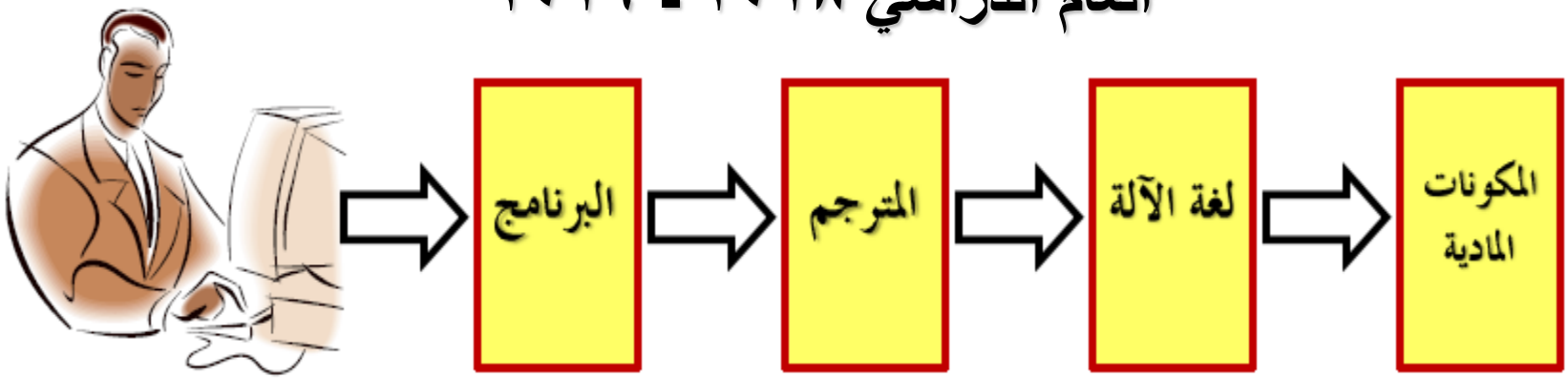
جامعة حماة
كلية العلوم في مصياف / السنة الأولى

المادة: البرمجة و الخوارزميات

المحاضرة الرابعة: الدوال (التوابع) في لغة البرمجة C++

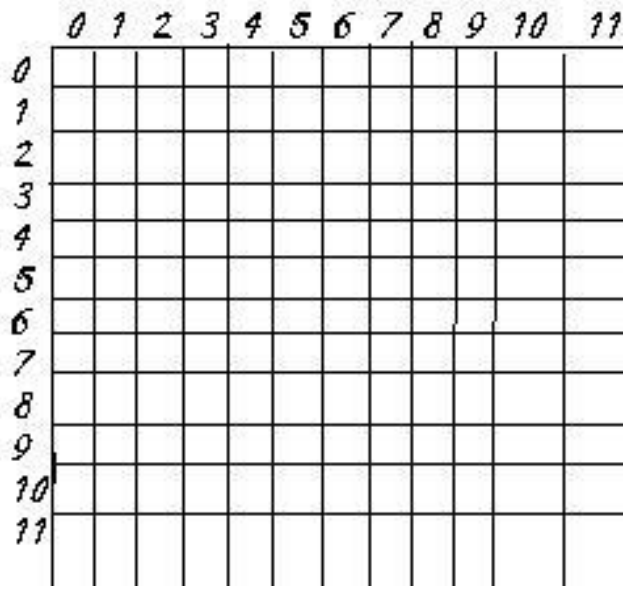
Functions in C++

العام الدراسي ٢٠١٨ - ٢٠١٩



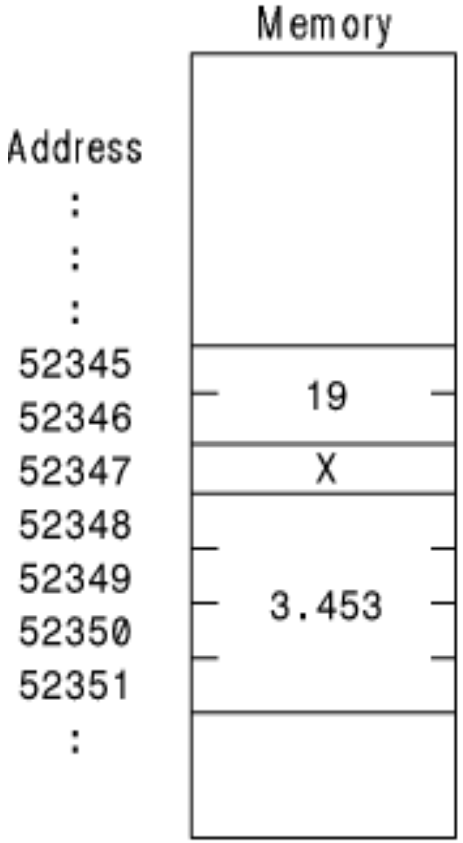
مواقع الذاكرة Memory

- يمكن تشبيه ذاكرة الحاسب بصناديق البريد، حيث تمثل كل خانة موقع في الذاكرة وتقوم بتخزين قيمة وحيدة (قد تكون صحيحة int او حقيقية float او رمز char أو منطقي boolean)، وكل موقع في الذاكرة له عنوان Address، ويتم تمثيل عناوين الذاكرة باستخدام النظام الست عشري Hexadecimal من باب التسهيل، لأنه بالأساس يمثل باستخدام النظام الثنائي Binary، فعلى سبيل المثال لتمثيل الخانة رقم 15 باستخدام النظام الثنائي فإننا سوف نحتاج إلى اربع خانات هي 1111 على عكس النظام الست عشري الذي يمثلها بخانة واحدة هي F .



عنونة المتغيّرات في ذاكرة الحاسب
Addressing and Memory

- عند تعريف المتغيّرات في بداية البرنامج، فإنها تحجز حيزاً في ذاكرة الحاسب بما يتناسب مع حجمها، فعلى سبيل المثال المتغيّرات من نوع int تحجز 4 bytes على عكس المتغيّرات من char والتي تحجز 1 byte.
- مثال :




```
main()
{
short i = 19;          ( 2 Byte)
char c = 'X';          ( 1 Byte)
float f = 3.453;       ( 4 Byte)
}
```

السؤال: هل يمكن ان نتعامل مع عنوان المتغيّر في الذاكرة بدل ان نتعامل معه بشكل مباشر ؟؟ ثم كيف يمكن ان نحصل على عنوان متغيّر ما في الذاكرة ؟

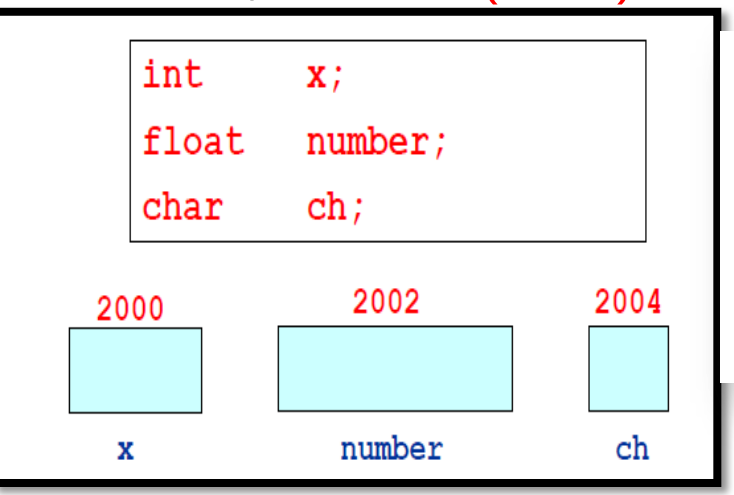
References المراجع

- كل متغيّر في ذاكرة الحاسب له عنوان Address مؤلف من (4 Byte) مكتوبة بالنظام الست عشري (ثمانية خانات ست عشرية)، و للحصول على عنوان متغيّر ما نستخدم العلامة & ونتبعها باسم المتغيّر .
- مثال : ليكن المتغيّر x من النمط integer ويحتوي على القيمة 10 أي :

```
int   x = 10;
cout<<x<<endl ;   ➔   ( 10)
cout<<&x<<endl;   ➔   (0x1BFE9C)
```

- لا يمكن التعديل على عنوان متغيّر ما ، أي أن الكتابة التالية هي خاطئة :

&x = 11; (Error)




```
int x;
float number;
char ch;
cout<< "Address of x is " << &x << endl;
cout << "Address of number is " << &number << endl;
cout << "Address of ch is " << &ch << endl;
```

```
Address of x is 2000
Address of number is 2002
Address of ch is 2004
```

المتغيّرات العامة و المتغيّرات الخاصة
Global & Local Variables

- المتغيّرات Variables التي تعرّف تحت تعريف المكتبات تسمى متغيّرات عامة تكون معرّفة بالنسبة إلى جميع أجزاء البرنامج أي على سبيل المثال لو عرّفنا متغيّر اسمه num يكون هذا المتغيّر معرّف بالنسبة إلى جميع الدوال functions() و الدّالة الرئيسية main()، على خلاف المتغيّرات الخاصة التي تعرف داخل الدوال وتكون فقط معرّفة ضمن الدّالة، وهذه المتغيّرات لا تنتهي حياتها إلا بانتهاء البرنامج main() وتبقى محافظة على قيمها الجديدة ولا تعود لقيمتها البدائية.

```
#include<     >
global variable  ←  متغيّرات عامة معرفة إلى جميع أجزاء البرنامج
main()
{
local variable  ←  متغيّرات خاصة تكون معرفة فقط داخل الدّالة (main)
}
```

- نستطيع تعريف واستخدام أكثر من دّالة في برنامج واحد

الدوال (التوابع) في Functions C++

- عندما نقوم بمعالجة مهمة ما نقوم بتقسيمها إلى مهام جزئية كل مهمة يقوم بتنفيذها دّالة ما.
- تعرّف الدّالة على أنها: جملة أو مجموعة جمل أو تعليمات ذات كيان خاص، تقوم بعملية أو مجموعة عمليات سواء كانت (عمليات إدخال أو إخراج أو عمليات حسابية أو منطقية) وتحتل الدّالة موقعا من البرنامج (أي أنها جزء منه) ويمكن القول أن برنامج C++ يتكون من مجموعة دوال.
- **فوائد الدوال :**

١. تساعد الدوال المخزنة في ذاكرة الحاسب على اختصار كود البرنامج إذ يكتفى باستعادتها باسمها فقط لتقوم بالعمل المطلوب.
٢. تساعد الدوال المخزنة في مكتبة الحاسب، أو التي يكتبها المبرمج على تلافي عمليات التكرار في خطوات البرنامج التي تتطلب عملا طويلا وشاقا.
٣. تساعد الدوال الجاهزة على تسهيل عملية البرمجة نفسها.
٤. توفر مساحة من الذاكرة المطلوبة.
٥. اختصار عمليات زمن البرمجة وتنفيذ البرنامج بأسرع وقت ممكن.

هيكلية الدوال Functions

- تقوم البرمجة الهيكلية على عدة دوال بدلاً من دّالة واحدة هي **main( )** وبإمكاننا تجزئة البرنامج إلى عدة دوال، وكل دّالة تقوم بوظيفة محددة ثم تسلمها للأخرى بعد أن تكون قد أنجزت المطلوب منها.
- الشكل العام للدّالة :

```
Type_ name_(parameter1,_ parameter2,.....)
{
Statement;
Return(type);
}
```

Header

Body

- أي دّالة بلغة C++ تتألف من قسمين أساسيين :

١. ترويسة الدّالة Function Header : توجد في بداية الدّالة وتتألف من ثلاث أجزاء :
   - Return Function Type هذا الجزء يحدد نوع البيانات الذي تعيده الدّالة .
   - Function Name: اسم الدّالة .
   - Types and Names of Parameters: أسماء وأنماط متغيّرات دخل الدّالة .

٢. جسم الدّالة Body : الجزء الذي يعبّر عن وظيفة الدّالة بحيث يعالج الدخل ويقوم بإعطاء خرج يتوافق مع مهمة الدّالة المكتوب.

هيكلية الدوال Functions

- **قائمة المتغيّرات Parameters List:**

١) يتم الفصل بين أسماء المتغيّرات بفاصلة مع ذكر نوعها.

٢) المتغيّرات هي local variables أي تموت هذه المتغيّرات بعد الخروج من الدّالة .

٣) لا يمكن تعريف متغيّر داخل الدّالة بنفس اسم متغيّر دخل للدّالة .

٤) لكتابة دّالة بدون متغيّرات إما نترك parameter list فارغة أو نكتب الكلمة void لتدل على لا شيء.

- **اسم الدّالة Function Name:**

١) يتم وضع اسم للدّالة بحيث يعبر عن وظيفته .

٢) يجب أن يكون اسم الدّالة غير محجوز مسبقاً .

٣) لا يمكن تعريف دّالة ضمن دّالة أخرى.

٤) يمكن استدعاء دّالة ضمن دّالة أخرى.

Functions الدوال

```
#include <iostream>
using namespace std;

int add(int x , int y);

void main()
{
    int first = 5;
    int second = 7;
    int result = add(first , second);
    cout<<result<<endl;
}

int add(int x , int y)
{
    return (x+y);
}
```

Prototype

Calling

Header + body

- يوجد طريقتين لكتابة دّالة بلغة C++ هما :

الطريقة الأولى : باستخدام Function Prototype .

- تصريح وتعريف: (التصريح عن الدّالة قبل تعريفها)
- يسمى هذا الشكل بـ prototype أو declaration للدّالة ، وينتهي بفاصلة منقوطة.
- يتم وضعه في global section قبل أماكن الاستدعاء ولمرة واحدة .
- ومن بعدها يتم وضع الدّالة بشكل كامل function definition في أي مكان يلي الـ prototype .

```
#include <iostream>
using namespace std;

int add(int x , int y)
{
    return (x+y);
}
void main()
{
    int first = 5;
    int second = 7;
    int result = add(first , second);
    cout<<result<<endl;
}
```

Header + body

Calling

الطريقة الثانية : باستخدام Function Definition

- التعريف فقط: (كتابة الدّالة بشكل كامل قبل أماكن استدعائها).
- يتم وضع الدّالة بشكل كامل في أي مكان ضمن global section وقبل مكان الاستدعاء ، كي يتم رؤيتها .

مثال على الدوال

```
#include <iostream>
using namespace std;
void check(int num)
 {
    if ((num%2)==0)
    cout<<"even"<<endl;
    else
     cout<<"ood"<<endl;
  }
void main()
{
int x;
cin>>x;
check (x);
}
```

- برنامج يقوم بالتحقق من رقم عشوائي مدخل بواسطة دّالة و طباعة العدد (فردي أو زوجي).
- لنفرض أننا أدخلنا القيمة x = 15
- نتيجة تنفيذ البرنامج ستكون (ood) لأن باقي قسمة (١٥) على (٢) لايساوي الصفر.
- لنفرض أننا أدخلنا القيمة x = 8
- نتيجة تنفيذ البرنامج ستكون (even) لأن باقي قسمة (٨) على (٢) تساوي الصفر.

Functions الدوال

١- دّالة بدون قيمة معادة إلى الدّالة الرئيسية main

هي الدوال التي لا تحتوي على قيمة مرجعة إلى البرنامج الرئيسي. أي أنها تنفذ التعليمات الموجودة بداخلها ولا تعيد أي قيمة خرج إلى البرنامج. قد تستقبل قيم لكنها لا تعيد أي قيمة وتعرف هكذا:

```
void name(parameter1, parameter2,…)
{
statement;
}
```

مثال: دّالة تطبع رقم معين يتم إرساله لها عند استدعائها

```
#include<iostream>
using namespace std;
void msgShow (int a)
{
cout<<" the number send is="<<a;
}
void main()
{
msgShow(3);
}
```

أمثلة على دّالة بدون قيمة معادة إلى الدّالة الرئيسية main

```
#include<iostream>
using namespace std;
void msgShow()
 {
cout<<“Welcome Dear Student";
 }

void main()
{
msgShow();
}
```

➔ Welcome Dear Student

```
#include<iostream>
using namespace std;
Draw(int i, int j);
void main()
{
Draw(5,5);
}

void Draw(int i, int j)
 {
For(int i=1;i<5;i++)  {
  For(int j=1;j<=i;j++)
cout<<j;  }
   cout<<endl;
  }
```

1
12
123
1234
12345

٢- الدوال التي تعيد قيمة إلى الدّالة الرئيسية main

- **الشكل العام لهذه الدّالة:**

```
type name(parameter1, parameter2,…….)
{
statement;
return(type);
}
```

- تعيد قيمة بعد استدعائها وتكون بالشكل التالي عند استدعائها ويجب أن تحتوي على قيمة لتعيدها للبرنامج:
  reslt =name(var1,var2,…..);
- تكتب الدّالة بعد التصريح عن المكتبات مباشرة.
- الدّالة التي تتم كتابتها تعامل داخل البرنامج حالها كحال أي دّالة من دوال اللغة.
- المتغيّرات المعرّفة داخل الدوال تنتهي حياتها بانتهاء تنفيذ أخر سطر في الدّالة.
- يوجد نوعان من الدوال التي تعيد قيمة إلى الدّالة الرئيسية main:
  ١. دّالة (تابع) الإرسال بالقيمة Value.
  ٢. دّالة (تابع) الإرسال بالمرجع Reference.

الدوال التي تعيد قيمة إلى الدّالة الرئيسية main
٢-١- دّالة الإرسال بالقيمة Value

٣- دّالة الإرسال بالقيمة Value: ترسل فقط نسخة من قيمة المتغير إلى الدّالة أي إذا تغيرت قيمة المتغير داخل الدّالة فلا تتغير قيمته الأصلية داخل الدّالة الرئيسية ()main لأننا أرسلنا فقط (نسخة عنه) إلى الدّالة لمعالجتها.

- دّالة الإرسال بالقيمة لا تعيد أكثر من قيمة واحدة إلى البرنامج الرئيسي main.
- أي تعديل على المتغيّرات داخل الدّالة سيجعل التعديل فقط داخلها من دون أن ينعكس على المتغيّر الأصلي في الدّالة الرئيسية main. .

أمثلة على الدوال التي تعيد الإرسال بالقيمة Value

١- دّالة تقوم بجمع رقميين وتعيد النتيجة إلى البرنامج لطباعته

٢- دّالة تقوم بضرب رقميين وتعيد النتيجة إلى البرنامج لطباعته

```
#include <iostream>
using namespace std;
int mult(int n1,int n2);
void main()
{
cout<< mult(3,7) <<"\n";
cout<< mult(2,3)<<"\n";
}
int mult(int n1,int n2)
  {
return(n1*n2);
  }
```

➔ 21

➔ 6

```
#include <iostream>
using namespace std;
int add(int a,int b)
  {
   return(a+b);
   }
void main()
{
cout<< add(3,7) <<"\n";
cout<< add(2,3)<<"\n";
}
```

➔ 10

➔ 5

أمثلة على الدوال التي تعيد الإرسال بالقيمة Value

١- دّالة تقوم بطباعة قيمة متغير ثم طباعته في الدّالة الرئيسية main

٢- دّالة تقوم بحساب أس رقمين وتعيد النتيجة إلى البرنامج لطباعتهما

```
#include<iostream>
using namespace std;
void pr(int x)
 {
cout<<“ x= “<<++x<<endl;
  }
void main()
{
 int num=5;
Pr(num);
cout<<“num=“<<num<<endl;
}
```

x= 6
Num=5

```
#include<iostream>
using namespace std;
int powA(int x,int n)
  {
reslt =1;
for(int i=0 ;i<n ;i++)   reslt = reslt *x;
return( reslt);
  }
void main()
{  int reslt ,x ,n ;
x=3; n=4;  reslt= powA (x,n);
cout<<" powA="<<reslt<<"\n";
x=5; n=2;  reslt= powA (x,n);
cout<<" powA="<<reslt<<"\n";
}
```

powA = 81

powA = 25

الدوال التي تعيد قيمة إلى الدّالة الرئيسية main
٢-٢- دّالة الإرسال بالمرجع Reference

٤- دّالة الإرسال بالمرجع Reference: ترسل موقع المتغيّر إلى الدّالة، أي أن الدّالة تستقبل المتغيّر نفسه المُرسل بنفس الاسم أو باسم اخر. فإذا تغيرت قيمة المتغيّر داخل الدّالة تتغير قيمته الأصلية داخل الدّالة الرئيسية main لأننا أرسلنا موقعه إلى الدّالة والتغيير يكون في محتوى الموقع. وشكلها كالتالي:

Type name(&parameter1**, &**parameter2**,……)**

{ statement; }

- نضع إشارة (&) قبل اسم كل (parameter) عندما نريد أن نعيد المتغيّر في قيمته للبرنامج بمعنى أننا سنستقبل الموقع.
- **فائدة** الإرسال بالمرجع هو أن الإرسال بالقيمة لا يعيد أكثر من قيمة متغير واحد إلى البرنامج، بينما بالإرسال بالمرجع نستطيع إعادة أكثر من قيمة متغيّر إلى البرنامج.
- أي تغيير في **parameter** سوف يؤثر على قيمته في البرنامج الرئيسي.
- .وإذا لم نضع (&)يبقى محافظا على قيمته في البرنامج الرئيسي.

أمثلة على الدوال التي تعيد الإرسال بالمرجع Reference

١- دّالة تقوم بجمع رقميين وتعيد النتيجة إلى البرنامج لطباعته

٢- دّالة تقوم بضرب رقميين وتعيد النتيجة إلى البرنامج لطباعته

```
#include <iostream>
using namespace std;
int mult(int &n1,int &n2);
void main()
{
int a=3; int b=7;
cout<<mult(&a,&b)<<"\n";
int c=2; int d=3;
cout<<mult(&c,&d)<<"\n";
}
int mult(int &n1,int &n2)
  {
return(n1*n2);
  }
```

➔ 21

➔ 6

```
#include <iostream>
using namespace std;
int add(int &a,int &b)
  {
   return(a+b);
   }
void main()
{
int n1=3; int n2=7;
cout<< add(&n1,&n2) <<"\n";
 int n3=4; int n4=5;
cout<< add(&n3,&n4)<<"\n";
}
```

➔ 10

➔ 9

أمثلة على الدوال التي تعيد الإرسال بالمرجع Reference

١- دّالة تقوم بطباعة قيمة متغيّر ثم طباعته في الدّالة الرئيسية main

٢- دّالة نرسل لها متغيّرين وتقوم بضرب كل واحد منهما بخمسة؟

```
#include<iostream>
using namespace std;
void pr(int &x)
 {
cout<<" x= "<<++x<<endl;
  }
void main()
{
 int num=5;
Pr(&num);
cout<<"num="<<num<<endl;
}
```

x= 6
Num=6

```
#include<iostream>
using namespace std;
Int mul5 (int &x,int &n) {
  x=x*5;
  n=n*5;
 return (x,n);
}
void main()
{
int x=3; int n=4;  mul5 (&x,&n);
cout<<" x="<<x<<"\n n="<<n<<"\n";
int y=5; int z=2;  mul5 (&y,&z);
cout<<" y="<<y<<"\n z="<<z<<"\n";
}
```

x= 15
n = 20

y = 25
z = 10

٣- الدوال الزائدة Function Overloading

٥- الدوال الزائدة : هي مجموعة دوال لها نفس الاسم وتختلف في القيم المعادة أو تختلف في نوع (parameter) المستقبل للدّالة.

- يمكننا إجراء ما يسمى بالتحميل الزائد للدوال أي صنع أكثر من دّالة مشتركين بنفس الاسم ومختلفين بالـ parameter list سواء بأنماط متغيّرات الدخل أو بترتيبهم أو بعددهم .
- عند استدعاء احد هذه الدوال وبما أنها جميعا بنفس الاسم، لذلك فإن المترجم سيستدعي الدّالة التي تستقبل اقرب نوع للمتغيّر الذي أرسلته لها أو نفس النوع.

- **مثال:** برنامج يحوي على الدوال الآتية احدها تجمع الرقمين والأخرى تقسمهم والأخرى تضربهم ولهم نفس الاسم

```
#include<iostream>
using namespace std;
int operation (int a, int b)
   {        return(a*b);        }
float operation (float a , int b)
   {        return(a+b);        }
float operation (float a, float b)
   {        return(a/b);        }
void main()
{  int a=3,b=2;
float c=2.5,d=4.5;
cout<<"reslt="<< operation (a,b);
cout<<"\n reslt="<< operation (c,d);
cout<<"\n reslt="<< operation (c,a);
 }
```

مثال على الدوال الزائدة

عند استدعاء احد هذه الدوال وبما أنها جميعا بنفس الاسم، لذلك فإن المترجم سيستدعي الدّالة الأنسب لعدد متغيراتها parameters التي تستقبلها و تعيد الناتج الجمع إلى الدّالة الرئيسية. كما و يلعب دور نوع للمتغير الذي أرسلته لها.

- **مثال:** برنامج يحوي على الدوال الآتية الأولى تجمع رقمين الثانية تجمع ثلاثة أرقام الثالثة تجمع أربعة أرقام ولهم نفس الاسم

```
#include<iostream>
using namespace std;
int sum (int n1, int n2)
   {        return(n1+n2);    }
int sum (int n1 , int n2 , int n3)
   {        return(n1+n2+n3);       }
int sum (int n1, int n2, int n3, int n4)
   {        return(n1+n2+n3+n4);       }
void main()
{
 int a=3, b=2, c=2 ,d=4 ;
cout<<"reslt="<< sum (a,b);
cout<<"\n reslt="<< sum (a,c,d);
cout<<"\n reslt="<< sum (a,b,c,d);
 }
```

٤- الدوال التي تتعامل مع القيم الافتراضية Default Arguments Functions

- يمكن وضع قيم افتراضية لمتغيّرات دخل الدّالة في حال عدم إدخالها !!
- مثال:

في الاستدعاء add(4) مثلاً سيتم وضع القيمة ٤ في المتغيّر n1 ، وبما أنه انتهت قيم الدخل عند الاستدعاء فإن باقي القيم ستؤخذ افتراضياً من ترويسة الدّالة .

```
#include<iostream>
using namespace std;
int add(int n1=1, int n2=2, int n3=3)
   {
return(n1+n2+n3);
   }
void main()
{
 int d=7 ;
cout<< add(1,3,6);
cout<<"\n"<< add(2,3);
cout<<"\n"<< add(4);
cout<<"\n"<< add();
cout<<"\n"<< add(4,5,d);
 }
```

```
10
8
9
6
16
```

٥- دّالة استدعاء الدّالة لنفسها (الدّالة العودية) Recursive Function

- الدوال العودية هي دوال يتم تنفيذها مرة واحدة أو أكثر ويتم الخروج منها عند عدم تحقق شرط ما يدعى شرط التوقف .
- الدّالة العودية هي تابع يقوم باستدعاء نفسه كل مرة إلى أن يصل إلى شرط التوقف فيتوقف عن الاستدعاء ويقوم بتحصيل النتائج المتراكمة عن الاستدعاءات السابقة .
- الشكل العام لهذه الدّالة:

```
type name(parameter1, parameter2,…….)
{
statement;
return( name(parameter1, parameter2,…….) );
}
```

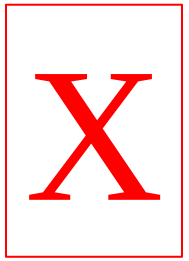
- تعيد قيمة بعد استدعائها وتكون بالشكل التالي عند استدعائها ويجب أن تحتوي على لتعيد قيمة للبرنامج:

```
reslt =name(var1,var2,…..);
```

مثال على الدّالة العودي (دّالة استدعاء الدّالة لنفسها) Recursive Function

١- مثال يقوم بإيجاد عاملي لعدد ما بشكل تكراري.

```
#include<iostream>
using namespace std;
void main()
{
int n=0;
cin>>n;
int fact=1;
for(int i=1; i<=n; i++)
   {
    fact*=i;
    }
cout<<n<<“ != “<<fact<<endl;
}
```



٢- مثال يقوم بإيجاد عاملي لعدد ما بشكل عودي.

```
#include<iostream>
using namespace std;
int fact(int n)
  {
   If(n==0)     return 1;
   else
   return (n*fact(n-1));
   }
void main()
{
int num=0;
cin>>num;
cout<<num<<“!=<<fact(num)<<endl;
}
```

مثال على دّالة استدعاء الدّالة لنفسها Recursive Function

مثال: بناء الدّالة (X^n) أي دّالة (pow) باستخدام أسلوب Recursive Function

```
#include<iostream>
using namespace std;
int power(int x,int n )
 {
if (n>0)
return(x*power(x,n-1 ));
else
return 1;
  }
void main()
{
cout<<power(4,3);
}
```

64

الدّالة استدعت نفسها أربع مرات والنتيجة = ٦٤

الاستدعاء الأول

1.power(x=4,n=3)

2. n=3 is large than zero

3. return(4*power(4,3-1))

الاستدعاء الثاني

1.power(x=4,n=2)

2. n=2 is large than zero

3. return(4*4*power(4,2-1))

الاستدعاء الثالث

1.power(x=4,n=1)

2. n=1 is large than zero

3. return(4*4*4*power(4,1-1))

الاستدعاء الرابع

1.power(x=4,n=0)

2. n=0 is equal to zero

3. return(4*4*4*1)