بسم الله الرحمن الرحيم

الحمد لله رب العالمين وصلى الله على سيدنا محمد وآله الطيبين الطاهرين

Functions

الدوال (التوابع) في لغة C++

مقدمـــــــــــــة:

بتوفيق من الله جل وعلى تم انهاء هذا الفصل ليكون بداية لتعلم الدوال في لغة c++ وقد حرصت قدر المستطاع ان يكون بسيطا ومفهوما من قبل كل المستويات وهذا ما ستلاحظه من خلال الاسلوب البسيط للشرح وبساطة الامثلة الموضوعة فهو لايتناول افكار البرمجة العميقة ولكن يعرض لطرق كتابة واستدعائات الدوال وطرق التعامل معها والغاية منها وكل ما استطعت الحصول علية في هذا المجال .

لا اخفيكم فقد كانت معظم معلوماتي التي استندت عليها في الشرح مستمدة من بعض المصادر التي اطلعت عليها من هنا وهناك وماتعلمته من الجامعة لكن الفضل الاكبر يعود لموقع www.cplusplis.com حيث في بداية تعلمي للتعامل مع الدوال لم اجد مصدر لشرح مبسط لذا قررت اللجوء الى المصادر الاجنبية ومنها هذا الموقع المميز وقد وضعت نفس الامثلة الموضوعة فيه لبساطتها وشموليتها .

ونصيحتي لك عند قراءتك لهذا الفصل ان تقراه من البداية الى النهاية بشكل متسلسل واعادة القراءة في حال كان هناك شئ من الغموض في موضوع ما .

ما آمله من القراء هو ان لايتم نشره تحت أي مسمى او اسنادة الى غير صاحبة

مثنى عبد الرسول محسن الفرطوسي
كلية شط العرب الجامعة
قسم علوم الحاسبات
2006
Compiler_x@ yahoo.com

الفهرس

الدوال functions :
ما هي الدالة ؟ :
الدالة وبكل بساطه هي مقطع برمجي يؤدي عمل معين لكن هذا المقطع يكون موقعه ليس ضمن جسم الدالة الرئيسية وانما خارج جسم الدالة الرئيسية الـ (()main) واعتقد ان سؤالاً سيقفز الى ذهنك مباشره وهو لماذا نستخدمها ؟ وسوف اجيبك فورا للاسباب التالية:

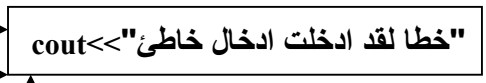
1- **لتسهيل كتابه البرامج الكبيرة**
2- **سهولة تتبع الاخطاء**
3- **سهوله التعديل و التطوير على البرنامج من دون الحاجة الى اعادة كتابه البرنامج كاملا**
4- **لجعل حجم البرنامج صغير**

وسأضيف على ذلك: باختصار ان الدالة(او يمكن تسميتها برنامج فرعي) تفيد في جعل كتابة البرامج سهله وسلسة واكثر حداثة وقوة
هل اكتفيت بهذا بالجواب عن معنى الدوال و الغرض منها ؟ ، لا.. لا اعتقد ذلك ، لانك لو اكتفيت بهذا الجواب لكنت قد غلقت الكتاب بدون الحاجة الى قراءة بقيته ، لكنك شخص فضولي وتود معرفة كيف سيكون برنامجك اكثر سهوله وحداثة وقوة واقل حجما بهذه الطريقة...
وسأجيبك بهذا الجواب:
في بعض برامجنا الكبيرة نحتاج مثلا ان نقرأ رقم معين وفي مرحلة التنفيذ اذا قام المستخدم بادخال قيمة خاطئه كأن تكون رمز او حرف سيطبع البرنامج عبارة (" خطا لقد ادخلت ادخال خاطئ ") .
وعند مرحلة اخرى في البرنامج احتجنا ان ندخل رقما اخر وايضا المطلوب منا تدقيق هذا الادخال بنفس الطريقة اعلاه أي شرط ان يكون رقماً وليس حرفا هنا سيتحتم علينا كتابة العبارة (" خطا لقد ادخلت ادخال خاطئ") مره اخرى ولو اردنا قرائه رقم ثالث سيتحتم علينا كتابتها مرة ثالثة وهكذا الى نهاية البرنامج ، اليس هذا مملا ؟ .
لغة سي ++ تفضلت مشكورة بوضع آليه لحل هذه المشكلة وهي ان نكتب هذه العبارة مرة واحدة خارج جسم الدالة الرئيسية ونقوم باستدعائها لتنفيذها كلما احتجنا اليها ولعدد غير محدد من المرات ، اليس هذا رائعاً ؟؟ شكرا يا لغة سي++.
واليك شكل البرنامج في الحالتين للتوضيح فقط :
البرنامج بدون استخدام الدوال

```
Int main()
{
.
.
.
cout<<" خطا لقد ادخلت ادخال خاطئ";
.
.
.
cout<<"خطا لقد ادخلت ادخال خاطئ";
.
.
cout<<"خطا لقد ادخلت ادخال خاطئ";
.
.
..
return 0;
}
```

نفس البرنامج السابق ولكن باستخدام الدوال

```
int main()
{
.
.
احتجنا الى الدالة نقوم باستدعائها فقط
.
.
احتجنا الى الدالة نقوم باستدعائها فقط
.
.
احتجنا الى الدالة نقوم باستدعائها فقط.

.
return 0;
}
```



هل لاحظت الفرق في الحالتين، اكيد سيوفر علينا الكتابة المتكررة لنفس الرسالة .

اضف الى ذلك انك تستطيع التعديل على البرنامج بكل سهولة وبساطة وسرعة فلو راينا بعد فترة من كتابة البرنامج ان هذه الرسالة واقصد ("خطا لقد ادخلت ادخال خاطئ") عبارة غامضة مثلا او ان فيها خطا معين وانه من الافضل ان استبدلها او اجري عليها تعديلا بسيطا لتكون مثلا(" ادخال خاطئ ، ادخل رقم") فكل ما ساقوم به هو تغيير الرسالة الموجودة في الدالة فقط وبذلك حققنا المطلوب بكل بساطة ، تخيل الان ان برنامجنا كان مكتوب بدون استخدام هذه الدالة أي اننا كتبناها في كل مكان في البرنامج احتجنا اليها فيه واردنا التعديل او تصحيح الخطا الموجود فيها فسيتوجب علينا البحث عن هذه العبارة في كل البرنامج وهذا شئ غير عملي خصوصا لو كان البرنامج متكون من عشرة الاف سطر مثلا ، فلنحمد الله على هذه النعمة .

```
int main()
{
.
.
احتجنا الى الدالة نقوم باستدعائها فقط
.
.
احتجنا الى الدالة نقوم باستدعائها فقط
.
.
return 0;
}
```

التغيير على الدالة فقط .

اعتقد انك الان اخذت فكرة ولو بسيطة عن معنى الدوال ، وسوف تتضح هذه الفكرة اكثر عندما نتقدم اكثر في الكتاب انشاء الله وايضا من خلال الامثلة . مع الاشارة ان المثال السابق هو مجرد مثال توضيحي فقط اما امكانات الدوال فهي تتعدى الرسائل بكثير فكل عملية (حسابية او معالجة او غيرها) تستطيع استخدام الدوال فيها.

بداية لنتفق على عدة اشياء

1- للتنوع في التسميات التي قرائها من المصادر الاخرى فأنني قد استخدم بعض من الكلمات المختلفه للدلالة على شئ واحد ومن هذة الكلمات

- (بارامیتر او معلمه او دلائل) تشير الى شئ واحد
- البرنامج الرئيسي او الدالة الرئيسية هما شئ واحد (وفي الحقيقة ان الاصح ان نقول دالة رئيسية وليس برنامج رئيسي لان البرنامج في لغة سي++ في الاساس هو عبارة عن دالة ولكن حتى لا يختلط الامر فنقول برنامج رئيسي)
- كومبايلر او مصرف او مترجم تشير الى شئ واحد وهو compiler وسوف استخدم كلمة (كومبايلر لاني تعودت عليها)

2- قد تظهر عدة اخطاء في بعض البرامج او ان بعض البرامج لا تنفذ وان كانت صحيحة وذلك يعود الى اختلاف النسخ من برنامج لغة سي ++

3- كل دالة تعيد قيمة واحدة فقط في اسمها وهذه ملاحظة مهمة جدا يجب ان تبقى في ذهنك لانها سترافقنا في كل الدوال تقريبا

لنبدأ الان في "الكلام العملي"

الصيغة العامة لكتابة الدوال

```
Type function_name (parameter1,parameter2, ….)
{
        statement 1;
        statement 2;
        statement 3;
        .
        .
        .
        return value
}
```

حيث ان :

Type : هو النوع البياني الذي تعود به الدالة كان يكون int او float او long او void .. الخ

Function_name : واضح انه يشير الى الاسم الذي سنطلقه على الدالة شرط ان يكون كلمة غير محجوزة للغة سي ++ وخاضع لقواعد التسمية.

Parameter : هو المتغير الذي نريد ارساله الى الدالة ويمكن ان نرسل عدد غير محدد من المتغيرات حسب احتياجنا (هذه المتغيرات او البارامترات تفيدنا في تنفيذ العمل المطلوب من الدالة) ويمكن ايضا ان لا نرسل أي باراميتر أي انها اختيارية.

Statement : هي العبارات او الجمل التي تكون الدالة (أي هي عبارات المقطع البرمجي الذي اخبرتك عنه سابقا)

Return : هي كلمة محجوزة للغة سي++ وهي تفيد في انهاء الدالة بصورة صحيحة وايضا تقوم باسناد القيمة التي بعدها وهي value الى اسم الدالة هل تذكر الملاحظة المهمة (كل دالة تعيد قيمة واحدة فقط باسمها).

Value: هي القيمة التي ستعيدها الدالة حيث ان كل دالة تعيد قيمة واحدة باسمها ، وهنا هذه القيمة يمكن ان تكون متغير او ثابت او رقم على شرط ان يكون من نفس النوع البياني الذي عرّفت فيه الدالة أي من نفس نوع الtype قبل اسم الدالة

لنأخذ المثال الاول عن الدوال وهو برنامج يستعمل دالة بسيطة لايجاد مجموع عددين صحيحين:

```
1       # include <iostream.h>

2       int sum (int x,int y)
        {
3               int z;
4               z=x+y;
5               return z;
        }

6       int main()
        {
7               int c;
8               c=sum(3,5);
9               cout<<" 3 + 5 = "<<c;
10              return 0;
        }
```

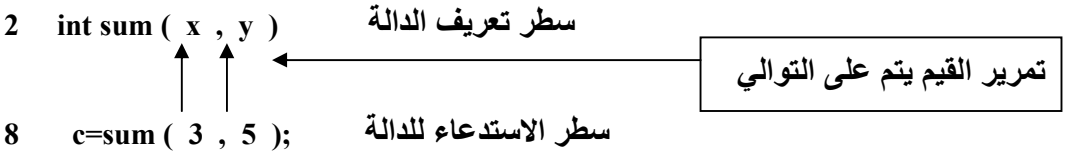
لقد قمت بترقيم الاسطر فقط من اجل سهولة الشرح ولا اساس لوجود هذه الارقام في البرنامج او الدالة
السطر رقم 1 يفيد لتضمين مكتبة الادخال و الاخراج القياسي iostram.h .
السطر 2 هو بداية تعريف الدالة وهي كما واضح من النوع الصحيح int ، وتحمل الاسم sum ، ولها بارامترين هما x و y هذه البارامترات سوف تحمل قيمة العددان المراد جمعهما ، ولا تشغل بالك بهما كثيرا الان لاننا سناتي على تفصيلهما بعد قليل وكل ما اريده منك الان ان تعلم ان x سوف ياخذ قيمة احد العددين المراد جمعهما و y ستاخذ قيمة العدد الاخر.
السطر 3 يعرف متغير z من النوع int و الذي سنضع فيه مجموع العددين .
السطر 4 يقوم بجمع قيمة x مع y ويضعها في المتغير z .
السطر 5 ينهي الدالة ويسند اليها قيمة z الناتجة من جمع العددين.

السطر 6 وهو يشير الى بداية الدالة الرئيسية
السطر 7 يعرف المتغير c من النوع int لنضع فيه مجموع العددين.
السطر 8 يقوم باستدعاء دالة الجمع التي عرفناها فوق وسيند قيمتها الى المتغير c .
السطر 9 يقوم بطباعة الجملة =3+5 متبوعة بقيمة المتغير c .

الان اريدك ان تسترخي قليلا لاننا سنعرف ما الذي سيحصل لو قمت بتنفيذ هذا البرنامج
بداية ضع في ذهنك الاتي
1- ان كل برنامج بلغة سي++ يبدا التنفيذ من عبارة main() .

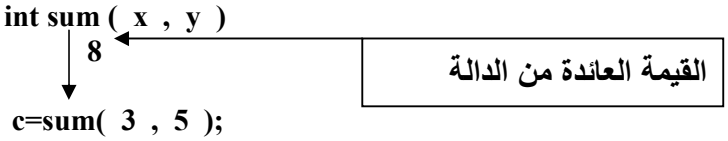
2- كل سطر يعرف فيه متغير او اكثر مثلا int a هذا السطر سنحتاجه مرة واحدة فقط في البرنامج وهي عند مرحلة ترجمة البرنامج compiling لكي يتم حجز حيز له في الذاكرة ولا يدخل هذا السطر ضمن عمل البرنامج اثناء وقت التنفيذ مطلقا.

اذن سنبدا برنامجنا من السطر رقم 8
الذي سيحدث ان العبارة sum(3,5) هي استدعاء للدالة sum التي قمنا بكتابتها حيث ان كل استدعاء لاي دالة يتم من خلال ذكر اسم الدالة مع بارامتراتها وهنا بارامترات الدالة هي القيم 3,5 حيث ان نسخة واوكد على ان نسخة من قيمة المتغيرات سترسل الى الدالة وليس المتغيرات الاصلية وحيثما اننا استدعينا الدالة بالصورة sum (3,5) وهذا يعني ان نسخة من 3 ستمرر وتعطى للمتغير x وايضا نسخة من 5 ستمرر و ستعطى الى y على الترتيب، لاحظ ان البارامتر الاول عند الاستدعاء سيعطى الى اول بارامتر في الدالة الاصلية و البارامتر الثاني الى ثاني بارامتر وهو الذي سيحصل عند السطر رقم 2 سطر بداية الدالة ،



لاحظ التشابه بين سطر تعريف الدالة وبين جملة استدعاء الدالة

بعدها ستبدا الدالة بعملها حيث اصبح الان ان المتغير x يحمل نفس قيمة 3 و المتغير y يحمل نفس قيمة 5 .
سيقوم السطر 4 (وليس السطر 3 لانه كما قلنا سطر تعريف) سيقوم السطر 4 بعملية جمع للمتغيرين x,y ويعطي ناتج الجمع للمتغير z و الذي يساوي 8 لان (8=5+3) ، اما في السطر 5 (return z) فسيتم اعطاء قيمة z وهو ناتج الجمع سيعطى للدالة sum وحيث ان كل دالة تعيد قيمة واحدة باسمها فاصبحت الدالة sum لها القيمة(8) هذه القيمة ستعطى الى المتغير c لان التنفيذ بعد انتهاء الدالة عاد الى البرنامج الرئيسي والى نفس المكان الذي استدعيت فيه الدالة c=sum (3,5) .



السطر 9 سيطبع ما موجود بين الاقواس متبوع بقيمة c والتي اصبحت تساوي 8.
السطر 10 ينهي البرنامج الرئيسي ويعيد القيمة 0 للدالة الرئيسية .

اليك مثال ثاني عن الدوال وهو مشابه للمثال السابق لكنه يعرض امكانيات اكثر للدوال من حيث الاستدعاء

```
// function example
#include <iostream.h>

int subtraction (int a, int b)
{
  int r;
  r=a-b;
  return (r);
}

int main ()
{
  int x=5, y=3, z;
  z = subtraction (7,2);
  cout << "The first result is " << z << '\n';
  cout << "The second result is " << subtraction (7,2) << '\n';
  cout << "The third result is " << subtraction (x,y) << '\n';
  z= 4 + subtraction (x,y);
  cout << "The fourth result is " << z << '\n';
  return 0;
}
```

النتائج ستكون كالاتي

```
The first result is 5
The second result is 5
The third result is 2
The fourth result is 6
```

سوف لن ادخل في تفاصيل الشرح للبرنامج كاملا لان ما يهمنا هو الدالة واستدعائها
في هذا المثال الدالة اسمها subtraction وهي دالة لطرح قيمتين a , b و الغرض من هذا المثال ككل هو لبيان الطرق الممكنة لاستدعاء الدوال. والان اريدك ان تتذكر ان الدالة تعيد قيمة واحدة باسمها (ملاحظتنا الشهيرة) وفي مرحلة تنفيذ البرنامج يمكن ان نقول ان الدالة تستبدل بالقيمة التي تعيدها .
واليك الطرق الممكنة للاستدعاء

```
z = subtraction (7,2);
cout << "The first result is " << z << '\n';
```

وهذه الطريقة نفس الطريقة التي درسناها في المثال الفائت. واعتقد انه لا داعي لاعاده شرحها.
السطر التالي

```
cout << "The second result is " << subtraction (7,2) << '\n';
```

له نفس نتيجة الاستدعاء الاول لكن الصيغة تختلف حيث استدعينا الدالة الى جملة الاخراج cout مباشرة بدون اسنادها الى متغير .

اما السطر التالي :

```
cout << "The third result is " << subtraction (x,y) << '\n';
```

فان الشئ الجديد هنا هو ان الاستدعاء تم الى قيم متغيرات مثل (x,y) بدل القيم المباشرة 7و2 وهذا شئ صحيح تماما ومقبول لان المتغيرات ايضا تحمل قيم ويتم تمرير قيم هذه المتغيرات الى بارامترات الدالة .

وايضا الحالة الرابعة :

```
z= 4 + subtraction (x,y);
```

ونستطيع كتابتها ايضا بالشكل

```
z= subtraction (x,y)+4;
```

حيث هنا تم الاستدعاء الى عملية حسابية وهي ايضا شئ صحيح تماما لان الدالة تستبدل بالقيمة العائدة بها وكانها الشكل التالي:

Z=4+2
Z=2+4

الدوال الخالية من الانواع القياسية void

لو كنت تذكر الصيغة العامة لكتابة الدوال

```
Type function_name (parameter1,parameter2, ….)
{
        statement 1;
        statement 2;
        statement 3;
        .
        .
        .
        return value
}
```

فان type يشير الى النوع البياني الذي ستعود به الدالة ، لكن ماذا لو اردنا ان لا نعيد أي قيمة من الدالة ... ؟؟؟
ففي بعض الدوال لانحتاج ان نعيد قيم من الدوال او ان الغرض من الدالة هو لطباعة جملة مثلا(وكاننا نتعامل مع بروسيجر procedure) فهل ذلك ممكن ؟ بالتاكيد هو ممكن لاننا نتعامل مع لغة سي++ ولغة سي++ لها امكانيات خارقة بالنسبة للغات البرمجة الاخرى،
في حالة لم ترغب باعادة قيمة من الدالة فيجب ان يكون النوع type الذي تعود به الدالة هو void وهذه الكلمة تعني فراغ او لاشئ ، بالاضافة الى ان الدالة يجب ان لا تحتوي على الكلمة return لانه ليس هناك قيمة سنعيدها،
واخيرا ليس هناك حاجة لاسناد الدالة الى متغير .
مثال

```
// void function example
#include <iostream.h>

void printmessage ()                    void  هو نوع الدالة
{
  cout << "I'm a function!";
}

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

```
I'm a function!
```

البرنامج التالي يحتوي على دالة لطباعة الجملة "I'm a function" اسم الدالة printmessage تظهر هذه العبارة على شاشة التنفيذ في كل مرة نستدعي فيها هذه الدالة داخل البرنامج الرئيسي، لاحظ ان النوع البياني هو void،ايضا لا وجود لجملة الاعادة return .
من الملاحظ ايضا في المثال ان الدالة printmessage( ) ليس لها أي معلمات (بارامترات) ولهذا تركت الاقواس فارغة او يمكن كتابتها بالشكل void printmessage(void) حيث void بين الاقواس تشير ان الدالة ليس لها بارامترات.
تذكر دائما ان استدعاء اي داله هو بكتابه اسمها وارفاق معلماتها بين اقواس ( ) ، عدم وجود معلمات للداله لا يعفينا من كتابه الاقواس . لذا يمكن ان نستدعي الداله printmessage بالشكل :

```
printmessage ();
```

تشير الاقواس بشكل واضح ان الاستدعاء للداله وليس لمتغير وهذه فائده من فوائد وجود الاقواس.
اما الجمله التاليه:

```
Printmessage;
```

فانها خاطئه لاستدعاء الداله ، حيث هنا الكومبايلر سيعتبرها متغير كأي متغير اخر لم يعلن عنه وهنا تكمن اهمية الاقواس
ايضا لاحظ ان استدعاء الدالة لم يسند الى متغير وانما تم الاستدعاء بذكر اسم الدالة فقط لانها لاتحمل أي قيمة .

ملاحظة مهمة جدا:
قد يتبادر الى ذهنك الى ان كل الدوال المعرّفة (void) لاتحتوي على بارامترات وهذا شئ غير صحيح اطلاقا ، فـ بارامترات الدالة تستخدم حسب احتياج الدالة اما المثال السابق فلم نذكر فيه بارامترات لاننا لانحتاج اليها في عمل الدالة لان الحاجة من الدالة هي لطباعة عبارة معينة ثابتة فقط هي I'm a function حيث بارامترات الدالة تتواجد حسب احتياج الدالة لاداء عملها
مثال:

```
// function example
#include <iostream>

void addition (int a, int b)
{
  cout<<"the result is"<<a+b;
}

int main ()
{

 addition (3,5);

  return 0;
}
```

```
the result is 8
```

فهذه الدالة هي لطباعة مجموع عددين وأؤكد لطباعة مجموع عددين وليس اعادة مجموع عددين وهناك فرق في الحالتين ، فعمل الدالة يتلخص بجمع العددين المرسلين اليها وطباعة ناتجهما فقط وليس اعادة ناتجهما فـ نوع الدالة هو void أي فراغ ولكنها تاخذ بارامترات لانها تحتاج في عملها الى بارامترات ، لذا فان بارامترات الدالة تكتب حسب حاجة الدالة .

مدى(او مجال) المتغيرات scope of variable

مدى المتغير هو المكان او المجال الذي يكون المتغير فيه صالح للعمل، فمثلا لو عرفنا متغير مثل a داخل جسم الدالة الرئيسية فان استخدامه يتم داخل الدالة الرئيسية فقط ولا يجوز استخدامه خارجها ويسمى المتغير المعرف بهذه الطريقة بمتغير محلي local variable للدالة الرئيسية وفي حالة نريد استخدام متغير بنفس الاسم داخل دالة اخرى فاننا نعرف هذا المتغير من جديد ويعتبر كمتغير محلي لتلك الدالة وهو يختلف عن المتغير المعرف في الدالة الرئيسية.
ايضا لو تم تعريف متغير داخل دالة فانه ليس من الممكن استعمال هذا المتغير داخل البرنامج الرئيسي بدون اعادة التعريف ويعتبر ايضا كمتغير محلي للدالة المعرف فيها .
ولتعريف متغير يمكن استخدامه داخل جميع اجزاء البرنامج فانه يعرف خارج أي دالة أي في جسم البرنامج ويسمى عندها بالمتغير العالمي global variable وهو معرف في كل اجزاء البرنامج

```
#include <iostream.h>

int age;     //global variable متغير عالمي

void print_info(char n[20])
{
        cout<<"name is : "<<n<<'\n';
        cout<<"age = "<<age;
}

void main()
{
        char name[20]="my name";    //local variable
        cin>>age;
        print_info(name);
}
```

المتغير age هو متغير عالمي global تم تعريفه خارج جسم الدالة الرئيسية ولذا امكننا استخدامه داخل أي مكان من البرنامج كما ترى سواء في الدالة الرئيسية او الدالة print_info بسهولة ودون الحاجة الى اعادة تعريفة ، اما المتغير name فانه متغير محلي local للدالة الرئيسية لذا لا يمكن استخدامة في الدالة print_info ويمكن ان تعرف متغير بنفس الاسم name داخل الدالة print_info وباي نوع تريده مع الاشارة الى انه مستقل عن المتغير name في الدالة الرئيسية ولا يوجد أي علاقة بينهما ويعاملان كانهما متغيرين مختلفين.

الارسال بواسطة المرجع والقيمة
Passed by value and reference

لقد قلنا سابقا اننا مررنا نسخة من البارامترات الى الدوال وليس البارامترات الاصلية هل تذكر ذلك ؟؟ اتوقع انك تذكر ، لذا سناتي هنا على ايضاح بعض الاشياء المهمة .
في جميع امثلتنا السابقة كنا نمرر نسخ من البارامترات الى الدوال وفي هذه الحالة فان أي تغيير نجريه على هذه البارامترات في داخل الدوال(تغيير في قيمة البارامتر مثلا) سوف لن يؤثر على القيم الحقيقية للمتغيرات داخل الدالة الرئيسية لاننا لم نرسل المتغيرات الحقيقية ولكننا ارسلنا نسخ عنها (وبمعنى اخر ان التغيير الذي يتم على البارامتر داخل الدالة سوف يكون غير مرئي من قبل البرنامج الرئيسي) تسمى هذه الحالة التمرير بواسطة القيمة(pass by value) أي اننا مررنا او ارسلنا قيم المتغير.

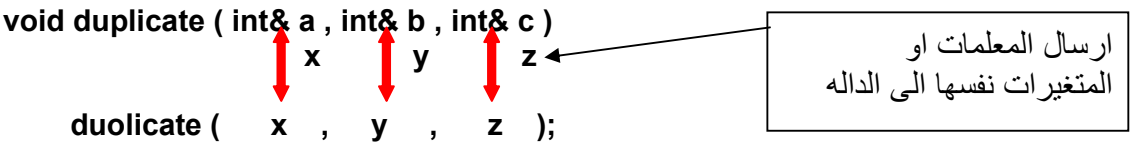
وفي احيان اخرى نحتاج ان نجري تغيير على القيمة الاصلية للمتغير المرسل بحيث ان هذاالتغيير على قيمة البارامتر داخل الدالة سيعود هذا التغيير نفسه الى المتغير الاصلي في الدالة الرئيسية وتسمى هذه الحالة التمرير بالمرجع (pass by reference) أي ان التغيير على البارامترات داخل الدالة سيرجع الى المتغير الاصلي.
والان اليك مثال توضيحي عن الحالة الثانية التمرير بالمرجع واما الحالة الاولى التمرير بالقيمة فهو كما موجود في جميع امثلتنا السابقة ولا يوجد اختلاف كبير في الصيغة بين النوعين

التمرير بالقيمة pass by reference

```
// passing parameters by reference
#include <iostream.h>

void duplicate (int& a, int& b, int& c)
{
  a*=2;
  b*=2;
  c*=2;
}

int main ()
{
  int x=1, y=3, z=7;
  duplicate (x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}
```

```
x=2, y=6, z=14
```

هذا البرنامج يحتوي على دالة اسمها duplicate وضيفتها مضاعفة القيم التي ترسل اليها
الشئ الجديد و الوحيد هو وجود علامة الجمع المنطقي (&) في سطر تعريف الدالة بعد النوع البياني للباراميتر
وقبل الباراميتر نفسه وهي دليل على ان المتغيرات a,b,c تم تمريرهم بواسطة المرجع by reference اما لو
سألتني عما جرى في البرنامج فاني ساخبرك انه:
عند استدعاء الدالة duplicate فان المتغيرات x و y و z ستعطى الى بارامترات الدالة والتي هي a و b و c على
التوالي أي انه اصبح الان a=1 و b=3 و c=7 وما قامت بة الدالة هو مضاعفة قيم كل من a,b,c وعند انتهاء
العمل في هذه الدالة تقوم العلامة & بالسماح بخروج قيمة a الجديدة والتي اصبحت تساوي 2 واسنادها الى المتغير
x في البرنامج الرئيسي وكذلك اعطاء قيمة b و التي اصبحت تساوي 9 الى المتغير y في البرنامج الرئيسي وكذلك
قيمة c ستعود الى المتغير z لذا فان المتغيرات في الدالة الرئيسية ستتغير بعد الاستدعاء للدالة duplicate لان
ارسالها تم بواسطة المرجع، وفي حالة عدم وجود علامات & فان قيم البارامترات لن تعود الى البرنامج الرئيسي
وتكون نتيجة التنفيذ للبرنامج هي x=1, y=3, z=7 .



ملاحظة :
لقد اخبرتك سابقا ان كل دالة تعيد قيمة واحده باسمها ، لكن الان اصبح بامكانك وبواسطة هذه الطريقة ان تعيد اكثر
من قيمة واحدة وحسب ما تحتاج اليه .

اليك المثال التالي:

```
// more than one returning value
#include <iostream.h>

void prevnext (int x, int& prev, int& next)
{
  prev = x-1;
  next = x+1;
}

int main ()
{
  int x=100, y, z;
  prevnext (x, y, z);
  cout << "Previous=" << y << ", Next=" << z;
  return 0;
}
```

```
Previous=99, Next=101
```

الدالة في المثال تقوم بايجاد القيمة السابقة previous و القيمة اللاحقة next للعدد المرسل

لاحظ ان المعلمه الاولى هي الرقم المراد ايجاد القيمه السابقه واللاحقه له ولا نحتاج ان نعيد منه قيمته لذا لم نضع في تعريف الداله له علامه & , بينما الرقم السابق واللاحق يتم ايجادهما داخل الداله ونحتاج اليهما خارج الداله لذا وضعنا عند تعريف الداله لهما علامات & . أي ان المعلمه الاولى تم ارسالها بواسطه القيمه بينما المعلمتان الاخرتان تم ارسالهما بواسطه المرجع .

القيم الافتراضية للبارامترات
Default values of parameters

عند الاعلان عن داله فانه يمكن ان نحدد قيمه افتراضيه لكل معلمه . هذه القيمة ستستعملها الداله في حاله تركنا مكان المعلمه فارغ اي لم نرسل للداله قيمه في مكان تلك المعلمه . ولعمل ذلك يجب ان نستخدم علامه الاسناد (=) و القيمه الافتراضيه التي سنضعها للمعلمه في تعريف الداله . حيث في حاله لم نمرر للداله اي قيمه في تلك المعلمه عند الاستدعاء فانها ستاخذ القيمه التي حددناها سابقا كـ قيمه افتراضيه , ولكن في حاله حددنا قيمه لتلك المعلمه عند الاستدعاء فانها ستتجاهل القيمه الافتراضيه وستستعمل القيمه المرسله بدل منها . مثلا:

```
// default values in functions
#include <iostream.h>

int divide (int a, int b=2)
{
  int r;
  r=a/b;
  return (r);
}

int main ()
{
  cout << divide (6);
  cout << endl;
  cout << divide (20,4);
  return 0;
}
```

```
3
5
```

كما نرى في المثال هناك استدعائين للداله divide الاول :

```
divide (6)
```

حيث هنا ارسلنا قيمه واحده للداله , ولكن الداله divide معرفه بمعلمتين , لذا فان الداله افترضت ان قيمه المعلمه الاخرى تساوي 2 لاننا حددناها مسبقا في تعريف الداله حيث كان تعريفها (int b = 2) . لذا سيكون ناتج استدعاء الداله هو 3 لانه ناتج (6/2) .

في الاستدعاء الثاني :

```
divide (20,4)
```

هنا يوجد معلمتان , لذا فان القيمه الافتراضيه المحدده سابقا للمتغير b (int b=2) سيتم تجاهلها وان b ستاخذ القيمه المرسله اليها وهي 4 والتي ستجعل الداله تعيد القيمه 5 لان 5=(20/4) .

زيادة تحميل الدوال
Overloaded functions

في c++ يمكن ان تاخذ دالتان مختلفتان نفس الاسم اذا كانت بارامتراتها مختلفه الانواع البيانية او مختلفة العدد , هذا يعني انك تستطيع ان تعطي نفس الاسم لعده دوال اذا كانت كل داله تختلف عن الاخرى بنوع البارامترات او عددها مثلا:

```
// overloaded function
#include <iostream.h>

int operate (int a, int b)
{
  return (a*b);
}

float operate (float a, float b)
{
  return (a/b);
}

int main ()
{
  int x=5,y=2;
  float n=5.0,m=2.0;
  cout << operate (x,y);
  cout << "\n";
  cout << operate (n,m);
  cout << "\n";
  return 0;
}
```

```
10
2.5
```

في هذه الحاله عرفنا دالتين بنفس الاسم وهو operate , لكن احدى الدالتين تقبل معلمات من النوع الصحيح int والاخرى تقبل معلمات من النوع الحقيقي float . مترجم البرنامج compiler يعرف اي من الدوال يذهب اليه في كل استدعاء من خلال المعلمات المرسله في الاستدعاء فاذا كانت المعلمات حقيقيه float فانه يستدعي الداله التي تقبل معلمات من النوع الحقيقي , اما اذا كانت المعلمات من النوع الصحيح فانه يستدعي الداله التي تاخذ معلمات من النوع int .

في الاستدعاء الاول للداله operate فان المعلمتين اللتان تم تمريرهما من النوع int , لذا فان الداله الاولى هي التي تستدعى ؛ هذه الداله تعيد قيمه حاصل ضرب المعلمتين الداخلتين لها . بينما الاستدعاء الثاني في البرنامج يرسل معلمتين من النوع float , لذا فان الداله الثانيه هي التي ستستدعى وهي تؤدي عمل مختلف عن الداله الاولى فهي تقسم المعلمه الاولى على الثانيه . لهذا تحديد عمل الداله operate يعتمد على نوع المعلمات المرسله اليها لوجود دالتين بنفس الاسم , وهذا ما يسمى بـ (زياده التحميل overloaded).

الدوال المتداخلة
Recursive functions

ظهرت هذه الخاصية في لغات البرمجة الحديثة وهي تتلخص في استدعاء الدالة لنفسها من داخلها (أي انك تجد استدعاء الدالة داخل نفس الدالة) وهذه الخاصية مفيدة في حل بعض المسائل التي تحتاج الى استدعاء متكرر بترتيب معين مثل عمليات ايجاد مضروب العدد او عمليات الفرز والترتيب التصاعدي والتنازلي وعمليات البحث وغيرها من العمليات المتكررة والمرتبة .
فمثلا لايجاد مضروب العدد 5 فان الحل الرياضي يكون بالطريقة التالية :

5!= 5 * 4 * 3 * 2 * 1

ويمكن ان يكتب بالصورة

5!= 5 * (4!)

لان مضروب 4 يكتب بالصورة :

4!= 4 * 3 * 2 *1

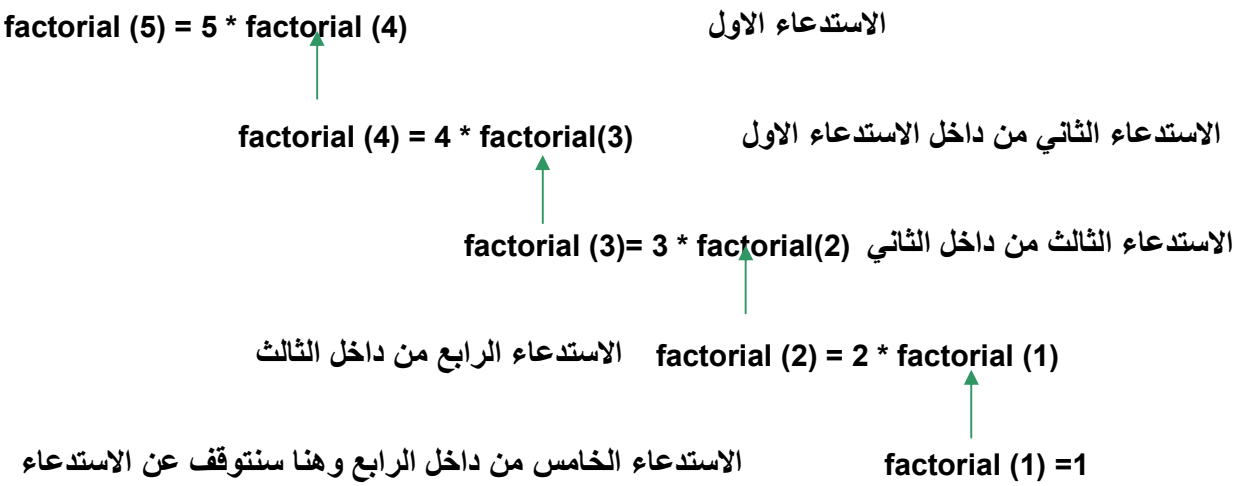
وان مضروب 4 يمكن ان يكتب 4* (!3) وان مضرو ب 3 يمكن ان يكتب 3*(!2) وان مضروب 2 يكتب 2*(!1)
وان مضروب 1=1 وباعادة التعويض العكسي نجد مضروب 5

```
// factorial calculator
#include <iostream.h>

long factorial (long a)
{
  if (a > 1)
   return (a * factorial (a-1));
  else
   return (1);
}

int main ()
{
  long number;
  cout << "Please type a number: ";
  cin >> number;
  cout << number << "! = " << factorial (number);
  return 0;
}
```

```
Please type a number: 5
5! = 120
```

هنا تلاحظ وجود استدعاء للدالة factorial من داخل نفس الدالة
الذي سيحصل عند تنفيذ البرنامج هو ان البرنامج سينتظر منك ادخال قيمة للمتغير number ولتكن اربعة مثلا ، وسيتم ايجاد المضروب لهذا المتغير حيث سترسل قيمة هذا المتغير للدالة factorial وتمرر الى المتغير a وهنا الدالة ستبدا بعملها حسب الاتي:

حسب ما قلنا في الحل الرياضي ان مضروب الاربعة مثلا هو(!3 *4) الا اننا نحتاج لايجاد !3 اولا وان !3 هو (!3*2) ايضا هنا نحتاج الى ايجاد !2 وان !2 هو (!1 * 2) وان !1 يساوي واحد



وعند التعويض العكسي عن كل استدعاء من الاسفل الى الاعلى سيتم حساب المضروب تدريجيا حتى نصل الى اول استدعاء فتنتهي الدالة هنا باعادة القيمة المطلوبة.
يجب ان نتعامل بعناية وحذر مع الدوال المتداخلة حيث يجب ان يتوقف الاستدعاء المتداخل عند شرط ما والا ستستمر الدالة في استدعاء نفسها الى قيم خارج المدى مسببة اخطاء جسيمة

مثال اخر

```
// factorial calculator
#include <iostream.h>

int power (int a,int b)
{
  if (b<=1)
        return a;
  else
        return (a * power(a,(b-1)));
}

int main ()
{
  int x=2,y=3;
  cout<<power(x,y);
  return 0;
}
```

```
8
```

هذه الدالة المتداخلة تجد القوة المرفوع لها المتغير a مرفوع للقوة(الاس) b حيث هنا الدالة تستدعي نفسها بعدد مرات b وعند الاستدعاء التالي تكون b قلت بمقدار واحد لغاية الوصول الى b=1 عندها يبدا التعويض العكسي و ستعيد a كناتج لها هذا الناتج سيعوض عند الدالة التي استدعتها وناتج هذه الدالة سيعود الى الدالة التي استدعتهاايضا وهكذا

الحل الرياضي لايجاد قوة الرقم 2 مرفوع للاساس 3 أي انه 2^3

2 ^3 = 2 * 2 * 2 = 8

التمثيل داخل الدالة المتداخلة

power(2,3)=2 * power(2,2)

power(2,2)= 2 * power (2,1)

power(2,1)=2

نلاحظ ان في كل استدعاء نضرب الاساس 2 في القيمة العائدة من الدالة وبعدد مرات الاس 3 ولا يدخل الاس في العملية الحسابية لان الغرض منه هو عدد مرات التكرار فقط.

التصريح عن الدوال (prototype)
Declaring functions

ختاما اود ان ابين شئ مهم جدا في الدوال ونصيحتي لك هي ان تستخدم الطريقة التالية عند كتابة أي دالة فهي دلالة على السبك القوي للبرنامج وتلافي اخطاء قد لا يفهمها المبرمج خصوصا المبرمجين المبتدئين
اليك المثال التالي

```
// declaring functions prototypes
#include <iostream.h>


void odd (int a)
{
  if ((a%2)!=0) cout << "Number is odd.\n";
  else even (a);
}

void even (int a)
{
  if ((a%2)==0) cout << "Number is even.\n";
  else odd (a);
}


int main ()
{
  int i;
  do {
    cout << "Type a number: (0 to exit) ";
    cin >> i;
    odd (i);
  } while (i!=0);
  return 0;
}
```

المفروض ان هذا البرنامج يقوم بمعرفة الرقم المدخل فردي ام زوجي باستخدام معامل باقي القسمة (%) ويستخدم دالتين لهذا الغرض .
الدالة الاولى ood لاختبار هل ان الرقم المدخل فردي وكما هو واضح تقوم بعملية اختبار للبراميتر a الممرر لها فاذا كان باقي القسمة على 2 لايساوي 0 فانه سيطبع عبارة .Number is odd (الرقم فردي)، اما لو كان باقي القسمة غير ذلك فانه سيستدعي الدالة الاخرى even لاختبار هل ان الرقم المدخل هو زوجي بنفس الاسلوب مع اختلاف الشرط وعبارة الطباعة وتلاحظ ايضا انه اذا لم يكن الرقم زوجي فان الدالة ستستدعي الدالة الاخرى odd

و الان قم باختيار الامر Run من قائمة debug او اضغط على المفتاحين ctrl+F9 (وهي الطريقة التي افضلها انا) لتنفيذ البرنامج ولكن ... لاتصاب بخيبة الامل فالبرنامج لن يعمل وستظهر رسالة خطا تبين ان هناك استدعاء لدالة غير معرفة هي even .. اليس هذا شئ غريب لقد قمنا بتعريف الدالة even وقمنا بكتابتها بصورة صحيحة فما الخطا في ذلك ؟؟ هل جن جنون الكومبايلر .. ؟؟؟ ، لا يا عزيزي الكومبايلر سليم 100/100 ولكن المشكلة تكمن في التالي :

ان الكومبايلر compiler يقوم بعملية تفسير البرنامج من اعلى البرنامج الى اسفله وعند وصوله الى الدالة odd وجد فيها استدعاء للدالة even (النظرية لحد الان صحيحة وسليمة) ولكن هذا الاستدعاء للدالة even تم قبل ان يصل الكومبايلر الى المكان الذي تم فيه الاعلان عن الدالة even وكانك تقوم باسناد قيمة الى متغير ومن ثم تقوم

بتعريف هذا المتغير مثلا

```
Int main()
{
x=4;
int x;
cout<<x;
return 0
}
```

ففي هذه الحالة الكومبايلر سيعترض على هذا البرنامج مشيرا الى ان هناك متغير لم يتم تعريفة نفس الامر الذي حدث في المثال السابق عن الدوال . فما الحل ..
حسنا .. الحل هو لو انك المحت للكومبايلر ان هناك دالة اسمها even من النوع void وتاخذ باراميتر من النوع الصحيح ولكنك لن تقوم باخبار الكومبايلر عن تفاصيل هذه الدالة ومحتواها الان وانك ستخبره عن محتوى الدالة لاحقا وفي مكان اخر من البرنامج فانه سيقبل عرضك هذا بكل رحابه صدر لان كومبايلر لغة سي++ متسامح بعض الشئ .

هذا الامر يحدث عمليا لو انك قمت بذكر تعريف بسيط للدالة فقط هو تعريف الدالة الاصلي بدون جسم الدالة هذا التعريف يكون متبوع بالفارزة المنقوطة (;) للدلالة على ان هذا السطر هو سطر تعريف عن دالة . اما الدالة فانها ستوضع بعد الدالة الرئيسية وستكتب كاملة من سطر التعريف و جسم الدالة واليك المثال السابق بعد اجراء التعديل عليه وهذه المره سيعمل بكل تاكيد

```
// declaring functions prototypes
#include <iostream.h>

void odd (int a);
void even (int a);

int main ()
{
  int i;
  do {
    cout << "Type a number: (0 to exit) ";
    cin >> i;
    odd (i);
  } while (i!=0);
  return 0;
}

void odd (int a)
{
  if ((a%2)!=0) cout << "Number is odd.\n";
  else even (a);
}

void even (int a)
{
  if ((a%2)==0) cout << "Number is even.\n";
  else odd (a);
}
```

```
Type a number (0 to exit): 9
Number is odd.
Type a number (0 to exit): 6
Number is even.
Type a number (0 to exit): 1030
Number is even.
Type a number (0 to exit): 0
Number is even.
```

ستشاهد ان مكان الدالتين استبدل بسطر التعريف لكل منهما حيث هنا عندما ياتي الكومبايلر ليترجم البرنامج سيعرف ان هناك دالة اسمها **odd** من النوع **viod** تاخذ باراميتر واحد وايضا هناك دالة اخرى اسمها **even** ايضا من النوع **void** الان اصبح الكومبايلر لديه خبر مسبق ان هناك دالتين **odd** و **even** ولكنه لا يعرف محتوى هاتين الدالتين لذا فانه سيتعامل معهما في الاستدعاء بصورة طبيعية ولكنه لابد ان يقوم بعملية ترجمة لهما حيث الدالة كاملة سنقوم بكتابتها بعد نهاية الدالة الرئيسية وهذا ما تم حدوثه .

ملاحظات:

* الذي يميز الاعلان عن الدالة في بداية البرنامج هو وجود الفارزة المنقوطة لذا انتبه لها جيدا

* يمكنك في سطر الاعلان عن الدالة ان تذكر عدد البارامترات ونوعها فقط دون الحاجة الى ذكر اسم الباراميتر مثلا هذا الاعلان :

```
Int  add (int a, int b);
```

يمكن ان يكتب بالشكل

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

* عند كتابة اكثر من دالة بهذه الطريقة فيجب ان تراعي الترتيب بين التصريح عن الدالة وترتيب الدوال بعد البرنامج الرئيسي حيث التصريح الاول للدالة الاولى و الثاني للثانية وهكذا.

* في هذه الطريقة من كتابة الدوال سوف لن تحصل على اخطاء عند الاستدعاءات بين الدوال المختلفة وايضا لن تفكر بعد الان ان هذه الدالة يجب تعريفها قبل تلك وما شابة ذلك

اضف الى ذلك ان شكل البرنامج سيبدو اكثر جمالية واناقة ووضوح حيث جميع الدوال امامك معروفة من مجرد النظر الى اعلاناتها وايضا وضوح الدالة الرئيسية عن الدوال الاخرى حيث شكل البرنامج سيكون جميل ومرتب، لذا عليك من الان ان تبدأ بكتابة برامجك بهذه الطريقة لانها هي طريقة المبرمجين المحترفين ولم نستعمل هذه الطريقة في بداية الكتاب لكي يكون الشرح بسيطا فقط ولذا عليك من الان ان تكون دوالك كلها مكتوبة بهذه الطريقة.

تم بعون الله

مثنى عبد الرسول محسن الفرطوسي
كلية شط العرب الجامعة
قسم علوم الحاسبات
2006
Compiler_x@ yahoo.com