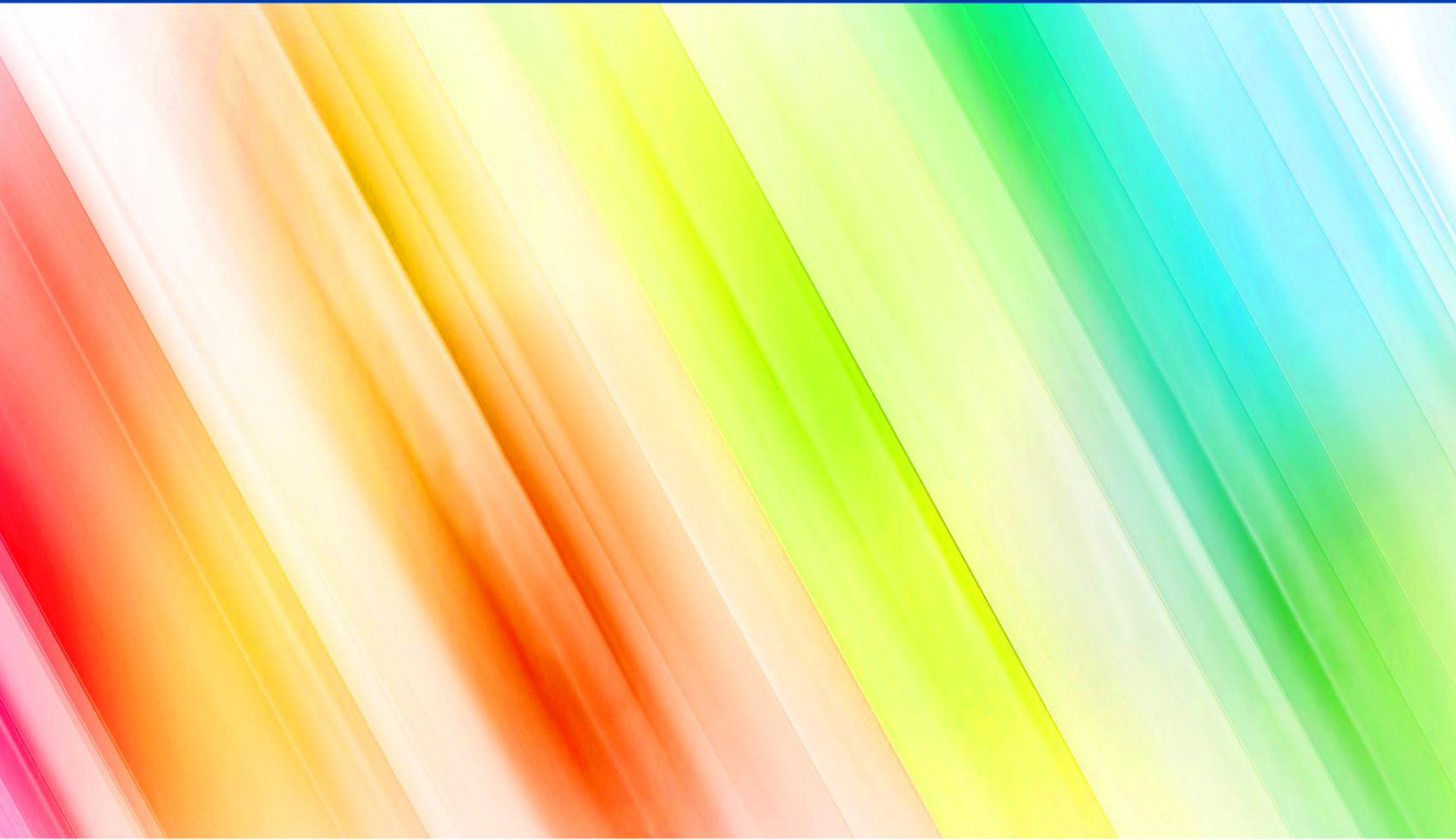


C

Languge Made Easy


لغة السي ببساطة


تعلم قواعد اللعبة ثم العب افضل من الباقيين
البرت اينشتاين




1st

Mohamed Hussein

 m.hussein1389@gmail.com

 [@mo7amed-hussein](#)

 [@m_hussein](#)

رابط الاكواد الخاصة بالكتاب في حالة عدم وجودها مرفقة مع الكتاب

<https://github.com/mo7amed-hussein/c-language-made-easy>

Table of Contents

مقدمة.....	- 7 -
الادوات المستخدمة	- 8 -
Hello World ما قبل	- 8 -
main function :الدالة الاساسية	- 9 -
Hello World الطباعة علي الشاشة	- 9 -
- طباعة علي اكثر من سطر	- 12 -
-escape sequences.....	- 13 -
Algorithmsالخوارزميات.....	- 13 -
الخلاصة	- 15 -
- سؤال وجواب	- 16 -
المتغيرات.....	- 19 -
Memoryالذاكرة.....	- 19 -
Storage unitsوحدات التخزين.....	- 20 -
Variablesالمتغيرات	- 20 -
Integers variableمتغيرات الاعداد الصحيحة.....	- 21 -
floating-point variablesمتغيرات الاعداد الحقيقية	- 24 -
characters متغير الحروف	- 25 -
constantsالثوابت	- 28 -
arithmetic operators - العمليات الحسابية	- 29 -
- سؤال وجواب	- 30 -
operatorsالرموز	- 33 -
arithmetic operators: رموز العمليات الحسابية	- 33 -
relational operators:رموز العمليات العلاقية.....	- 33 -
logical operatorsرموز العمليات المنطقية	- 34 -
increment and decrement operatorsالمؤثرات الزيادة والنقصان	- 36 -
bitwise operatorsرموز خاصة بالبتات.....	- 37 -
assignment operatorsمؤثرات تعيين القيم.....	- 38 -
cast operator:مؤثر التحويل من نوع لآخر	- 39 -
expressionsالعبارات الجبرية.....	- 39 -
expression الاسبقية في حساب قيمة الـ	- 40 -
استخدامات المؤثرات الخاصة بالبتات	- 40 -
الخلاصة	- 45 -
سؤال وجواب	- 45 -

التحكم في سير البرنامج	- 48 -
statement block	- 48 -
program control التحكم في سير البرنامج	- 48 -
if - جملة الشرط	- 48 -
switch جملة الشرط	- 58 -
break & continue - تعديل الحلقات باستخدام	- 69 -
الخلاصة	- 73 -
سؤال وجواب	- 74 -
امثلة عملية	- 77 -
سؤال وجواب	- 105 -
pointers المؤشرات	- 107 -
الخلاصة	- 116 -
functions الدوال -	- 118 -
variable scope - مجال المتغيرات	- 124 -
الخلاصة	- 136 -
التعامل مع الملفات	- 150 -
انواع الملفات -	- 162 -
الخلاصة	- 168 -
Preprocessor التوجيهات	- 170 -
الخلاصة	- 179 -
الدوال والمؤشرات المتقدمة	- 181 -
self-referential structure - البنية ذاتية المرجعية	- 194 -
الخلاصة	- 194 -
امثلة عملية	- 196 -
المراجع	- 200 -

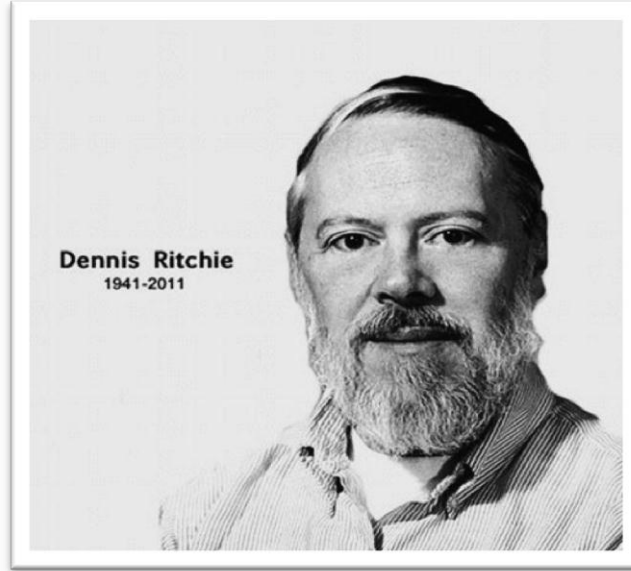


Introduction

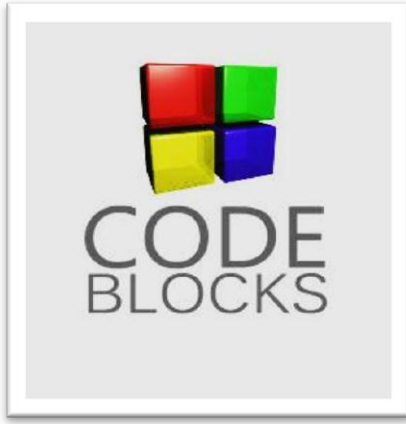
مقدمة

لغة السي هي لغة متعددة الاغراض general-purpose, اذ ليست موجهة لنوع معين من التطبيقات , وذلك علي الرغم من ارتباطها بنظام اليونكس UNIX حيث تم تطوير النظام ومعظم البرامج الخاصة به بلغة السي , وتسي ايضا بلغة برمجة الانظمة لانها تستخدم في تطوير المترجمات وانظمة التشغيل وغيرها , وقديما قيل " ان لغة السي يتعلمها من لايعرف ماذا يريد ان يفعل " , ان تطور لغة السي جاء من لغتين سابقتين لها هما BCPL و B و BCPL تم تطويرها سنة 1967 بواسطة مارتن ريتشارد Martin Richards , في 1970 قام كين تومسون Ken Thompson بتطوير لغة B واستخدمها لكتابة النسخة الاولى من نظام يونكس في Bell Laboratories, كلتا اللغتين لم يكن بهما انواع للبيانات ولذا سميت "typeless" حيث يشغل كل عنصر في البيانات 2 بايت في الذاكرة .

تم تطوير السي بعد لغة B بواسطة دينسي ريتشي Dennis Ritchie حيث تم اضافة انواع للبيانات وسمات اخري , اصبحت اللغة شائعة الاستخدام وادي ذلك الي تطوير مكتبات ودوال لنسخ مختلفة من لغة السي وتلك النسخ من اللغة لم تكن متوافقة مع بعضها لذا تم تعريف النسخة القياسية من اللغة ANSI c سنة 1989 بواسطة المعهد الامريكي للمعايير American National Standards Institute , في عام 1999 تم تحديث النسخة القياسية حيث تم تحسين وتوسيع بعض الخصائص وتسي تلك النسخة C99 , بعد ذلك تم اطلاق اللغة في جميع انحاء العالم بالتعاون بين المعهد الامريكي للمعايير والمنظمة العالمية للمعايير وسميت بـ ISO c , من اهم مراجع لغة السي كتاب " The C Programming Language " للمؤلفان دينس ريتشي وبريان كرنيجان Brian W. Kernighan and Dennis M. Ritchie



دينسي ريتشي



الاداة المستخدمة في هذا الكتاب هي Code Blocks وهذا النوع من الادوات يسمى IDE بيئة التطوير المتكاملة حيث تحتوي علي محرر اكواد ومترجم وادوات اخري ويمكن الحصول علي هذه الاداة من الموقع الخاص بها <http://www.codeblocks.org> ويمكن استخدام ادوات اخري مثل eclipse او Pellse c او visual studio او غيرها.

ما قبل Hello World

اغلب كتب تعليم لغات البرمجة تبدأ بـ Hello World والذي يقوم بطباعة كلمة Hello World علي الشاشة ولكننا سنبدأ قبل ذلك بخطوة empty world , كالتالي :

- قم بإنشاء مشروع جديد من قائمة File واختر Project
- ستظهر لك نافذة اختر منها Console Application
- اختر C من القائمة
- قم بكتابة اسم المشروع Project title كما تشاء
- اضغط إنهاء finish

سيقوم البرنامج بإنشاء ملف main.c تحت sources , قم بفتح الملف وحذف محتوياته ليصبح فارغ blank file , وقم بعد ذلك بعمل بناء للبرنامج build

ستجد هذا الخطأ بالأسفل في Build messages

```
undefined reference to `WinMain@16'
error: ld returned 1 exit status
```

الخطأ هنا ليس ان الملف فارغ فمن الممكن ان نضيف عشرات الاسطر من الاكواد وملفات اخري لهذا المشروع ويظهر نفس الخطأ , الخطأ هنا ان الملف لا يحتوي علي الدالة الاساسية والتي يبدأ عندها تنفيذ البرنامج والتي تسمى بـ entry point

- تعامل مع رسائل الخطأ بكل مرونة , اقرأ الخطأ جيداً لتعرف سبب الخطأ واذا لم تصل لشيء قم بالبحث عنه كما هو في جوجل
- ما ستتعلمه من رسائل الخطأ , غالباً لن تنساه
- لا تتوقع ان تكتب كود بلا اخطاء من اول مرة سواء كنت مبتدئ او محترف الجميع يخطئ

الدالة الاساسية : main function

لكل برنامج مكتوب بلغة السي دالة اساسية main function والتي يبدأ عندها تنفيذ البرنامج وللتوضيح ان فكرة الدالة الاساسية ليست خاصة بلغة السي فقط , فاللغات الاخرى مثل الجافا والسي شارب لديها هي الاخرى دالة رئيسية

قم بالتعديل علي الملف السابق ليصبح كالتالي

```
int main()
{

    return 0;
}
```

01 - intro\01.c

قم بتشغيل البرنامج ستجد انه يعمل بدون اخطاء , وتظهر نافذة ال Console , هذا البرنامج فعليا لا يقوم بعمل شئ مفيد هو فقط يحتوي علي الدالة الاساسية , سنتطرق الي شرح الدوال لاحقا ولكن ما اريدك ان تعرفه اننا لكي نبدأ في كتابة برنامج يؤدي وظيفة معينة سنضع الاكواد الخاصة بين تلك الاقواس {} curly brackets

الطباعة علي الشاشة Hello World

الان سنقوم بكتابة برنامج للطباعة علي شاشة ال Console جملة Hello World , قم بكتابة الكود التالي

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

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

Hello World

01 - intro\02.c

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

يقوم هذا السطر باضافة الملف `stdio.h` الى المشروع , هذا الملفات تسمى الملفات الراسية Header files ,وتحتوي علي ثوابت ونماذج prototypes دوال نحتاج اليها في الكود (سنتطرق لهذا لاحقا) , ماذا عن هذا الملف تحديدا ؟ `stdio` هي اختصار لـ Standard input and output اي ان هذا الملف يحوي كل نماذج الدوال الخاصة بالدخل والخرج .وظيفة `#include` هي اضافة الملفات الراسية الي الكود

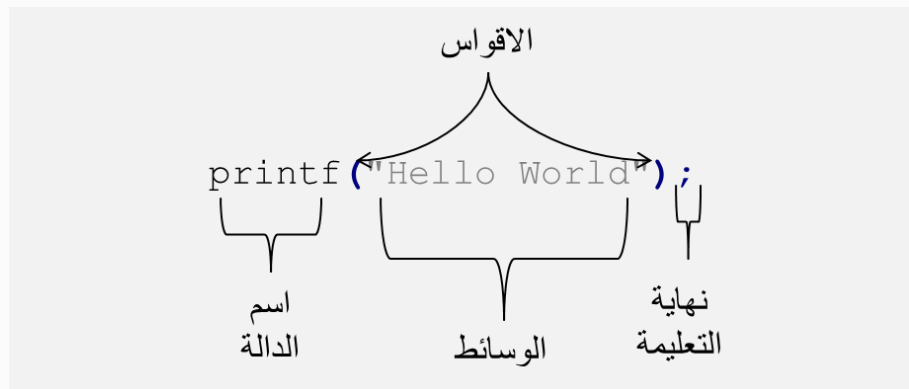
الثاني

```
printf("Hello World");
```

هذا الدالة هي احد الدوال الموجودة ضمن الملف `stdio.h` ووظيفتها هي الطباعة علي الشاشة او الخرج وهي اختصار لـ Print format اي طباعة تنسيق معين علي الشاشة ,ولها اكثر من استخدام سنراه لاحقا وهذا ابسطها الذي يقوم بطباعة مابين الاقواس " " علي الشاشة

هذه الدالة تتكون من ثلاثة اجزاء :

- اسم الدالة `printf`
- الاقواس () وتستخدم لاستقبال الوسائط التي تحتاجها الدالة , وليس شرطاً ان تحتاج الدالة الي وسائط يمكن ان نجد دوال لا تحتاج الي وسائط ,ولكن لا يمكن الاستغناء عن الاقواس ,فهي مميزة للدوال
- الوسائط وهي القيم التي تحتاج اليها الدالة لاداء مهمتها في حالتنا النص المراد طباعته
- الفاصلة المنقوطة ; semicolon وهي تعني نهاية التعليمة او السطر في السي statement end



من الاشياء المهمة بالدوال الخاصة بالطباعة هو هل الدالة تقوم باضافة سطر جديد new line بعد طباعة النص ام لا ,في حالة تلك الدالة لا فمثلا الكود السابق يمكن كتابته بطريقة اخري

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

```
int main()
```

```
{
```

```
printf("Hello ");
```

```
printf("World");
```

```
return 0;
```

```
}
```

Hello World

01 - intro\03.c

-يمكن كتابة اكثر من تعليمة statement في نفس السطر بشرط الفصل بينها ب فاصلة , او فاصلة منقوطة ;

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

او

```
printf("Hello ");printf("World");
```

- ايضا يمكن للتعليمة الواحدة ان تكتب علي اكثر من سطر ولكن سنحتاج الي اضافة \ anti-slash اذا تم الفصل بين علامتي الاقتباس "

```
printf("Hello \
```

```
World");
```

- التعليقات : Comments

لنفترض انك كتبت برنامجا يؤدي وظيفة معينة , وبعد فترة من الزمن لاي سبب من الاسباب احتجت ان تعرف كيف يعمل هذا الكود او كيف يؤدي وظيفته , دعك من البرنامج البسيط الذي كتبناه بالاعلي نحن نتحدث عن اسطر بالعشرات علي الاقل ,هل ستفهم كيف يعمل الكود وقتها بالطبع لا

لنفترض انك بعد كتابة الكود , ارسلته لاحد اصدقائك الذي احتاج هو الاخر لاي سبب ان يفهم الكود ,لعلك صادفت احد الاكواد يوما خاصة open source ولم تفهم كيف يعمل , ومن هنا تاتي اهمية التعليقات ,لجعل الكود واضح

```
/*
```

```
author : mohamed hussein
```

```
date : 1/9/2017, 10:10:00 pm
```

```
description : hello world program
```

```
*/
```

```
#include<stdio.h>//include header file
```

```
int main()  
{  
    //print hello word statement  
    //using printf function  
    printf("Hello World");  
  
    return 0;  
}
```

Hello World

01 - intro\04.c

نتيجة البرنامج ستبقي كما هي , اذا ان التعليقات يتم تجاهلها اثناء بناء البرنامج build , هنا استخدمنا نوعين من التعليقات اولا النصوص متعددة الاسطر والتي يجب ان توضع بين /* */ , ثانيا التعليق بالسطر الواحد والذي يسبقه // , لا يوجد قيود علي وضع التعليقات ولكن لا تكثر منها بحيث يبدو الكود كما لو كان تعليقات وبها بعض الاكواد .

- طباعة علي اكثر من سطر :

ماذا لو اردنا طباعة كلمة ABC متفرقة كل حرف في سطر منفصل , يمكننا فعل ذلك باستخدام نفس الدالة printf , مع اضافة الرمز (\n) n هي اختصار new line ل سطر جديد

```
#include<stdio.h>  
  
int main()  
{  
    //using \n to add new line  
    printf("A\nB\nC");  
    return 0;  
}
```

A
B
C

01 - intro\06.c

ال escape sequences -

مجموعة من الحروف كلا منها يبدأ بعلامة (\) وللكل منها معني معين كما في الجدوال التالي

\n	سطر جديد	\?	تعني ؟
\t	مسافة افقية مثل الضغط علي مفتاح Tab	\b	مسافة عادية
\\	تعني \	\v	مسافة راسية مثل Tab مفتاح ولكن بشكل راسي
\'	تعني '	\ooo	رقم بالنظام الثماني
\"	تعني "	\xhh	رقم بالنظام السداسي عشر
\f	تسعي form feed	\r	تسعي Carriage return

تقوم ال \t باضافة مسافة افقية مساوية للضغط علي مفتاح Tab

<code>printf("name : \t mohamed \n");</code>	name: mohamed
--	---------------

تستخدم \, \', \", \? لطباعة الحروف التي تليها

<code>printf("name : \\ mohamed \\ \n");</code>	name:\ mohamed \
<code>printf("name : \' mohamed \' \n");</code>	name:'mohamed'
<code>printf("name : \" mohamed \" \n");</code>	name:"mohamed"
<code>printf("how are you \? \n");</code>	how are you ?



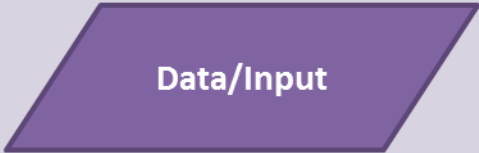
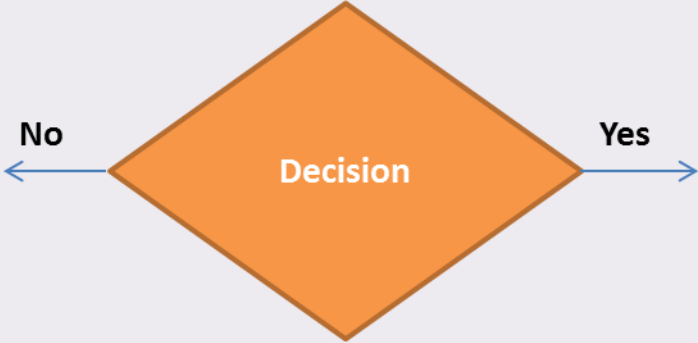
وللتوضيح هي لا تستخدم في الطباعة فقط , هي في الاساس توضع داخل النصوص او كحرف منفصل ولكن المعني يبقى كما هو , كما في المثال التالي سنشرح المتغيرات بداية من الفصل القادم

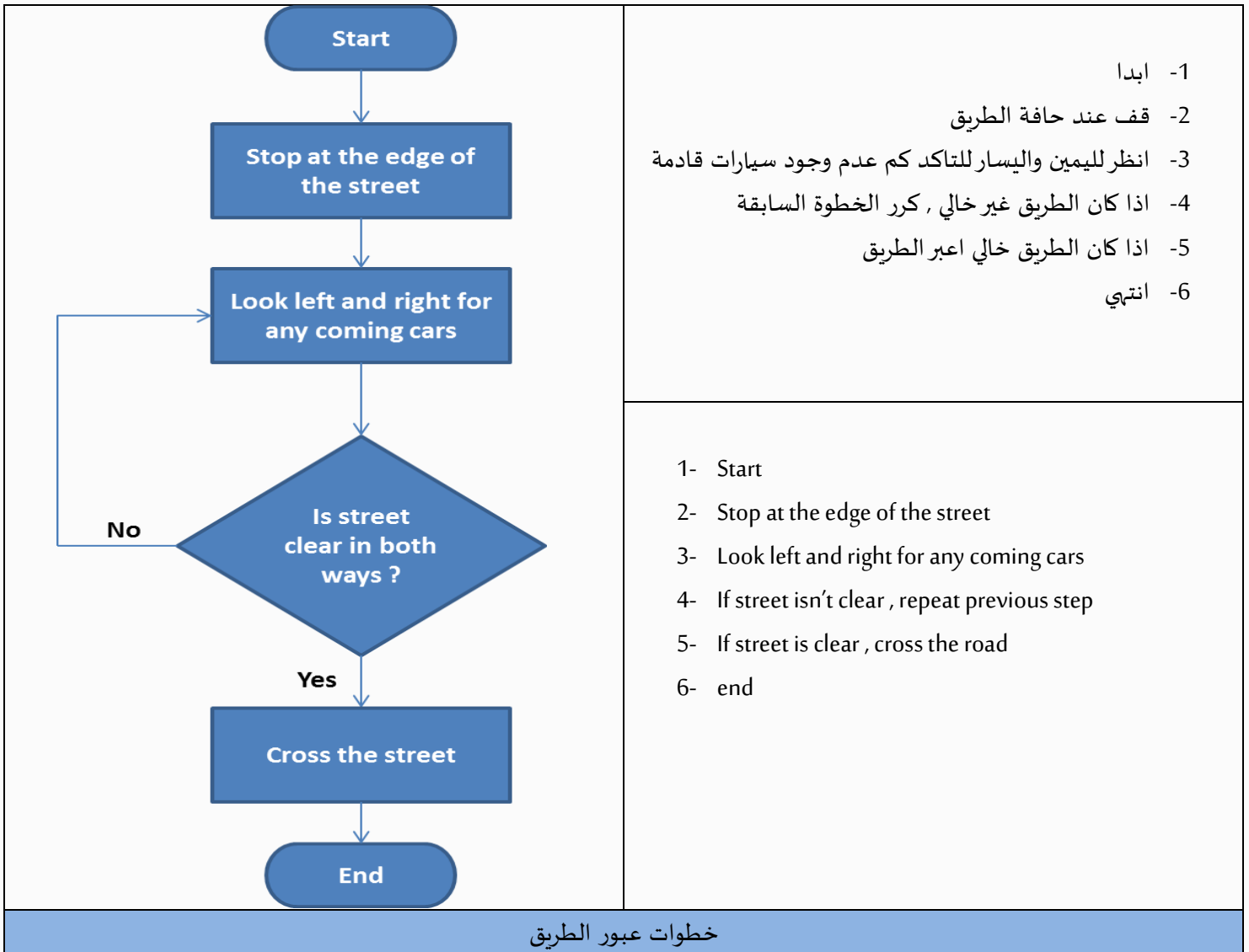
<code>char *msg = "OK\r\f";</code>
<code>char ch = '\\"';</code>

الخوارزميات Algorithms

اغلب البرامج التي نقوم بتطويرها هي حل لمشكلة ما , لحل اي مشكلة ما لابد اولاً من فهمها جيداً وبعد ذلك نشرع في ايجاد الحل المناسب , وهذا الحل ليس هو كتابة الكود وانما هو ايجاد مجموعة الخطوات التي ستتبع لحل المشكلة بصرف النظر عن لغة البرمجة المستخدمة ومن هنا جاءت فكرة الخوارزميات والتي تعرف بانها مجموعة من الاجراءات والخطوات المتبعة لحل مشكلة ما وهناك طريقتين لكتابة حل مشكلة ما

- 1- الكود المزيف Pseudo code : تكتب خطوات الحل باي لغة كالعربية او الانجليزية مثلاً وتكون مرتبة علي شكل خطوات واضحة ومحددة
- 2- مخطط التدفق Flow chart : مجموعة من الاشكال الهندسية المتعارف عليه لتوصيف حل اي مشكلة والموضحة بالجدول التالي

الشكل	الاستخدام
	<p>لتحديد بداية ونهاية البرنامج او الحل , حيث توضع start في البداية , بينما توضع end في النهاية</p>
	<p>تستخدم في معالجة البيانات واظهار معلومات للمستخدم</p>
	<p>يستخدم في حالة التفاعل مع المستخدم مثل ادخال قيم معينة</p>
	<p>يستخدم في القرارات , مثل جملة الشرط If في حالة تحقق الشرط يتجه التنفيذ الي الفرع نعم Yes وفي حالة عدم تحقق الشرط يتجه التنفيذ الفرع لا No</p>
الاشكال الهندسية الخاصة بمخطط التدفق	



الخلاصة

- يبدأ تنفيذ البرنامج في السي من الدالة الاساسية
- لابد لاي برنامج سي ان يحتوي علي دالة اساسية واحدة فقط
- تستخدم الدالة printf للطباعة علي الشاشة
- التعليقات هامة جدا , فهي تساعد علي فهم البرنامج بعد ذلك
- يجب قبل البدا في كتابة الاكواد ان يكون لدينا تصور واضح للمشكلة , ولخطوات الحل


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

```
int Main()
```

```
{
```

```
printf("Hello World");
```

```
return 0;
```

```
}
```

- بالطبع لا يمكن ذلك لان لغة السي حساسة لحالة الاحرف case sensitive ف Main تختلف عن main ولو قمنا بتشغيل هذا البرنامج سيظهر الخطأ الذي ظهر لنا سابقا بان البرنامج لا يحتوي علي الدالة الاساسية

2- ماذا سيحدث اذا قمنا بكتابة الدالة الاساسية اكثر من مرة

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

```
int main()
```

```
{
```

```
printf("Hello World");
```

```
return 0;
```

```
}
```

```
int main()
```

```
{
```

```
printf("Hello World");
```

```
return 0;
```

```
}
```

- لن يعمل البرنامج وسيظهر خطأ (redefinition of main) والذي يعني اننا قمنا بتعريف الدالة main اكثر من مرة وهذا غير مسموح به في السي اذا لو حدث ذلك فايهما ستعتبر الدالة الاساسية, وليس لدينا امكانية دمج تعريفات الدالة جميعا في تعريف واحد لا في السي ولا غيرها

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

- سيعمل البرنامج , ولكن ستظهر رسالة تحذير warning تختلف من مترجم لآخر ولكن خلاصتها ان المترجم لم يستطع التحقق من استخدامنا للدالة printf لانه لم يجد النموذج prototype الخاص بها (سنعرفه لاحقا) , وبالتالي سيفترض ان تلك الدالة تحتاج عدد متغير من الوسائط , جرب الكود التالي وستعرف الفرق

```
int main()
{
    printf(22);
    return 0;
}
```

2

Variables and Arithmetic operators

المتغيرات والعمليات الحسابية

المتغيرات

تبني الفكرة الأساسية للحاسب الآلي علي اننا لدينا مجموعة من البيانات Data نقوم بعمل معالجة Processing لها بطريقة ما حسب كل مشكلة وتحويلها الي معلومات Information , لناخذ برنامج الالة الحاسبة كمثال سنجد انه ياخذ بيانات من المستخدم وهي الاعداد ويقوم بعمل معاجة لها ك الجمع او الطرح او الضرب او القسمة ويعطي النتيجة

لنفترض اننا نريد كتابة برنامج ياخذ من المستخدم رقمين ويعطي نتيجة جمع الرقمين , بحيث ياخذ الشكل التالي

- please insert the first number : 12
- please insert the second number: 14
- the result of 12 + 14 is 26

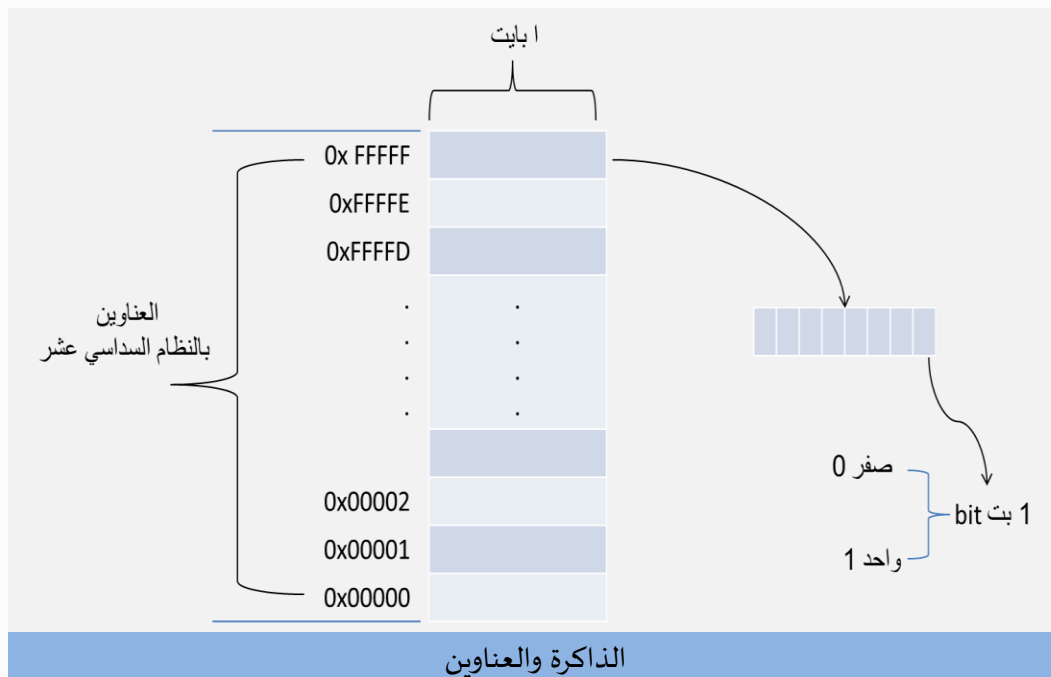
السؤال هنا اين سنحتفظ بقيمة الرقم الاول والثاني لحين اجراء عملية الجمع ؟ حسنا تماما مثلنا نحن البشر فان الحاسب لديه ذاكرة هو الآخر علي اختلاف انواعها

الذاكرة Memory

الذاكرة هي مكان لتخزين البيانات ويوجد نوعين منها :

- الذاكرة العشوائية random access memory RAM : تعرف بانها ذاكرة متطايرة Volatile memory اي انها تفقد محتوياتها بمجرد انقطاع التيار ولكنها تسمح بالقراءة منها والكتابة عليها
- ذاكرة القراءة فقط read only memory ROM : تتميز بانها ذاكرة غير متطايرة non-volatile memory اي انها تحتفظ بمحتوياتها حتي عند انقطاع التيار وهي في الاصل وحيث يشير الاسم تسمح بالقراءة منها فقط ولكن الان توجد انواع تسمح بالقراءة والكتابة

تتكون كل ذاكرة من مجموعة من البايتات , كل بايت له عنوان (بالسداسي عشر) يشير اليه , كما في الشكل التالي



وحدات التخزين Storage units

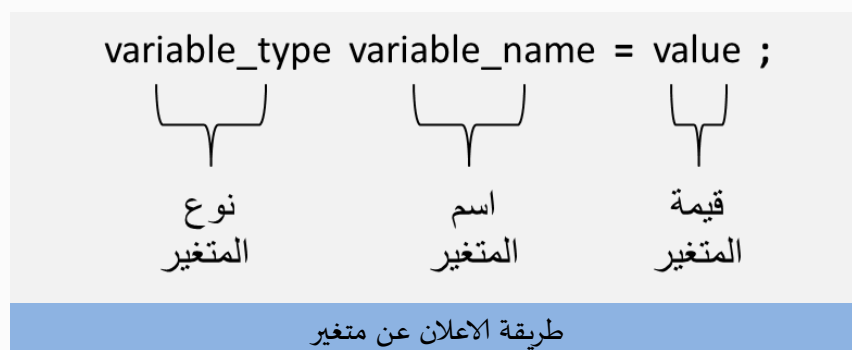
لكل شئ وحدة تميزه عن غيره فالطول يقاس بالمتر والسنتيمتر وهكذا , والوزن يقاس الكيلوجرام والجرام اما في الحاسب فان اصغر وحدة هي البت Bit والذي يكون داخليا عبارة عن ترانستور ومكثف وهو اما يحتوي علي 1 او 0 وكل 8 بت تكون بايت وكل 1024 بايت تكون كيلوبايت وهكذا كما في الجدوال التالي

الوحدة	الوصف
البت Bit	اما صفرا او واحد
البايت Byte	8 بت
الكلمة Word	2 بايت , 16 بت
الكيلوبايت Kilobyte KB	1024 بايت
الميجابايت Megabyte MB	1024 كيلوبايت
الجيجابايت Gigabyte GB	1024 ميغابايت
التيرابايت Terabyte TB	1024 جيجابايت

المتغيرات Variables

هي اماكن يتم حجزها في الذاكرة لغرض استخدامها في تخزين البيانات , وتختلف في الحجم Size من متغير لآخر فلدينا 1 بايت و 2 بايت و 4 بايت و 8 بايت و 10 بايت , ولكل نوع من البيانات نوع من المتغيرات للتعامل معها فالذي يتعامل مع الارقام الصحيحة يختلف عن الذي يتعامل مع الارقام الحقيقية , واخير ان لكل متغير اسم مميز له Identifier يتم الاشارة اليه به سواء للكتابة عليه او للقراءة منه .

- لابد قبل استخدام اي متغير ان نقوم بالاعلان عنه Variable declaration او بمعنى اخر حجز مكان له اولا في الذاكرة , ويتم ذلك من خلال كتابة نوع المتغير ثم اسم المتغير ثم قيمته في حال اعطيناه قيمة مباشرة او مبدئية ويفضل ذلك .



- السي لغة حساسة لحالة الاحرف case sensitive فـ a تختلف عن A .

- يتم اعطاء قيمة للمتغير من خلال رمز التساوي (=) assignment

- لقراءة محتوى اي متغير يتم كتابة اسمه فقط identifier

- اسم المتغير يجب ان يكون ضمن هذا الشروط

- لا يبدأ برقم مثل 2num
- لا يحتوي علي مسافات first num
- لا يحتوي علي رموز \$,%,#,@ وهكذا ما عدا (_) underscore
- لا يكون مستعمل مسبقا
- لا يكون من الكلمات المحجوزة للسي وهي الكلمات الخاصة باللغة ذاتها مثل char, struct, float, int

امثلة لاسماء متغيرات

اسماء خاطئة	اسماء صحيحة
<code>int 5num; //start with number</code>	<code>int number ;</code>
<code>char #ch; //start with symbol #</code>	<code>char _ch;</code>
<code>float first num ; //contain space</code>	<code>float result;</code>

متغيرات الاعداد الصحيحة Integers variable

المتغيرات التي تتعامل مع الاعداد الصحيحة هي

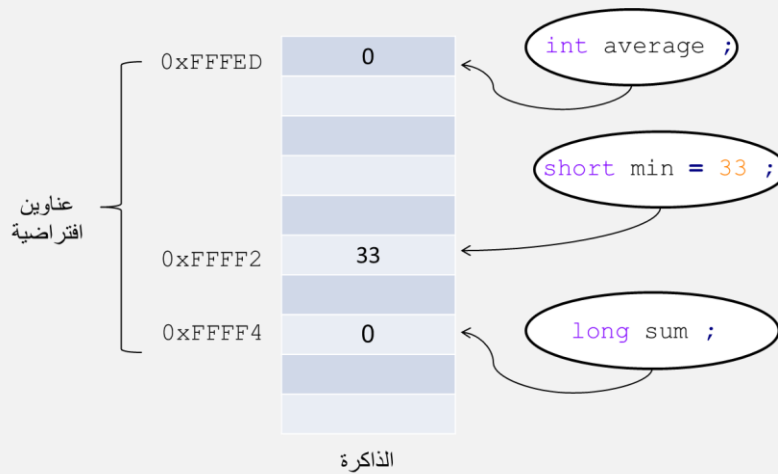
المتغير	الحجم	ملاحظات
int	غالبا 4 بايت	
short	2 بايت	اختصار لـ short int
long	8 بايت	اختصار لـ long int

لكل متغير حدود limits للقيم التي يتعامل معها , لذا يوجد اكثر من حجم حسب التطبيق ولتحديد حدود اي متغير

- اذا كان المتغير لا يتعامل مع الاعداد السالبة Unsigned numbers فان اقل قيمة هي الصفر واقصي قيمة نستخدم تلك الصيغة $(2^n - 1)$ حيث n هو عدد البتات bits فمثلا المتغير من نوع short عدد ال bits يساوي عدد البايتات * 8 اي $8 * 2$ يساوي 16 بت و 2^{16} تساوي 65536 وطرح واحد تصبح اقصي قيمة 65535
- اذا كان الرقم يتعامل مع الارقام السالبة signed numbers فان اقل قيمة (-2^{n-1}) واقصي قيمة فمثلا المتغير $(2^{n-1} - 1)$ فمثلا المتغير من نوع short عدد ال bits يساوي عدد البايتات * 8 اي $8 * 2$ يساوي 16 بت و 2^{16} تساوي 65536 وطرح واحد تصبح اقصي قيمة 65535

- قمنا بطرح واحد من عدد البتات في حالة الاعداد السالبة لانه في النظام الثنائي binary system يتم تخصيص اخر بت من جهة اليسار least significant bit للإشارة 0 تعني موجب و 1 تعني سالب

```
int average ;
short min = 33 ;
long sum ;
```



مثال : سنكتب برنامج يقوم بحساب نتيجة جمع رقمين وطباعة الناتج

```
//first number to be added
short firstNumber = 12 ;

//second number to be added
short secondNumber = 14 ;

//result of addition
short sum = firstNumber + secondNumber ;

//print addition result
printf("sum is %d \n",sum);
```

sum is 26

برنامج جمع رقمين باستخدام المتغيرات [02 - var&arithm\01.c]

- التعليقات كما عرفنا مسبقا توضح الغرض من البرنامج ومن كل سطر بعد ذلك

```
short firstNumber = 12 ;
short secondNumber = 14 ;
```

- قمنا بالاعلان عن متغيرات من نوع short واستخدمنا هذا النوع لان الارقام التي تعاملنا معها صغيرة وسيؤدي الغرض , بالطبع الانواع الاخرى

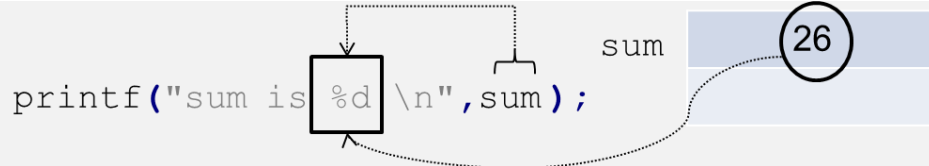
صحيحة , لكننا يجب عليك ان تختار النوع الذي يناسب احتياجاتك فقط حتي لا تهدر الذاكرة بلا جدوى

- firstNumber , secondNumber هي اسماء تلك المتغيرات , قمنا باعطاء قيم مباشرة لكل متغير وذلك باستخدام علامة التساوي (=) assignment operator الذي يستخدم لاعطاء قيمة لمتغير ما

```
short sum = firstNumber + secondNumber ;
```

- اعلنا عن متغير لتخزين ناتج جمع الرقمين , الجزء علي يمين علامة التساوي يسمى expression والذي يقوم بجمع محتوى المتغير firstNumber والمتغير secondNumber من خلال علامة الجمع وهي احدي العلامات الخاصة بالعمليات الحسابية في السي

```
printf("sum is %d \n",sum);
```

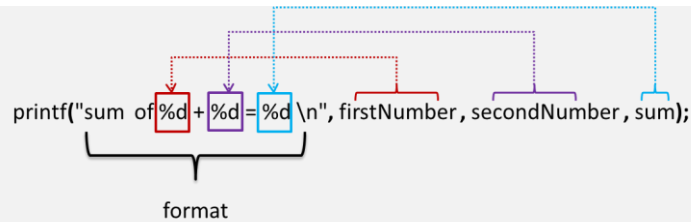


- استخدمنا دالة الطباعة printf ونلاحظ وجود الرمز %d الذي يخبر الدالة ان تستبدل مكان تلك الرمز بمحتوي المتغير sum , واستخدمنا هذا الرمز تحديدا لاننا نتعامل مع اعداد صحيحة حيث d تعني decimal ويمكن استبدالها بـ %i والتي تعني integer

- اضفنا المتغير sum كوسيط ثاني الدالة ليتم استبداله مكان %d

- %d , %i وغيرها كما سنري لاحقا حسب كل نوع تسمي conversion specifier ويجب ان يكون عدد الوسائط بعد اول وسيط مساوي لعدد تلك الرموز وبنفس الترتيب حسب كل نوع , والوسيط يمكن ان يكون متغير او قيمة مباشرة من نفس النوع فيمكننا كتابة التالي

```
printf("sum of %d + %d = %d \n", firstNumber , secondNumber , sum);
```



Sum of 12 + 14 = 26

ويمكن لتلك الدالة ان تاخذ قيم مباشرة ك 12 في المثال نفس النتيجة السابقة

```
printf("sum of %d + %d = %d \n", 12, secondNumber , sum);
```

- يمكن ان نقوم بلاعلان عن اكثر من متغير من نفس النوع في سطر واحد

```
int first , second = 44 , third , forth ;
```


- لتغير محتوى اي متغير يكتب اسم المتغير الذي سبق ان حجزنا له في الذاكرة ثم علامة (=) ثم القيمة المراد تعيينها له , كما يمكن للمتغير ان ياخذ قيمة متغير اخر

```
first = 22;  
second = 56;  
third = first;
```

متغيرات الاعداد الحقيقية floating-point variables

- المتغيرات التي تتعامل مع الاعداد الحقيقية هي

المتغير	الحجم	ملاحظات
float	4 بايت	
double	8 بايت	

- السؤال هنا لماذا يوجد نوع مختلف للتعامل مع الاعداد الحقيقية ؟ لان الاعداد الحقيقية يتم تخزينها بطريقة مختلفة في الذاكرة

مثال سنكتب برنامج يقوم بايجاد حاصل ضرب رقمين حقيقيين وطباعة الناتج

```
float firstNum , secondNum , result;  
  
//assign value to each variable  
  
firstNum = 2.4 ;  
secondNum = 2.0 ;  
  
//calculate result  
result = firstNum * secondNum ;  
  
//print result  
printf("result is %f \n",result);
```

result is 4.800000

[02 - var&aritm\02.c]

- البرنامج لا يختلف كثيرا عن برنامج الجمع باستخدام متغيرات الاعداد الصحيحة , قمنا هنا باستخدام متغيرات من نوع float لاننا نتعامل مع اعداد حقيقية

- استخدمنا هنا علامة (*) لاجراء عملية الضرب , وهي ايضا احد العلامات لاجراء العمليات الحسابية في السي

-قمنا باستخدام الرمز %f الذي يستخدم في طباعة الاعداد الحقيقية من نوع float

- السي تستخدم الفاصلة العشرية (.) للتمييز بين الاعداد الصحيحة والاعداد الحقيقية ف 2.0 غير 2

متغير الحروف characters

يتم الاعلان عنه باستخدام كلمة char , وهو اصغر نوع في المتغيرات 1 بايت فقط , من 0 الي 255 في حالة القيم الموجبة فقط , من -128 الي 127 في حالة القيم السالبة وما علاقة الحروف بالاعداد السالبة والموجبة ؟ لان هذا النوع يتم تخزينه داخليا في الذاكرة بنفس الطريقة المستخدمة في الاعداد الصحيحة فيمكن التعامل معه علي انه من متغيرات الاعداد الصحيحة ايضا

مثال : برنامج يقوم بطباعة كلمة Hello باستخدام متغير الحروف

```
char ch1 = 'H' , ch2 = 'e' , ch3 = 'l' , ch4 = 'l' , ch5 = 'o' ;  
  
//print char variables content  
printf("%c%c%c%c%c \n", ch1 , ch2 , ch3 , ch4 , ch5 );
```

Hello

[02 - var&aritm\03.c]

- قمنا بالاعلان عن 5 متغيرات من نوع char واعطينا كل متغير قيمة مباشرة الحروف في السي توضع بين علامتي اقتباس

('') , استخدمنا الرمز %c الخاص بطباعة الحروف في السي

- مثال برنامج جمع رقمين باستخدام متغير الحروف

```
//first number to be added  
char firstNumber = 12 ;  
  
//second number to be added  
char secondNumber = 14 ;  
  
//result of addition  
char sum = firstNumber + secondNumber ;  
  
//print addition result  
printf("sum is %d \n",sum);
```

sum is 26

[02 - var&aritm\04.c]

- قمنا بالتعديل علي برنامج الجمع السابق واستبدلنا كل متغير short بـ char , النتيجة لم تتغير

- نلاحظ اننا استخدمنا الرمز %d لطباعة محتوى متغير ولم نستخدم الرمز %c لاننا نريد من الدالة ان تتعامل interpret مع المتغير علي انه متغير اعداد صحيحة

- لو استخدمنا الرمز %c سيتم طباعة القيمة المقابلة لـ 26 من جدول الاسكي (سنطرق اليه لاحقا) وهي (→)

```
printf("sum is %c\n",sum);
```

sum is →

- امثلة علي تنسيق طباعة المتغيرات

- طباعة رقم صحيح مع تحديد اقل عدد للارقام باستخدام %2d , %3d %4d وهكذا

```
int firstStd = 120;
```

```
int secondStd= 90;
```

```
printf("weight is %3d KG\n",firstStd);
```

```
printf("weight is %3d KG\n",secondStd);
```

weight is 120 KG

weight is 90 KG

- حددنا اقل عدد للارقام 3 , قتم اضافة مسافة للرقم 90 ليصبح 3 ارقام digits

- طباعة رقم حقيقي مع تحديد عدد الارقام بعد العلامة العشرية

```
float first = 34.567 ;  
float second = 7844.3215 ;  
  
printf("degree is %.2f\n",first);  
printf("degree is %.3f\n",second);
```

```
degree is 34.57  
degree is 7844.321
```

- طباعة قيمة بالنظام السداسي عشروالنظام الثماني باستخدام الرموز %x,%o علي الترتيب

```
int value = 127 ;  
  
printf("value is %x in hexadecimal\n",value);  
printf("value is %o in octal\n",value);
```

```
value is 7f in hexadecimal  
value is 177 in octal
```

- الاعداد الموجبة و الاعداد السالبة :

يتم تحديد اذا كان المتغير سيتعامل مع اعداد سالبة ام لا من خلال الكلمتين unsigned للاعداد الموجبة فقط او signed للاعداد الموجبة والسالبة ,
الحالة الافتراضية لمتغيرات الاعداد الصحيحة انها تتعامل مع الاعداد الموجبة والسالبة اي signed كما في المثال التالي

```
signed int first = -127 ;
unsigned int second = 12;
int third = -140;

printf("value is %d \n",first);
printf("value is %d \n",second);
printf("value is %d \n",third);
```

```
value is -127
value is 12
value is -140
```

الثوابت constants

الثوابت مثل المتغيرات في الغرض منها و انواعها ولكنها تختلف عن المتغيرات انها بمجرد ان تاخذ قيمة معينة لا يمكن تغييرها اي ان محتواها ثابت ولذا فهي تاخذ قيمتها مباشرة عند الاعلان عنها ويوجد طريقتين للاعلان عن الثوابت

- باستخدام الكلمة المحجوزة const قبل نوع المتغير في الاعلان عنه
- باستخدام الكلمة #define

```
#define constant_name constant_value
```

اسم الثابت قيمة الثابت

باستخدام الكلمة #define

```
const variable_type variable_name = value ;
```

نوع المتغير اسم المتغير قيمة المتغير

باستخدام الكلمة const

الفرق بين الطريقتين ان الاولى تقوم بحجز مكان في الذاكرة لهذا الثابت , اما الثانية تسمي string replacement حيث يتم استبدال اي اسم الثابت داخل البرنامج بقيمته وذلك قبل ترجمة البرنامج

جري العرف ان تكون اسماء الثوابت حروف كبيرة capital letters وذلك ليتم تمييزها عن المتغيرات داخل الاكواد , امثلة للاعلان عن ثوابت

```
#define MAX 20
#define SIZE 100

const int LIMIT = 33 ;
const char CH = 'A' ;
```

مثال

```
//define max constant
#define MAX 20

int main()
{
    //define another constant
    const int num = 80 ;

    printf("MAX value is %d \n",MAX );

    printf("num value is %d \n ",num);

    return 0;
}
```

MAX value is 20
num value is 80

[02 - var&aritm\05.c]

عرفنا الثابت الاول MAX باستخدام #define واعطيناه القيمة 20 ونلاحظ عدم وجود فاصلة منقوطة في نهاية التعريف لان هذا الطريقة ليست احدي تعليمات البرنامج انما هي احد التوجيهات directives التي يتم تنفيذها قبل ترجمة البرنامج

```
#define MAX 20
```

عرفنا ثابت اخر بالطريقة الاعلان عن المتغيرات وباستخدام الكلمة const واعطيناه القيمة 80

```
const int num = 80 ;
```

قمنا بطباعة الثابتين باستخدام الدالة printf

اذا حاولنا تغيير قيمة احد الثوابت لن يعمل البرنامج وستظهر رسالة خطأ اننا نحاول تغيير قيمة ثابت read-only variable

- العمليات الحسابية arithmetic operators :

تتم العمليات الحسابية في السي باستخدام رموز + , - , * , / , % والتي تعرف بـ arithmetic operators

الرمز	الوظيفة
+	الجمع
-	الطرح

الضرب	*
القسمة	/
باقي القسمة	%

مثال برنامج يوضح استخدام تلك الرموز

```
//operands
int firstNumber = 100 , secondNumber = 3 ;

printf("addition result is %d \n",firstNumber + secondNumber );
printf("subtraction result is %d \n",firstNumber - secondNumber );
printf("multiplication result is %d \n",firstNumber * secondNumber );
printf("division result is %d \n",firstNumber / secondNumber );
printf("remainder result is %d \n",firstNumber % secondNumber );
```

addition result is 103
subtraction result is 97
multiplication result is 300
division result is 33
remainder result is 1

[02 - var&aritm\06.c]

- سؤال وجواب

- هل يمكن استخدام متغير قبل ان نعلن عنه ؟

```
int main()
{
    printf("value is %d \n",num);
    return 0;
}
```

لا يمكن , سيحدث خطأ , وهو ان المتغير num لم يتم الاعلان عنه مسبقا undeclared

- هل يمكن تسمية المتغير باسم مشابه لمتغير من نوع آخر؟

```
int num;  
float num;
```

لا يمكن ذلك , هذا يسمى تعارض في نوع المتغير conflicting types , نفس الاسم احدهما int والآخر float

- هل يمكن استخدام متغير قبل اعطاء قيمة مبدئية او مباشرة له ؟

```
int main()  
{  
    int num;  
    printf("value of num is %d \n",num);  
    return 0;  
}
```

هنا الكود سيعمل بدون اخطاء ولكن قيمة المتغير غير متوقعة ولذا ينصح باعطاء قيمة مبدئية للمتغيرات عند الاعلان عنها وذلك لتجنب القيم الغير متوقعة.



Operators and Expressions

المؤثرات والعبارات

الرموز operators

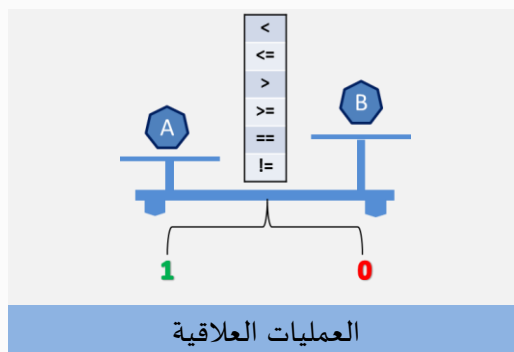
تحتوي السي علي بعض الرموز التي تستخدم في العمليات الحسابية , العمليات المنطقية وعمليات المقارنة وغيرها وهي كالآتي

رموز العمليات الحسابية : arithmetic operators

هي الرموز الخاصة باجراء العمليات الحسابية ك الجمع والطرح وخلافه وقد سبق لما التعامل معها في الفصل السابق

رموز العمليات العلاقية :relational operators

هي الرموز الخاصة باجراء عمليات المقارنة بين القيم ونتيجة تلك العمليات اما صحيحة (1) او خاطئة (0)



الرمز	الوظيفة
$a > b$	هل a اكبر من b ام لا
$a \geq b$	هل a اكبر من او تساوي b ام لا
$a < b$	هل a اصغر من b ام لا
$a \leq b$	هل a اصغر من او تساوي b ام لا
$a == b$	هل a تساوي b ام لا
$a != b$	هل a لا تساوي b ام لا

تستخدم تلك الرموز غالبا للتحقق من شرط معين كما سنري بعد ذلك

مثال برنامج يوضح استخدام تلك الرموز

```
printf("12 > 10 = %d \n", 12 > 10);
printf("10 >= 12 = %d \n", 10 >= 12);
printf("10 >= 10 = %d \n", 10 >= 10);
printf("44 < 55 = %d \n", 44 < 55);
printf("44 <= 22 = %d \n", 44 <= 22);
printf("44 <= 44 = %d \n", 44 <= 44);
printf("10 == 55 = %d \n", 10 == 55);
printf("10 == 10 = %d \n", 10 == 10);
printf("10 != 10 = %d \n", 10 != 10);
printf("10 != 44 = %d \n", 10 != 44);
```

```
1 = 1 < 12
0 = 12 <= 10
1 = 10 <= 10
1 = 55 > 44
0 = 22 >= 44
1 = 44 >= 44
0 = 55 == 10
1 = 10 == 10
0 = 10 != 10
1 = 44 != 10
```

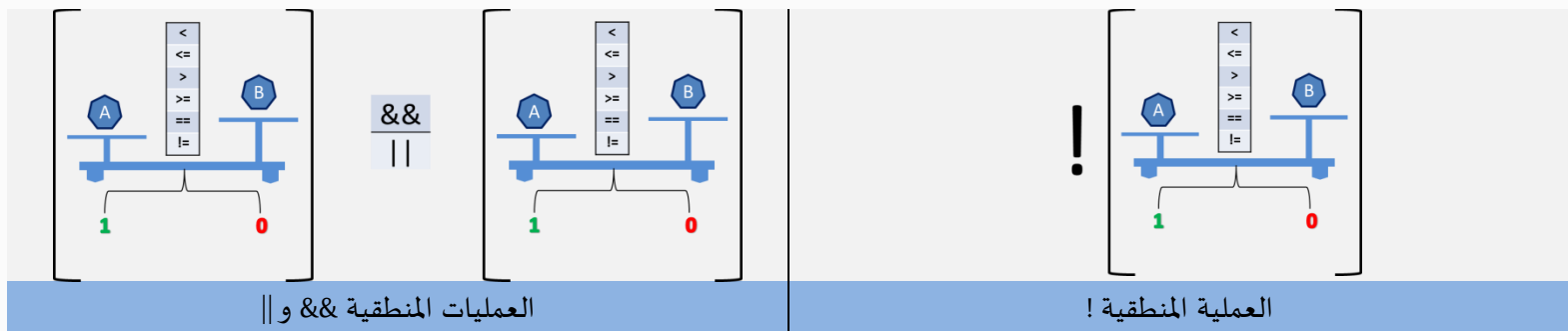
[03 - operators\01.c]

رموز العمليات المنطقية : logical operators

تستخدم مع رموز العمليات العلاقية للجمع بين اكثر من شرط وهي:

- && بمعنى (و) تعطي true اذا كان الطرفين true , غير ذلك تعطي false
- || بمعنى (او) وتعطي true اذا كان احد الطرفين او كليهما true , وتعطي false في حالة واحدة اذا كان الطرفين false
- ! تقوم بعكس النتيجة من true الي false والعكس

يفضل استخدام الاقواس حول كل شرط للوضوح ولتجنب الازعاج



جدول الحقيقة لكل عملية منها يوضح كل احتمالات طرفي العملية ونتيجة العملية في كل حالة ,يمكن اجراء تلك العمليات علي اكثر من شرط وليس شرطين فقط

A	الناتج	A	B	الناتج	A	B	الناتج
0	1	0	0	0	0	0	0
1	0	1	0	1	1	0	0
		0	1	1	0	1	0
		1	1	1	1	1	1
!				&&			

مثال برنامج يوضح استخدام العمليات المنطقية

```
printf("(12 >10) && (12 > 11) = %d \n", (12 >10) && (12 > 11) );
printf("(14 >10) && (12 != 11) = %d \n\n", (14 >10) && (12 != 11) );

printf("(12 >10) || (12 > 11) = %d \n", (12 >10) || (12 > 11) );
printf("(14 >10) || (12 != 11) = %d \n\n", (14 >10) || (12 != 11) );

printf("!(12 >10) = %d \n", !(12 >10) );
printf("!(14 >10) = %d \n", !(14 >16) );
```

1 = (11 < 12) && (10 < 12)

1 = (11 != 12) && (10 < 14)

1 = (11 < 12) || (10 < 12)

1 = (11 != 12) || (10 < 14)

0 = (10 < 12)!

1 = (10 < 14)!

[03 - operators\02.c]

مؤثرات الزيادة والنقصان :increment and decrement operators

تستخدم لزيادة قيمة المتغير بقدر واحد او انقصه بقدر واحد , احد الرموز التي تستخدم بكثرة مع الحلقات التكرارية , ويوجد منه نوعين

- Pre يوضح الرمز قبل المتغير مثل ++num , --count
- Post يوضع الرمز بعد المتغير مثل num++ , coun--

الفرق بينهما لا يختلف علي المتغير ذاته تأثير الزيادة او النقصان سيحدث حتما , الفرق يظهر اذا قمنا بتعين تلك القيمة لمتغير اخر فمثلا

```
int num = 4 ;
int x = ++num ;
```

هنا المتغير num سيصبح 5 , اما المتغير x سيصبح 4 ام 5 في حالة استخدام pre فان الزيادة تحصل اولاً ثم تعين القيمة للمتغير اي ان الكود بالاعلي مساوي للكود التالي , المتغير x يساوي 5

```
int num = 4 ;
num = num + 1;
int x = num ;
```

اما في حال post فان تعين القيمة يحدث اولاً ثم الزيادة

```
int num = 4 ;
int x = num++ ;
```

قيمة num تساوي 5 وقيمة x تساوي 4 , هذا الكود مساوي لـ

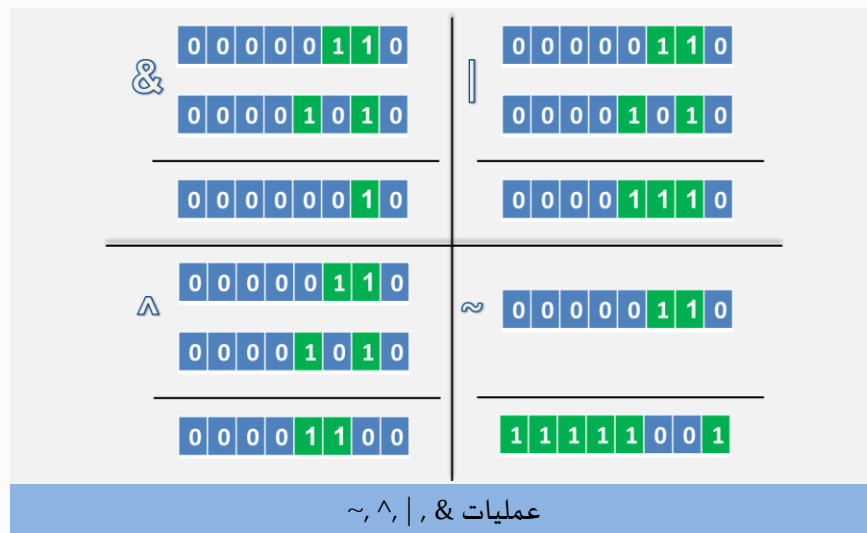
```
int num = 4 ;
int x = num ;
num = num + 1;
```

ونفس الشيء بالنسبة لمؤثر النقصان .

رموز خاصة بالبتات bitwise operators :

تلك الرموز تتعامل مع المتغير او القيمة علي مستوي البت وهي

الرمز	المعني
&	And
	Or
^	Xor
>>	Right shift الازاحة لليمين
<<	Left shift الازاحة لليساار
~	Not عكس



من الهام ان نعرف ان ترتيب البتات يبدأ من الصفر فاول بت جهة اليمين والذي يسمى Least significant bit ترتيبه رقم صفر والذي يليه رقم 1 وهكذا

مثال للتوضيح :

```
char firstNum = 4 ;//binary = 0000 0100
char secondNum = 10 ;//binary = 0000 1010

char andResult = firstNum & secondNum ; // (0000 0100) & (0000 1010) = 0000 0000
char orResult = firstNum | secondNum ; // (0000 0100) | (0000 1010) = 0000 1110
char xorResult = firstNum ^ secondNum ; // (0000 0100) & (0000 1010) = 0000 1110
char notResult = ~firstNum ; // ~(0000 0100) = 1111 1011

char rightShiftResult = firstNum >> 2; // (0000 0100) = 0000 0001
char leftShiftResult = firstNum << 2; // (0000 0100) = 0001 0000
```

لاحقا سنعرف الاستخدامات العملية لتلك المؤثرات , ولكن احد الملاحظات الهامة هنا ان الازاحة لليمين تساوي القسمة علي 2 * عدد مرات الازاحة وكذلك الازاحة لليساو تساوي الضرب في 2 * عدد مرات الازاحة كالتالي

```
4 >> 2 = 1 ; //0000 0100 >> 2 = 0000 0001 which equal 4 / (2 * 2)
4 << 1 = 8 ; //0000 0100 << 1 = 0000 1000 whhich equal 4 * 2 * 1
```

مؤثرات تعيين القيم assignment operators :

هذه المؤثرات لا تفعل شئ جديدا من الناحية الوظيفية غير انها تسهل في كتابة الكود فبدلا من ان نكتب (x = x+3) يمكن ان نكتب (x += 3) كالتالي

```
x += 3 ; // equal to x = x + 3
x -= 3 ; // equal to x = x - 3
x *= 3 ; // equal to x = x * 3
x /= 3 ; // equal to x = x / 3
```

```
x %= 3; // equal to x = x % 3
x &= 3; // equal to x = x & 3
x |= 3; // equal to x = x | 3
x ^= 3; // equal to x = x ^ 3
x >>= 3; // equal to x = x >> 3
x <<= 3; // equal to x = x << 3
```

مؤثر التحويل من نوع لآخر :cast operator

تسعى عملية تحويل المتغير او البيانات من نوع لآخر بال casting ويوجد نوعين :

- تحويل ضمني implicit يقوم بها المترجم ذاته وتسمى ايضا promotion , حيث انه لحساب قيمة expression يجب ان تكون جميع الحدود من نفس النوع لذا يقوم المترجم بتحويل الحدود المختلفة في النوع الي النوع الاعلي
- تحويل صريح explicit يقوم به المبرمج لتحويل نوع معين لآخر وذلك من خلال الصيغة الاتيه

```
int first=12;
float second = 23.8;
float total = first + second ;
```

هنا سيقوم المترجم بتحويل المتغير first الي float ليصبح محتواه 12.0 ويكون ال expression عبارة عن 23.8 + 12.0 وهذا هو التحويل الضمني

```
int first=12;
float second = 23.8;
float third = (float)first;
float total = first + third ;
```

قمنا بتحويل المتغير first من int الي float بشكل صريح , وبالتالي لن يحتاج المترجم الي اجراء اي تحويل لان الحدود من نفس النوع الان

يجب ملاحظة ان تحويل نوع اعلي الي نوع اقل مثل تحويل int الي char يؤدي الي فقد في البيانات truncate

العبارة الجبرية expressions :

هي مجموعة من الحدود بينها احد الرموز الموجود بالسي , وهذا الحدود اما متغيرات او قيم مباشرة وهي اشبه بالمعادلة الرياضية مثل الاتي

```
h=x+y*z;  
g=x/3-4+t;  
l=x>4||y<6;
```

كل ما هو علي يمين علامة (=) يسمى expression و يسمى بالـ rvalue(right value) اي انه ياتي علي يمين علامة (=) فقط ولا يمكن ان ياتي علي اليسار , اما lvalue(left value) فهي التي تاتي علي اليسار ويمكن ان تاتي علي اليمين ايضا وهي عبارة عن متغير الذي يمثل في النهاية عنوان في الذاكرة

الاسبقية في حساب قيمة الـ expression:

هناك قواعد لتحديد اي ترتيب حساب قيمة كل حد في المعادلة فلو لدينا $(5*4+2)$ هل النتيجة 22 ام 30 , بالنسبة لقواعد الاسبقية النتيجة 22

الاسبقية في العمليات الحسابية هي () ثم * ثم / ثم % ثم + ثم - , والافضل ان نكثر من استخدام () لجعل المعادلة اكثر وضوحا ولتجنب الخطاء وليتم حساب قيمة الحدود بالترتيب الذي نريده

جدول الترتيب

استخدامات المؤثرات الخاصة بالبتات

- تصفير قيمة بت معين reset ويتم ذلك باستخدام &
- التحقق من قيمة بت معين هل 1 او 0 ويتم ذلك باستخدام &
- وضع قيمة بت معين set بـ 1
- تبديل قيمة بت معين Toggle اذا من 1 الي 0 والعكس وذلك باستخدام ^

تصفير قيمة بت معين reset

تعطي & القيمة 1 اذا كانت القيمتين بـ 1 اما اذا كان احدهما بـ 0 فالنتيجة دائما 0 , لذا اذا اردنا ان نضع قيمة بت معين بـ 0 نقوم بعمل & لتلك القيمة مع قيمة اخري بها 0 مكان البت المراد تصفيره و 1 في الاماكن الاخري , ولتحديد القيمة الاخري تلك يمكن ان نستخدم الاشارة مع مؤشر النفي او نحدد القيمة مباشرة بانفسنا كما يلي



مثال

```
//value
unsigned short value = 0b1000110;
//generate the second value
unsigned short reset = ~ (1<< 2);

//before
printf("value before is 0x%x \n",value);
//apply reset
value &= reset ;
//after
printf("value after is 0x%x \n",value);
```

value before is 0x46

value after is 0x42

[03 - operators\03.c]

حجزنا متغير صحيح واعطيناه قيمة بالنظام الثنائي وذلك باستخدام (0b) قبل القيمة , ثم لايجاد القيمة التي سيتم عمل & معها وذلك لتصفير البت الثاني استخدمنا الازاحة جهة اليسار مع النفي

```
unsigned short reset = ~ (1<< 2);
```

حيث 2 هي رقم البت المراد تصفيره , ما حدث هنا اننا قمنا اولا باجراء ازاحة للرقم 1 جهة اليسار برقم البت المراد تصفير ثم عكس ناتج الازاحة وذلك لوضع صفر مكان الواحد وبقيّة الرقم بواحد

```
(1<< 2) = (0000 0001) << 2 = (0000 0100)
~(1<< 2) = ~ (0000 0100) = (1111 1011)
```

وبذلك فان المتغير reset سيحوي القيمة (1111 1011) , وكان يمكن ان نضع له تلك القيمة مباشرة بدلا من استخدام الازاحة والنفي لكنها اسهل بكثير , قمنا بعد ذلك بعمل & للقيمة الاصلية مع قيمة المتغير reset وتخزين الناتج مكان القيمة الاصلية

```
value &= reset ;
```

```
value = 0 1 0 0 0 1 1 0
&
reset = 1 1 1 1 1 0 1 1
-----
value = 0 1 0 0 0 0 1 0
```

وتم طبع القيم بالنظام السداسي عشر

التحقق من قيمة بت معين

يتم ذلك بعمل & لهذا للقيمة الاصلية مع قيمة اخري للتحقق بشرط ان تتكون تلك القيمة من واحد مكان البت المراد التحقق منه والباقي اصفار , ويستخدم كشرط لجملة if (سنتطرق اليها لاحقا) , كما بالشكل



مثال

```
unsigned short value = 0b1010110;
//generate the check value
unsigned short check = (1<< 4); //bit number is 4

if( (value & check ) == check )
{
    printf("bit value is 1\n");
}
else
{
    printf("bit value is 0\n");
}
```

bit value is 1

[03 - operators\04.c]

- استخدمنا الازاحة (1<< 4) لايجاد القيمة التي سيتم التحقق بها , للتحقق من البت الرابع , فستكون القيمة (0001 000) وكان يمكن وضع تلك القيمة مباشرة بدلا من استخدام الازاحة , عند عمل & بين تلك القيمة والقيمة الاصلية اذا كانت قيمة البت بـ 1 سيكون الناتج هو نفس قيمة التحقق اما اذا كانت قيمة البت بـ 0 سيكون الناتج 0 , استخدمنا جملة الشرط للتحقق من الناتج , فاذا كان ناتج العملية مساوي لقيمة التحقق فان قيمة البت 1 والا فهي 0

```
check = (1<< 4) = (0001 0000)
```

```
value = 0 1 0 1 0 1 1 0
&
check = 0 0 0 1 0 0 0 0
-----
(0 0 0 1 0 0 0 0) == check => true ,printf("bit value is 1\n");
```

وضع قيمة بت معين set

تماما مثل تصفير البت باستخدام & ولكن هنا نستخدم | وذلك لانها تعطي 1 اذا كان احد البتين او كليهما قيمته 1 , لذا سنقوم بعمل | مع قيمة بها 1 في مكان البت المراد وضعه بـ 1



مثال

```
unsigned short value = 0b1010110;
//generate the set value
unsigned short set = ( 1<< 5 );

//before
printf("value before is 0x%x \n",value);
//apply set
value |= set ;
//after
printf("value after is 0x%x \n",value);
```

value before is 0x56

value after is 0x76

[03 - operators\05.c]

بعد حجز متغير للقيمة الاصلية واعطاءه قيمة مباشرة بالنظام الثنائي , استخدمنا الازاحة لتحديد القيمة التي ستستخدم في وضع قيمة البت , لاننا نريد وضع قيمة البت الخامس قمنا بعمل ازاحة بمقدار 5 , وبذلك ستكون القيمة (0010 0000) , قمنا بعدها بعمل | بين تلك القيمة والقيمة الاصلية وتخزين الناتج في مكان القيمة الاصلية

```
value |= set ;
```

```
set = (1<< 5) = (0000 0001) << 5 = (0010 0000)
```

```
value = 0 1 0 1 0 1 1 0
```

```
|
```

```
set = 0 0 1 0 0 0 0 0
```

```
-----
```

```
value = 0 1 1 1 0 1 1 0
```

تبديل قيمة بت معين Toggle

تستخدم في التبديل \wedge , حيث انه اذا تم عمل \wedge ب 1 مع قيمة اخري فاذا كانت تلك القيمة 0 سيكون الناتج 1 واذا كانت 1 سيكون الناتج 0 وبذلك قمنا بتبديل تلك القيمة



مثال

```
//value
unsigned short value = 0b1010110;
//generate the set value
unsigned short toggle = ( 1<< 1 );

//before
printf("value before is 0x%x \n",value);
//apply reset
value ^= toggle ;
//after
printf("value after is 0x%x \n",value);
```

value before is 0x56

value after is 0x54

[03 - operators\06.c]

استخدمنا الاشارة ايضا لايجاد القيمة المستخدمة في التبديل , الاشارة هنا بقدر 1 لاننا نريد تبديل البت رقم 1 , قمنا بعدها بعمل | بين تلك القيمة والقيمة الاصلية وتخزين الناتج في مكان القيمة الاصلية

```
value ^= toggle ;
```

```
toggle = (1<< 1) = (0000 0001) << 1 = (0000 0010)

value  = 0 1 0 1 0 1 1 0
^
toggle = 0 0 0 0 0 0 1 0
-----
value  = 0 1 0 1 0 1 0 0
```

الخلاصة

- تحتوي السي علي مجموعة من الرموز لاجراء العمليات الحسابية والمنطقية والعلاقية وغيرها
- ان تحويل نوع اعلي الي نوع اقل مثل تحويل int الي char يؤدي الي فقد في البيانات truncate
- يجب مراعاة الاسبقية في العمليات عند كتابة expressions ويفضل استخدام الاقواس تجنبنا للاخطاء وللوضوح
- تستخدم المؤثرات bitwise لتعامل علي مستوي البت

سؤال وجواب

- ما نتيجة العملية التالية؟

```
(12 && -44)
```

النتيجة هي 1 لان السي تعتبر الصفر false واي عدد اخر true

- نتيجة الطباعة الاتية ؟

```
int num = 12;
printf("%d\n", num++);
```

النتيجة هي طباعة القيمة 12 , وقيمة المتغير ستصبح 13



Program Control

التحكم في سير البرنامج

- الـ statement :

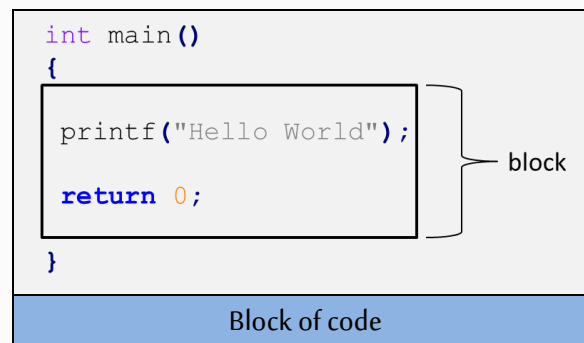
هي امر مكتمل ينتهي بفاصلة منقوطة (;) واما ان يكون دالة او معادلة expression المهم ان تنتهي بفاصلة منقوطة كالتالي

```
h = x + y * z ;  
y = 2 ;  
printf("hello world");  
return 0 ;  
! = x > 4 || y < 6 ;
```

كل سطر يعتبر statement لانه ينتهي بفاصلة منقوطة

الـ statement block :

مجموعة من statement التي تبدأ بـ { وتنتهي بـ } , مثل تلك الموجودة في الدالة الرئيسية main function , فنحن نكتب الاكواد الخاصة بنا داخل الـ block الخاص بتلك الدالة

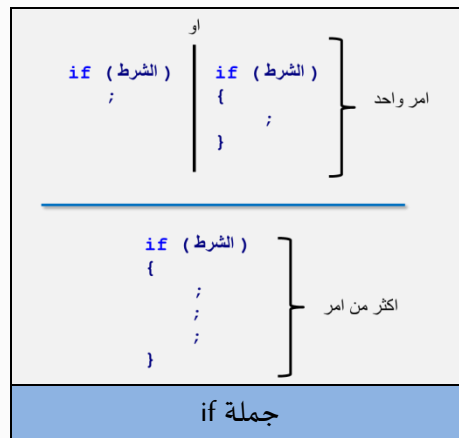


التحكم في سير البرنامج program control :

هو تحديد الترتيب الذي سيتم من خلاله تنفيذ البرنامج , فقد نحتاج الي تنفيذ جزء من البرنامج بناء علي شرط معين او تنفيذه عدة مرات

- جملة الشرط if :

تستخدم كتعبير عن القرار decision , حيث يتم من خلالها تنفيذ التعليمات بناء علي شرط معين , ويمكن تنفيذ تعليمة واحدة حيث تكتب تلك التعليمة مباشرة بعدها , اما في حال تنفيذ block من التعليمات فلا بد من استخدام الاقواس {} كما بالشكل



مثال برنامج يقوم بطباعة جملتين مختلفتين بناء علي شرط

```
int num = 30 ;

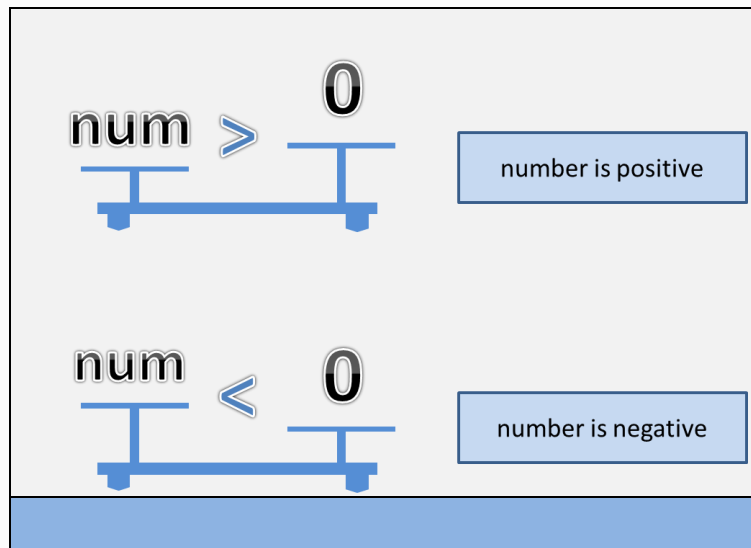
if( num > 0)
{
    printf("number is positive\n");
}

if( num < 0)
{
    printf("number is negative\n");
}
```

number is positive

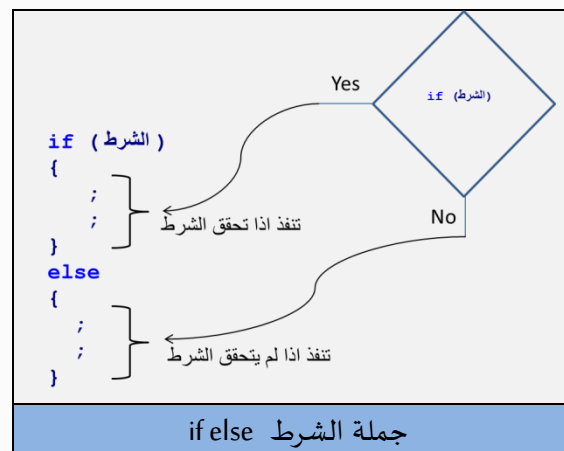
[04 - programControl\01.c]

هنا استخدمنا المؤثرات العلاقية relational operator للتحقق من الشرط , اذا كان الرقم اكبر من صفر , او اذا كان اقل من الصفر , الشرط الاول هو الذي تحقق ما الشرط الثاني لو يتحقق وبالتالي تم تجاهل التعليمات الخاصة به



- جملة الشرط if else :

هنا يتم تنفيذ تعليمات معينة اذا تحقق الشرط وتنفيذ تعليمات اخرى اذا لم يتحقق الشرط , الجزء الخاص بـ else اختياري ليس شرطاً ان يستخدم مع if .



مثال تعديل علي البرنامج السابق باستخدام else

```
int num = -11 ;

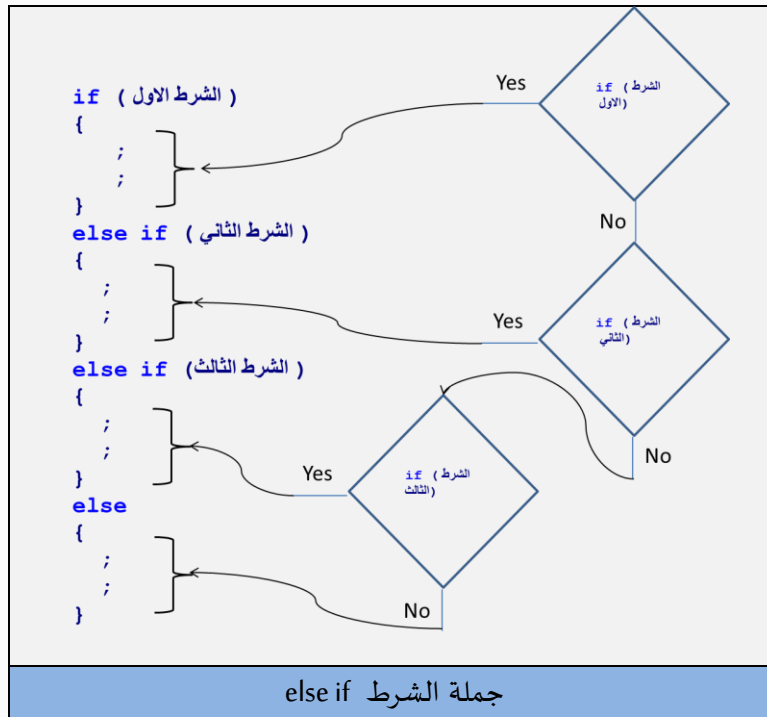
if( num > 0)
{
    printf("number is positive\n");
}
else
{
    printf("number is negative\n");
}
```

number is negative

الشرط الاساسي لم يتحقق وبالتالي تم تجاهل التعليمات الخاصة بـ if وتم تنفيذ تلك الخاصة بـ else

- جملة الشرط else if :

هنا لدينا اكثر من شرط للتحقق منه , لكل شرط التعليمات الخاصة به , ويمكن استخدام else ايضا في حالة عدم تحقق اي شرط



```

int num = -11 ;

if( num == 0)
{
}
else if( num > 0)
{
    printf("number is positive\n");
}
else
{
    printf("number is negative\n");
}
  
```

number is negative

هنا الشرط الاول لم يتحقق وايضا الثاني لم يتحقق قتم تنفيذ التعليمات الخاصة ب else

مثال : برنامج يقوم بالتحقق من الرقم زوجي ام فردي

```
int num = 13 ;  
  
if( (num % 2) == 0)  
{  
    printf("number is even\n");  
}  
else  
{  
    printf("number is odd\n");  
}
```

number is odd

[04 - programControl\04.c]

الفكرة هنا في شرط التحقق ذاته , حيث ان الارقام الزوجية هي تلك التي تقبل القسمة علي 2 اوبمعني اخر ان باقي القسمة علي 2 يساوي صفر, وما عدا ذلك ارقام فردية ويمكن كتابة الشرط بطريقة اخري

```
if( !(num % 2) )  
{  
    printf("number is even\n");  
}  
else  
{  
    printf("number is odd\n");  
}
```

حيث ان السي تعتبر الصفر false واي رقم اخر non zero تعتبره true

```
//student degree
int degree = 260;

//total degree
int total = 300;

//grade
float grade = (float)degree / total ;

if( grade >= 0.85)
{
    printf("Excellent\n");
}
else if ( ( grade >= 0.75 ) && ( grade < 0.85 ) )
{
    printf("Very good\n");
}
else if ( ( grade >= 0.65 ) && ( grade < 0.75 ) )
{
    printf("good\n");
}
else if ( ( grade >= 0.55 ) && ( grade < 0.65 ) )
{
    printf("pass\n");
}
else
{
    printf("fail\n");
}
```

Excellent

[04 - programControl\05.c]

يتم حساب التقدير عن طريق قسمة درجة الطالب علي الدرجة النهائية وحسب حدود كل تقدير يتم تحديد تقدير الطالب

```
float grade = (float)degree / total ;
```

هنا ناتج القسمة دائما اقل من الواحد , وحيث ان البسط والمقام كلاهما من نوع int فالناتج سيكون صفر دائما لاي درجة لذا قمنا بتحويل البسط الي float بشكل صريح وسيقوم المترجم بتحويل المقام بشكل ضمني لانه كما قلنا مسبقا ان المعاملات يجب ان تكون من نفس النوع , واذا لم يحدث يقوم المترجم بعمل تحويل للاعلي وسيجد البسط من نوع float لذا سيحول المقام الي نفس النوع لان float اعلي من int

```
( grade >= 0.75 ) && ( grade < 0.85 )
```

استخدمنا المؤثر && للجمع بين شرطين , لاحظ استخدام الاقواس لجعل الشرط اكثر وضوحا وتجنبنا للاخطاء

كان يمكن ان نجعل المتغير degree من نوع float بدلا من int ليكون

```
//student degree
float degree = 260;
//total degree
int total = 300;

//grade
float grade = degree / total ;
```

- جملة الشرط nested if :

يمكن ان نضع اكثر من جملة شرط داخل التعليمات الخاصة بشرط معين , ويراعي استخدام الاقواس {} لوضوح الكود

مثال برنامج للتحقق اذا كان الرقم زوجي او فردي بشرط ان يكون اقل من او يساوي 100

```
int num = 200 ;

if (num <= 100)
{
    //nested if
    if( (num % 2) == 0)
    {
        printf("number is even\n");
    }
    else
```

```

{
    printf("number is odd\n");
}

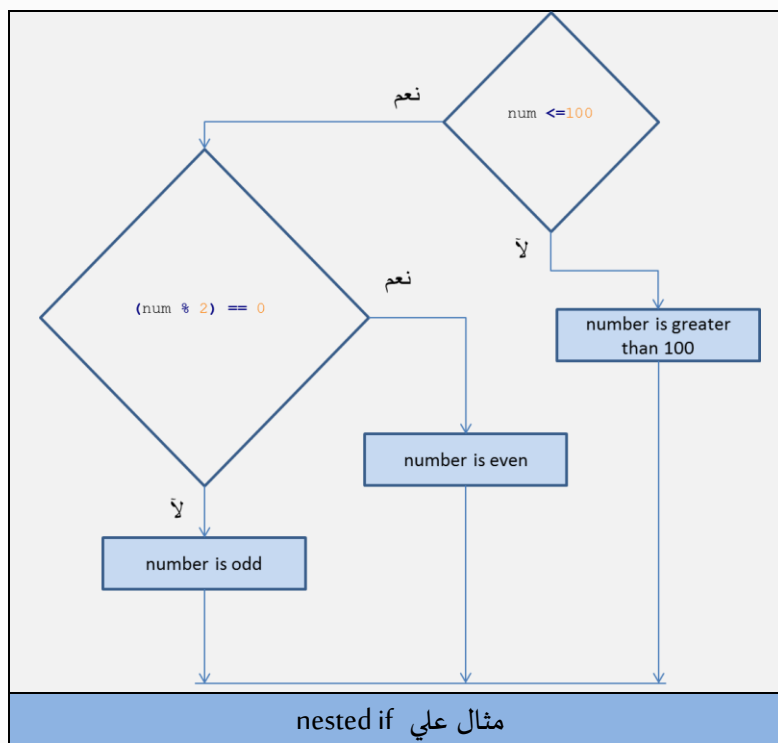
else
{
    printf("number is greater than 100");
}

```

number is greater than 100

[04 - programControl\06.c]

بالرغم من ان 200 رقم زوجي الا ان البرنامج لم يستطع التحقق من ذلك لانها خالفت الشرط الاعلي وهي ان تكون اقل من 100 كما بالشكل الاتي



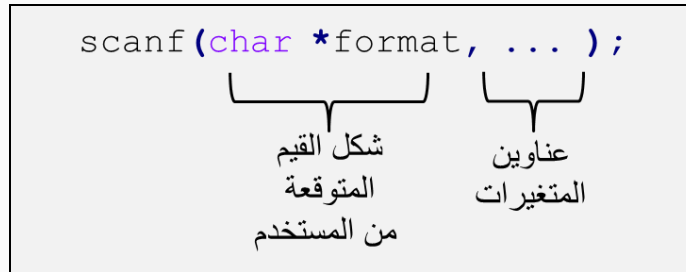
- استقبال قيم من لوحة المفاتيح input :

تعاملنا في السابق مع الدالة printf للطباعة علي وحدة الاخراج الرئيسية وهي الشاشة , هناك دالة اخري تسمي scanf تستخدم لاستقبال قيم من وحدة الادخال الرئيسية وهي لوحة المفاتيح , كثيرا ما نحتاج الي تفاعل المستخدم مع البرنامج من خلال ادخال القيم المختلفة

تأخذ هذه الدالة وسيطين الأول هو شكل القيم المتوقعة من المستخدم format وهي تأخذ رموز مثل %d,%c الثاني هو عناوين المتغيرات التي سيتم وضع القيم بها

عنوان المتغير يتم معرفته باستخدام الرمز & متبوعا باسم المتغير (&num), سنعرف لاحقا لماذا عنوان المتغير وليس المتغير ذاته , تأخذ تلك الدالة حروف من لوحة المفاتيح وتقوم بالتعامل معهم interpret وفقا لما هو موجود في شكل القيم format

تماما مثل الدالة printf يجب ان يكون عدد الوسائط بعد الأول مساوي لعدد الرموز في format



مثال برنامج يقوم باستقبال قيمة من المستخدم وطباعتها

```
//to save user input
int value;

//ask user to type value
printf("please insert your value : ");

//get user value
scanf("%d",&value);

//print user value
printf("your value is %d",value);
```

```
please insert your value : 44
your value is 44
```

[04 - programControl\07.c]

قمنا باستخدام الرمز %d لاننا نتعامل مع ارقام صحيحة في خانة format وبعدها قمنا بتمرير عنوان المتغير كوسيط ثان

```
//to save user inputs
int number ;
char ch ;

//ask user to type value
printf("please insert number and character separated by space : ");
//get user value
scanf("%d %c",&number,&ch);
//print user value
printf("number is %d\n",number);
printf("character is %c\n",ch);
```

```
please insert number and character separated by space : 67 h
number is 67
character is h
```

[04 - programControl\08.c]

استخدمنا الرمزين %d و %c بينهما مسافة داخل الـ format وبذلك ستقوم الدالة بانتظار قيمتين من المستخدم ستتعامل مع الاول علي انه `int` والثاني علي انه `char`

يمكن كتابة تلك الدالة مرتين لكل قيمة بدلا من مرة واحدة كالتالي

```
//get user value
scanf("%d",&number);
fflush(stdin);
scanf("%c",&ch);
```

الجديد اننا سندخل الرقم ونضغط علي مفتاح الادخال Enter وبعدها ندخل الحرف ونضغط ايضا علي مفتاح الادخال , الدالة `scanf` لا تقرا من لوحة المفاتيح بشكل مباشر وانما تقرا من `input buffer` عبارة عن وحدة تخزينة تحتوي علي كل القيم التي تم استقبالها من لوحة المفاتيح ولم يتم قراءتها من قبل الدالة `scanf`, لذا قمنا بتفريغ محتويات `input buffer` قبل القراءة منه لانه يمكن ان يحتوي علي قيم سابقة غير مرغوب فيها عن طريق الدالة `fflush(stdin)` والتي تاخذ وسيط وهو الـ `buffer` المراد تفريغه , وفي حالتنا `stdin` اي `standard input`

يمكن ان نحدد الحرف الفاصل بين كل قيمة واخري separator في حال استفبال اكثر من قيمة في نفس الدالة كالتالي

```
//to save user inputs
int number ;
char ch ;

//ask user to type value
printf("please insert number and character separated by # :");
//get user value
scanf("%d#%c",&number,&ch);
//print user value
printf("number is %d\n",number);
printf("character is %c\n",ch);
```

```
please insert number and character separated by # : 44#s
number is 44
character is s
```

[04 - programControl\09.c]

حددنا # حفاصل بين كل قيمة بدلا من المسافة (space) , لذا اصبح الـ "%d#%c" وعند ادخال القيم كانت 44#s

جملة الشرط switch:

تستخدم عندما نريد مقارنة قيمة متغيرة مع قيم ثابتة , هذه القيم اما عدد صحيح integer او حرف character ولاشئ غير ذلك فلا يمكن ان يكون عدد حقيقي او سلسلة حروف string

```
(القيمة المتغيرة)
switch
{
case قيمة ثابتة :
;
break;
case قيمة ثابتة :
;
break;
default:
;
}
```

```
// user choice
int choice;

//show menu
printf("... Main Menu ...\n");
printf("1 : Start Game \n");
printf("2 : Game Options \n");
printf("3 : High scores \n");
printf("4 : Exit Game \n");

//ask user to select from menu
printf("Your choice : ");

//get user choice
scanf("%d",&choice);

switch (choice)
{
case 1:
    printf("loading game scene ... \n");
    break;
case 2:
    printf("loading game options ... \n");
    break;
case 3:
    printf("loading high scores ... \n");
    break;
case 4:
    printf("game termination");
    break;
default:
    printf("unknown choice !!!\n");
}
```

... Main Menu ...:

1 : Start Game

2 : Game Options

3 : High scores

4 : Exit Game

Your choice : 1

loading game scene ...

[04 - programControl\10.c]

قمنا بعرض القائمة , وطلبنا من المستخدم ادخال اختيار , استخدمنا جملة switch للتحقق من قيمة الاختيار , تأخذ switch وسيط واحد وهو المتغير المراد التحقق من قيمته ويجب ان يكون متغير اعداد صحيحة في حالتنا كان choice , تستخدم كلمة case لاضافة القيم الثابتة المتوقعة للمتغير choice وتكتب case وبعدها احد القيم الثابتة ثم : colon ثم التعليمات المراد تنفيذها

- كلمة case :

تستعمل فقط مع ال switch وهي تعني حالة وبعدها تاتي القيمة الصحيحة الثابتة المراد مقارنتها بالمتغير الموجود كوسيط ل switch اذا كانت المقارنة صحيحة سيتم تنفيذ ما بعدها من تعليمات اما ان كانت خاطئة سيتم الانتقال الى الحالة التي تليها وهكذا , تعتبر كلمة case علامة label لبداية التعليمات الخاصة بها , اي ان المقارنة لا تحدث عند case مثل ال if وانما تحدث في مكان ما ويتم توجيه التنفيذ الى تلك ال case ان كانت صحيحة لو اردنا تنفيذ نفس التعليمات لأكثر من قيمة ثابتة نستخدم لكل قيمة وتوضع التعليمات بعد اخر حالة كالتالي

switch (choice)

```
{  
  case 'a':  
  case 'A':  
  case '1':  
    printf("loading game scene ... \n");  
    break;  
}
```

هذا السطر سيتم تنفيذه اذا كانت قيمة المتغير تساوي 1 او A او a

```
printf("loading game scene ... \n");
```

وهو يساوي للكود التالي

```
if(choice == '1' || choice == 'A' || choice == 'a')
{
    printf("loading game scene ... \n");
}
```

- كلمة break :

تستخدم مع switch للخروج من منها , لانها في حالة عدم وجودها في حالة case معينة وبدا تنفيذ الكود الخاص بتلك الحالة , فان التنفيذ سيستمر الي الحالات التي تليها حتي يجد احد اوامر الخروج من switch ال break اما او return واذا لم يجد سيستمر حتي نهاية ال switch, لنقم بحذف تلك الكلمة من الحالات الاولى والثانية ونقوم بادخال الرقم 2 مثلا لنري النتيجة

```
switch (choice)
{
    case 1:
        printf("loading game scene ... \n");

    case 2:
        printf("loading game options ... \n");

    case 3:
        printf("loading high scores ... \n");
        break;

    case 4:
        printf("game termination");
        break;

    default:
        printf("unknown choice !!!\n");
}
```

Your choice : 2
loading game options...
loading high scores...

عند تحقق الشرط في ال case 2 تم تنفيذ السطر

```
printf("loading game options ... \n");
```

ولعدم وجود تم تنفيذ السطر الخاص بالحالة 3 case

```
printf("loading high scores ... \n");
```

حتى كلمة break في الحالة ال 3 case

```
break;
```

- كلمة default:

هذه الكلمة اختيارية , يمكن كتابة ال switch بدونها , وهي تشبه ال else في جملة الشرط if else ,ينفذ ما بعدها في حال فشل المقارنة في كل الحالات , اذا قمنا بادخال القيمة 10 مثلا للبرنامج السابق ستكون النتيجة كما يلي وذلك لان القيمة 10 لاتساوي اي قيمة في الحالات cases

```
Your choice : 10
unknown choice!!!
```

هل وجود كلمة break في ال default له فائدة ؟

اذا تم كتابة الحالة default في اخر ال switch مثل البرنامج السابق ليست لها فائدة لانها الاخيرة علي اي حال , اما غير ذلك فيجب اضافة break لنفس اسباب ال case اذا قمنا بتبديل الحالة 4 case مع الحالة default في البرنامج السابق وادخلنا القيمة 10 مرة اخري , لعدم وجود break في الحالة default استمر التنفيذ للحالات التالية كما اوضحنا سلفا

```
default:
    printf("unknown choice !!!\n");
case 4:
    printf("game termination");
    break;
```

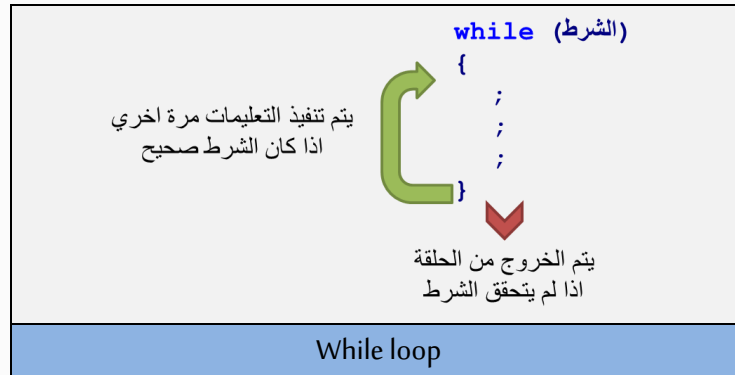
```
Your choice : 10
unknown choice!!!
game termination
```

- الحلقات التكرارية loops :

تستخدم لتكرار تنفيذ مجموعة من التعليمات عدة مرات وفقا لشرط معين , وتنفذ طالما ان الشرط صحيح

- التكرار بـ while :

تأخذ while وسيط واحد عبارة عن expression يتم تنفيذ التعليمات بداخل الـ while طالما ان قيمته لا تساوي الصفر non-zero , واذا ساوى الصفر يتم الخروج منها , هنا يجب ان نحرص على ان قيمة الـ expression تتغير حتي تصل الي نهاية معينة حتي لا تصبح الـ while غير منتهية infinite loop



مثال برنامج لطباعة الارقام من 1 حتي 10

```
// loop counter ,with initialized value of 1
int counter=1;

//print counter value before loop
printf("counter before loop is %d\n",counter);

//loop if counter <= 10
while(counter <= 10)
{
    printf("%d ",counter);

    //increment counter
    counter++;
}

//print counter value after loop
printf("\ncounter after loop is %d\n",counter);
```


counter before loop is 1

1 2 3 4 5 6 7 8 9 10

counter after loop is 11

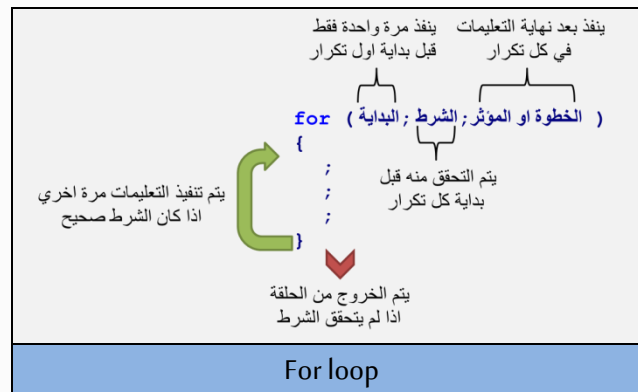
[04 - programControl\11.c]

اعلنا عن المتغير counter واعطيناه قيمة مبدئية 1 , استخدمناه للتحقق من الشرط وايضا في طباعة الاعداد من 1 الى 10 , قمنا بطباعة قيمته قبل بدا الحلقة التكرارية

لتتبع تغير القيمة , تقوم الحلقة بطباعة قيمة المتغير counter طالما ان قيمته اقل من او تساوي 10 (counter <= 10) , استخدمنا مؤثر الزيادة ++ لزيادة المتغير بمقدار واحد في كل مرة حتي نستطيع ان نطبع ارقام من 1 الي 10 , وحتى يصل الـ expression الي نهاية اذا لولم نفعل ذلك ستبقى قيمة المتغير counter دائما تساوي 1 وسيبقى شرط الحلقة صحيح باستمرار والذي يعني ان الحلقة لن تنتهي ابدا infinite loop , عندما تصبح قيمة المتغير 11 سيكون الشرط خاطئ وبالتالي سيتم الخروج من الحلقة الي السطر الذي يليها , طبعنا قيمة المتغير مرة اخري بعد الحلقة .

- التكرار بـ **for** :

تأخذ ثلاث وسائط الاول البداية start , الثاني النهاية او الشرط end or condition , الثالث الخطوة step ونفصل بين كل وسيط بفاصلة منقوطة .(.)



كان تخبر احد ان يقوم بالعد من 0 الي 10 بـ 2 للخطوة فالنتيجة 10 8 6 4 2 0 كالكود التالي

```
// loop counter
```

```
int counter = 0;
```

```
//print counter value before loop
```

```
printf("counter before loop is %d\n",counter);
```

```
//loop if counter <= 10
for(counter =0 ; counter <= 10 ; counter +=2)
{
    printf("%d ",counter);
}
//print counter value after loop
printf("\ncounter after loop is %d\n",counter);
```

counter before loop is 0
0 2 4 6 8 10
counter after loop is 12

[04 - programControl\12.c]

الثلاث وسائط لـ for هي :

- counter =0 يمثل البداية وهنا اعطينا للمتغير قيمة مبدئية لذا احيانا يسمى هذا الوسيط بـ initialization
- counter <= 10 يمثل الشرط الذي هو بمثابة النهاية للحلقة ايضا
- counter +=2 يعبر عن الخطوة او مقدار الزيادة او النقصان في كل مرة ويسمى احيانا بـ iterator , increment , وهنا استخدمنا المؤثر += لزيادة المتغير بمقدار 2

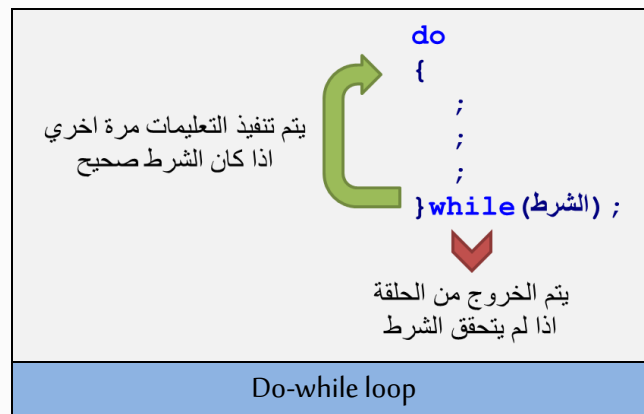
تتحقق الـ for في كل مرة من الشرط فان كان صحيح يتم تنفيذ التعليمات والا يتم الخروج منها , الجزء الخاص بالزيادة او الخطوة وهو الوسيط الاخير يتم تنفيذه بعد تنفيذ التعليمات داخل الحلقة , يتضح ذلك من المقارنة بين while و for لنفس الكود

```
for(counter =0 ; counter <= 10 ; counter +=2)
{
    printf("%d ",counter);
}
```

```
counter =0;//first parameter in for --> start
while(counter <= 10)//second parameter in for
{
    printf("%d ",counter);
    counter +=2;//third parameter in for-->step
}
```

- جملة do while :

لا تختلف كثيرا عن جملة الا انها تضمن تنفيذ التعليمات داخلها مرة واحدة علي الاقل بصرف النظر عن الشرط صحيح ام لا , حيث انها تقوم بتنفيذ التعليمات داخلها اولاً ثم التحقق من الشرط



مثال برنامج ياخذ حروف من المستخدم وطباعها والتوقف عند ادخال حرف q

```
//to store key from user
char ch;

do
{
    //ask user to enter key
    printf("\npress any key , q to exit : ");

    //empty input buffer
    fflush(stdin);

    //get key form user
    scanf("%c",&ch);

    //print user key
    printf("\nyour key is %c\n",ch);
}
while(ch != 'q');//check for condition

printf("\nprogram terminated .... \n");
```

```
press any key , q to exit : o
your key is o
press any key , q to exit : g
your key is g
press any key , q to exit : q
your key is q
program terminated ...
```

[04 - programControl\13.c]

لكتابة نفس البرنامج بـ while او for سنحتاج اي كتابة الاكواد داخل do while مرة اخرة قبل ال while او for لضمان التنفيذ مرة واحدة علي الاقل

```
printf("\npress any key , q to exit : ");
fflush(stdin);
scanf("%c",&ch);
printf("\nyour key is %c\n",ch);
while(ch != 'q')
{
    printf("\npress any key , q to exit : ");
    fflush(stdin);
    scanf("%c",&ch);
    printf("\nyour key is %c\n",ch);
}
```

لذا تستخدم حلقة do while في الحالات التي نحتاج فيها ان يتم تنفيذ التعليمات داخلها مرة واحدة علي الاقل

- التكرار بواسطة goto:

لاستعمل كثيرا , تستخدم للتنقل داخل الكود من مكان لآخر باستخدام العلامات labels

مثال برنامج علي طريقة استخدامها

```
//jump to second label
goto second;
printf("first");
//define second label
second :
printf("second");
```

second

[04 - programControl\14.c]

لن يتم تنقيذ السطر التالي , لان goto ستنقل التنفيذ الي second

```
printf("first");
```

قمنا تعريف العلامة label واعطيناه اسم second ثم :

لكي تستخدم لعمل التكرار لابد من استخدام جملة الشرط if else لكي نضيف التحقق من شرط معين

مثال تعديل علي البرنامج الخاص بـ do while

```
//to store key from user
char ch;
again:
    //ask user to enter key
    printf("\npress any key , q to exit : ");
    //empty input buffer
    fflush(stdin);
    //get key form user
    scanf("%c",&ch);
    //print user key
    printf("\nyour key is %c\n",ch);
    if (ch != 'q')//check for condition
    {
        goto again;
    }
    printf("\nprogram terminated .... \n");
```

```
press any key , q to exit : r
your key is r
press any key , q to exit : q
your key is q
program terminated...
```

[04 - programControl\15.c]

قمنا بتعريف علامة again label , واستخدمنا جملة الشرط if للتحقق من الشرط الا يكون الحرف يساوي q والانتقال مرة اخري الي again

-لا يفضل استعمال goto علي الاطلاق لاسباب

- ان هناك بديلا عنها افضل واكثر وضوحا
- تجعل من الصعب تتبع الكود سواء لفهمه او لاصلاح او تتبع خطأ ما debugging

- تعديل الحلقات باستخدام **break & continue**:

تستخدم كلمة break للخروج من الحلقة عن قصد استخدمناها سابقا للخروج من switch

مثال برنامج يقوم باختيار رقم من 1 الي 10 ويعطي المستخدم 4 محاولات لتخمين الرقم

```
//selected number
int number = 6;
//user guess
int guess;
//trials counter
int trials;

//loop for 4 times
for(trials = 1 ; trials <= 4 ; trials++)
{
    //ask user to guess number
    printf("guess number between 1 ,10 : ");
    //get user guess
    scanf("%d",&guess);

    //check if guess is right
    if(guess == number)
    {
        //exit loop if guess is right
        break;
    }
    //inform user that guess was wrong
    printf("wrong guess,try again \n");
```

```

}

//check if user guess number correctly or not
if(guess == number)
{
    printf("\nExcellent\n");
}
else
{
    printf("\nnumber is %d\n",number);
}

```

```

guess number between 1 ,10 : 1
wrong guess,try again
guess number between 1 ,10 : 2
wrong guess,try again
guess number between 1 ,10 : 6

Excellent

```

[04 - programControl\16.c]

يتم الخروج من الحلقة لاحد سببين الاول ان تنتهي عدد المحاولات للمستخدم , الثاني ان يخمن المستخدم الرقم الصحيح ويتم الخروج في تلك الحالة باستخدام break;

```

if(guess == number)
{
    //exit loop if guess is right
    break;
}

```

- اما كلمة continue تؤدي للانتقال الي اول الحلقة مرة اخري وتخطي اي تعليمات بعدها وتستخدم فقط مع الحلقات التكرارية, الانتقال الي اول الحلقة يعني بدء التكرار الذي يلي الحالي next loop iteration , اي ان الشرط يتم التحقق منه اولاً ايضاً وهكذا

```
//loop counter
int counter;
//loop from 1 to 10 with step =1
for(counter =1 ;counter <= 10 ; counter++)
{
    //if number is odd don't print it
    if(counter % 2 != 0)
    {
        //go to loop start and skip the rest
        continue;
    }
    //print number
    printf("%d ",counter);
}
```

2 4 6 8 10

[04 - programControl\17.c]

استخدمنا for للمرور علي جميع الارقام من 1 الي 10 , والتحقق من كل رقم اذا كان فردي (لايقبل القسمة علي 2 , هناك باقي قسمة) اذهب الي اول الحلقة مرة اخري ولاتكمل تنفيذ التعليمات المتبقية , اذا كان زوجي سيتم طباعته

```
//if number is odd don't print it
if(counter % 2 != 0)
{
    //go to loop start and skip the rest
    continue;
}
```

- اخيرا يمكننا ان نستخدم حلقة تكرارية داخل اخري nested loops


```
//rectangle dimension
int height = 6,width =12;

//loops counters
int i,j;

//outer loop for rectangle height
for(i=0;i<height;i++)
{
    //inner loop for width
    for(j=0;j<width;j++)
    {
        printf("*");
    }
    printf("\n");
}
```

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

[04 - programControl\18.c]

هنا الحلقة الخارجية مسئولة عن ارتفاع المستطيل (عدد السطور) اما الحلقة الداخلية مسئولة عن ملء كل سطر بعدد من النجوم (العرض) , ان لكل تكرار من الحلقة الخارجية تبدا الحلقة الداخلية من جديد حتي النهاية وهكذا

- الشرط باستخدام **?: conditional expression** :

عبارة عن جملة الشرط if-else في سطر واحد

مثال برنامج يقوم بطباعة الرقم الأكبر بين رقمين مرة باستخدام :? واخري باستخدام جملة الشرط if-else

```
//numbers
```

```
int first = 23 , second = 50;
```

```
//conditional expression
```

```
int bigger = (first > second) ? first : second ;
```

```
printf("the bigger number is %d \n", bigger);
```

```
int first = 23 , second = 50;
```

```
if( first > second )
```

```
{
```

```
    bigger = first;
```

```
}
```

```
else
```

```
{
```

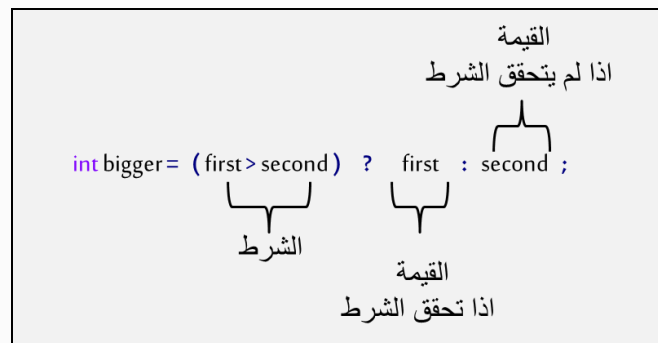
```
    bigger = second;
```

```
}
```

```
printf("the bigger number is %d \n", bigger);
```

the bigger number is 50

قيمة ال expression تتوقف علي الشرط فاذا كان $first > second$ تصبح قيمة الشرط first والا تصبح second



الخلاصة

- تستخدم جملة if لتنفيذ التعليمات كلا حسب شرط معين
- نستقبل قيم من المستخدم باستخدام الدالة scanf
- تستخدم switch عندما نريد مقارنة قيمة متغيرة مع قيم ثابتة
- لتنفيذ مجموعة تعليمات عدة مرات وفقا لشرط معين نستخدم الحلقات التكرارية
- في الحلقة التكرارية do-while يتم تنفيذ التعليمات اولاً ثم التحقق من الشرط بعكس الحلقات الاخرى
- تستخدم الكلمة break للخروج من الحلقة التكرارية او من switch

- ماذا يعني الشرط التالي بالنسبة لجمله الشرط if-else ؟

```
if( 1 )
{
    printf("inside true");
}
else
{
    printf("inside false");
}
```

تعني ان الشرط صحيح دائما وذلك لان السي تعتبر الصفر false واي قيمة اخري non-zero تعتبرها true

- تكرار نفس الشرط في جملة else-if ؟

```
if( x > 10 )
{
    printf("x i bigger than 10");
}
else if( x > 10 )
{
    printf("x is positive number");
}
else
{
    printf("x i less than 10");
}
```

سيتم تنفيذ التعليمات التي تلي اول شرط صحيح فقط اي السطر

```
printf("x i bigger than 10");
```

- اعطاء الدالة scanf عدد متغيرات اقل مما هو متوقع

```
scanf("%d %d",&x);
```

سيعمل البرنامج , وسيقوم باستقبال الرقمين من المستخدم , ولكننا لن نعرف الا الرقم الاول فقط الذي سيتم تخزينه في x

- اعطاء الدالة scanf عدد اكبر مما هو متوقع

```
scanf("%d",&x,&y);
```

سيظهر تحذير بان عدد الوسائط كبير ولكن هذا لن يوقف تشغيل البرنامج

- ما هي الحلقة الغير منتهية بالامثلة ؟

هي الحلقة التي يكون فيها الشرط صحيح دائما وذلك اما لان الشرط لاشي يتغير فيه او لان الشرط قيمة ثابتة رقم غير الصفر مثلا, ولا يتم الخروج منها ابدا , لا تحتوي علي كلمة break او return تتسبب في الخروج منها

<pre>while(1) { //statements }</pre>	<pre>for(;;) { //statements }</pre>	<pre>for(1;2;3) { //statements }</pre>
--	---	--

- هل يصح استخدام جملة switch مع متغير من نوع float

```
float num=1.0;
switch(num)
{
    case 2.0:
    case 3.0:
    case 1.0:
}
```

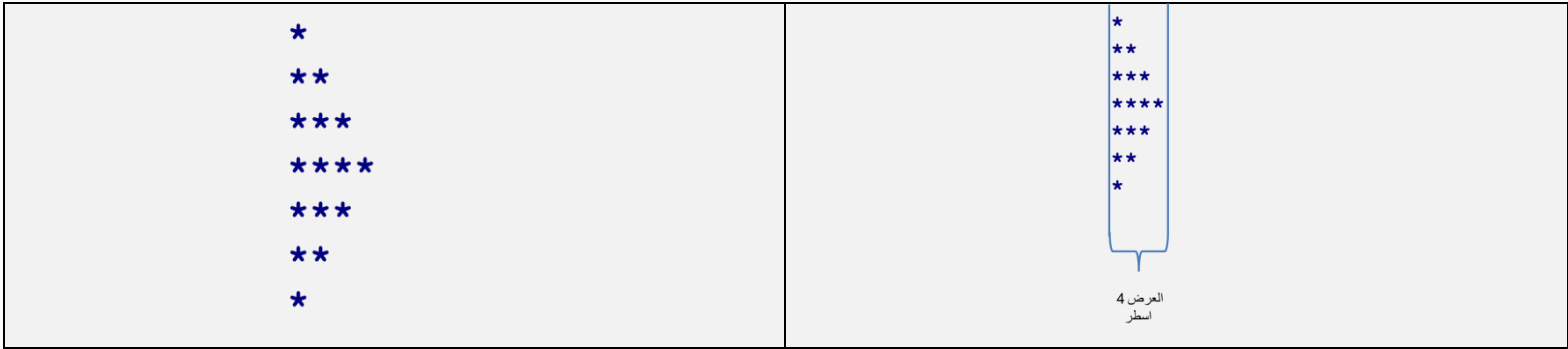
لن يعمل البرنامج وسيظهر خطأ يفيد بان الجملة switch تتعامل مع الاعداد الصحيحة فقط



Practical Examples

امثلة عملية

1- اكتب برنامج يقوم برسم الشكل الاتي علي ان يقوم المستخدم بتحديد العرض بالاسطر

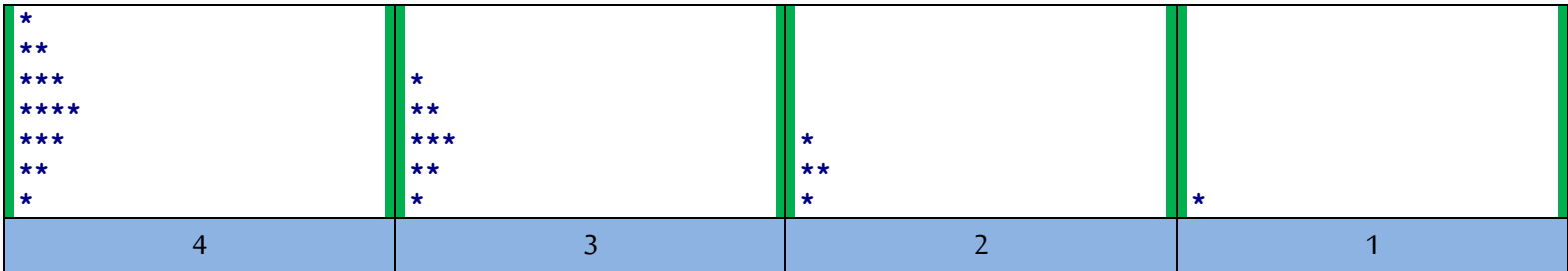


التفكير في الحل :

لو طلب منا ان نقوم برسم هذا الشكل تحديدا سيكون الحل بسيط جدا حيث سنستخدم دالة الطباعة printf عدة مرات لطباعة الشكل وسيكون الكود كالآتي

```
printf("*\n");
printf("**\n");
printf("***\n");
printf("****\n");
printf("****\n");
printf("****\n");
printf("****\n");
printf("****\n");
```

ولكن الشكل هنا متغير في العرض حسب القيمة التي يريدها المستخدم , سنحتاج في البداية ان نحدد كيف سيتغير الشكل حسب قيمة العرض لذا سنقوم بافتراض عدة قيم للعرض ونرسمها كالآتي



وسنجد ان الشكل عبارة عن مجموعة من الاسطر الافقية التي بالتاكيد لها علاقة بالعرض , وكل سطر افقي به عدد من الرموز * والتي ايضا لها علاقة بشكل او باخر بالعرض , بالنظر الي الاشكال بالاعلي سنجد الاتي

- الاسطر الافقية عددها يساوي $(2*w - 1)$ حيث تمثل w العرض

العرض	عدد الاسطر الافقية
1	$1 = (2*1 - 1)$
2	$3 = (2*2 - 1)$
3	$5 = (2*3 - 1)$
4	$7 = (2*4 - 1)$

- عدد * في كل سطر يساوي ترتيب السطراته حتي يصل ترتيب السطور الي قيمة العرض , يبدأ الترتيب في التناقص حتي يصل الي 1



الحل

يوجد اكثر من حل بناء علي الملاحظات بالاعلي , وسنقوم هنا برسم الشكل علي مرحلتين الاولى رسم الشكل حتي يصبح عدد السطور مساوي لقيمة العرض , المرحلة الثانية رسم باقي الشكل بالتناقص من قيمة العرض حتي 1 , سنستخدم الحلقات التكرارية سواء لرسم السطور او لرسم ال * في كل سطر ليصبح الكود كالآتي

```
//shape width
int width ;
//loop variable
int i ,j;
//ask user to type width
printf("please type shape width : \t");
//get shape width
scanf("%d",&width);
//draw first part
for(i = 1 ;i<=width ; i++)
{
    j=i;//number of stars in each line
    while(j-->0)
    {
        printf("*");
    }
    printf("\n");
}
//draw second part
for(i = width-1 ; i>0 ; i--)
{
    j=i;
    while(j-->0)
    {
        printf("*");
    }
    printf("\n");
}
```

please type shape width : 3

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

please type shape width : 2

```
*
**
*
```

please type shape width : 1

```
*
```

[05 - practical examples\draw.c]

2- اكتب برنامج يقوم بحساب مضروب factorial اي قيمة من المستخدم

التفكير في الحل :

من المعلوم ان مضروب اي رقم هو حاصل ضرب القيم بداية من الرقم وحتى 1 فمثلا

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

$$3! = 3 \times 2 \times 1 = 6$$

العلامة ! تستخدم للتعبير عن المضروب , ايضا مضروب 1 او 0 يساوي 1 , سنجد ان عدد مرات الضرب مساوي للرقم , ان القيم المستخدمة في عملية الضرب تتناقص بمقدار 1 في كل مرة

الحل

سنستخدم الحلقات التكرارية في الحل كالآتي

```
//number
unsigned int num;
//factorial value
unsigned int fact = 1;
//loop counter
int i;

//ask user to enter number
printf("please type number :\t");
//get the number
scanf("%d",&num);

for(i = num ; i > 0 ; i--)
{
    //check if value is 0 or 1
    if(num == 1 || num == 0)
    {
        fact = 1;
        break;
    }
    //multiply by current i
    fact *= i ;
}

printf("factorial of %d is %d\n",num,fact);
```

please type number : 1

factorial of 1 is 1

please type number : 4

factorial of 4 is 24

please type number : 8

factorial of 8 is 40320

[05 - practical examples\factorial.c]

3- اكتب برنامج يقوم بطباعة جدول الضرب لاي قيمة من المستخدم كالمثال التالي جدول الـ 5

```
5 * 1 = 1
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
```



```
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

الحل

سنستخدم الحلقات التكرارية في طباعة الجدول كالآتي

```
//number
unsigned int num;
//loop counter
int i;

//ask user to enter number
printf("please type number :\t");
//get the number
scanf("%d",&num);

for(i = 1 ; i <= 10 ; i++)
{
    printf("%d * %2d = %d \n",num , i , num*i);
}
```

please type number: 5

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

[05 - practical examples\multiplication.c]

4- اكتب برنامج يقوم بطباعة تسلسل من رقمين لعدد من المرات مثال الارقام 5 و 11 وطول التسلسل 8

```
5 11 5 11 5 11 5 11
```

التفكير في الحل :

يمكن طباعة هذا التسلسل باكثر من طريقة كالآتي مثلا

```

//first number
int first = 11;
//second number
int second = 5;
//sequence length
int length = 8;
//int loop counter
int i;
//last value
int last = first ;
for(i =0 ; i<length ; i++)
{
    if(last == first)
    {
        last = second;
    }
    else
    {
        last = first ;
    }
    printf("%d ",last);
}

```

لكن هناك طريقة اخري باستخدام مؤثر ^ فاحد مميزات هذا المؤثر ان ناتج $A \wedge B$ يعطي C مثلا , اذا قمنا بعمل \wedge لـ C مع ايا من A او B

يعطي الآخر

```

C = A ^ B ;
C ^ A = B ;
C ^ B = A ;

```

وبالتالي سيكون الحل كما يلي

```

//first number
int first = 11;
//second number
int second = 5;
//sequence length
int length = 8;
//int loop counter
int i;
//last value
int last = second ;
//flipper
int flip = first ^ last;

for(i =0 ; i<length ; i++)
{
    last ^=flip ;
    printf("%d ",last);
}

```

11 5 11 5 11 5 11 5

[05 - practical examples\sequence.c]

5- اكتب برنامج يقوم بايجاد عدد 1 في اي رقم موجب ما اذا ما تم تحويله للنظام الثنائي

امثلة

الرقم	عدد 1	ملاحظات
2	1	$10 = 2$ بالنظام الثنائي
3	2	$11 = 3$ بالنظام الثنائي
7	3	$111 = 7$ بالنظام الثنائي

التفكير في الحل :

يمكن الحل باحدي طريقتين اما نستخدم الطريقة المتبعة لتحويل الارقام الموجبة الي النظام الثنائي وذلك بالقسمة علي 2 حتي يصل الرقم الي 0 , او من خلال التحقق من كل بت في الرقم من خلال استخدام bitwise وذلك ان الارقام تخزن بالنظام الثنائي

مثال علي الرقم 10 حيث عدد 1 يساوي 2

<div><div>الباقي /2</div><div><div>10</div><div>5</div><div>2</div><div>1</div><div>0</div></div><div><div>0</div><div>1</div><div>0</div><div>1</div><div></div></div></div>	<div><div>الرقم في الذاكرة</div><div><div>0</div><div>0</div><div>0</div><div>0</div><div>1</div><div>0</div><div>1</div><div>0</div></div></div> <div><div>باستخدام & للتحقق من قيمة كل بت</div><div>هل 1 او 0</div></div>
تحويل 10 الي النظام الثنائي	الرقم 10 كما في الذاكرة

الحل

سنستخدم طريقة القسمة علي 2 لايجاد عدد 1 كالآتي

```
//number
unsigned int num ;
//ones counter
int count=0;
//ask user to type number
printf("please type number : \t");
//get number
scanf("%d",&num);
//loop until num =0
while(num != 0)
{
    //check if remainder of divide by 2 = 1
    if(num % 2 == 1)
    {
        count++;
    }

    //divide number by 2
    num /=2;
}
printf("number of ones is %d \n",count);
```

please type number : 7 number of ones is 3	please type number : 10 number of ones is 2	please type number : 127 number of ones is 7
[05 - practical examples\count_ones.c]		

6- اكتب برنامج يقوم بطباعة جدول اسكي ASCII Table

لكل مفتاح في لوحة المفاتيح تمثيل بالنظام الثنائي , وتلك المفاتيح منها ما هو حروف وارقام ومنها ما يخص التحكم وهي في مجملها 128

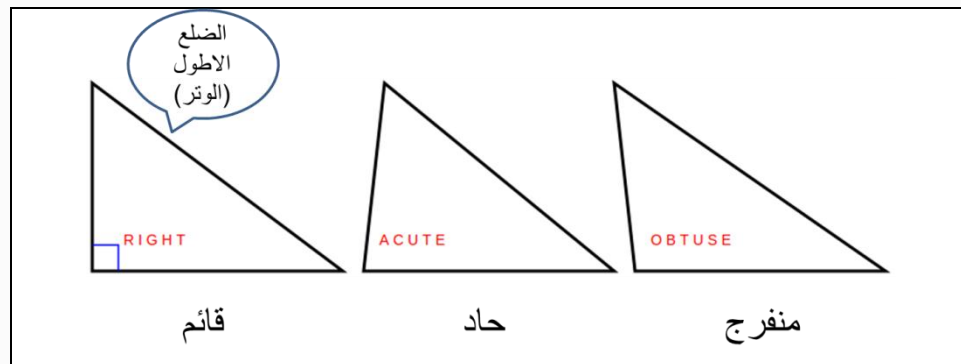
الحل

سنستخدم الحلقة التكرارية لطباعة الجدول

<pre>//counter int count = 0 ; printf("Dec\tHex\tchar"); for(count=0 ;count < 128 ;count++) { printf("%d\t%x\t%c\n",count,count,count); }</pre>
[05 - practical examples\ascii.c]

7- اكتب برنامج يقوم بتحديد اذا كان المثلث قائم او حاد او منفرج وذلك بمعرفة اطوال اضلاعه الثلاثة حيث تعطي الاطوال من المستخدم

حيث الاول والثاني جانبي الزاوية والثالث هو الضلع الاطول (الوتر) مثل الشكل التالي



التفكير في الحل :

الحل هنا يعتمد بشكل كبير علي معرفتنا بالرياضيات اكثر من البرمجة ,الحل هنا قائم علي نظرية فيثاغورث والتي تنص علي ان

$c^2 = a^2 + b^2$ $c = \sqrt{a^2 + b^2}$
--

حيث c هو الوتر و a و b جانبي الزاوية القائمة

هذا للمثلث القائم اما

- المثلث الحاد يكون اطول ضلع فيه اقل من طول الوتر اذا اعتبرنا هذا المثلث قائم
- المثلث المنفرج يكون اطول ضلع فيه اكبر من طول الوتر اذا اعتبرنا هذا المثلث قائم

اي اننا سنعتبر اي مثلث انه قائم بشكل افتراضي ونقوم بحساب طول الوتر , ونقارنه بالضلع الاطول لكل مثلث والذي سيكون الطول الثالث من المستخدم للتبسيط , وبناء علي المقارنة سنحدد نوع المثلث

الحل

سنستخدم هنا الدالة sqrt لايجاد الجذر وهي احد دوال الملف math.h الذي يحوي الدوال الخاصة بالعمليات الرياضية

```
//triangle dimensions
int a , b ,h ;
//calculated hypotenuse
int hCalc;
//ask user to type triangle dimensions
printf("please type triangle dimensions as a b h: ");
scanf("%d %d %d",&a,&b,&h);

//calculate h
hCalc= sqrt(a*a + b*b );

if( h == hCalc )
{
    printf("triangle is right\n");
}
else if (h > hCalc)
{
    printf("triangle is obtuse\n");
}
else
{
    printf("triangle is acute\n");
}
```

please type triangle dimensions as a b h: 3 4 5
triangle is right

please type triangle dimensions as a b h: 6 8 9
triangle is acute

[05 - practical examples\triangle.c]

8- اكتب برنامج يقوم بطباعة عدد الخانات في رقم معين , امثلة

```
23 --> 2
6 --> 1
4521 --> 4
76234 --> 5
```

التفكير في الحل :

قد تكون اول فكرة تاتي لنا ان نقارن الرقم بـ 10 , 100 , 1000 , 10000 , الخ فاذا كان الرقم اقل من اي من تلك القيم يمكننا تحديد عدد الخانات

تلك الطريقة علي بساطتها الي انها تحتاج الي جملة else-if كثيرا لكل مقارنة , بالتالي يصبح الكود اطول كالآتي

```

int digits;
int num = 22 ;
if(num < 10 )
{
    digits = 1;
}
else if (num < 100 && num >=10)
{
    digits = 2 ;
}
.
.
.

```

لذا نحتاج الي طريقة اخري اسهل وافضل , لمعرفة عدد الخانات في رقم نحتاج الي طريقة لاستخراج كل خانة من الرقم حتي يصبح الرقم يساوي 0

لذا سنجد انه بالقسمة علي 10 يمكننا تقليل عدد الخانات بقدر 1 كما بالمثال

```

234 / 10 = 23
23 / 10 = 2
2 / 10 = 0

```

الحل

سنستخدم القسمة وباقي القسمة

```

//number
unsigned int num ;
//ones counter
int count=0;
//ask user to type number
printf("please type number : \t");
//get number
scanf("%d",&num);

//loop until num =0
while(num != 0)
{
    count++;
    //divide number by 10
    num /=10;
}
printf("number of digits is %d \n",count);

```

please type number: 12

number of digits is 2

please type number: 2341

number of digits is 4

please type number: 2

number of digits is 1

[05 - practical examples\count_digits.c]

9- اكتب برنامج يقوم بحساب مجموع الخانات في اي رقم , امثلة

```

23 --> 2 + 3 = 5
6 --> 6
4521 --> 4 + 5 + 2 + 1 = 12

```

التفكير في الحل :

البرنامج هنا شبيه بالبرنامج السابق , ولكننا هنا سنحتاج الي استخراج القيم الخاصة بكل خانة يتم ذلك بايجاد باقي القسمة علي 10 حيث يمكننا استخراج اول خانة من الرقم بالاضافة الي استخدام القسمة لتقليل عدد الخانات بقدار 1 كالاتي

234 / 10 = 23	234 % 10 = 4
23 / 10 = 2	23 % 10 = 3
2 / 10 = 0	2 % 10 = 2

الحل

```
//number
unsigned int num ;
//summation
int sum=0;
//ask user to type number
printf("please type number : \t");
//get number
scanf ("%d",&num) ;

//loop until num =0
while(num != 0)
{
    sum += (num % 10 ) ;
    //divide number by 10
    num /=10;
}
printf("sum of digits is %d \n",sum) ;
```

please type number : 12

sum of digits is 3

please type number : 4521

sum of digits is 12

please type number : 987123

sum of digits is 30

[05 - practical examples\sum_digits.c]

10- اكتب برنامج يقوم بعكس الخانات الخاصة باي رقم , امثلة

123 --> 321
4587 --> 7854
34 --> 43

التفكير في الحل :

تعلمنا من البرنامجين السابقين كيف يمكن تقليل الخانات واستخراج الخانة الاولى وذلك باستخدام القسمة وباقي القسمة , ولكننا هنا ايضا نحتاج الي تحريك الخانات الي اليسار وازداحة الخانة الجديدة وذلك في الرقم الجديد لذا سنستخدم الضرب لتحريك الخانات الي اليسار والجمع لاضافة الخانة الجديدة

```

//number
unsigned int num ;
//summation
int newNum=0;
//shift value
int shift =1 ;
//ask user to type number
printf("please type number : \t");
//get number
scanf ("%d",&num);

//loop until num =0
while(num != 0)
{
//shift to left by one digit
newNum *= shift;
//add new digit
newNum += (num % 10 ) ;
//increase shift
shift =10;
//divide number by 10
num /=10;
}
printf("reverse of digits is %d \n",newNum);

```

please type number: 1234

reverse of digits is 4321

please type number: 23789

reverse of digits is 98732

please type number: 892

reverse of digits is 298

[05 - practical examples\reverse_digits.c]



Arrays

المصفوفات

لفهم المصفوفات وللمعرفة اهميتها دعونا نفترض انه طلب منا ان نكتب برنامج ياخذ من المستخدم خمسة ارقام وطباعتهم بنفس الترتيب , وفقا لما تعلمناه حتي الان سنفكر كالاتي :

- الاعلان عن خمسة متغيرات لتخزين القيم بها
- نستقبل القيم من المستخدم بشكل متتالي باستخدام الدالة scanf
- نعرض القيم بنفس الترتيب المتغيرات باستخدام الدالة printf

ليكون لدينا البرنامج التالي

```
//user input variables
int num1,num2,num3,num4,num5;
//ask user to enter value
printf("enter value #1 : ");
//get the value
scanf("%d",&num1);
//ask user to enter value
printf("enter value #2 : ");
//get the value
scanf("%d",&num2);
//ask user to enter value
printf("enter value #3 : ");
//get the value
scanf("%d",&num3);
//ask user to enter value
printf("enter value #4 : ");
//get the value
scanf("%d",&num4);
//ask user to enter value
printf("enter value #5 : ");
//get the value
scanf("%d",&num5);
//print values
printf("values as received \n");
printf("%d\n%d\n%d\n%d\n%d\n",num1,num2,num3,num4,num5);
```

```

enter value #1 : 11
enter value #2 : 22
enter value #3 : 33
enter value #4 : 44
enter value #5 : 55
values as received
11
22
33
44
55

```

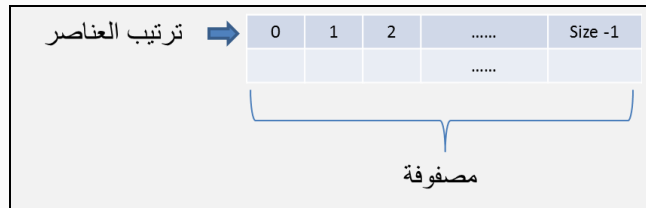
[06 - arrays\01.c]

هذا قد يكون مقبولا نوعا ما , ماذا لو طلب منا ان نستقبل 10 او 20 قيمة من المستخدم سنحتاج الي الاتي

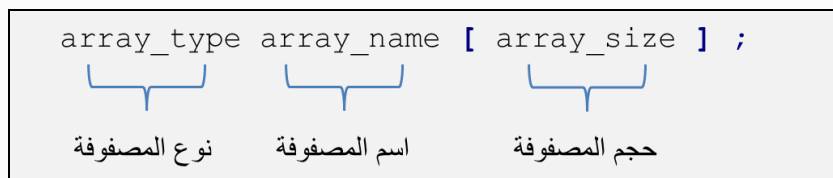
1. حجز متغير لكل قيمة
2. زيادة اسطر استقبال القيم
3. زيادة في اسطر طباعة القيم

بالتالي فان عدد اسطر البرنامج سيكون كبير , ونحن نتحدث في النهاية عن مهمة بسيطة جدا بالمقارنة بالبرامج الفعلية ومن هنا جاءت اهمية المصفوفات

- المصفوفة : مجموعة من الاماكن المتتالية في الذاكرة , تستخدم لتخزين بيانات من نفس النوع , وقد تكون من نوع int او float او char ويعتبر كل مكان من تلك الاماكن متغير يتم التعامل معه مثل المتغيرات , يتم الاشارة لكل عنصر داخل المصفوفة من خلال اسم المصفوفة متبوعا برقم يسمي index يبدأ من الصفر ويعبر عن ترتيب العنصر داخل المصفوفة



- الاعلان عن مصفوفة



```
int nums[20]; //array of 20 intergers
float results[10]; //array of 10 float
char sequence[5]; //array of 5 characters
```

مثال البرنامج السابق باستخدام المصفوفات

```
//user inputs array
int nums[5];

// loop counter
int counter ;

//loop to get values
for(counter =0 ; counter <5 ; counter++)
{
    //ask user to enter value
    printf("enter value # %d : ", counter+1);

    //get value
    scanf("%d",&nums[counter]);
}

//print values
printf("values as received \n");

//loop to print values
for(counter =0 ; counter <5 ; counter++)
{
    printf("%d \n",nums[counter]);
}
```

```
enter value #1: 77
enter value #2: 88
enter value #3: 99
enter value #4: 111
enter value #5: 222
values as received
77
```

88
99
111
222

[06 - arrays\02.c]

قمنا بحجز مصفوفة للقيم بها 5 اماكن من نوع int

```
int nums[5];
```

يتم الاشارة الي اي عنصر بالمصفوفة من خلال اسم المصفوفة متبوعا بـ [] بداخله ترتيب العنصر داخل المصفوفة index , وهذا الفهرس للمصفوفة يبدأ من الصفر وحتى عدد اماكنها مطروحا منه واحد وبالتالي فالمصفوفة التي لدينا عناصرها هي num[0] , num[1], num[2], num[3], num[4] , لذا استخدمنا الحلقة لاستقبال القيم لان الذي يتغير من عنصر لآخر هو رقم الفهرس فقط , وبعد ذلك استخدمنا الحلقة مرة اخري لطباعة القيم من المصفوفة , الملاحظ هنا ان عدد اسطر البرنامج اقل بكثير من السابق , كما انه لن يتغير اذا زاد عدد القيم

مثال برنامج يقوم بحساب متوسط القيم الموجودة بالمصفوفة وطباعته

```
//user inputs array
int nums[5]={22,55,77,88,99};

//average
float average=0.0;

// loop counter
int counter ;

//loop to add all array values
for(counter =0 ; counter <5 ; counter++)
{
    //add each element to average value
    average += nums[counter];
}

average /=5.0;

//print values
printf("average is %.2f\n",average);
```

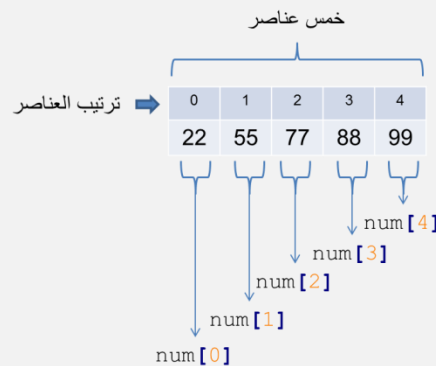
average is 68.20

[06 - arrays\03.c]

اعلنا عن مصفوفة بها 5 عناصر من نوع int واعطيناها قيم مباشرة , يتم اعطاء عناصر المصفوفة قيم باحد طريقتين اما اعطاء قيم لكل العناصر مرة واحد من خلال استخدام الاقواس {} وبين كل قيم واخري فاصلة , او اعطاء كل عنصر القيمة بشكل منفصل , كما هو موضح

```
int nums[5]={22,55,77,88,99};
```

```
int nums[5];  
nums[0]=22;  
nums[1]=55;  
nums[2]=77;  
nums[3]=88;  
nums[4]=99;
```



استخدمنا الحلقة التكرارية لتجميع عناصر المصفوفة , استخدمنا الرمز += للاجراء الجمع وقمنا بعد ذلك بقسمة مجموع العناصر علي عددهم ليعطي المتوسط

مثال برنامج يقوم بالبحث داخل مصفوفة عن قيمة معينة واذا وجدها يطبع ترتيبها داخل المصفوفة (هذا النوع من البحث يسمى بحث خطي linear search)

```
//user inputs array  
int nums[5]={34,25,76,98,12};  
//value to search for  
int value=98;  
//value index  
int key=-1;  
// loop counter  
int counter;
```

```

//loop until find the value
for(counter =0 ; counter <5 ; counter++)
{
    //check if value match
    if(nums[counter] == value)
    {
        //assign value index to key
        key = counter ;
        //exit loop
        break;
    }
}

//check if value was found or not
if(key != -1)
{
    printf("value %d located in array at index %d \n",value,key);
}
else
{
    printf("unknown value\n");
}

```

value 98 located in array at index 3

[06 - arrays\04.c]

قمنا بالاعلان عن مصفوفة للقيم واعطيناها قيم مباشرة , حجزنا متغير key لتخزين ترتيب القيمة في المصفوفة واعطيناه قيمة مبدئية 1- للمقارنة بعد ذلك ولان الترتيب لا يمكن ان يكون سالب , استخدمنا الحلقة التكرارية للمرور traverse علي كل عناصر المصفوفة واذا حدث تطابق match بين القيمة والعنصر يتم حفظ ترتيب العنصر والخروج من الحلقة , يتم التحقق هل تم ايجاد القيمة ام لا من خلال مقارنة قيمة المتغير key بالقيمة المبدئية التي اعطيت له , اذا لم تساويها هذا يعني انه تم تغييرها داخل الحلقة اي ان القيمة تم ايجادها

```
//user inputs array
int nums[8]={34,25,76,98,12,100,56,14};

// max values
int max;

// loop counter
int counter ;

//assume first element in array is max
max = nums[0];

//loop and check for larger
for(counter =0 ; counter <8 ; counter++)
{
    //check if a value is larger
    if(nums[counter] > max )
    {
        //replace larger value
        max = nums[counter] ;
    }
}

printf("Maximum value in the array is %d \n",max);
```

Maximum value in the array is 100

[06 - arrays\05.c]

بعد الاعلان عن المصفوفة واعطائها قيم مباشرة , حجزنا متغير max لتخزين اكبرقيمة بالمصفوفة , وافترضنا ان اول عنصر بالمصفوفة هو اكبرعنصر , لانه من المفترض اننا نتعامل مع مصفوفة لانعلم محتوياتها , قمنا بعد ذلك بالمرور علي كل عناصرالمصفوفة اذا وجد عنصر اكبر من الموجود بالمتغير max يتم استبداله به وهكذا حتي نهاية المصفوفة

- ابعاد المصفوفة

- مصفوفة ثنائية الابعاد

تتكون تلك المصفوفة من صفوف واعمدة وهي تشبه الجدول ,هي مثل المصفوفة ذات البعد الواحد في التعامل من حيث اعطاء القيم او قرائتها يتم انشاءها كما بالشكل

array_type array_name [rows] [cols] ;

نوع المصفوفة اسم المصفوفة عدد الصفوف عدد الاعمدة

array_type array_name [3] [3] ;

	0	1	2	3	
0					الصفوف
1					
2					
3					
الاعمدة					

مثال برنامج يقوم بطباعة محتويات مصفوفة ثنائية

//create array, and assign it's value

```
char shape[5][5]={
    { ' ', ' ', '*', ' ', ' ' },
    { ' ', '*', ' ', '*', ' ' },
    { '*', ' ', '*', ' ', '*' },
    { ' ', '*', ' ', '*', ' ' },
    { ' ', ' ', '*', ' ', ' ' },

};
```

//rows and columns counters

```
int rows ,cols;
```

//loop on rows

```
for(rows =0 ; rows <5 ; rows++)
```

```
{
```

//loop on columns

```
for(cols=0 ; cols<5 ; cols++)
```

```
{
```

```
printf("%c ",shape[rows][cols]);
```

```
}
```

```
printf("\n");
```

```
}
```

```

*
* * *
* * * * *
* * *
*

```

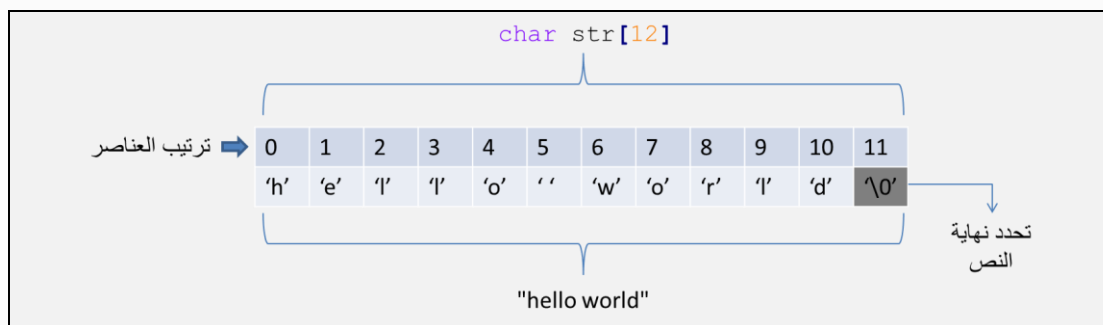
[06 - arrays\06.c]

انشانا مصفوفة حروف من 5 صفوف و 5 اعمدة واعطيناها قيم مباشرة ليصبح شكل المصفوفة مثل الجدول الموضح , استخدمنا nested for للمرور علي عناصر المصفوفة , الحلقة الخارجية للتنقل من صف لآخر والحلقة الداخلية للتنقل من عمود لآخر ثم طباعة محتوى العنصر

	0	1	2	3	4
0			*		
1		*	*	*	
2	*	*	*	*	*
3		*	*	*	
4			*		

- النصوص strings:

لا يوجد في السي نوع اساسي لتخزين النصوص , لذا يتم التعامل مع النص علي انه مصفوفة من الحروف ذات بعد واحد ولذا تسمي سلسلة حروف , ولا بد ان يكون العنصر الاخير في المصفوفة به القيمة (\0) والتي تسمي بـ Null character وتكمن اهميتها انها تحدد نهاية النص داخل المصفوفة السي تستخدم " " للنصوص و ' ' للحروف



مثال طباعة hello world باستخدام النص string

```

//string
char str [13] = {"hello world"};

//print string using %s
printf("%s \n",str);

```

hello world

[06 - arrays\07.c]

انشانا مصفوفة من الحروف واعطيناها قيمة مباشرة وهي النص المراد طباعته , هذه الطريقة تقوم تلقائيا باضافة (\0) في اخر السلسلة, يمكن استخدام الطريقة الاخرى عن طريق اعطاء كل عنصر في السلسلة قيمته بشكل منفصل

```
char str [12];  
str[0] = 'h'; str[1] = 'e';  
str[2] = 'l'; str[3] = 'l';  
str[4] = 'o'; str[5] = ' ';  
str[6] = 'w'; str[7] = 'o';  
str[8] = 'r'; str[9] = 'l';  
str[10] = 'd'; str[11] = '\0';
```

يستخدم الرمز %s لطباعة النصوص في الدالة printf , وتأخذ تلك الدالة اسم مصفوفة الحروف كوسيط في تلك الحالة
مثال برنامج لاستقبال الاسم الاول للمستخدم وطباعته مع عدد حروف الاسم

```
//string  
char str [100];  
  
//chars counters  
int count =0;  
  
//ask user to type his first name  
printf("Your first name : ");  
  
//get user name  
scanf("%s",str);  
  
//count characters
```

```
while(str[count] != '\0')
{
    count++;
}

printf("%s contain %d characters \n",str,count);
```

Your first name : mohamed
mohamed contain 7 characters

[06 - arrays\08.c]

حجزنا مصفوفة حروف حجمها 100 علي افتراض ان اسم الاول للمستخدم لن يتجاوز 99 حرف , وقمنا باستقبال الاسم من المستخدم بالدالة scanf واستخدمنا الرمز %s مع تلك الدالة للدلالة علي ان القيمة نص واعطيناه اسم مصفوفة الحروف كوسيط ثاني , لعد حروف الاسم قمنا باستخدام الحلقة while بشرط الا يكون العنصر مساوي (\0) لانه يعبر عن نهاية السلسلة , وان وجدت تلك القيمة يكون ترتيبها هو عدد حروف الاسم لانه ليس شرطا ان ياخذ الاسم المصفوفة باكملها , يوجد دالة خاصة بالسي لمعرفة طول النص سنراها في البرنامج التالي , وهي ايضا داخليا تستخدم نفس الفكرة

مثال برنامج ياخذ كلمة من المستخدم ويقوم بطباعته معكوسا

```
//string
char str [100];

//chars counters
int count;

//ask user to type his first name
printf("Your first name : ");

//get user name
scanf("%s",str);

//use string length function to get string length
count = strlen(str);
```

```
//print string in reverse order
```

```
while(count--)
```

```
{
```

```
    printf("%c",str[count]);
```

```
}
```

Your first name : hussein

niessuh

[06 - arrays\09.c]

بعد ان حجزنا مصفوفه للاسم واستقبلنا الاسم من المستخدم , استخدمنا الدالة strlen وهي اختصار string length تستخدم لمعرفة عدد حروف النص وهي احد دوال الملف الراسي string.h والذي يحتوي علي الثوابت وتعريفات دوال التعامل مع النصوص , قمنا بعد ذلك بطباعة النص بشكل عكسي من النهاية للبداية من خلال الحلقة التكرارية while وقمنا بدمج الشرط ومؤثرالنقصان معا ليصبح (while(count--)) , سيخرج عندما يصل المتغير count الي صفر, اي ان ترتيب الطباعة سيكون str[6],str[5],str[4],str[3],str[2],str[1],str[0] نلاحظ هنا ان النقصان في المتغير يحدث بعد التحقق من الشرط , لانه لو استخدمنا المؤثر الاخر (while(--count)) سيحدث النقصان اولاً ثم المقارنة وبالتالي لن يتم طباعة الحرف الاول من الاسم

مثال برنامج يقوم بعكس النص reverse string ثم طباعته

```
//string
```

```
char str []={"welcome to c strings"};
```

```
//chars counters
```

```
int count;
```

```
//use string length function to get string length
```

```
count = strlen(str);
```

```
//for swapping
```

```
char temp;
```

```
//loop counter
```

```

int i;

printf("before : %s \n",str);

//loop and swap
for(i=0 ; i<(count/2) ;i++)
{
    //store str[i] in temporarily location
    temp = str[i];
    //replace str[i] with str[count-1-i]
    str[i] =str[count-1-i];
    //replace str[count-1-i] with temp content
    str[count-1-i]=temp;
}

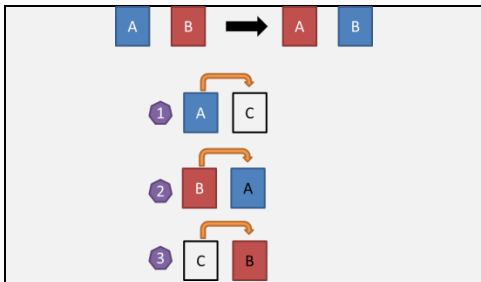
printf("after : %s \n",str);

```

before : abc def ghi jkl
after : lkj ihg fed cba

[06 - arrays\10.c]

قمنا بحجز سلسلة حروف ولكن هذه المرة لم نحدد الحجم ,فبدل من ان نقوم بعد حروف النص لتحديد الحجم المناسب سيقوم المترجم بذلك بدلا منا فهي للتسهيل علينا , فكرة عكس حروف النص بمعنى ان الاخير يصبح الاول والاول يصبح الاخير وهكذا تعتمد علي المبادلة swap تعرف بانها عملية تبادل محتويات المتغيرات , ولها اكثر من طريقة هذه ابسطها وبعيدا عن الكوب لنفرض ان لدينا كوبين احدهما به ماء والاخر به لبن ونريد تبديل محتوياتهما , فاننا سنحتاج الي كوب اخر فارغ كالتالي



1. نفرغ محتويات الماء في الكوب الفارغ
2. نفرغ محتويات كوب البن في كوب الماء , نكون بذلك وضعنا البن مكان الماء
3. نفرغ محتويات الكوب الذي به ماء في كوب البن , بذلك وضعنا الماء مكان البن

استخدمنا حلقة for لاننا نريد تبديل كل المحتويات وليس عنصرين فقط ونلاحظ ان نهاية الحلقة $i < (\text{count}/2)$ لاننا بمعنى اخر نقوم بتبديل نصف السلسلة مع النصف الاخر

```
//string
char str []={"mohamed#ali#omar#reda"};

//delimiter
char delimiter = '#';

//chars counters
int count;

//use string length function to get string length
count = strlen(str);

//loop counter
int i;

printf("string is : %s \n",str);

//loop and print char or skip if delimiter found
for(i=0 ; i<count ;i++)
{
    //check for delimiter ,if skip the rest
    if(str[i]==delimiter)
    {
        //add new line for next word
        printf("\n");
        continue;
    }
    //print chars
    printf("%c",str[i]);
}
```

```
string is : mohamed#ali#omar#reda
mohamed
ali
```

omar
reda

[06 - arrays\11.c]

بعد حجز متغير للنص , حجزنا كتغير لقيمة المحدد delimiter واعطيناه القيمة (#) ليتم الفصل بناء عليها , قمنا بالمرور علي كل حروف النص اذ لم يساوي المحدد يتم طباعته , واذا تطابق مع المحدد يتم تجاهل الطباعة والعودة الي بداية الحلقة

- الدوال الشائعة للتعامل مع النصوص : يحتوي الملف الراسي علي ثوابت وتعريفات دوال خاصة بالتعامل مع النصوص منها

-الدالة gets: احد مشاكل الدالة scanf في استقبال النصوص انها تتوقف عند وجود مسافة space بمعنى لو طلبنا من المستخدم ادخال اسمه كاملا , سيقوم المستخدم لادخال الاسم وبين كل اسم سيضع مسافة حتما , لذا ستقوم الدالة scanf بتخزين الاسم الاول فقط والدالة gets حل لتلك المشكلة حيث انها خاصة باستقبال النصوص فقط من المستخدم مثال لتوضيح طريقة الاستخدام والفرق بينهما

```
//name string
char str [255];
//ask user to type his name
printf("type your full name : ");
//get user name
gets(str);
//print it
printf("your name is : %s \n",str);
```

type your full name : mohamed hussein
your name is : mohamed hussein

```
//name string
char str [255];
//ask user to type his name
printf("type your full name : ");
//get user name
scanf("%s",str);
//print it
printf("your name is : %s \n",str);
```

type your full name : mohamed hussein
your name is : mohamed

- الدالة puts: تقوم بطباعة نص علي الشاشة , تتعامل فقط مع النصوص وما يميزها عن الدالة printf انها تضيف سطر جديد تلقائيا بعد كل طباعة

```
//string message
char msg[]={"welcome to string functions"};
//print string
puts(msg);
puts("this function used to print on stdio");
```


welcome to string functions
this function used to print on stdio

- الدوال strcpy & strncpy :

تستخدم تلك الدوال لنسخ النصوص من متغير لآخر، وهما اختصار لـ string copy ، الدالة تقوم بنسخ القيمة كاملة ، اما الدالة تقوم بنسخ عدد معين من الحروف تاخذ الدالة strcpy وسيطين الاول هو الجهة destination المراد النسخ اليها والثاني هو المصدر source اما متغير او نص ثابت ، اما الدالة strncpy مثل الاول اضافة الي متغير لعدد الحروف التي ستنسخ

<pre>//source string char src[] = {"c strings"}; //destination string char dest[50]; //copy strcpy(dest,src); //print puts(dest);</pre>	<pre>//source string char src[] = {"c strings"}; //destination string char dest[50]; //copy strncpy(dest,src,5); dest[5] = '\0'; //print puts(dest);</pre>
c strings	c str

يمكن ان يكون المصدر نص ثابت

```
strncpy(dest,"c strings",5);
```

- الدالة strcmp :

تقوم بالمقارنة بين نصين وترجع 0 في حالة التساوي وقيمة اكبر من الصفر اذا كان النص الاول اكبر من الثاني واخير قيمة اقل من الصفر اذا كان النص الاول اقل من الثاني

```
char src[] = {"c strings"};
```

```
printf("%s cmp with c strings = %d\n",src,strcmp(src,"c strings"));
```

```
printf("%s cmp with a strings = %d\n",src,strcmp(src,"a strings"));
```

```
printf("%s cmp with d strings = %d\n",src,strcmp(src,"d strings"));
```

c strings cmp with c strings = 0

c strings cmp with a strings = 1

c strings cmp with d strings = -1

- الدوال strcat, strncat :

تقوم بدمج لنصين في نص واحد باضافة نص الي نهاية نص اخر, الدالة strcat تاخذ وسيطين الاول المتغير الذي سيتم الدمج اليه , الثاني القيمة التي سيتم اضافتها او دمجها الدالة strncat مثل الاولى ولكن تحتاج وسيط اضافي وهو عدد الحروف التي ستدمج

```
//message header
```

```
char msg[50]="welcome";
```

```
//concatenate strings
```

```
strcat(msg,"Mr Mohamed");
```

```
puts(msg);
```

welcome Mr Mohamed

```
//message header
```

```
char msg[50]="welcome";
```

```
//concatenate strings
```

```
strncat(msg,"Mr Mohamed",6);
```

```
puts(msg);
```

welcome Mr Moh

سؤال وجواب

- ما الفرق بين "a" و 'a' ؟

- 'a' هي حرف لانها بين "" وحجمها واحد بايت فقط
- "a" هي نص لانها بين "" وحجمها 2 بايت الاول للحرف a والثاني لنهاية النص \0

- مصفوفة بها 4 عناصر وقمنا بحاوله قراءة العنصر رقم 6

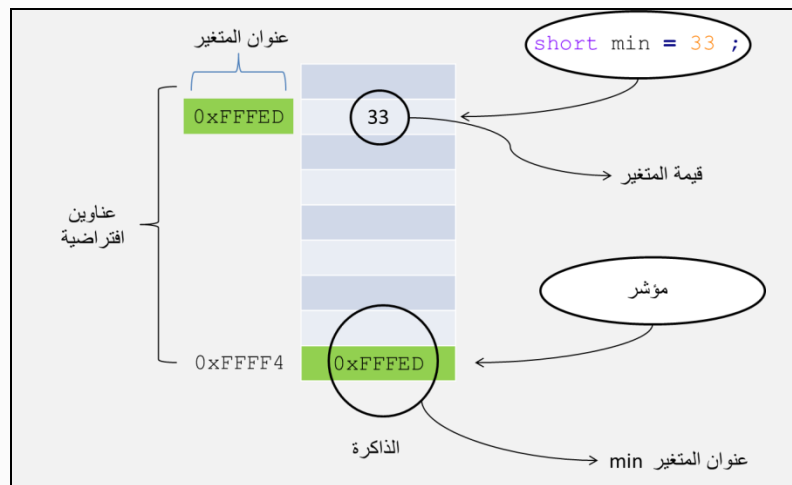


Pointers

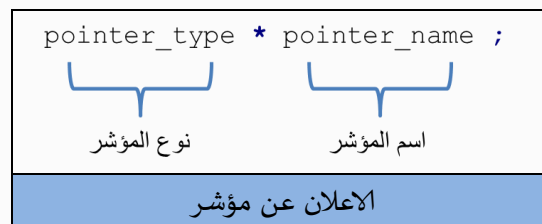
المؤشرات

المؤشرات pointers:

المؤشرات هي متغير يحتوي علي عنوان address متغير اخراي انها تستخدم للاشارة الي عناوين في الذاكرة , كما ذكرنا سلفا ان الذاكرة عبارة عن مجموعة من البايتات لكل منها عنوان , كما بالشكل التالي



انواع المؤشرات هي نفس انواع المتغيرات , القواعد الخاصة بتسمية المتغير هي نفسها الخاصة بتسمية المؤشر, يمكن لاكثر من مؤشر الاشارة الي نفس المكان في الذاكرة , طريقة الاعلان عن مؤشرا تختلف كثيرا عن طريقة الاعلان عن متغير, الفرق اننا نضيف علامة النجمة او الضرب (*) قبل الاسم



امثلة لمؤشرات

```
int * intPtr; //pointer to integer
char * charPtr; //pointer to char
float * floatPtr; //pointer to float
```

يستخدم المؤثر (&) للحصول علي عنوان اي متغير تسمي تلك العملية المرجع reference

مثال برنامج يقوم بطباعة عنوان متغير في الذاكرة

```
//declare int variable
int num = 12;
//declare pointer to integer
```

```
int * ptr ;  
//assign num address to ptr  
ptr = &num;  
  
//print num address  
printf("address is 0x%p \n",ptr);
```

address is 0x28ff18

[07 - pointers\01.c]

اعلنا عن مؤشر من نوع int

```
int * ptr ;
```

اعطينا للمؤشر عنوان المتغير num باستخدام الرمز & متبوعا باسم المتغير المراد الحصول علي عنوانه في الذاكرة

```
ptr = &num;
```

استخدمنا الرمز %p لطباعة قيمة مؤشر

- يتم الاشارة الي القيمة التي يشير اليها المؤشر باستخدام الرمز * dereference operator, ويستخدم اما لقراءة المحتوي الذي يشير المؤشر او لتغييره

مثال برنامج يقوم بقراءة وتغيير قيمة متغير باستخدام المؤشرات

```
//declare int variable  
int num =12;  
//declare pointer to integer  
int * ptr ;  
//assign num address to ptr  
ptr = &num;  
  
//print value that ptr point to  
printf("value is %d \n", (*ptr) );  
  
//change variable value via ptr
```

```
(*ptr) = 150;
//print num value
printf("new value is %d \n",num);
```

value is 12
new value is 150

[07 - pointers\02.c]

بعد ان انشانا متغير ptr واعطيناه عنوان المتغير num , قمنا بالاشارة الي محتويات المؤشر من خلال (*ptr), والتي فعليا هي محتويات المتغير num, واستخدمنا نفس الطريقة ولكن هذه المرة لتعديل المحتوي الذي يشير اليه المؤشر (*ptr) = 150 واخيرا قمنا بطباعة قيمة المتغير باستخدام المتغير ذاته للتأكد من القيمة الجديدة وحتى لا يحدث تشويش و خلط بين المفاهيم

الكود

المعني

متغير من نوع اعداد صحيحة به القيمة 12	int num =12;
مؤشر يستخدم للاشارة الي متغيرات اعداد صحيحة	int * ptr;
اعطينا المؤشر عنوان المتغير باستخدام الرمز &	ptr = #
قمنا بالاشارة الي محتوى المؤشر باستخدام الرمز * للتعديل عليه	(*ptr) = 150;

مثال برنامج يقوم قراءة قيمة متغير باستخدام مؤشر وتغييرها باستخدام مؤشر اخر

```
char ch = 'A';

//first ptr
char *firstPtr = &ch;

//second ptr
char *secondPtr = &ch;

//read value via firstPtr
printf("value is %c \n", (*firstPtr));

//change value via secondPtr
(*secondPtr) = 'B';

//print to check changes
printf("value is %c \n",ch);
```

value is A
value is B

[07 - pointers\03.c]

انشانا مؤشرين لحروف واعطيناهما عنوان نفس المتغير , وقمنا بعرض قيمة المتغير باستخدام المؤشر الاول (*firstPtr) , وقمنا بتغيير محتويات المتغير من خلال المؤشر الثاني (*secondPtr) = 'B' وطبعنا قيمة المتغير للتأكد من تغير القيمة

- المؤشرات والمصفوفات pointers and arrays :

- المصفوفات عبارة عن مجموعة من الاماكن المتتالية في الذاكرة , وحيث انها متتالية فبالتالي اذا عرفنا عنوان احد عناصر اي مصفوفة يمكننا التحرك داخلها وذلك لان السي تسمح بالعمليات الحسابية (الجمع والطرح) علي المؤشرات

مثال برنامج لطباعة محتويات مصفوفة من الحروف مرة باستخدام الدالة puts واخري باستخدام المؤشر

```
//char array
char str[] = "string again";
//char pointer
char *ptr = &str[0];
//string length
int len = strlen(str);
//loop counter
int i;
//print using puts function
puts(str);
//print using pointer
for(i=0; i<len; i++)
{
    printf("%c", *(ptr+i));
}
```

string again
string again

[07 - pointers\04.c]

انشانا نص واعطيناه قيمة مباشرة

```
char str[] = "string again";
```

اعلنا عن مؤشر واعطيناه عنوان اول حرف في المصفوفة

```
char *ptr = &str[0];
```

ثم قمنا بطباعة النص باستخدام الدالة puts , ثم اعتمادا علي ان مصفوفة الحروف موجودة في الذاكرة بشكل متتالي ولدينا الان عنوان اول حرف , اذا بزيادة العنوان بمقدار واحد نحصل علي عنوان الحرف الثاني , وزيادة اخري نحصل علي عنوان الحرف الثالث وهكذا اي ان $str[i] = *(ptr+i)$

المصفوفة	المؤشر
str[0]	*(ptr)
str[1]	*(ptr+1)
str[2]	*(ptr+2)
str[3]	*(ptr+3)
str[4]	*(ptr+4)

لا بد من وجود الاقواس حول المؤشر والعملية الحسابية التي تجري عليه , اذ لا بد من ان نفرق بين تغيير في العنوان وبين التغيير في القيمة التي يشير اليها العنوان

- يمكن التعامل مع المصفوفة بنفس الطريقة ايضا فبدلا من كتابة str[1] يمكن كتابة *(str+1) وذلك لان اسم المصفوفة يمكن اعتباره مؤشر لبداية المصفوفة لنري المثال التالي

مثال برنامج لطباعة عناوين عناصر مصفوفة بطريقتين احدهما باستخدام الرمز & والاخري باستخدام اسم المصفوفة + ترتيب العنصر

```
//char array
char str[] = "have fun";
//string length
int len = strlen(str);
//loop counter
int i;

printf("&str[i]    (str+i)  str[i]\n");

for(i=0; i<len; i++)
{
    printf("0x%p  0x%p  %c\n", &str[i], (str+i), str[i]);
}
```


&str[i]	(str+i)	str[i]
0x0028FF0F	0x0028FF0F	h
0x0028FF10	0x0028FF10	a
0x0028FF11	0x0028FF11	v
0x0028FF12	0x0028FF12	e
0x0028FF13	0x0028FF13	
0x0028FF14	0x0028FF14	f
0x0028FF15	0x0028FF15	u
0x0028FF16	0x0028FF16	n

[07 - pointers\05.c]

نلاحظ ان العناوين واحدة وهذا يؤكد ان $(str+i) = \&str[i]$ ومن هنا بدلا من ان نكتب

```
char *ptr = &str[0];
```

يمكن كتابة

```
char *ptr = str;
```

- النصوص والمؤشرات :

النصوص هي عبارة عن مصفوفة من الحروف , لذا يمكن استخدام المؤشرات مع النصوص الثابتة فمثلا يمكن كتابة

```
char *ptr = "string again";
```

بدلا من ان نكتب

```
char str[] = "string again";
//char pointer
char *ptr = str;
```

وفي تلك الحالة فان المؤشر سيشير الي عنوان اول حرف فقط في النص , النص "string again" يسمي نص ثابت string constant

ملاحظات علي تلك الطريقة $char *ptr = "string again"$

- هذا النص ثابت اي لا يمكن تغييره
- يمكن لهذا المؤشر ان يتغير عنوانه ليشير الي اي نص اخر
- المؤشر يحتوي علي عنوان اول حرف فقط بعكس المصفوفة $char str[] = "string again"$ التي تحتوي علي النص ذاته

```
//constant string
char *str = "have fun";

printf("%s\n",str);
```

have fun

[07 - pointers\06.c]

لنفرق بينهما ثانية

`char *str = "string again"`

str هو مؤشر

لا يمكن تغيير اي شيء في هذا النص لانه ثابت لايمكن مثلا `*(str+2)='w'` يمكن ان يشير الي اي سلسلة نصية اخري

`char str[] = "string again"`

str هنا هي مصفوفة

يمكن تغيير محتوياتها يمكن كتابة مثلا `str[3]='e'`

str ستظل دائما تحمل نفس المساحة التخزينية التي حجزت لها

- لاستخدامها في الادخال يجب ان تشير اولا الي مساحة تخزينية فمثلا لايمكن كتابة الاتي

```
char *str ;
scanf("%s",str);
```

في تلك الحالة سيحدث crashing لان المؤشر لا يشير الي عنوان معين والحل هو

```
char *str ;
char ch[40] ;
str=ch;
scanf("%s",str);
```

هنا المؤشر يشير الي مساحة تخزينية وهي المصفوفة , حيث يمكن وضع القيم بها

- مصفوفة من المؤشرات

يمكن انشاء مصفوفة من المؤشرات حيث كل عنصر عبارة عن مؤشر امثلة

<code>int*ptr[3];</code>	مصفوفة بها 3 مؤشرات الي اعداد صحيحة
<code>float *fPtr[5];</code>	مصفوفة بها 5 مؤشرات الي اعداد حقيقية
<code>char *cPtr[4];</code>	مصفوفة بها 5 مؤشرات الي حروف

- مصفوفة النصوص

يمكن انشاء مصفوفة من النصوص , حيث كل عنصر بها هو مؤشر الي نص ما

مثال برنامج توضيحي

```
//array of strings,each element is pointer to char
char *colors[]={ "white" , "black" , "red" , "green" , "blue" };
int index;
//loop on array and print
for(index=0 ; index < 5 ; index++)
{
    puts(colors[index]);
}
```

white
black
red
green
blue

[07 - pointers\07.c]

قمنا بالاعلان عن مصفوفة من المؤشرات , حيث كل عنصر عبارة عن مؤشر الي نص ثابت

- المؤشرات الثابتة `const pointers` :

- مؤشر لقيمة ثابتة ويسمي read-only location اي لا يمكن تغير القيمة المشار اليها مثال

```
int num = 22;
```

```
const int *ptr = &num;
```

وبناء علي ذلك لا يمكن محاولة تغير القيمة المشار اليها من خلال المؤشر , فالكود التالي سيعطي خطأ ولن يعمل

```
*(ptr)=40;
```

- مؤشر ثابت ويسمى read-only pointer اي بمجرد اعطائه عنوان لا يمكن تغير هذا العنوان الي اي عنوان اخر, لابد ان ياخذ عنوان عند الاعلان عنه مباشرة

```
int num = 22;  
int *const ptr = &num;
```

وايضا اي محاولة لجعله يشير الي اي عنوان اخر هي خطأ كالتالي, البرنامج لن يعمل

```
int num = 22;  
int *const ptr = &num;  
ptr = &index;
```

- اسلوب كتابة الاكواد coding style :

نخرج بعيدا قليلا عن المؤشرات والمصفوفات والنصوص , سنتحدث عن اسلوب كتابة الاكواد والبرامج , فمطوري البرامج ربما يستغرقون اوقات اكبر من التي احتاجوها لكتابة البرامج في تنقيح البرامج للبحث عن خطأ ما او تحديث الاكواد لاضافة شئ او لتحسينه , وبالتالي يجب مراعاة اننا لا نقوم بكتابة مجموعة من التعليمات للحاسب ليقيم بتفيذها فقط بل ينبغي ان يكون الكود واضح وبه تعليقات كافية لفهم ماذا يفعل وكيف يفعل , وهناك قاعدة مشهورة هنا KIS التي تعني keep it simple . الخلاصة ان الكود يجب ان يكون سهل وواضح وصريح

فمثلا الكود الذي لا يحتوي علي تعليقات هو بمثابة قنبلة موقوتة لانه عاجلا او اجلا سنعود الي هذا الكود لاي سبب مثلا لاصلاح خطأ ظهر مؤخرا او لتحسين شئ ما او لاضافة شئ وقتها ستكون المهمة صعبة .

يجب ان تاخذ وقتا كافيا للتفكير في البرنامج كيف سيؤدي وظيفته , ويجب ان يكون الحل واضحا قبل الشروع في كتابة اي كود , يمكن بعد ذلك ان تبدا بكتابة التعليقات اولا علي شكل خطوات واضحة للحل وبذلك سيكون وقت كتابة الكود اقل بكثير وبأخطاء اقل ايضا

- الملاحظات الشائعة في اسلوب كتابة الاكواد

- الاسماء يجب ان تدل علي ما تفعله سواء اسماء المتغيرات او المؤشرات ... الخ , ويفضل اضافة تعليق بجانبه لاي ملاحظات تتعلق به

```
int carSpeed;//car speed in kilometer/hour
```

- الفراغات في البداية indentation

يفضل اضافة فراغ جديد بواسطة مفتاح Tab في بداية كل block او في جمل الشرط والحلقات المتداخلة , حتي يبدو اكثر تنظيما ووضوح

```
if(index > 4)
{
    if(index == 5)
    {

    }
}
else
{
    if(index == 2)
    {

    }
}
```

- يفضل استخدام الاقواس في حالات الشرط الذي يجمع بين اكثر من علاقة شرطية

```
if( ( grade >= 0.75 ) && ( grade < 0.85 ) )
{
    printf("Very good\n");
}
```

- التعليقات ثم التعليقات

الخلاصة

- تستخدم المؤشرات للإشارة الي عناوين في الذاكرة
- يستخدم & قبل اسم المتغير للحصول علي عنوانه في الذاكرة لذا يسمي reference operator
- يستخدم * قبل اسم المؤشر لقراءة او لتعديل المحتوي الذي يشير اليه المؤشر dereference operator
- يمكن اجراء عمليات الجمع والطرح علي المؤشرات
- تستخدم المؤشرات مع المصفوفات وذلك للقراءة منها والتعديل عليها



Functions

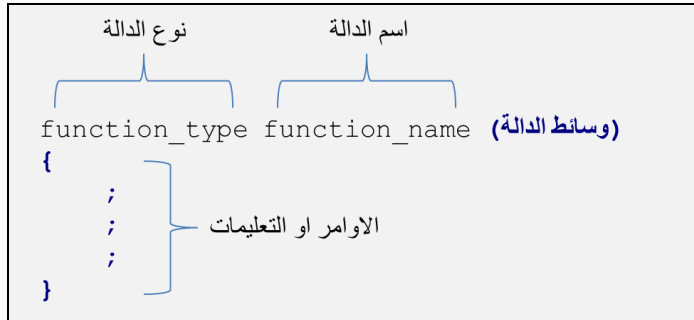
الدوال

- الدوال functions

الدوال هي حجر الاساس لاي برنامج مكتوب بالسي , فالسي تحتوي علي مجموعة كبيرة من الدوال المهمة استخدمنا بعضها مثل printf,scanf,puts ,gets, وبالإضافة لتلك الدوال لدينا الحرية لكتابة دوال خاصة بنا واستخدامها , وايضا استخدام دوال كتبت بواسطة مطورين اخرين .

الدالة هي مجموعة من التعليمات والبيانات المرتبطة باداء مهمة معينة ولها اسم مميز وايضا لها ال block الخاص بها , ويتم استدعاء تلك الدالة في اماكن مختلفة في البرنامج , وهي بذلك ساعدت علي ان يقل حجم الكود المكتوب وكذلك حجم البرنامج بعد ذلك

تتكون اي دالة من :



- نوع الدالة function type
- اسم الدالة function name
- وسائط الدالة function parameters
- الاوامر او التعليمات function statements

- نوع الدالة function type

نوع الدالة هو نوع البيانات التي ستقوم الدالة بارجاعها بعد تنفيذ الاوامر الخاصة بها فقد يكون مثلا int او float او char , ويمكن للدالة الا ترجع اي قيمة تسمي في تلك الحالة void

- اسم الدالة function name

هو اسم مميز للدالة يتم استدعائها به , ويتبع قواعد التسمية مثل المتغيرات

- وسائط الدالة function parameters

هي القيم والبيانات التي تحتاجها الدالة لاداء وظيفتها , ويمكن للدالة الا تاخذ وسائط , ويتم وضع كلمة void مكان الوسائط ولكنه ليس شرطاً

- الاوامر او التعليمات function statements

هي الاوامر التي يتم تنفيذها لكي تؤدي تلك الدالة مهمتها

- توجد طريقتين في تعريف الدوال

- يتم تعريف define الدالة قبل الدالة الاساسية main اي قبل استخدامها
- يتم الاعلان declare عن الدالة اولاً قبل الدالة الاساسية ثم تعريفها define or implement بعد ذلك وتسمي تلك الطريقة بنموذج الدالة function prototype

- في الطريقة الثانية يتم الاعلان عن الدالة اولا وذلك لاعلام المترجم بها , حيث يستخدم المترجم هذا الاعلان للتحقق من استدعاء الدالة بالشكل السليم من حيث عدد الوسائط وانواعها ونوع الدالة , في حالة عدم اعطاء الاعلان سيفرض المترجم ان الدالة من نوع int وانها تقبل باي عدد من الوسائط , سنستخدم تلك الطريقة في الكتاب

مثال دالة تقوم بطباعة جملة علي الشاشة

```
#include<stdio.h>

//function prototype
void sayHello();

int main()
{
    //call function
    sayHello();

    return 0;
}

void sayHello()
{
    printf("hello from function\n");
}
```

hello from function

[08 - functions /01.c]

قمنا بالاعلان عن الدالة قبل الدالة الاساسية , قمنا بكتابة نموذج الخاص function prototype , الدالة هنا لاتاخذ وسائط ولا تعيد اي قيم لذا void هي

```
//function prototype
void sayHello();
```

قمنا بتعريف الدالة بالاسفل بعد الدالة الاساسية


```
void sayHello()
{
    printf("hello from function\n");
}
```

قمنا باستدعاء الدالة من داخل الدالة الاساسية حيث يتم تنفيذ البرنامج , حتي ان كانت الدالة لا تأخذ وسائط فلا بد من كتابة الاقواس بعد اسم الدالة لان الاقواس هي ما يميز الدالة عن المتغيرات

```
int main()
{
    //call function
    sayHello();

    return 0;
}
```

- يمكن استدعاء دالة من داخل دالة اخري كما في المثال قمنا باستدعاء الدالة printf من داخل الدالة sayHello

- كان يمكن كتابة اعلان الدالة كالتالي , حيث كلمة void تعني ايضا انها لا تحتاج وسائط

```
void sayHello(void);
```

- يمكن الاعلان عن الدالة وتعريفها مباشرة قبل الدالة الاساسية , ولكن الطريقة المستخدمة اكثر تنظيما

```
#include<stdio.h>

void sayHello()
{
    printf("hello from function\n");
}

int main()
{
    //call function
    sayHello();

    return 0;
}
```

```
//function prototype
int add(int first,int second);

int main()
{
    int num1 = 20 ,num2 = 40,result;

    //call with constant values
    printf("result of 13 + 46 = %d \n", add( 13 , 46 ));
    //call with variable
    result = add(num1,num2);
    printf("result of %d + %d = %d \n",num1,num2,result);

    return 0;
}

int add(int first,int second)
{
    //add two numbers
    int sum = first + second ;
    //return addition result
    return sum;
}
```

[08 - functions /02.c]

الدالة من نوع int وتحتاج الي وسيطين كلاهما من نوع int

```
int add(int first,int second);
```

يمكن كتابة نموذج الدالة بدون اسامي للوسائط فقط الانواع هي الاهم هنا , اما في التعريف فلا بد ان تعطي اسماء للوسائط

```
int add(int,int);
```

في تعريف الدالة حجزنا متغير صحيح وقمنا بتخزين ناتج جمع الوسيطين به , واستخدمنا الكلمة المحجوزة return لارجاع ناتج الجمع

```
int add(int first,int second)
{
    //add two numbers
    int sum = first + second ;
    //return addition result
    return sum;
}
```

قمنا باستدعاء الدالة بطريقتين الاولى باعطائها قيم مباشرة وطباعة الناتج مباشرة , الثانية باعطائها قيم متغيرات صحيحة وتخزين الناتج ثم طباعته بعد ذلك

```
printf("result of 13 + 46 = %d \n", add( 13 , 46 ));
```

```
result = add(num1,num2);
printf("result of %d + %d = %d \n",num1,num2,result);
```

- راينا انه يمكن ان نستدعي دالة من وسيط دالة اخري , بشرط ان يكون نوع الدالة ونفس نوع الوسيط مثل ذلك

```
printf("result of 13 + 46 = %d \n", add( 13 , 46 ));
```

هنا سيتم استدعاء الدالة add(13 , 46) اولاً ثم استدعاء الدالة printf واستبدال الوسيط الثاني بالقيمة التي تم ارجاعها من الدالة add

مثال دالة للتحقق من الرقم الصحيح هل زوجي او فردي حيث تعيد 1 اذا كان زوجي وصفر اذا كان فردي

```
//function prototype
int isEven(int);

int main()
{
    int number =30;

    if( isEven(number) )
    {
        printf("%d is even\n",number);
    }
    else
```

```
{  
    printf("%d is odd\n",number);  
}
```

```
return 0;  
}
```

```
int isEven(int num)
```

```
{  
    if(num%2)  
    {  
        //number is odd  
        return 0;  
    }  
    else  
    {  
        //number is even  
        return 1;  
    }  
}
```

30 is even

[08 - functions /03.c]

- الدالة الاساسية main function

الدالة الاساسية هي اهم دالة في السي حيث يبدأ منها تشغيل البرنامج , وهي دالة من نوع int حيث تعيد تلك القيمة لنظام التشغيل operating system وهذه اما تكون صفرا والتي تعني ان البرنامج انتهى بشكل طبيعي وبدون مشاكل اي قيمة اخري non-zero في حالة حدوث اخطاء او مشاكل

```
int main()  
{  
  
    return 0;  
}
```

- يمكن للدالة الأساسية ان تأخذ وسائط , وتكون من نظام التشغيل وسنتعرض لهذا لاحقا

- مجال المتغيرات variable scope

هذا يحدد اين يمكننا ان نستخدم المتغير ولدينا هنا عدة انواع

- Block scope وذلك للمتغيرات المعرفة داخل الـ block بين {} ويمكن استخدام المتغير بداية من الاعلان عنه وحتى نهاية الـ block مثل المتغيرات المعرفة في الدالة الأساسية , ويسمى متغير محلي local variable وسائط اي دالة تعتبر محلية وخاصة بتلك الدالة

```
int main()
{
    // يمكن استخدامه بداية من هنا وحتى نهاية الدالة
    int first = 20;

    return 0;
}
```

المتغير هنا يوصف بأنه block scope ويمكن استخدامه بداية من السطر الذي تم الاعلان عنه فيه وحتى نهاية الـ block اي }

- Nested Block Scope يمكن ان يكون لدينا blocks متداخلة , فالمتغيرات المعلن عنها في الـ block الخارجي يمكن استخدامها في الـ block الداخلي

```
{

    int first = 20;

    {
        //very local variable
        int second;

        printf("value of first is %d\n",first);
    }

    return 0;
}
```

هنا المتغير first تم استخدامه في الـ block الداخلي ، اما المتغيرات المعرفة داخليا لا يمكن ان تستخدم في الـ الخارجي تماما مثل زجاج سيارات الشخصيات الهامة من في الداخل يري من في الخارج اما من بالخارج لايري من بالداخل ، فالكود التالي خاطئ ولن يعمل لانه لا يعرف هذا المتغير

```
{
    //very local variable
    int second;
}

printf("value of second is %d\n",second);
```

لو اننا اعلنا عن متغير في بلوك داخلي بنفس اسم متغير موجود في بلوك خارجي الاولوية تكون دائما للبلوك الداخلي ، كالمثال التالي ولدينا متغيرين بنفس الاسم

```
{
    int num=99;
    {
        int num=10;

        printf("%d",num);
    }
}
```

دالة الطباعة هنا ستطبع القيمة الاقرب لها الموجودة في البلوك الداخلي اي 10

- Function Scope هنا المتغير يكون مرئي ويمكن استخدامه من بداية الدالة وحتى اخرها وهو مرتبط بـ goto فقط حيث العلامات labels تكون مرئية من اي مكان في الكود سواء قبلها او بعدها

```
int main ()
{
    goto middle;
    .
    .

    middle :
    .
    .
    return 0;
}
```

- Program Scope للمتغيرات التي يتم الاعلان عنها خارج الدوال ,ويسمى بالشامل global وهو مرئي ويمكن استخدامه من اي ملف من ملفات البرنامج

```
//function prototype
void printGlobal();

//global variable
int number =150;

int main()
{
    printf("value of global variable is %d , from main \n", number );
    //call printGlobal function
    printGlobal();

    return 0;
}

void printGlobal()
{
    printf("value of global variable is %d , from printGlobal \n", number );
}
```

value of global variable is 150 , from main
value of global variable is 150 , from printGlobal

[08 - functions /04.c]

قمنا بالاعلان عن المتغير خارج الدالة الاساسية لذا هو شامل global لكل الكود

```
//global variable
int number =150;
```

استخدمنا هذا المتغير مرتين احدهما من الدالة الاساسية والاخري من الدالة printGlobal

- file scope المتغير مرئي ويمكن استخدامه بالملف الذي تم الاعلان عنه فيه فقط , وهو يعرف خارج الدوال مثل global ولكن يضاف كلمة static قبل نوع المتغير

```
//- 127 -variable visible to this file only
```

```
static int number =90;
```

- فئات التخزين storage classes

تحدد عمر المتغير lifetime الفترة التي سيبقي فيها في الذاكرة وايضا مجال استخدام المتغير وهي

- تلقائي auto

المساحة التخزينية للمتغير مؤقتة , اي يمكن استخدامها لاي غرض اخر بمجرد انتهاء ال block الخاص بالمتغير مثل المتغيرات التي تعرف داخل الدوال

التي تنتهي بمجرد انتهاء تنفيذ الدالة وتلك الكلمة auto لا تستخدم لان المتغيرات التي تعرف داخل اي block هي مؤقتة بشكل افتراضي

```
int add(int first,int second)
{
    //add two numbers
    int sum = first + second ;
    //return addition result
    return sum;
}
```

المتغير sum تلقائي لانه بانتهاء تنفيذ الدالة لن يصبح له وجود

- ساكن static

ويختلف معناها حسب الاستخدام ولها ثلاثة استخدامات

1. اذا استخدمت مع المتغيرات داخل اي دالة , هذا يعني ان المساحة التخزينية لهذا المتغير بمجرد حجزها اول مرة ستبقي خاصة به

, مثال برنامج يقوم بطباعة قيمة المتغير داخل دالة بعد زيادته

```
void printValue()
{
    static int num =20;
    num++;
    printf("value is %d \n",num);
}
```

وباستدعاء تلك الدالة 3 مرات مثلا يتكون النتيجة

```
printValue();
printValue();
printValue();
```


value is 21
value is 22
value is 23

عند استدعاء الدالة للمرة الاولى سيتم حجز مساحة للمتغير num وسيعطي قيمة مباشرة 20 , نتيجة لمؤثر الزيادة ستصبح القيمة 21 , عند انتهاء الدالة لن يتم حذف المتغير من الذاكرة لانه من نوع static , عند استدعاء الدالة للمرة الثانية ستجد ان المتغير موجود في الذاكرة , بالتالي سيتم تطبيق مؤثر الزيادة مباشرة وستصبح القيمة 22 وهكذا لاي استدعاء تالي للدالة .2. اذا استخدمت مع المتغيرات المعرفة خارج الدالة الاساسية تعني ان هذا المتغير خاص بهذا الملف فقط ولا يمكن الاشارة اليه من اي ملف اخر , حيث ان السي تسمح باستخدام متغير معرف في ملف اخر

// - 128 - variable related to this file only

```
static int number = 90;
```

```
int main()  
{  
    return 0;  
}
```

3. اذا استخدمت مع تعريف الدوال تعني ان تلك الدالة لا يمكن استخدامها خارج الملف المعرفة به

- مسجل register

يتم تخزين المتغير في احد المسجلات الداخلية للمعالج , لان القراءة والكتابة في المسجلات اسرع بكثير من الذاكرة , ومن النقاط المهمة هنا اننا لا نستطيع الحصول علي عنوان هذا المتغير لانه ليس في الذاكرة , وهذا النوع يساعد في تسريع البرنامج

```
register int counter = 0;
```

```
for(counter = 0 ; counter < 5 ; counter ++)  
{  
    printf("%d", counter);  
}
```

- خارجي extern

في السي يمكن استخدام متغير معرف في ملف اخر من ملفات البرنامج , وتستخدم مع المتغيرات الشاملة global التي لها program scope لدينا عنا ملفين علي اليمين تم تعريف المتغير , واستخدمناه في الملف علي اليسار باستخدام الكلمة extern

```
extern int x;
```

```
int x = 99;
```

<pre>int main () { return 0; }</pre>	<pre>int main () { return 0; }</pre>

• متطايير volatile

يستخدم في حال ان قيمة المتغير سيتم تغييرها بعيدا عن البرنامج اي بدون استخدام جملة (var = 5) مثلا التي تستخدم لتعين قيمة لمتغير , ونحن بذلك نعلم المترجم الا يقوم بعملية تحسين optimization علي المتغير , مثال ان يكون لدينا متغير به القيم المدخلة من لوحة المفاتيح , عنوان المتغير موجود باحد المسجلات الخاصة بلوحة المفاتيح وبالتالي قيمة المتغير يتم تغييرها بشكل تلقائي عند الضغط علي اي مفتاح ,

- الدوال الخاصة بالرياضيات math library functions

تحتوي السي علي مجموعة من الدوال الخاصة بالعمليات الرياضية , وهذه الدوال موجودة في الملف الراسي math.h , نستعرض الدوال الشائعة منها

الدالة	الوظيفة	مثال
sqrt(x)	الجزر التربيعي للقيمة ل x	sqrt(16.0) = 4.0
fabs(x)	القيمة المطلقة ل x	fabs(-40.5) = 40.5
pow(x, y)	قيمة x مرفوعة لاس y	pow(5, 2) = 25.0
sin(x)	جيب الزاوية x حيث x بالتقدير الدائري radian	sin(0.0) = 0.0
cos(x)	جيب تمام الزاوية x حيث x بالتقدير الدائري radian	cos(0.0) = 1.0
tan(x)	ظل الزاوية x حيث x بالتقدير الدائري radian	tan(0.0) = 0.0
log10(x)	لوغاريتم x للأساس 10	log10(100.0) = 2.0
ceil(x)	التقريب لأكبر قيمة	ceil(9.2) = 10.0
floor(x)	التقريب لأقل قيمة	floor(9.2) = 9.0

- الارقام العشوائية

تستخدم الدالة rand() في توليد الارقام العشوائية من صفروحتي قيمة الثابت RAND_MAX والذي يساوي 0x7FFF اي 32767 وهذا المدي ربما لا نحتاج اليه فمثلا ربما نحتاج رقم عشوائي من صفرا الي 10 يتم ذلك باستخدام (%) باقي القسمة وذلك لان باقي القسمة علي رقم لابد ان تكون اقل من هذا الرقم فمثلا

rand() % 10

تعطي رقم عشوائي من صفروحتي 9 , لو اردنا ان يكون الرقم العشوائي بداية من الصفر نقوم باضافة 1 كالتالي

```
rand() % 10 + 1
```

تعطي رقم عشوائي من 1 الي 10

- الدالة srand

تستخدم هذه الدالة لاعداد الدالة rand وذلك لتوليد ارقام جديدة في كل مرة نستخدم فيها البرنامج ونستخدم دالة الوقت كوسيط

```
srand(time(NULL));
```

فلو افترضنا ان الاعداد العشوائية عبارة عن كتاب وكل صفحة بها رقم عشوائي , فكل مرة نستخدم فيها البرنامج ونستدعي الدالة rand فانها تبدأ من اول الكتاب وتعيد نفس الارقام في كل مرة , اما لو استخدمنا الدالة srand فانها بشكل او باخر تجعل rand تبدأ من مكان اخر متغير من الكتاب

مثال برنامج يقوم بتوليد رقم عشوائي من 1 الي 10 ويعطي المستخدم 4 محاولات لتخمين الرقم

```
int main()
{
    //seed rand
    srand(time(NULL));
    //generate random number
    int random = (rand() % 10) + 1;
    //user guess
    int guess;
    //trials
    int trials;

    for(trials = 1; trials <= 4; trials++)
    {
        printf("guess number from 1..10: ");
        scanf("%d", &guess);

        if(guess == random)
        {
            break;
        }
    }
}
```

```

if(guess == random)
{
    printf("Excellent , in %d trial\n",trials);
}
else
{
    printf("number is %d \n",random);
}

return 0;
}

```

```

guess number from 1..10 : 1
guess number from 1..10 : 9
guess number from 1..10 : 7
guess number from 1..10 : 4
number is 6

```

[08 - functions /07.c]

- طرق استدعاء الدوال calling functions

طريقة استدعاء الدالة هي الطريقة التي تعطي بها الدالة الوسائط وهي احدي طريقتين :

- استدعاء بالقيمة calling by value وفيها يتم اخذ نسخة من الوسائط للعمل عليها داخل الدالة , اي تغيير في تلك النسخة لا يؤثر علي المتغير الاصيل
- استدعاء بالمؤشر calling by reference وفيها يتم اخذ نسخة ايضا ولكن التغيير في النسخة يغير في المتغير الاصيل مباشرة , ويظهر عندما تكون وسائط الدالة من نوع المؤشرات

لفهم الفرق جيدا سنقوم بتعريف دالة مهمتها تبديل محتوى متغيرين swap احدهما بالقيمة والاخري بالمؤشر وسنري الفرق

```
//declare swap function by reference
```

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

```
int main()
```

```
//declare swap function by value
```

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

```
int main()
```

<pre> { //test variables int x = 10 ,y = 50; //print value before swap printf("before x = %d , y= %d \n",x,y); //swap by reference swap(&x , &y); //print value after swap printf("after x = %d , y= %d \n",x,y); return 0; } void swap(int *first,int *second) { int temp = (*first); (*first) = (*second); (*second) = temp; } </pre>	<pre> { //test variables int x = 10 ,y = 50; //print value before swap printf("before x = %d , y= %d \n",x,y); //swap by value swap(x,y); //print value after swap printf("after x = %d , y= %d \n",x,y); return 0; } void swap(int first,int second) { int temp = first; first = second; second = temp; } </pre>
<p>before x = 10 , y= 50 after x = 50 , y= 10</p>	<p>before x = 10 , y= 50 after x = 10 , y= 50</p>
[08 - functions /09.c]	[08 - functions /08.c]

في الحالة التي علي اليمين تم تبديل محتوى متغيرات الوسائط ولكن هذا التغير لم يحدث في المتغيرات الاصلية , اما في الحالة التي علي اليسار تم تبديل محتوى متغيرات الوسائط وظهر التغير مباشرة في المتغيرات الاصلية لانها تستخدم المؤشرات

في حالة استدعاء بالقيمة تبديل المحتويات يحدث فعليا ولكن في المتغيرات الخاصة بالدالة التي تعتبر نسخة من الاصل وليس المتغيرات الاصلية

اما في حالة الاستدعاء بالمؤشر فانه عند اخذ نسخة منه داخل الدالة يبقى يحتوي علي العنوان الاصيل , لانه عند النسخ القيم تبقي كما هي ولذا التغير في المحتوي الذي يشير اليه المؤشر يغير مباشرة في المتغير الاصيل

- المصفوفات يتم اعطائها كوسيط للدوال بالمؤشر بشكل افتراضي

```
//declare print array function
void printArray(int array[], int len);

int main()
{
    //array of integers
    int nums[] = {11,54,78,90,123,46,74,91};

    printArray(nums,8);

    return 0;
}

void printArray(int array[], int len)
{
    int index;
    printf("\n[ ");
    //loop ,print array elements
    for(index =0 ;index <len ;index++)
    {
        printf(" %d ",array[index]);
    }

    printf(" ]\n");
}
```

[11 54 78 90 123 46 74 91]

[08 - functions /10.c]

اعلنا عن نموذج دالة لطباعة محتوى مصفوفة اعداد صحيحة والتي تحتاج الي المصفوفة المراد طباعتها كوسيط اول وحجم المصفوفة كوسيط

ثان

```
void printArray(int array[], int len);
```

يستخدم `int array[]` لتحديد الوسيط من نوع مصفوفة لاعداد صحيحة , الدالة ذاتها تقوم بالمرور علي عناصر المصفوفة وطباعتها

مثال دالة تقوم بعكس محتويات مصفوفة اعداد صحيحة , تستخدم الدالة السابقة لطباعة محتوى المصفوفة للتحقق من ذلك

```
//declare reverse array function
```

```
void reverse(int array[],int len);
```

```
int main()
```

```
{  
    //array of integers  
    int nums[]={11,54,78,90,123,46,74,91};  
    //print content before reverse  
    puts("before");  
    printArray(nums,8);  
    //apply reverse  
    reverse(nums,8);  
    //print content after reverse  
    puts("after");  
    printArray(nums,8);  
  
    return 0;  
}
```

```
void reverse(int array[],int len)
```

```
{  
    //loop index , temp storage for swap  
    int l,temp;  
    //loop and swap each two elements  
    for(i=0;i<len/2;i++)  
    {  
        temp = array[i] ;//store it in temp  
        array[i] = array[len-i-1] ;//take content of other element  
        array[len-i-1] = temp ;//take original content of the other element  
    }  
}
```

before

[11 54 78 90 123 46 74 91]

after

[91 74 46 123 90 78 54 11]

[08 – functions /11.c]

حيث ان المصفوفة تعطي كوسيط بالمؤشر , فان عمليات التبديل بين العناصر ظهرت بشكل مباشر في المصفوفة الاصلية , عملية عكس محتويات المصفوفة تقوم علي تبديل محتويات كل عنصرين متقابلين الاول والاخير , الثاني وقبل الاخير وهكذا , الحلقة for تبدأ من صفرو حتي نصف حجم المصفوفة لاننا نقوم بتبديل نصف المصفوفة مع النصف الاخر ولو اكملت الحلقة حتي حجم المصفوفة ستعود المصفوفة كما كانت .

مثال دالة تقوم بحساب متوسط مصفوفة

```
//declare array average function
int average(int array[], int len);
int main()
{
    //array of integers
    int nums[] = {11,54,78,90,123,46,74,91};
    int avg = average(nums , 8 );
    printf("array average is %d \n",avg);

    return 0;
}

//declare array average function
int average(int array[], int len)
{
    int i,sum=0;
    for(i=0; i<len ;i++)
    {
        sum +=array[i];
    }
    return sum /len ;
}
```


array average is 70

[08 - functions /12.c]

الخلاصة

- الدوال هي حجر الأساس لأي برنامج مكتوب بالسي
- استدعاء بالقيمة calling by value وفيها يتم اخذ نسخة من الوسائط للعمل عليها داخل الدالة , أي تغيير في تلك النسخة لا يؤثر على المتغير الأصلي
- استدعاء بالمؤشر calling by reference وفيها يتم اخذ نسخة أيضا ولكن التغيير في النسخة يغير في المتغير الأصلي مباشرة , ويظهر عندما تكون وسائط الدالة من نوع المؤشرات
- فئات التخزين storage classes تحدد عمر المتغير lifetime الفترة التي سيبقي فيها في الذاكرة وأيضا مجال استخدام



Structures

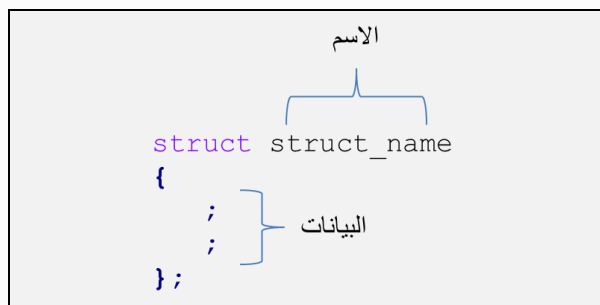
البنيات

- الـ structures

هي مجموعة من متغير او اكثر من انواع مختلفة , تجمع تحت اسم واحد لسهولة التعامل , بالتالي هي تسمح بتجميع مجموعة من المتغيرات المرتبطة مع بعضها من اجل التعامل معها كوحدة واحدة , عناصر البيانات الخاصة بكل structure تسمي حقول field او اعضاء members لتلك الـ structure

هي تختلف عن المصفوفات ان المصفوفة بها بيانات من نفس النوع , ويتم الاشارة لكل عنصر بالمصفوفة من خلال الترتيب الخاص بالعنصر داخل المصفوفة , اما الـ structure يمكن ان تحوي بيانات من انواع مختلفة , ويتم الاشارة لعناصرها باسم كل عنصر

الاعلان عن بنية الـ structure



الاعلان عن متغير من نوع structure

```
struct struct_name variable_name ;
```

- يجب ان نفرق بين الاعلان عن شكل وبنية الـ والاعلان عن متغير من نوعها

- الاعلان عن شكل الـ structure نقوم بتحديد عدد المتغير ونوع واسم كل متغير وترتيب وجودهم داخلها , ونحن بذلك لم نتعامل مع الذاكرة بعد
- الاعلان عن متغير من نوع structure معينة نقوم بحجز مساحة تخزينية في الذاكرة لها

امثلة

<pre>struct dimen { int height; int width; };</pre>	<pre>//declare box1 struct dimen box1; //declare box2 struct dimen box2;</pre>	<p>بنية لتخزين ابعاد شئ ما بها عنصرين الاول لتخزين الارتفاع والثاني لتخزين العرض</p>
<pre>struct coord {</pre>	<pre>//declare intial struct coord intial;</pre>	<p>بنية لتخزين احداثيات نقطة معينة في الفراغ , بها ثلاثة عناصر لتخزين الابعاد علي المحاور</p>

<pre>int x; int y; int z; };</pre>	<pre>//declare final struct coord final;</pre>	الثلاثة Z, Y, X
<pre>struct emp { char name[255]; float salary; };</pre>	<pre>//declare emp1 struct emp emp1; //declare emp2 struct emp emp2;</pre>	بنية اتخزين بيانات موظف , بها عنصرين احدهما لتخزين اسم الموظف والاخر راتبه

يتم الاشارة لكل عنصر داخل الـ structure من خلال اسم المتغير من نوع الـ structure متبوعا بنقطة (.) ثم اسم العنصر مثل

```
box1.height = 33 ;
box2.width = 23 ;
```

مثال توضيحي

```
#include<stdio.h>

//declare structure form
struct emp
{
    char name[100];
    float salary;
};

int main()
{
    //declare variable of type emp
    struct emp emp1;
    //get employee info from user
    printf("type employee name : ");
    gets(emp1.name);
    printf("type employee salary : ");
```

```
scanf("%f",&emp1.salary);
```

```
//print employee info
```

```
printf("employee name is %s , his salary is %.2f\n", emp1.name , emp1.salary );
```

```
return 0;
```

```
}
```

type employee name : omar - 140 -ohamed

type employee salary : 1234.5

employee name is omar - 140 -ohamed , his salary is 1234.50

[09 - structures/01.c]

قمنا بالاعلان عن شكل ال structure , تتكون من عنصرين الاول اسم الموظف والآخر الراتب

```
struct emp
```

```
{
```

```
char name[100];
```

```
float salary ;
```

```
};
```

حجزنا متغير من نوع struct emp

```
struct emp emp1;
```

اشرنا الي عنصر داخلها سواء للقراءة او الكتابة باستخدام اسم المتغير متبوعا بنقطة ثم اسم العنصر

```
emp1.name
```

```
&emp1.salary
```

- يمكن ان نجمع بين الاعلان عن البنية وحجز المتغير هكذا ويمكن وقتها الاستغناء عن اسم البنية

```
struct emp
```

```
{
```

```
char name[100];
```

```
float salary ;
```

```
} emp1 , emp2 ;
```

```
struct
```

```
{
```

```
char name[100];
```

```
float salary ;
```

```
} emp1 , emp2 ;
```

- يتم اعطاء قيم مباشرة لعناصر البنية باحدي طريقتين

- بالاشارة المباشرة لكل عنصر كما في المثال السابق `box1.height = 33`
- باستخدام قائمة التهيئة `initializer list` حيث توضع القيم داخل اقواس `{}` بنفس ترتيب العناصر داخل البنية كما يلي

```
struct emp emp1 = {"omar - 141 - ohamed", 1234.7};
```

- يمكن لبنية ان تحتوي علي عنصر من نوع بنية اخري `nested structure` كالمثال , برنامج لحساب مساحة مستطيل بمعرفة نقطتين متقابلتين بشكل قطري ولكل نقطة احداثيات `x, y`

```
//describe point structure
struct point
{
    int x;
    int y;
};

//describe rectangle points structure
struct rect
{
    //left lower point
    struct point p1;
    //right upper point
    struct point p2;
};

int main()
{
    struct rect r ;

    //left lower point ( 1 , 1 )
    r.p1.x = 1;
    r.p1.y = 1;
    //right upper point ( 3 , 4 )
    r.p2.x = 4;
    r.p2.y = 3;
    //width = x upper right – x lower left
    int width = r.p2.x – r.p1.x;
```

```
//height = y right upper – y lower left
```

```
int height = r.p2.y – r.p1.y;
```

```
printf(“rectangle area is %d \n”, width * height );
```

```
return 0;
```

```
}
```

```
rectangle area is 6
```

[09 - structures/02.c]

يتم الاشارة الي عناصر البنية الداخلية باستخدام النقطة ايضا (.)

```
r.p1.x = 1;
```

- البنيات والمؤشرات structures and pointers

يتم انشاء مؤشر الي بنية تماما مثل المؤشرات الاخرى , مثال

```
struct emp * ptr ;
```

عندما يكون لدينا مؤشر لبنية يتم الاشارة الي عناصر تلك البنية باستخدام الرمز (->) بدلا من (.) , بالطبع يمكننا استخدام (.) ولكن سنحتاج الي استخدام (*) كما هو موضح الفرق بين الطريقتين

```
struct  
{  
    char name[255];  
    float salary;  
} emp , *ptr;
```

```
ptr = &emp;
```

```
ptr->salary = 1234.5;
```

```
ptr->name = “omar - 142 -ohamed”;
```

```
(*ptr).salary = 1234.5;
```

```
(*ptr).name = “omar - 142 -ohamed”;
```

- البنيات والدوال structures and functions

- يمكن ان يكون لدينا دالة من نوع بنية معينة , فهي تقوم بارجاع قيمة من نوع تلك البنية , مثال دالة تقوم بانشاء بنية وارجاعها

```
//describe point structure
struct point
{
    int x;
    int y;
};

//declare getPoint function
struct point getPoint(int x,int y);

int main()
{
    struct point pt ;
    //call getPoint and assign returned value to pt
    pt = getPoint(12 , 44);
    printf("points are %d ,%d \n", pt.x , pt.y );
    return 0;
}

struct point getPoint(int x,int y)
{
    //create temporarily variable
    struct point temp ;
    temp.x = x;
    temp.y = y;
    return temp ;
};
```

points are 12 , 44

[09 - structures/03.c]

- يمكن لدالة ان تاخذ بنية كوسيط اما بالقيمة او بالمؤشر, مثال لدالة تقوم بطباعة محتوى بنية احدهما بالقيمة والاخرى بالمؤشر

```
//describe point structure
```

```
struct point
```

```
{
```

```
    int x;
```

```
    int y;
```

```
} pt;
```

```
void printPoint(struct point p)
```

```
{
```

```
    printf("points are %d,%d\n", p.x, p.y);
```

```
}
```

```
void printPoint(struct point *p)
```

```
{
```

```
    printf("points are %d,%d\n", p->x, p->y);
```

```
}
```

يفضل دائما ان نستخدم الاستدعاء بالمؤشر خاصة اذا كانت البنية كبيرة , لانه كما علمنا مسبقا ان الوسائط يتم اخذ نسخة منها داخل الدالة للعمل عليها , واذا كانت البنية كبيرة فهذا حتما يؤدي الي استهلاك الذاكرة

- مصفوفة بنيات array of structures

نستطيع ان ننشأ مصفوفة من البنيات تماما مثل مصفوفة المتغيرات الاخرى حيث يكون كل عنصر في المصفوفة عبارة عن بنية مثال

```
struct point shape[4];
```

مثال برنامج توضيحي

```
//describe book structure
```

```
struct book
```

```
{
```

```
    char author[100];
```

```
    char title[255];
```

```
    int year;
```

```
};
```

```
int main()
```

```
{
```

```
    //create array and initialize it
```

```

struct book books[3] = {
    {" - 145 -hilip sedgewick","algorithms in c",2000},
    {"peter van der linden","expert c programming",2002},
    {" - 145 -hilip - 145 -hilips","image processing in c",1999}
};

//print books

//first book
printf("%s By %s At %d\n",books[0].title,books[0].author,books[0].year);
//second book
printf("%s By %s At %d\n",books[1].title,books[1].author,books[1].year);
//third book
printf("%s By %s At %d\n",books[2].title,books[2].author,books[2].year);

return 0;
}

```

```

algorithms in c By - 145 -hilip sedgewick At 2000
expert c programming By peter van der linden At 2002
image processing in c By - 145 -hilip - 145 -hilips At 1999

```

[09 - structures/04.c]

- تسمية انواع المتغيرات باستخدام typedef

تتيح لنا السي اعادة تسمية انواع المتغيرات باسماء جديدة خاصة بنا باستخدام الكلمة المجوزة typedef ثم كاننا نقوم بحجز متغير من نوع معين وحيث يكون الاسم الجديد مكان اسم المتغير مثل

```
typedef int myInt;
```

اعلنا عن نوع جديد يسمى myInt يمكن استخدامه للاعلان عن متغيرات كالتالي

```

myInt number ;
myInt *ptr;
myInt arr[3];

```

```
int number ;
int *ptr;
int arr[3];
```

فقط التغير في المسميات ولكن تبقي الوظيفة كما هي , من الاستخدامات الهامة ان نعطي اسماء للبنيات التي نعلن عنها فبدلا من كتابة كلمة struct ثم اسم البنية ثم اسم المتغير نعطي لها تسمية جديدة ونستخدمها لدينا مثالين احدهما ب typedef والاخر بدونها

```
struct emp
{
    char name[255];
    float salary;
};
//create new type name
typedef struct emp empRecord;

empRecord emp1;
empRecord emp2;
```

```
struct emp
{
    char name[255];
    float salary;
};

struct emp emp1;
struct emp emp2;
```

من اهم استخدامات typedef

- لو استخدمت مع متغيرات يتغير حجمها من جهاز لآخر حسب بنية المعالج الخاص بالجهاز , سنقوم بالتعديل علي typedef فقط ضبط التغير في حجم المتغير
- ان يكون مسميات لها معني ف empRecord لها معني اكثر من struct emp

- البنية باستخدام unions

هي مثل ال structure في انها تحوي عناصر من انواع مختلفة , ولكن المساحة التخزينية التي يتم حجزها ليست مجموع مساحات عناصرها وانما مساحة العنصر الاكبر , ولذا فان القيمة الموجودة بها في لحظة معينة تخص فعليا احد عناصرها وليس كل العناصر , يمكن النظر اليها كالاناء الذي يمكن ان يحوي اكثر من نوع من السوائل ولكن في لحظة معينة يحوي شئ واحد فقط ماء مثلا لنري المثال التالي

```
//declare union form
union value
{
    int x;
```

```

int y;
};

int main()
{
    union value v1;
    v1.x = 20;
    printf("value of x is %d \n", v1.x);
    printf("value of y is %d \n\n", v1.y);

    v1.y = 50;
    printf("value of x is %d \n", v1.x);
    printf("value of y is %d \n\n", v1.y);

    return 0;
}

```

value of x is 20
value of y is 20

value of x is 50
value of y is 50

[09 - structures/05.c]

في البداية اعلنا عن شكل ال union تماما مثل ال structure

```

union value
{
    int x;
    int y;
};

```

المحتوي هنا للتبسيط لمتغيرين من نفس النوع , المساحة التخزينية لمتغير من نوع union value ليست 8 بايت وانما 4 بايت كما قلنا تاخذ مساحة العنصر الاكبر وهنا العنصرين من نفس النوع , قمنا بعد ذلك بحجز متغير وتغير قيمة x

```
v1.x = 20 ;
```

عند طباعة محتوى العنصرين تم طباعة نفس القيمة لانه كما قلنا القيمة الموجودة في لحظة معينة تخص احد العناصر فقط , فعند طباعة y اخذت القيمة الموجودة والعكس حدث بعد ذلك فقد قمنا بتغيير قيمة y وطباعة محتوى العنصرين فتم ايضا طباعة نفس القيمة , النتائج في هذا المثال كانت مفهومة لان العنصرين من نفس النوع لنري المثال التالي

```
//declare union form
union value
{
    int x;
    char ch;
};

int main()
{
    union value v1;
    v1.x = 20 ;
    printf("value of x is %d \n", v1.x);
    printf("value of ch is %c \n\n", v1.ch);

    v1.ch = 'A' ;
    printf("value of x is %d \n", v1.x);
    printf("value of ch is %c \n\n", v1.ch);

    return 0;
}
```

value of x is 20

value of ch is  

value of x is 65

value of ch is A

[09 - structures/06.c]

هنا النتائج مختلفة لاننا لدينا نوعين مختلفين , المساحة التخزينية هنا للمتغير v1 ستكون 4 بايت وليس 5 بايت

- يمكن معرفة الحجم الذي يشغله اي متغير في الذاكرة باستخدام المؤثر sizeof الذي يعيد الحجم بالبايت لاي نوع متغير او المتغير ذاته مثال

<code>sizeof(int);</code>	4
<code>sizeof(char);</code>	1
<code>sizeof(union value);</code>	4

- البيانات المعروفة قيمها مسبقا enum

تستخدم في الحالات التي نعرف كل القيم التي يمكن ان ياخذها المتغير, فلو ان لدينا متغير لتخزين اليوم فنحن نعرف مسبقا ان هذا المتغير قيمته هي احد تلك القيم فقط (السبت , الاحد , الاثنين , الثلاثاء , الاربعاء , الخميس , الجمعة) كالتالي

```
enum days{ SATURDAY ,SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};  
enum days today = MONDAY ;
```

انشانا لكل القيم التي يمكن ان ياخذها المتغير, وانشانا متغير من نفس النوع, لست مضطرتلك الطريقة هي فقط الافضل والاكثر وضوحا وتنظيما
مثال اخر لنفرض ان لدينا متغير يحمل تقدير طالب, فاحتمالات القيم هي (امتياز, جيد جدا, جيد, مقبول) لذا التعريف سيكون

```
enum grades {EXCELLENT , VERYGOOD , GOOD , PASS};
```

والمتغير

```
enum grades myGrade = VERYGOOD ;
```

بقي ان نعرف ان المترجم يعطي كل واحد من تلك القيم عدد صحيح داخليا ليفرق بينها, وهذا العدد يبدأ من صفر ثم 1 وهكذا, بل ويمكننا ان نعطي نحن تلك القيم مباشرة

```
enum grades {EXCELLENT = 12, VERYGOOD , GOOD , PASS};
```

القيم التي تليها ستأخذ 13 ثم 14 ثم 15

- ضغط حجم البنية packed structures or bit fields

تسمح لنا تلك الطريقة باعلان عن بنية ما بحيث تأخذ اقل حجم في الذاكرة, حيث نقوم بتحديد عدد البتات الكافية لكل عنصر داخلها, بدلا من ترك العناصر بحجمها الافتراضي, يتم ذلك باضافة نقطتين (:) بعد اسم العنصر ثم عدد البتات المطلوبة كالآتي

```

struct status
{
    int isEven:1; //1 if even ,0 if odd
    int isZero:1; //1 if zero ,0 if non-zero
    int isPrime:1; //1 if prime number or 0 if not
    int isPositive: 1; //1 if +ve or 0 if -ve
    char number ; //number from -128 to 127
};

```

ستأخذ مساحة اقل من لو لم نستخدم packing

```

struct status
{
    int isEven ;//1 if even ,0 if odd
    int isZero ;//1 if zero ,0 if non-zero
    int isPrime ;//1 if prime number or 0 if not
    int isPositive;//1 if +ve or 0 if -ve
    char number ; //number from -128 to 127
};

```

يفضل استخدامه مع الاعلام flags والتي تأخذ احد قيمتين اما true او false وكل منها لها معنى حسب ال flag فمثلا isEven هذا علم علي الرقم اذا كان زوجي او فردي

الخلاصة

- ال structures تسمح بتجميع مجموعة من المتغيرات المرتبطة مع بعضها من اجل التعامل معها كوحدة واحدة
- يجب ان نفرق بين الاعلان عن شكل وبنية ال والاعلان عن متغير من نوعها
- يتم انشاء مؤشر الي بنية تماما مثل المؤشرات الاخرى
- تستخدم enum في الحالات التي نعرف كل القيم التي يمكن ان ياخذها
- ال unions هي مثل ال structure في انها تحوي عناصر من انواع مختلفة , ولكن المساحة التخزينية التي يتم حجزها ليست مجموع مساحات عناصرها وانما مساحة العنصر الاكبر

10

Files

الملفات

- **الملف** هو مجموعة من البيانات المرتبطة مع بعضها ,تتعامل السي مع اي ملف علي انه مجموعة من البايتات المتتالية series of bytes

تحتوي السي علي مجموعة من الدوال للتعامل مع الملفات , موجودة في الملف الراسي stdio.h

التعامل مع اي ملف يحتاج 4 خطوات

1. انشاء مؤشر من نوع ملف , يستخدم للإشارة للملف الذي يتم التعامل معه

2. فتح الملف سواء للقراءة او الكتابة او الاثنين معا

3. القراءة او الكتابة للملف

4. اغلاق الملف

- الاعلان عن مؤشر من نوع ملف : يتم ذلك بالكلمة المحجوزة FILE فعليا هذا المؤشر يشير الي بنية struct والتي تحوي معلومات عن الملف مثل عنوان ال buffer , موقع الحرف الحالي داخل ال buffer , هل الملف للقراءة ام للكتابة , هل هناك اخطاء وهكذا كالتالي

```
FILE * filePtr;
```

- فتح الملف يتم باستخدام الدالة fopen والتي تاخذ وسيطين الاول اسم الملف المراد فتحه , الثاني كيف سنستخدم الملف هل للقراءة او الكتابة, وتقوم بارجاع مؤشر للملف او القيمة NULL في حالة حدوث اخطاء مثل عدم وجود الملف مثلا

```
filePtr = fopen("test.txt", "r");
```

- القراءة والكتابة للملف باحد الدوال الخاصة بذلك

- اغلاق الملف بالدالة fclose , وتاخذ وسيط هو مؤشر الملف المراد اغلاقه

```
fclose(filePtr);
```

- قراءة وكتابة الحروف

تستخدم الدوال fputc , fgetc لقراءة وكتابة الحروف للملفات وهما كما بالشكل

<pre>int fputc(int c , FILE *stream);</pre> <div><div>الحرف الذي تم كتابته</div><div>الحرف المراد كتابته</div><div>مؤشر الملف</div></div> <div>او EOF في حالة حدوث خطأ</div>	<pre>int fgetc(FILE *stream);</pre> <div><div>الحرف التالي في القراءة</div><div>مؤشر الملف</div></div> <div>او EOF في حالة حدوث خطأ او نهاية الملف</div>
كتابة حرف في ملف	قراءة حرف من ملف

```
//char array
char ch[18]={ 'w','e','l','c','o','m','e',' ','t','o',' ','c',' ','f','i','l','e','s' };
//file pointer
FILE *filePtr;
//open file for writing
filePtr = fopen("01.txt", "w");
//check if error occur
if(filePtr == NULL)
{
    printf("can't open file /n");
}
else
{
    //loop index
    int i;
    for(i=0;i<18;i++)
    {
        //write char by char to file
        fputc(ch[i],filePtr);
    }

    printf("file written successfully \n");
    //close file
    fclose(filePtr);
}
```

[10 - files/01.c]

يبدأ الكود بإنشاء مصفوفة الحروف وإعطائها قيم مباشرة , قم بإنشاء مؤشر لملف

```
FILE *filePtr;
```

استخدمنا الدالة fopen لفتح الملف للكتابة حيث الوسيط الثاني "w" الذي يستخدم في حالة الكتابة

```
filePtr = fopen("01.txt", "w");
```

- من الهام هنا ان نعلم انه في حالة فتح ملف للكتابة , اذا كان الملف غير موجود , فسيتم انشاء الملف ان امكن , واذا كان موجود سيتم الكتابة فوق محتوياته , اي ان البيانات السابقة المخزنة بالملف ستفقد لا لذا اسخدمنا "a" بدلا من "w" والتي تقوم بالاضافة الي اخر الملف وليس الكتابة فوق محتوياته

في حالة حدوث خطأ في فتح الملف لاي سبب تقوم الدالة fopen بإرجاع القيمة NULL ولذا لابد من التأكد ان الملف تم فتحه بدون اخطاء

```
if(filePtr == NULL)
{
    printf("can't open file /n");
}
```

الاكود الاخري الخاصة بالكتابة علي الملف توضع في الحالة الاخري من الشرط لاننا لانستطيع التعامل مع ملف حدث خطأ في فتحه ولان مؤشر الملف ستكون قيمته NULL وبالتالي لا نستطيع التعامل معه ايضا

```
else
{
    //loop index
    int i;
    for(i=0;i<18;i++)
    {
        //write char by char to file
        fputc(ch[i],filePtr);
    }

    printf("file written successfully \n");
    //close file
    fclose(filePtr);
}
```

تستخدم الدالة fputc لكتابة حرف لملف حيث تاخذ وسيطين الاول هو الحرف والثاني مؤشر الملف

```
fputc(ch[i],filePtr);
```

واخير اغلقنا الملف بعد الانتهاء من عملية الكتابة

```
fclose(filePtr);
```

```
//file pointer
FILE *filePtr;

//open file for writing
filePtr = fopen("01.txt", "r");

//check if error occur
if(filePtr == NULL)
{
    printf("can't open file /n");
}
else
{
    char ch;
    //loop till end of a file
    while(1)
    {
        //read char
        ch = fgetc(filePtr);
        //check for end of file
        if(ch == EOF)
        {
            break;
        }
        printf("%c",ch);
    }
    //close file
    fclose(filePtr);
}
```

welcome to c files

تم فتح الملف واستخدمنا "r" للدلالة علي اننا نفتح الملف للقراءة ثم بعد تم التحقق من عملية فتح الملف اذا كانت بها اخطاء اولاً, استخدمنا الدالة fgetc لقراءة حرف من الملف وتأخذ وسيط هو مؤشر الملف وترجع قيمة الحرف وتستمر تلك الدالة في القراءة حتي نهاية الملف وعندها ترجع القيمة EOF وهي قيمة ثابتة لذا نقارن بها للتحقق من نهاية الملف

استخدمنا حلقة غير منتهية لاننا لا نعلم مسبقا عدد الحروف في الملف , ولكن استخدمنا الكلمة المحجوزة break للخروج من الحلقة عند الوصول لنهاية الملف

```
char ch;
//loop till end of a file
while(1)
{
    //read char
    ch = fgetc(filePtr);
    //check for end of file
    if(ch == EOF)
    {
        break;
    }
    printf("%c",ch);
}
```

اخیر اغلقنا الملف بعد انتهاء قراءة محتوياته

١٠ - كتابة وقراءة النصوص

تستخدم الدوال `fgets` , `fputs` لقراءة وكتابة النصوص وهما كما بالشكل

<pre>int fputs(const char * string, FILE * stream);</pre> <p> { { { </p> <p> EOF في حالة حدوث خطأ مؤشر النص المراد كتابته مؤشر الملف </p>	<pre>char * fgets(char * dst, int max, FILE * stream);</pre> <p> { { { { </p> <p> مؤشر للنص الذي تم قراءته أو null في حالة حدوث خطأ أو نهاية الملف مؤشر لمكان تخزين النص حجم النص المراد قراءته مؤشر الملف </p>
كتابة نص في ملف	قراءة نص من ملف

```
//string array
char *colors[6] = { "white", "blue", "red", "green", "yellow", "black" };
//file pointer
FILE *filePtr;
//open file for writing
filePtr = fopen("02.txt", "w");
//check if error occur
if(filePtr == NULL)
{
    printf("can't open file /n");
}
else
{
    int i;
    for(i=0; i<6; i++)
    {
        //write string in file
        fputs(colors[i], filePtr);
        //add space between each string
        fputc(' ', filePtr);
    }
    printf("file written successfully \n");
    //close file
    fclose(filePtr);
}
```

[10 - files/03.c]

يبدأ البرنامج بإنشاء مصفوفة النصوص ثم مؤشر للملف , بعد ذلك فتح الملف للكتابة , بعد التحقق من مؤشر الملف استخدمنا الدالة fputs لكتابة عناصر المصفوفة من النصوص واحد تلو الآخر وبينهما مسافة باستخدام الدالة fputc

```
//string array
char str[50];
//file pointer
FILE *filePtr;
//open file for reading
filePtr = fopen("02.txt", "r");
//check if error occur
if(filePtr == NULL)
{
    printf("can't open file /n");
}
else
{
    //loop until reach end of file or error occur
    while( fgets( str , sizeof(str), filePtr ) )
    {
        printf("%s",str);
    }
    //close file
    fclose(filePtr);
}
```

white blue red green yellow black

[10 - files/04.c]

يبدأ الكود بحجز متغير نصي يستخدم أثناء القراءة من الملف ,وبعد فتح الملف للقراءة والتحقق من مؤشر الملف , استخدمنا الدالة لقراءة النصوص من الملف حيث تأخذ 3 وسائط الأولى المتغير النصي الذي سيوضع به البيانات , الثاني أقصى حجم للقراءة حيث تقوم الدالة بالقراءة من الملف حتي هذا العدد -1 وذلك للمرة الواحدة , فلو ان هذا الوسيط 3 مثلاً سيتم قراءة 3 حروف في كل مرة الي ان ينتهي الملف , واخر وسيط هو مؤشر الملف , في حال اعطاء الدالة حجم كبير ستقوم بالتوقف عند نهاية السطر (\n) اي انها في كل مرة اما تقرا العدد المسموح او تقرا حتي نهاية السطر حتي اذا كان عدد حروف السطر اقل بكثير من العدد المطلوب لذا تعرف هذه الدالة انها خاصة بقراءة السطور

ترجع هذه الدالة NULL في حالة ان الوصول الي اخر الملف او حدث خطأ لذا استخدمناها في شرط التحقق مباشرة

- كتابة وقراءة الاعداد الصحيحة والحقيقية

تماما مثل التعامل مع الشاشة , حيث لايمكننا طباعة قيمة متغير صحيح او حقيقي مباشرة لابد من استخدام printf , مع الملفات نستخدم الدالة fscanf بنفس الطريقة تماما وفي حالة القراءة نستخدم fscanf

<pre>int fscanf(FILE * stream, const char * format, ...);</pre> <p>عدد القيم التي تم قراءتها او EOF في حالة الخطأ</p> <p>مؤشر الملف</p> <p>التنسيق</p> <p>وسائط التنسيق</p>	<pre>fprintf(FILE * stream, const char * format, ...);</pre> <p>مؤشر الملف</p> <p>التنسيق</p> <p>وسائط التنسيق</p>
fscanf	fprintf

برنامج لكتابة محتوى مصفوفة اعداد صحيحة في ملف "03.txt"

```
//numbers array
int nums[6] = {23, 54, 78, 91, 11, 38};

//file pointer
FILE *filePtr;

//open file for writing
filePtr = fopen("03.txt", "w");

//check if error occur
if(filePtr == NULL)
{
    printf("can't open file /n");
}
else
{
    int i;
    for(i=0;i<6;i++)
    {
        //write number to
        fprintf(filePtr, "%d ", nums[i]);
    }
    printf("file written successfully \n");
    //close file
    fclose(filePtr);
}
```

[10 - files/05.c]

تحتاج الدالة fprintf الي 3 وسائط الاول مؤشر الملف والثاني والثالث مثل printf تماما

- برنامج لقراءة محتوى الملف كاعداد صحيحة وطباعته

```
//file pointer
FILE *filePtr;

//open file for writing
filePtr = fopen("03.txt","r");

//check if error occur
if(filePtr == NULL)
{
    printf("can't open file /n");
}
else
{
    //store read number
    int num;

    //fscanf return value
    int ret ;

    while(1)
    {
        //read number at a time
        ret = fscanf(filePtr,"%d",&num);
        if(ret == 0 || ret == EOF)
        {
            break ;
        }
        printf("%d ",num);
    }

    //close file
    fclose(filePtr);
}
```

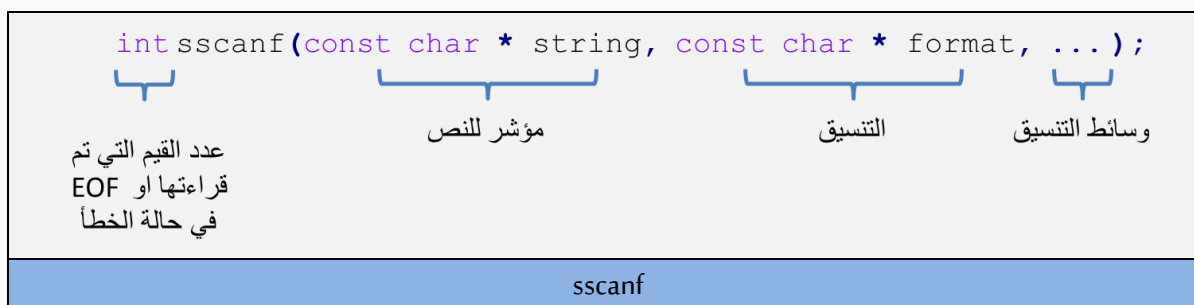
23 54 78 91 11 38

[10 - files/06.c]

تستخدم الدالة fscanf تماماً مثل الدالة scanf غير انها تحتاج الي مؤشر للملف كوسيط اول , القيمة التي ترجعها تلك الدالة اما EOF نهاية الملف او صفراً والتي تعني وجود بيانات ولكن لا يمكن تحويلها للنوع المطلوب , او عدد القيم التي استطاع تحويلها

- استخراج بيانات من نص

بدلاً من ان يتم قراءة قيم من لوحة المفاتيح او من ملف , يمكن ان نقرا قيم من نص وذلك باستخدام الدالة sscanf



```
//string
char *str = "24 30";
//nums variables
int first, second;
//sscanf return value
int ret;
//calling sscanf
ret = sscanf(str, "%d %d", &first, &second);
if (ret == 2)
{
    printf("numbers are %d , %d\n", first, second);
}
else
{
    puts("error!");
}
```

numbers are 24 , 30

[10 - files/07.c]

يبدأ الكود بحجز مؤشر لنص ثابت به قيمة ثابتة عبارة عن رقمين ونريد استخراج هذين الرقمين , استخدام الدالة التي تأخذ 3 وسائط الأول النص , الثاني الصيغة او الانواع المراد التحويل اليها وفي حالتنا عددين صحيحين "%d %d" والثالث عناوين المتغيرات التي سيوضع فيها القيم ان وجدت , تلك الدالة مثل fscanf في القيمة التي ترجعها اما EOF او صفر او عدد القيم التي تم تحويلها بنجاح

- انواع الملفات

1. ملفات الاسكي او الملفات النصية ASCII files تتكون تلك الملفات من مجموعة من الحروف المتتالية , تستخدم تلك الملفات لتخزين البيانات النصية ومثل التي تعاملنا معها في الامثلة السابقة امثلة ملفات الاكواد , ملفات النصوص (.txt).
2. الملفات الثنائية binary files تحتوي علي مجموعة من البايتات المتتالية , تستخدم لتخزين الملفات الغير نصية امثلة ملفات البرامج , ملفات الصور بانواعها , ملفات الفيديو , ملفات الصوت

-انماط التعامل مع الملفات

النمط	الوصف
"r"	فتح ملف موجود بالفعل للقراءة منه
"w"	انشئ ملف جديد ان لم يكن موجود وتجاهل محتوياته
"a"	انشئ ملف ان لم يكن موجود , لكن اكتب من نهايته , ابق محتوياته
"r+"	افتح ملف موجود لتحديث محتوياته سواء للقراءة او الكتابة
"w+"	انشئ ملف ان لم يكن موجود , تجاهل المحتوي , للقراءة والكتابة ايضا
"a+"	انشئ ملف ان لم يكن موجود , اكتب من نهايته , للقراءة والكتابة ايضا
"rb"	مثل "r" ولكن مع الملفات الثنائية binary files
"wb"	مثل "w" ولكن مع الملفات الثنائية binary files
"ab"	مثل "a" ولكن مع الملفات الثنائية binary files
"rb+"	مثل "r+" ولكن مع الملفات الثنائية binary files
"wb+"	مثل "w+" ولكن مع الملفات الثنائية binary files
"ab+"	مثل "a+" ولكن مع الملفات الثنائية binary files

- الملفات الثنائية binary files

يتم التعامل مع هذا النوع من الملفات من خلال دالتين هما fread , fwrite

<pre>size_t fwrite(void * src, size_t size, size_t num, FILE * stream);</pre>	<pre>size_t fread(void * dst, size_t size, size_t num, FILE * stream);</pre>
<p>مؤشر الملف</p> <p>عدد العناصر التي تم كتابتها</p> <p>مؤشر لمكان البيانات</p> <p>حجم العنصر الواحد</p> <p>عدد العناصر التي ستكتب</p>	<p>مؤشر الملف</p> <p>عدد العناصر التي سيتم قراءتها</p> <p>حجم العنصر الواحد</p> <p>مؤشر لمكان تخزين البيانات بعد قراءتها</p> <p>عدد العناصر التي تم قراءتها</p>
fwrite	fread

```
//declare structure book
struct book
{
    char author[100];
    char title[100];
    short year ;
};

int main()
{
    struct book b = { "Leland L.Beck" , "System Software" , 1997 };

    //open file ,for writing in binary mode
    FILE *filePtr = fopen("04.dat","wb");

    //check for file open error
    if(filePtr == NULL)
    {
        puts("can't open file ");
    }
    else
    {
        //write b variable as memory block to file
        fwrite( &b ,sizeof(b),1,filePtr);

        puts("data was written successfully ");

        fclose(filePtr);
    }
    return 0;
}
```

في البداية اعلنا عن شكل البنية , ثم حجزنا متغير من نوع البنية struct book واعطيناه قيم مباشرة

```
struct book b = { "Leland L.Beck" , "System Software" , 1997 };
```

قمنا بفتح الملف للكتابة واستخدمنا "wb" لاننا سنتعامل مع الملف كملف ثنائي binary file

```
FILE *filePtr = fopen("04.dat","wb");
```

وبعد التحقق من عدم وجود الخطاء في فتح الملف استخدمنا الدالة للكتابة بالملف والتي تأخذ 4 وسائط , الاول مؤشر للبيانات المراد كتابتها , الوسيط الثاني حجم العنصر الواحد من تلك البيانات في حالتنا استخدمنا sizeof(b) لايجاد حجم المتغير , الوسيط الثالث عدد العناصر المراد كتابتها واخيرا مؤشر الملف

```
fwrite( &b,sizeof(b),1,filePtr);
```

لو ان لدينا مصفوفة اعداد صحيحة بها 10 عناصر ستكتب الدالة كما يلي

```
fwrite(array , sizeof(int) , 10 , filePtr);
```

مثال برنامج لقراءة البنية من الملف السابق وطباعة محتواها

```
//declare structure book
struct book
{
    char author[100];
    char title[100];
    short year;
};

int main()
{
    struct book b;

    //open file ,for writing in binary mode
    FILE *filePtr = fopen("04.dat","rb");

    //check for file open error
    if(filePtr == NULL)
    {
        puts("can't open file ");
    }
}
```

```

}
else
{
    //read binary block from file , size is size of b struct
    fread(&b , sizeof(b) , 1 , filePtr);
    fclose(filePtr);

    //print structure content
    printf("book : %s , %s , %d \n",b.title ,b.author ,b.year);
}
return 0;
}

```

book : System Software , Leland L.Beck , 1997

[10 - files/09.c]

في البداية حجزنا متغير لتخزين البيانات التي سيتم قراءتها , قمنا بفتح الملف للقراءة بالنمط الثنائي "rb"

```
FILE *filePtr = fopen("04.dat","rb");
```

استخدمت الدالة للقراءة من الملف حيث تأخذ 4 وسائط الاول مؤشر للمتغير الذي سيتم تخزين البيانات فيه , الثاني حجم العنصر , الثالث عدد العناصر , الرابع مؤشر لملف , تقوم تلك الدالة بارجاع الحجم الذي تم قراءته , فاذا كان اقل من حجم العنصر الواحد فهذا يعني حدوث خطأ او نهاية الملف

```
fread(&b , sizeof(b) , 1 , filePtr);
```

- طرق الوصول لملف file access

هناك طريقتين للوصول الى بايت معين او مكان معين داخل ملف

- الوصول المتتالي sequential access وهو مثل شريط الكاسيت عندما تريد الوصول الى موقع معين position في الشريط فانك تتحرك من حيث انت بشكل متتابع حتي تصل الي حيث تريد , اي اننا لانذهب اليها مباشرة وهو مثل الكتاب الذي ليس به فهرس للعناوين يجب ان تمر علي كل الصفحات من البداية حتي تصل الي ما تريد
- الوصول العشوائي random access عكس السابق حيث يمكن في هذا النوع الوصول لاي مكان مباشرة ويجدر الاشارة هنا الي RAM (random access memory) والتي تعرف بذاكرة الوصول العشوائي وذلك لان كل بايت لديه عنوان خاص به وبالتالي يمكن الوصول لاي بايت مباشرة بمعرفة عنوانه

- لكل ملف يتم التعامل معه في السي مؤشر داخلي يشير الى البايت الذي سيتم قراءته او الكتابة فيه , يمكن تحريك هذا المؤشر باستخدام الدالة fseek وهي تأخذ 3 وسائط الاول مؤشر للملف , والثاني مقدار الازاحة offset والآخر البداية التي سيتم الازاحة منها وهي تأخذ احد 3 قيم ثابتة هي

- SEEK_SET من بداية الملف
- SEEK_CUR من الموقع الحالي في الملف
- SEEK_END من نهاية الملف

ولذا امكننا الوصول الى الملف بشكل عشوائي نوعا ما لاننا نعرف مسبقا حجم البيانات المخزنة داخل الملف وبممكن تحريك المؤشر الى اي مكان نريد لقراءة بيانات معينة

هناك الدالة rewind التي ترجع بالمؤشر مباشرة الى اول الملف وتحتاج الى مؤشر الملف فقط كوسيط

```
rewind( filePtr );//seek file position to the beginning
```

امثلة علي تحريك المؤشر

الكود	الوصف
fseek (filePtr , 4 , SEEK_SET);	حرك المؤشر بمقدار 4 بايت من بداية الملف
fseek (filePtr , 5 , SEEK_CUR);	حرك المؤشر بمقدار 5 بايت من الموقع الحالي للمؤشر
fseek (filePtr , -5 , SEEK_END);	ارجع المؤشر 5 بايت من نهاية الملف

- الملفات القياسية في السي standard files

من المهم ان نعرف ان السي تتعامل مع الاجهزة من خلال الملفات , كما هو الحال في نظام الينكس , ويوجد ثلاث ملفات قياسية في السي

الملف	الوصف
Standatd input (stdin)	وحدة الخرج الاساسية وهي الشاشة
Standatad ouput (stdout)	وحدة الادخال الاساسية وهي لوحة المفاتيح
Standard error (stderr)	وحدة الخرج للاخطاء وغالبا تكون الشاشة

ومن هنا يجب ان نعرف انه يمكننا ان نستخدم اي دالة من دوال التعامل مع الملفات الاسكي في الخرج للطباعة علي الشاشة واستقبال القيم من المستخدم مع استخدام احد الملفات القياسية بدل مؤشر الملف

-

استقبال حرف من المستخدم باستخدام fgetc

```
char ch ;  
ch = fgetc(stdin);
```

- طباعة نص باستخدام fputs

```
fputs("hello",stdout);
```

- طباعة قيم متغيرات صحيحة وحروف

```
int num =34;  
char ch = 'A';  
fprintf( stdout , "%d , %c" , num , ch );
```

- استقبال قيم صحيحة من المستخدم

```
int num ;  
fscanf(stdin , "%d" , &num);
```

- استقبال نص من المستخدم

```
char str[20];  
fgets(str,sizeof(str),stdin);
```

- طرق الوصول لوحدات الادخال والاخراج I/O devices access

1. ال high level I/O او Buffered I/O

وفيه لا يتم الكتابة بشكل مباشر الي الملف , انما تحفظ البيانات في ال buffer حتي يصبح به حجم كافي للكتابة مرة واحدة او يتم عمل flush من خلال الدالة fflush, لان عملية الوصول I/O devices access للاجهزة الخاصة بالاخراج والادخال مثل القرص الصلب مثلا تاخذ وقتا , فان هذا الطريقة تساعد علي تقليل عدد المرات الوصول لتلك الاجهزة يمكن ان تفكر به كمركب لنقل الناس بين ضفتي نهر لا يتحرك الا اذا كان هناك عدد مناسب بالمركب او في الحالات الضرورية بصرف النظر عن العدد

2. ال low level I/O او Unbuffered I/O

وفيه يتم الكتابة بالملف بشكل فوري

جميع عمليات الدخول والخروج بشكل افتراضي Buffered I/O واذا اردنا ان نستخدم طريقة الوصول الاخرى هناك دوال اخرى لذلك.

- تتعامل السي مع اي ملف علي انه مجموعة من البايتات المتتالية series of bytes
- يوجد نوعين من الملفات ملفات الاسكي او الملفات النصية ASCII files و الملفات الثنائية binary files

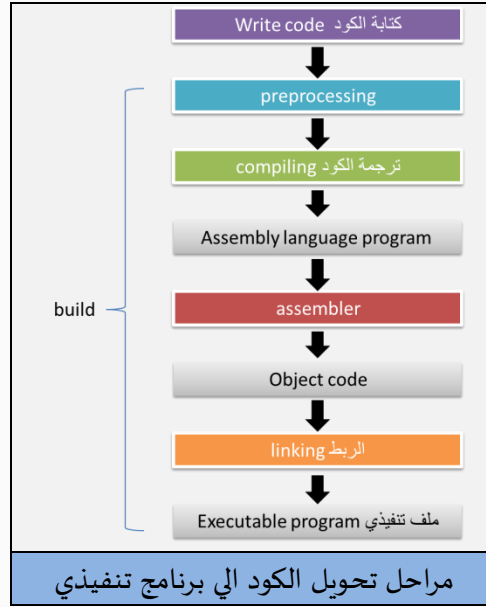
11

Preprocessor

التوجيهات

تكتب البرمجيات عادة بلغات برمجة مفهومة لنا كبشر ولكن الحاسب الالى لا يفهم تلك اللغة , وانما يفهم لغة الالة machine language والتي تتكون من اصفار وواحد , ولكي يتم تنفيذ البرامج علي الحاسب يمر البرامج بمجموعة من المراحل هي

1. كتابة الاكواد write codes
2. ما قبل ترجمة الكود preprocessing
3. ترجمة الكود compiling
4. تحويل الي لغة الة assembler
5. الربط linking



من المهم ان نوضح انه ليس علينا ان نقوم بتنفيذ تلك المراحل خطوة خطوة حسب الادوات الخاصة بكل مرحلة وذلك لوجود ادوات تسهل ذلك وهي IDE بيئة التطوير المتكاملة التي تحتوي علي كل الادوات المطلوبة وذلك في اداة واحدة امثلة علي تلك الادوات code block,eclipse , visual studio فنحن عندما نكتب الكود ونضغط build تقوم الاداة بالمراحل جميعها بشكل تلقائي , وعلي الرغم من ذلك الا اننا نحتاج ان نفهم تلك المراحل وما يحدث فيها

1. كتابة الاكواد write codes
تتم تلك المرحلة بعد ان يكون لدينا تصور واضح للبرنامج المراد كتابته ولدينا تصميم للبرنامج , وتكتب الاكواد باستخدام اي محرر نصوص مثل الـ ++notepad علي الويندوز او vi,nano علي اللينكس او اذا تم استخدام IDE فان يحتوي علي محرر اكواد خاص به
2. ما قبل ترجمة الكود preprocessing
عبارة عن تعليمات خاصة للمترجم تنفذ قبل البدء في الترجمة وهي مثل اضافة الملفات الراسية باستخدام كلمة include حيث تتم تلك المرحلة قبل ترجمة الكود وتسمي ايضا بالتوجيهات directives
3. ترجمة الكود compiling

وهنا يتم تحويل الكود من لغة السي عالية المستوى الى كود مكتوب بلغة الاسبلي منخفضة المستوى

4. تحويل الى لغة الة assembler

هنا يتم تحويل الكود المكتوب بالاسبلي الى لغة الالة ويسمي ايضا object file حيث ياخذ الامتداد التالي (.o) في الويندوز , ولكن هذا الكود علي الرغم من انه اصبح بلغة الالة الا انه غير جاهز للتنفيذ , لان هذا المرحلة تتم لكل ملف بشكل منفصل

5. الربط linking

كل ملف من ملفات البرنامج يحتوي علي متغيرات ودوال معرفة في ملفات اخري داخل البرنامج , هنا ييم الربط بين كل الملفات وبعضها بحيث ينتج ملف واحد يتم تشغيل البرنامج من خلاله وهو الملف التنفيذي executable file وياخذ الامتداد (.exe) في الويندوز لتشغيل البرنامج بعد ذلك يستخدم المحمل loader الذي يقوم بنقل محتويات البرنامج الي ذاكرة الحاسب ليتم تشغيله

- التوجيهات directives or preprocessors

هي تعليمات للمترجم قبل البدء بعملية الترجمة , وهذه التعليمات تبدأ دائما بالرمز (#) وتوضع في اول الملف عادة , وكل توجيه يكتب في سطر خاص به ولا ينتهي بفاصلة منقوطة لانه ليس من تعليمات اللغة

- التوجيه #include

يقوم بضم محتوي ملفات راسية الي ملف الكود , حيث تحتوي تلك الملفات علي ثوابت وتعريفات دوال خاصة وظيفه معينة ويستخدم بطريقتين

- الاولى ان يكتب اسم الملف بين علامتي <> وهنا نخبر المترجم ان هذا الملف هو احد الملفات الراسية الموجودة مع المترجم في المجلد include غالبا

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

- الثانية ان يكتب اسم الملف بين علامتي "" عندما يكون الملف ضمن ملفات المشروع حيث يكتب الملف بالمسار اذا كان داخل مجلد تحت مجلد المشروع

```
#include "headers/node.h"
```

- التوجيه #define

هذا التوجيه له استخدامين

- تعريف الثوابت كما علمنا سلفا, مثل

```
#define MAX 20
```

- تعريف المختصرات macros كما علمنا مثل

```
#define max(a,b) ((a > b)? a : b)
```

- التوجيه #undef

هذا التوجيه عكس السابق تماما فلو عرفنا ثابت ثم الغينا تعريف بهذا التوجيه يعتبر الثابت غير موجود مثال

```
#define MAX 20
#undef MAX
int main()
{
    printf("%d",MAX);
    return 0;
}
```

هنا خطأ لاننا بعد ان عرفنا الثابت MAX الغينا تعريف مرة اخرى وبالتالي فهو مجهول بالنسبة للمترجم وسيعطي رسالة خطأ انه غير معرف

- التوجيهات #if, #else, #elif, #endif

تستعمل في الترجمة الشرطية conditional compilation وهي مثل جملة الشرط if ولكن هذه تنفذ قبل الترجمة للكود مثل

```
#define LEVEL 0
#if LEVEL == 0
    #define MAX 10
#elif LEVEL == 1
    #define MAX 20
#else
    #define MAX 30
#endif
```

- التوجيهان #ifdef, #ifndef

يستخدم للتحقق اذا كان ثابت او مختصر معين معرف ام لا , مثل استخدامه لاضافة ملفات راسية معينة حسب كل نظام تشغيل كالآتي

```
#ifdef __WINDOWS__
    #include <stdlib.h>
    #include <sys \stat.h>
    #include <io.h>
#endif // __WINDOWS__

#ifdef __UNIX__
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>
#endif // __UNIX__
```

او في حالة اعطاء امر المترجم معين اذا تم استخدامه

```
#ifdef __BORLANDC__
    #pragma hdrstop
#endif
```

ولها استخدام هام في الملفات الراسية كما سنري لاحقا للتأكد ان الملف يتم اضافته مرة واحدة فقط

```
#ifndef PNG_H
#define PNG_H
.
.

#endif /* PNG_H */
```

والكود السابق مساوي للآتي باستخدام الكلمة defined والرمز ! للنفي

```
#if !defined(PNG_H)
#define PNG_H
.
.
.

#endif /* PNG_H */
```

- الرمز # في المختصرات

هذا الرمز يحول القيمة التي بعده الى نص بين علامتي تنصيص كالآتي

```
#define msg(x) printf(#x);  
msg(hello) -> printf("hello");
```

-الرمز ## في المختصرات

يستخدم للدمج بين وسيطين في المختصرات كالآتي

```
#define CAT(x,y) x##y  
CAT(num,1) -> num1
```

- الثوابت الرمزية المعرفة مسبقا predefined symbolic constants

مجموعة من الثوابت المعرفة مسبقا وكل منها يبدأ _ وينتهي ب _ وهما كما بالجدول التالي

الثابت	الوصف
__LINE__	رقم السطر في الكود
__FILE__	اسم الملف الحالي
__DATE__	تاريخ ترجمة الملف
__TIME__	وقت ترجمة الملف
__STDC__	قيمه 1 اذا كان المترجم يدعم لغة السي القياسية

مثال

```
int main()  
{  
    printf("__LINE__ is %d \n",__LINE__);  
    printf("__FILE__ is %s \n",__FILE__);  
    printf("__DATE__ is %s \n",__DATE__);  
    printf("__TIME__ is %s \n",__TIME__);  
    printf("__STDC__ is %d \n",__STDC__);  
    return 0;  
}
```

__LINE__ is 9
__FILE__ is \11 – Preprocessor\01.c
__DATE__ is Sep 11 2017
__TIME__ is 00:53:12
__STDC__ is 1

[11 - Preprocessor/01.c]

- الملفات الرأسية header files

تحتوي تلك الملفات علي ثوابت ونماذج الدوال functions prototype و لتلك الملفات دور هام في تنظيم اكواد البرامج حيث تستخدم في فصل الاعلانات declarations عن التعريفات definitions or implementations للدوال , ونلاحظ من الملفات الرأسية التي تعاملنا معها ان كل ملف يحوي نماذج دوال مرتبطة مع بعضها , الملف الراسي يسمى احيانا بالحد المشترك interface حيث يحتوي علي النماذج للدوال فقط وليس التعريفات واي مبرمج يحتاج ان يعرف الوسائط وانواعها لكل دالة ونوع القيمة التي ترجعها الدالة وهذا ما توفره له الملفات الرأسية , الجدول التالي يوضح الملفات الرأسية الشائعة في السي

الملف	الوصف
<stdio.h>	يحتوي نماذج الدوال الخاصة بالدخل والخرج , وايضا الثوابت الخاصة بها
<math.h>	يحتوي نماذج الدوال الخاصة بالعمليات الرياضية , وايضا الثوابت الخاصة بها
<string.h>	يحتوي نماذج الدوال الخاصة بالعمليات علي النصوص , وايضا الثوابت الخاصة بها
<time.h>	يحتوي نماذج الدوال الخاصة بالتعامل مع الوقت والتاريخ , وايضا الثوابت الخاصة بها
<stdlib.h>	يحتوي نماذج الدوال الخاصة بتحويل الارقام الي نصوص والعكس ,الارقام العشوائية وغيرها , وايضا الثوابت الخاصة بها
<stdarg.h>	يحتوي نماذج الدوال الخاصة بالتعامل مع الوسائط الغير محددة للدوال , وايضا الثوابت الخاصة بها
<limits.h>	يحتوي كل القيم العظمي والصغري لمتغيرات الاعداد الصحيحة
<float.h>	يحتوي كل القيم العظمي والصغري لمتغيرات الاعداد الحقيقية

لنفرض اننا نحتاج ان نتقوم بكتابة برنامج عبارة عن آلة حاسبة بسيطة للاعداد الصحيحة بها عمليات اساسية وهي جمع وطرح وضرب وقسمة وباقي قسمة

1- نقوم بانشاء ملف ليكون الملف الراسي لنا ونضع به كل نماذج الدوال الخاصة بنا وليكن "calc.h"


```

#ifndef __CALC__
#define __CALC__

//addition function
int add(int a,int b);

// subtract function
int sub(int a ,int b);

//multiplication function
int mult(int a ,int b);

//division function
int div(int a ,int b);

//remainder function
int rem(int a ,int b);

#endif // __CALC__

```

[11 - Preprocessor\calc.h]

2- نقوم بانشاء ملف اخر سيكون به تعريفات الدوال , ويمكن لتعريفات الدوال ان توجد في اكثر من ملف , ليكن اسم الملف "calc.c"

```

#include "calc.h"

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

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

int mult(int a ,int b)

```

```

{
    return (a * b);
}

int div(int a, int b)
{
    return (a / b);
}

int rem(int a, int b)
{
    return (a % b);
}

```

[11 - Preprocessor\calc.c]

بدا الملف باضافة الملف الراسي الخاص بنا ولاحظ استخدام " " بدلا من <> لان الملف موجود مع ملفات المشروع , بعد ذلك اضفنا تعريف كل دالة

3- استخدام تلك الدوال داخل الدالة الاساسية او اي كود اخر

```

#include<stdio.h>
#include "calc.h"

int main()
{
    printf("10 + 3 = %d \n", add(10,3));
    printf("10 - 3 = %d \n", sub(10,3));
    printf("10 * 3 = %d \n", mult(10,3));
    printf("10 / 3 = %d \n", div(10,3));
    printf("10 %% 3 = %d \n", rem(10,3));
    return 0;
}

```

```

10 + 3 = 13
10 - 3 = 7
10 * 3 = 30
10 / 3 = 3
10 % 3 = 1

```

[11 - Preprocessor/main.c]

اضفنا الملف الراسي بالاعلي واستخدمنا الدوال الخاصة بنا لاجراء الحسابات

- ما فائدة تلك التوجيهات في الملف الراسي

```
#ifndef __CALC__  
#define __CALC__  
.  
.  
.  
#endif // __CALC__
```

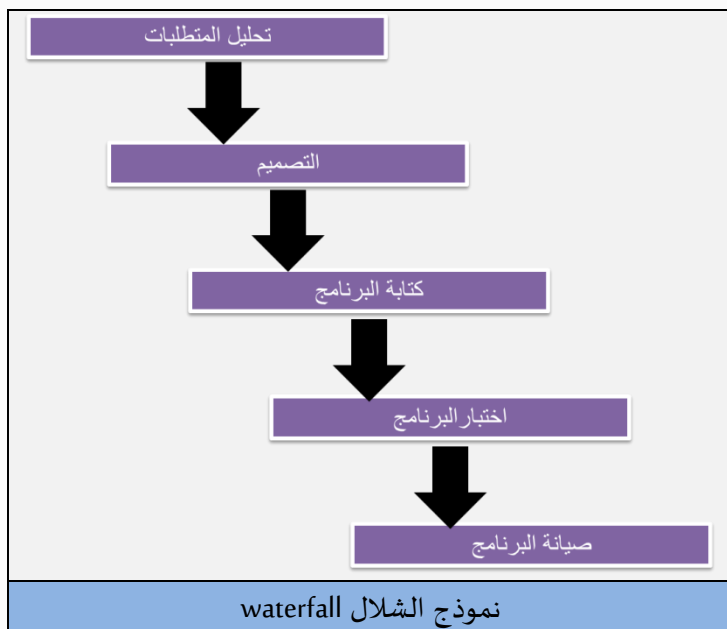
لمنع هذا الملف من اضافته اكثر من مرة في نفس الملف , فلو كتبنا الاتي

```
#include "calc.h"  
#include "calc.h"
```

اضفنا الملف مرتين , نتيجة للتوجيهات directives التي بالاعلي سيتم اضافة الملف مرة واحدة فقط , لو لم نضيف تلك التوجيهات سيتم اضافة الملف مرتين وستظهر مشاكل اثناء الترجمة وهي وجود اكثر من اعلان عن دوال بنفس الاسم

- مراحل تطوير البرامج (SDLC) software development life cycle

تطوير البرمجيات ليس فقط كتابة الاكواد فاي برنامج يمر بمراحل معينة منذ البداية الي ان يكون منتج جاهز للاستخدام , يوجد اكثر من نموذج يتبع اثناء تطوير البرمجيات منها نموذج الشلال waterfall ومنها Agile based و V-model وغيرها ولكن هناك مراحل توجد في اي نموذج بصرف النظر عن ترتيب المراحل او اي المراحل يتم تكرارها , هذه المراحل كالآتي



1- تحليل المتطلبات requirements analysis

البرنامج يكون اما فكرة خاصة بنا او ان عميل customer طلب منا ان نطور برنامج معين له في جميع الحالات يجب معرفة كل السمات features والخصائص للبرنامج الذي سنعمل عليه , وان تكون كل السمات واضحة بالنسبة لنا وببساطة فان هذا المرحلة تجاوب علي السؤال الاتي ماذا نريد ؟ او ماذا يريد العميل ؟ , ومن الوثائق الهامة هنا srs او software requirement specifications اي المواصفات الخاصة بالبرنامج وهذه الوثيقة تحتوي علي وصف للبرنامج وماذا سيؤدي البرنامج من وظائف

2- تصميم البرنامج code design

هذه المرحلة تجاوب علي سؤال , كيف سنطور المتطلبات التي

نريدها ؟ , يتم عمل تصميم للبرنامج ككل , تصميم الخوارزميات الرئيسة ... الخ

3- كتابة الاكود coding

يتم كتابة الاكود وفقا لتصميم البرنامج

4- الاختبار testing

يتم اختبار البرنامج للتأكد ان السمات التي تم تطويرها هي التي يحتاجها العميل , وايضا لاكتشاف الاخطاء

5- صيانة maintenance

هنا يتم تصحيح اي اخطاء وجدت في مرحلة الاختبارات

الخلاصة

- لكي يتم تنفيذ البرامج علي الحاسب يمر البرامج بمجموعة من المراحل بداية من كتابة الكود الي التنفيذ
- التوجيهات directives or preprocessors هي تعليمات للمترجم قبل البدء بعملية الترجمة , وهذه التعليمات تبدأ دائما بالرمز (#) وتوضع في اول الملف عادة
- الملفات الراسية header files تحتوي تلك الملفات علي ثوابت ونماذج الدوال functions prototype و لتلك الملفات دور هام في تنظيم اكواد البرامج

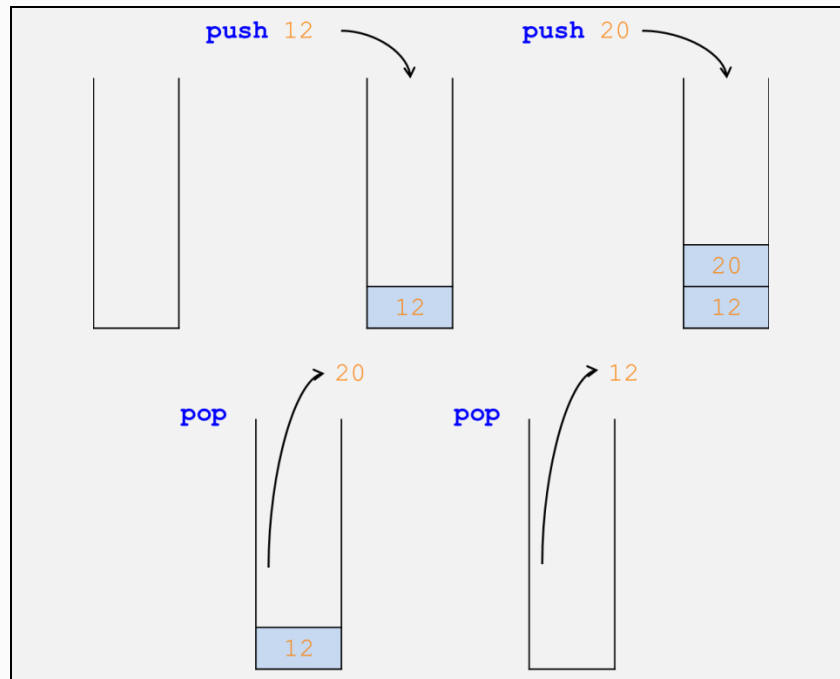
12

Advanced Functions and Pointers

الدوال والمؤشرات

- المكس stack

هو جزء من الذاكرة يتم تخزين البيانات فيه وقراءتها تبعا لمبدأ Last in first out LIFO اي ما تم تخزينه اخرا يتم خروجه (قراءته) اولا وعملية تخزين بيانات فيه تسمى push والقراءة اخر قيمة موجودة فيه pop كما بالشكل



- تسلسل تنفيذ الكود اثناء استدعاء الدوال

عند استدعاء دالة ما , فان هذه الدالة يجب ان تعرف كيف تعود للكود الاصلي الذي تم مناداتها منه , ولذا فان العنوان الذي ستعود اليه الدالة بعد تنفيذ مهمتها يتم تخزينه في ال stack , ويتم بعد ذلك تخصيص جزء من ال stack يسمى هذا الجزء stack frame or activation record ويتم انشاءه عند اي استدعاء لدالة , ويستخدم لتخزين البيانات الخاصة بالدالة مثل قيم الوسائط , القيمة المرجعية منها , تخزين مؤقتة للمتغيرات داخل تلك الدالة وعندما تنتهي الدالة من اداء مهمتها وتعود الي الكود الاصلي , يتم سحب هذا ال stack frame من ال stack

- تكرار استدعاء الدالة Recursion

الدالة التكرارية هي الدالة التي تستدعي نفسها سواء بشكل مباشر او من خلال دالة اخري , ويجب لتلك الدالة ان تحافظ علي شرطين اساسين هما :

- ان يكون لها نقطة نهاية
- ان تسهل حل المشكلة

من الامثلة التي تساعد علي فهم هذا النوع من الدوال هو حساب المضروب ومن المعلوم ان مضروب اي رقم هو حاصل ضرب الارقام بداية من الرقم وحتى الواحد , سنقوم بكتابة دالة عادية جدا لحساب المضروب واخري باستخدام التكرار لمعرفة الفرق

```
//declare fact function
```

```
int fact(int n);
```

```
int main()
```

```
{
```

```
printf("4! = %d \n", fact(4));
```

```
printf("6! = %d \n", fact(6));
```

```
return 0;
```

```
}
```

```
int fact(int n)
```

```
{
```

```
if (n == 0 || n == 1)
```

```
{
```

```
return 1;
```

```
}
```

```
else
```

```
{
```

```
int result = 1;
```

```
while(n)
```

```
{
```

```
result *= n;
```

```
n--;
```

```
}
```

```
return result;
```

```
}
```

```
//declare fact function
```

```
int fact(int n);
```

```
int main()
```

```
{
```

```
printf("4! = %d \n", fact(4));
```

```
printf("6! = %d \n", fact(6));
```

```
return 0;
```

```
}
```

```
int fact(int n)
```

```
{
```

```
if (n == 0 || n == 1)
```

```
{
```

```
return 1;
```

```
}
```

```
else
```

```
{
```

```
return n*fact(n-1);
```

```
}
```

```
}
```

4! = 24

6! = 720

4! = 24

6! = 720

[12 – advanced functions and pointers/01.c]

[12 - advanced functions and pointers/02.c]

في الدالة التي علي اليمين تم استخدام التكرار للدالة في حساب المضروب وسنجد ان الكود الخاص بالدالة ابسط من كود الدالة الاخرى

```

int fact(int n)
{

    if (n == 0 || n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}

```

التكرار يتم في حالة ان الرقم ليس صفرا او واحد تستدعي الدالة نفسها بعد انقاص القيمة بمقدار واحد

```
return n*fact(n-1);
```

ان هذا السطر ييتحول فعليا الي الاتي اذا كنا نريد مثلا حساب مضروب 4

```
4 * fact(3) = 12 * fact(2) = 24 * fact(1) = 24
```

وسنجد ان هذه الدالة حققت شرطي التكرار, ان لها نقطة نهاية $n == 1$ || $n == 0$, انها تبسط حل المشكلة , ومن الهام هنا ان نعرف اننا اذا اردنا حساب مضروب رقم سالب فان هذا الدالة لن تنتهي ابدا وستؤدي الي امتلاء ال stack وحدوث stack overflow لذا فهي تعمل مع الاعداد الموجبة فقط

- متسلسلة فيبوناتشي Fibonacci series

هي متتالية رياضية تبدأ ب 1 ثم 1 وبعد ذلك كل حد في المتتالية يساوي مجموع الحدين السابقين له وهي كالتالي

```
1, 1, 2, 3, 5, 8, 13, 21 ...
```

سنقوم بكتابة دالة تقوم بحساب قيمة الحد في المتتالية , لتكون

```

fab(0) = 1 ;
fab(1) = 1 ;
fab(2) = 1 ;
fab(3) = 2 ;
fab(n) = fab(n-1) + fab(n-2)

```



```

//declare fab function
int fab(unsigned int nth);

int main()
{
    //print the first 10 elements in series
    int i;
    for(i=0; i<10; i++)
    {
        printf("%d ",fab(i) );
    }

    return 0;
}

int fab(unsigned int nth)
{
    if (nth == 0 || nth== 1)
    {
        return 1;
    }
    else
    {
        return fab(nth-1) + fab(nth-2);
    }
}

```

1 1 2 3 5 8 13 21 34 55

[12 - advanced functions and pointers/03.c]

تقوم الدالة بحساب قيمة الحد في المتتالية باستخدام التكرار حيث يتم تكرار استدعاء الدالة طالما ان الحد ليس رقم 0 او 1

```

int fab(unsigned int nth)
{
    if (nth == 0 || nth== 1)

```

```

{
    return 1;
}
else
{
    return fab(nth-1) + fab(nth-2);
}
}

```

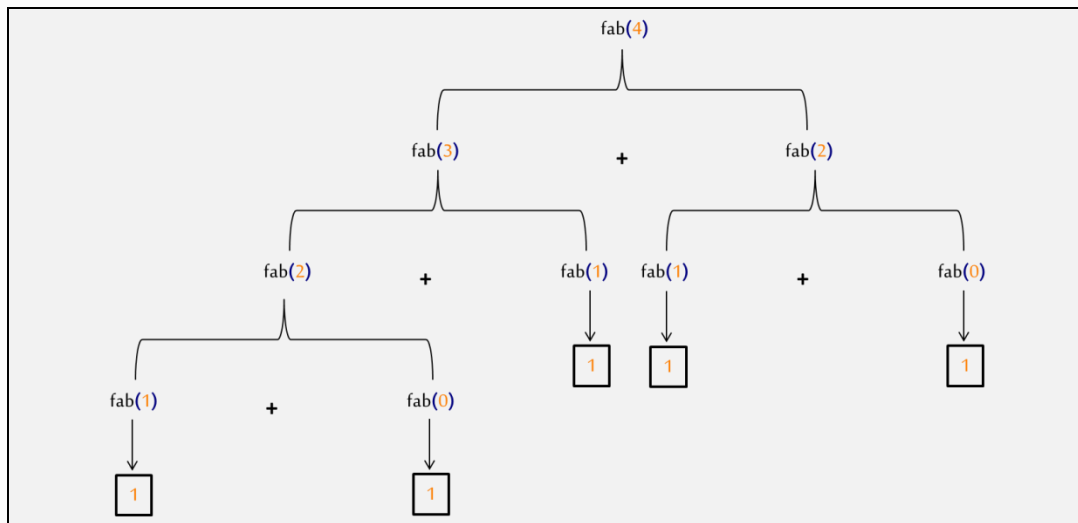
ولنفرض اننا نريد حساب قيمة الحد رقم 4 مثلا سيكون التكرار كالآتي

```

fab(4) = fab(3) + fab(2) = fab(2) + fab(1) + fab(1) + fab(0)
= fab(1) + fab(0) + 1 + 1 + 1 = 1 + 1 + 3 = 5

```

وهذا الدالة تحقق شرطي التكرار حيث ان لها نقطة نهاية $nth == 1 \parallel nth == 0$ وتبسط الحل للمشكلة ولفهم كيف تقوم بعملها نستخدم الشجرة لحساب ذلك كما في الشكل التالي



مثال برنامج يقوم باستقبال حروف من المستخدم وطباعتها بالعكس بعد ادخال $(\backslash n)$

```

//declare printChar function
void printChar();

int main()
{

    puts("type any chars , press enter to finish ");
}

```

```

printChar();

return 0;
}

void printChar()
{
    char ch;
    //read char for stdin
    ch = getc(stdin);

    //check for \n char
    if(ch != '\n' )
    {
        printChar();
    }

    printf("%c",ch);
}

```

type any chars , press enter to finish
 this is test for reverse chars after enter

retne retfa srahc esrever rof tset si siht

[12 – advanced functions and pointers/04.c]

تقوم الدالة باستدعاء نفسها مرة أخرى إذا لم يكن الحرف هو (\n) ويتم طباعة الحرف بعد العودة من الاستدعاء ان حدث

- المؤشر العام generic pointer

تعاملنا مع انواع مختلفة من المؤشرات حيث كل نوع مرتبط بنوع معين من البيانات للتعامل معه, تحتوي السي علي مؤشر عام يمكن ان يشير الي اي نوع ولكن يجب تحويله الي نوع معين عند التعامل معه اي قبل الاستخدام , يتم الاعلان عنه باستخدام الكلمة المحجوزة void كالآتي

```
void *ptr;
```

ويتم تحويله باستخدام casting

```
void *ptr;
int *iptr = (int*)ptr;
```

- تخصيص مساحة تخزينية في الذاكرة اثناء تشغيل البرنامج dynamic memory allocation

كل المساحات التي تم حجزها حتي الان للمتغيرات بجميع انواعها معروفة مسبقا قبل تشغيل البرنامج من حيث الحجم المطلوب , هناك حالات نحتاج الي حجز مساحة اضافية في الذاكرة اثناء تشغيل البرنامج , يتم ذلك باستخدام الدوال malloc او calloc ثم free بعد ذلك عند الانتهاء ويتم حجز الذاكرة داخل الجزء من ذاكرة البرنامج المسمي بال heap والمخصص لذلك , ويجب بعد الانتهاء من استخدام مساحة معينة ان نعيدها للنظام مرة اخري لاعادة استخدامها مرة اخري وحتي لا نستهلك الذاكرة بدون فائدة ويتم ذلك باستخدام الدالة free

- الدوال malloc و calloc تقوم باعادة مؤشر شامل void pointer والذي يشير الي عنوان المساحة التي تم حجزها

تستخدم sizeof مع تلك الدوال لتحديد الحجم الذي سيتم حجزه

<pre>void * malloc(size_t size);</pre> <div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <p>مؤشر للمساحة التخزينية او null اذا لم يتم تخصيص المساحة</p> <p>الحجم المراد تخصيصه</p>	<pre>void * calloc(size_t num, size_t size);</pre> <div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <p>مؤشر للمساحة التخزينية او null اذا لم يتم تخصيص المساحة</p> <p>عدد العناصر المراد حجزها</p> <p>حجم العنصر الواحد</p>
malloc	calloc

مثال برنامج يقوم بحجز متغير صحيح اثناء التشغيل واعادته مرة اخري

```
//pointer to hold new storage address
int * ptr;
//allocate memory
ptr = (int*)malloc(sizeof(int));
//change value
(*ptr)=20;
//print content
printf("value is %d \n", (*ptr));
//free allocated memory
free(ptr);
```

value is 20

[12 - advanced functions and pointers/05.c]

في البداية حجزنا مؤشر لتخزين عنوان المساحة التي سيتم حجزها , واستخدمنا الدالة malloc لحجز مساحة بحجم متغير من نوع int حيث تأخذ تلك الدالة وسيط واحد وهو الحجم المراد حجزه بالبايت لذا استخدمنا sizeof لتحديد الحجم بالضبط

```
ptr = (int*)malloc(sizeof(int));
```

قمنا بعد ذلك بتغيير المحتوي وطباعته واخير قمنا باعادة تلك المساحة الى النظام بعد الانتهاء من استخدامها باستخدام الدالة free

```
free(ptr);
```

واذا استخدمنا الدالة الاخرى calloc لحجز نفس المساحة ستكون كما يلي

```
ptr = (int*)calloc(1, sizeof(int));
```

الوسيط الاول هو عدد الوحدات المراد حجزها , الوسيط الثاني حجم الوحدة .

ما الفرق بين ما فعلناه وبين ان نكتب

```
int x;
```

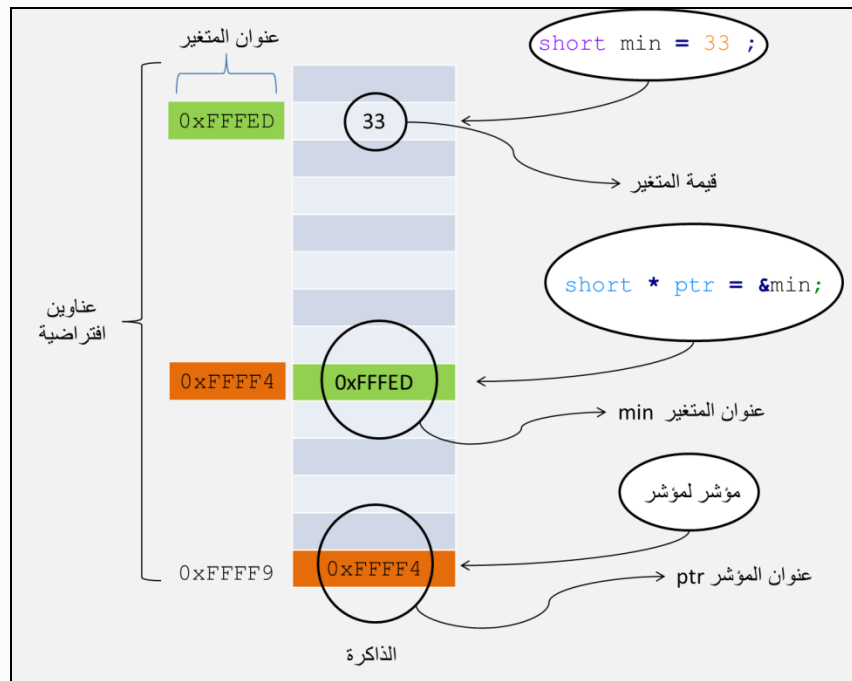
هنا سيتم حجز مكان للمتغير بشكل تلقائي , وسيتم اعادتها الى النظام بعد الانتهاء منها بشكل تلقائي

- تسرب الذاكرة memory leak

يحدث عندما يتم حجز مساحة ما ولا يتم اعادتها الى الذاكرة بعد الانتهاء من استخدامها , فالنظام يعتبر تلك المساحة محجوزة لنا واننا نستخدمها ومن جانبنا نحن استخدمناها ولم نعد بحاجة اليها فتركناها كما هي دون اعادتها للنظام

- مؤشر لمؤشر pointer to pointer

تعاملنا من قبل مع المؤشرات وعرفنا انها تحمل عناوين لاماكن في الذاكرة , يمكن ان ننشئ مؤشر لمؤشر الذي بدوره يحمل عنوان مؤشر في الذاكرة , يجب ان نؤكد هنا علي ان المؤشر هو متغير وياخذ مساحة تخزينية في الذاكرة الفرق ان محتوى تلك المساحة هو عنوان لمتغير وليس قيمة لها معني



طريقة انشاء مؤشر لمؤشر (ثنائي)

```
void **ptr;
```

وبالمثل يمكن ان نجد مؤشر لمؤشر لمؤشر (ثلاثي)

```
void ***ptr;
```

- احد استخدامات هذا النوع من المؤشرات عندما نريد تغير العنوان الذي يحمله المؤشر من داخل دالة , مثال دالة تقوم بحجز متغيرين من نوع `int` و `float` وذلك باستخدام الدالة `malloc`

```
//declare function prototype
void createVar( float **fptr , int **iptr);

int main()
{
    //pointer to hold new storage address
    int * iptr;
    float *fptr;
    createVar(&fptr,&iptr);

    (*iptr) = 40;
    (*fptr) = 12.3;
```

```

printf("values are %d , %.2f\n", (*iptr), (*fptr));
//release memory
free(iptr);
free(fptr);
return 0;
}

void createVar( float **fptr , int **iptr )
{
    //allocate for float
    (*fptr) = (float*)malloc(sizeof(float));
    //allocate for int
    (*iptr) = (int*)malloc(sizeof(int));
}

```

values are 40 , 12.30

[12 - advanced functions and pointers/06.c]

ربما السؤال الاول لنا سيكون لماذا تم استخدام مؤشر لمؤشر الا يمكن ان نستخدم مؤشر فقط ؟ وتكون الدالة كالتالي

```

void createVar( float *fptr , int *iptr )
{
    //allocate for float
    fptr = (float*)malloc(sizeof(float));
    //allocate for int
    iptr = (int*)malloc(sizeof(int));
}

```

الاجابة هي لا يمكن استخدام المؤشر فقط , والكود بالاعلي سيتم حجز مساحة فعليا ولكن عنوانه لن يتم تخزينه في المؤشرات الخاصة بنا

لفهم ما حدث يجب ان نتذكر اولاً ان الوسائط كمتغيرات يتم اخذ نسخة منها داخل الدالة وتحديدًا في stack frame

- مؤشر لدالة function pointer

يستخدم للإشارة الى الدوال , ويستخدم لاستدعاء الدوال تمام مثل استدعاء الدوال باسمائها

امثلة

مؤشر لدالة من نوع void ولا تأخذ وسائط	<code>void (*ptr)(void);</code>
مؤشر لدالة من نوع void وتأخذ وسيط من نوع int	<code>void (*ptr)(int);</code>
مؤشر لدالة من نوع int وتأخذ وسيطين من نوع int	<code>int (*ptr)(int,int);</code>

مثال برنامج به دالة لجمع رقمين ويتم استدعاؤها من خلال مؤشر لدالة

```
//declare function prototype
int add(int,int);

int main()
{
    //pointer to function of type int and take 2 parameter of type int also
    int (*ptr)(int,int);

    ptr = add;

    printf(" 10 + 4 = %d \n",ptr(10,4));

    return 0;
}

int add(int a,int b)
{
    return a+b;
}
```

10 + 4 = 14

[12 – advanced functions and pointers/07.c]

يبدأ البرنامج بحجز مؤشر لدالة من نوع int والتي تأخذ وسيطين من نوع int

```
int (*ptr)(int,int);
```

اعطينا المؤشر عنوان دالة add والتي من نوع int والتي تأخذ وسيطين من نوع int ايضاً

```
ptr = add;
```


ويمكن ان ياخذ عنوان الدالة باستخدام الرمز & ايضا

```
ptr = &add;
```

ثم قمنا باستدعاء الدالة من خلال المؤشر

```
printf(" 10 + 4 = %d \n", ptr(10,4) );
```

او باستخدام الرمز * ايضا

```
printf(" 10 + 4 = %d \n",(*ptr)(10,4));
```

احد استخدامات هذا المؤشر في callback functions والتي تعني انه لدينا دالة تاخذ احد وسائطها مؤشر لدالة وتقوم بمناداة تلك الدالة من داخلها

مثال توضيحي للفكرة

```
//declare function prototype
int add(int,int);
int sub(int,int);
int calc(int , int , int (*func)(int,int) );

int main()
{
    //pointer to function of type int and take 2 parameter of type int also
    int (*ptr)(int,int);
    //assign add function to ptr
    ptr = add;

    printf(" 10 + 4 = %d \n",calc(10,4,ptr));
    //assign sub function to ptr
    ptr =sub;
    printf(" 10 - 4 = %d \n",calc(10,4,ptr));
    return 0;
}

int add(int a,int b)
{
    return a+b;
```

```

}

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

int calc(int a, int b, int (*func)(int,int) )
{
    return func(a,b);
}

```

10 + 4 = 14
10 - 4 = 6

[12 - advanced functions and pointers/08.c]

اعلنا عن دالة calc والتي تاخذ ثلاث وسائط الثالث هو مؤشر لدالة من نوع وتاخذ وسيطين من نفس النوع

```
int calc(int , int , int (*func)(int,int) );
```

تقوم تلك الدالة باعادة استدعاء callback دالة الوسيط الثالث من داخلها لحساب نتيجة العملية التي من الممكن ان تكون جمع او طرح

```

int calc(int a, int b, int (*func)(int,int) )
{
    return func(a,b);
}

```

داخل الدالة الاساسية قمنا بحجز مؤشر لدالة من نوع int والتي تاخذ وسيطين من نوع int وقمنا باعطاءه عنوان الدالة add وعنوان الدالة sub مرة اخري وبناء عليه تغيرت القيمة التي ترجعها الدالة calc في كل مرة

```

ptr = add;
printf(" 10 + 4 = %d \n",calc(10,4,ptr));
//assign sub function to ptr
ptr =sub;
printf(" 10 - 4 = %d \n",calc(10,4,ptr));

```

- لانشاء مصفوفة مؤشرات لدوال من نوع معين نكتب الاتي

```
void (*ptr[10])(void);
```

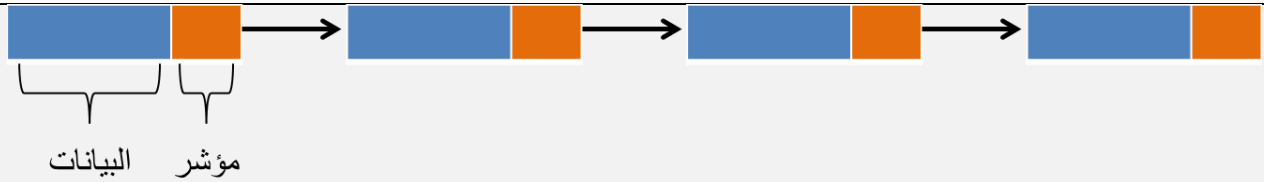
```
void (*ptr[4])(int);
int (*ptr[7]) (int,int);
```

- البنية ذاتية المرجعية self-referential structure

هي ان يكون احد عناصر البنية هو مؤشر لبنية من نفس النوع كالاتي

```
struct node
{
    int data;
    struct node *nextPtr;
};
```

تحتوي هذه البنية علي المؤشر `struct node *nextPtr` الذي يشير الي بنية من نفس النوع. ويمثل هذا المؤشر حلقة الوصل بين بنية واخري , حيث يمكن من خلاله التنقل من بنية لآخري لانه يحمل عنوان البنية التي تليه , البنية التي تليه بها ايضا مؤشر للبنية التي تليه وهكذا وبهذا تم تكوين سلسلة متصلة من البنيات , ولذا تستخدم في هياكل البيانات data structures مثل القائمة المرتبطة linked list, المكس stack وغيرها حيث تعتمد جميعها علي نفس الفكرة ان لدينا بنيات متصلة مع بعضها بالمؤشرات وبعضها يحتاج مؤشر اخر للاشارة الي البنية التي تسبقه , بحيث يستطيع التنقل للامام او للخلف



الخلاصة

- المكس stack هو جزء من الذاكرة يتم تخزين البيانات فيه وقراءتها تبعا لمبدأ Last in first out LIFO
- الدالة التكرارية هي الدالة التي تستدعي نفسها سواء بشكل مباشر او من خلال دالة اخري
- تحتوي السي علي مؤشر عام يمكن ان يشير الي اي نوع ولكن يجب تحويله الي نوع معين عند التعامل معه اي قبل الاستخدام
- تستخدم الدوال malloc او calloc لحجز مساحات في الذاكرة داخل الجزء من ذاكرة البرنامج المسمي بال heap والمخصص لذلك



Practical Examples

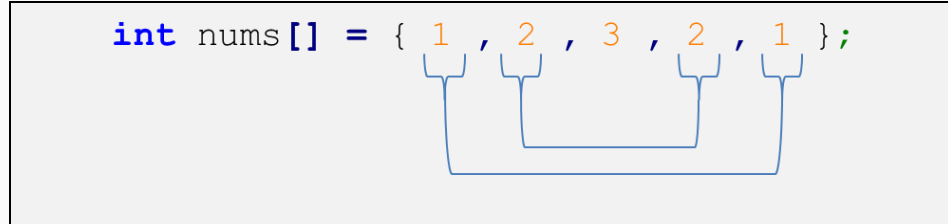
امثلة عملية

1- اكتب دالة للتحقق من مصفوفة اذا كانت منعكسة mirrored ام لا , مثال لمصفوفتين احدهما منعكسة والاخرى لا

<code>int nums[] = {1,2,3,4,5};</code>	<code>int nums[] = {1,2,3,2,1};</code>
غير منعكسة	منعكسة

التفكير في الحل

من المثالين بالاعلي سنجد ان المصفوفة تكون منعكسة اذا كان اول عنصر يساوي اخر عنصر, ثاني عنصر يساوي العنصر قبل الاخير وهكذا



الحل

سنستخدم الحلقة التكرارية لمقارنة القيم ببعضها

```
int isMirroed(int arr[] ,int len);

int main()
{
    //array of integers
    int num [5] = {1,2,3,2,1};

    if( isMirroed(num,5) == 1 )
    {
        puts("array is mirrored");
    }
    else
    {
        puts("array isn't mirrored");
    }
    return 0;
}

int isMirroed(int arr[] ,int len)
{
    //loop counter
    int I ;
    for(I =0 ; i<len/2 ;i++)
    {
        if(arr[i] != arr[len-1-i])
        {
            //array isn't mirrored
            return 0 ;
        }
    }
    //array is mirrored
    return 1;
}
```

[13 - practical examples /mirrored.c]

2- اكتب برنامج يقوم بطباعة الارقام من 1 وحتى 10 بدون استخدام الحلقات التكرارية

التفكير في الحل

ربما تفكر كالآتي بما انه يريد الطباعة بدون استخدام الحلقات التكرارية سنقوم بالطباعة باستخدام جملة الطباعة printf , واستدعاء تلك الدالة حتي يتم طباعة كل تلك الارقام ليصبح البرنامج مثل الآتي

```
printf("1 \n 2 \n 3 \n 4 \n 5 \n");
printf("6 \n 7 \n 8 \n 9 \n 10 \n");
```

ربما هذا الحل سيؤدي الغرض , ولكن اذا كانت اردنا الطباعة من 1 الي 100 , تلك الطريقة ستجعل الكود اطول , كما اننا هنا لسنا بصدد التدريب علي استخدام تلك الدالة , هناك حل اخر باستخدام recursion , حيث سنعرف دالة تستدعي نفسها عدة مرات حتي تنتهي من الطباعة وتلك الدالة تاخذ وسيطن الاول هو نهاية الطباعة والثاني هو الرقم الذي سيتم طباعته

الحل

```
void print(int end ,int current);

int main()
{
    print(10 , 1);
    return 0;
}

void print(int end , int current)
{
    printf(" %d \n",current);

    if(end != current)
    {
        print(end , current+1 );
    }
}
```

[13 - practical examples /print_numbers.c]

3- اكتب برنامج للتحقق من النص هل هو palindrome ام لا وذلك يعني ان النص يمكن قراءته من كلا الاتجاهين , مثل level و refer

التفكير في الحل

البرنامج هنا مثل البرنامج الخاص بالمصفوفه المنعكسة , ولكننا هنا سنتعامل مع النصوص

الحل

```
int isPalindrome(char * str );

int main()
{
    char * str = "level";
```

```

if( isPalindrome(str) == 1 )
{
    puts("string is palindrome");
}
else
{
    puts("string isn't palindrome");
}
return 0;
}

int isPalindrome(char * str )
{
    //loop counter
    int I ;
    //string length
    int len = strlen(str);
    for(I =0 ; i<len/2 ;i++)
    {
        if( *(str+i) != *(str+len-1-i) )
        {
            //string isn't palindrome
            return 0 ;
        }
    }
    //string is palindrome
    return 1;
}

```

[13 – practical examples /palindrome.c]

4- اكتب دالة تقوم باستخراج الارقام من نص وطباعتها , امثلة

```

abc12dfg4 → 124
hgy6w3b90c1 → 6390

```

التفكير في الحل

يمكن الحل باكثر من طريقة , حيث يمكن مقارنة كل حرف بالارقام من 0 الي 9 للتحقق من كل حرف , لذا سنستخدم جدول اسكي , حيث الارقام من الارقام من 0 الي 9 يعادلها في جدول اسكي القيم من 48 الي 57 كحروف , فاذا كانت قيمة الحرف ليست في النطاق من 48 الي 57 فهي ليست رقم

الحل

```

void extractNum(char * str);

int main()
{
    char *str = "hgy6w3b90c1";
    extractNum(str);
    return 0;
}

void extractNum(char * str)

```

```

{
    //loop counter
    int I ;
    //string length
    int len = strlen(str);
    for(I =0 ; i<len ;i++)
    {
        if( *(str+i) >= 48 && *(str+i) <= 57)
        {
            //number
            printf("%c",*(str+i) );
        }
    }
}

```

6390

[13 - practical examples /extract_numbers.c]

-5 اكتب دالة تقوم بحذف المسافات من بداية ونهاية النص , مثال

" have fun " --> "have fun"

الكود متروك للقارئ .

-6 اكتب دالة تقوم بايجاد عدد الكلمات في نص , امثلة

practise make perfect --> 3
have fun --> 2

التفكير في الحل

عدد الكلمات في نص يعتمد علي عدد المسافات spaces فيه حيث يتم الفصل بين كل كلمة واخري بمسافة , وهو يساوي عدد المسافات + 1 بشرط عدم وجود مسافات في بداية النص او في اخره

الكود متروك للقارئ .

تم بحمد الله

المراجع :

- C Programming Language , Brian W. Kernighan and Dennis M. Ritchie
- Practical C Programming Third Editon ,Steve Qualline
- الشامل في لغة السي , خليل اونيس
- Expert C Programming , Peter van der Linden
- C By Example, Greg Perry
- www.stackoverflow.com