الفصل 1: البرامج الجزئية: الدوال والإجراءات

1. مقدمة

البرنامج هو مجموعة من التعليمات المتسلسلة لحل مشكل معين. ولإيجاد طريقة الحل (الخوارزمية) يجب تقسيم المشكلة إلى مشاكل جزئية مختلفة يكون حلها أقل تعقيدًا. حيث يمكن حل هذه المشاكل الجزئية باستخدام برامج جزئية.

2. تعريفات

البرنامج الجزئي: هو مجموعة من التعليمات المستقلة التي لها اسم تستدعى للتنفيذ ويكون المستدعي اما البرنامج الرئيسي او برنامج جزئي اخر. عندما يصل البرنامج اثناء التنفيذ الى التعليمة التي تستدعي الاجراء يتحول سياق التنفيذ إلى محتوى البرنامج الجزئي وما أن ينتهي من تنفيذه حتى يعود الى تنفيذ التعليمة التي تلي الاستدعاء مباشرة.

تعرف البرامج الجزئية أيضا بـ: إجراءات (Procédures) دوال او توابع (fonctions) منهجيات (méthodes) روتين (routines) روتين فرعي (subroutines).

فوائد استخدام البرامج الجزئية:

- المقروئية: استعمال البرامج الجزئية ينظم البرنامج ويبسطه مما يساعد في فهم كود البرنامج
- السرعة في البرمجة: عدم تكرار نفس تسلسل التعليمات عدة مرات داخل البرنامج.
- تقليص حجم البرنامج
- يسهل عملية الصيانة
- إعادة الاستعمال: حيث يمكن تخزينها داخل مكتبات لإعادة استعمالها داخل برامج أخرى.

الاجراء :هو برنامج جزئي لا يُرجع أي قيمة في اسمه إلا انه يمكنه ارجاع النتائج عن طريق الوسائط. يمكن استخدام اسم الاجراء كتعليمة كاملة. مثلا:

خوارزم	C
`SomeProc`	`SomeProc();`
`OtherProc(x)`	`OtherProc(x);`

الدالة: هي برنامج جزئي يُرجع نتيجة واحدة إلزاما في اسمه حيث يعتبر اسمها كأنه متغير يحمل قيمة معينة لذلك يمكن استخدام استدعاء الدالة كمتغير في عمليات الاسناد والعبارات الأخرى. على سبيل المثال:

`Y← SomeFunction (X)*5`	`Y= SomeFunction (X)*5 ;`

ملاحظة: يمكن تحويل أي اجراء يرجع نتيجة واحدة كوسيط الى دالة.

التصريحات

الاجراء: التصريح عن اجراء يأخذ الشكل التالي:

خوارزم	C
(قائمة الوسائط**)** اسم_الاجراء **procedure** التصريحات المحلية **Debut** التعليمات **Fin**.	void اسم_الاجراء **(**قائمة الوسائط**){** التصريحات المحلية التعليمات **}**

الدالة: يشبه التصريح عنها التصريح عن الاجراء الا انه يجب تحديد نوع قيمة النتيجة المرجعة ويأخذ الشكل التالي:

Algorithme	C
نوع_النتيجة : (قائمة الوسائط) اسم_الدالة fonction التصريحات المحلية Debut التعليمات Fin.	نوع_النتيجة اسم_الدالة **(**قائمة الوسائط**){** التصريحات المحلية التعليمات **}**

- اسم_الاجراء أو اسم_الدالة: أي معرف مقبول
- قائمة الوسائط (اختيارية): مجموعة المتغيرات التي من خلالها يتم ادخال المعطيات واسترجاع النتائج مفصولة بـ «,» تكون داخل قوسين وتكون من الشكل اسم: نوع nomParam : type مثل (a :entier, b :réel) وتدعى بالوسائط الشكلية paramètres formels

  في لغة C قائمة الوسائط تأخذ الشكل نوع اسم type nomParam مثل (int a, float b) القوسين () ضروريين حتى ولو لم تحتوي على أي معامل.
- التصريحات المحلية (اختيارية): عبارة عن قائمة المتغيرات المحلية من الشكل: var varLoc : type
- التعليمات: مجموعة التعليمات التي ستنفذ عند مناداة البرنامج الجزئي وتكون من أي نوع. حيث يمكن استعمال جميع المتغيرات المصرح بها في قائمة الوسائط او التصريحات المحلية والتي تسمى بالمتغيرات المحلية كما يمكن استعمال المتغيرات المصرح بها في البرنامج الرئيسي وتسمى بالمتغيرات العامة
- نوع_النتيجة: عندما يكون البرنامج عبارة عن دالة يجب تحديد نوع القيمة التي سترجعها الدالة إلى البرنامج الذي استدعاها كما يجب اسناد قيمة الى اسم الدالة وفي العادة تكون آخر تعليمة في الدالة وتكون من الشكل عبارة ←اسم_الدالة حيث يتصرف اسم الدالة كمتغير خاص يحتوي على القيمة المرجعة من طرف الدالة.

في لغة C بمكن الاستغناء عن نوع النتيجة إذا كان البرنامج الجزئي عبارة عن اجراء لكن بعض الإصدارات تستخدم كلمة void والتي تعني ان الدالة لا ترجع شيئا كما تستخدم كلمة **return** لاسناد قيمة الى اسم الدالة

- **return**: تؤدي التعليمة **return** الى مغادرة البرنامج الجزئي واستئناف البرنامج الذي استدعاه عند التعليمة التي تلي الاستدعاء مباشرة. حيث يمكنها ارجاع قيمة الى البرنامج الذي استدعى البرنامج الجزئي إذا كان عبارة عن دالة.

الشكل: return [<expression>] ;

مثال:

إذا كان دالة `return 5*x ;`

إذا كان اجراء (أي دالة من نوع void) `return ;`

ملاحظات هامة:

- لمعرفة الوسائط نطرح السؤال ما الذي نعطيه للبرنامج الجزئي كمدخلات وما الذي يرجعه كنتائج.
- يجب أن تكون قائمة المعاملات في جزء تعريف البرنامج الجزئي هي نفسها المستخدمة في جزء الاستدعاء عددا ونوعا.
- يسمى السطر الأول من تصريح الدالة او الاجراء أي: نوع الدالة، اسم الدالة او الاجراء، نوع وترتيب وعدد الوسائط، ما عدا أسماء الوسائط بـ الراس entête او النموذج الأولي prototype.
- لا يتم تجميع الوسائط ان كانت من نفس النوع مثل (x, y:entier) وانما نضع (x:entier, y:entier) (int x, int y)
- أي نوع ارجاع يختلف عن **void** يدل على ان البرنامج عبارة عن دالة وليس اجراء.
- `void main()` او ببساطة `main()` عبارة عن اجراء، اما `int main()` عبارة عن دالة لذلك تحتاج **return.**
- `scanf()` و `printf()` عبارة عن دالتين مصرحتين داخل المكتبة `stdio`

مكان التصريح عن البرامج الجزئية: في الخوارزمية يكون بعد التصريح عن المتغيرات وقبل كلمة **debut** الخاصة بالبرنامج الرئيسي

في برنامج C يتم التصريح بها قبل التصريح بالدالة الرئيسية ()main

ملاحظة: ترتيب البرامج مهم اذ يجب تعريف كل دالة قبل استعمالها. أي إذا كانت الدالة ()f1 تستدعي الدالة ()f2 فيجب تعريف الدالة ()f2 قبل الدالة ()f1.

الاستدعاء

- لاستدعاء وتنفيذ **الاجراء** نستعمل اسمه كتعليمة مستقلة ونسند قيم و/او متغيرات الي الوسائط بين قوسين وتدعى بالوسائط الفعلية paramètres effectifs. كما يمكن الاستغناء عن القوسين في حالة عدم وجود أي وسيط اما في C فهي الزامية.
- نفس الشيء بالنسبة لاستدعاء **دالة** حيث يعتبر اسمها كأنه متغير يحمل قيمة معينة لذلك يمكن استخدام استدعاء الدالة كمتغير في عمليات الاسناد والعبارات الأخرى.
- يجب أن تتوافق المعاملات الفعلية من حيث العدد والنوع والترتيب مع المعاملات الشكلية.

امثلة على الدوال

- مربع عدد تأخذ عدد وترجع مربعه **fonction** carre(x :réel) : réel
- مساحة مستطيل تأخذ عددين وتجع المساحة **fonction** surface(long :réel, large :réel) : réel
- حل معادلة من الدرجة الأولى يأخذ معاملين ويرجع حل واحد **fonction** eq1(a :réel, b :réel) : réel
- مجموع جدول يأخذ جدول ويرجع المجموع **fonction** som(t :tableau de réel, size :entier) : réel
- معرفة هل العدد اولي ام لا **fonction** estPremier(x : entier) : booléen

امثلة على الإجراءات

- يظهر على الشاشة اعداد الأقل من حد معين حيث يأخذ الحد الأعلى ولا يرجع أي شيء

procedure afficheNbs(n : entier)

- اظهار قيم جدول على الشاشة يأخذ جدول ولا يرجع أي شيء

procedure afficheTab(t :tableau de réel, n :entier)

- حل معادلة من الدرجة الثانية حيث يأخذ ثلاث معاملات ويرجع حلين

procedure eq2(a : entier, b : entier, c : entier, var x1 : entier, var x2 : entier)

مثال:

اسم البرنامج	**Algorithme** test
متغير عام	**var** z : réel
اسم الاجراء الذي يأخذ متغيرn من نوع صحيح كوسيط	**procedure** afficheNbrs (n:entire)
متغير محلي	**var** i:entier
بداية الاجراء	**Début**
تعليمات الاجراء	**pour** i←1 **à** n **faire** Ecrire(i) **finpour**
نهاية الاجراء	**Fin**

fonction sommeNbrs (x:entier, y:entier) : entier	اسم الدالة التي تأخذ متغيرين صحيحين وترجع نتيجة صحيحة. لا يتم تجميع x وy رغم كونهما من نفس النوع. x, y:entier مرفوضة
Début	بداية الدالة
sommeNbrs ←x+y	اسم الدالة يتصرف كمتغير ويأخذ نتيجة المجموع
Fin	نهاية الدالة
Début	بداية البرنامج
afficheNbrs (5)	استدعاء الاجراء afficheNbrs حيث يتم اسناد 5 الى n فيقوم الاجراء بإظهار الاعداد من 1 الى 5
z←sommeNbrs (5, 3)	استدعاء sommeNbrs فيقوم البرنامج بإسناد القيمة 5 لـx والقيمة 3 للمتغيرy ثم يقوم بحساب المجموع واسناده الى z
Ecrire("la somme est ", z)	يظهر على الشاشة la somme est 8
Fin.	نهاية البرنامج

مثال C:

`#include <stdio.h>`	استخدام المكتبة stdio داخل البرنامج
`float z ;`	متغير عام
`void afficheNbrs (int n)`	اسم الاجراء الذي يأخذ متغيرn من نوع صحيح كوسيط
{	بداية الاجراء
`int i ;`	متغير محلي
`for (i=1 ; i<=n; i++)` `printf("%d\t",i);`	تعليمات الاجراء
}	نهاية الاجراء
`int sommeNbrs (int x, int y)`	اسم الدالة التي تأخذ متغيرين صحيحين وترجع نتيجة صحيحة. لا يتم تجميع x وy رغم كونهما من نفس النوع. int x, y مرفوضة
{	بداية الدالة
`return x+y ;`	اسم الدالة يتصرف كمتغير ويأخذ نتيجة المجموع
}	نهاية الدالة
`int main()`**{**	بداية الدالة الرئيسية
`afficheNbrs (5);`	استدعاء الاجراء afficheNbrs حيث يتم اسناد 5 الى n فيقوم الاجراء بإظهار الاعداد من 1 الى 5
`Z=sommeNbrs (5, 3);`	استدعاء sommeNbrs فيقوم البرنامج بإسناد القيمة 5 لـx والقيمة 3 للمتغيرy ثم يقوم بحساب المجموع واسناده الى z
`printf("la somme est %d", z);`	يظهر على الشاشة la somme est 8
return `0 ;`**}**	نهاية الدالة الرئيسية

3. المتغيرات المحلية والمتغيرات العامة

- **المتغير العام او الشامل (variable globale)** هو متغير يتم الإعلان عنه خارج جسم أي برنامج جزئي، وبالتالي يمكن استخدامه في أي مكان في البرنامج. نظرًا لأن المتغير عام، فليس من الضروري تمريره كمعامل لاستخدامه في البرامج الجزئية. اما مدة حياته أي وجوده بالذاكرة فيتم إنشاؤه عند تحميل البرنامج الى الذاكرة ولا يتم حذفه الا عند انتهاء البرنامج من التنفيذ.

- **المتغير المحلي (variable locale)** هو متغير لا يمكن استخدامه إلا داخل البرنامج الجزئي أو الكتلة حيث يتم تعريفه ويتم إنشاؤه عند استدعاء الدالة ويتم حذفه عند الانتهاء من تنفيذها.
- ينصح باستعمال المتغيرات المحلية والوسائط بدل العامة لتجنب الأخطاء وضمان استقلالية الدالة.

مثال:

Algorithme glob_loc	
Var glob, b : entier	متغيرات عامة
Procedure tst	
Var b, loc : entier	متغيرات محلية
Début	
glob←11	يمكن الوصول الى المتغيرات العامة داخل الاجراء
b←22	متغيرb المحلي يقوم بإخفاء المتغيرb الشامل
loc←33	
Ecrire(“dans tst : glob=”, glob, “b=”, b, “loc=”, loc)	
End	
Début	
glob←1	
b←2	متغيرb هو متغير عام
Ecrire(“avant tst : glob=”, glob, “b=”, b)	لا يمكن الوصول الى المتغيرات المحلية مثل loc
tst	استدعاء الاجراء
Ecrire(“après tst : glob=”, glob, “b=”, b)	
end	

مثال C:

`#include <stdio.h>`	
`int glob, b ;`	متغيرات عامة
`tst(){`	
`int b, loc ;`	متغيرات محلية
`glob=11;`	يمكن الوصول الى المتغيرات العامة داخل الاجراء
`b=22;`	متغيرb المحلي يقوم بإخفاء المتغيرb الشامل
`loc=33;`	
`printf(“dans tst : glob=%d b=%d loc=%d”, glob, b, loc);`	
`}`	
`int main(){`	
`glob=1;`	
`b=2;`	متغيرb هو متغير عام
`printf(“avant tst : glob=%d b=%d”, glob, b);`	
لا يمكن الوصول الى المتغيرات المحلية مثل loc	
`tst();`	استدعاء الاجراء
`printf(“après tst : glob=%d b=%d”, glob, b);`	

```
    return 0 ;}
```

الشاشة:

avant tst : glob=1 b=2
dans tst : glob=11 b=22 loc=33
après tst : glob=11 b=2

الشرح:

قبل استدعاء tst	اثناء استدعاء tst	بعد استدعاء tst
glob 1, b 2	glob 11, b 2; tst: b 22, loc 33	glob 11, b 2

قبل الاستدعاء لا يوجد الا متغيرين glob و b ولكن عند استدعاء الاجراء tst يقوم المعالج بحجز متغيرين آخرين loc و b . يمكن للإجراء الوصول الى المتغيرات العامة الا ان المتغير b المحلي يقوم بحجب b العام وعند الانتهاء من الاجراء يقوم المعالج بحذف جميع المتغيرات المحلية.

4. تمرير الوسائط

الوسائط هي المتغيرات التي يمكن من خلالها تبادل المعلومات بين البرامج اي ادخال البيانات من البرنامج المنادي الى البرنامج الجزئي و/او إخراج النتائج من البرنامج الجزئي الى البرنامج المنادي.

يوجد طريقتان لتمرير المعاملات او الوسائط

- تمرير بالقيمة (passage par valeur):

في هذا النمط يتم نسخ قيمة لمتغير الأصلي الى الوسيط (الشكلي) ويتم العمل على هاته النسخة (متغير محلي) ولا يتم تغيير المتغير الأصلي كما يمكن في هذا النمط تمرير قيمة ثابتة او عبارة ولا يشترط ان تكون متغير.
يستعمل هذا النمط فقط من اجل ادخال المعلومات للبرنامج الجزئي ولا يستعمل لاستقبال النتائج.

- تمرير بالمرجع أو العنوان او المتغير(passage par référence, adresse ou variable) :

لا يتم تمرير القيمة فقط وانما يتم تمرير مكان المتغير (العنوان) الأصلي الى المتغير الشكلي فيصبحان بمثابة متغير واحد واي تعديل للمعامل في البرنامج الجزئي المستدعى يؤدي إلى تعديل المتغير الأصلي الذي تم تمريره كمعامل.
لا يمكن في هذا النمط تمرير قيمة ثابتة او عبارة وانما يشترط ان تكون متغير لذلك يطلق عليه تمرير بالمتغير.

يستعمل هذا النمط من اجل ادخال المعلومات للبرنامج الجزئي خاصة إذا كانت كبيرة الحجم لتجنب نسخها مثل الجداول والمصفوفات كما يستعمل لاستقبال النتائج.

تستعمل كلمة **var** قبل التصريح باسم الوسيط للإشارة الى ان التمرير هو تمرير بالمتغير او بالمرجع.

لتمرير الوسائط بالعنوان في C نقوم باستعمال المؤشرات التي سنراها في الفصل الثالث من هذا الدرس حيث يُسبق اسم المعامل الشكلي بـ* وذلك اثناء التصريح واثناء الاستخدام، اما عند استدعاء الدالة فيُسبق المتغير الحقيقي بـ&

تصريح `int f(int *x)`

الاستعمال `*x=5;`

استدعاء `f(&a);`

في C++ تم إخفاء التعامل بالمؤشرات وذلك باستعمال الرمز $ اثناء التصريح فقط ويسمى بالمرجع

تصريح `int f(int $x)`

استدعاء `f(a);`

ملاحظة: لإدخال البيانات لا نستعمل كلمة **var** ولإخراج النتائج نستعمل **var** (* في C)

مثال:

تمرير بالقيمة	تمرير بالمرجع أو العنوان او المتغير

Algorithme passage_valeur
  var a, c: réel
Procedure carre (x: réel, y: réel)
Debut
  y← x*x
fin
debut
  c←0
  a←3
  ecrire("avant carre c=", c)
  carre(a ,c) // carre(3,c) يمكن استعمال
  ecrire("aprés carre c=", c)
fin
	Algorithme passage_variable
	  var a, c: réel
	Procedure carre (x: réel, **var** y: réel)
	Début
	  y← x*x
	fin
	début
	  c←0
	  a←3
	  ecrire("avant carre c=", c)
	  carre(a,c) // carre(3,c) لا يمكن استعمال
	  ecrire("aprés carre c=", c)
	fin

الشاشة	
avant carre c=0 aprés carre c=0	avant carre c=0 aprés carre c=9

مثال C:

تمرير بالقيمة	تمرير بالمرجع أو العنوان او المتغير
`#include <stdio.h>` `void carre (float x, float y){`	`#include <stdio.h>`

`y= x*x;`
`}`
`int main(){`
`float a, c;`
`c=0;`
`a=3;`
`printf ("avant carre c=%f ", c);`
`carre(a ,c);`
`// carre(a,5) يمكن استعمال`
`printf ("aprés carre c=%f ", c);`
`return` `0 ;}`
	`void carre (float x, float *y)`
	`{`
	`*y=x*x;`
	`}`
	`int main(){`
	`float a, c;`
	`c←0;`
	`a←3;`
	`printf ("avant carre c=%f ", c);`
	`carre(a,&c);`
	`// carre(a,5) لا يمكن استعمال`
	`printf ("aprés carre c=%f", c);`
	`return` `0 ;}`

الشاشة	
avant carre c=0 aprés carre c=0	avant carre c=0 aprés carre c=9

التحويل من اجراء الى دالة:

يمكن تحويل أي اجراء يرجع نتيجة وحيدة الى دالة حيث نغير كلمة **Procedure** الى **fonction** وننزل الوسيط الذي يرجعه الاجراء الى متغير محلي ونضع نوع الدالة هو نوع الوسيط وقبل انهاء الدالة نسند قيمة المتغير الى اسم الدالة.

مثلا البرنامج الجزئي الذي يحسب القيمة المطلقة لعدد حقيقي:

على شكل دالة	على شكل اجراء
fonction abs (x: réel) : réel **var** y: réel **Début** si x<0 alors y← -x sinon y← x finsi abs←y **fin**	**Procedure** abs (x: réel, **var** y: réel) **Début** si x<0 alors y← -x sinon y← x finsi **fin**
الاستدعاء	
z←abs (-5)	abs(-5, z)

`float abs (float x){`
`  float y;`
`  if (x<0)`
`  y= -x;`
`  else`
`  y= x;`
`  return y ;`
`}`
	`void abs (float x, float *y){`
	`  if (x<0)`
	`  *y= -x;`
	`  else`
	`  *y= x;`
	`}`

`float abs (float x){`
`  if (x<0)`
`  return -x;`
`  return x;`
`}`
	يمكن الاستغناء عن المتغير y كما يمكن الاستغناء عن else التي قبلها **return**

الاستدعاء	
`z=abs (-5);`	`abs(-5, &z);`

5. التراجعية او العودية la récursivité:

التراجعية هي طريقة بسيطة وأنيقة لحل بعض المسائل ذات الطابع التراجعي.

البرنامج التراجعي هو كل برنامج يستدعي نفسه. حيث أن البرنامج المعرف يُستعمل في تعريف نفسه. من الناحية التطبيقية البرنامج التراجعي هو برنامج يقوم بجزء من العمل ثم يستدعي نفسه من اجل إتمام الباقي.

ملاحظة: يمكن تحويل أي حلقة pour او tant que الي برنامج تراجعي.

شرط التوقف

نظرًا الى ان البرنامج التراجعي يستدعي نفسه، فمن الضروري توفير شرط لإيقاف التكرار، وهي الحالة التي لا يقوم فيها البرنامج باستدعاء نفسه وإلا فلن يتوقف أبدًا.

من الافضل اختبار شرط الإيقاف أولاً، وبعد ذلك، إذا لم يكن الشرط محققا، نقوم بإعادة استدعاء البرنامج حيث يؤدي الاستدعاء إلى حالة التوقف تدريجيا.

مثال:

`void affiche (int i)`
`{`
`  printf("%d",i);`
`  affiche (i +1);`
`}`
	Procédure affiche (i :entier)
	début
	ecrire(i)
	affiche (i +1)
	Fin.

مثلا نستدعيه affiche (1) فيقوم بإظهار 1 ثم يستدي نفسه من اجل i=i+1=2 فيقوم بإظهار 2 ثم الى ما لا نهاية لذلك وجب تزويد الخوارزمية بشرط الايقاف مثلا:

`void affiche (int i)`
`{`
`  if (i<10) {`
`  printf("%d",i);`
`  affiche (i +1);`
`  }`
`}`
	Procédure affiche (i :entier)
	début
	si (i<10) alors
	ecrire(i)
	affiche (i +1)
	fsi
	Fin.

الشكل العام للبرنامج التراجعي:

`void recursive(paramètres) {`
`  if (condition d'arrêt)`
`  <instructions du point d'arrêt>`
`  else`
`  {`
`  <instructions>`
`  Appel récursif (paramètres changés)`
`  <Instructions>`
`  }`
`}`
	Procédure récursive (paramètres)
	début
	si (condition d'arrêt) **alors**
	<instructions du point d'arrêt>
	Sinon
	<instructions>
	Appel récursif (paramètres changés)
	<Instructions>
	Fsi;
	Fin.

مثال:

1. العاملي

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{fact}(n-1) & \text{if } n > 0 \end{cases}$$

بالإمكان كتابة الدالة كعلاقة تكرارية:

$$b_0 = 1$$
$$b_n = nb_{n-1}$$

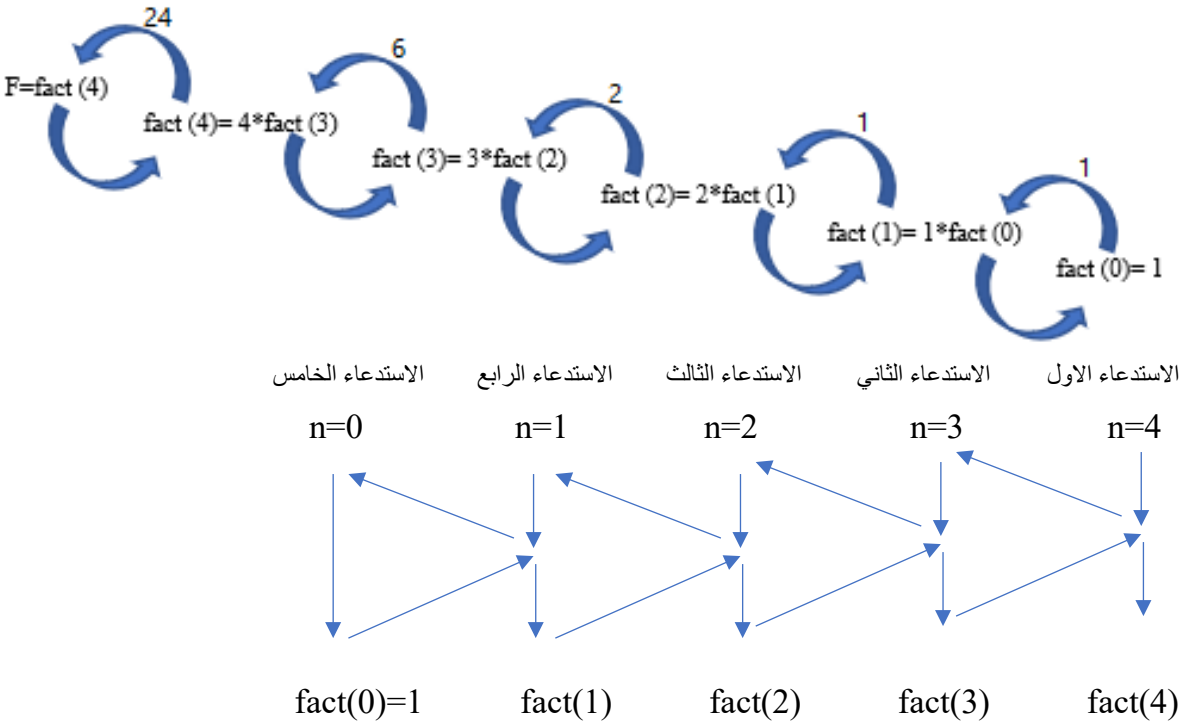
تراجعي	تكراري
Fonction fact (n : Entier) : Entier **début** **si** (n = 0) alors fact←1 **sinon** fact←n*fact (n-1) **fsi** ; **fin**.	**Fonction** fact (n : Entier) : Entier var i, f: Entier **début** f←1 **pour** i←2 **à** n **faire** f← f * i **fpour** fact←f **fin**.

`int fact (int n){`
`  if (n == 0) return 1;`
`  return n*fact(n-1);`
`}`
	`int fact (int n){`
	`  int f=1;`
	`  for (i=2 ;i<= n , i++)`
	`  f← f * i`
	`  return f;`
	`}`

كيفية العمل

نستدعي الدالة fact من اجل n=4 لحساب !4

نستدعي F=fact(4) والذي بدوره يستدعي fact(3) التي تستدعي fact(2) الى ان تستدعي fact(0) والتي تنتهي وترجع 1 مما يسمح بحساب fact(1) والتي تسمح بحساب fact(2) الى ان يتم حساب fact(4).انظر للمخطط ادناه.



مكدس التنفيذ La pile d'exécution: هو موقع من الذاكرة مخصص لحفظ المعاملات والمتغيرات المحلية ومكان تخزين النتيجة لكل برنامج جزئي قيد التشغيل.

في العادة البرمجة بالطريقة التراجعية تكون سهلة وأكثر مقروئية لكنها تستهلك قدرا كبيرا من الذاكرة فمثلا لحساب !4 نحجز في المكدس مكان لوضع النتيجة واخر لوضع المعامل n=4 ثم مكان اخر لوضع نتيجة !3 والمعامل n=3 وهكذا الى ان يتم حساب !0 ليتم نزع المعامل n=0 ثم يتم حذف المعاملات والنتائج بترتيب معاكس لما أنشئت به.

التراجعية المتبادلة: يمكن للبرنامج التراجعي أن يستدعي نفسه مباشرة أو بطريقة غير مباشرة حيث يستدعي برنامج اخر الذي بدوره يقوم باستدعاء البرنامج الاول.

مثال

لحساب π نستخدم العلاقة التالية $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ldots$ ننشئ دالتين تراجعيتين الأولى تضيف 1/n وتستدعي الثانية من اجل n=n-2 ثم تنقص 1/n والتي بدورها تستدعي الأولى لتضيف وهكذا حتى يصبح n معدوم.

```c
#include <stdio.h>

float f2(int n);

float f1(int n) {
  if (n <= 0) return 0;
  return 1. / n + f2(n - 2);
}

float f2(int n) {
  if (n <= 0) return 0;
  return -1. / n + f1(n - 2);
}

void main() {
  printf("%f\n", f1(4*100+1) * 4);
}
```

```
fonction f1(n: entier)
début
  si n<=0 alors
    f1←0
  else
    f1←1/n+f2(n-2)
  fsi
fin

fonction f2(n: entier)
début
  si n<=0 alors
    f2←0
  else
    f2←-1/n+f1(n-2)
  fsi
fin
```

الدالة f1 تحسب $\frac{\pi}{4}$ ولحساب π نضرب الناتج في 4

ملاحظة هامة: نظرا ان الدالة f1 تستدعي الدالة f2 التي لم تعرف بعد في لغة C يجب إضافة راس الدالة f2 دون جسمها (السطر الأول) قبل تعريف الدالة f1 على ان يأتي تعريفها من بعد.