اسم المادة : معالجة البيانات

تجمع طلبة كلية التكنولوجيا والعلوم التطبيقية - جامعة القدس المفتوحة

acadeclub.com

وُجد هذا الموقع لتسهيل تعلمنا نحن طلبة كلية التكنولوجيا والعلوم التطبيقية وغيرها من خلال توفيروتجميع كتب وملخصات وأسئلة سنوات سابقة للمواد الخاصة بالكلية, بالإضافة لمجموعات خاصة بتواصل الطلاب لكافة المواد:

للوصول للموقع مباشرة اضغط هنا

وفقكم الله في دراستكم وأعانكم عليها ولا تنسوا فلسطين من الدعاء♡

الوحدة الرابعة
جمل الدوران والمصفوفات

جمل التكرار

- تستخدم لتكرار جملة أو مجموعة من الجمل عددا من المرات
- نستخدم الحلقات التكرارية for – while – do..while
- كل واحدة منها تقوم بدور الأخرى ولا يوجد فرق إلا في الصيغة فقط

جمل الدوران FOR

- تستخدم هذه الجملة في جميع اللغات مثل باسكال وبيسك بنفس المعنى تقريبا لكن استخدامها في لغة سي أشمل وأعم.
- تستخدم بنفس معنى جملة while ونكون على علم بعدد تكرار الإجراء المطلوب.
- أي أن هذه الجملة تعمل على تكرار جملة أو مجموعة من الجمل عددا من المرات.
- الصيغة المستخدمة لها كما يلي:

```
for  ( exp1;  exp2;   exp3 )
     statement;
```

جمل الدوران FOR

- **التعبير الأول exp1** لإنشاء أو تأسيس بداية التكرار، أي لتحديد القيمة الأولى للمتغير المستخدم للتحكم في عدد التكرار (عداد) .
- **التعبير الثاني exp2** فهو يمثل شرطا يتطلب تحقيقه للبقاء داخل التكرار، أي ما دام هذا الشرط صحيحا (التعبير لا يساوي صفرا) فإن التكرار يستمر، أي أن هذا الشرط يحدد نهاية أو توقف التكرار (يمكن لأحد أن يفكر، ويقارن باللغات الأخرى، على أن هذا يحدد القيمة القصوى لمتغير التحكم).
- **التعبير الثالث exp3** يستخدم لبيان أو تحديد التغير الذي يتم على متغير التحكم، أي مدى تحديث قيمة المتغير (العداد) في كل دورة، مدي الزيادة
- **الجملة statement** فهو الجملة أو مجموعة من الجمل التي يتم تنفيذها في كل دورة من التكرار، وهذا يمثل جسم التكرار.

جمل الدوران FOR

ويمكننا معرفة استراتيجية عمل جملة for من خلال الخطوات الآتية:

1- عندما يأتي الدور على تنفيذ جملة for في البرنامج فإنه سيبدأ بـ expression1 حتى يرى القيمة الابتدائية لهذا التكرار، وتسند هذه القيمة للمتغير المطلوب.

2- بعد أن يحصل المترجم على نقطة البدء فإنه يتجه إلى expression2 الذي يقوم بالتحقق من قيمة هذا المتغير هل تحقق الشرط المطلوب أم لا؛ أي أن حلقة التكرار لن تكرر إذا كان ناتج التحقق خطأ.

3- بعد التحقق من الشرط يتجه البرنامج إلى تنفيذ الـ statement، ويجب ملاحظة أن expression3 لا بد من النظر إليها حالياً.

4- بعد تنفيذ statement يتجه البرنامج إلى expression3 حيث يتم الزيادة أو النقصان حسب البرنامج.

5- بعد تنفيذ expression3 يتجه البرنامج إلى expression2، وذلك حتى يتم التحقق هل القيمة الحالية لهذا المتغير تحقق الشرط، فإذا كانت صحيحة نفذت الـ statement ، وإذا كانت خطأ يخرج البرنامج من حلقة التكرار for.

جمل الدوران FOR

مثال (13):

```
#include<stdio.h>
void main( void )
{
int a;
for(a=1; a<=10; a++)
printf("hello world\n");
}
```

مثال

```
#include<stdio.h>
main()
{
int i,sum=0;
for(i=1;i<=4;i+=1)
sum +=i;
printf("sum=%d",sum);
}
```

i	sum	
0	0	
1	1	
2	3	
3	6	
4	10	
5	-	

البرنامج التالي يطبع القيمة: sum = 10

جملة التكرار بينما WHILE

- تستخدم هذه الجملة لتنفيذ إجراء **جمل عدة مرات (تكرار)** ما دام الشرط condition محققا.
- إذا أصبح هذا الشرط غير محقق فيتم الخروج من حلقة الدوران.
- أي ما دام الشرط محققا سيتم تنفيذ الأوامر التي بداخل التكرار.
- لاحظ بأن التحقق من الشرط يتم في بداية حلقة الدوران.
- صيغة هذه الجملة كما يلي:

```
while ( condition )
    statement;
```

مثال ١

```
#include<stdio.h>
void main( ) {
        int  x ;
        x =  0 ;
   while  ( x  <  4 )
        x  =  x + 1 ;
   printf (“x = %d  \n” ,  x );
}
```

البرنامج التالي يقوم بطباعة مضروب الرقم الذي يدخل المستخدم

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

void main(){
        clrscr();
        int x,f=1;
        printf("Enter number :");
        scanf("%d",&x);
        int k=x;
        while(x>0){
          f=f*x;
          x-=1;
          }
        printf("The %d factorial is:%d",k,f);
        getch();
        }
```

جملة الدوران DO & WHILE

- يشبه عمل هذه الجملة عمل جملة while ما عدا أن التحقق من الشرط يتم عند نهاية حلقة الدوران بدلا من بدايتها.
- ولهذا السبب فإن الجزء الواقع داخل حلقة الدوران يتم تنفيذه مرة واحدة على الأقل.
- تكتب هذه الجملة حسب التركيب التالي:

```
do{
    statement;
}while  (condition);
```

ومعناها do أي نفذ الجمل التالية وهى statement وما يليها طالما كان الشرط (condition) صحيح.

تابع

- غالبا ما تستخدم هذه الصيغة في فحص ما تم إدخاله من قبل المستخدم،
- فمثلا يمكن تنفيذ البرنامج مادامت المدخلات أرقاما موجبة فقط،
- أو أننا لم نصل إلى نهاية الملف وهكذا، كما هو مبين :

```
do
  {
     printf (“ Enter positive  no.”);
     scanf (“%d”, &x);
  }  while  (x <= 0 );
```

- هنا نطلب من المستخدم إدخال عدد موجب. نقرأ العدد، فإذا وجدناه سالبا، نطلب منه مرة أخرى وهكذا إلى حين إدخال الشيء المطلوب.

مضرب الرقم
بطريقة DO
WHILE

```
[■]═══ NONAME00.CPP ═══
#include<stdio.h>
#include<conio.h>

void main(){
        clrscr();
        int x,f=1;
        printf("Enter number :");
        scanf("%d",&x);
        int k=x;
        do{
          f=f*x;
          x-=1;
          }while(x>0);_
        printf("The %d factorial is:%d",k,f);
        getch();
        }
```

المصفوفات ARRAYS

المصفوفات Arrays

- نستخدم المصفوفات **لتجميع عدد من المتغيرات** لها **نفس النوع** تحت اسم واحد بدلا من إعطاء اسم خاص لكل منها.
- **أسباب استخدام المصفوفات بدل المتغيرات:**
- عندما يصبح عدد المتغيرات كبير جدا يصبح صعب التعامل مع هذا العدد.
- المتغيرات بحاجة لمخزن لتخزين كل متغير وهذا غير مجدي في العدد الكبير
- صعبة استخدام المتغير العادي كمصفوفة ذات بعدين أو ثلاثة

المصفوفات Arrays

- المصفوفة متغير جماعي يضم تحته عددا من العناصر يسمي المتغير الرقمي.
- يشبه المقسم الداخلي للهاتف في مؤسسة فكل موظف له رقمه الداخلي ويخزن في البدالة ويكون التعامل مع رقم رئيسي
- المصفوفة ما هي إلا تمثيل لجدول table يحوي القيم المطلوب وضعها في المصفوفة.
- يتم تعريف المصفوفة في لغة سي كمتغير له نوع ما متبوع بقوسين يحويان حجم المصفوفة مثل: [N] وهنا N تمثل حجم المصفوفة أي عدد الخانات التي تتكون منها المصفوفة.
- فمثلا يمكن تمثيل علامات 20 طالبا كما يلي:

- int grade[20];
- ومعنى ذلك أن يتم بناء مصفوفة أسمها grade مكونة من 20 خانة لتحوي نوع عدد صحيح.

تابع المصفوفات

- يمكن حجز خانة واحدة أو أكثر في الذاكرة كمتغير عن طريق تعريف المتغير في البرنامج.
- فمثلا : int cat يحجز مكان واحد في الذاكرة تحت اسم cat
- للوصول إلى بيانات المتغير نستخدم اسم المتغير cat.
- أما التعريف int cats[5]; **فيحجز 5 أماكن في الذاكرة تحت اسم cats.** جميع الأماكن هنا تحمل نفس الاسم cats.
- الشريحة التالية تبين أسماء المتغيرات والأماكن المحجوزة لها في الذاكرة:

int b;

ترتيب الخانات	1	2	3	4	5
الترتيب النسبي	0	1	2	3	4

الشكل (8): اسماء المتغيرات والأماكن المحجوزة لها في الذاكرة

تابع المصفوفات

- للوصول إلى أي من الخانات في المتغير الثاني يجب أن نستخدم اسم المتغير مع مؤشر أو فهرس يدل على إحدى هذه الخانات المنوي الوصول إليها واستخدامها.
- ولتحديد خانة معينة من المصفوفة يتم استخدام الصيغة : cats[i] على أن تمثل i الترتيب النسبي للخانة ابتداء من الصفر أي قيمة الفهرس index.
- الجدول الآتي يبين استخدام المصفوفة cats [i]

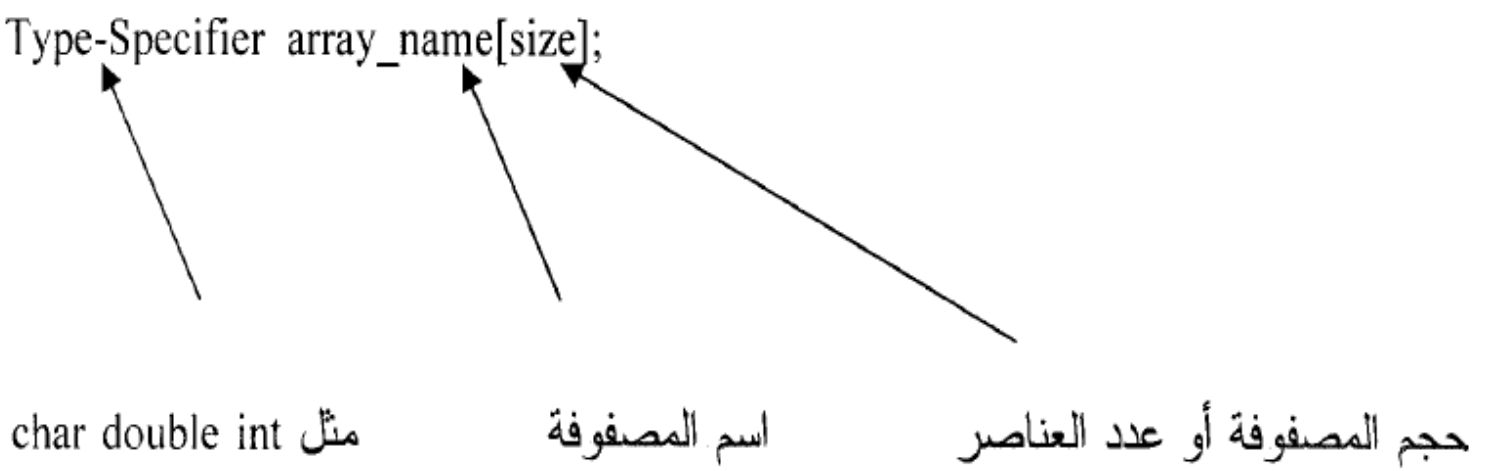
- cats[0] الخانة الأولى
- cats[1] الخانة الثانية
- cats[2] الخانة الثالثة
- cast[3] الخانة الرابعة
- cats[4] الخانة الخامسة

المصفوفات ذات البعد الواحد

- مجموعة العناصر التي تكون بشكل متجه Vector تمثل مصفوفة ذات بعد واحد .
- وتمثل رياضياً على الشكل التالي [a1 a2 a3.....an]
- وممكن أن تكون بشكل عمودي

$$\begin{pmatrix} A1 \\ A2 \\ \dots \\ \dots \\ An \end{pmatrix}$$

- شكل المصفوفة في لغة سي



المصفوفات ذات البعد الواحد

- تعريف المصفوفة البعد الواحد int a[20];
- مصفوفة من عشرين عنصر من الأعداد الصحيحة اسمها a

char name[15];

مصفوفة رمزية، اسمها name ، يحجز لها خمسة عشر عنصراً من النوع الرمزي لها، وهكذا ... .

- يمكن تأسيس قيمة أي متغير في أثناء تعريفه، وكذلك يمكن تأسيس قيمة المصفوفة عند تعريفها، ويتم ذلك كما هو مبين تاليا:

- int A [4] = { 40, 13, 20, 6 } ;

المصفوفات ذات البعد الواحد

- إذا كان عدد القيم المذكورة في تعريف المصفوفة أقل من طول المصفوفة، فإنه يتم تعبئة القيم الناقصة بالقيمة null أي \٠ .
- لا يجوز أن يتم تعريف قيم أكثر من عدد خانات المصفوفة.
- مثال:
- char name[9] = {‘A’, ‘B’,'C’, ‘D’ }
- يعتبر صحيحاً بينما
- char name[٣] = {‘A’, ‘B’,'C’, ‘D’ }
- لا يعتبر صحيحاً
- في بعض الحالات التي يتم فيها ذكر جميع قيم المصفوفة يمكن أن لا نذكر طول المصفوفة كما في الجملة التالية:
- int M[ ] = { 6, 0, 1, 4, 2 } ;

مثال

- أكتب برنامج يعمل على تعريف المصفوفة a[20] من عشرين عنصر مع إسناد القيم من 1– 20؟

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

main ()
{
int a[20];
int I;
for (I=0;I<20;++I)
a[I]=I+1;
return 0;
}
```

مثال

- أكتب برنامج باستخدام مصفوفة ذات بعد واحد من الأعداد الصحيحة ، ثم يطبع معدل الأعداد في المصفوفة؟

```
#include<stdio.h>
void main() {
        int a[4];
        int I, sum=0;
        float avg;
    for(i=0;i<4;i++)
    {
    scanf("%d",&a[i]);
    sum = sum + a[i];
    }
avg = sum/4;
printf("\n the average is  =%f",avg);
}
```

مثال (18):

```
#include<stdio.h>
#include<stdlib.h>
main ()
{
int x[5], y[5];
int I;
for (I=0;I<5;++I)
{
x[I]=I;
y[I]=I*I;
printf("%d  %d\n", x[I], y[I]);
}
return 0;
}
```

وستكون قيم النتائج على النحو الآتي:

0	0
1	1
2	4
3	9
4	16

مثال (20):

اكتب برنامجاً يقوم بإيجاد مجموع ومعدل علامات الطالب في 5 مواد، وهذه العلامات كالآتي: 55، 90، 81، 67، 87

```
#include<stdio.h>
#include<stdlib.h>
int i;
main ()
{
int a[5]={87,67,81,90,55};
int s=0;
for(i=0;i<5;i++)
s=s+a[i];
float avg=s/5; //قيمة المعدل لجميع العلامات
printf("%f \n  %d \n", avg, s);
return 0;
}
```

المعدل 87 والمجموع 735.

المصفوفات ذات البعدين

- جميع المصفوفات السابقة هي ذات بعد واحد.
- لاحظ أننا نستخدم لهذا النوع فهرسا واحدا. بما أن المصفوفة ما هي إلا جدول، فيمكن أن يكون الجدول ذو بعدين أو أكثر.(صفوف وأعمدة)
- وفي هذه الحالة نحتاج أن نستخدم فهرس لكل بعد.
- وكل بعد يمثل بقوسين من النوع [ ].
- صيغة مصفوفة ذات بعدين هي ;[m][n] type name
- حيث أن:
- name يمثل اسم المصفوفة
- type نوع البيانات المخزونة بها.
- n يمثل البعد الأول - الصفوف
- m يمثل البعد الثاني - الأعمدة

المصفوفات ذات البعدين

- int no [2][3];
  - تعرف no مصفوفة ذات بعدين من نوع int للتعامل مع هذه المصفوفة نحتاج إلى فهرسين واحد يحدد الخانة من البعد الأول والثاني يحدد الخانة من البعد الثاني.
  - لتحديد مثلا الخانة الموجودة [0] من البعد الأول و[2] من البعد الثاني : ;[2][0]no
  - ويمكن تصور البعد الأول ليمثل عدد الصفوف والبعد الثاني يمثل الأعمدة في الجدول.
  - نذكر بأن المصفوفة تحتوي على 2*3=6 خانات تأخذ صف صف ويتم الوصول إلى:
  - عنوان الخانة الأولي في المصفوفة تكون [0][0]no

س: ما هي مخرجات البرنامج التالي:

```
#include<stdio.h>
void main( ) {
    int no[2][3] = { 32, 27, 64, 18, 95, 14  };
    int k,j ;
      for ( k = 0; k < 2; k++ )
       {
         for( j=0;j<3;j++) {
            printf( "%d   ", no[k][j] );
       }
       printf("\n");
  }
```

أي سؤال أو استفسار

الوحدة الرابعة
الدوال الصديقة والعمليات
Friend Functions and Operators

يوسف ابوزر

المواضيع

- طرق تمرير العوامل للدوال Parameter Passing
- الدوال والأصناف الصديقة Friend Functions and Classes
- العمليات Operators
- صنف المجموعات Class Set

طرق تمرير العوامل للدوال Parameter Passing

1. **تمرير العوامل بالقيمة** يتم تمرير العوامل بالقيمة الى الدوال وأي تغيير على القيم داخل الدالة **لا يغير القيم الحقيقية للعوامل**
2. **تمرير العوامل بالإشارة** يتم تمرير العوامل برقم المرجع الي الدوال وأي تغيير على القيم داخل الدالة **يغير على قيم للعوامل**

تمرير العوامل بالقيمة

```
void swap1(int X, int Y)
        { int T=X;
              X=Y;
              Y=T;
        }

int main()
        {int A=10;
         int B=20;
         cout<<"A="<<A<<"  B="<<B<<endl;
         swap1(A,B);
         cout<<"A="<<A<<" B="<<B<<endl; return 0;
         }
```

العوامل الرسمية
Formal Arguments

العوامل الحقيقية
actual arguments

Memory		
Reference	Variable Name	Value
1214	a	10
1215	b	20
1216	x	10
1217	y	20
1218	z	10
.....		

A=10 B=20
A=10 B=20

تمرير العوامل بالإشارة

```
void swap2(int* X,int* Y)
        { int T=*X;
        *X=*Y;
        *Y=T;
        }
int main()
        {int A=10;
        int B=20;
        cout<<"A="<<A<<"  B="<<B<<endl;
        swap2(&A,&B);
        cout<<"A="<<A<<" B="<<B<<endl; return 0;
        }
```

A=10 B=20
A=20 B=10

parameter passing by reference تمرير العوامل بالإشارة

```
void swap3(int &X, int &Y)
  { int T=X;
       X=Y;
      Y=X;
}
```

Memory		
Reference	Variable Name	Value
1214	a	10
1215	b	20
1216	x	20
1217	y	10
1218	z	10
.....		

```
int main()
{   int A=10;
    int B=20;
    cout<<"A="<<A<<"  B="<<B<<endl;
    swap3(A,B);
    cout<<"A="<<A<<" B="<<B<<endl;
Return 0;
}
```

A=10 B=20
A=20 B=10

طرق تمرير العوامل للدوال Parameter Passing

تمرير العوامل بالقيمة parameter passing by value.	تمرير العوامل بالإشارة parameter passing by reference.	
void swap1(int X, int Y) { int T=X; X=Y; Y=T; }	void swap2(int* X,int* Y) { int T=*X; *X=*Y; *Y=T; }	void swap3(int &X, int &Y) { int T=X; X=Y; Y=X; }

main()
{int A=10;
int B=20;
 	
cout<<"A="<<A<<"
B="<<B<<endl;
swap1(A,B);
 	
cout<<"A="<<A<<"
B="<<B<<endl;
}
	main()
	{int A=10;
	int B=20;
	
	cout<<"
	A="<<A<<"
	B="<<B<<endl;
	swap2(&A,&B);
	
	cout<<"A=
	"<<A<<"
	B="<<B<<endl;
	}
		main()
		{int A=10;
		int B=20;
		
		cout<<"A="<<A<<"
		B="<<B<<endl;
		swap3(A,B);
		
		cout<<"A="<<A<<"
		B="<<B<<endl;
		}

A=10 B=20 A=10 B=20	A=10 B=20 A=20 B=10	A=10 B=20 A=20 B=10

مثال

```
#include <iostream>
using namespace std;
void Increment(int& Number){
 Number = Number + 1;
 cout << "The parameter Number: " << Number << endl;
}

int main(){
 int I = 10;
 Increment(I);    // parameter is a variable
 cout << "The variable I is: " << I << endl; return 0;
}
```

pass by reference and pass by value parameters in the same function

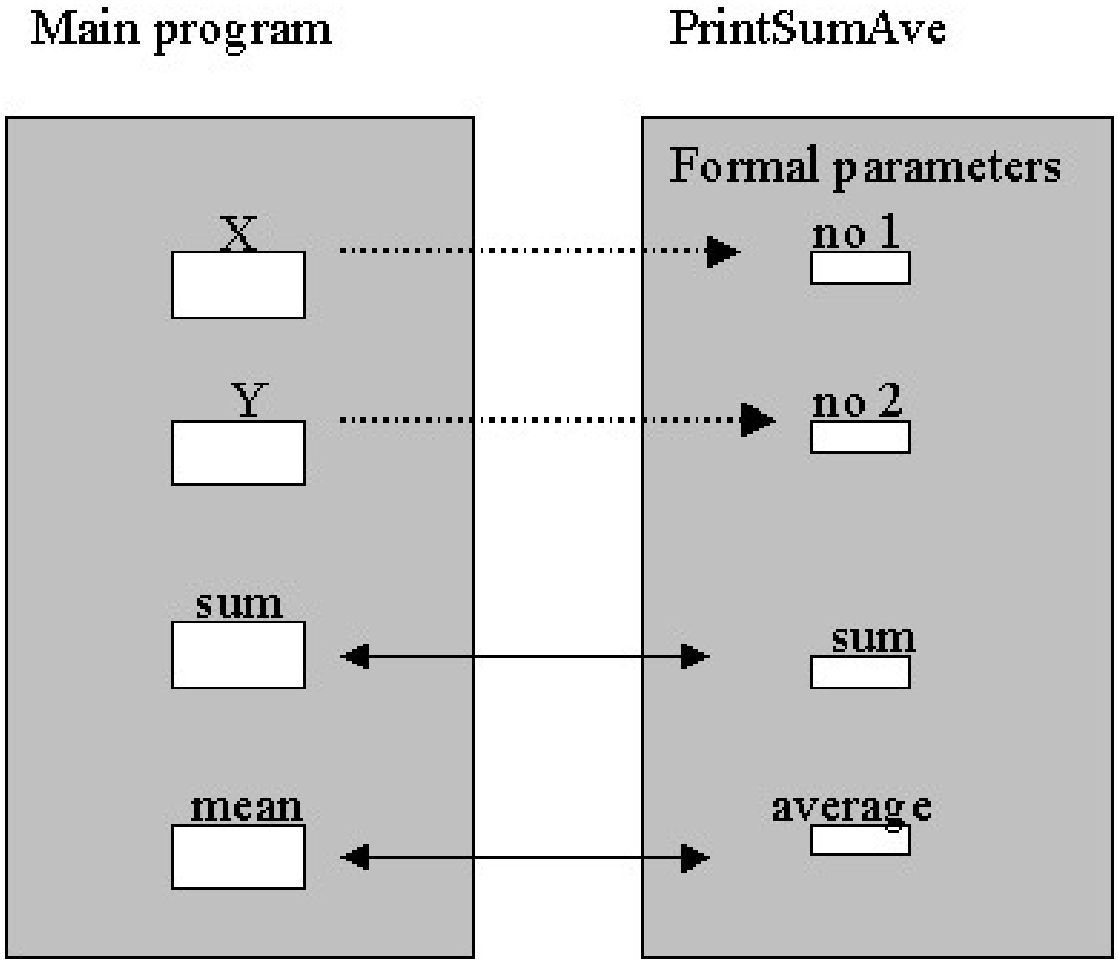
```
// Print the sum and average of two numbers
// Input: two numbers x & y
// Output: sum - the sum of x & y
//         average - the average of x & y
#include <iostream>
using namespace std;
void SumAve(double, double, double&, double&);
```

pass by reference and pass by value parameters in the same function

```
#include <iostream>
using namespace std;
void SumAve(double no1, double no2, double& sum, double& average) {
        sum = no1 + no2;
        average = sum / 2;
}
int main ( ) {
        double x, y, sum, mean;
        cout << "Enter two numbers: ";
        cin >> x >> y;
        SumAve (x, y, sum, mean);
        cout << "The sum is " << sum << endl;
        cout << "The average is " << mean << endl; return 0;
}
```

```
Enter two numbers: 4
5
The sum is 9
The average is 4.5
```

Pass by Reference: Example



Data areas after call to SumAve:

مثال

```
// Compare and sort three integers
#include <iostream>
using namespace std;
void swap (int&, int&);
int main ( ) {
        int first, second, third;  // input integers
        // Read in first, second and third.
        cout << "Enter three integers: ";
        cin >> first >> second >> third;
      if (first > second) swap (first, second);
      if (second > third) swap (second, third);
      if (first > second) swap (first, second);
      cout << "The sorted integers are " << first <<
          " , " << second << " , " << third << endl;
return 0;}
```

مثال

```
void swap3(int &X, int &Y)
  { int T=X;
        X=Y;
       Y=X;
}
```

```
// Compare and sort three integers
#include <iostream>
using namespace std;
void swap (int&, int&);
int main ( ) {
        int first, second, third;  // input integers
        // Read in first, second and third.
        cout << "Enter three integers: ";
        cin >> first >> second >> third;
      if (first > second) swap(first, second);
      if (second > third) swap(second, third);
      if (first > second) swap(first, second);
      cout << "The sorted integers are " << first <<
     " , " << second << " , " << third << endl;
return 0;}
```

```
Enter three integers: 7
5
9
The sorted integers are 5 , 7 , 9
```

مثال

```
// Pass-by-reference verses pass-by-value example
#include <iostream>
using namespace std;
void One(int a, int b, int& c) {
    int d;
    a = 10; b = 11; c = 12; d = 13;
    cout << "The values of a, b, c, and d in One: ";
    cout << a << " " << b << " " << c << " " << d << endl;
}
void Two(int a, int b, int& d) {
    int c = 0;
    cout << "The values of a, b, c, and d in Two: ";
    cout << a << " " << b << " " << c << " " << d << endl;
}
```

Pass by Reference: Example 4

```
int main () {
    int a = 1, b = 2, c = 3, d = 4;
    One(a, b, c);
    cout << "The values of a, b, c, and d after One: ";
    cout << a << " " << b << " " << c << " " << d << endl;
    Two(a, b, d);
    cout << "The values of a, b, c, and d after Two: ";
    cout << a << " " << b << " " << c << " " << d << endl;
return 0;}
```

```
The values of a, b, c, and d in One: 10 11 12 13
The values of a, b, c, and d after One: 1 2 12 4
The values of a, b, c, and d in Two: 1 2 0 4
The values of a, b, c, and d after Two: 1 2 12 4
```

Array Element Pass by Value

- يمكن تمرير عناصر المصفوفة الفردية حسب القيمة أو حسب الاشارة-Pass-by
- مثال القيمة :value example

```
#include <iostream>
using namespace std;
void printcard(int c) {
    if(c==1)
                cout << "A";
//……
     }
int main() {
   int cards[5]={1,2,3,4,5} ;
     for(int n=0; n<5; n++)
          printcard(cards[n]);
//……
     return 0;}
```

```
/tmp/CYxLPzN0X0.o
A
```

تمرير المصفوفات (عنصر)

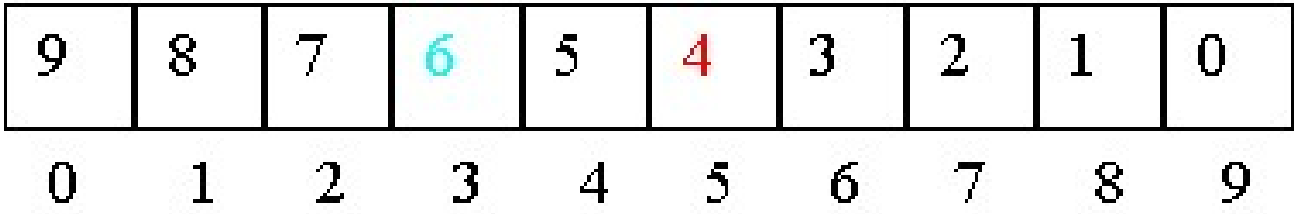
- Pass-by-reference example: مثال حسب الإشارة

```
#include <iostream>
using namespace std;
void swap(int& x, int& y) {
      int temp;
            if (x > y){
                  temp = x;
                  x = y;
                  y = temp;              }
}
int main() {
int A[10] = {9,8,7,6,5,4,3,2,1,0};
for(int n=0; n<9; n++)
      cout<<A[n]<< "  ";
swap(A[3], A[5]);
cout<<endl;
for(int n=0; n<9; n++)
      cout<<A[n]<< "  ";
return 0;          }
```

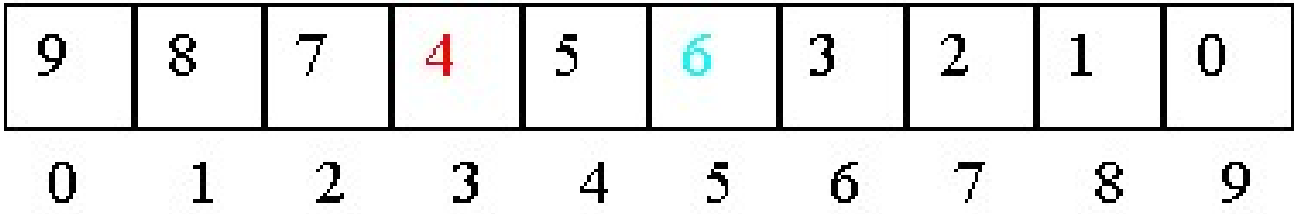
```
9  8  7  6  5  4  3  2  1
9  8  7  4  5  6  3  2  1
```

Array Element Pass by Reference

- قبل:



- بعد:



تمرير المصفوفات

- لا يمكن تمرير المصفوفة بالإشارة.
- يتم التمرير بواسطة عنوان أول عنصر وبعد المصفوفة.

```
#include <iostream>
using namespace std;
void f(int A[]) {
    A[0] = 5;}
//
int main() {
    int B[10];
    B[0] = 2;
    f(B);
    cout << B[0] << endl; return 0; // the output is 5
}
```

بالرغم من ان معاملات الاقتران f تظهر انها تمرر بالقيمة (لعدم وجود إشارة &) , وحيث ان المتغير هو مصفوفة فانة يتم تمريره بالإشارة.

لذا سيقوم البرنامج بطباعة قيمة 5 بدلا من 2.

```
/tmp/CYxLPzN0X0.o
5
```

Arrays to Functions: Example 1

```
//Find the largest value in an array
//input: n - number of elements to check
//    a[ ] - array of elements
// output:index to the largest element
#include <iostream>
using namespace std;
int max_element(int n, const int a[]) {
        int max_index = 0;
        for (int i=1; i<n; i++)
                if (a[i] > a[max_index])
                        max_index = i;
        return max_index;
}
int main() {
         int A[10] = {9,8,7,6,5,4,10,2,1,0};
        cout << A[max_element(10,A)] << endl; return 0;
}
```

```
/tmp/CYxLPzNOXO.o
10
```

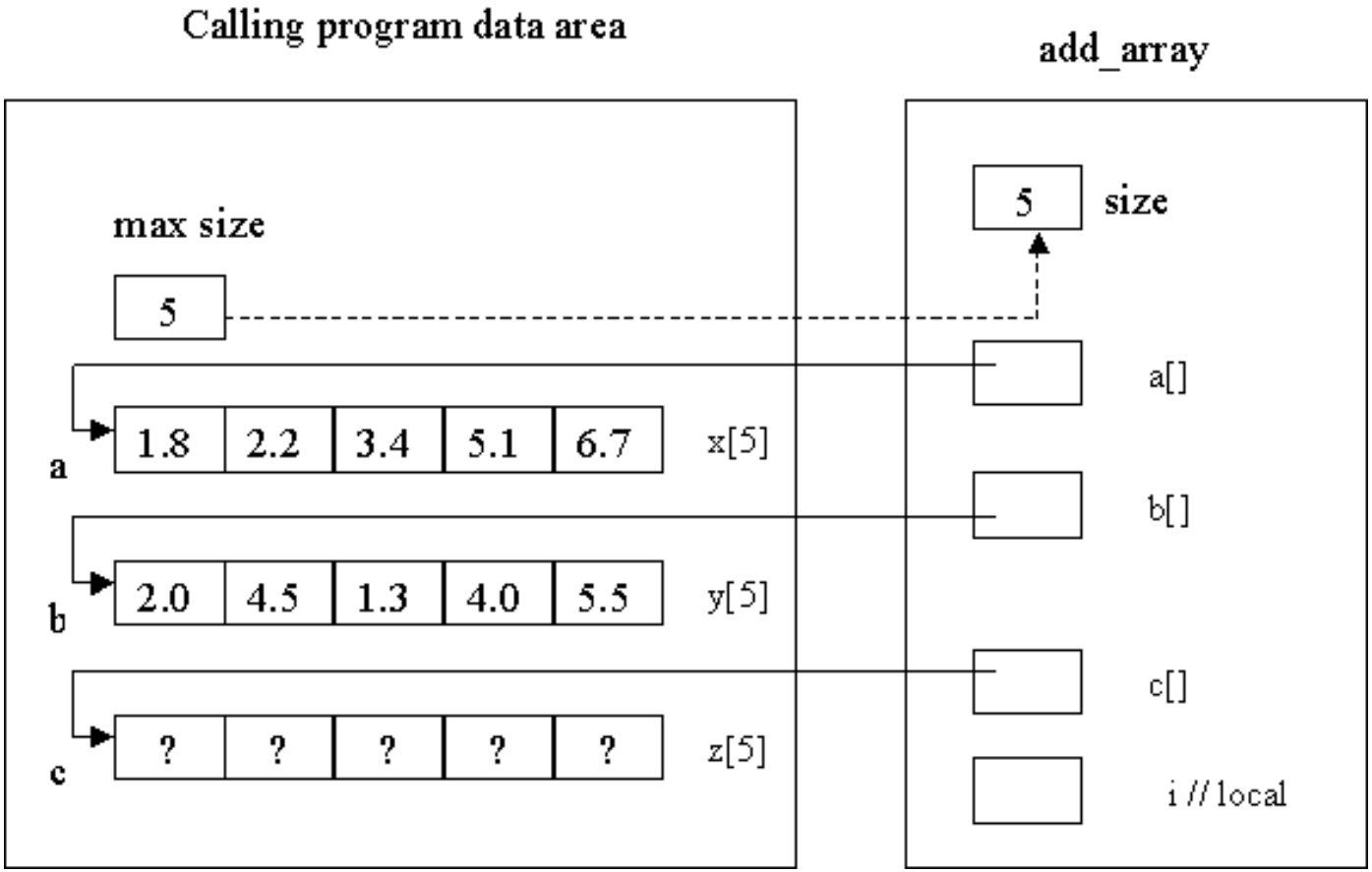
Arrays to Functions: Example 2

```
//Add a[i] and b[i] and store the sum in c[i]
//Array elements with subscripts ranging from
//0 to size-1 are added element by element
void  add_array(int size, const double a[],
                const double b[], double c[]){
    for (int i=0; i<size; i++)
        c[i] = a[i] + b[i];
}
```

In main():

```
    add_array (5, x, y, z );
```

Arrays to Functions: Example 2



Passing Multidimensional Arrays

- كيفية تمرير مصفوفة متعدد الأبعاد إلى دالة:

```
void displayBoard(int b[][4]);
// function prototype requires variable name for arrays

void main(){
     int board [4][4];
     ...
     displayBoard(board);
     ...
}
void displayBoard(int b[][4]){
// could also be:      void displayBoard(int b[4][4]){
// but NOT:      void displayBoard(int b[][]){
     ...
}
```

- عند تمرير مصفوفة متعدد الأبعاد ، يكون حجم البعد الأول فقط اختياريًا ، ويجب تحديد الأبعاد الثانية والثالثة وما إلى ذلك.

الدوال الصديقة والأصناف الصديقة Friend Functions and Classes

- تتيح الصداقة صلاحية الوصول للمتغيرات والدوال لخاصة Private
- تضاف كلمة Friend للدالة وللصنف لتصبح الدالة صديقة للصنف.
- يمكن أن تكون الدالة أو الصنف صديقة لأكثر من صنف.
- علاقة الصداقة ليست عكسية:
  - أي أنه إذا كان الصنف T1 صديق للصنف T2 فهذا لا يعني أن الصنف T2 صديق للصنف T1
- اذا كانت الدالة Addهي دالة منتمية فإنها تستدعى كالتالي Calc. Add
- اذا كانت الدالة Addهي دالة صديقة فإنها تستدعى كالتالي Add(C)
- يمكن عمل صداقة بين كل من:
  - صنف ودالة.
  - صنفيين معا.
  - صنف ودالة منتمية.

مثال

- افترض ان لديك الصنفين التاليين manager, employee
- اكتب الجمل اللازمة ل:
- تعريف دالة max كدالة صديقة من نوع صحيح للصنف employee
-
- الاجابة:

```
friend int max (employee x);
```

مثال

- اكتب برنامجا بلغة C++ تعرف فيه صنفا اسمه rectangle بحيث يحتوي على متغير منتمي
  - اسمه l
  - و متغير منتمي اسمه w
- ودالة بناء لإسناد قيم للمتغيرات l=10 و w=4
- دالة صديقة اسمها area لحساب مساحة المستطيل.
- ثم اكتب الدالة الرئيسية لإنشاء كائن اسمه rec
- وقم بطباعة مساحة ذلك المستطيل

```
#include<iostream>
using namespace std;
class rectangle{
int  l,w;
public:
rectangle () {l =10;w=4;}
friend int area(rectangle x){return x.l * x.w;}
};

int main ()
{
rectangle rec;
cout <<"the area is "<<area(rec); return 0;
}
```

the area is 40

لاحظ مايلى بالنسبة للدالة الصديقة:

-هي مسبوقة بالكلمة friend

-- معاملاتها وهنا معامل واحد فقط هو x كائن من نفس الصنف التي هي صديقة له اي صديقة للصنف rectangle

-- كيف تم التعامل مع المتغيرات المنتمية في الدالة الصديقة؟

-لاحظ ان التعامل معهم ليس مثل التعامل مع الدوال المنتمية

-حيث تم التعامل مع المتغير l من خلال الكائن x

-وكذلك التعامل مع المتغير المنتمي w من خلال الكائن x كالتالي: x.l * x.w

بينما لو كانت دالة منتمية سيكون التعامل مع تلك المتغيرات المنتمية كالتالي : l*w

اما بالنسبة لاستدعاء الدالة الصديقة فتم مثل استدعاء دالة عادية وليست منتمية كالتالي : rea(rec)

بينما لوكانت الدالة area دالة منتمية سيكون استدعاؤها كما هو مر سابقا من خلال الكائن الذي يريد ان يستدعيها اي بالشكل التالي: rec.area();

صداقة صنفيين

- لا حظ الصنفيين المقابلين **T1 , T2**
- تم عمل صداقة للصنف الأول مع الصنف الثاني. أي أن الصنف T2 صديق للصنف T1
- علاقة الصداقة ليست عكسية. أي أن الصنف **T1** ليس صديق للصنف **T2**
- لاحظ أن الصداقة أتاحت للصنف T2 الوصول للمتغير **x** الخاص للصنف **T1** كما هو موضح في الدالة print()
- بعد عمل الصداقة يتم التعامل مع جميع دوال ومتغيرات الصنف الصديق كدوال ومتغيرات منتمية.

```
Class T2;
Class T1
{
Private: int x;
Public : int y;
friend T2
};
Class T2
{
public: void print() { cout <<x};
```

الأصناف الصديقة

- ان علاقة الأصناف الصديقة ليست تبادلية ولا علاقة تعديه .
- أي أن كون الصنف العرض Display صديق للصنف المخزن Storage ، لا يعني ان الصنف المخزن Storage أيضا صديق للصنف العرض Display.
- فإذا أردت ان تجعل صنفين أصدقاء لبعضهم البعض، فيجب على الاثنين أن يعلنان على ان الصنف الأخر صديق. أخيرا،
- إذا كان الصنف A صديق B ، و B صديق C ، هذا لا يعني ان الصنف A صديق للصنف C .

مثال

```
#include<iostream>
using namespace std;
class Storage
{
private:
    int m_nValue;
    double m_dValue;
public:
    Storage(int nValue, double dValue)
    {
        m_nValue = nValue;
        m_dValue = dValue;
    } // Make the Display class a friend of Storage
    friend class Display;
};
```

مثال

```
class Display
{ private:
   bool m_bDisplayIntFirst;
public:
   Display(bool bDisplayIntFirst) { m_bDisplayIntFirst = bDisplayIntFirst; }
   void DisplayItem(Storage &cStorage)
   {       if (m_bDisplayIntFirst)
           cout << cStorage.m_nValue << " " << cStorage.m_dValue << endl;
       else // display double first
           cout << cStorage.m_dValue << " " << cStorage.m_nValue << endl;
   }
};
```

مثال

```
int main()
{
   Storage cStorage(5, 6.7);
   Display cDisplay(false);

   cDisplay.DisplayItem(cStorage);

   return 0;
}
```

```
Output
/tmp/CYxLPzN0X0.o
6.7 5
```

النتيجة التي نحصل عليها من بعد تنفيذ البرنامج هي كما يلي: 5 6.7

الدوال الصديقة والأصناف الصديقة

- **استخدام الدوال المنتمية أفضل من الصديقة إلا في الحالات التالية:**

1. **إذا كانت الدالة تحتاج الى عاملين أو أكثر. وانتمى كل واحد منهما الى صنف مختلف.**
2. **إذا أردنا تحميل الدالة أكثر من تعريف من خلال عدد العوامل ونوعهم.**
3. **إذا كانت الدالة تستخدم متغيرات منتمية لعملها.**

العمليات Operators

- إعادة التعريف Overloading إعادة تعريف العمليات لتعمل مع الكينونات المشتقة من الأصناف المختلفة.
- مثال- نستطيع تعريف العملية + كعملية منتمية للصنف section لتعني إضافة طالب معين إلى الشعبة والعملية – لتعني شطب طالب معين من الشعبة .
- يقوم مترجم C++ بتحديد التعريف المطلوب من عدد و نوع العوامل.
- اسم العملية يتكون من الكلمة operator متبوعا برمز العملية على سبيل المثال operator+

```
void operator+(student s);
```

- لاستدعاء العملية فيمكننا استدعاءها كأي دالة منتمية، فمثلا لإضافة الطالبS للشعبة cs100 يمكننا أن نستدعي العملية + كما يلي :

```
cs100.operator+(S);
```

- **أما الطريقة الأفضل لاستدعاء العملية فهي أن نستخدمها كأي عملية جمع أخرى كما يلي:**

```
cs100 + S;
```

Operators العمليات

```
class section{
  student sec[50];
  int size;
  int bsearch(long stno);
public:
  section(){size=0;}
  void operator+(student s);
  void StAdd(student s);
  int StDelete(long stno);
  void Stlist();
  void StRetrieve(long stno);
};
```

```
void section::operator+(student s)
  int pos=0;
// search for the proper insertion position
  while(s.get_stno() > sec[pos].get_stno() && pos<size)
     pos++;
// shift element up one position
 for(int i=size-1;i>=pos;i--)
    sec[i+1]=sec[i];
 sec[pos]=s;
 size++;
}
```

cs100.operator+(S);
cs100 + S;

Operators العمليات

```
class student{
        long stno;
        int csno;
        double grades[100];
        char StName[20];
    public:
 friend double average(student s);
         void initialize();
         long get_stno() {return  stno;}
         char* get_stname(){return StName;}
 };
```

```
class section{
  student sec[50];
  int size;
  int bsearch(long stno);
public:
  section(){size=0;}
  void operator+(student s);
  void StAdd(student s);
  int StDelete(long stno);
  void Stlist();
  void StRetrieve(long stno);
};
```

cs100.operator+(S);
cs100 + S;

العمليات Operators

- العمليات التي يمنع إعادة التحميل لها:

: :	عملية تقرير المجال
Sizeof	عملية Sizeof
.*	اختيار الأعضاء مباشرة
.	عملية النقطة
? :	الخاصة بالتعابير الشرطية

- العمليات المتاحة لإعادة التحميل:

العمليات المسموح اعادة تعريفها					
+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

العمليات Operators

- **لا يمكن تغيير الخصائص التالية للعمليات عند إعادة تعريفها:**

1. **عدد العوامل.**
2. **الأسبقية (الأولوية).**
3. **رتيب التنفيذ:**
   - **إذا كانت من اليمين الى اليسار فلا يجوز تغييرها للعكس.**

this

- **عند استدعاء الدوال المنتمية فإن هناك مؤشر يمرر ضمنيا يدعى** this
- **الدوال الصديقة لا تملك هذا المؤشر لأنها ليست منتمية.**
- **يستخدم عادة لتمييز بين الدوال والمتغيرات المنتمية أو الصديقة أو الممررة.**

```
class Box{
public:
Box(double l =2.0 , double b = 2.0 , double h = 2.0 )
cout<<"Constructor called." << endl
length = l;
breadth = b;
height = h; }
double Volume() { return length * breadth * height; }
int compare(Box box) {return this-->Volume() > box.Volume ();}
private:
double length; double breadth; double height;
};
void main(void)
Box Box1(3.3 , 1.2 , 1.5 ); // Declare box 1
Box Box2(8.5 , 6.0 , 2.0 ); // Declare box 2
if(Box1.compare(Box2 ))
Cout << "Box 2 is smaller than Box 1 " << endl
else
cout << "Box 2 is equal to or larger than Box 1 " << endl
}
```

```
class Box {
  public:
    Box(double l = 2.0, double b = 2.0, double h = 2.0) {
      cout <<"Constructor called." << endl;
      length = l;
      breadth = b;
      height = h; }

    double Volume() {    return length * breadth * height;    }

    int compare(Box box) {
      return this->Volume() > box.Volume(); }

  private:
    double length;    double breadth;    double height;
};

void main(void) {
  Box Box1(3.3, 1.2, 1.5);   // Declare box1
  Box Box2(8.5, 6.0, 2.0);   // Declare box2

    if(Box1.compare(Box2))
        cout << "Box2 is smaller than Box1" <<endl;
    else
        cout << "Box2 is equal to or larger than Box1" <<endl;
}
```

العناصر الثابتة Static

- نستخدم الدوال الثابتة لضمان عدم تعديل هذه الدوال لاي قيمة لاي متغير منتمي.
- تستخدم الكلمة const لتعريف الدالة الثابتة. وتوضع بعد تعريف الدالة.

```
Class T1
{
Private: int x;
Public :void print() const {cout <<y};
};
```

مثال const

```
#include<iostream>
using namespace std;
//void aFunc(int& a, const int& b);  //declaration
//------------------------------------------------------------
void aFunc(int& a, const int& b)   //definition
  {
  a = 107;   //OK
  b = 111;   //error: can't modify constant argument
}
int main()
  {
  int alpha = 7;
  int beta = 11;
  aFunc(alpha, beta);
  return 0;
 }
```

Output

Clear

```
g++ /tmp/CYxLPzN0X0.cpp
/tmp/CYxLPzN0X0.cpp: In function 'void aFunc(int&, const int&)':
/tmp/CYxLPzN0X0.cpp:8:6: error: assignment of read-only reference 'b'
    8 |    b = 111;   //error: can't modify constant argument
```

- تستخدم الكلمة ألمفتاحيه const مع الدوال المنتمية في الأصناف اذا أردنا ضمان عدم تعديل تلك الدوال المنتمية لأي من عناصر البيانات المنتمية المعرفة في تلك الأصناف.

```
//demonstrates const member functions

class ConstFuncClass
  {
  private:
    int age;
  public:
    void normalFunc()       //non-const member function
      { age = 55; }    //OK

    void conFunc() const  //const member function
      { age = 55; }    //error: can't modify a member
  };
```

الوحدة الرابعة
الدوال الصديقة والعمليات
Friend Functions and Operators

يوسف ابوزر

```
#include<iostream.h>
class table;
class product{
int price;
char name[10];
public:
product(){price=20;}
friend class table;
};
class table{
int price;  char name[10];
public:
void set() {cout<<"enter the price";cin>>price;
cout<<"enter the name";cin>>name;}
void add(){price+2;}
void print(){cout<<“\n the price became \n"<<price;}
void add(product &x) {x.price=x.price+price;}
void print(product y)  {cout<<"the total price for both is "<<y.price;}};
```

```
void main(){
product pr;
table tp;
tp.set();
 tp.add();
tp.print();
tp.add(pr);
tp.print(pr);
```

الدوال الصديقة والأصناف الصديقة

- **صداقة صنف ودالة منتمية:**
- **يسبق اسم الدالة اسم الصنف متبوعا بالإشارة (::).**
- **صداقة صنف ودالة:**
- **يجب كتابة تعريف الدالة كاملا عند عمل الصداقة (القيمة المرجعة والعوامل).**
- **لاحظ الصداقة بين الصنف الثاني ودالة main()**

```
Class T1
{Private:int x;
Public :int y;
friend T2 ::print();
};
Class T2
{public:void print(){ cout <<x;}
friend int main();
};
```

الدالة ترجع قيمة بالإشارة

- نوع القيمة المرجعة هو int& أي إشارة reference لمتغير صحيح .
- تقوم الدالة max بتحديد العامل صاحب القيمة الأكبر وإعادة إشارة إليه

```
int& max(int A, int B)
{if (A>B)
return A;
else
return B;
}
```

```
int A=10;
int B=100;
max(A,B)=5;
```

نتيجة تنفيذ هذه الجملة هي تخزين القيمة 5 في المتغير صاحب أكبر قيمة وهو المتغير .