:   printf("Arr25: ");
25:   scanf("%d", arr25);
26:
27:}
```

تخيل...كم سيكون عدد أسطر البرنامج إن قمنا بكتابة جميع المتغيرات و ترك المستخدم يعطي لها قيم، ثم ماذا لو أردنا طبع النتائج، سيكون عدد أسطر البرنامج أكثر من 200 سطر و سيكون البرنامج جامد، غير مرن و غير مفهوم.

هنا تكمن فائدة المصفوفات، طريقة استعملها و التعامل مها مشابه لطريقة التعامل مع المتغيرات، كل ما ستفعله هو كتابة نوع المصفوفة ثم اسمها ثم يأتي حجمها و هذه صورة توضيحية:

و الآن سأكتب المثال السابق باستخدام المصفوفات:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[24];
6:    int i;
7:
8:    for(i=0;i<25;i++)
9:    {
10:         printf("Arr%d: ", i);
11:         scanf("%d", &arr[i]);
12:   }
13:
14:   printf("***********- LIST -***********\n");
15:
16:   for(i=0;i<25;i++)
17:   {
18:         printf("Arr%d: %d\n", i, arr[i]);
19:   }
20:
21:}
```

الآن أصبح البرنامج أكثر مرونة و مفهوم مع قلة عدد الأسطر، إن كان هذا المثال صعب فيمكن الرجـــوع إليه بعدما تقرأ هذا الدرس جيدا.

2.3.1 أساسيات في المصفوفات:

إذا كانت لدين مصفوف بها العدد *4* أي `int arr[4]` فهذا يعني أننا قمنا بأخذ أربعة أماكن في الذاكر كل واحدة مهنا بحجم المتغير `int`، توجد طريقتين لإعطاء قيمة للمصفوفات، الطريقة الأولى هي كتابة المصفوفة ثم بعدها مباشرة القيم التي بها، مثال توضيحي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[4] = {10, 12, 13, 15};
6:
7:    printf("[%d][%d][%d][%d]\n", arr[0], arr[1], arr[2], arr[3]);
8:}
```

حيث ستكون طريقة وضع القيم منظمة، مثلا إن وضعنا ثلاثة قيم لمصفوفة ذات أربعة أماكن فإن المكان الرابع سيبقى بدون قيمة، و ترى أنه عندما طبعنا النتائج في السطر السابع بدأنا بالمصفوف 0 حيث جميع المصفوفات تبدأ من *0*، لأن *0* يعتبر قيمة.

لا يمكن استعمال المثال السابق بهذه الطريقة:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[4];
6:    arr[4] = {10, 12, 13, 15};
7:
8:    printf("[%d][%d][%d][%d]\n", arr[0], arr[1], arr[2], arr[3]);
9:}
```

أي أنه يجب إعطاء القيم مباشرة بعد الإعلان عن المصفوفة، توجد طريقة أخرى و هي الإعلان عن المصفوفة ثم إعطاء كل صف قيمته و هذا مثال عن طريقة استعمالها:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[4];
6:    arr[0] = 10;
7:    arr[1] = 12;
8:    arr[2] = 13;
9:    arr[3] = 15;
10:
11:   printf("[%d][%d][%d][%d]\n", arr[0], arr[1], arr[2], arr[3]);
12:}
```

و هي الطريقة التي تستعمل بكثرة.

2.3.2 **المصفوفات الثنائية الأبعاد:**

المصفوفات التي درسنها سابقا هي مصفوفات ذات بعد واحد، الآن سندس مصفوفات ذات بوعدين حيث ستكون طريقة الاستعمال بسيطة و مشابه للمصفوفات ذات بوعد واحد.

لكتابة مصفوفة ذات بوعدين، سنكتب مصفوفة طبيعية ثم نظيف إليها صف آخر و هذا مثال سأقوم بشرحه:

```
1:#include<stdio.h>
2:
3:main()
4:{
```

```
5:    int arr2d[2][2];
6:    int i,j;
7:
8:    arr2d[0][0] = 10;
9:    arr2d[0][1] = 20;
10:
11:   arr2d[1][0] = 30;
12:   arr2d[1][1] = 40;
13:
14:   for(i=0;i<=1;i++)
15:   {
16:         for(j=0;j<=1;j++)
17:         {
18:               printf("arr2d[%d][%d] = %d\n", i, j, arr2d[i][j]);
19:         }
20:   }
21:
22:}
```

في السطر الخامس قمنا بإنشاء المصفوفة الثنائية، أعطين قيم لكل صف من صفوف المصفوفات في كل من السطر الثامن و التاسع للمصفوفة الأولى [0]، و السطر الحادي عشر و الثاني عشر للمصفوفة الثانية [1].

مصفوفات ثنائية الأبعاد هي المصفوفات بها مصفوفات، أي مصفوفة رئيسية و مصفوفة ثانوية، فإذا كانت لدينا مصفوفة رئيسية بحجم 2 من نوع أعداد صحيحة و مصفوفة ثانوية بحجم 2 فهذا يعني أن المصفوفة الرئيسية الأولى تنقصم إلى مصفوفتين ثانويتين، و المصفوفة الرئيسية الثانية أيضا تنقسم إلى مصفوفتين ثانويتين.

في المصفوفات الثنائية الأبعاد توجد طريقتين أيضا لإعطائها قيم سابقة، الطريقة الأولى وضعنها في المثال السابق، أما الطريقة الثانية فهي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr2d[2][2] = {{10, 20}, {30, 40}};
6:    int i,j;
7:
8:    for(i=0;i<=1;i++)
9:    {
10:         for(j=0;j<=1;j++)
11:         {
12:               printf("arr2d[%d][%d] = %d\n", i, j, arr2d[i][j]);
13:         }
14:   }
15:
16:}
```

و هي مشابها لطريقة استعمالها في المصفوفات ذات بوعد واحد.

2.3.2 المصفوفات الثلاثية الأبعاد:

الآن يمكن حتى عمل مصفوفات ذات أربعة أو خمسة أبعاد، بنفس الطرق السابقة، فمثلا إذا أردنا وضع مصفوفة ثلاثية الأبعاد و إعطاء لها قيم سابقة أو ترك المستخدم يضع لها قيم فسنكتب الآتي:

في حالة إعطاء قيم سابقة للمصفوفات نكتب:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr3d[2][2][2] = {{{10, 20},{30, 40}}, {{50, 60},{70, 80}}};
6:    int i, j, k;
7:
8:    for(i=0;i<=1;i++)
9:    {
10:         for(j=0;j<=1;j++)
11:         {
12:               for(k=0;k<=1;k++)
13:               {
14:                     printf("arr3d[%d][%d][%d]    =    %d\n",    i,    j,    k,
arr3d[i][j][k]);
15:               }
16:         }
17:   }
18:
19:}
```

و في حالة ترك المستخدم يقوم بإدخال القيم نكتب:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr3d[2][2][2];
6:    int i, j, k;
7:
8:    for(i=0;i<=1;i++)
9:    {
10:         for(j=0;j<=1;j++)
11:         {
12:               for(k=0;k<=1;k++)
13:               {
14:                     printf("arr3d[%d][%d][%d] : ", i, j, k);
15:                     scanf("%d", &arr3d[i][j][k]);
16:               }
17:         }
18:   }
19:
20:   for(i=0;i<=1;i++)
21:   {
22:         for(j=0;j<=1;j++)
23:         {
24:               for(k=0;k<=1;k++)
25:               {
26:                     printf("arr3d[%d][%d][%d]    =    %d\n",    i,    j,    k,
arr3d[i][j][k]);
```

```
27:                 }
28:             }
29:     }
30:
31:}
```

2.3.3 مصفوفة ذات حجم غير معروف:

يمكننا إنشاء مصفوفة ذات حجم غير معروف، حيث ستكون المصفوفة ديناميكية الحجم أي أن حجم المصفوفة سيزيد حسب الطلب، و لكن من شروط المصفوفات الديناميكية يجب أن تكون القيم معطاة سابقا أي لا يمكن للمستخدم إدخال قيم في مصفوفة مجهولة الحجم.

سأعطي أمثلة حول المصفوفات ذات حجم غير معروف، ثم نناقشها:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[] = {10, 20, 30};
6:
7:    printf("%d\n", arr[0]);
8:    printf("%d\n", arr[1]);
9:    printf("%d\n", arr[2]);
10:}
```

في السطر الخامس:

```
5:    int arr[] = {10, 20, 30};
```

هي الطريقة الوحيدة التي يمكن استعمالها حاليا، و لا يمكن استعمالها في مصفوفات ثانوية، أي مثلا في المصفوفات الثلاثية الأبعاد لا يمكن استعمال مثل الآتي:

```
int arr3d[][][] = {{{10, 20},{30, 40}}, {{50, 60},{70, 80}}};
```

الإمكانيات الوحيدة في المصفوفات الثنائية و الثلاثية الأبعاد أو أكثر هي وضع المصفوفة الرئيسية بدون حجم، و كي يصبح المثال السابق صحيح نكتب:

```
int arr3d[][2][2] = {{{10, 20},{30, 40}}, {{50, 60},{70, 80}}};
```

2.3.4 السلاسل الحرفية (النصوص):

الآن سنعرف طريقة التعامل مع النصوص، باستخدام المصفوفات، و تسمى بسلاسل من حروف.

و تسمى بسلاسل حرفية لأنها في الحقيقة هي عبارة عن سلاسل بها أماكن و كل مكان به حرف، رمز، أو رقم، توجد طرق كثيرة لتعامل مع السلاسل الحرفية باستخدام المصفوفات منها:

إعطاء حجم لمصفوفة من نوع char حيث يكون حجمها هو الحد الأقصى لعدد الحروف التي يمكن إدخالها، مثال توضيحي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char text[14] = "Hello, World!";
6:
7:    printf("%s\n", text);
8:}
```

أو يمكن تجزئة النص كما يلي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char text[14];
6:
7:    text[0] = 'H';
8:    text[1] = 'e';
9:    text[2] = 'l';
10:   text[3] = 'l';
11:   text[4] = 'o';
12:   text[5] = ',';
13:   text[6] = ' ';
14:   text[7] = 'w';
15:   text[8] = 'o';
16:   text[9] = 'r';
17:   text[10] = 'l';
18:   text[11] = 'd';
19:   text[12] = '!';
20:   text[13] = '\0';
21:
22:   printf("%s\n", text);
23:}
```

في السطر العشرين قمنا بإضافة الرمز \0 و الذي يعني نهاية السلسلة، و عدم وجوده سيعطي نتائج غير مرغوب فيها، أما في الطريقة الأولى فيكفي أن نعطيه مكان إضافي و سيتم إضافته تلقائيا، و إذا انتبهت إلى المثال السابق في:

```
5:    char text[14] = "Hello, World!";
```

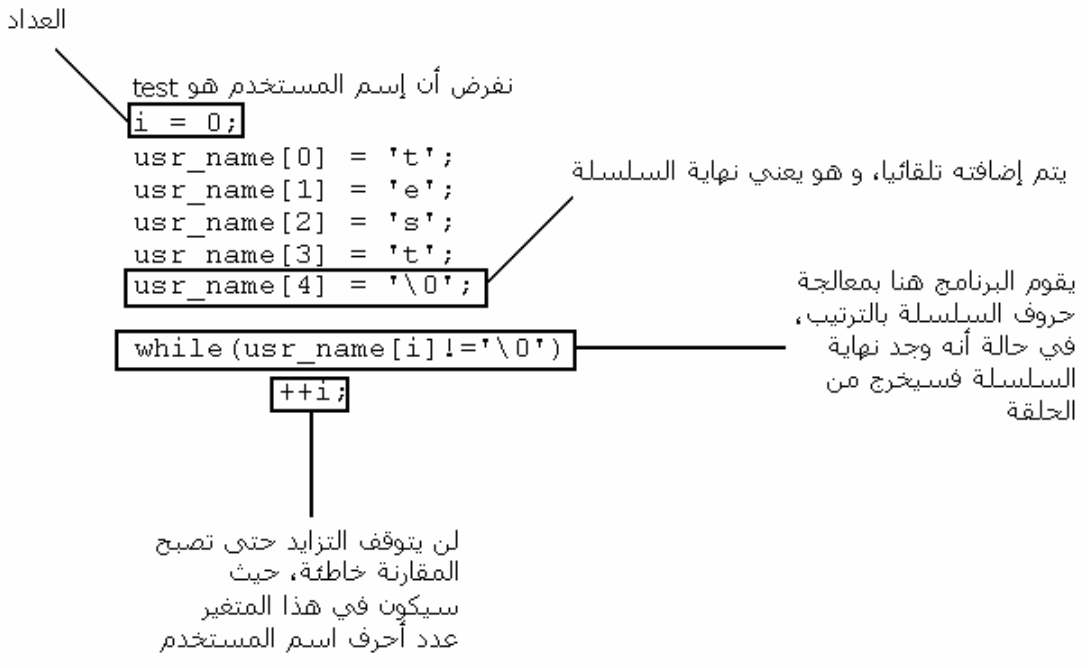
فستلاحظ أن عدد الأحرف هو *13* (الفراغ يعتبر حرف) و نحن قمنا بحجز *14* مكان، المكان الرابع عشر سيكون لرمز \0 حيث سيتم إضافته تلقائيا.

الآن سنقوم بكتابة برنامج يطلب من المستخدم إدخال اسمه، ثم يطبع له البرنامج عدد أحرف اسمه:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char usr_name[255];
6:    int i;
7:    i = 0;
8:
9:    printf("Your name: ");
10:   scanf("%s", &usr_name);
11:
12:   while(usr_name[i] != '\0')
13:   {
14:         ++i;
15:   }
16:
17:   printf("%s = %d characters\n", usr_name, i);
18:}
```

شرح البرنامج هو:

في السطر الخامس قمن بالإعلان عن سلسلة حرفية باسم usr_name، بحجم 255 مكان و هو أقصى احتمال يمكن الوصول إليه، حيث ستكون تلك السلسلة هي المكان الذي سنضع فيه اسم المستخدم، و في السطر السادس قمن بالإعلان عن متغير و الذي سيكون العداد لاسم المستخدم، ثم أعطيناه القيمة صفر في السطر السابع، ثم طلبنا من المستخدم إدخال اسمه في السطر التاسع، و خذنا اسم المستخدم في السطر العاشر حيث ستجد الرمز %s و الحرف s مختصر من *string* و هو الرمز الخاص بالسلاسل الحرفية، و في السطر الثاني عشر قمنا بإنشاء حلقة لا تنتهي إلا بانتهاء اسم المستخدم، و كي تفهم هذه الأخيرة إليك الصورة التالية:



و في السطر السابع عشر يقوم البرنامج بطباعة اسم المستخدم و عدد أحرفه.

و إذا أردت جعل البرنامج أكثر مرونة و أكثر تعمق جرب المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char usr_name[255];
6:    int i;
7:    i = 0;
8:
9:    printf("Your Fullname: ");
10:   scanf("%s", &usr_name);
11:
12:   while(usr_name[i]!='\0')
13:   {
14:         printf("%i: %c\n", i+1, usr_name[i]);
15:         ++i;
16:   }
17:
18:   printf("%s = %d characters\n", usr_name, i);
19:}
```

2.3.4.1 الدالة gets:

تحدثنا عن الدالة gets في الفصل الأول، و لكننا لم نتحدث عن طريقة استعمالها.

هي خاص بإدخال النصوص حيث بدل استعمال scanf("%s", &string_name) نستعمل gets(string_name) أفضل، مثال:

```
1:#include<stdio.h>
2:
```

```
3:main()
4:{
5:    char usr_name[255];
6:
7:    puts("Your name:");
8:    gets(usr_name);
9:
10:   puts("nice to meet you");
11:   puts(usr_name);
12:}
```

و الفرق بين الدالة scanf و الدالة gets هو عند إدخال الأسماء، في الدالة gets إن كتبت اسمين و فصلت بينهما بفراغ فسيقوم بطباعة كلاهما، أما في الدالة scanf فإنه ستتوقف عند الفراغ الأول و تقوم بطبع ما هو قبل الفراغ لا أكثر.

2.3.4.2 الدالة strcpy و الدالة strncpy:

الدالة strcpy من دوال الملف الرأسي string.h حيث به مجموعة من الدوال الخاصة بالتعامل مع السلاسل الحرفية، و اسم الدالة مختصر من *string copy*، و هي تقوم بنسخ و لصق الحروف من سلسلة إلى أخـــرى، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main()
5:{
6:    char String[] = "String!";
7:    char Empty_String[20];
8:
9:    strcpy(Empty_String, String);
10:
11:   printf("Empty_String = %s\n", Empty_String);
12:}
```

الوسيط الأول من الدالة نقوم بالكتابة فيه اسم السلسلة الحرفية التي نريد أن ننسخ بها النص، و في الوسيط الثاني نكتب السلسلة التي نريد نسخها.

أيضا الدالة strncpy من دوال الملف الرأسي string.h، و هي مثل الدالة السابقة strcpy مـــع إضـــافة بسيطة و هي تحديد عدد الأحرف التي نريد نسخها، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main()
5:{
6:    char String[] = "String!";
7:    char Empty_String[20];
8:
9:    strncpy(Empty_String, String, 3);
```

```
10:
11:   printf("Empty_String = %s\n", Empty_String);
12:}
```

في الوسيط الثالث من الدالة strncpy نقوم بتحديد عدد الأحرف التي نريد نسخها.

و يمكن أيضا كتابة التالي:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main()
5:{
6:    char Empty_String[20];
7:
8:    strcpy(Empty_String, "String!");
9:
10:   printf("Empty_String = %s\n", Empty_String);
11:}
```

2.3.4.3 الدالة strcat و الدالة strncat:

الدالة strcat من دوال الملف الرأسي string.h، و هي مختصرة من *string concatenate*، و هي تقـــوم نسخ نص و إضافته في نهاية سلسلة حرفية، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main()
5:{
6:    char String[20] = "String!";
7:
8:    strcat(String, ", String2");
9:
10:   printf("String = %s\n", String);
11:}
```

في الوسيط الأول من الدالة strcat نقوم بكتابة اسم السلسلة التي سنضيف إليها النص، و في الوسيط الثاني نقوم بكتابة النص.

الدالة strncat مثل الدالة strcat مع إضافة بسيطة و هي تحديد عدد الأحرف التي نريد نسخها، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main()
5:{
6:    char String[20] = "String!";
7:
8:    strncat(String, ", String2", 3);
9:
10:   printf("String = %s\n", String);
```

```
11:}
```

في الوسيط الثالث نقوم بتحديد عدد الأحرف التي نريد نسخها.

2.3.5 طريقة أخرى لتعامل مع المصفوفات:

يمكننا كتابة `*(arr+0) = 10` في مكان `arr[0] = 10`، حيث أنها مكافئة للسابقة، و هذا مثال طبيعي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[1];
6:    arr[0] = 10;
7:
8:    printf("%d\n", arr[0]);
9:}
```

و هذا المثال السابق باستعمال الطريقة الثانية:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[1];
6:    *(arr+0) = 10;
7:
8:    printf("%d\n", *(arr+0));
9:}
```

الشفرة `arr[0]` هي نفسها الشفرة `*(arr+0)`.

و يمكن استعمال مؤثرات مثل الجمع، الطرح، القسمة و الظرب للإشارة إلى عنصر من مصفوفة مثل:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[2];
6:
7:    arr[0+0] = 10;
8:    printf("%d\n", arr[0+0]); /* or printf("%d\n", arr[0]); */
9:
10:   arr[0-0] = 20;
11:   printf("%d\n", arr[0-0]); /* or printf("%d\n", arr[0]); */
12:
13:   arr[1*1] = 30;
14:   printf("%d\n", arr[1*1]); /* or printf("%d\n", arr[1]); */
15:
16:   arr[1/1] = 40;
17:   printf("%d\n", arr[1/1]); /* or printf("%d\n", arr[1]); */
18:}
```

2.3.6 الأخطاء المحتملة:

1- نفلاض أنه لديك مصفوفة بحجم *2* و اسمها arr، ثم أردت وضع قيم للمصفوفة و قمت بكتابة التالي:

```
int arr[2] ;
arr[0] = 10 ;
arr[8] = 20 ;
```

فهنا سيحدث انتهاك في الذاكرة، و ربما يؤدي إلى توقف البرنامج عن العمل.
و الطرقة الصحيحة للمثال السابق هي:

```
int arr[2] ;
arr[0] = 10 ;
arr[1] = 20 ;
```

2- في السلاسل الحرفية عند كتابة نص مباشرة بعد الإعلان عنها فيجب عليك دائما وضع مكان إضافي لرمز \0 و إلا ستكون النتائج غير صحيحة.

3- الدالة gets لا تستعمل إلا في إدخال النصوص.

2.3.7 تمارين:

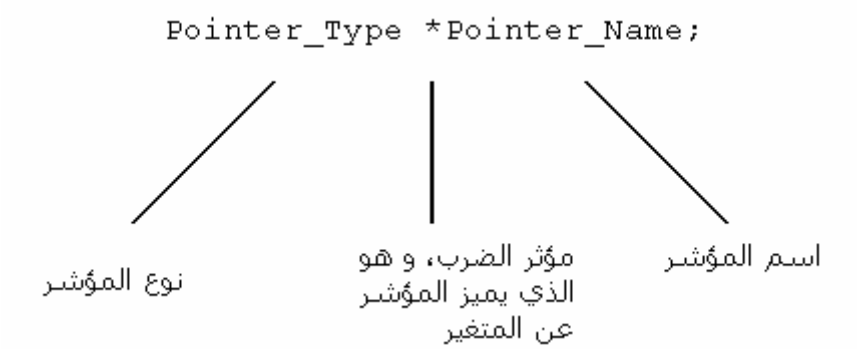
1- أكتب برنامج يطلب من المستخدم إدخال اسمه، ثم يطبع له البرنامج عدد أحرف اسمه(باستخدام الدالتين puts و gets).
2- أكتب برنامج يطلب من المستخدم إدخال أعداد حقيقية، ثم يقوم البرنامج بإعادة طبعها (باستخدام المصفوفات).
3- أكتب برنامج به مصفوفة بها النص *Hello, World !* ثم قم بعمل حلقة تقوم بعد عدد رموز أو أحرف تلك الجملة(إذا وجدت *13* حرفا فإن النتيجة خاطئة) يجب أن تكون النتيجة *12* مع طريقة استعمال سليمة يعني بدون استعمال *-1*.

2.4 المؤشرات pointers

كلمة مؤشر تعني الإشارة إلى شيء، و في لغة C المؤشرات تشير إلى عناوين في الذاكرة.

طريقة الإعلان عن مؤشر مماثلة لطريقة الإعلان عن المتغيرات، و المؤشرات هي نفسها المتغيرات، و لكن المؤشرات أفضل منها و هي من ميزات لغة C.

صورة توضح لطريقة الإعلان عن مؤشر:



الفرق الوحيد بين الإعلان عن متغير و الإعلان عن مؤشر هو مؤثر الضرب في المؤشرات و الذي يكون قبل اسم المؤشر.

2.4.1 نوع المؤشر Pointer Type:

للمؤشرات أنوان هي نفسها أنواع المتغيرات،و هي `int, float, double, long, short, unsigned, signed, char,`.

2.4.2 اسم المؤشر Pointer Name:

لاسم المؤشر شروط هي نفسها شروط المتغيرات و هي:

- أن لا يتجاوز اسم المؤشر أكثر من *31* حرف.
- أن لا يبدأ اسم المؤشر بأرقام.
- أن لا يكون اسم المؤشر يحتوي على مؤثرات مثل الجمع، الطرح، ....
- أن لا يكون اسم المؤشر يحتوي على رموز مثل % و # و } و...(باستثناء الرمز _).
- أن لا يكون اسم المؤشر مستعمل سابقا في دالة أو متغير أو مؤشر آخر.

§ أن لا يكون اسم المؤشر من أسماء الكلمات المحجوزة.

المؤشرات تحمل عناوين في الذاكرة و لا يمكن أن نعطيها قيم مباشرة إلا في حالات، و في العناوين نجد قيم و لكل عنوان قيمته، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int *ptr, i;
6:    i = 10;
7:    ptr = &i;
8:
9:    printf("*ptr = %d\n", *ptr);
10:}
```

المؤشر موجود في السطر الخامس مع متغير، اسم المؤشر هو ptr و اسم المتغير هو i، أعطينا للمتغير i القيمة *10* في السطر السادس، و في السطر السابع أعطينا للمؤشر ptr عنوان المتغير i أي أنه عند كتابة &Variable_Name فهذا يعني عنوان متغير، و أخيرا السطر التاسع حيث كتبتا *ptr و الذي يعني القيمة الموجود في عنوان المؤشر ptr، و في حالة أننا كتبنا اسم المؤشر بدون مؤثر الضرب فسيتم طباعة عنوان المؤشر ptr.

في المثال السابق إن قمنا بتغير القيمة الموجودة في عنوان المؤشر ptr فستتغير القيمة الموجود في المتغير i، لأننا أخذنا عنوان المتغير i و الذي هو نفسه عنوان المؤشر ptr، جرب المثال السابق بهذه الطريقة:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int *ptr, i;
6:    i = 10;
7:    ptr = &i;
8:    *ptr = 100;
9:
10:    printf("*ptr = %d\n", *ptr);
11:}
```

هنا يمكننا إعطاء قيمة للمؤشر ptr لأنه لديه عنوان و هو عنوان المتغير i و سيقوم بحذف القيمة السابقة الموجود في عنوان i و يقوم بتحديثها إلى العدد *100*.

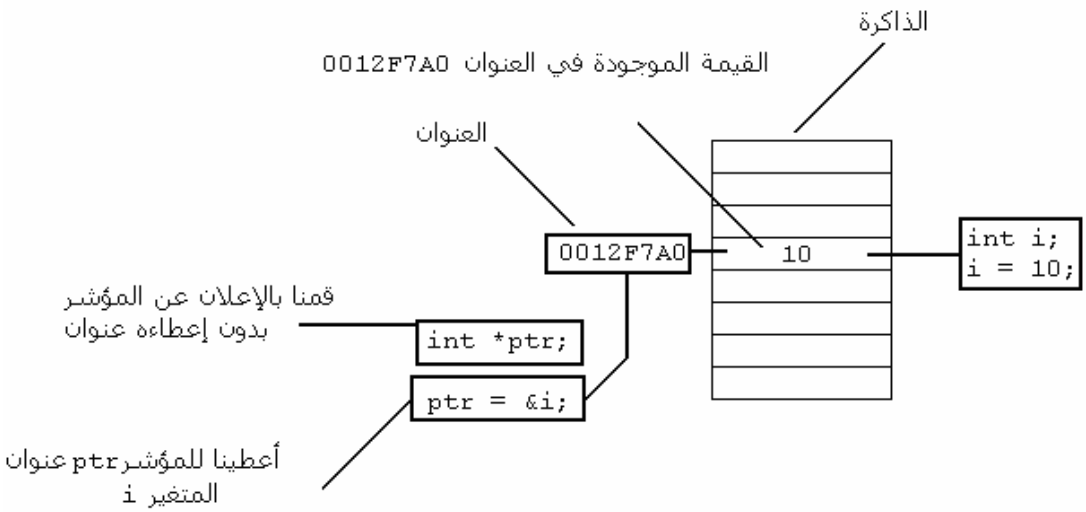
و لكي تفهم المؤشرات جيدا جرب المثال التالي:

```
1:#include<stdio.h>
2:
```

```
3:main()
4:{
5:    int *ptr, i;
6:    i = 10;
7:    ptr = &i;
8:
9:    printf("i = %d\n", i);
10:   printf("*ptr = %d\n", *ptr);
11:   printf("&i = %p\n", &i);
12:   printf("ptr = %p\n", ptr);
13:}
```

في السطر التاسع و العاشر قمنا بطباعة كل من القيمة الموجودة في المتغير i و القيمة الموجـــود في عنـــوان المؤشر ptr و لكي نرى قيمة موجودة داخل مؤشر نكتب مؤثر الضرب قبل اسمه و أيضا يمكننا وضع قيمة له بنفس الطريقة (يجب أن يكون لديه عنوان كي نستطيع وضع له قيمة)، و في السطر الحـــادي عـــشر و السطر الثاني عشر قمنا بطباعة كل من عنوان المتغير i و عنوان المؤشر ptr، و تلاحظ أنه وضعنا الرمز %p و هو مختصر من *pointer* و يمكن استعماله مع المؤشرات أو عنوان لمتغير حيث قمنا باستعماله كي يتم طباعة العناوين بشكل صحيح.

صورة توضيحية للمثال السابق:



2.4.3 المؤشرات و المصفوفات:

المؤشرات هي نفسها المصفوفات و لكن الأفضل هي المؤشرات لمرونتها، و كي ترى الشبه الكبير بينـــهما ألقي نظرة على المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[2];
6:    int *ptr;
```

```
7:
8:    arr[0] = 10;
9:    arr[1] = 20;
10:
11:   ptr = &arr[0];
12:
13:   printf("%d\n", ptr[0]);
14:   printf("%d\n", ptr[1]);
15:}
```

قمنا بالإعلان عن مصفوفة بحجم 2، ثم قمنا بالإعلان عن مؤشر، و في السطر الثامن و التاسع أعطينا قـــيم للمصفوفة، و في السطر الحادي عشر أعطينا للمؤشر ptr عنوان بداية المصفوفة، و في السطر الثالث عشر و الرابع عشر قمنا بطباعة ما هو موجود في عنوان المؤشر ptr.

و يمكننا كتابة المثال السباق بهذه الطريقة:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[2];
6:    int *ptr;
7:
8:    *(arr+0) = 10;
9:    *(arr+1) = 20;
10:
11:   ptr = &*(arr+0);
12:
13:   printf("%d\n", *(ptr+0));
14:   printf("%d\n", *(ptr+1));
15:}
```

و يمكن أيضا كتابته بالطريقة التالية:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[2];
6:    int *ptr;
7:
8:    arr[0] = 10;
9:    arr[1] = 20;
10:
11:   ptr = &arr[0];
12:
13:   printf("%d\n", *ptr);
14:   printf("%d\n", *++ptr);
15:}
```

و هذا مثال لا يمكن تطبيقه على المصفوفات:

```
1:#include<stdio.h>
2:
```

```
3:main()
4:{
5:    int arr[2];
6:    int *ptr;
7:
8:    arr[0] = 10;
9:    arr[1] = 20;
10:
11:   ptr = &arr[0];
12:
13:   printf("%d\n", arr);
14:   printf("%d\n", ++arr);
15:}
```

2.4.4 التعامل مع النصوص:

التعامل مع النصوص باستخدام المؤشرات مشابه للتعامل مع النصوص باستخدام المصفوفات،مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char *str = "Hello, World!";
6:
7:    printf("%s\n", str);
8:}
```

أما إذا أردنا استخدام المؤشرات في الإدخال فيوجد شروط يجب التقيد بها و إلا ستحدث أخطاء ربما في المصدر البرنامج أو أثناء تشغيل البرنامج، فمثلا لا يمكننا استعمال مثل ما هو في المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char *str;
6:
7:    printf("Enter a string: ");
8:    scanf("%s", str);
9:    printf("%s\n", str);
10:}
```

في المترجمات الجديد ربما لن يسمح لك المترجم باستعمال هذه الطريقة، و السبب في ذلك هو عدم تحديد للمؤشر str عنوانا، أي أنه بلا عنوان، و لكي تصبح طريتنا صحيح فيجب على الأقل الإعلان عن متغير حرفي و نعطي للمؤشر str عنوان ذلك المتغير و الذي سيكون بداية المؤشر str و سيصبح المثال على الشكل التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char *str;
6:    char adr_str;
7:
```

```
8:    str = &adr_str;
9:
10:   printf("Enter a string: ");
11:   scanf("%s", str);
12:
13:   printf("%s\n", str);
14:}
```

و ستلاحظ في السطر الحادي عشر قمنا بكتابة str بدون الرمز &، ذلك لأننا استعملنا مؤشـــر و الوضـــع الافتراضي للمؤشرات هي عنوانها.

و يمكننا أيضا كتابة اسم مصفوفة بدون رمز العنوان & في الدالة scanf، مثال توضيحي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char arr[255];
6:
7:    printf("Enter a string: ");
8:    scanf("%s", arr);
9:
10:   printf("%s\n", arr);
11:}
```

و من إمكانيات المؤشرات أنه يمكن أن نعطيه قيم مصفوفة باستخدام:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int arr[3] = {10, 20, 30};
6:    int *ptr;
7:    int i;
8:
9:    ptr = arr;
10:
11:   for(i=0;i<=2;i++)
12:         printf("ptr[%d] = %d\n", i, ptr[i]);
13:}
```

و لا يمكننا كتابة العكس، arr = ptr.

2.4.5 المرجع reference:

المرجع هو أخذ عنوان متغير، عند كتابة:

```
1:#include<stdio.h>
2:
3:main(){
4:    int ref;
5:    int *ptr;
6:    ref = 10;
```

```
7:    ptr = &ref;
8:}
```

هنا الـــ ptr هو المؤشر، و الـــ &ref هي المرجع.

كتابة المؤثر & ثم اسم متغير يعني أخذ عنوان ذلك المتغير، و تسمى هذه العملية بالمرجع *reference*.

2.4.6 مؤشر لـــ void:

لا يمكننا الإعلان عن متغير باستخدام الكلمة المحجوزة void، وسبب في ذلك أنه ليس لها حجم كي يـــتم الحفظ فيه القيم المرادة، و لكننا يمكن أن نقوم بالإعلان عن مؤشر لـــ void حيث يعتبر الأكثر مرونة مـــع المؤشرات، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    void *p_void;
6:    /* can't use void p_void; */
7:
8:    (int)p_void = 10;
9:    printf("(int)p_void = %d\n", p_void);
10:
11:   (char)p_void = 'H';
12:   printf("(char)p_void = %c\n", p_void);
13:
14:   (char *)p_void = "Hello";
15:   printf("(char *)p_void = %s\n", p_void);
16:}
```

عندما نريد وضع قيم لمؤشر void نكتب (Type)Pointer_Name، في مكان Type نكتب نوع القيمة الـــتي سنقوم بإدخالها، ثم اسم المؤشر ثم نعطيه القيمة.

2.4.7 مؤشر لمصفوفة:

عندما نعلن عن مؤثر لمصوفة فهذا يعني أن كل عنصر من تلك المصفوفة يمكن أي يحمل عنوانا، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int *arr[2];
6:    int a = 10, b = 20;
7:
8:    printf("A = %d, ", a);
9:    printf("B = %d\n", b);
10:
11:   arr[0] = &a;
12:   *arr[0] = 5;
13:
```

```
14:   arr[1] = &b;
15:   *arr[1] = 10;
16:
17:   printf("A = %d, ", a);
18:   printf("B = %d\n", b);
19:}
```

و يمكن استعمال مؤشر لمصفوفة ثنائية أو ثلاثية الأبعاد. و يمكن أيضا استعمال مؤشر لسلسلة حرفية حيث كل عنصر من تلك السلسلة يمكن أن يحمل نص مثل:

```
1:#include<stdio.h>
2:
3:main(){
4:    char *arr[] = {"Text 1", "Text 2", "Text 3"};
5:
6:    printf("arr[0] = %s\n", arr[0]);
7:    printf("arr[1] = %s\n", arr[1]);
8:    printf("arr[2] = %s\n", arr[2]);
9:}
```

2.4.8 مؤشر لمؤشر:

مؤشر لمؤشر قليلة الاستعمال. مؤشر لمتغير يعني أن نشير إلى عنوان المتغير، و مؤشر لمؤشر يعني أن نشير إلى عنوان مؤشر، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int p;
6:    int *pt;
7:    int **ptr;
8:
9:    p = 10;
10:   printf("p = %d\n", p);
11:
12:   pt = &p;
13:   ptr = &pt;
14:   **ptr = 5;
15:
16:   printf("p = %d\n", p);
17:}
```

تم الإعلان عن مؤشر لمؤشر في السطر السابع.

و يمكن أن نقوم بإعلان عن مؤشر لمؤشر من نوع الحرفي حيث كل مؤشر يمكن أن يحمل سلسلة من حروف مثل:

```
1:#include<stdio.h>
2:
3:main(){
4:    char *AdrPtr;
```

```
5:    char **ptr = &AdrPtr;
6:
7:    ptr[0] = "Text 1";
8:    ptr[1] = "Text 2";
9:    ptr[2] = "Text 3";
10:
11:   printf("ptr[0] = %s\n", ptr[0]);
12:   printf("ptr[1] = %s\n", ptr[1]);
13:   printf("ptr[2] = %s\n", ptr[2]);
14:}
```

و يمكن عمل أكثر من مؤشر لمؤشر، أي يمكن الإعلان عن int *****ptr، أو أكثر.

2.4.9 **الأخطاء المحتملة:**

1- في المترجمات الجديد لا يسمح بإعطاء قيمة لمؤشر بدون عنوان، أي لا يمكن كتابة كما في المثال التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int *ptr;
6:    *ptr = 10;
7:
8:    printf("%d\n", *ptr);
9:}
```

و أيضا لا يمكن كتابة:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int *ptr;
6:    int i;
7:    *ptr = i;
8:
9:}
```

2- لا يمكن إعطاء عنوان لمتغير طبيعي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int *ptr;
6:    int i;
7:    i = ptr;
8:
9:}
```

2.4.10 **تمارين:**

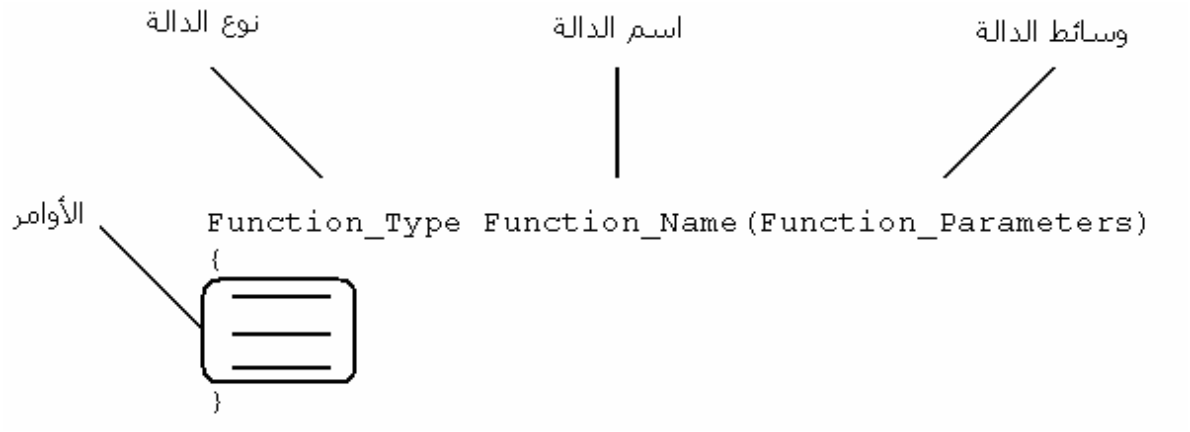
1- أكتب البرنامج التالي باستخدام المؤشرات:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    char usr_name[255];
6:    int i;
7:    i = 0;
8:
9:    printf("Your Fullname: ");
10:   scanf("%s", &usr_name);
11:
12:   while(usr_name[i]!='\0')
13:   {
14:         printf("%i: %c\n", i+1, usr_name[i]);
15:         ++i;
16:   }
17:
18:   printf("%s = %d characters\n", usr_name, i);
19:}
```

2.5 الدوال Functions

الدوال هي مجموعة من الأوامر و البيانات نعطي لها اسم حيث يمكن استدعاءها من أماكن مختلفة في البرنامج. و من فوائدها تقسيم مصدر البرنامج إلى أجزاء تكون مفهومة و منظمة.

توجد طريقتين للإعلان عن الدوال، أولا ألقي نظرة على الصورة التالية:



و يمكن أن لا تحتوي الدالة على وسائط.

إذا أردنا عمل دالة تقوم بطباعة الجملة *Hello, World !* فسيكون برنامجنا كالتالي:

```
1:#include<stdio.h>
2:
3:void Func_HelloWorld()
4:{
5:    printf("Hello, World!\n");
6:}
7:
8:main()
9:{
10:   Func_HelloWorld();
11:}
```

هذه طريقة، أما الطريقة الثانية فهي الإعلان عن الدالة (يسمى بالنموذج *prototype*) ثم نقوم بإعطائها الأوامر بعد الدالة الرئيسية و سيصبح المثال السابق كالتالي:

```
1:#include<stdio.h>
2:
3:void Func_HelloWorld();
4:
5:main()
6:{
7:    Func_HelloWorld();
8:}
9:
```

```
10:void Func_HelloWorld()
11:{
12:   printf("Hello, World!\n");
13:}
```

و هي الطريقة الأفضل من حيث التنظيم.

و توجد طريقة أخرى و لكن لا يفضل استعمالها من ناحية التنظيم و أيضا بعض المترجمات لا تقبلها و هي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    Func_HelloWorld();
6:}
7:
8:void Func_HelloWorld()
9:{
10:   printf("Hello, World!\n");
11:}
```

و إن كان مترجمك قد نبهك عن وجود خطأ فسيكون عن الخطأ عن نموذج *prototype* الدالة Func_HelloWorld، و لك في الأصل هذه الطريقة هي من طرق لغة *C*، يعني أنها طريقة صحيحة فقط بعض المترجمات لا تدعمها.

الكلمة المحجوزة void تستعمل مع الدوال حيث حجمها *0* بايت، و هي لا تقوم بإرجاع أي قيم.

سنقوم بكتابة دالة بها وسيط عبارة عن سلسلة حروف ثابتة، حيث ستقوم تلك الدالة بطباعة ما هو داخل الوسيط الخاص بها:

```
1:#include<stdio.h>
2:
3:void Func_Print(const char *str);
4:
5:main()
6:{
7:    Func_Print("Func_Print:\n");
8:    Func_Print("Hello, World!\n");
9:}
10:
11:void Func_Print(const char *str)
12:{
13:   printf("%s", str);
14:}
```

هذه ليست إلا أمثلة بسيطة حول طريقة عمل الدوال، يمكن أن نقوم بعمل دالة تقوم بـــالجمع، الطـــرح، القسمة و الضرب، سيكون المثال على الشكل التالي:

```
1:#include<stdio.h>
2:
3:void Func_Add(const int num1, const int num2);
4:void Func_Sub(const int num1, const int num2);
5:void Func_Mul(const int num1, const int num2);
6:void Func_Div(const int num1, const int num2);
7:
8:main()
9:{
10:   Func_Add(30, 10);
11:   Func_Sub(30, 10);
12:   Func_Mul(30, 10);
13:   Func_Div(30, 10);
14:}
15:
16:void Func_Add(const int num1, const int num2)
17:{
18:   printf("%d + %d = %d\n", num1, num2, num1+num2);
19:}
20:
21:void Func_Sub(const int num1, const int num2)
22:{
23:   printf("%d - %d = %d\n", num1, num2, num1-num2);
24:}
25:
26:
27:void Func_Mul(const int num1, const int num2)
28:{
29:   printf("%d * %d = %d\n", num1, num2, num1*num2);
30:}
31:
32:
33:void Func_Div(const int num1, const int num2)
34:{
35:   printf("%d / %d = %d\n", num1, num2, num1/num2);
36:}
```

و يمكن أن نجعل هذا البرنامج أكثر مرونة بالطريقة التالية:

```
1:#include<stdio.h>
2:
3:void Func_(const int num1, const char sign, const int num2);
4:
5:main()
6:{
7:    Func_(30, '+', 10);
8:    Func_(30, '-', 10);
9:    Func_(30, '*', 10);
10:   Func_(30, '/', 10);
11:}
12:
13:void Func_(const int num1, const char sign, const int num2)
14:{
15:   switch(sign)
16:   {
17:   case '+':
```

```
18:         printf("%d %c %d = %d\n", num1, sign, num2, num1+num2);
19:         break;
20:   case '-':
21:         printf("%d %c %d = %d\n", num1, sign, num2, num1-num2);
22:         break;
23:   case '*':
24:         printf("%d %c %d = %d\n", num1, sign, num2, num1*num2);
25:         break;
26:   case '/':
27:         printf("%d %c %d = %d\n", num1, sign, num2, num1/num2);
28:         break;
29:
30:   default:
31:         printf("ERROR!\n");
32:   }
33:}
```

2.5.1 نوع الدالة Function Type:

لـــ الدوال أنواع و هي نفسها أنواع المتغير، يمكن استعمال دالة من نوع أعداد صـــحيحة int أو أعـــداد حقيقية float.

بالنسبة لدوال من نوع أعداد صحيحة فهي لها قيمة تقوم بإرجاعها، أي في نهاية الدالة نقوم بإرجاع قيمـــة باستخدام الكلمة المحجوزة return كما في المثال التالي:

```
1:#include<stdio.h>
2:
3:int Int_Func(const int num);
4:
5:main()
6:{
7:    printf("%d\n", Int_Func(5));
8:}
9:
10:int Int_Func(const int num)
11:{
12:   return num;
13:}
```

هنا قمنا بإرجاع قيمة الوسيط int num إلى الدالة Int_Func، و في السطر السابع، في الدالة printf يمكننا كتابة دوال التي تقوم بإرجاع قيم كما في هذا المثال، و لا يمكن استعمال هذه الطريقة مع الكلمة المحجوزة void لأنها بدون حجم و لا يمكنها حمل قيم.

يمكن كتابة الدالة بدون نوع بالنسبة للمثال السابقة، لأن دالتنا من نوع أعداد صحيحة، و في لغة C الوضع الافتراضي لدوال بدون اسم هو int و المثال السابق سيصبح على الشكل التالي:

```
1:#include<stdio.h>
2:
3:Int_Func(const int num);
4:
```

```
5:main()
6:{
7:    printf("%d\n", Int_Func(5));
8:}
9:
10:Int_Func(const int num)
11:{
12:   return num;
13:}
```

و يمكن أيضا كتابة اسم متغير بدون نوع، ثم نقوم بكتابة نوعه أسفل اسم الدالة التي هي بعد الدالة الرئيسية، مثال:

```
1:#include<stdio.h>
2:
3:Int_Func(const int num);
4:
5:main()
6:{
7:    printf("%d\n", Int_Func(5));
8:}
9:
10:Int_Func(num)
11:const int num;
12:{
13:   return num;
14:}
```

و يمكن استعمال المثال:

```
1:#include<stdio.h>
2:
3:int Int_Func(const int num);
4:
5:main()
6:{
7:    printf("%d\n", Int_Func(5));
8:}
9:
10:int Int_Func(const int num)
11:{
12:   return num;
13:}
```

باستخدام short:

```
1:#include<stdio.h>
2:
3:short Short_Func(const short num);
4:
5:main()
6:{
7:    printf("%d\n", Short _Func(5));
8:}
9:
10:short Short _Func(const short num)
11:{
```

```
12:   return num;
13:}
```

و باستخدام كل من long، float، double، signed، unsigned بنفس الطريقة، أما char فهـــذا مثـــال لطريقة استعمالها:

```
1:#include<stdio.h>
2:
3:char Char_Func(const char ch);
4:
5:main()
6:{
7:    printf("%c\n", Char_Func('C'));
8:}
9:
10:char Char_Func(const char ch)
11:{
12:   return ch;
13:}
```

و الدوال التي ترجع قيم يمكن إعطاءها لمتغير طبيعي مباشرة مثل:

```
1:#include<stdio.h>
2:
3:int Int_Func(const int num);
4:
6:main()
7:{
8:    int i = Int_Func(5);
9:
10:   printf("i = %d\n", i);
11:}
12:
13:int Int_Func(const int num)
14:{
15:   return num*2;
16:}
```

و يمكن أيضا عمل دالة تقوم بإرجاع سلسلة حرفية باستخدام المؤشرات، مثال:

```
1:#include<stdio.h>
2:
3:char *string(char *str){
4:    return str;
5:}
6:
7:main()
8:{
9:    printf("%s\n", string("Hello, World!"));
10:}
```

2.5.2 **اسم الدالة** Function Name:

لاسم الدالة حدود و هي مثل اسم المتغير:

- أن لا يتجاوز اسم الدالة *31* حرف.
- أن لا يبدأ اسم الدالة بأرقام.
- أن لا يكون اسم الدالة يحتوي على مؤثرات مثل الجمع، الطرح، ....
- أن لا يكون اسم الدالة يحتوي على رموز مثل % و # و } و...(باستثناء الرمز _).
- أن لا يكون اسم الدالة مستعمل سابقا في متغير أو دالة أخرى.
- أن لا يكون اسم الدالة من أحد أسماء الكلمات المحجوزة.

2.5.3 وسائط الدالة Function Parameters:

الوسائط هي متغيرات نقوم بوضعها على حسب متطلباتنا حيث تكون من الأنواع `unsigned , signed, short, long, int, float, double, char, char*, char[]`.
و يمكن أيضا أن تكون الوسائط عبارة عن مصفوفات أو مؤشرات من كل الأنواع، أو ثوابت و بنيات، أو حتى دوال أخرى.

2.5.4 الأوامر:

الأوامر يمكن كتابتها بحرية مثل ما نكتبها على الدالة الرئيسية `main`.

2.5.5 الدوال باستخدام الموجه #define:

لا تسمى بدوال و لكنها شبيها لها، تسمى بالـــ *macros* أي المختصرة، و هذا مثال لطريقة الإعلان عنها:

```
1:#include<stdio.h>
2:
3:#define Add(a, b) (a+b)
4:
5:main()
6:{
7:    int a, b;
8:    a = 50;
9:    b = 100;
10:
11:   printf("%d + %d = %d\n", a, b, Add(a, b));
12:}
```

و هنا عند استعمال متغيرات لا يمكن كتابة أو تحديد نوعها، سيتم تحديد نوع المتغير تلقائيا مـــن الموجـــه `#define`.

مثال آخر:

```
1:#include<stdio.h>
```

```
2:
3:#define PrintF_(String) printf("%s", String)
4:
5:main()
6:{
7:    PrintF_("Macro...\n");
8:}
```

2.5.6 الفرق بين الإجراء و الدالة:

في لغة C الإجراءات يمكن القول عليها هي نفسها الدوال لأنها مدمج معها، و الإجراء *Procedure* هو دالة لا تقوم بإرجاع قيمة و يمكن القول أن دوال من نوع void تسمى إجراء لأنها لا ترجع قيم مثل الدوال من نوع int أو float أو غيرها، حيث تقوم الإجراءات بتنفيذ أوامر أما الدوال فهي تقوم بعمليات و تعطـــي نتيجة.

2.5.7 دوال لها وسائط من نوع دوال:

يمكننا عمل دوال بها دوال أخرى، و ذلك كتالي:

```
1:#include<stdio.h>
2:
3:void CallFunc(void Func_Name(void));
4:void Function();
5:
6:main(){
      CallFunc(Function);
}

void CallFunc(void Func_Name(void)){
      printf("Call Function:\n");
      Func_Name();
}

void Function(){
      printf("This is a function!\n");
}
```

و يمكن أن تكون الوسائط عبارة عن دوال ترجع قيم، أو دوال بها وسائط أخرى، أو استدعاء دوال أخرى في وسائط الدوال، و أكثر من ذلك. و هذا مثال لطريقة استعمال وسائط دوال لها وسائط أخرى:

```
1:#include<stdio.h>
2:
3:void CallFunc(void Func_Name(int a, int b));
4:void Function(int a, int b);
5:
6:main(){
7:    CallFunc(Function);
8:}
9:
10:void CallFunc(void Func_Name(int a, int b)){
11:   printf("Call Function:\n");
12:   Func_Name(10, 20);
```

```
13:}
14:
15:void Function(int a, int b){
16:   printf("%d + %d = %d\n", a, b, a+b);
17:}
```

2.5.8 الأخطاء المحتملة:

1- لا نضع الفواصل المنقوطة في نهاية دوال نقوم بإعطائها أوامر، مثال:

```
1:#include<stdio.h>
2:
3:void Function();
4:
5:main()
6:{
7:     Function();
8:}
9:
10:void Function();
11:{
12:   /*Empty Funtion;*/
13:}
```

و الخطأ في السطر العاشر، عند الإعلان عن دالة بدون أوامر يجب كتابة الفاصلة المنقوطة في نهاية الدالة، أما عند الإعلان عن دوال لكي نعطي لها أوامر فلا يجب أن نكتب فاصلة منقوطة في نهاية الدالة.

2- لا يمكن الإعلان عن دالتين بنفس الاسم.

2.5.9 تمارين:

1- أكتب دالة تقوم بـــ عد عدد أحرف نص، حيث يكون اسم الدالة `StrLen` و بها الوسيط `const char *str`.

2- أكتب دالة تقوم بإخراج القيمة المطلقة للعدد الذي أدخله المستخدم، اسم الدالة هو `abs` مع الوسيط `int value`.

2.6 الملفات الرأسية Header files

كل من stdio.h و conio.h عبارة عن ملفات رأسية، حيث توجد بها ثوابـــت، نمـــاذج دوال و بنيـــات تساعدنا في برامجنا، سنتحدث في هذا الدرس عن طريقة إنشاء ملفات رأسية و طريقة استعمالها.

سنقوم بكتابة ملف رأسي به دوال لكل من الجمع، الطرح، القيمة و الضرب، أولا نقوم بإنشاء ملف نصي و حفظه على صيغة h.، و يكون اسم ملفنا الرأسي functions.h و نكتب فيه المثال التالي:

```
1:/*Functions Header file*/
2:
3:int Add(int num1, int num2)
4:{
5:    return num1+num2;
6:}
7:
8:int Sub(int num1, int num2)
9:{
10:   return num1-num2;
11:}
12:
13:int Mul(int num1, int num2)
14:{
15:   return num1*num2;
16:}
17:
18:int Div(int num1, int num2)
19:{
20:   return num1/num2;
21:}
```

ثم قم بإنشاء الملف الرأسي للمشروع في نفس المكان الذي قمت بإنشاء فيه الملف الرأسي functions.h، و قم بالكتابة فيه ما يلي:

```
1:#include<stdio.h>
2:#include"functions.h"
3:
4:main()
5:{
6:    int num1 = 30, num2 = 10;
7:
8:    printf("%d + %d = %d\n", num1, num2, Add(num1, num2));
9:    printf("%d - %d = %d\n", num1, num2, Sub(num1, num2));
10:   printf("%d * %d = %d\n", num1, num2, Mul(num1, num2));
11:   printf("%d / %d = %d\n", num1, num2, Div(num1, num2));
12:}
```

في السطر الثاني قمنا بإضافة الملف الرأسي functions.h و لكن بطريقة مختلفة و هي وضع اسم الملف بين إقتباسين و ذلك لأن الملف الرأسي functions.h موجود في نفس المكان الذي موجود به الملف النـــصي الرئيسي main.c، أما إذا أردت كتابة:

```
1:#include<stdio.h>
2:#include<functions.h>
3:
4:main()
5:{
6:    int num1 = 30, num2 = 10;
7:
8:    printf("%d + %d = %d\n", num1, num2, Add(num1, num2));
9:    printf("%d - %d = %d\n", num1, num2, Sub(num1, num2));
10:   printf("%d * %d = %d\n", num1, num2, Mul(num1, num2));
11:   printf("%d / %d = %d\n", num1, num2, Div(num1, num2));
12:}
```

فيجب عليك وضع الملف الرأسي functions.h في نفس المكان الذي موجود به الملف الرأسي stdio.h.

و توجد طريقة أخرى في الملفات الرأسية، و هي إنشاء ملف رأسي و ملف مصدر بصيغة c. بنفس الإسم، فقط يكون الإختلاف في الإمتداد، حيث يمكن عمل نماذج لدوال في الملف الرأسي، ثم أوامـــر الـــدوال في الملف المصدري c.، مثال:

الملف المصدري الرئيسي:

```
1:/*main.c*/
2:#include"func.h"
3:
4:main(){
5:    prints();
6:}
```

الملف الرأسي *func.h*:

```
1:/*func.h*/
2:/*Function prototype*/
3:void prints();
```

الملف المصدري *func.c*:

```
1:/*func.c*/
2:#include<stdio.h>
3:#include"func.h"
4:
5:/*Function*/
6:void prints(){
7:    printf("Hello, World!\n");
8:}
```

2.6.1 اسم الملف الرأسي:

يمكن كتابة أرقام في بداية اسم الملف الرأسي و أيضا يمكن استعمال مؤثرات الجمع و الطرح، و الرموز التي لا يمكن استعمالها في اسم الملف الرأسي هي:

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

و أقصى حد يمكن إعطاءه لاسم الملف الرأسي هو *256* رمز.

2.6.2 متى نستعمل الملفات الرأسية:

تستعمل الملفات الرأسية عند كتابة برامج كبيرة، مثلا إذا كنت تستعمل كثير دوال الرياضيات في برامج فيستحسن أن تقوم بكتابة ملف رأسي باسم `math.h` حيث تضع به جميع الدوال التي تريد استعمالها في المستقبل، و إذا كنت تستعمل دوال الرسم أيضا قم بإنشاء ملف رأسي لها باسم `design.h` أو `graphics.h` حيث تكون به أغلب دوال الرسم، و هكذا حتى تكون لديك مكتبة كبيرة خاصة بك.

2.6.3 الأخطاء المحتملة:

1- عند إضافة ملف رأسي يجب التأكد أنه موجود في نفس مكان المشروع.

2.6.4 تمارين:

1- أكتب ملف رأسي باسم `math.h` حيث به الدالة `abs` و التي تقوم بإخراج القيمة المطلقة للعدد المدخل، ثم قم باستعمال الدالة `abs` في برنامجك.

2.7 الإدخال و الإخراج في الملفات Files I/O

سنعرف في هذا الدرس طريقة التعامل مع الملفات في كل من الإدخال و الإخراج (قراءة الملفات، و إنشاء الملفات و كتابة عليها).

2.7.1 الإخراج في الملفات:

الإخراج يعني بناء برنامج يقوم بإخراج (إنشاء) ملفات ذات امتداد يقوم بتحديده المبرمج، حيث تحتوي تلك الملفات على بيانات.

المثال الأول في هذا الدرس سيكون عبارة عن برنامج يطلب من المستخدم كتابة اسم الملف الذي يريد إنشاءه مع امتداده، و يطلب منه أيضا إدخال النص الذي يريد حفظه في الملف، المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    FILE *File;
6:    char FileName[255];
7:    char String[255];
8:
9:    printf("Enter the name of file(with type)\n[MAX Character 255]: ");
10:   gets(FileName);
11:
12:   printf("Creating File...\n");
13:   File = fopen(FileName, "w");
14:   printf("File Created.\n");
15:
16:   printf("TEXT:\n");
17:   gets(String);
18:
19:   printf("Save Text to the file %s...\n", FileName);
20:   fprintf(File, "%s", String);
21:}
```

في السطر الخامس قمنا بإنشاء مؤشر للبنية FILE و ذلك لأن الدوال الخاصة بالملفات جميع تتطلب مؤشر للبنية FILE.

في السطر السادس قمنا بالإعلان سلسلة حرفية و التي ستكون اسم الملف.

و في السطر السابع سلسلة حرفية و التي ستكون النص الذي سنضعه في الملف.

و في السطر العاشر طلبنا من المستخدم إدخال اسم الملف مع امتداده و أنه أقصى عدد الأحرف التي يمكن إدخالها هو *255*.

و في السطر العاشر قمنا باستعمال الدالة `gets` بدل الدالة `scanf` لأخذ اسم الملف، و سبب ذلك هو:

إن استعملنا الدالة `scanf` لإدخال اسم الملف فربما يقوم المستخدم بإدخال اسم مكون من كلمتين منفصلتين مثل *test.txt test* فهنا الدالة `scanf` ستتوقف عن الفراغ أي أن اسم الملف سيكون *test* و بدون امتداد، و هذا هو سبب استعمال الدالة `gets` لأنها تأخذ الاسم كامل حتى و إن كانت فراغات.

و في السطر الثالث عشر قمنا بإعطاء المؤشر `File` عنوان الدالة `fopen` و التي هي مختصرة من *file open*، الدالة لها وسيطين، الوسيط الأول خاص باسم الملف و الوسيط الثاني فهو النمط أي نوع استخدام الملف، و هنا استعملنا الكتابة لذا كتبنا الرمز `w` و الذي يعني *write*، و توجد أحرف أخرى سنعرفها فيما بعد.

و في السطر السابع عشر ينتظر البرنامج من المستخدم لكي يقوم بإدخال نص و استعمالنا هو الدالة `gets` لنفس السبب السابق.

و أخيرا السطر العشرين حيث توجد الدالة التي تقوم بإدخال النص إلى الملف، و الدالة هي `fprintf` و هي مثل الدالة `printf` و لكنها تتعامل مع الملفات أما `printf` فهي تتعامل مع الشاشة، الدالة `fprintf` مختصرة من *file print format* و لها وسيط إضافي على الدالة `printf` و هو الوسيط الأول و هو مؤشر الملف الذي نريد الكتابة فيه أما باقي الوسائط فهي نفسها وسائط الدالة `printf` و أيضا نفس طريقة استعمالها.

سنقوم الآن بكتابة المثال السابق باستخدام المؤشرات و الدوال و الملفات الرأسية (مع جعل البرنامج أكثر مرونة) كي نعتاد علي استعمالها.

أولا نقوم بإنشاء ملف رأسي باسم `iofile.h` و نقوم بالكتابة عليه التالي:

```
1:/*iofile.h header file
2:for files functions*/
3:#include<stdlib.h> /*for exit() fonction*/
4:
5:void CreateFile(const char *FileName, /*for the name of file*/
6:                const char *String)      /*for text of file*/
7:{
8:    FILE *FileOut;
9:
10:   if(*FileName == '\0')
11:   {
```

```
12:         printf("Error in the name of file!\n");
13:         exit(1);
14:   }
15:
16:   if((FileOut = fopen(FileName, "w"))==NULL){
17:         printf("Can't create file!\n");
18:         exit(1);
19:   }else{
20:         fprintf(FileOut, "%s", String);
21:   }
22:   fclose(FileOut);
22:}
```

هذا الملف هو المرحلة الأولى من البرنامج، و شرحه هو:

في السطر الثالث أضفنا الملف الرأسي stdlib.h و هو مختصر من *standard library*، و أضفناه لاستعمال الدالة exit(…) و التي تقوم بالخروج من البرنامج بدون تنفيذ باقي الأوامر.

قمنا بالإعلان عن الدالة CreateFile في السطر الخامس مع وسيطين الأول لاسم الملف و الثاني لنص الملف.

في السطر العاشر قمنا بوضع مقارنة بين اسم الملف الذي أدخله المستخدم و الصفر، و هو في حالة أن المستخدم لم يدخل أي حرف أو اسم للملف فسيتم الخروج من البرنامج لتجنب الأخطاء و تنبيه المستخدم عن وجود خطأ في اسم الملف.

و في السطر السادس عشر قمنا بمقارنة أخرى و هي بين المؤشر FileOut و NULL حيث NULL هي:

```
#define NULL 0
```

و يفضل كتابة NULL أحسن من الصفر، المقارنة هي إن كانت أخطاء مثل إدخال أحرف غير متفق عليها مثل <> في اسم الملف فهنا أيضا سيتم الخروج من البرنامج بدون إكمال تنفيذ باقي الأوامر، وفي حالة أن المقارنة خاطئة فسيتم كتابة النص إلى الملف.

و استعملنا في السطر الثالث عشر و الثامن عشر الدالة exit حيث لها وسيط واحد و هو الوضع أما *1* و تعني صحيح *true* هنا سيتم الخروج من البرنامج، أو *0* و تعني خطأ *false* و هنا لن يتم الخروج من البرنامج.

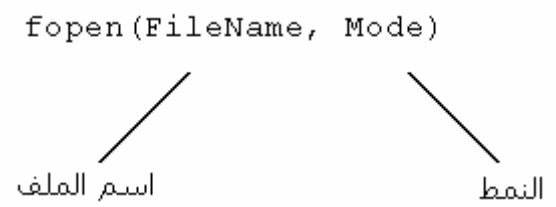
و أخيرا السطر الثاني و العشرين و هي الدالة fclose و هي مختصرة من *file close* و التي تقـــوم بـــإغلاق الملف عند الانتهاء منه حتى يمكننا استعماله مرة أخرى ستلاحظ ذلك فيما بعد.

هذا بالنسبة للملف الرأسي iofile.h، الآن نقوم بإنشاء الملف الرئيسي لمشروعنا باسم main.c:

```
1:#include<stdio.h>
2:#include"iofile.h"
3:
4:main()
5:{
6:    char FileName[255];
7:    char String[255];
8:
9:    printf("Enter the name of file(with type)\n");
10:   printf("[MAX Character 255]: ");
11:   gets(FileName);
12:
13:   printf("TEXT:\n");
14:   gets(String);
15:
16:   printf("Creating File and save text...\n");
17:   CreateFile(FileName, String);
18:}
```

2.7.1.1 الدالة **fopen**:

هي من دوال الملف الرأسي stdio.h، و هي خاصة بالتعامل مع الملفات و لها وسيطين، الأول به سلـــسلة حرفية و هي اسم الملف، و الوسيط الثاني هو النمط أو الوضع الذي تريد استعمالها (القراءة أو الكتابـــة)، صورة توضيحية:



2.7.1.2 الدالة **fclose**:

و هي أيضا من دوال الملف الرأسي stdio.h، و هي أيضا خاصة بالتعامل مع الملفات و لها وسيط واحد و هو مؤشر لـــ * FILE، نكتب فيه اسم مؤشر البنية لكي يتم إغلاقها، و فائدة إغلاقها هي كي نـــستطيع قراءتها مرة أخرى في البرنامج.

2.7.1.3 الدالة **exit**:

من دوال الملف الرأسي stdlib.h، لها وسيط واحد حيث يمكن أن يحمل القيمة *1* أو القيمة *0*، إذا كانت القيمة *1* تعني الخروج من البرنامج مباشرة بدون تنفيذ باقي الأوامر، أما إذا كانت *0* فسيتم تجاهلها.

2.7.2 إضافة نص في نهاية الملف:

الأمثل السابقة في هذا الدرس عندما تقوم بكتابة اسم ملف موجود سابقا فسيتم فقده، و هنا سنعرف كيفية نقوم بإضافة نص في أخر الملف بدون فقد البيانات السابقة.

نفس الطريقة السابقة تمام فقط بدل الحرف w في الدالة fopen نكتب الحرف a، في السطر السادس عشر:

```
16:   if((FileOut = fopen(FileName, "w"))==NULL){
```

و يصبح على الشكل التالي:

```
16:   if((FileOut = fopen(FileName, "a"))==NULL){
```

و الحرف *a* يعني *appending*.

2.7.3 الإدخال في الملفات:

الإدخال تعني القراءة، و في الملفات هي قراءة محتوى ملف و استعماله في البرنامج.

في نفس الملف الرأسي السابق iofile.h قم بإضافة التالي:

```
1:void DisplayFile(const char *FileName)
2:{
3:    FILE *FileIn;
4:    char String[1024];
5:
6:    if(*FileName == '\0')
7:    {
8:          printf("Error in name of file!\n");
9:          exit(1);
10:   }
11:
12:   if((FileIn = fopen(FileName, "r"))==NULL){
13:         printf("Can't Open file!\n");
14:         exit(1);
15:   }else{
16:         fgets(String, 1024, FileIn);
17:         printf("%s", String);
18:   }
19:   fclose(FileIn);
20:}
```

هنا نكون قد أعلنا عن الدالة التي ستقوم بقراءة الملفات، و هي مشابه بدالة إنشاء الملفات.

الدالة تحتوي على وسيط واحد و هو سلسلة حرفية لاسم الملف المراد قراءته.

في السطر الرابع قمنا بالإعلان عن مصفوفة و هناك نقوم بوضع نص الملف.

في السطر الثاني عشر، في الدالة `fopen` استعملنا النمط *r* بدل *w*، الحرف *r* يعني *read*.

في السطر السادس عشر استعملنا الدالة `fgets`، و هي من دوال الملف الرأسي `stdio.h` و هي مختصرة من *file get string*، لها ثلاثة وسائط، الأول لاسم السلسلة الحرفية، الثاني لحجم السلسلة و الثالث لمؤشر ملف الإدخال `FILE *FileIn`.

تقوم الدالة `fgets` بوضع سلسلة حروف الملف (بحجم الوسيط الثاني) في الوسيط الأول و الذي هو عبارة عن سلسلة حروف، ثم طبع سلسلة حروف في السطر السابع عشر.

و الآن قم بإضافة الأوامر التالية في نهاية الدالة الرئيسية `()main` في الملف الرأسي الرئيسي `main.c`:

```
1:    printf("/////////Reading\\\\\\\\\n");
2:    DisplayFile(FileName);
```

و هنا استعملنا الدالة لاستعراض محتوى الملف.

2.7.4 النمط +w و +a و +r:

درسنا سابقا كل من الأنماط *w* (للكتابة) و *a* (لإضافة نص في نهاية ملف) و *r* (لقـــراءة ملـــف)، و الآن سنرى نفس الأنماط السابقة مع إضافات و هي:

2.7.4.1 النمط +w:

هنا يتم إنشاء ملف فارغ للقراءة و الكتابة معا، و إذا كان الملف موجود سابقا فسيتم فقد جميع محتوياته.

2.7.4.2 النمط +a:

هنا يتم إضافة نص في نهاية الملف إذا كان موجود و إن لم يكون موجود يتم إنشاءه، و أيضا يستعمل هذا النمط للقراءة.

2.7.4.3 النمط +r:

هذا النمط للقراءة و الكتاب و لكن يجب أن يكون الملف موجود.

و توجد أنمط أخرى و هي b، s، r، t و d، و لكهان غير مهمة، الأنماط السابقة هي المهمة و التي تستعمل بكثرة.

2.7.5 دوال أخرى خاصة بالتعامل مع الملفات:

توجد دوال عديدة لتعامل مع الملفات و كلها شبيه بالتي قرأنها سابقا، و جميعها من دوال الملف الرأسي stdio.h، الدوال هي:

2.7.5.1 الدالة fprintf و الدالة fscanf:

الدالة fprintf درسنها سابقا، أما الدالة fscanf فهي مكافئة للدالة scanf و هي مختصرة من *file scan file*، و لكنها هنا لا تأخذ قيم من المستخدم، بل تأخذها من قيم من ملف، أي أنها خاصة بالإدخال للملفات، مثال سريع:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    FILE *FileOut;
6:    char Str[100];
7:    int Var;
8:
9:    FileOut = fopen("fscanf.txt", "w+");
10:
11:   fprintf(FileOut, "%s %d", "Hello", 255);
12:
13:   fseek(FileOut, 0, SEEK_SET);
14:
15:   fscanf(FileOut, "%s", Str);
16:   fscanf(FileOut, "%d", &Var);
17:
18:   printf("%s\n", Str);
19:   printf("%d\n", Var);
20:   fclose(FileOut);
21:}
```

في السطر الخامس قمنا بالإعلان عن مؤشر للبنية FILE باسم FileOut.

في السطر السادس سلسلة حرفية بحجم *100*، و هي التي ستحمل نص الملف.

في السطر السابع متغير و هو الذي سيحمل القيمة الموجودة في الملف.

في السطر التاسع قمنا بوضع اسم الملف الذي سنفتحه و نقرأ منه البيانات الموجودة.

في السطر الحادي عشر وضعنا في الملف سلسلة حرفية و هي *Hello* و القيمة *255*.

في السطر الثالث عشر توجد دالة و هي fseek و هي من دوال الملف الرأسي stdio.h و هي مختصرة من *file seek*، و هي تحرك مؤشر الملف إلى مكان يقوم بتحديده المبرمج، لها ثلاثة وسائط الأول هو اسم لمؤشر البنية FILE و الثاني هو عدد البايتات التي يبدأ منها الوسيط الثالث و غالبا ما تكون *0* لكي يتم قراءة جميع البيانات، و الوسيط الثالث هو وضع المؤشر و له ثلاثة ثوابت و هي:

1- SEEK_SET و هو وضع مؤشر الملف في البداية.
2- SEEK_CUR و هو الموقع الحالي لمؤشر الملف.
3- SEEK_END و هو وضع مؤشر الملف في نهايته.

و في مثالنا السابق استعمالنا SEEK_SET كي نبدأ بقراءة بيانات الملف من البداية و وضعنا القيمة *0* في الوسيط الثاني كي نقرأ نفحص جميع البيانات.

و في السطر الرابع عشر قمنا بفحص سلسلة الحروف الموجود في الملف و نسخه في السلسلة الحرفية Str، و هو النص Hello.

في السطر الخامس عشر قمنا بفحص العدد الصحيح *255* و نسخه في المتغير Var.

و في السطر الثامن عشر و التاسع عشر قمنا بطباعة النتائج.

2.7.5.2 الدالة fgets و fputs:

الدالة fgets درسنها سابقا، و الدالة fputs هي مكافئة للدالة puts و لكنها خاصة بالملفات و هي مختصرة من *file put string*، و هي تقوم بطباعة النصوص في الملفات و ليست مثل الدالة puts حيث تقوم بطباعة نص على الشاشة، مثل سريع حول الدالة fputs:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    FILE *FileOut;
6:
7:    FileOut = fopen("fputs.txt", "w");
8:
9:    fputs("Hello, World!\n", FileOut);
10:   fclose(FileOut);
11:}
```

في السطر التاسع استعملنا الدالة fputs حيث الوسيط الأول هو النص المراد طبعه، و الوسيط الثاني هو اسم المؤشر FileOut حيث فيه سيتم طباعة النص.

2.7.5.3 الدالة fgetc و الدالة fputc:

الدالة fgetc تأخذ حرف واحد من ملف، و الدالة fputc تقوم بطباعة حرف واحد إلى ملف معين، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    FILE *FileOut;
6:    FILE *FileIn;
7:    char ch;
8:
9:    FileOut = fopen("fputc.txt", "w");
10:   FileIn = fopen("fputc.txt", "r");
11:
12:   fputc('A', FileOut);
13:   fclose(FileOut);
14:
15:   ch = fgetc(FileIn);
16:   fclose(FileIn);
17:
18:   printf("%c\n", ch);
19:}
```

في السطر الثالث عشر إن لم تستعمل الدالة fclose فسترى نتائج غير مرغوب بها، و هنا ستعرف أهميتها.

و يمكن كتابة المثال السابق على هذه الطريقة:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    FILE *FileInOut;
6:    char ch;
7:
8:    FileInOut = fopen("fputc.txt", "w");
9:
10:   fputc('A', FileInOut);
11:   fclose(FileInOut);
12:
13:   FileInOut = fopen("fputc.txt", "r");
14:
15:   ch = fgetc(FileInOut);
16:   fclose(FileInOut);
17:
18:   printf("%c\n", ch);
19:}
```

هذه هي الدوال المهمة حاليا في العامل مع الملفات، توجد دوال أخرى كثيرة حول التعامل مع الملفـــات و سندرسها في درس المكتبة القياسية للغة C.

2.7.6 الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

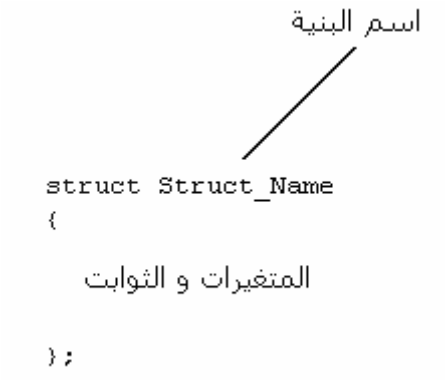
2.7.7 تمارين:

1- أكتب برنامج يقوم بعمليات الجمع، الطرح، الضرب و القسمة، و يتم حفظ النتائج في ملف باسم *results.dat* تلقائيا.

2.8 التراكيب structures

التراكيب (البنيات) هي مجموعة من متغير واحد أو أكثر تجمع تحت اسم واحد يسهل استعمالها، و المتغيرات في التراكيب ليس مثل المتغيرات في المصفوفات، يمكن أن تكون به متغيرات مختلفة الأنواع، و التراكيب يمكن أن تحمل أي نوع من متغيرات لغة C حتى مصفوفات أو مؤشرات أو تراكيب داخل تراكيب أخرى، و جميع المتغيرات الموجود داخل التراكيب تسمى بأعضاء لتراكيب.

للإعلان عن بنية نقوم بكتابة الكلمة المحجوزة struct ثم اسم البنية و نقوم بفتح حاضنة و نكتب المتغيرات و الثوابت التي نريدها (الأعضاء) ثم نغلق الحاضنة مع كتابة فاصلة منقوطة، صورة توضيحية:



2.8.1 اسم البنية Struct Name:

اسم البنية له شروط مثل شروط اسم المتغير و هي:

- أن لا يتجاوز اسم البنية أكثر من 31 حرف.
- أن لا يبدأ اسم البنية بأرقام.
- أن لا يكون اسم البنية يحتوي على مؤثرات مثل الجمع، الطرح، ....
- أن لا يكون اسم البنية يحتوي على رموز مثل % و # و } و...(باستثناء الرمز _).
- أن لا يكون اسم البنية مستعمل سابقا لاسم دالة أو متغير أو بنية أخرى.
- أن لا يكون اسم البنية من أحد أمساء الكلمات المحجوزة.

و الإعلان عن البنيات يستحسن أن يكون دائما خارج الدالة الرئيسية و قبلها، مثال:

```
1:#include<stdio.h>
2:
3:struct _2D
```

```
4:{
5:    int x;
6:    int y;
7:};
8:
9:main()
10:{
11:   struct _2D Struct_2D;
12:   int x, y;
13:
14:   printf("Enter X: ");
15:   scanf("%d", &Struct_2D.x);
16:
17:   printf("Enter Y: ");
18:   scanf("%d", &Struct_2D.y);
19:
20:   for(x=0;x<=Struct_2D.x;x++)
21:   {
22:         printf("%c", 219);
23:         for(y=0;y<Struct_2D.y;y++)
24:               printf("%c", 219);
25:         printf("\n");
26:   }
27:
28:}
```

هذا البرنامج يقوم برسم المربعات حسب القيم التي يقوم بإدخالها المستخدم.

في السطر الثالث قمنا بالإعلان عن البنية _2D.

و في السطر الخامس و السادس قمنا بالإعلان عن متغيرين x و y، حيث يعتبران عضوين للبينة _2D.

في السطر الحادي عشر قمنا بكتابة الكلمة المحجوزة struct مع اسم البنية السابقة _2D و الاسم Struct_2D و الذي سيتم العمل عليه في هذا البرنامج، و لا يمكن كتابة_2D Struct_2D بدون الكلمة المحجوزة struct، و لا يمكن استعمال البنية _2D مباشرة لذا يجب أن نقوم بإعلان عن متغير للبنية _2D و التي هي Struct_2D في برنامجنا هذا.

أما باقي السطور فهي مفهومة.

و يمكن الإعلان عن أكثر من بنية، فمثلا المثال السابق في السطر الحادي عشر يمكننا كتابة:

```
11:   struct _2D Struct_2D_1, Struct_2D_2;
```

و هنا يمكن استعمال كل من البنيتين Struct_2D_1 و البنية Struct_2D_2.

أما إذا أردت تجاهل كتابة السطر الحادي عشر، فيمكن ذلك و لكن يجب أن نعطي لبنيتنا معرف و سنكتبه قبل الفاصلة المنقطة:

```
1:struct _2D
2:{
3:    int x;
4:    int y;
5:}Struct_2D;
```

و في حالة نريد التعريف عن أكثر من اسم البنية فنفصل بين اسم و اسم بفاصلة مثل:

```
1:struct _2D
2:{
3:    int x;
4:    int y;
5:}Struct_2D_1, Struct2D_2;
```

و يصبح المثال على الشكل التالي:

```
1:#include<stdio.h>
2:
3:struct _2D
4:{
5:    int x;
6:    int y;
7:}Struct_2D;
8:
9:main()
10:{
11:   int x, y;
12:
13:   printf("Enter the position X: ");
14:   scanf("%d", &Struct_2D.x);
15:
16:   printf("Enter the position Y: ");
17:   scanf("%d", &Struct_2D.y);
18:
19:   for(x=0;x<=Struct_2D.x;x++)
20:   {
21:         printf("%c", 219);
22:         for(y=0;y<Struct_2D.y;y++)
23:               printf("%c", 219);
24:         printf("\n");
25:   }
26:
27:}
```

و توجد طرق أخرى لتعامل مع البنيات، مثلا يمكن إعطاء قيمة سابقة لمتغير في بنية و استعماله في البرنامج، مثال:

```
1:#include<stdio.h>
2:
3:struct _Value
4:{
```

```
5:    int x;
6:    float y;
7:    char *Str;
8:}Value;
9:
10:main()
11:{
12:   Value.x = 10;
13:   Value.y = 10.00;
14:   Value.Str = "Hello, World";
15:
16:   printf("%d\n", Value.x);
17:   printf("%f\n", Value.y);
18:   printf("%s\n", Value.Str);
20:}
```

أو يمكن إعطاء القيم مباشرة بعد التعرف عن اسم للبنية مثل:

```
1:#include<stdio.h>
2:
3:struct _Value
4:{
5:    int x;
6:    float y;
7:    char *Str;
8:}Value = {10, 10.00, "Hello, World"};
9:
10:main()
11:{
12:   printf("%d\n", Value.x);
13:   printf("%f\n", Value.y);
14:   printf("%s\n", Value.Str);
15:}
```

و لا يمكن إعطاء قيمة عند الإعلان عن البنية مثل:

```
1:struct _Value
2:{
3:    int x = 10;
4:    float y = 10.00;
5:    char *Str = "Hello World";
6:};
```

لأن في هذه الحالة لا معنى للبنية Value_ و لا معنى لاستعمال البنية أصلا لأنه يمكن كتابة:

```
1:#include<stdio.h>
2:
3:int x = 10;
4:float y = 10.00;
5:char *Str = "Hello World";
6:
7:main()
8:{
9:    printf("%d\n", x);
10:   printf("%f\n", y);
11:   printf("%s\n", Str);
12:}
```

2.8.2 البنيات باستخدام الكلمة المحجوزة union:

يمكننا الإعلان عن البنيات باستعمال الكلمات المحجوزة union بنفس الطرق السابقة، و الفرق الوحيد بين استعمال البنية باستخدام الكلمة المحجوزة struct و الكلمة المحجوزة union هو عند استعمال البنيات باستخدام union فالنتائج لن تكون مثل البنيات التي بـ struct مثلا أنظر إلى المثال التالي:

```
1:#include<stdio.h>
2:
3:union _Union
4:{
5:    int x;
6:    int y;
7:}Union;
8:
9:main()
10:{
11:   Union.x = 10;
12:   Union.y = 15;
13:
14:   printf("Union.x = %d\n", Union.x);
15:   printf("Union.z = %d\n", Union.y);
16:}
```

في هذا البرنامج بدل أن تكون النتائج *10 15* فسيتكون *15 15*، و أيضا ستختلف النتائج إن استعملت التالي:

```
1:#include<stdio.h>
2:
3:union _Union
4:{
5:    int x;
6:    int y;
7:}Union;
8:
9:main()
10:{
11:   Union.y = 15;
12:   Union.x = 10;
13:
14:   printf("Union.x = %d\n", Union.x);
15:   printf("Union.z = %d\n", Union.y);
16:}
```

يمكنك الآن استنتاج الفرق بين البنيات باستخدام الكلمة المحجوزة struct و البنيات باستخدام الكلمة المحجوزة union، و الذي هو إشتراك جميع المتغيرات في عنوان واحد، و إن غيرنا قيمة متغير واحدة فستكون تلك القيمة لجميع متغيرات البنية، مثال:

```
1:#include<stdio.h>
2:
3:union _Union
4:{
```

```
5:    int x1, x2;
6:    int y1, y2;
7:}Union;
8:
9:main()
10:{
11:   Union.x1 = 10;
12:
13:   printf("Value of Union.x1 = %d\n", Union.x1);
14:   printf("Value of Union.x2 = %d\n", Union.x2);
15:   printf("Value of Union.y1 = %d\n", Union.y1);
16:   printf("Value of Union.y2 = %d\n", Union.y2);
17:
18:   printf("Address of Union.x1 = %p\n", &Union.x1);
19:   printf("Address of Union.x2 = %p\n", &Union.x2);
20:   printf("Address of Union.y1 = %p\n", &Union.y1);
21:   printf("Address of Union.y2 = %p\n", &Union.y2);
22:}
```

و عند استعمال بنيات باستخدام الكلمة المحجوزة union بها متغيرات مختلفة الأنواع فستكون النتائج غـــير موثوقة، مثال:

```
1:#include<stdio.h>
2:
3:union _Union
4:{
5:    int x;
6:    float y;
7:}Union;
8:
9:main()
10:{
11:   Union.x = 10;
12:   Union.y = 15.00;
13:
14:  printf("Union.x = %d\n", Union.x);
15:   printf("Union.z = %f\n", Union.y);
16:}
```

بالنسبة لنتيجة متغير العدد الحقيقي y ستكون صحيحة، أما المتغير x فستكون نتيجتها غير مرغوبة، و يمكنك استنتاج السبب.

2.8.3 المصفوفات و المؤشرات على البنيات:

لا أقصد بنية بها أعضاء من نوع مصفوفات أو مؤشرات، بل أقصد البنية نفسها، بالنسبة للمصفوفات مـــع البنية فهي شبيه بطريقة الإعلان عن مصفوفات طبيعية، مثال:

```
1:#include<stdio.h>
2:
3:struct _Arr
4:{
5:    int x;
6:}Arr[2];
7:
```

```
8:main()
9:{
10:   Arr[0].x = 10;
11:   Arr[1].x = 20;
12:
13:   printf("Arr[0].x = %d\n", Arr[0].x);
14:   printf("Arr[1].x = %d\n", Arr[1].x);
15:}
```

و أيضا يمكن كتابة:

```
1:struct _Arr
2:{
3:    int x;
4:}Arr[2] = {10, 20};
```

بدل:

```
1:    Arr[0].x = 10;
2:    Arr[1].x = 20;
```

و يمكن أيضا كتابة مصفوفة لبنية ثنائية أو ثلاثية الأبعاد أو أكثر بنفس الطرق السابقة التي درسنها في درس المصفوفات، و يمكن أيضا كتابة مصفوفة لبنية تحتوي على متغيرات لأنواع عديدة مثل:

```
1:#include<stdio.h>
2:
3:struct _Arr
4:{
5:    int x;
6:    float y;
7:    char *Str;
8:}Arr[2] = {
9:    {10, 10.00, "Str1"},
10:   {20, 20.00, "Str2"}
11:};
12:
13:main()
14:{
15:   /*Arr[0] :*/
16:   printf("Arr[0].x = %d\n", Arr[0].x);
17:   printf("Arr[0].y = %f\n", Arr[0].y);
18:   printf("Arr[0].Str = %s\n", Arr[0].Str);
19:
20:   /*Arr[1] :*/
21:   printf("Arr[1].x = %d\n", Arr[1].x);
22:   printf("Arr[1].y = %f\n", Arr[1].y);
23:   printf("Arr[1].Str = %s\n", Arr[1].Str);
24:}
```

و هذا مثال للمؤشرات:

```
1:#include<stdio.h>
2:
3:struct _ptr
4:{
```

```
5:    int x;
6:}Addr_ptr, *ptr;
7:
8:main()
9:{
10:   ptr = &Addr_ptr;
11:   ptr->x = 10;
12:
13:   printf("ptr->x = %d\n", ptr->x);
14:
15:   /*Or*/
16:   (*ptr).x = 20;
17:   printf("(*ptr).x = %d\n", (*ptr).x);
18:}
```

طريقة استعمال مؤشر لبنية تختلف قليلا عن استعمال مؤشرات طبيعية، بالنسبة للإعلان فهي نفسها، أمـــا إعطاء العنوان و القيمة تختلف.

في السطر العاشر أعطينا للمؤشر ptr عنوانا و هو عنوان البنية Adrr_ptr.

و في السطر الحادي عشر أعطينا للعضو x القيمة *10*، و يتم إعطاء قيم لأعضاء مؤشر بنية عبر الرمزين <- ثم اسم العضو، ثم قيمته.

2.8.4 إعلان بنية داخل بنية:

يمكن استعمال بنية داخل بنية، مثلا إذا أردنا أن نرسم خط مستقيم، هذا يحتاج إلى نقـــطتين، الأولى هـــي بداية المستقيم و الثانية هي نهاية المستقيم، و يجب أن يكون لكل نقطة مكانها على شاشة الحاسوب في كل من *x* و y، مثال:

```
1:#include<stdio.h>
2:
3:struct _line
4:{
5:    struct _point
6:    {
7:          int x;
8:          int y;
9:    }point_1, point_2;
10:}line;
11:
12:main()
13:{
14:   /*point 1:*/
15:   line.point_1.x = 10;
16:   line.point_1.y = 10;
17:   /*point 2:*/
18:   line.point_2.x = 100;
19:   line.point_2.y = 10;
20:}
```

و هنا استعملنا البنية point_ داخل البنية line_، و يجب أن نقوم بالتعرف لأسماء point_ مباشـــرة عنـــد كتابتها كي نستطيع استعمال المتغيرين x و y.

2.8.5 حقول البت Bit-fields:

حقول البت هي تحديث حجم عضو أو أعضاء بنية بالبت، مثال:

```
1:#include<stdio.h>
2:
3:struct _Bit_Fields{
4:    unsigned _8Bit: 8;           /*1 Byte*/
5:    unsigned _16Bit: 16;    /*2 Byte*/
6:}Bit_Fields;
7:
8:main(int argc, char *argv[]){
9:    Bit_Fields._8Bit = 255;
10:   Bit_Fields._16Bit = 65535;
11:
12:   printf("_8Bit = %d\n", Bit_Fields._8Bit);
13:   printf("_16Bit = %d\n", Bit_Fields._16Bit);
14:}
```

حجزنا للمتغير _8Bit *8* بتّات أي *1* بايت، و لا يمكن أن يحمل قيمة أكثر من *255*، و حجزنـــا للمـــتغير _16Bit *16* بتّات أي *2* بايت، و لا يمكن أن يحمل قيمة أكثر من *65535*.

2.8.6 الأخطاء المحتملة:

1- لا يمكن استعمال بنية مباشرة مثل:

```
1:#include<stdio.h>
2:
3:struct Struct
4:{
5:    int x, y;
6:};
7:
8:main()
9:{
10:   Struct.x = 10;
11:   Struct.y = 20;
12:}
```

2- لا يمكن الإعلان عن بنيتين من نفس الاسم.

2.8.7 تمارين:

1- أكتب بنية بسم time بها ثلاثة أعضاء و هي hh, mm, ss و كلها من النوع int، و نطلب من المستخدم إدخال الساعة و الدقيقة و الثانية الحالية و نعطي تلك القيم للبنية time ثم نطبع الساعة على الطريقة ***HH :MM :SS*.**

2.9 ملخص للفصل الثاني، معا إضافات

درسنا سابقا دالة الإدخال scanf، و قلنا أنه يجب أن نعطيه اسم متغير مع مؤثر العنوان &، و ذلك يعني أنه عند كتابة:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int i;
6:
7:    scanf("%d", &i);
8:}
```

فهنا سيتم وضع القيمة المدخلة في عنوان المتغير i، لأن الدالة scanf تتعامل مع المؤشرات، و يمكن كتابة المتغير بدون مؤثر، و لكن النتائج لن تكون صحيحة.

2.9.1 معنى دالة بها وسيط void:

أحيانا ما تجد في بعض الأمثل دوال بها وسيط من نوع void مثل:

```
1:#include<stdio.h>
2:
3:Function(void);
4:
5:main()
6:{
7:
8:}
9:
10:Function(void)
11:{
12:
13:}
```

أو تجد الدالة الرئيسية نفسها بها هذا الوسيط، مثال:

```
1:main(void)
2:{
3:
4:}
```

مثل هذه الدوال الوسيط الخاص بها لا يعني شيء، و كلمة void إنجليزية و هي تعني فراغ، و هذا هو معنى كتابة دوال بها وسيط من هذا النوع، يعني التأكيد على أن الدالة فارغة الوسائط، و مثل تلك الدوال يمكن تجاهل كتابة الكلمة المحجوزة void، حيث أن كلا من هذه الأمثلة:

```
1:#include<stdio.h>
```

```
2:
3:Function();
4:
5:main()
6:{
7:
8:}
9:
10:Function()
11:{
12:
13:}
```

و المثال:

```
1:main()
2:{
3:
4:}
```

مكافئة للأمثلة السابقة.

2.9.2 الكلمة المحجوزة static:

كلمة static تعني ساكن و هي تستعمل مع المتغيرات حيث تجعلها ثابت بطرقة ستفهمها من هذا المثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    int x;
6:
7:    for(x=0; x<10; x++)
8:    {
9:          static int Static = 0;
10:         Static++;
11:
12:         printf("%d\n", Static);
13:   }
14:}
```

في السطر التاسع استعملنا الكلمة المحجوزة static مع المتغير Static داخل حلقة و أعطيناه القيمة *0*، و استعملنا مؤشر الزيادة في السطر العاشر مع نفس المتغير Static، و في السطر الثاني عشر قمنا بطباعة النتائج.

إذا قمنا بإزالة الكلمة المحجوزة static من المتغير Static فستكون النتائج ثابتة و هي *1* عشرة مرات، أما عند استعمالها فسترى أن النتيجة غير ثابتة و تعمل بتزايد.

و تستعمل الكلمة المحجوزة static بكثرة في الدوال، مثال:

```
1:#include<stdio.h>
2:
3:int Test_static(int a);
4:
5:main()
6:{
7:    int i;
8:
9:    for(i=0;i<=10;i++)
10:         printf("%d * %d = %d\n", i, i, Test_static(i));
11:}
12:
13:int Test_static(int a)
14:{
15:   static int c = 0;
16:   int a_c = a*c;
17:   c++;
18:
19:   return a_c;
20:}
```

و في حالة إزالة الكلمة المحجوزة static فستكون جميع النتائج *0*.

إن لم نقم بتحديد قيمة لمتغير static فستكون قيمته *0* تلقائيا، و ليس مثل المتغيرات الطبيعية التي تنتج قيم عشوائية.

2.9.3 الكلمة المحجوزة typedef:

تستعمل الكلمة المحجوزة typedef مع كل من المتغيرات و البنيات، و هي تعطي إمكانيات الإعلان عنـــها مثل البنية، مثال:

```
1:#include<stdio.h>
2:
3:typedef int Decimal;
4:
5:main()
6:{
7:    /* or typedef int Decimal; here*/
8:    Decimal Var_1, Var_2;
9:    Decimal Var_3;
10:
11:   Var_1 = 10;
12:   Var_2 = 20;
13:   Var_3 = Var_1 + Var_2;
14:
15:   printf("%d\n", Var_3);
16:}
```

في السطر الثالث قمنا بالإعلان عن متغير من نوع int مع الكلمة المحجوزة typedef باسم Decimal.

في السطر الثامن و التاسع استعملنا المتغير Decimal للإعلان عـــن المـــتغيرات Var_1 و Var_2 و Var_3 و تعاملنا معا تلك المتغيرات كأي كتغيرات أخرى.

و في مثالنا هذا استبدلنا الكلمة المحجوزة int بـــ Decimal حيث حجمها سيكون حجم النوع الـــذي تم الإعلان به، و لا يمكن إعطاءها قيم.

أما بالنسبة للمبنية فهي نفسها مشابه مثلا:

```
1:#include<stdio.h>
2:
3:struct _2D
4:{
5:    int x, y;
6:};
7:
8:main()
9:{
10:   struct _2D A;
11:   struct _2D B;
12:
13:   A.x = 10;
14:   B.y = 20;
15:
16:   printf("%d\n", A.x);
17:   printf("%d\n", B.y);
18:}
```

و لكنها تجعلها أكثر مرونة من السابقة، مثال:

```
1:#include<stdio.h>
2:
3:typedef struct _2D
4:{
5:    int x, y;
6:}Struct1, Struct2;
7:
8:main()
9:{
10:   Struct1 S1_1, S1_2;
11:   Struct2 S2_1, S2_2;
12:
13:   S1_1.x = 10, S1_2.x = 20;
14:   S2_1.y = 30, S2_2.y = 40;
15:}
```

ملاحظة: عند استعمال متغيرات أو دوال أو بنيات مع الكلمة المحجوزة typdef فلا يمكن إعطاءهـــا قـــيم سابقة مثل:

```
1:#include<stdio.h>
2:
3:main()
```

```
4:{
5:    typedef float PI;
6:    PI = 10.00;
7:    printf("%f\n", PI);
8:}
```

2.9.4 برامج تدريبية:

في هذا الجزء من الدرس سنرى بعض البرامج التي ستساعدك على فهم لغة C بشكل مبسط، مع شرح سريع لكل برنامج:

2.9.4.1 البرنامج الأول، النسخ:

في هذا البرنامج نقوم بكتابة دالة تقوم بنسخ سلسلة حروف إلى سلسلة حروف أخرى فارغة، المثال:

الملف str.h:

```
1:/* string.h */
2:
3:/*strcopy(pointers)*/
4:
5:void strcopy(char *From, char *To){
6:    while((*To++ = *From++)!='\0')
7:          ;
8:}
9:
10:/*strcopy(arrays);*/
11:/*void strcopy(char From[], char To[]){
12:   int i;
13:   i = 0;
14:
15:   while((To[i] = From[i])!='\0')
16:         ++i;
17:}*/
```

الملف الرئيسي:

```
1:/*main.c*/
2:
3:#include<stdio.h>
4:#include"str.h"
5:
6:main()
7:{
8:    char *From = "STRING";
9:    char Empty[6];
10:
11:   strcopy(From, Empty);
12:
13:   printf(Empty);
14:}
```

هنا الدالة ستقوم بنسخ ما هو موجود في السلسلة From إلى السلسلة Empty، ثم نطبع محتـــوى السلـــسلة Empty كي نتأكد من النتائج.

2.9.4.2 تبادل القيم بين وسيطين:

في هذا البرنامج نقوم بإنشاء دالة تقوم بوضع قيمة المتغير الوسيط الأول في الوسيط الثاني و قيمة الوســـيط الثاني في الوسيط الأول، المثال:

```
1:#include<stdio.h>
2:
3:void Change(int *a, int *b)
4:{
5:    int c;
6:    c = *a;
7:    *a = *b;
8:    *b = c;
9:}
10:
11:main()
12:{
13:   int a, b;
14:   a = 5;
15:   b = 10;
16:
17:   printf("a = %d, b = %d\n", a, b);
18:
19:   Change(&a, &b);
20:
21:   printf("a = %d, b = %d\n", a, b);
22:}
```

إذا استعملنا متغيرات في مكان المؤشرين *a, *b فإن النتائج البرنامج ستكون خاطئة، و هنا نـــرى فائـــدة التعامل مع عناوين الذاكرة.

2.9.4.3 التغير في قيم ثوابت:

قلنا سابقا أنه لا يمكن التغير في قيم ثوابت، و لكن عبر المؤشرات يمكننا ذلك، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    const int Const = 10;
6:    int *ptr = &Const;
7:
8:    printf("Const = %d\n", Const);
9:    *ptr = 5;
10:   printf("Const = %d\n", Const);
11:
12:}
```

و هذا ما يسمى بالتلاعب في ذاكرة الحاسوب.

2.9.4.4 عكس سلسلة نصية:

في هذا البرنامج نقوم بإنشاء دالة بها وسيط لسلسلة من حروف، حيث تقوم تلك الدالة بعكس تلك السلسلة في نفسها، المثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:void Reverse_Str(char *);
5:
6:main()
7:{
8:    char *str = "Hello, World!";
9:
10:   printf("%s\n", str);
11:   Reverse_Str(str);
12:   printf("%s\n", str);
13:}
14:
15:void Reverse_Str(char *String){
16:   int i = strlen(String)-1, j = 0;
17:   char ch;
18:
19:   while(j<i){
20:         ch = String[j];
21:         String[j] = String[i];
22:         String[i] = ch;
23:         j++, i--;
24:   }
25:}
```

2.9.4.5 التحويل من النظام العشري إلى النظام الثنائي:

من خلال هذا البرنامج يمكنك استنتاج كيفية التحويل إلى باقي الأنظمة، المثال:

```
1:#include<stdio.h>
2:
3:void ConvertToBinary(int);
4:
5:main()
6:{
7:    int Decimal;
8:
9:    printf("Decimal Number: ");
10:   scanf("%d", &Decimal);
11:   printf("%d in Binary = ", Decimal);
12:   ConvertToBinary(Decimal);
13:}
14:
15:void ConvertToBinary(int num){
16:   int i = 0, Binary[32];
17:
18:   while(num>0){
19:         if((num%2)==0){
20:               Binary[i] = 0;
```

```
21:               num /= 2, ++i;
22:         }
23:         else
24:         {
25:               Binary[i] = 1;
26:               num /= 2, ++i;
27:         }
28:   }
29:
30:   --i;
31:
32:   while(i>=0)
33:   {
34:         printf("%d", Binary[i]), --i;
35:
36:   }
37:
38:   printf("\n");
39:}
```

2.9.4.6 التحويل من الحروف الصغيرة إلى الحروف الكبيرة:

في هذا البرنامج نقوم بإنشاء دالة تقوم بتحويل سلسلة حرفية من الحروف الصغيرة إلى الحروف الكـــبيرة، المثال:

```
1:#include<stdio.h>
2:
3:void To_Capital_letter(char ch[]);
4:
5:main()
6:{
7:    char *ch = "hello";
8:
9:    printf("Small Letters: %s\n", ch);
10:   To_Capital_letter(ch);
11:   printf("Capital Letters: %s\n", ch);
12:}
13:
14:void To_Capital_letter(char ch[])
15:{
16:   int i=0;
17:
18:   while(ch[i]!='\0'){
19:         if(ch[i]>=97 && ch[i]<=122)
20:               ch[i] = ch[i]-32;
21:         ++i;
22:   }
23:}
```

و يمكن أيضا استعمال العكس، من الأحرف الكبيرة إلى الأحرف الصغيرة، استنتج ذلك.

2.9.5 الدالة wcscpy و الدالة wcsncpy:

الدالة wcscpy مكافئة للدالة strcpy و هي من دوال الملف الرأسي string.h و اسمها مختصر مـــن *wide character string copy*، تقوم بنفس عمل الدالة strcpy فقط هي للأحرف العريضة، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main()
5:{
6:    wchar_t *wStr = L"Hello";
7:    wchar_t Empty[20];
8:
9:    wcscpy(Empty, wStr);
10:
11:   wprintf(L"%s\n", Empty);
12:}
```

و أيضا الدالة wcsncpy مكافئة للدالة strncpy و هي من دوال الملف الرأسي string.h، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main()
5:{
6:    wchar_t *wStr = L"Hello";
7:    wchar_t Empty[20];
8:
9:    wcsncpy(Empty, wStr, 4);
10:
11:   wprintf(L"%s\n", Empty);
12:}
```

2.9.6 الدالة wcscat و الدالة wcsncat:

و كلا من الدالتين wcscat و wcsncat مكافئة للدالتين strcat و strncat، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main()
5:{
6:    wchar_t *wStr = L"Hello";
7:    wchar_t *wStr2 = L", World!";
8:
9:    wcscat(wStr, wStr2);
10:
11:   wprintf(L"%s\n", wStr);
12:
13:   /*
14:   wcsncat(wStr, wStr2, 4);
15:
16:   wprintf(L"%s\n", wStr);
17:   */
18:}
```

2.9.7 الدالة getwchar و putwchar:

الدالة getwchar مكافئة لدالة getchar، و هي من دوال الملف الرأسي stdio.h، واسمها مختصر من *get wide character*، و أيضا الدالة putwchar مكافئة لدالة putchar، و مختصرة مـــن *put wide character*، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    wchar_t wch;
6:
7:    wch = getwchar();
8:    putwchar(wch);
9:}
```

2.9.8 الدالة **getws_** و **putws_**:

الدالة getws_ مكافئة لدالة gets، و الدالة putws_ مكافئة لدالة puts، و هما من دوال الملـــف الرأســـي stdio.h، و كل من الحرفين الإضافيين ws مختصرين من *wide string*، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    wchar_t wch[255];
6:
7:    _getws(wch);
8:    _putws(wch);
9:}
```

2.9.9 **جدول** ASCII **(صورة)**:

كلمة *ASCII* مختصرة من *American Standard Code for Information Interchange*، هو جدول لمجموعة من الرموز مستندة على الأبجدية الرومانية تمثل رموز في الحاسبات ، الرموز على الجدول التالي:

جدول ASCII

0:	1:☺	2:☻	3:♥	4:♦	5:♣	6:♠	7:	8	9:
	10:								
	14:♫	15:☼	16:►	17:◄	18:↕	19:‼	20:¶	21:§	22:▬
23:↨	24:↑	25:↓	26:→	27:←	28:∟	29:↔	30:▲	31:▼	32:
33:!	34:"	35:#	36:$	37:%	38:&	39:'	40:(	41:)	42:*
43:+	44:,	45:-	46:.	47:/	48:0	49:1	50:2	51:3	52:4
53:5	54:6	55:7	56:8	57:9	58::	59:;	60:<	61:=	62:>
63:?	64:@	65:A	66:B	67:C	68:D	69:E	70:F	71:G	72:H
73:I	74:J	75:K	76:L	77:M	78:N	79:O	80:P	81:Q	82:R
83:S	84:T	85:U	86:V	87:W	88:X	89:Y	90:Z	91:[	92:\
93:]	94:^	95:_	96:`	97:a	98:b	99:c	100:d	101:e	102:f
103:g	104:h	105:i	106:j	107:k	108:l	109:m	110:n	111:o	112:p
113:q	114:r	115:s	116:t	117:u	118:v	119:w	120:x	121:y	122:z
123:{	124:|	125:}	126:~	127:⌂	128:Ç	129:ü	130:é	131:â	132:ä
133:à	134:å	135:ç	136:ê	137:ë	138:è	139:ï	140:î	141:ì	142:Ä
143:Å	144:É	145:æ	146:Æ	147:ô	148:ö	149:ò	150:û	151:ù	152:ÿ
153:Ö	154:Ü	155:ç	156:£	157:¥	158:₧	159:ƒ	160:á	161:í	162:ó
163:ú	164:ñ	165:Ñ	166:ª	167:º	168:¿	169:⌐	170:¬	171:½	172:¼
173:¡	174:«	175:»	176:░	177:▒	178:▓	179:|	180:┤	181:╡	182:╢
183:╖	184:╕	185:╣	186:||	187:╗	188:╝	189:╜	190:╛	191:┐	192:└
193:┴	194:┬	195:├	196:─	197:┼	198:╞	199:╟	200:╚	201:╔	202:╩
203:╦	204:╠	205:═	206:╬	207:╧	208:╨	209:╤	210:╥	211:╙	212:╘
213:╒	214:╓	215:╫	216:╪	217:┘	218:┌	219:█	220:▄	221:▌	222:▐
223:▀	224:α	225:ß	226:Γ	227:π	228:Σ	229:σ	230:µ	231:τ	232:Φ
233:Θ	234:Ω	235:δ	236:∞	237:ø	238:ϵ	239:∩	240:≡	241:±	242:≥
243:≤	244:⌠	245:⌡	246:÷	247:≈	248:°	249:∙	250:·	251:√	252:ⁿ
253:²	254:■	255:							

إذا أردت استعمل أي رمز من الرموز السابقة قم بكتابة التالي:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    printf("The number %d is the character %c in ASCII code.\n", 210, 210);
6:}
```

و هنا سيتم طبع الرمز رقم *210* من جدول *ASCII* و هو الرمز ╥، و يمكن اختيار أي رمز من الرموز السابقة فقط نكتب الرقم و نطبعه على شكل حرف(رمز).

2.9.10 معلومات أكثر حول المتغيرات:

توجد متغيرات خارجية و متغيرات محلية، الأولى هي متغيرات عامة يمكن استعمالها بصفة عامة، أما المتغيرات المحلية فهي متغيرات لها حدودها.

2.9.10.1 المتغيرات المحلية:

هي متغيرات يمكن استعمالها في الدالة التي تم الإعلان عنها، مثال:

```
1:#include<stdio.h>
2:
3:void Func()
4:{
5:    int a;      /*Local Variable*/
6:}
7:
8:main()
9:{
10:   int b;      /*Local Variable*/
```

```
11:}
```

هنا كل من المتغير a في الدالة Func و المتغير b في الدالة الرئيسية يعتبرا متغيرات محلية حيث لا يمكن استعمال المتغير a في الدالة الرئيسية لأنه خاص بالدالة Func، و لا يمكن استعمال المتغير b في الدالة Func لأنه معرف في الدالة main، و هذا هو مفهوم المتغيرات المحلية.

2.9.10.2 المتغيرات الخارجية (العامة):

هي متغيرات يمكن التعامل معها في جميع الدوال، أي في البرنامج بأكمله، و يتم الإعلان عنها خارج جميع الدوال، مثال:

```
1:#include<stdio.h>
2:
3:int ab;               /*External Variable*/
4:
5:void Func()
6:{
7:    ab = 10;    /*Use The External Variable ab*/
8:}
9:
10:main()
11:{
12:   Func();
13:   printf("%d\n", ab); /*print The External Variable ab*/
14:}
```

المتغير ab هو المتغير العام للبرنامج، حيث يمكن التعامل معه في جميع الدوال، أعطين للمتغير ab قيمة في الدالة Func ثم قنا بتنفيذ الدالة Func في الدالة الرئيسية ثم طبع محتوى المتغير الخارجي ab في السطر الثالث عشر.

إذا لم تكن للمتغيرات العامة قيم سابقة فستكون قيمها *0* تلقائيا.

2.9.10.3 الكلمة المحجوزة extern:

تستعمل الكلمة المحجوزة extern مع المتغيرات داخل دوال، و تستعمل لجعل متغير محلي مشترك مع متغير خارجي، مثلا يمكننا الإعلان عن متغير محلي و متغير خارجي بنفس الاسم و لكن بقيم مختلفة، مثال:

```
1:#include<stdio.h>
2:
3:int ab;
4:
5:void Func()
6:{
7:    int ab;
8:    ab = 10;
9:    printf("%d\n", ab);
10:}
11:
12:main()
```

```
13:{
14:   ab = 5;
15:   Func();
16:   printf("%d\n", ab);
17:}
```

في هذا المثال المتغير المحلي ab الموجود داخل الدالة Func لا علاقة له مع المتغير الخارجي ab، أما إذا أردنا أن نجعل قيمتهما مشترك نضيف الكلمة المحجوزة extern إلى المتغير المحلي، مثال:

```
1:#include<stdio.h>
2:
3:int ab;
4:
5:void Func()
6:{
7:    extern int ab;
8:    ab = 10;
9:    printf("%d\n", ab);
10:}
11:
12:main()
13:{
14:   ab = 5;
15:   Func();
16:   printf("%d\n", ab);
17:}
```

هنا ستكون قيمة المتغير الخارجي ab هي نفس قيمة المتغير الداخلي ab.

2.9.10.4 الكلمة المحجوزة auto:

تستعمل الكلمة المحجوزة auto مع المتغيرات، و هي تعني *automatic* أي آلي، و تستعمل مع المتغيرات لتبين أن تلك المتغيرات افتراضية أي طبيعية و ليست ساكنة static، مثال:

```
1:#include<stdio.h>
2:
3:void Func()
4:{
5:    static int Static = 0;
6:    auto int Auto = 0;
7:
8:    printf("Static = %d\n", Static++);
9:    printf("Auto   = %d\n", Auto++);
10:}
11:
12:main()
13:{
14:   int i = 0;
15:
16:   while(i<=3){
17:         Func();
18:         i++;
19:   }
20:}
```

و يمكن كتابة المتغير Auto بدون استخدام الكلمة المحجوزة auto.

2.9.10.5 الكلمة المحجوزة register:

أيضا تستعمل مع المتغيرات العددية، و لا يمكن استعمالها مع مصفوفات أو بنيات أو متغيرات خارجية أو متغيرات ساكنة، استعمال متغيرات بها الكلمة المحجوزة register تعني وضع تلك المتغيرات في سجل الحاسوب، و سجل الحاسوب موجود في وحدة المعالجة المركزية CPU (*Central Processing Unit*)، مثال:

```
1:#include<stdio.h>
2:
3:main()
4:{
5:    register Reg_Var = 4;
6:
7:    printf("Reg_Var = %d\n", Reg_Var);
8:}
```

و لا يمكن استعمال الإدخال لمتغيرات السجل.

2.9.11 الكلمة المحجوزة sizeof:

تستعمل الكلمة المحجوزة sizeof لمعرفة أحجام البيانات، و منها يمكن معرفة الحجم الكامل المستعمل لبرامج، تكون النتيجة بالبايت *bytes*، مثال لمعرفة أحجام أنواع المتغيرات:

```
1:#include<stdio.h>
2:
3:
4:main()
5:{
6:    int SizeOfArray[800];
7:
8:    printf("short       = %d Byte(s)\n", sizeof(short));
9:    printf("int     = %d Byte(s)\n", sizeof(int));
10:   printf("unsigned    = %d Byte(s)\n", sizeof(unsigned));
11:   printf("signed      = %d Byte(s)\n", sizeof(signed));
12:   printf("long        = %d Byte(s)\n", sizeof(long));
13:   printf("float       = %d Byte(s)\n", sizeof(float));
14:   printf("double      = %d Byte(s)\n", sizeof(double));
15:   printf("SizeOfArray = %d Byte(s)\n", sizeof(SizeOfArray));
16:}
```

وهنا ستتعرف على أحجام أنواع المتغيرات التي تريدها، و يمكن أيضا معرفة حجم مصفوفة كما في المثال.

2.9.12 الكلمة المحجوزة volatile:

الكلمة المحجوزة volatile هي معاكسة للكلمة المحجوزة const، تستعمل مع المتغيرات و هي تعني أن المتغير يمكن أن تتغير قيمته أثناء تشغيل البرنامج، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    volatile int Vol_Var = 10;
5:
6:    printf("Vol_Var = %d\n", Vol_Var);
7:}
```

2.9.13 استدعاء دالة لنفسها:

يمكننا إضافة هذه الطريقة إلى التكرار، حيث نستدعي دالة من نفسها، إن لم تكـــون شـــروط في الدالـــة فستكون الدالة بلا نهاية، مثال:

```
1:#include <stdio.h>
2:
3:void Func(int num)
4:{
5:    printf("%d\n", num);
6:    Func(num+1);
7:}
8:
9:main()
10:{
11:   Func(5);
12:}
```

هنا استدعينا الدالة لنفسها في السطر السادس، و في هذا البرنامج سيتم التكرار إلى أن يصل إلى الحد القصى من القيم التي يمكن أن يحملها نوع المتغير، و ها سيتوقف البرنامج عند الرقم *65535* لأنه العـــدد الأقـــصى الذي يمكن أن يحمله المتغير `int num`، لنجعل الدالة تقوم بتركرار محدود نستعمل شرط مثل التالي:

```
1:#include <stdio.h>
2:
3:void Func(int num)
4:{
5:    if(num <= 10)
6:    {
7:          printf("%d\n", num);
8:          Func(num+1);
9:    }
10:}
11:
12:main()
13:{
14:   Func(5);
15:}
```

هنا سيقوم البرنامج بالتكرار من العدد *5* إلى العدد *10*.

2.9.14 التحكم في طباعة النتائج:

في الدالة printf، يمكننا التحكم في طريقة طبع النتائج، سواء كانت النتائج عبارة عن أرقام أو حـــروف، فمثلا إذا كان لدينا عدد حقيقي له أربعة أرقام وراء الفاصلة و لا نريد طباعة إلا إثنين منها نستعمل الطرق التالية:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    char str[] = "Hello, World!";
5:    float flt  = 3.1415F;
6:    int dec    = 1000;
7:
8:    /*String*/
9:    printf("%s\n", str);
10:   printf("%5s\n", str);
11:   printf("%-5s\n", str);
12:   printf("%5.5s\n", str);
13:   printf("%-5.5s\n", str);
14:
15:   printf("\n");
16:
17:   /*Floating*/
18:   printf("%10.0f\n", flt);
19:   printf("%.1f\n", flt);
20:   printf("%.2f\n", flt);
21:   printf("%.3f\n", flt);
22:   printf("%.4f\n", flt);
23:}
```

جرب كل واحدة من الطرق السابقة حتى تستنتج كل طريقة و فائدتها.

2.9.15 الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

2.9.16 تمارين:

1- أكتب برنامج يقوم بالتحويل من النظام العشري إلى النظام السداسي عشر.

الفصل الثالث - التقدم في لغة C

3.1 الحساب *Enumeration*

3.2 *Command-line Arguments*

3.3 التوجيهات *Directives(Preprocessor)*

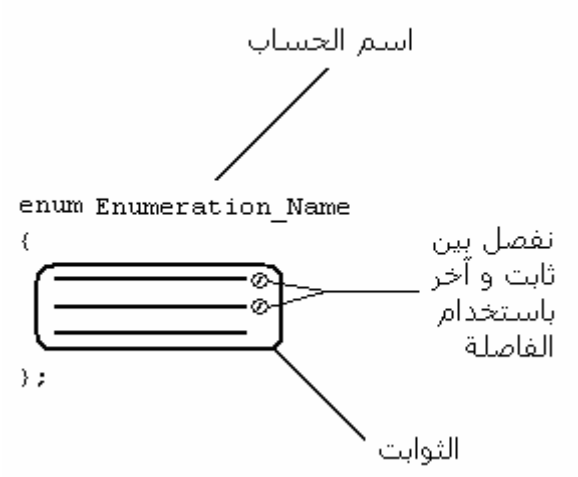
3.4 دوال ذات وسائط غير محددة

3.5 المكتبة القياسية *Standard Library*

3.1 الحساب Enumeration

الحساب *(Enumeration)* هو مجموعة من ثوابت أعداد صحيحة int، و يتم الإعلان عنها باستخدام الكلمة المحجوزة enum و التي هي مختصرة من *enumeration*، و هي مشابها للبنية في الإعلان عنها و طريقة التعامل معها.

و هذه صورة توضيحية لطريقة استعمال الحسابات:



3.1.1 اسم الحساب Enumeration Name:

لاسم الحساب شروط هي نفسها الشروط السابقة للمتغيرات، البنيات و الدوال، و الشروط هي:

- أن لا يتجاوز اسم الحساب أكثر من *31* حرف.
- أن لا يبدأ اسم الحساب بأرقام.
- أن لا يكون اسم الحساب يحتوي على مؤثرات مثل الجمع، الطرح، ....
- أن لا يكون اسم الحساب يحتوي على رموز مثل % و # و } و...(باستثناء الرمز _).
- أن لا يكون اسم الحساب مستعمل سابقا لاسم دالة، متغير، بنية أو حساب آخر.
- أن لا يكون اسم الحساب من أحد أسماء الكلمات المحجوزة.

3.1.2 ثوابت الحساب:

ثوابت الحساب يمكن أن تحميل قيم أعداد صحيحة فقط، و يتم الفصل بين ثوابت و آخر باستعمال الفاصلة. إذا لم نحدد لتلك الثوابت قيم فسيتم إعطاءها قيم إفتراضية متسلسلة تبدأ من الصفر للثابت الأول ثم تتزايد حسب عدد الثوابت.

سنقوم الآن بكتابة أبسط مثال لكيفية استعمال الكلمة المحجوزة enum و هو:

```
1:#include<stdio.h>
2:
3:enum _COLOR
4:{
5:    RED = 0,
6:    BLUE,
7:    GREEN
8:};
9:
10:main()
11:{
12:   enum _COLOR Color;
13:
14:   /*Color = RED;*/
15:   Color = BLUE;
16:   /*Color = GREEN;*/
17:
18:   if(Color == RED){
19:         printf("Color = %d\n", Color);
20:   }else if(Color == BLUE){
21:         printf("Color = %d\n", Color);
22:   }else if(Color == GREEN){
23:         printf("Color = %d\n", Color);
24:   }else
25:         printf("Error!\n");
26:}
```

و يمكن أيضا كتابة المثال بالطريقة التالية:

```
1:#include<stdio.h>
2:
3:enum _COLOR
4:{
5:    RED = 0,
6:    BLUE,
7:    GREEN
8:}Color;
9:
10:main()
11:{
12:   /*Color = RED;*/
13:   Color = BLUE;
14:   /*Color = GREEN;*/
15:
16:   if(Color == RED){
17:         printf("Color = %d\n", Color);
18:   }else if(Color == BLUE){
19:         printf("Color = %d\n", Color);
20:   }else if(Color == GREEN){
21:         printf("Color = %d\n", Color);
22:   }else
23:         printf("Error!\n");
24:}
```

و يمكن أيضا إعطاء القيمة المختار مباشرة عند الإعلان عن معرف للحساب _COLOR مثل كتابة:

```
enum _COLOR Color = BLUE;
```

أو كتابة:

```
enum _COLOR
{
      RED = 0,
      BLUE,
      GREEN
}Color = BLUE;
```

سنشرح الآن المثال السابق مع القليل من التفاصيل.

قلنا سابقا أنه عند استعمال الكلمة المحجوزة enum بها قيم فهذا يعني أن تلك القيم ثابتة و لا يمكن التغير فيها، و يمكن تسمية أعضاء الكلمة المحجوزة enum بثوابت.

و ثوابت الكلمة المحجوزة enum نقوم بكتابة أسماءها بدون كتابة نوعها، حيث قلنا سابقا أن نوعها هو أعداد صحيحة int.

أعطينا لثابت RED القيمة *0* و هذا يعني أن الثابت BLUE يساوي *1* و الثابت GREEN يساوي *2*. و يمكن أن لا نعطي للثابت RED القيمة *0*، لأنه يتم إعطاءه القيمة *0* تلقائيا إن لم تكن لديه قيمة سابقة.

أما إذا أعطينا لثابت BLUE القيمة *0* فهنا ستصبح جميع الثوابت السابقة تحمل القيمة *0* إن لم تكن لديها قيم، و هذا يعني أن الثابت RED سيحمل القيمة *0* و الثابت BLUE أيضا يحمل القيمة *0* أما الثابت GREEN فسيحمل القيمة *1*.

و يمكن أن نعطي أكثر من معرف للحسابات مثل البنيات، مثال:

```
1:#include<stdio.h>
2:
3:enum _COLOR
4:{
5:    RED = 0,
6:    BLUE,
7:    GREEN
8:};
9:
10:main()
11:{
12:   enum _COLOR cRed = RED,
13:         cBlue = BLUE,
14:         cGreen = GREEN;
15:}
```

أو الإعلان عنها مباشرة بعد الحساب مثل:

```
1:#include<stdio.h>
2:
3:enum _COLOR
4:{
5:    RED = 0,
6:    BLUE,
7:    GREEN
8:}cRed = RED, cBlue = BLUE, cGreen = GREEN;
9:
10:main()
11:{
12:
13:}
```

و توجد طرق أخرى كثيرة يمكن استعمالها مع الحسابات منها إعطاء لمتغير قيمة حساب مثل:

```
1:#include<stdio.h>
2:
3:enum _NUMBER
4:{
5:    Num = 10,
6:    Num2 = 20
7:};
8:
9:main()
10:{
11:   enum _NUMBER Number = Num;
12:   int i = 0;
13:
14:   printf("i      = %d\n", i);
15:   i = Number;
16:
17:   printf("Number = %d\n", Number);
18:   printf("i      = %d\n", i);
19:}
```

و يمكن أيضا استعمال الإدخال على الحساب مثل:

```
1:#include<stdio.h>
2:
3:enum _COLOR
4:{
5:    BLACK,
6:    WHITE,
7:    RED,
8:    GREEN,
9:    BLUE
10:};
11:
12:main()
13:{
14:   enum _COLOR Color;
15:
16:   printf("colors:\n");
17:   printf("0)BLACK\n1)WHITE\n2)RED\n3)GREEN\n4)BLUE\n");
18:   printf("choose a color:");
19:   scanf("%d", &Color);
```

```
20:
21:   if(Color == BLACK){
22:         printf("Your choice are Black\n");
23:   }else if(Color == WHITE){
24:         printf("Your choice are White\n");
25:   }else if(Color == RED){
26:         printf("Your choice are Red\n");
27:   }else if(Color == GREEN){
28:         printf("Your choice are Green\n");
29:   }else if(Color == BLUE){
30:         printf("Your choice are Blue\n");
31:   }
32:}
```

3.1.3 **الأخطاء المحتملة:**

1- لا يمكن استعمال اسم الحساب مباشرة مثل:

```
1:#include<stdio.h>
2:
3:enum _NUMBER
4:{
5:    Zero,
6:    One,
7:};
8:
9:main()
10:{
11:   enum _NUMBER = ZERO;
12:}
```

2- لا يمكن الإعلان عن حسابين من نفس الاسم.

3.1.4 **تمارين:**

1- أكتب برنامج به حساب باسم `_POWER` به ثابتين، الأول باسم `Off` و الثاني باسم `On`، ثم قم باستخدام معرفة للحساب `_POWER` باسم `Power` و استعملها للإدخال مباشرة باستخدام الدالة `scanf`، فإذا كانت النتيجة المدخل *0* أي `Off` فسيتم الخروج من البرنامج، أما إذا كانت العكس أي *1 (On)* فسيبقى البرنامج يستمر حتى تصبح القيمة المدخلة للحساب `Power` هي *0*.

Command-line Arguments 3.2

قلنا سابقا أن الدالة الرئيسية main منها يبدأ البرنامج، و منها ينتهي البرنامج، و استعملنا سابقا الدالة الرئيسية، و في كل مثال كتبناه كانت الدالة الرئيسية فارغة الوسائط (main(void، و الآن سنعرف كيفية تمرير وسائط لدالة الرئيسية حيث تسمى تلك الوسائط بـ *Command-line Arguments*، و أهم الوسائط هما إثنين، الأول هو متغير لعدد سلسلة من سلاسل حرفية و الثاني سلسلة من سلاسلة حرفية.

سنبدأ الآن بأبسط مثال حول أول وسيط لدالة الرئيسية، و هو كتابة المتغير int argc كوسيط لدالة main، و يمكن تسميته كما نريد و لكن اسمه مستعمل بكثرة بهذه الطريقة حيث الكلمة argc مكون من قسمين، القسم الأول هو *argument* و القسم الثاني هو *count*، و هذا مثال يوضح طريقة استعمالها:

```
1:#include<stdio.h>
2:
3:main(int argc){
4:    printf("argc = %d\n", argc);
5:}
```

و استعمال الوسيط argc لوحده لا يعني شيء، لذا نستعمل وسيط آخر يكون سلسلة من سلاسل حروف أي مؤشر لمصوفة، و غالبا ما يكون اسمها argv و هو مختصر من *argument vector*، و قيمة المتغير argc هي عدد السلاسل الحرفي التي هي موجودة في المصوفة argv، مثال لطريقة استعمال المصفوفة argv:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    printf("argv[argc] = %s\n", argv[argc-1]);
5:}
```

هنا سيتم طباعة مسار الملف التنفيذي مع اسمه، و نقوم بكتابة argc-1 لأننا سنتعمل الجزء الأول من السلسلة الحرفية حيث يبدأ من الصفر أي إذا مررنا المتغير argc مباشرة فذلك يعني أننا كتبنا argv[1] لأن المتغير argc يبدأ من *1*، و يمكن كتابة argv[0].

3.2.1 الأخطاء المحتملة:

لا توجد أخطاء محتملة في هذا الدرس.

3.2.2 تمارين:

لا توجد تمارين في هذا الدرس.

3.3 التوجيهات (Directives(Preprocessor

التوجيهات هي عبارة عن تعليمات خاص بالمترجم يتم تنفيذها قبل البدأ في الترجمة، و يمكن تميزها دائما بالرمز # في بداية اسم كل تعليمة، و هي كثيرة و سنتحدث عن كل منها مع البعض من التفاصيل.

3.3.1 التوجيه #include:

يعتبر هذا التوجيه من أهم التوجهات، و قد قمنا باستعمله و دراسته سابقا، و هو يطلب من المترجم بضم محتويات ملفات رأسية إلى مشروعنا، حيث يمكن استعمل دوال و ثوابت تلك الملفات الرأسية منها مباشرة بدون إعداة كتابتها. و طريقة استعملها طريقتين هما:

الطريقة الأولى هي ضم ملفات رأسية خاصة بالمترجم، و غالبا ما تكون في مجلد باسم *include*، في هذه الحالة نكتب:

```
1:#include<FileName.h>
```

و هنا نكتب اسم الملف الذي نريد ضمه إلى مشروعنا في مكان FileName.

الطريقة الثانية هي ضم ملف رأسي موجود في نفس المجلد الموجود به مشروعنا، و طريقة ضم الملفات في هذه الحالة تكون كالتالي:

```
1:#include"FileName.h"
```

و الفرق الوحيد بين الطريقتين هو كتابة الرمزين <> عندما نريد إضافة ملف رأسي من المجلد المترجم *(include)*، و كتابة الإقتباسين " " في حالة أن الملف الرأسي موجود في نفس المجلد المشروع.

و يمكن أن نقول أن التوجيه #include خاص بربط الملفات ببعضها البعض.

3.3.2 التوجيه #define:

يستعمل التوجيه #define في الإعلان الثوابت و التي تحدثنا عنها سابقا، و أيضا يستعمل في الإعلان عن المختصرات *Macros*، بالنسبة للإعلان عن الثوابت باستخدام التوجيه #define فهي كتالي:

```
#define Constant_Name Constant_Value
```

في مكان Constant_Name نقوم بكتابة اسم الثابت، و في مكان Constant_Value نكتب قيمة الثابت. و أما عن نوع القيمة المعطاة للثابت فإن التوجيه #define يقوم يقوم بتحديدها تلقائيا حيث يمكن استعمل أي نوع من الأنواع، مثال:

```
1:#define String        "Hello"
2:#define Character     'H'
3:#define fNumber       3.14
4:#define iNumber      10
```

أما بالنسبة للمختصرات *Macros* فهي مشابه للدوال، و طريقة استعملها كتالي:

```
1:#define Macro_Add(i, j) i+j
```

هنا قمنا بإنشاء مختصر به وسيطين، (و أيضا هنا لا نقوم بتحديد نوع تلك الثوابت) و قيمته هي الجمع بين الثابت i و الثابت j، مثال كامل:

```
1:#include<stdio.h>
2:
3:#define Macro_Add(i, j) i+j
4:
5:main(int argc, char *argv[]){
6:    int l;
7:
8:    l = Macro_Add(10, 20);
9:
10:   printf("%d\n", l);
11:
12:   /*or*/
13:
14:   printf("%d\n", Macro_Add(10, 20));
15:}
```

3.3.3 التوجيه #undef:

التوجيه #undef هو معاكس لتوجيه #define، حيث تقوم بإلغاء الثوابت و المختصرات التي نريد أن نقوم بإلغاءها و إعادة تعرفيها، و لكن في حالة الثوابت فيجب أن تلك الثوابت معلنة باستخدام التوجيه #define، مثال:

```
1:#include<stdio.h>
2:
3:#define Constant           5
4:#define Macro_Add(i, j) i+j
5:
6:#undef Constant
7:#undef Macro_Add
8:
9:main(int argc, char *argv[]){
10:   printf("%d\n", Constant); /*Error C2065: 'Constant':
11:                                        undeclared identifier*/
12:
```

```
13:   Macro_Add(5, 10);        /*Errors*/
14:}
```

في السطر الثالث قمنا بالإعلان عن الثابت Constant و أعطيناه القيمة 5، و في السطر الرابع قمنا بالإعلان عن المختصر Macro_Add، ثم قمنا بإلغاء كل من الثابت و المختصر في السطر السادس و السطر السابع، و عند استعمال الثابت Constant أو المختصر Macro_Add بعد إلغاءهما فسينبهك المترجم عن وجود آخطاء.

3.3.4 التوجيهات #if، #elif، #else و #endif:

هي توجيهات تستعمل لعمليات شرطية قبل الترجمة، و طريقة استعملها هي نفس الطريقة معا كل من if و else، فهنا التوجيه #if مشابه لشرط if، و التوجيه #elif مشابه لاستعمال else if معا، و أخير التوجيه #else و هو أيضا مشابه لشرط else. و التوجيه #endif يعني نهاية الشروط، مثال:

```
1:#include<stdio.h>
2:
3:#define dNum    20
4:
5:#if dNum == 10
6:    #define Result "dNum = 10"
7:#elif dNum == 20
8:    #define Result "dNum = 20"
9:#else
10:   #define Result "dNum = ??"
11:#endif
12:
13:main(int argc, char *argv[]){
14:   printf("%s\n", Result);
15:}
```

3.3.5 التوجيه #ifdef و التوجيه #ifndef:

يستعمل التوجيه #ifdef في حالة أننا أردنا أنرى نرى إن كان هناك ثابت أو مختصر معرف سابق، مثال:

```
1:#include<stdio.h>
2:
3:#define dNum
4:
5:#ifdef dNum
6:    #define Result "dNum Defined"
7:#else
8:    #define Result "dNum Undefined"
9:#endif
10:
11:main(int argc, char *argv[]){
12:   printf("%s\n", Result);
13:}
```

هنا ستكون نتيجة البرنامج هي dNum Defined، لأننا قمنا بالإعلان عن الثاتب dNum سابقا، و يمكن كتابة المثال السابق بهذه الطريقة:

```
1:#include<stdio.h>
2:
3:#define dNum
4:
5:#if defined dNum
6:    #define Result "dNum Defined"
7:#else
8:    #define Result "dNum Undefined"
9:#endif
10:
11:main(int argc, char *argv[]){
12:   printf("%s\n", Result);
13:}
```

و التوجيه `#ifndef` معاكس لتوجيه `#ifdef`، مثال:

```
1:#include<stdio.h>
2:
3:#define dNum
4:
5:#ifndef dNum
6:    #define Result "dNum Undefined"
7:#else
8:    #define Result "dNum Defined"
9:#endif
10:
11:main(int argc, char *argv[]){
12:   printf("%s\n", Result);
13:}
```

و يمكن أيضا كتابة هذا مثال بالطريقة التالي:

```
1:#include<stdio.h>
2:
3:#define dNum
4:
5:#if !defined dNum
6:    #define Result "dNum Undefined"
7:#else
8:    #define Result "dNum Defined"
9:#endif
10:
11:main(int argc, char *argv[]){
12:   printf("%s\n", Result);
13:}
```

3.3.6 التوجيه `#line`:

يستعمل التوجيه `#line` لتحديد سطر للملف الحالي أو سطر لملف غير الحالي، و ذلك غير مهم إلا في عملية بناء المترجمات، و طريقة استعمله كالتالي:

```
1:#line LineNumber
```

هذه الطريقة في حالة أردنا تحديد رقم سطر للملف الحالي، حيث نكتب رقم السطر في مكان LineNumber، و الطريقة الأخرى هي:

```
1:#line LineNumber "FileName"
```

و عند استعمال هذه الطريقة يجب دائما كتابة اسم الملف دالة الاقتباسين " ".

ستفهم طريقة استعمال هذا التوجيه أكثر عندما تعرف المختصرات المعرفة(سندرسها فيما بعد).

3.3.7 التوجيه #error:

يساعد هذا التوجيه في تنبيه المبرمج في مرحلة الترجمة مباشرة، لأخطاء يقوم بتجهيزها المبرمج في حالة الوقوع فيها، مثال:

```
1:#ifndef dNum
2:#error dNum are not declared!
3:#endif
4:
5:main(int argc, char *argv[]){
6:
7:}
```

هنا البرنامج لم يعمل و الخطأ هو الرسالة المكتبة بعد التوجيه #error، أي أن الثابت dNum غير معرف سابقا.

3.3.8 التوجيه #pragma:

يستعمل التوجيه #pragma لتحكم في المترجم حسب رغبة المبرمج، و هي تستختلف من مترجم لآخر، يستحسن مراجعة المساعد الخاص بالمترجم الذي تستعمله.

3.3.9 الأسماء المعرفة Predefined Names:

هي مجموعة من المختصرات *Macros* جاهزة، و يمكن تميزها بالرمزين *Underscore* في بداية و نهاية اسم المختصر، و هذا جول لجميع الأسماء المعرفة:

المختصر	الشرح
__LINE__	ثابت عشري يحمل رقم سطر المصدر الحالي
__FILE__	سلسلة حرفية تحمل اسم الملف الجاري ترجمته
__TIME__	سلسلة حرفية تحميل الوقت الذي تم فيه الترجم

__DATE__	سلسلة حرفية تحميل التاريخ الذي
__STDC__	تكون قيمة هذا الثابت *1* إذا كان المترجم المستعمل مترجم قياسي للغة *C*.

مثل حول طريقة استعمال تالك الأسماء:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    printf("Line: %d\n", __LINE__);
5:    printf("File: %s\n", __FILE__);
6:    printf("Date Of Compilation: %s\n", __DATE__);
7:    printf("Time Of Compilation: %s\n", __TIME__);
8:    printf("ANSI C(0=false, 1=true): %d\n", __STDC__);
9:}
```

3.3.10 **الأخطاء المحتملة:**

لا توجد أخطاء محتملة في هذا الدرس.

3.3.11 **تمارين:**

لا توجد تمارين في هذا الدرس.

3.4 دوال ذات وسائط غير محددة

درسنا سابقا الدوال، و قلنا أنه يمكن عمل لها وسائط و عند استعمال تلك الدال يجب أولا أن نـــوفر لهـــا الوسائط التي تريدها، فمثلا لا يمكن عمل مثلما هو موضح في هذا المثال:

```
1:#include<stdio.h>
2:
3:int Add(int a, int b);
4:
5:main(int argc, char *argv[]){
6:    int a = 10, b = 20, c = 30, d;
7:
8:    d = Add(a, b, c);
9:
10:   printf("%d\n", d);
11:}
12:
13:int Add(int a, int b){
14:   return a+b;
15:}
```

هنا البرنامج لن يعمل، و الخطأ في السطر الثامن و هو أن الدالة Add تحتوي على وسيطين فقط، أما نحن فقد مررنا لها ثلاثة وسائط.

و ستستنتج أن الدالتين printf و الدالة scanf لهما وسائط غير محددة و يمكن أن نزيد عليهـــا وســـائط حسب رغباتنا، و سنقوم الآن بعمل استنساخ صغير لدالة printf و جعلها أكثر مرونة من السابقة:

```
1:#include<stdio.h>
2:#include<stdarg.h>
3:
4:void NewPrintf(char *, ...);
5:
6:main(int argc, char *argv[]){
7:    NewPrintf("%d\a, %d\a, %d\a\n", 1, 2, 3);
8:          NewPrintf("Hello!\n");
9:}
10:
11:void NewPrintf(char *format, ...){
12:   va_list Argument;
13:   char *str;
14:   int Decimal;
15:
16:   va_start(Argument, format);
17:
18:   for(str=format;*str;str++){
19:         if(*str != '%'){
20:               putchar(*str);
21:               continue;
22:         }
23:
24:         switch(*++str){
25:               case 'd':
26:               case 'i':
```

```
27:                case 'D':
28:                case 'I':
29:                      Decimal = va_arg(Argument, int);
30:                      printf("%d", Decimal);
31:                      break;
32:
33:                default:
34:                      putchar(*str);
35:                      break;
36:          }
37:    }
38:
39:    va_end(Argument);
40:}
```

أولا يجب ضم الملف الرأسي stdarg.h، و اسم الملف مختصر من *standard argument*، و هو الملف الذي سيعطينا الإيمكانيات لجعل الدوال لها وسائط غير محددة.

في السطر الرابع قمنا بالإعلان عن الدالة الجديد لطبعة باسم NewPrintf، حيث لها وسيط واحد و هو الوسيط الرئيسي، أما تلك النقاط ... فهي تعني أنه ستم تمرير لدالة عدد من الوسائط غير محددة.

في السطر الثاني عشر قمنا بالإعلان عن مؤشر لـ va_list، و هو مختصرة من *variable list*، و هو من مؤشرات الملف الرأسي stdarg.h، حيث سيحمل هذا المؤشر عدد وسائط الدالة NewPrintf.

في السطر السادس عشر قمنا باستعمال المختصر va_start، حيث له وسيطين، الوسيط الأول هو مؤشر لـ va_list، و الوسيط الثانية هو تمرير الوسيط الأول لدالة NewPrintf، و هنا سنقوم بتمرير المؤشر format.

في السطر التاسع و الشرعين قمنا باستعمال المختصر va_arg، و هو مختصر من *variable argument*، و له وسيطين، الوسيط الأول مؤشر لـ va_list، و الوسيط الثاني هو نوع المتغير الذي سيتم أخذ قيمته من مكان الوسيط الإضافي ثم نقله إلى المتغير Decimal، ثم طبعه مباشرة.

و هذا مثال لكيفية جعل المثال الأول يعمل بشكل صحيح، المثال:

```
1:#include<stdio.h>
2:#include<stdarg.h>
3:
4:int Add(int NumOfArg, ...);
5:
6:main(int argc, char *argv[]){
7:    int a = 10, b = 20, c = 30, d;
8:
9:    d = Add(3, a, b, c);
10:
```

```
11:   printf("%d\n", d);
12:}
13:
14:int Add(int NumOfArg, ...){
15:   va_list Arg;
16:   int sum = 0, i;
17:
18:   va_start(Arg, NumOfArg);
19:
20:   for(i=0;i<NumOfArg;i++)
21:         sum += va_arg(Arg, int);
22:
23:   return sum;
24:}
```

3.4.1 **الأخطاء المحتملة:**

لا توجد أخطاء محتملة في هذا الدرس.

3.4.2 **تمارين:**

لا توجد تمارين في هذا الدرس.

3.5 المكتبة القياسية Standard Library

في هذا الدرس سنتعرف على أهم ثوابت، مختصرات و دوال لغة C القياسية، و كل مجموعة منها موجـــودة في ملف رأسي ذات اسم يدل على دورها.

3.5.1 الملف الرأسي assert.h:

يحتوي هذا الملف الرأسي على دالة و هي assert، بها وسيط واحد، و هي دالة تقوم بالمقارنة بين قيمـــتين حيث تكون النتيجة إما صفر أي خاطئة، أو غير الصفر أي صحيحة، إن كانت النتيجة خاطئة أي *0* فسيتم استعداء تلك الدالة كمختصر *macro*، حيث يقوم ذلك المختصرة بطباعة الخطأ مع اسم الملف الموجود بـــه الخطأ و رقم السطر أثناء تشغيل البرنامج، و هنا سيتوقف البرنامج، أما إن كانت النتيجة غير الصفر فسيتم تجاهل المختصر assert، و لن تقوم الدالة assert بعمل شيء، مثال:

```
1:#include<stdio.h>
2:#include<assert.h>
3:
4:main(int argc, char *argv[]){
5:    assert(10<20); /*true*/
6:    assert(20<10); /*false*/
7:}
```

أولا قمنا بضم الملف الرأسي assert.h في السطر الثاني، و استعملنا الدالة assert في كل مـــن الـــسطر الخامس و السادس.

في السطر الخامس قمنا بإعطاءة لدالة assert مقارنة صحيح، و في هذه الحالة سيتم تجاهل الدالة، و لـــن يقوم البرنامج بعمل أي شيء من طرف هذه الدالة.

في السطر السادس أعطينا لدالة assert مقارنة خاطئة، و هنا سيتم التحويل إلى المختصرة assert و الذي سيقوم بطباعة الخطأ أثناء تشغيل البرنامج، و الخطأ هو *20<10* في الملف *main.c* في السطر *6*.

3.5.2 الملف الرأسي ctype.h:

الملف الرأسي ctype.h يحتوي على مجموعة من الدوال الخاص أنواع الرموز، و اسمه مختصر من *character type*، و جميع دواله ذات وسيط واحد و هو لمتغير يحمل رقم الرمز.

3.5.2.1 الدالة isalnum:

اسم الدالة مختصر من *is alphanumeric*، و تستعمل هذا الدالة لإختبار القيمة المعطاة لها، إذا كانت من القيمة لرمز حرفي أو رقمي فستكون النتيجة غير الصفر، أما في حالة العكس فنتيجة ستكون *0*، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 1, c2 = 65;
6:
7:    printf("%d = %c\n", isalnum(c), c);
8:    printf("%d = %c\n", isalnum(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة *0* أي خاطئة، لأن الرقم *1* في جدول *ASCII* عبارة عن أيقونة.

في السطر الثامن ستكون النتيجة *1* أي صحيحة، لأن الرقم *65* هو الحرف *A* في جدول *ASCII*، و هو من الرموز الحرفية.

3.5.2.2 الدالة **isalpha**:

اسم الدالة مختصر من *is alphabetic*، و تستعمل لإختار القيمة المعطاة لها إذا كانت من حروف الأبجدية أم لا، إذا كان من الحروف الأبجدية فنتيجة ستكون *1*، و في حالة العكس فستكون النتيجة *0*، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 49, c2 = 65;
6:
7:    printf("%d = %c\n", isalpha(c), c);
8:    printf("%d = %c\n", isalpha(c2), c2);
9:}
```

في السطر السابع، ستكون النتيجة *0*، لأن الرقم *49* هو العدد *1* في جدول *ASCII*، و هو ليس من الحروف الأبجدية.

في السطر الثامن ستكون النتيجة *1*، لأن العدد *65* هو الحرف *A* في جدول *ASCII*، و الحرف *A* من الحروف الأبجدية.

3.5.2.3 الدالة **iscntrl**:

اسم الدالة مختصر من *is control*، و هي تقوم بإختبارة القيمة المعطاة لها إذا كانت من رموز التحكم، و رموز التحكم تكون بين *0* و *31*، و أيضا من *128* إلى *255*، و هنا ستكون النتيجة غير الصفر أي صحيحة،

أما إذا كانت القيمة بين *32* و *127* فستكون النتيجة خاطئة لأن كل من تلك الرموز عبارة عـــن أرقـــام و حروف لا أكثر، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 65, c2 = 0;
6:
7:    printf("%d = %c\n", iscntrl(c), c);
8:    printf("%d = %c\n", iscntrl(c2), c2);
9:}
```

في السطر السابع النتيجة ستكون *0* أي خاطئة، لأن الرقم *65* يحمل الحرف *A* في جدول *ASCII* و هو ليس من رموز التحكم.

في السطر الثامن ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم *0* من رموز التحكم في جدول *ASCII*.

3.5.2.4 الدالة **isdigit**:

تقوم هذه الدالة بإختبار القيمة المعطاءة لها إذا كانت من الأرقام العشرية حيث تكون النتيجة غير الصفر إن كانت القيمة المعطاة بين الرقم *48* و *57*، و في حالة أن النتيجة لم تكن بين الرقم *48* و *57* مـــن جـــدول *ASCII* فستكون النتيجة *0* أي خاطئة، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 65, c2 = 48;
6:
7:    printf("%d = %c\n", isdigit(c), c);
8:    printf("%d = %c\n", isdigit(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة *0* أي خاطئة، لأن الرقم *65* هو الحرف *A* في جدول *ASCII* و الحـــرف *A* ليس رقما.

في السطر الثامن ستكون النتيجة غير الصفر، لأن الرقم *48* في جدول *ASCII* هو الرقم *0*.

3.5.2.5 الدالة **isgraph**:

تقوم هذا الدالة بإختبار القيمة المعطاة لها إذا كان رموز غير مرئية أو مرئية، إذا كانت غير مرئية فـــستكون النتيجة غير الصفر أي صحيحة، أما في حالة العكس فستكون النتيجة *0* أي خاطئة، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 0, c2 = 48;
6:
7:    printf("%d = %c\n", isgraph(c), c);
8:    printf("%d = %c\n", isgraph(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة *0* أي خاطئة، لأن الرقم *0* في جدول *ASCII* عبارة عن رمز مرئي.

في السطر الثامن ستكون النتيجة غير الصفر، لأن الرقم *48* في جدول *ASCII* هو الرقم *0* و هو رمز غير مرئي.

3.5.2.6 الدالة **islower**:

تقوم هذا الدالة بإختبار القيمة المعطاة لها، إذا كانت من الأحرف الصغيرة فستكون النتيجة غير الصفر أي صحيحة، أما إذا كانت غير ذلك فستكون النتيجة *0*، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 65, c2 = 97;
6:
7:    printf("%d = %c\n", islower(c), c);
8:    printf("%d = %c\n", islower(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة *0* أي خاطئة، لأن الرقم *65* هو الرمز *A* في جدول *ASCII*، و الرمز *A* ليس من الحروف الصغيرة.

في السطر الثامن ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم *97* في جدول *ASCII* هو الرمز a و هو من الرمز الصغيرة.

3.5.2.7 الدالة **isprint**:

تقوم هذه الدالة بإختبار القيمة المعطاة لها، إذا لم كانت القيمة بين *32* و *126* من جدول *ASCII* فستكون النتيجة غير الصفر، و هي الرموز المستعملة في الطباعة الإفتراضية (الفراغ يعتبر رمز أيضا)، أما إذا كانت القيم غير ذلك فستكون النتيجة *0* أي خاطئة، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
```

```
4:main(int argc, char *argv[]){
5:    int c = 1, c2 = 65;
6:
7:    printf("%d = %c\n", isprint(c), c);
8:    printf("%d = %c\n", isprint(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة *0*، لأن الرقم *1* في جدول *ASCII* عبارة عن رسم لإبتسامة، و لا تستعمل الإبتسامات في النصوص الإفتراضية.

في السطر الثامن ستكون النتيجة غير الصفر، لأن الرقم *65* هو الحرف *A* في جدول *ASCII* و هو حرف يمكن استعماله في النصوص الإفتراضية.

3.5.2.8 الدالة **ispunct**:

اسم الدالة مختصر من *is punctuation*، و هي تقوم بإختبار القيمة المعطاة لها، إذا كانت من أحد رموز الترقين فستكون النتيجة غير الصفر أي صحيحة، أما غير رموز الترقين فستكون النتيجة *0* أي خاطئة، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 44, c2 = 65;
6:
7:    printf("%d = %c\n", ispunct(c), c);
8:    printf("%d = %c\n", ispunct(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم *44* هو الفاصلة في جدول *ASCII*، الفاصلة من رموز الترقين.

في السطر الثامن ستكون النتيجة صفر أي خاطئة، لأن الرقم *65* هو الحرف *A* في جدول *ASCII*، و الحروف ليست رموز ترقين.

3.5.2.9 الدالة **isspace**:

تقوم الدالة بإختبار القيمة المعطاة لها، إذا كانت من رموز الفضاء الأبيض أما لا، و رموز الفضاء الأبيض تبدأ من *9* إلى *13*، و في هذه الحالة ستكون النتيجة غير الصفر أي صحيح، أما إن لم تكون القيمة المعطاة من أحد الرموز الفضاء الأبيض فإن النتيجة ستكون *0* أي خاطئة، مثال:

```
1:#include<stdio.h>
```

```
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 9, c2 = 97;
6:
7:    printf("%d = %c\n", isspace(c), c);
8:    printf("%d = %c\n", isspace(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم *9* في جدول *ASCII* هو من أحد رموز الفضاء الأبيض.

في السطر الثامن ستكون النتيجة *0* أي خاطئة، لأن الرقم *97* في جدول *ASCII* هو الحرف *a*، و هو ليس من رموز الفضاء الأبيض.

3.5.2.10 الدالة **isupper**:

تقوم هذا الدالة بإختبار القيمة المعطاة لها، إذا كانت من الرموز الكبيرة فستكون النتايجة غير الصفر أي صحيحة، أما إذا كانت قيم لأحرف صغيرة فستكون النتيجة *0* أي خاطئة، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 65, c2 = 97;
6:
7:    printf("%d = %c\n", isupper(c), c);
8:    printf("%d = %c\n", isupper(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم *65* في جدويل *ASCII* هو الحرف *A* و هو من الأحرف الكبيرة.

في السطر الثامن ستكون النتيجة *0* أي خاطئة، لأن الرقم *97* في جدول *ASCII* هو الحرف *a* و هو ليس حرف كبير.

3.5.2.11 الدالة **isxdigit**:

اسم هذا الدالة مختصر من *is hexadecimal digital*، و تقوم هذا الدالة بإختبار الدالة المعطاة لها، إذا كانت القيمة من أعداد النظام السداسي عشر أي من *A-F* و من *0 – 9* فستكون النتيجة غير الصفر أي صحيحة، أما إذا كانت غير تلك القيم فستكون النتيجة *0* أي خاطئة، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
```

```
3:
4:main(int argc, char *argv[]){
5:    int c = 70, c2 = 71;
6:
7:    printf("%d = %c\n", isxdigit(c), c);
8:    printf("%d = %c\n", isxdigit(c2), c2);
9:}
```

في السطر السابع ستكون النتيجة غير الصفر أي صحيحة، لأن الرقم *70* هو الحرف *F* في جدول *ASCII*، و الحرف *F* من النظام السداسي عشر و في النظام العشري هو العدد *15*.

في السطر الثامن ستكون النتيجة صفر أي خاطئة، لأن الرقم *71* هو الحرف *G* في جدول *ASCII*، و هو الحرف *G* ليس من النظام السداسي عشر.

3.5.2.12 الدالتين toupper و tolower:

الدالة toupper تقوم بتحويل الأحرف الصغير إلى الأحرف الكبيرة، و الدالة tolower تقوم بتحويل الأحرف الكبيرة إلى أحرف صغيرة أي عكس ما تقوم به الدالة toupper، مثال:

```
1:#include<stdio.h>
2:#include<ctype.h>
3:
4:main(int argc, char *argv[]){
5:    int c = 'a', c2 = 'B';
6:
7:    printf("%c = %c\n", c, toupper(c));
8:    printf("%c = %c\n", c2, tolower(c2));
9:}
```

في هذا البرنامج يتم طباعة الحرف *a* و هو على شكل حرف صغير، ثم إعادة كتابته بعد تمريره لدالة toupper حيث سيصبح من الأحرف الكبيرة، و ستم طباعة الحرف *B* على شكل حرف كبير، ثم إعادة كتابته بعد تمريره لدالة tolower حيث تحويله إلى حرف صغير.

3.5.3 الملف الرأسي errno.h:

يوجد في الملف الرأسي errno.h مجموعة من المختصرات تستعمل لكشف الأخطاء أثناء وقت تشغيل البرنامج. حيث تستعمل تلك المختصرات مع الدالة perror.

3.5.3.1 الدالة perror:

اسم الدالة perror مختصر من *print error*، و هي دالة تقوم بطباعة أخطاء خاصة بالملفات.

لدالة وسيط واحد و هو سلسلة لحروف التي ستم طباعتها قبل طباعة الخطأ، أو يمكن ترك السلسل فراغـــة إن لم نريد إضافة شيء إلى رسالة الخطأ، مثال حول طريقة استعمال الدالة:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:#include<errno.h>
4:
5:main(int argc, char *argv[]){
6:    FILE *File;
7:    char FileName[255];
8:
9:    perror("situation");
10:
11:   printf("Enter The File Name: ");
12:   gets(FileName);
13:
14:   File = fopen(FileName, "r");
15:
16:   if(!File){
17:         perror("situation");
18:         printf("errno = %d\n", errno);
19:         exit(1);
20:   }else{
21:         perror("situation");
22:         printf("File Founded!\n");
23:         fclose(File);
24:   }
25:}
```

في السطر التاسع قمنا باستعمال الدالة perror مع تمرير لها النص *situation* و التي ستكون هذه الكلمة قبل رسالة الخطأ، و لكن هنا لا يوجد أي خطأ و الرسال ستكون *No error*.

استعمال الدالة perror في السطر السابع عشر، فإن كان اسم الملف الذي طلبه المستخدم غـــير موجـــود فسيتم طباعة الرسالة عدم وجود الملف، و أيضا قمنا باستعمال المختصر errno في السطر الثامن عشر عبر الدالة printf، حيث ستم طباعة رقم رسالة الخطأ الذي تم إيجاده، رسالة الخطأ هنا رقمها *2*.

و للمختصر errno ثوابت تستعمل لتبين على رسالة الخطأ التي نريد إرسالها، من أهم تلك الثوابت هي:

اسم الثابت	الرسالة
EINVAL	*Invalid argument*
ERANGE	*Result too large*
EBADF	*Bad file descriptor*
EDOM	*Domain error*
EACCES	*Permission denied*
EMFILE	*Too many open files*
EEXIST	*File exists*
E2BIG	*Arg list too long*
EXDEV	*Improper link*

No child processes	ECHILD
Resource temporarily unavailable	EAGAIN
Exec format error	ENOEXEC
Not enough space	ENOMEM
No such file or directory	ENOENT
No space left on device	ENOSPC

و توجد رسائل أخرى يمكن رأيتها باستخدام التكرار، مثال:

```
1:#include<stdio.h>
2:#include<errno.h>
3:
4:main(int argc, char *argv[]){
5:    int i;
6:
7:    for(i=0;i<=42;i++){
8:          errno = i;
9:          printf("%d:", i);
10:         perror("");
11:   }
12:}
```

و كل رقم منها له ثابت معرف في الملف الرأسي errno.h، و هي على الشكل التالي:

```
1:#define EPERM           1
2:#define ENOENT          2
3:#define ESRCH           3
4:#define EINTR           4
5:#define EIO             5
6:#define ENXIO           6
7:#define E2BIG           7
8:#define ENOEXEC         8
9:#define EBADF           9
10:#define ECHILD          10
11:#define EAGAIN          11
12:#define ENOMEM          12
13:#define EACCES          13
14:#define EFAULT          14
15:#define EBUSY           16
16:#define EEXIST          17
17:#define EXDEV           18
18:#define ENODEV          19
19:#define ENOTDIR         20
20:#define EISDIR          21
21:#define EINVAL          22
22:#define ENFILE          23
23:#define EMFILE          24
24:#define ENOTTY          25
25:#define EFBIG           27
26:#define ENOSPC          28
27:#define ESPIPE          29
28:#define EROFS           30
29:#define EMLINK          31
30:#define EPIPE           32
31:#define EDOM            33
32:#define ERANGE          34
33:#define EDEADLK         36
34:#define ENAMETOOLONG    38
```

```
35:#define ENOLCK          39
36:#define ENOSYS          40
37:#define ENOTEMPTY       41
38:#define EILSEQ          42
```

3.5.4 الملف الرأسي `float.h`:

يوجد بهذا الملف الرأسي مجموعة من الثابت خاصة بالأعداد الحقيقية، و تلك الوثابت هي:

اسم الثابت	قيمته
FLT_RADIX	*2*
FLT_ROUNDS	*1*
FLT_MAX	*1^E+37*
FLT_MAX_EXP	*128*
DBL_MAX	*1^E+37*
DBL_MAX_EXP	*1024*
FLT_DIG	*6*
DBL_DIG	*15*
FLT_EPSILON	*1^E-5*
DBL_EPSILON	*1^E-9*
FLT_MANT_DIG	*24*
DBL_MANT_DIG	*53*
FLT_MIN	*1^E-37*
FLT_MIN_EXP	*-125*
DBL_MIN	*1^E-37*
DBL_MIN_EXP	*-1021*

تعتبر هذا الثوابت من أهم الثوابت الموجود في الملف الرأســـي `float.h`، توجد ثوابت أخرى كثيرة يمكن رأيته في الملف الرأسي `float.h`.

3.5.5 الملف الرأسي `limits.h`:

بهذا الملف الرأسي مجموعة من الثوابت لأحجام جميع الأنواع المتكاملة، الثوابت هي:

اسم الثابت	قيمته
CHAR_BIT	*8*
CHAR_MIN	*-128*
INT_MIN	*-32767*
LONG_MIN	*-2147483647*
SHRT_MIN	*-32768*
SCHAR_MIN	*-128*
LONG_MAX	*2147483647*
SCHAR_MAX	*127*
SHRT_MAX	*32767*
UCHAR_MAX	*255*

UINT_MAX	*65535*
ULONG_MAX	*4294967295*
USHRT_MAX	*65535*
INT_MAX	*32767*
CHAR_MAX	*127*

و توجد ثوابت أخرى يمكن رأيتها في الملف الرأسي limits.h.

3.5.6 الملف الرأسي locale.h:

بهذا الملف الرأسي مجموعة من تراكيب، ثوابت، مختصرات و دوال مستعمل من قبل روتينات المواقع(المناطق)، ثوابت هذا الملف الرأسي على الشكل التالي:

```
1:#define LC_ALL          0
2:#define LC_COLLATE      1
3:#define LC_CTYPE        2
4:#define LC_MONETARY     3
5:#define LC_NUMERIC      4
6:#define LC_TIME         5
7:#define LC_MIN          LC_ALL
8:#define LC_MAX          LC_TIME
```

لا يعتبر هذا الملف الرأسي مهم، هو خاص بتحديد المنتطقة أو البلاد التي تريد استعمال أصنافها في برنامجك، منها اللغة.

ألقي لنظرة على الملف الرأسي locale.h لرأية باقي محتوياته.

3.5.7 الملف الرأسي math.h:

يحمل هذا الملف الرأسي مجموعة من الثوابت و المختصرات و الدوال الخاص بالرياضيات.

بالنسبة للثوابت فهي معرفة على الشكل التالي:

```
1:#define M_E        2.71828182845904523536
2:#define M_LOG2E    1.44269504088896340736
3:#define M_LOG10E   0.43429448190325182765l
4:#define M_LN2      0.69314718055994530942
5:#define M_LN10     2.30258509299404568402
6:#define M_PI       3.14159265358979323846
7:#define M_PI_2     1.57079632679489661923
8:#define M_PI_4     0.78539816339744830962
9:#define M_1_PI     0.31830988618379067154
10:#define M_2_PI     0.63661977236758134308
11:#define M_2_SQRTPI 1.12837916709551257390
12:#define M_SQRT2    1.41421356237309504880
13:#define M_SQRT1_2  0.70710678118654752440
```

أما المختصرات و الدوال فسندرس أهمها، أما الباقي فيمكن رأيتها من الملف الرأسي math.h.

3.5.7.1 الدالة **sin**:

لدالة *sin* وسيط واحد و هو لمتغير من نوع double، و هي تقوم بإرجاعة جيب *sine* القيمة التي تم تمريرها إليها، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x = 3.14/2;
6:
7:    printf("sine x = %f\n", sin(x));
8:}
```

3.5.7.2 الدالة **cos**:

اسم الدالة cos مختصر من *cosine*، و لها وسيط واحد من نوع double، و هي تقوم بإرجاع جيب التمام للقيمة التي تم تمريرها إليها، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x = 3.14/2;
6:
7:    printf("cosine x = %f\n", cos(x));
8:}
```

3.5.7.3 الدالة **tan**:

اسم الدالة tan مختصر من *tangent*، و لها وسيط واحد لمتغير من نوع double،و هي تقوم بإرجاع الظـــلّ القيمة التي تم تمريرها إليها، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x = 3.14/4;
6:
7:    printf("tangent x = %f\n", tan(x));
8:}
```

3.5.7.4 الدالة **exp**:

اسم الدالة exp مختصر من *exponential*، و لها وسيط واحد من نوع double، حيث تكون القيمة المعطـــاة له هي أسّ e، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
```

```
3:
4:main(int argc, char *argv[]){
5:    double x = 3.0;
6:
7:    printf("exponential x = %f\n", exp(x));
8:}
```

3.5.7.5 الدالة **log**:

اسم الدالة log مختصر من *logarithm*، و بها وسيط واحد و هو لمتغير من نوع double، و تقـــوم الدالـــة بإرجاع لوغاريتمية القيمة التي تم تمريرها إليها، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x = 100.0;
6:
7:    printf("logarithm x = %f\n", log(x));
8:}
```

3.5.7.5 الدالة **pow**:

اسم الدالة pow مختصر من *power*، و بها وسيطين، الوسيط الأول هو متغير من نوع double، و الثاني أيضا متغير من نوع double، حيث الوسيط الثاني يكون عدد أسّ قيمة الوسيط الثاني، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x=5.0, y=3.0;
6:
7:    printf("%f\n", pow(x, y));
8:}
```

3.5.7.6 الدالة **sqrt**:

اسم الدالة sqrt مختصر من *square*، و لها وسيط واحد لمتغير من نوع double، و هي تقوم بإرجاع الجـــذر التربيعي للقيمة المعطاة له، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x=9.0;
6:
7:    printf("%f\n", sqrt(x));
8:}
```

3.5.7.7 الدالة **ceil**:

للدالة ceil وسيط واحد و هو لمتغير من نوع double، و هي تقوم بأخذ العدد المعطاة لها على شكل عدد صحيح، فإذا كانت القيمة العطاة لها هي *3.1* فستكون النتيجة *4*، أما إذا كانت *3.0* فستكون النتيجة هـــي نفسها *3.0*، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x=9.5;
6:
7:    printf("%f\n", ceil(x));
8:}
```

3.5.7.8 الدالة floor:

الدالة floor هي معاكس لدالة ceil، فإذا كانت القيمة المعطاة لها *3.1* فستكون النتيجة *3.0*، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x=10.5;
6:
7:    printf("%f\n", floor(x));
8:}
```

3.5.7.9 الدالة fabs:

اسم الدالة fabs مختصر من *float absolute*، و تقوم هذه الدالة بإرجاع القيمة المطلقة للعدد الذي تم تمريره لها، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x=-10.5, y=10.5;
6:
7:    printf("%f\n", fabs(x));
8:    printf("%f\n", fabs(y));
9:}
```

3.5.7.10 الدالة ldexp:

لهذه الدالة وسيطين، الأول لمتغير من نوع double، و الوسيط الثاني متغير من نوع int، حيث تقوم هـــذه الدالة بضرب قيمة الوسيط الأول في *2* أسّ قيمة الوسيط الثاني، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x=2.0, y=4.0;
```

```
6:
7:    printf("%f\n", ldexp(x, y)); /*y = 2*2*2*2, x = 2; x*y = 32.000000*/
8:}
```

3.5.7.11 الدالة **fmod**:

لهذه الدالة وسيطين، كلاهما متغيران من نوع double، حيث تقوم هذه الدالة بقسمة قيمة الوسيط الأول على قيمة الوسيط الثاني، مثال:

```
1:#include<stdio.h>
2:#include<math.h>
3:
4:main(int argc, char *argv[]){
5:    double x=2.0, y=4.0;
6:
7:    printf("%f\n", fmod(x, y));
8:}
```

3.5.8 الملف الرأسي **setjmp.h**:

يحتوي هذا الملف الرأسي على دالتين هما setjmp و longjmp، حيث تحتوي الدالة setjmp على وسيط واحد و هو مؤشر لبنية jmp_buf و هي معرفة في نفس الملف الرأسي، و تحتوي الدالة longjmp على وسيطين، الأول للبنية jmp_buf و الوسيط الثاني لمتغير من نوع **int**. تستعمل الدالة setjmp مع الدالة longjmp، و الهدف منهما هو القفز إلى مختلف أماكن البرنامج، فمثلا نستعمل الكلمة المحجوزة goto للقفز من مكان لآخر في دالة معنية، و لا يمكن استعمالها للقفز العام (*Global*)، مثال:

```
1:#include<stdio.h>
2:#include<setjmp.h>
3:#include<stdlib.h>
4:
5:void Return(void);
6:jmp_buf jmp;
7:
8:main(int argc, char *argv[]){
9:    int value;
10:
11:   value = setjmp(jmp);
12:   {
13:         printf("Block 1:\n");
14:         if(value==1){
15:               printf("longjmp = %d\n", value);
16:               exit(1);
17:         }
18:   }
19:
20:   {
21:         printf("Block 2:\n");
22:         printf("Call Return...\n");
23:         Return();
24:   }
25:}
26:
27:void Return(void){
```

```
28:   longjmp(jmp, 1);
29:}
```

3.5.9 الملف الرأسي signal.h:

يحتوي هذه الملف الرأسي على دالتين هما signal و raise، و الدالة المستعملة بكثرة هي الدالة raise.

3.5.9.1 الدالة raise:

تحتوي هذه الدالة على وسيط من نوع int، و عملها هو إنهاء البرنامج في حالات مثل حدوث إنتهاك في ذاكرة الحاسوب، و توجد ثوابت خاصة بها في نفس الملف الرأسي، و كل ثابت و حالته، و هذا جدول لأهم الثوابت التي تستعمل مع هذه الدالة:

اسم الثابت	قيمته	الشرح
SIGABRT	22	في حالة الإنتهاء الغير الطبيعي للبرنامج، يستدعي الدالة abort
SIGFPE	8	في حالة خطأ رياضي
SIGILL	4	في حالة حدوث أمر غير شرعي
SIGINT	2	المقاطعة
SIGSEGV	11	في حالة حدوث إنتهاك لذاكرة
SIGTERM	15	إنهاء البرناج

3.5.10 الملف الرأسي stdarg.h:

اسم هذا الملف الرأسي مختصر من *standard argument*، و هو خاص باستعمال الوسائط المتعددة لدوال.

يحتوي الملف الرأسي stdarg.h على المؤشر va_list، و هو الذي سيحمل عدد الوسائط أو الوسائط التي سيتم استعمالها في الدالة، و يجب دائما تشغيل *initialize* المؤشر باستخدام المختصر va_start، حيث يحتوي هذا المختصر على وسيطين، الوسيط الأول هو لمؤشر va_list الذي سيتم تشغيله، و الوسيط الثاني هو اسم الوسيط الأول لدالة. و بعد ذلك نستعمل المختصر va_arg، و الذي يحتوي على وسيطين، الوسيط الأول هو مؤشر لـ va_list الذي سيتم استعماله في الدالة، و الوسيط الثاني نضع فيه نوع الوسيط الذي سيتم أخذه، و في النهاية نقوم بإنهاء تشغيل المؤشر va_list و ذلك باستخدام المختصر va_end، مثال:

```
1:#include<stdio.h>
2:#include<stdarg.h>
3:
4:void Arg_List(int list, ...); /*Function prototype*/
```

```
5:
6:main(int argc, char *argv[]){
7:    Arg_List(5, 10, 20, 30, 40, 50);
8:}
9:
10:void Arg_List(int list, ...){
11:   va_list pList;
12:   int iArg=0, i;
13:
14:   printf("There are %d argument(s)\n", list);
15:
16:   va_start(pList, list); /*initialize the pList pointer*/
17:
18:   for(i=01;i<=list;i++){
19:         iArg = va_arg(pList, int);
20:         printf("argument %d = %d\n", i, iArg);
21:   }
22:
23:   va_end(pList);
24:
25:}
```

3.5.11 الملف الرأسي stddef.h:

اسم هذا الملف الرأسي مختصر من *Standard Definition*، و هو يحتوي على الثابت NULL و هو معرف على الشكل التالي:

```
1:#define NULL    0
```

و يحتوي على المتغير size_t و هو معرف على الشكل التالي:

```
1:typedef unsigned size_t;
```

و طريقة استعماله كتالي:

```
1:#include<stdio.h>
2:#include<stddef.h>
3:
4:main(int argc, char *argv[]){
5:    size_t uInt = 65535;
6:
7:    printf("%u\n", uInt);
8:}
```

و يحتوي أيضا على المتغير ptrdiff_t، و هو معرف بطريقتين هما:

```
1:typedef long    ptrdiff_t;
```

و الطريقة:

```
1:typedef int     ptrdiff_t;
```

حيث يتم إختيار واحد من الطريتين حسب القيمة المعطاة أو المستعملة، مثال:

```
1:#include<stdio.h>
2:#include<stddef.h>
3:
4:main(int argc, char *argv[]){
5:    ptrdiff_t Int = 65535;
6:    ptrdiff_t Long = 2147483647;
7:
8:    printf("Int = %d\n", Int);
9:    printf("Long = %d\n", Long);
10:}
```

3.5.12 الملف الرأسي stdio.h:

يعبر من أهم الملفات الرأسية القياسية، و اسمه مختصر من *Standard Input Output*، حيث يحتوي على أهم الدوال و المختصرات التي يمكن استعمالها في أغلب البرامج، و دراسة جميع تلك الدوال و المختصرات بأمثل يأخذ الكثير من الصفحات لذا نكتفي بمعرفة اسماء تلك الدوال و المختصرات و الهدف منها.

ينقسم هذا الملف الرأسي إلى أصناف، لكل صنف مجموعة من الدوال و المختصرات الخاصة به.

3.5.12.1 الدالة printf:

تحتوي هذه الدالة على وسائط غير محددة، تتزايد حسب رغبة المبرمج، و هي تقوم بالتعامل مع كل من الأحرف و الأرقام و النصوص، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    int Number = 100;
5:    char Char = 'A';
6:    char String[] = "Hello, World!\n";
7:
8:    printf("%d\n", Number);
9:    printf("%c\n", Char);
10:   printf(String);
11:}
```

3.5.12.2 الدالة sprintf:

أيضا هذه الدالة تحتوي على وسائط غير محدودة، و لكن لديه وسائط إضافي في بدايتها حيث هو عبارة عن متغير لسلسلة حروف، و تقوم هذه الدالة بكتابة نص في تلك السلسلة النصية، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    char Buffer[255];
5:
6:    sprintf(Buffer, "Number = %d\nCharacter = %c\nString = %s\n",
```

```
7:          100, 'A', "Hello, World!");
8:
9:    printf(Buffer);
10:}
```

3.5.12.3 الدالة **vprintf**:

مثل الدالة printf و لكنها تأخذ الوسائط على شكل va_list، مثال:

```
1:#include<stdio.h>
2:#include<stdarg.h>
3:
4:void ArgLst(char *format, ...);
5:
6:main(int argc, char *argv[]){
7:    int Number = 100;
8:    char Char = 'A';
9:    char String[] = "Hello, World!";
10:
11:   ArgLst("Number = %d\nChar = %c\nStrin = %s\n",\
12:         Number, Char, String);
13:}
14:
15:void ArgLst(char *format, ...){
16:   va_list Lst;
17:
18:   va_start(Lst, format);
19:   vprintf(format, Lst);
20:   va_end(Lst);
21:}
```

3.5.12.4 الدالة **vfprintf**:

هي مطابقة لدالة vprintf فقط هي خاصة بالتعامل مع الملفات، مثال:

```
1:#include<stdio.h>
2:#include<stdarg.h>
3:
4:void ArgLst(char *format, ...);
5:
6:main(int argc, char *argv[]){
7:    int Number = 100;
8:    char Char = 'A';
9:    char String[] = "Hello, World!";
10:
11:   ArgLst("Number = %d\nChar = %c\nStrin = %s\n",\
12:         Number, Char, String);
13:}
14:
15:void ArgLst(char *format, ...){
16:   va_list Lst;
17:   FILE *File;
18:
19:   File = fopen("Info.txt", "w");
20:
21:   va_start(Lst, format);
22:   vfprintf(File, format, Lst);
23:   va_end(Lst);
24:   fclose(File);
```

```
25:}
```

3.5.12.5 الدالة **vsprintf**:

هذه الدالة مدمجة مع كل من الدالة sprintf و الدالة vprintf، مثال:

```
1:#include<stdio.h>
2:#include<stdarg.h>
3:
4:void ArgLst(char *format, ...);
5:
6:main(int argc, char *argv[]){
7:    int Number = 100;
8:    char Char = 'A';
9:    char String[] = "Hello, World!";
10:   char buffer[255];
11:
12:   ArgLst(buffer, "Number = %d\nChar = %c\nStrin = %s\n",\
13:         Number, Char, String);
14:
15:   printf(buffer);
16:}
17:
18:void ArgLst(char *buffer, char *format, ...){
19:   va_list Lst;
20:
21:   va_start(Lst, format);
22:   vsprintf(buffer, format, Lst);
23:   va_end(Lst);
24:}
```

3.5.12.6 الدالة **scanf**:

تقوم هذه الدالة باستقبال الرموز من لوحة المفاتيح، و هي دالة خاصة بالإدخال، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    char string[255];
5:
6:    printf("Your name, Please: ");
7:    scanf("%s", string);
8:    printf("Nice to meet you %s!\n", string);
9:}
```

3.5.12.7 الدالة **fscanf**:

تقوم هذه الدالة بأخذ رموز من ملف نصي إما على شكل أرقام أو أحرف أو سلسلة نصوص، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:    char buffer[255];
6:
7:    File = fopen("Info.txt", "r");
8:
```

```
9:    fscanf(File, "%s", buffer);
10:
11:   printf("%s\n", buffer);
12:}
```

3.5.12.8 الدالة sscanf:

تقوم هذه الدالة بنشر سلسلة حروف على مجموعة من المتغيرات بشكل منفصل، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    char buffer[] = "Hello, World! 15";
5:    char string1[255], string2[255];
6:    int number;
7:
8:    sscanf(buffer, "%s%s%d", string1, string2, &number);
9:
10:
11:   printf("String = %s %s\n", string1, string2);
12:   printf("Number = %d\n", number);
13:}
```

3.5.12.9 الدالة fgetc:

تقوم هذه الدالة بأخذ أحرف من ملف نصي، حيث كلما يتم استعمال هذه الدالة يتقدم مؤشـــر الملـــف بخطوة، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:    char ch;
6:
7:    File = fopen("Info.txt", "r");
8:
9:    while(feof(File)==0){
10:         ch = fgetc(File);
11:         printf("%c", ch);
12:   }
13:}
```

3.5.12.10 الدالة fgets:

مثل الدالة fgetc فقط تأخذ سطر كامل بدل حرف واحد، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:    char buffer[255];
6:
7:    File = fopen("Info.txt", "r");
8:
9:    while(feof(File)==0){
10:         fgets(buffer, 255, File);
```

```
11:         printf(buffer);
12:   }
13:
14:   fclose(File);
15:}
```

3.5.12.11 الدالة **fputc**:

تقوم هذه الدالة بكتابة حرف إلى ملف نصي، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    File = fopen("Info.txt", "w");
7:
8:    fputc('A', File);
9:    fclose(File);
10:}
```

3.5.12.12 الدالة **fputs**:

تقوم هذه الدالة بكتابة سلسلة حرفية في ملف نصي، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    File = fopen("Info.txt", "w");
7:
8:    fputs("Hello, World!", File);
9:    fclose(File);
10:}
```

3.5.12.13 الدالة **getc**:

مثل الدالة fgetc و لكنها لا تحتوي على وسائط، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:    char  ch;
6:
7:    File = fopen("Info.txt", "r");
8:
9:    while(feof(File)==0){
10:         ch = getc(File);
11:         printf("%c", ch);
12:   }
13:
14:   fclose(File);
15:}
```

3.5.12.14 الدالة **getchar**:

تقوم هذه الدالة بالإدخال، حيث تستقبل حرف واحد فقط من المستخدم، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    char ch;
5:
6:    printf("Enter a character: ");
7:    ch = getchar();
8:    printf("Your character is: %c\n", ch);
9:}
```

3.5.12.15 الدالة **gets**:

تقوم هذه الدالة الإدخال أيضا، و هي تستقبل سلسلة حرفية من المستخدم، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    char str[255];
5:
6:    printf("Enter a string(Max character 255): ");
7:    gets(str);
8:    printf("Your string are: %s\n", str);
9:}
```

3.5.12.16 الدالة **putc**:

مثل الدالة `fputc`، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    File = fopen("Info.txt", "w");
7:
8:    putc('A', File);
9:    fclose(File);
10:}
```

3.5.12.17 الدالة **putchar**:

تقوم هذه الدالة بطباعة حرف على شاشة الحاسوب، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    char ch = 'A';
5:
6:    printf("ch = ");
7:    putchar(ch);
8:    putchar('\n');
```

```
9:}
```

3.5.12.18 الدالة puts:

تقوم هذه الدالة بطباعة سلسلة حرفية على شاشة الحاسوب،حيث يتم الرجوع إلى سطر جديد في كل سلسلة حرفية، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    char str1[] = "Line 1: Hello, World!";
5:    char str2[] = "Line 2: Hello, World!";
6:
7:    puts(str1);
8:    puts(str2);
9:}
```

3.5.12.19 الدالة ungetc:

تقوم هذه الدالة بحذف حرف من ملف نصي، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    File = fopen("Info.txt", "w");
7:
8:    ungetc('A', File);
9:    fclose(File);
10:}
```

3.5.12.20 الدالة fopen:

تستعمل هذه الدالة لقراءة و كتابة الملفات، حيث تحتوي على وسائط، الوسيط الأول لسلسلة حرفية التي هي اسم الملف الذي سيتم التعامل معه، و البارمتر الثاني هو النمط الذي سيتم استعماله مع الملف، حيث يكون ذلك النمط إما للكتابة أو القراءة أو كلهما على الملفات، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    File = fopen("FileName.Ext", "r");
7:
8:    if(File==NULL)
9:          printf("File does not exist!\n");
10:   else
11:         printf("File exist now!\n");
12:
13:   File = fopen("FileName.Ext", "w");
14:   if(File!=NULL)
15:         printf("File Created!\n");
```

```
16:
17:   if(File==NULL)
18:         printf("File does not exist!\n");
19:   else
20:         printf("File exist now!\n");
21:
22:   fclose(File);
23:}
```

3.5.12.21 الدالة freopen:

تستعمل هذه الدالة مثل الدالة fopen مع وسيط إضافي لمؤشر بينة FILE و الذي يكون اما stdin، stdout أو stderr، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    File = freopen("Info.txt", "w", stderr);
7:
8:    if(!File)
9:          fprintf(stdout, "Error!\n");
10:   else
11:         fprintf(File, "String!");
12:
13:   fclose(File);
14:}
```

3.5.12.22 الدالة fclose:

تقوم هذه الدالة بغلق ملف حيث يمكن استعماله مرة أخرى، و توجد الدالة _fcloseall و هي تقوم بغلق جميع الملفات الغير مغلق و ترجع قيمة لعدد الملفات التي تم إغلاقها، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File1, *File2, *File3;
5:
6:    File1 = fopen("Info1.txt", "w");
7:    File2 = fopen("Info2.txt", "w");
8:    File3 = fopen("Info3.txt", "w");
9:
10:   fclose(File1);
11:
12:   printf("%d file(s) closed by _fcloseall()\n",\
13:         _fcloseall());
14:}
```

3.5.12.23 الدالة remove:

تقوم هذه الدالة بحذف الملفات، مثال:

```
1:#include<stdio.h>
2:
```

```
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    /*Create temporary.tmp*/
7:    File = fopen("temporary.tmp", "w");
8:    fclose(File);
9:
10:   /*remove temporary.tmp*/
11:   remove("temporary.tmp");
12:}
```

3.5.12.24 الدالة **rename**:

تقوم هذه الدالة بإعادة تسمية ملف، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    printf("Rename Info.txt To Data.dat...\n");
5:    rename("Info.txt", "Data.dat");
6:    printf("Operation Terminate...\n");
7:}
```

3.5.12.25 الدالة **tmpfile**:

تقوم هذه الدالة بإنشاء ملف مؤقت *temporary file* للبرنامج، حيث يتم حذف الملف المؤقت عند الخروج من البرنامج أو نهاية البرنامج، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    printf("Create a Temporary File...\n");
7:    File = tmpfile();
8:    printf("Temporary File Created...\n");
9:
10:   /*At exit*/
11:   printf("Temporary File deleted...\n");
12:}
```

3.5.12.26 الدالة **fread**:

تقوم هذه الدالة بقراءة محتوى ملف و وضع نسخة لها في سلسلة حرفية، و تحتوي على أربعة وسائط، الوسيط الأول هو لسلسلة الحرفية التي سيتم وضع فيها نسخة من نص الملف، و الوسيط الثاني هو حجم المحتوى الواحد بالبايتات، و الوسيط الثالث هو عدد الأحرف التي نريد نسخها، و الوسيط الأخير و مؤشر للبنية FILE، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
```

```
5:    char buffer[255];
6:
7:    File = fopen("Info.txt", "r");
8:
9:    fread(buffer, sizeof(char), 255, File);
10:   printf("%s\n", buffer);
11:}
```

3.5.12.27 الدالة **fwrite**:

تقوم هذه الدالة بالكتابة على الملفات، و هي الدالة المعاكسة لدالة fread، و لها نفس وسائط الدالة fread، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:    char buffer[] = "String!";
6:
7:    File = fopen("Info.txt", "w");
8:
9:    fwrite(buffer, sizeof(char), 7, File);
10:   fclose(File);
11:}
```

3.5.12.28 الدالة **fseek**:

تقوم هذه الدالة بالتحكم في مكان مؤشر لملف نصي، حيث تحتوي على ثلاثة وسائط، الوسيط الأول هو مؤشر للبنية FILE و هو الملف الذي سيتم تحديد مكان مؤشره، و الوسيط الثاني هو متغير من نوع long و هو عدد البايتات من الوسيط الثالث، مثلا إذا أعطيناه القيمة 5 فسيتم تجاهل خمسة بايتات بعد المؤشر الذي تم تحديده في الوسيط الثالث، و أخيرا الوسيط الثالث، و هو متغير من نوع int، و من هذا الوسيط نقوم بتحديد المؤشر لنص، و له ثوابت و هي SEEK_CUR و SEEK_SET و SEEK_END، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:    char ch_set, ch_cur, ch_end;
6:
7:    File = fopen("Info.txt", "r");
8:
9:    /*Beginning of file*/
10:   fseek(File, 0L, SEEK_SET);
11:
12:   printf("SEEK_SET Begin:\n");
13:   while(feof(File)==0){
14:         ch_set = fgetc(File);
15:         printf("%c", ch_set);
16:   }
17:   printf("\nSEEK_SET End.\n");
18:
19:   /*Current position of file pointer*/
```

```
20:   fseek(File, 0L, SEEK_CUR);
21:
22:   printf("SEEK_CUR Begin:\n");
23:   while(feof(File)==0){
24:         ch_cur = fgetc(File);
25:         printf("%c", ch_cur);
26:   }
27:   printf("\nSEEK_CUR End.\n");
28:
29:   /*End of file*/
30:   fseek(File, 0L, SEEK_END);
31:
32:   printf("SEEK_END Begin:\n");
33:   while(feof(File)==0){
34:         ch_end = fgetc(File);
35:         printf("%c", ch_end);
36:   }
37:   printf("\nSEEK_END End.\n");
38:}
```

3.5.12.29 الدالة **ftell**:

تقوم هذه الدالة بإرجاع قيمة لمكان مؤشر النص الحالي، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:    int Position;
6:    char buff[3];
7:
8:    File = fopen("Info.txt", "r");
9:
10:   /* Move the pointer of file 3 bytes
11:   by reading character*/
12:   fread(buff, sizeof(char), 3, File);
13:
14:   Position = ftell(File);
15:   printf("Position after read 3 characters is %d\n",\
16:         Position);
17:}
```

3.5.12.30 الدالة **rewind**:

تقوم هذه الدالة بإرجاع مؤشر ملف إلى البداية، و هي مكافئة لـــ `fseek(File, 0L, SEEK_SET)`، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:    char ch;
6:
7:    File = fopen("Info.txt", "r");
8:
9:    fseek(File, 0L, SEEK_END);
10:   printf("fseek(File, 0L, SEEK_END):\n");
11:   while(feof(File)==0){
12:         ch = fgetc(File);
13:         printf("%c", ch);
```

```
14:   }
15:
16:   printf("\n----------------\n");
17:
18:   rewind(File);
19:   printf("rewind(File):\n");
20:   while(feof(File)==0){
21:         ch = fgetc(File);
22:         printf("%c", ch);
23:   }
24:
25:   printf("\n");
26:}
```

3.5.12.31 الدالة **feof**:

تقوم هذه الدالة بإختبار إذا كان مؤشر ملف قد وصل إلى نهاية الملف، حيث ترجع هذه الدالة القيمة *0* إذا مؤشر الملف في النهاية، و ترجع قيمة غير الصفر إن لم يكن المؤشر في نهاية الملف، مثال:

```
1:#include<stdio.h>
2:
3:main(int argc, char *argv[]){
4:    FILE *File;
5:
6:    File = fopen("Info.txt", "r");
7:
8:    fseek(File, 0L, SEEK_END);
9:    if(feof(File)==0)
10:         printf("Position of pointer is in the end of file\n");
11:   else
12:         printf("Position of pointer is not in the end of file\n");
13:}
```

3.5.13 الملف الرأسي **stdlib.h**:

الاسم *stdlib* مختصر من *Standard Library*، حيث يحتوي هذا الملف الرأسي على مجموعة من الـــدوال في كل من دوال تحويل الأرقام، دوال تخصيص الذاكرة(التخزين) و دوال أخرى.

3.5.13.1 الدالة **atof**:

تقوم هذه الدالة بتحويل أعداد موجود في سلسلة حرفية إلى أعداد حقيقية من نوع double، حيث يمكـــن التعامل مع تلك الأرقام بسهولة، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    char str[3] = "123";
6:    double num;
7:
8:    num = atof(str);
9:
```

```
10:   printf("%f\n", num);
11:}
```

3.5.13.2 الدالة **atoi**:

تقوم هذه الدالة بنفس ما تقوم به الدالة atof إلا أنها تحول أعداد السلسلة الحرفية إلى أعداد صحيحة من نوع int، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    char str[3] = "123";
6:    int num;
7:
8:    num = atoi(str);
9:
10:   printf("%d\n", num);
11:}
```

3.5.13.3 الدالة **atol**:

تقوم هذه الدالة بنفس ما تقوم به الدالة atoi إلا أنها تحول أعداد السلسلة الحرفية إلى أعداد صحيحة من نوع long، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    char str[3] = "123";
6:    long num;
7:
8:    num = atoi(str);
9:
10:   printf("%ld\n", num);
11:}
```

3.5.13.4 الدالة **rand**:

تقوم هذه الدالة بتوليد أرقام عشوائية، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    int i, j=0;
6:    for(i=0;i<=10;i++){
7:          j=rand()%100;
8:          printf("j = %d\n", j);
9:    }
10:}
```

و هنا لن تتعد الأعداد العشوائية العدد *100*، و يمكننا أن نجعلها أكثر أو أقل من ذلك فقط نقوم بالتغير في العدد *100* الذي موجود بعد رامز باقي القسمة.

3.5.13.5 الدالة **srand**:

تقوم هذه الدالة بتشغيل مولد الأرقام العشوائية rand، حيث يجعل تلك القيم العشوائية متغير من لحظة لأخرى عندما نمرر له الدالة time، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:#include<time.h>
4:
5:main(int argc, char *argv[]){
6:    int i;
7:
8:    srand(time(NULL));
9:
10:   for(i=0;i<=10;i++){
11:         printf("%d\n", rand()%100);
12:   }
13:}
```

3.5.13.6 الدالة **abort**:

تستعمل هذه الدالة في حالة وجود أخطاء في البرنامج، حيث تقوم بإقاف البرنامج و إضهار رسالة تخبر المستخدم أن طريقة إنتهاء البرنامج غير طبيعية، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    FILE *File;
6:
7:    if(!(File=fopen("FileName.txt", "r"))){
8:          printf("Can't open file\n");
9:          abort();
10:   }
11:
12:   fclose(File);
13:}
```

3.5.13.7 الدالة **exit**:

تقوم هذه الدالة بالإنهاء البرنامج إذا تم التمرير إليها القيمة *1*، حيث يوجد ثوابت باسم EXIT_FAILURE و هو نفسه القيمة *1*، و الثابت EXIT_SUCCESS و هو القيمة *0* و الذي يعني الخروج السليم للبرنامج، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
```

```
4:main(int argc, char *argv[]){
5:    FILE *File;
6:
7:    if(!(File=fopen("FileName.txt", "r"))){
8:          printf("Can't open file\n");
9:          exit(EXIT_FAILURE);
10:   }else
11:         exit(EXIT_SUCCESS);
12:
13:   fclose(File);
14:}
```

3.5.13.8 الدالة **atexit**:

تقوم هذه الدالة باستعداء دوال أخرى، حيث يتم تنفيذ تلك الدوال عند نهاية البرنامج، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:void AtExit(){
5:    printf("Program - end\n");
6:}
7:
8:main(int argc, char *argv[]){
9:    atexit(AtExit);
10:
11:   printf("Program - start\n");
12:}
```

3.5.13.9 الدالة **system**:

تقوم هذه الدالة بتنفيذ أوامر النظام مثلا إذا أردنا مسح شاشة الجهاز باستخدام أوامر النظام نكتب:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:
6:    printf("Message!\n");
7:
8:    /*In DOS OS*/
9:    system("cls");
10:
11:   /*In Unix/Linux OS*/
12:   system("clear");
13:}
```

3.5.13.10 الدالة **abs**:

تقوم هذه الدالة بإرجاع القيمة المطلقة من النوع int للعدد الذي تم تمريره إليها، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    int i=-10, j=20;
6:
```

```
7:    printf("absolute value if 'i' is %d\n", abs(i));
8:    printf("absolute value if 'j' is %d\n", abs(j));
9:}
```

3.5.13.11 الدالة **labs**:

نفس الدالة abs إلا أنها ترجع القيمة المطلق من النوع long، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    long i=-10, j=20;
6:
7:    printf("absolute value if 'i' is %d\n", labs(i));
8:    printf("absolute value if 'j' is %d\n", labs(j));
9:}
```

3.5.13.12 الدالة **div**:

تقوم هذه الدالة بتقسيم قيمة الوسيط الأول على قيمة الوسيط الثاني، و الناتج يكون من نوع int، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    int a=12, b=6;
6:
7:    printf("%d/%d = %d\n", a, b, div(a, b));
8:}
```

3.5.13.13 الدالة **ldiv**:

تقوم هذه الدالة بتقسيم قيمة الوسيط الأول على قيمة الوسيط الثاني، و الناتج يكون من نوع long، مثال:

```
1:#include<stdio.h>
2:#include<stdlib.h>
3:
4:main(int argc, char *argv[]){
5:    long a=12, b=6;
6:
7:    printf("%d/%d = %d\n", a, b, ldiv(a, b));
8:}
```

3.5.14 الملف الرأسي **string.h**:

في الملف الرأسي string.h توجد مجموعتين من الدوال، المجموعة الأولى تبدأ أسماءها بـــ str و هي تعـــني *string*، و المجموعة الثانية تبدأ أسماءها بـــ mem و هي تعني *memory*.

سندرس أهم دوال هذا الملف الرأسي، و التي أغلبها تبدأ بـــ str.

3.5.14.1 الدالة strcpy و الدالة strncpy:

تقوم كلا من الدالتين بنسخ نص إلى سلسلة حرفية، الدالة الأولى strcpy بها وسيطين، الوسيط الأول هو سلسلة الحروف التي سيتم نسخ فيها النص، و الوسيط الثاني هو النص الذي سيتم نسخه. و الدالة الثانية strncpy بها ثلاثة وسائط، الوسائط الأول و الوسائط الثاني هما نفس الوسائط الأول و الثاني لدالة strcpy، أما الوسيط الثالث فهو عدد الأحرف التي نريد نسخها، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str[] = "Hello";
6:    char empty[5];
7:    char empty2[5];
8:
9:    strcpy(empty, str);
10:   strncpy(empty2, str, 3);
11:
12:   printf("empty  = %s\n", empty);
13:   printf("empty2 = %s\n", empty2);
14:}
```

أما إذا كانت السلسلة empty بها نص سابقا، فسيتم حذفه، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str[] = "Hello";
6:    char empty[5] = "empty";
7:
8:    strcpy(empty, str);
9:
10:   printf("empty  = %s\n", empty);
11:}
```

3.5.14.2 الدالة strcat و الدالة strncat:

تقوم الدالة strcat بنسخ نص و إضافته إلى سلسلة حرفية، و أيضا الدالة strncat تقوم بنسخ نص و إضافته إلى سلسلة حرفية مع تحديد عدد الأحرف التي نريد إضافتها، مثال لطريقة استعمال الدالة strcat:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str[] = "Hello";
6:
7:    printf("str = %s\n", str);
8:    strcat(str, ", World");
9:    printf("str = %s\n", str);
10:}
```

و هذا مثال لطريقة استعمال الدالة strncat:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str[] = "Hello";
6:
7:    printf("str = %s\n", str);
8:    strncat(str, ", World", 3);
9:    printf("str = %s\n", str);
10:}
```

3.5.14.3 الدالة **strcmp** و الدالة **strncmp**:

تقوم الدالة strcmp بالمقارنة بين سلسلتين، إذا كانت السلسلة الحرفية الأولى هي الأكبر فستكون النتيجة أكبر من *0*، و في حالة أن السلسلة الحرفية الثانية هي الأكبر فستكون النتيجة أصغر من *0*، و أيضا الدالة strncmp تقوم بالمقارنية بين سلسلتين مع تحديد عدد حروف السلسلة الحرفية الأولى التي نريد مقارنتها مع السلسلة الحرفية الثانية، مثال حول طريقة استعمال الدالة strcmp:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str1[] = "Hello";
6:    char str2[] = "Hello2";
7:    int cmp = strcmp(str1, str2);
8:
9:    if(cmp>0)
10:         printf("str1 > str2\n");
11:   else
12:         printf("str1 < str2\n");
13:}
```

و هذا مثال لطريقة استعمال الدالة strncmp:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str1[] = "Hello";
6:    char str2[] = "Hello2";
7:    int cmp = strncmp(str1, str2, 3);
8:
9:    if(cmp>0)
10:         printf("str1 > str2\n");
11:   else
12:         printf("str1 < str2\n");
13:}
```

3.5.14.4 الدالة **strchr** و الدالة **strrchr**:

تستعمل الدالة strchr للبحث عن مكان حرف في سلسلة حرفية، و إذا كان الحرف الذي نريد البحث عنه في السلسلة الحرفية موجود مرتين فسيتم أخذ مكان أول حرف، و الدالة strrchr مثل الدالة strchr و لكنها تأخذ الحرف الأخير بدل الأول، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str[] = "Hello", *strdest;
6:    char ch = 'l';
7:    int result;
8:
9:    printf("%s\n", str);
10:   printf("12345\n");
11:
12:   strdest = strchr(str, ch);
13:   result = (int)(strdest-str+1);
14:   printf("First '%c' in position %d\n", ch, result);
15:
16:   strdest = strrchr(str, ch);
17:   result = (int)(strdest-str+1);
18:   printf("Last '%c' in position %d\n", ch, result);
19:}
```

3.5.14.5 الدالة **strspn** و الدالة **strcspn**:

باستخدام الدالة strspn يمكن معرفة عدد مجموعة من الأحرف في سلسلة حرفية، حيث يجب أن تكون تلك الأحرف هي بداية السلسلة الحرفية و إلا ستكون النتيجة *0*، أما الدالة strcspn فهي تقوم بتحديد مكان مجموعة من الحروف في سلسلة حرفية، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str1[] = "HHHeeelllooo";
6:    char str2[] = "He";
7:    char str3[] = "lllooo";
8:    int result;
9:
10:   result = strspn(str1, str2);
11:   printf("There are %d character(s) of '%s' in string '%s'\n",\
12:         result, str2, str1);
13:
14:   result = strcspn(str1, str3);
15:   printf("First '%s' in string '%s' is start at character %d\n",\
16:         str3, str1, result);
17:}
```

و لكن مرونة الدالتين ليست كبيرة، لذا يفضل أن يتم كتابتهما من جديد على حسب رغبات المبرمج.

3.5.14.6 الدالة **strpbrk**:

تقوم الدالة strpbrk بنسخ سلسلة حرفية و لصقها في سلسلة حرفية أخرى، حيث يجب أن نقوم بتحديد بداية النسخة باستخدام أحرف، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str[] = "Hello";
6:    char *result = strpbrk(str, "l");
7:
8:    printf("%s\n", result);
9:}
```

3.5.14.7 الدالة strstr:

الدالة strstr مطابقة لدالة strchr في إختلاف بسيط و هي أنها تبحث عن مكان نص بدل حرف واحد، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str1[] = "Hello, World!", str2[] = "World";
6:    char *strdest = strstr(str1, str2);
7:    int result;
8:
9:    result = (int)(strdest-str1+1);
10:   printf("The word '%s' is at position %d in string '%s'\n",\
11:         str2, result, str1);
12:
13:}
```

3.5.14.8 الدالة strlen:

تقوم بحسب عدد أحرف سلسلة حرفية، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char str[] = "Hello";
6:    int result = strlen(str);
7:
8:    printf("'%s' = %d character(s)\n", str, result);
9:}
```

3.5.14.9 الدالة strerror:

تحمل هذه الدالة مجموعة من رسائل الأخطاء الخاص بالسلاسل الحرفية، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:#include<stdlib.h>
4:
```

```
5:main(int argc, char *argv[]){
6:    char str[10];
7:
8:    printf("Enter a string (Max 10 characters): ");
9:
10:   if((strlen(gets(str)))>10){
11:         printf("%s\n", strerror(12));
12:         exit(1);
13:   }else
14:         printf("'%s' = %d character(s)\n",\
15:         str, strlen(str));
16:
17:}
```

و يمكن رأية باقي الرسائل باستخدام التكرار.

3.5.14.10 الدالة strtok:

باستعمال الدالة strtok نحدد مجموعة من الأحرف أو الرموز لسلسلة حرفية حيث لا يتم طباعتها، مثال:

```
1:#include<stdio.h>
2:#include<string.h>
3:
4:main(int argc, char *argv[]){
5:    char *string,
6:         str[] = "(Hello, World)",
7:         tok[] = " (),I";
8:    int i;
9:
10:   string = strtok(str, tok);
11:
12:   for(i=0;i<2;i++){
13:         printf("%s\n", string);
14:         string = strtok(NULL, tok);
15:   }
16:}
```

3.5.15 الملف الرأسي time.h:

يحتوي الملف الرأسي time.h على مجموعة من الدوال الخاصة بمعالجة التاريخ و الوقت، و يحتوي هذا الملف الرأسي على بنية باسم tm و هي معرفة بالشكل التالي:

```
1:struct    tm    {
2:    int   tm_sec;
3:    int   tm_min;
4:    int   tm_hour;
5:    int   tm_mday;
6:    int   tm_mon;
7:    int   tm_year;
8:    int   tm_wday;
9:    int   tm_yday;
10:   int   tm_isdst;
11:};
```

و يحتوي أيضا على متغيرين باسم clock_t و time_t و هما معرفان على الشكل التالي:

```
1:typedef long time_t;
2:typedef long clock_t;
```

سنتحدث عن أهم دوال هذا الملف الرأسي، أما الباقي الدوال يمكن رأيتها في الملف الرأسي time.h.

3.5.15.1 الدالة clock:

تقوم هذه الدالة بالعد، حيث يبدأ عدها عند بداية البرنامج، حيث تبدأ من الصفر، و قسمة قيمة هذه الدالة على الثابت CLOCKS_PER_SEC أو CLK_PER_SEC يرجع الوقت بالثواني، مثال:

```
1:#include<stdio.h>
2:#include<time.h>
3:
4:main(int argc, char *argv[]){
5:    for(;;){
6:          printf("%d\n", clock()/CLOCKS_PER_SEC);
7:          if((clock()/CLOCKS_PER_SEC)==5)
8:                break;
9:    }
10:}
```

و يمكن أن نجعل العد بالدقائق أو الساعات أو الأيام، فقط نقوم بقسم الطريقة السابقة على 60 في حالة أردنا العد بالدقائق، و نزيد القسمة على 60 إذا أردنا العد يكون بالساعات، و نزيد أيضا القسمة على 24 إذا أردنا أن يكون العد بالأيام. و باستخدام هذه الدالة يمكن عمل دالة تأخر تساعدنا كثيرا في برامجنا، الدالة ستكون كتالي:

```
1:#include<stdio.h>
2:#include<time.h>
3:
4:void delay(int second);
5:
6:main(int argc, char *argv[]){
7:    int waitsec;
8:
9:    printf("wait(in seconds): ");
10:   scanf("%d", &waitsec);
11:   delay(waitsec);
12:   printf("Time terminated...\n");
13:}
14:
15:void delay(int second){
16:   int sec;
17:
18:   for(;;){
19:         sec = clock()/CLOCKS_PER_SEC;
20:         if(sec==second)
21:               break;
22:   }
23:}
```

3.5.15.2 الدالة time:

تقوم هذه الدالة بإرجاع عدد الثواني التي مرت من الساعة *00:00* في اليوم *1* جانفي *1970*، مثال:

```
1:#include<stdio.h>
2:#include<time.h>
3:
4:main(int argc, char *argv[]){
5:    time_t Seconds, Minutes, Hours, Days;
6:
7:    Seconds = time(NULL);
8:    Minutes = Seconds/60;
9:    Hours   = Minutes/60;
10:   Days    = Hours/24;
11:
12:   printf("%ld\tseconds since 01/01/1970.\n", Seconds);
13:   printf("%ld\tminutes since 01/01/1970.\n", Minutes);
14:   printf("%ld\t\thours since 01/01/1970.\n", Hours);
15:   printf("%ld\t\tdays since 01/01/1970.\n", Days);
16:}
```

هنا ستقوم الدالة time بإرجاعة قيمة للمتغير Seconds لأن الوسيط الذي تم تمريره إليه فارغ NULL، و يمكن كتابة المثال السابقة بالطريقة التالية:

```
1:#include<stdio.h>
2:#include<time.h>
3:
4:main(int argc, char *argv[]){
5:    time_t Seconds, Minutes, Hours, Days;
6:
7:    time(&Seconds);
8:    Minutes = Seconds/60;
9:    Hours   = Minutes/60;
10:   Days    = Hours/24;
11:
12:   printf("%ld\tseconds since 01/01/1970.\n", Seconds);
13:   printf("%ld\tminutes since 01/01/1970.\n", Minutes);
14:   printf("%ld\t\thours since 01/01/1970.\n", Hours);
15:   printf("%ld\t\tdays since 01/01/1970.\n", Days);
16:}
```

3.5.15.3 الدالة difftime:

لهذه الدالة وسيطين، كلاهما لمتغيرات من نوع time_t، و الدالة تقوم بإرجاع الفرق بين الوقت الأول الموجود في الوسيط الأول و بين الوقت الثاني و الموجود في الوسيط الثاني بالثواني،أي تقوم بطرح قيمة الوسيط الأول على الوسيط الثاني، مثال:

```
1:#include<stdio.h>
2:#include<time.h>
3:
4:main(int argc, char *argv[]){
5:    time_t Start, End;
6:    int i;
```

```
7:
8:    Start = time(NULL);
9:    for(i=0;i<=40000;i++){
10:         printf("%d\n", i);
11:   }
12:   End = time(NULL);
13:
14:   printf("Loop taken %.0f seconds to terminate...\n",\
15:         difftime(End, Start));
16:}
```

3.5.15.4 الدالة localtime:

تقوم هذه الدالة بتحويل عدد الثواني من عام *1900* إلى التوقيت محلي ثم تمريرها إلى أعضاء البنية tm، مثال:

```
1:#include<stdio.h>
2:#include<time.h>
3:
4:main(int argc, char *argv[]){
5:    time_t Seconds;
6:    int Year, Month, Day, Hour, Minute, Second;
7:    struct tm* Time;
8:
9:    time(&Seconds);
10:
11:   Time = localtime(&Seconds);
12:   Year = Time->tm_year+1900, Month= Time->tm_mon+1,
13:         Day = Time->tm_mday;
14:
15:   Hour = Time->tm_hour, Minute = Time->tm_min,
16:         Second = Time->tm_sec;
17:
18:   printf("Date: %.4d/%.2d/%.2d\n",Year, Month, Day);
19:   printf("Time: %.2d:%.2d:%.2d\n", Hour, Minute, Second);
20:}
```

3.5.15.5 الدالة asctime:

تقوم هذه الدالة بتحويل بيانات البنية التي تم تمريرها إليها إلى سلسلة حروف من الـــشكل *DDD MMM D HH :MM :SS YYYY*، مثال:

```
1:#include<stdio.h>
2:#include<time.h>
3:
4:main(int argc, char *argv[]){
5:    time_t Seconds;
6:    struct tm* Time;
7:
8:    time(&Seconds);
9:
10:   Time = localtime(&Seconds);
11:
12:   printf("Date/Time: %s", asctime(Time));
13:}
```

3.5.15.6 الدالة ctime:

تقوم هذه الدالة بتحويل عدد الثواني الدالة time() إلى سلسلة حـــروف مـــن الـــشكل *DDD MMM D HH :MM :SS YYYY*، مثال:

```
1:#include<stdio.h>
2:#include<time.h>
3:
4:main(int argc, char *argv[]){
5:    time_t Seconds;
6:
7:    time(&Seconds);
8:
9:    printf("Date/Time: %s", ctime(&Seconds));
10:}
```

الخاتمة، مواقع و مشاريع

كل الطرق تأدي إلى روما، و لنفرض أن روما هي هدفنا في البرمجة (أي البرنامج الذي سنقوم ببرمجتـــه)، و نضع الطرق التي تأدي إلى روما هي اللغات البرمجة (مثلا Java، Delphi،...). سنجد أن الفرق بين طريق و أخرى ليس بكبير، فمثلا ربما نذهب على طريق نجده سهل، أو طريق نجده صعب، أو طريق ســـريع أو بطيئ، أو...، و هذه هو الفرق بين لغة برمجة و أخرى، مثال:

الفرق بين لغة C++ و لغة Assembly (التجميع) هو:

- سهولة في فهم لغة C++، لأنها لغة عالية المستوى، و من ميزات اللغات العالية المستوى هي جلعها لغة تحاكي لغة الإنسان، أما لغة التجميع هي لغة منخفظة المستوى، و اللغات المنخفظة المستوى هي لغات تحاكي لغة الحاسوب مما يجعلها لغة صعبة.

- سرعة برنامج مكتوب بلغة التجميع أكبر من سرعة برنامج مكتوب بلغة C++.

- ستجد حجم برنامج مكتوب بلغة التجميع أصغير بكثير من حجم برنامج مكتوب بلغة C++.

لذا نجد أن أغلب فيروسات الحاسوب مبرمجة بلغات منخفظة المستوى، و بالخاص لغة التجميع.

و الآن سنتحدث عن أهم المواقع العربية و أهم المشاريع العربية التي تحتاج إلى الدعم.

موقع فيجوال سي للعرب، و هو موقع خاص بكل ما يتعلق بلغة C و عائلتها (C++, C#) في جميع مجالاتها (برمجة الأنظمة، برمجة الألعاب، الذكاء الإصطناعي و الشبكات العصبية الصناعية، أمن نظم المعلومات، بناء المترجمات، ...)، رابط الموقع: www.vc4arab.com.

موقع الفريق العربي للبرمجة، موقع به كل ما يخص بالبرمجة في جميع اللغات، مقالات، دروس، دورات، ...، رابط الموقع: www.arabteam2000.com.

موقع الكتب العربية، أكبر مكتبة عربية تحتوي على مجموعات كبيرة من الكتب الإلكتروني العربية المجانية في كل ما يتعلق بعلوم الحاسوب سواء برمجة، تصميم، صيانة أو ...، رابط الموقع: www.cb4a.com.

موقع المختار للبرمجة، موقع أحد أصدقائي الأعزاء، و هو موقع به دروس حول أغلب لغـــات البرمجـــة، و بالخاص لغة *BASIC*، رابط الموقع: *www.2jm2.com*.

موقع عروة نت، أيضا موقع لأحد أصدقائي، و هو أيضا موقع به دروس حول أغلب لغـــات البرمجـــة، و بالخاص لغة *Delphi*، رابط الموقع: *www.orwah.net*.

موقع فريق تطوير الألعاب، أيضا موقع لأحد أصدقائي، و هو موقع خاص بكل ما يتعلق بتصميم و برمجـــة الألعاب، رابط الموقع: *www.gamesdevteam.com*.

و من المشاريع العربية، نظام *Linux* العربي *Arabian*، و لا يزال يحتاج إلى مطويرين، لمزيد من المعلومـــات: */http://www.arabian.arabicos.com*

و مشروع نظام المنتـــديات العـــربي mysmartbb، و يمكـــن الوصـــول إليـــه مـــن الـــرابط التـــالي: */http://www.mysmartbb.com*

تم بحمد الله