بسم الله الرحمن الرحيم

Object Oriented Programming

البرمجة الشيئية

عدد الساعات: ٢ نظري + ٢ عملي

المتطلبات السابقة: ٢١١حسب

أ/المادة: م. نجلاء حسن

Lecture 11

- مقدمة عن المؤشرات Pointers
- التعددية الشكلية Polymorphism
- الدوال والأصناف الصديقة friends functions and friend classes

العناوين و المؤشرات (Pointers)

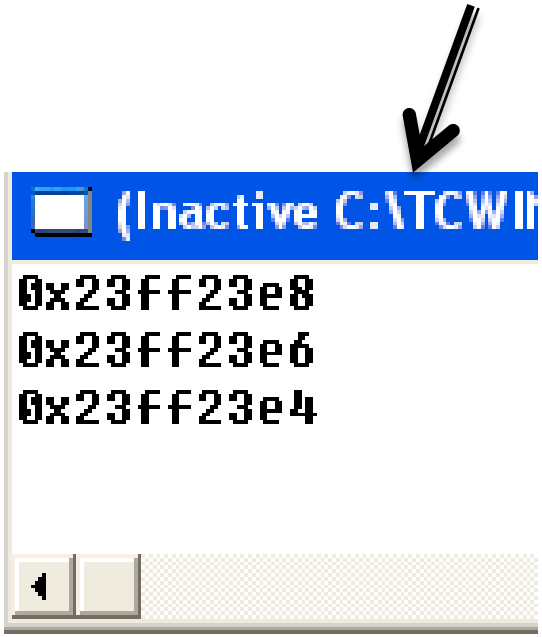
- كل بايت داخل ذاكرة الحاسب يكون له عنوان محدد address .
- يتم تمثل هذه العناوين بأرقام تبدأ من 0 وتنتهي برقم ما تبعاً لحجم الذاكرة المستخدمة.
- بمجرد تحميل البرنامج داخل الذاكرة ، يشغل هذا البرنامج مدى معين داخل اهذه الذاكرة، وهو ما يعني أن كل متغير وكل دالة داخل البرنامج تبدأ من عنوان معين.

استخدام المعامل &

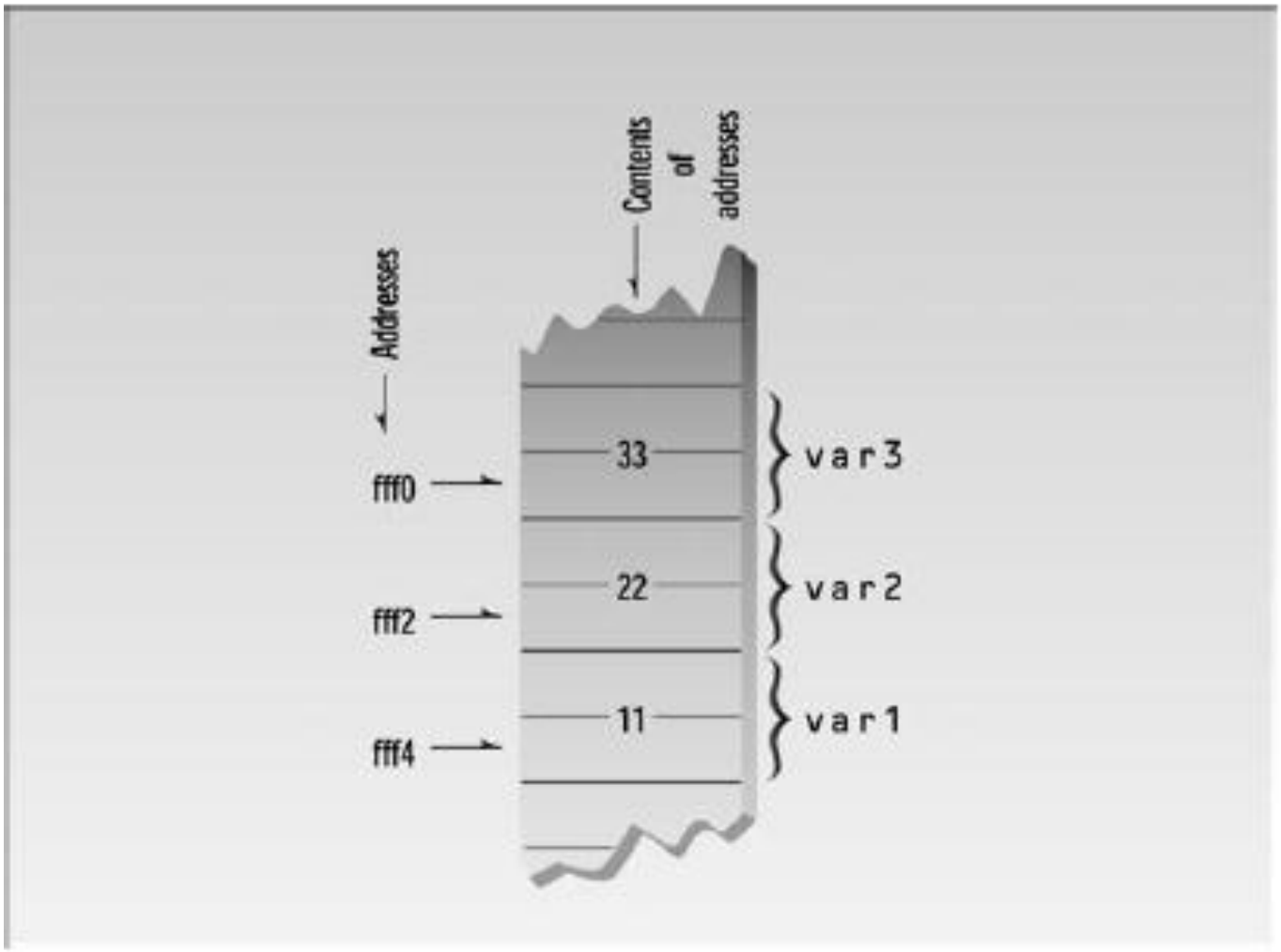
- البرنامج التالي يقوم بتعريف ثلاث متغيرات من النوع int مع تخصيصها بالقيم 11 و و3322 ثم طباعة عناوينها باستخدام المعامل &

```
#include<iostream.h>
int main()
{
int var1=11;
int var2=22;
int var3=33;
cout<<&var1<<endl
        <<&var2<<endl
         <<&var3<<endl;
return 0;
}
```

تختلف النتيجة من حاسب لآخر



لاحظي الفرق بين عنوان المتغير وقيمته (محتويات المتغير)



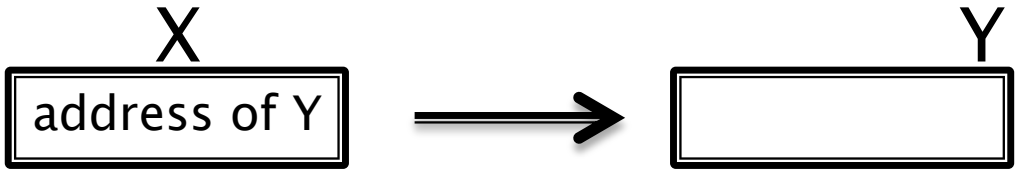
Addresses and contents of variables.

المؤشرات (Pointers)

- لا توجد فائدة من معرفة عنوان أحد المتغيرات أو مكانه داخل الذاكرة ، وانما الافضل تخزين هذه العناوين داخل متغيرات ومن ثم استخدام هذه المتغيرات في الوصول لتلك العناوين.
- يطلق على هذا النوع من المتغيرات متغيرات المؤشرات أو اختصارا المؤشرات (Pointers)

تعريف المؤشر :-

هو متغير يحتوي على عنوان لأحد عناصر البيانات ونستخدمه للإشارة إلى مكان تخزين عنصر البيانات في الذاكرة (يحمل عنوان مخزن آخر).



X تؤشر إلى Y

:

الإعلان عن المؤشرات:

- مثل بقية المتغيرات يجب التصريح عن المؤشرات قبل استخدامها، ويتم التصريح عن المؤشر بوضع إشارة * قبل اسم المؤشر.
- يمكن أن يتم التصريح عن المؤشر ليؤشر على قيمة أو متغير من أي نوع من أنواع البيانات الأساسية في لغة C++

الصيغة العامة للإعلان عن المؤشر

data type *pointer name ;

حيث :-

data type هي نوع البيانات التي يشير إليها المؤشر ، pointer name هو اسم المؤشر .

أمثلة :

1. int * p;

إعلان عن مؤشر يشير إلى متغير من النوع الصحيح

2. int i,j,a[10],b[2],*p,*q;

إعلان عن مجموعة من المتغيرات والمصفوفات والمؤشرات من النوع الصحيح

3. char *r;

إعلان عن مؤشر يشير إلى متغير من النوع الحرفي

4. float *m, *n

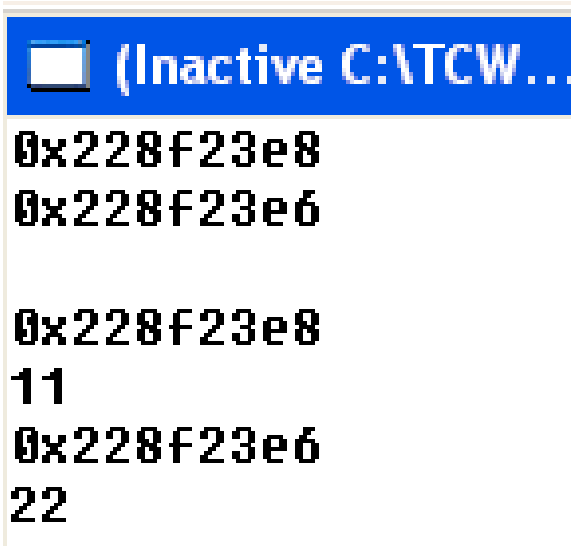
إعلان عن مؤشرين باسم m,n ليؤشرا على قيمتين حقيقيتين.

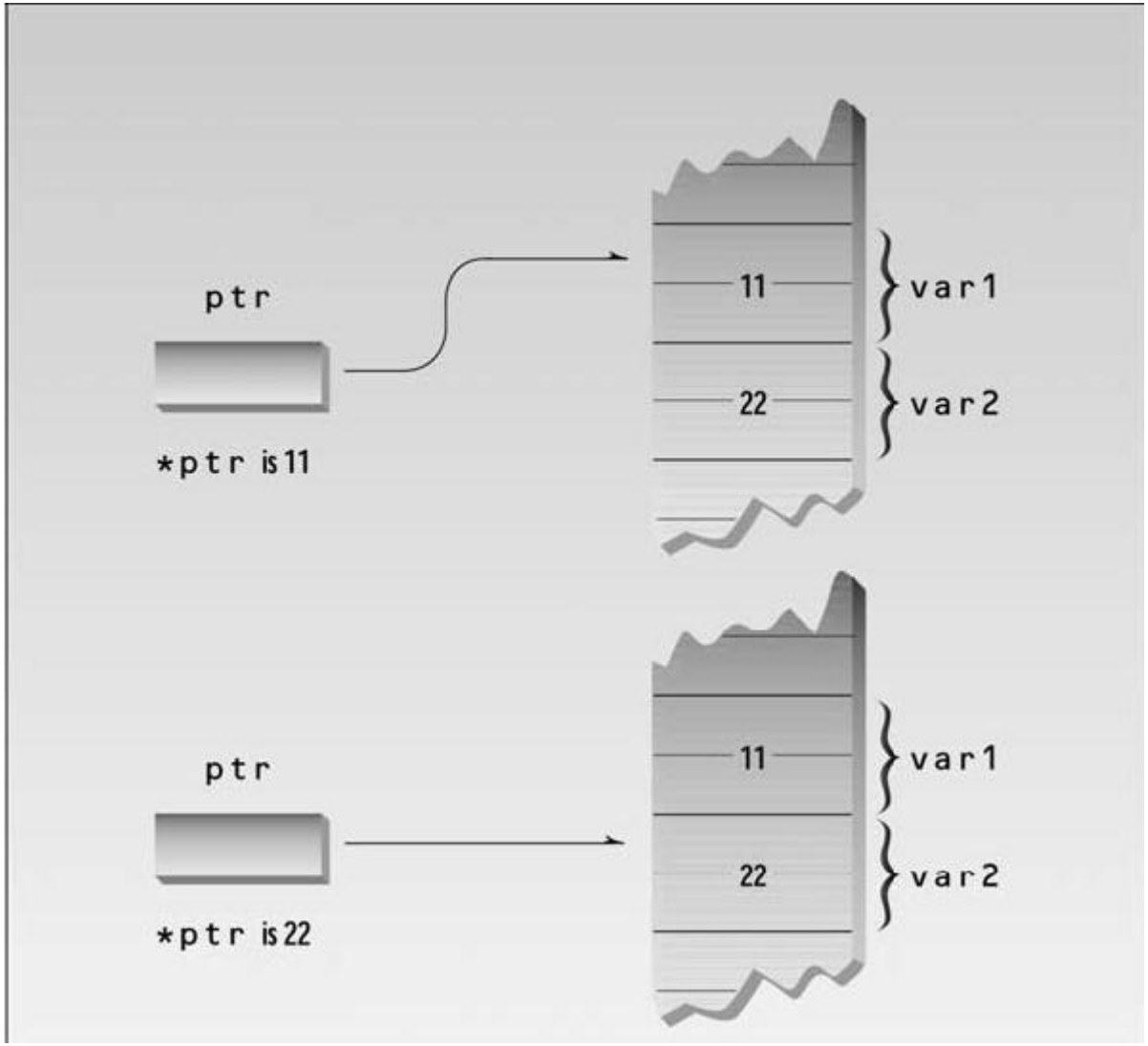
مثال: يوضح الاعلان عن المؤشرات بالاضافة الى الوصول الي محتويات المتغير بمعرفة عنوانه داخل الذاكرة (بفرض اننا لا نعرف اسمعه)

```
#include<iostream.h>
int main()
{
int var1=11;
int var2=22;
cout<<&var1<<endl
        <<&var2<<endl<<endl ;

int* ptr;
ptr=&var1;
cout<<ptr<<endl;
cout<<*ptr<<endl;


ptr=&var2;
cout<<ptr<<endl;
cout<<*ptr<<endl;
return 0;
}
```





Access via pointer.

عوامل المؤشر:-

١/ عامل العنوان &:-

العامل & يسمى عامل العنوان وهو عامل أحادى يستعمل لإعطاء عنوان متغير في الذاكرة الذي يحتله متغير ما فمثلاً إذا استعملنا الإعلان:

```
int y= 5;
int *yptr;
```

العبارة: yptr =&y;

تقوم بتعيين عنوان المتغير y للمؤشر yptr ويقال أن yptr يشير لـ y .

٢-/ العامل * :

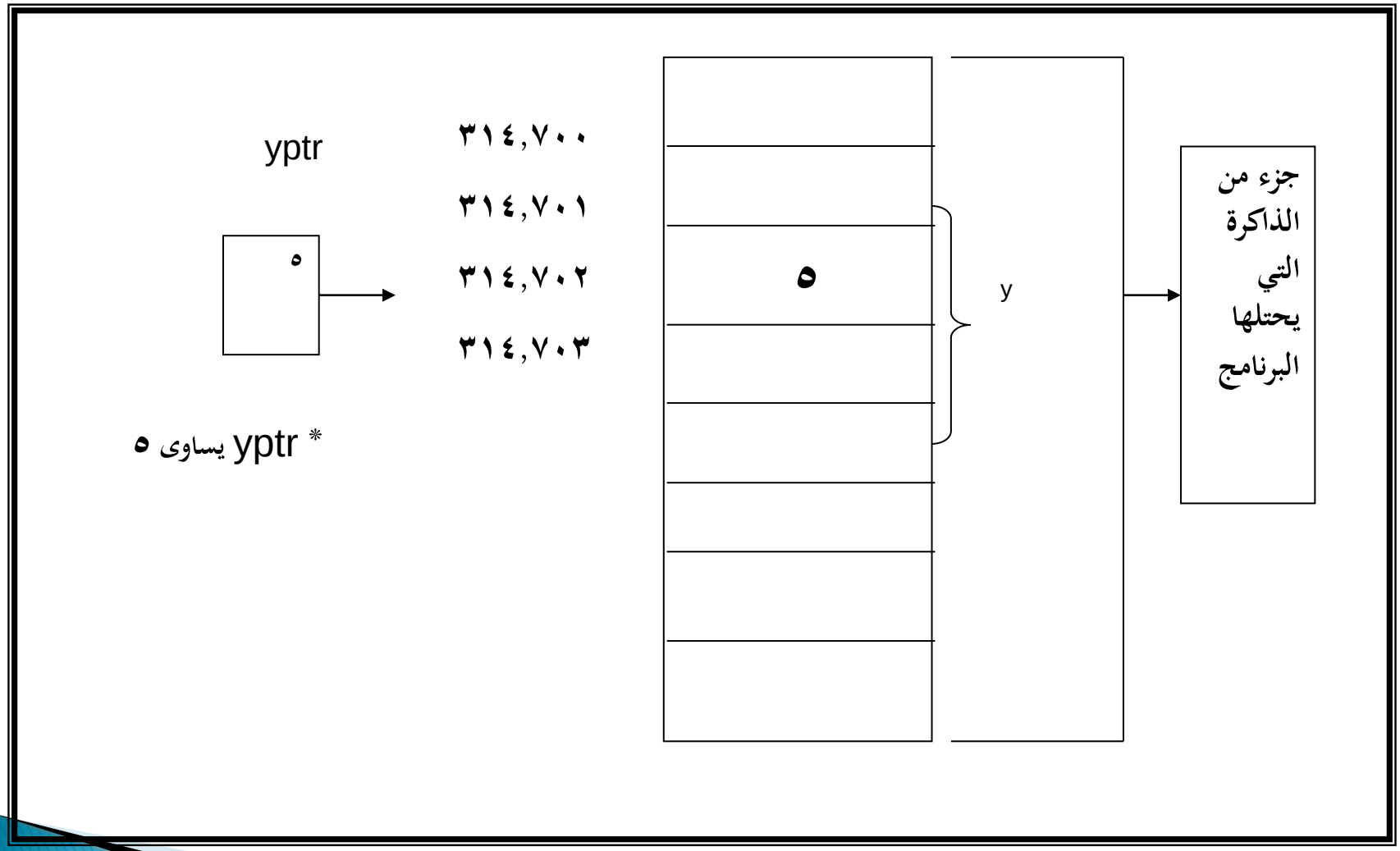
العامل * : يستخدم لاعطاء قيمة المتغير للمؤشر (المشار اليه) أي قيمة y في المثال السابق.

العبارة :; cout << * yptr << endl

تقوم بطباعة قيمة المتغير yوالتي هي 5 .(مرادف لـ cout<<y;)

والعبارة: cout<<yptr; تقوم بطباعة القيمة 314,701 والتي هي عنوان المتغير y ، بعد أن تم تعيين المتغير y إلى yptr .

(تأخذ الأداتان & و* أولوية في التنفيذ على سائر العمليات(المؤثرات) المختلفة)



شكل (١–٥) يوضح المخرج من **yptr***

أمثلة:

- الإعلان عن مؤشر iptr ليشير إلى متغير من النوع int:

```
int  *iptr;
```

- تخصيص عنوان المتغير value1 للمؤشر iptr:

```
iptr=&value1;
```

- طباعة القيمة التي يشير إليها المؤشر iptr:

```
cout<<*iptr;
```

- طباعة عنوان المتغير value1:

```
cout<<iptr
```

- استخدام المؤشر iptr لتخصيص القيمة 30للمتغير value1 :

```
*iptr =30;
```

محتوى أي مؤشر هو عدد من النوع صحيح بصرف النظر عن نوع البيانات التي يشير اليها المؤشر . فمحتوى المؤشر هو عنوان موقع بالذاكرة (عدد من النوع الصحيح)أي رقم بالبايت الذي يبدأ عنده تخزين نوع بيانات معين.

الا أن المؤشرات تختلف عن بعضها البعض حسب نوع البيانات التي تشير اليها .

المؤشرات والمصفوفات

```
int main()
   {
   int intarray[] = { 31, 54, 77, 52, 93 }; //array
   int* ptrint;                             //pointer to int
   ptrint = intarray;                       //points to intarray

   for(int j=0; j<5; j++)                   //for each element,
      cout << *(ptrint++) << endl;          //print value
   return 0;
   }
```

العمليات الحسابية على المؤشرات:

تستطيع المؤشرات الإشارة إلى المتغيرات المفردة كما وتشير إلى عناصر المصفوفة array

لنفترض أن a,p عرفت كما يلي:

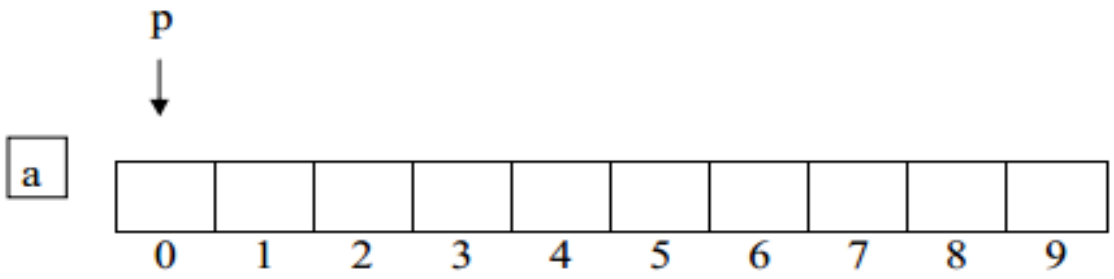
```
int a[10],*p;
```

فبهذا يمكن استخدام المؤشر p للإشارة إلى العنصر الأول من المصفوفة a (a[0])

كما يلي:

p=&a[0]; وهي مكافئة للعبارة p=a;

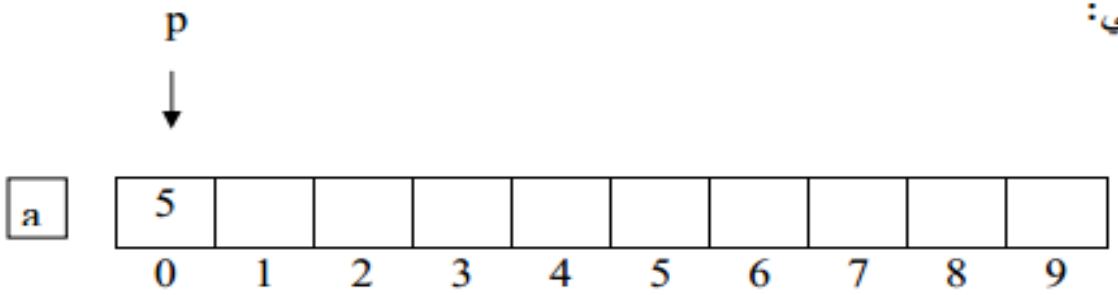
أي



وبهذا يمكن الوصول إلى a[0] عن طريق p وبإمكاننا تخزين القيمة 5 على سبيل المثال كعنصر في الموقع الأول من المصفوفة كما يلي:

```
*p=5;
```

فيصبح الشكل كما يلي:



يمكن تنفيذ ثلاث عمليات حسابية أساسية على المؤشر وهي:

1. زيادة رقم صحيح إلى المؤشر.
2. طرح رقم صحيح من المؤشر.
3. طرح مؤشر من مؤشر.

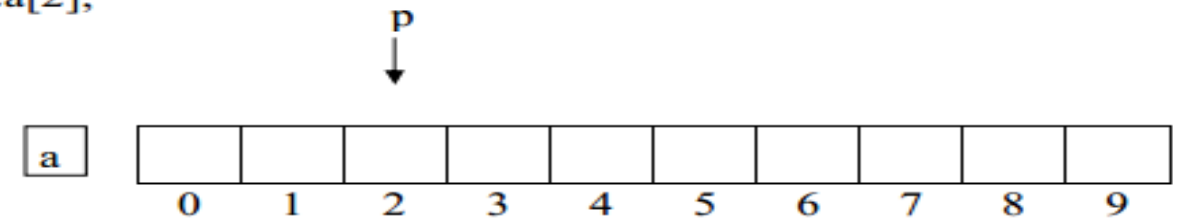
لنفترض التعريف التالي:

```
int a[10],*p,*q;
```

فإذا كان المؤشر p يشير إلى a[i] فإن زيادة j إلى المؤشر تعني أن المؤشر سوف يشير إلى العنصر a[i+j]

ويمكن توضيح العملية كما يلي:

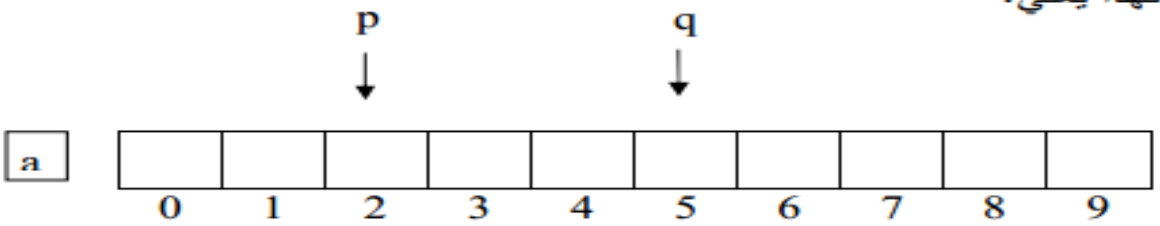
```
p=&a[2];
```



إذا جعلنا

```
q=p+3;
```

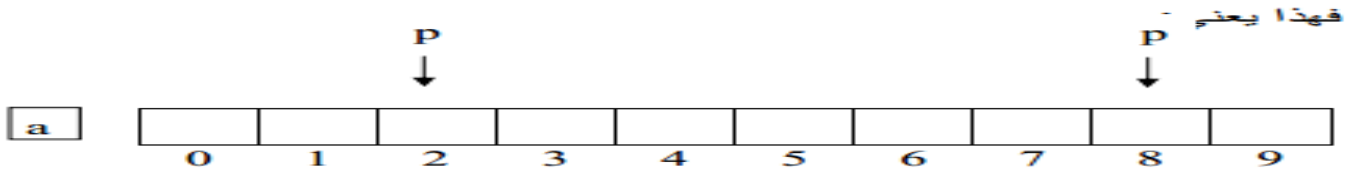
فهذا يعني:



إذا جعلنا

```
p+=6;
```

فهذا يعني:



- السلاسل هي مجرد مصفوفات من النوع char

مثال:اكتبي برنامج يقوم بطباعة سلسلة حرفية باستخدام المؤشرات.

```
#include<iostream.h>
#include<string.h>
void main()
{
    char n[10],*c1;
    strcpy(n,"amera");
    c1=&n[0];
    // طباعة حروف سلسلة حرفية
    while(*c1!='\0')
    {
        cout<<*c1;    c1++;
    }
}
```

ادارة الذاكرة باستخدام المعاملين new و delete

- يستخدم المعامل new لحجز جزء محدد من الذاكرة:

pointername= new data type ;

- يستخدم المعامل delete لتحرير الذاكرة من البيانات التي لم تعد بحاجة اليها:

Delete pointername;

مؤشرات الكائنات

- تستخدم المؤشرات للاشارة الى الكائنات بنفس طريقة عملها مع أنواع البيانات الأساسية والمصفوفات.
- يعمل معامل الوصول (<-) مع مؤشرات الكائنات بنفس طريقة عمل المعامل (.)
- يمكن استخدام المعامل new في انشاء الكائنات اثناء التنفيذ(في حالة لا نعرف عدد الكائنات التي نرغب في انشائها وقت انشاء البرنامج)
- يتم تحرير الذاكرة المستخدمة من قبل الكائن باستخدام المعامل delete

تابع:مؤشرات الكائنات

- تناول بيانات كائن عن طريق المؤشر:
- اشتقاق كائن وتعريف مؤشر ← Distance dist1 ,*cptr1;

- ويمكن استخدام المؤشر للاشارة الى أي كائن النوع Distance كما يلي:

cptr1=&dist; ← //المؤشر يشير الى كائن

- لكن يجب التفريق بين تناول بيانات الصنف من النوع private أو النوع public .

ادخال بيانات كائن (من النوع عام): cin>>cptr1->feet;

استخراج بيانات كائن (من النوع عام): cout<<cptr1->feet

لكن المتغيرات من النوع private فلا يمكن تناولها الا من خلال دوال اعدت خصيصا لذلك:

cptr1->getdist();

cptr1->showdist()

مثال:

```
class Distance               //English Distance class
   {
   private:
      int feet;
      float inches;
   public:
      void getdist()         //get length from user
         {
         cout << "\nEnter feet: ";  cin >> feet;
         cout << "Enter inches: ";  cin >> inches;
         }
      void showdist()        //display distance
         { cout << feet << "\'-" << inches << '\"'; }
   };

int main()
   {
   Distance dist;            //define a named Distance object
   dist.getdist();           //access object members
   dist.showdist();          //   with dot operator

   Distance* distptr;        //pointer to Distance
   distptr = new Distance;   //points to new Distance object
   distptr->getdist();       //access object members
   distptr->showdist();      //   with -> operator
   cout << endl;
   return 0;
   }
```

الدوال التخيلية
virtual functions

- عندما يعلن عن مؤشر ليشير إلى كائنات صنف اساسي يمكن استخدام نفس المؤشر ليشير إلى كائنات الصنف المشتق، فعند التعامل مع الصنف المشتق من خلال مؤشر فان المترجم ينفذ الدالة الخاصة بالصنف الأساسي فقط.
- ولحل المشكلة يتم استخدام الدوال التخيلية وهي دوال يتم تعريفها ضمن الأعضاء الدالية في الصنف الاساسيbase تسبقها كلمة virtualويعاد تعريفها في الأصناف المشتقة.

التعددية الشكلية Polymorphism

الكائنات من النوع المختلف تستجيب استجابة مختلفة لنفس الدالة وهذا يتحقق مع الدوال التخيلية virtual.

لدينا صنف اساسي A يحتوى على دالة int sum (int , int) مشتق منه صنف B يحتوى على دالة int sum (int , int) .

لجعل A تنفذ الدالة sum التابعة له و جعل B ينفذ الدالة sum التابعة له لابد من جعل sum فى الصنف الاساسي base class من النوع virtual.

Virtual int sum(int , int)

اذا لم نضع virtual فعند التعامل مع المشتقة من خلال مؤشر فان المترجم ينفذ sum الخاصة بالصنف الاساسي.

```
class A
{   …   };
class B: public A
{    …   };
void main()
{     A a *s;        // A يشير الى S
      B b1;
      s=&b1;   // A  لانها متفرعة من  b1  يشير الى S
}
```

S مازالت من نوع A

مثال:

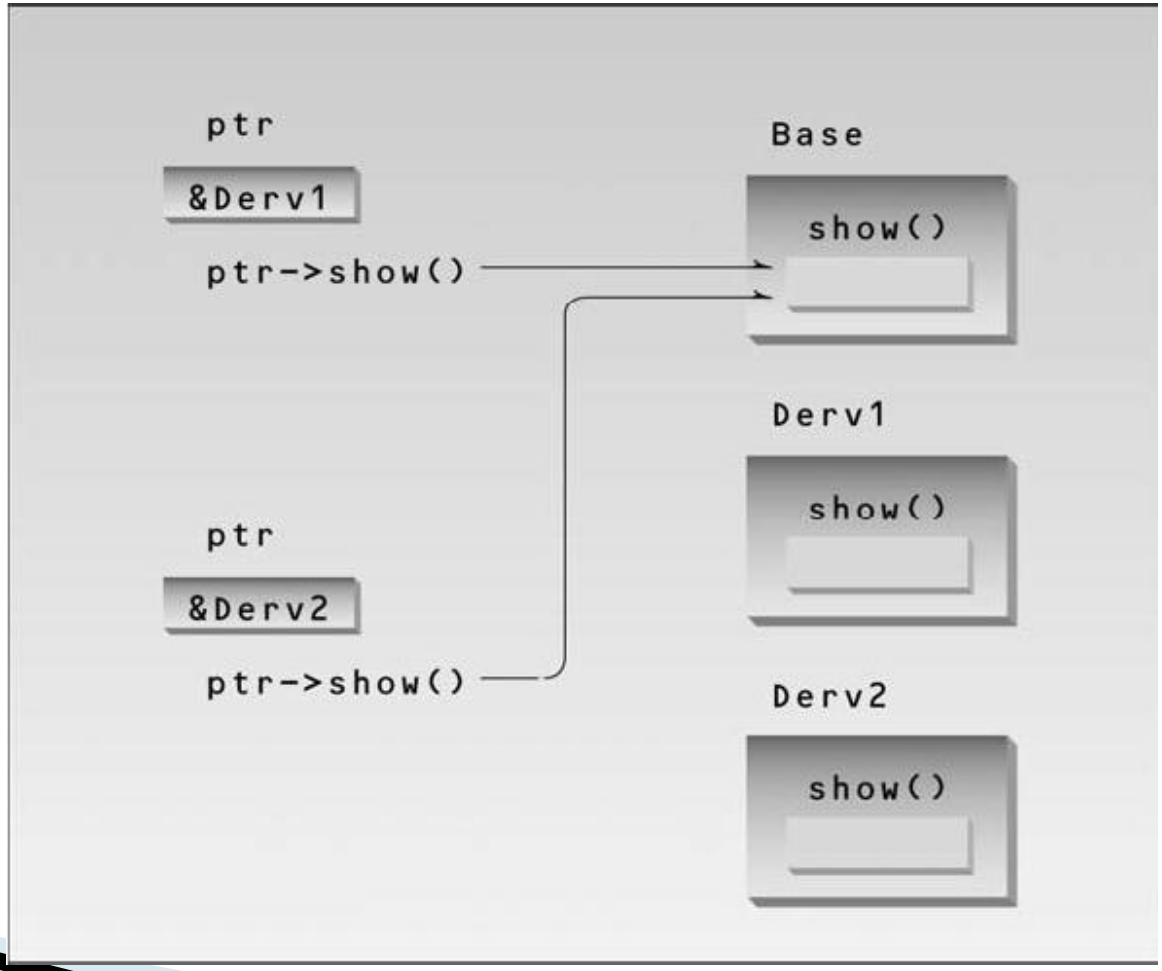
```
class Base                          //base class
   {
   public:
      void show()                   //normal function
         { cout << "Base\n"; }
   };
class Derv1 : public Base          //derived class 1
   {
   public:
      void show()
         { cout << "Derv1\n"; }
   };

class Derv2 : public Base          //derived class 2
   {
   public:
      void show()
         { cout << "Derv2\n"; }
   };
  int main()
     {
     Derv1 dv1;             //object of derived class 1
     Derv2 dv2;             //object of derived class 2
     Base* ptr;             //pointer to base class

    ptr = &dv1;            //put address of dv1 in pointer
    ptr->show();           //execute show()
   ptr = &dv2;            //put address of dv2 in pointer
   ptr->show();           //execute show()
   return 0;
   }
```

Which of the `show()` functions is called here? The output from the program answers these questions:

```
Base
Base
```



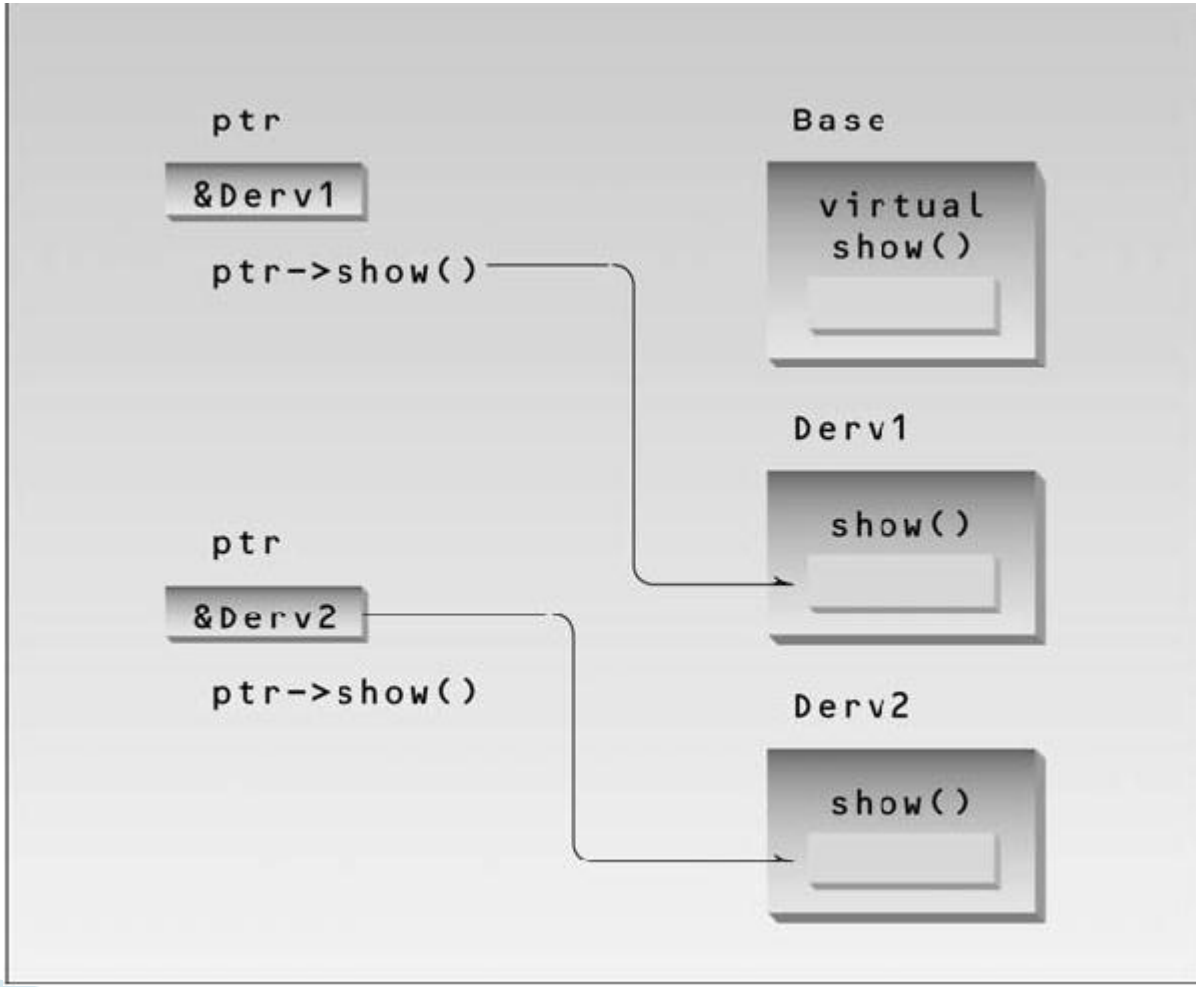
Nonvirtual pointer access.

الوصول الي الدوال التخيلية بالصنف من خلال المؤشرات:

```
class Base                          //base class
   {
   public:
      virtual void show()           //virtual function
         { cout << "Base\n"; }
   };
 class Derv1 : public Base           //derived class 1
    {
   public:
      void show()
         { cout << "Derv1\n"; }
   };
 class Derv2 : public Base           //derived class 2
    {
    public:
       void show()
          { cout << "Derv2\n"; }
    };
 int main()
    {
    Derv1 dv1;             //object of derived class 1
    Derv2 dv2;             //object of derived class 2
    Base* ptr;             //pointer to base class
   ptr = &dv1;             //put address of dv1 in pointer
   ptr->show();            //execute show()
  ptr = &dv2;             //put address of dv2 in pointer
  ptr->show();            //execute show()
  return 0;
  }
```

The output of this program is

```
Derv1
Derv2
```



Virtual pointer access.

الدوال الصديقة(Friend Functions)

- هي دالة يصرح عنها داخل الصنف وتكتب تفاصيلها (جسم الدالة)خارج الـ Classمع امتيازها بحق الوصول الى أعضاء الـClass الخاصة (المعرفة داخل القسم private) أو المحمية (المعرفة داخل القسم protected)
- للتصريح عن الدالة الصديقة نستخدم الكلمة المفتاحية friendكما يلي:

Friend DataType FunctionName (Parameters)

ملاحظات:

– مع أن الاعلان عن الدالة الصديقة يتم داخل الصنف ، إلا أنها لا تعتبر دالة عضو من أعضاء الصنف ، لذلك من الخطأ استدعاؤها باستخدام اسم الكائن مع أداة الوصول(.)

- – يتم استدعاء الدالة الصديقة عن طريق اسمها: Function name(Object name);

–لا تؤثر مستويات الحماية على التصريح عن الدالة الصديقة لذلك يمكن التصريح عن الدالة الصديقة في أي موضع داخل الصنف

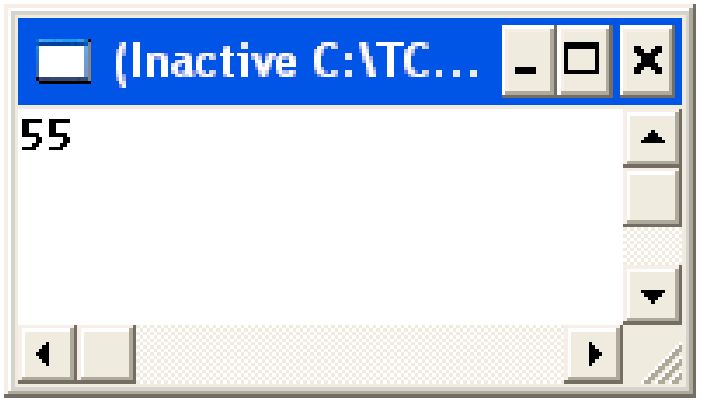
مثال ١ :

```
class Distance                    //English Distance class
   {
   private:
      int feet;
      float inches;
   public:
      Distance() : feet(0), inches(0.0) //constructor (no args)
         {  }
                                        //constructor (two args)
      Distance(int ft, float in) : feet(ft), inches(in)
         {  }
      void showdist()                   //display distance
         { cout << feet << "\'-" << inches << '\"'; }
      friend float square(Distance);    //friend function
   };
float square(Distance d)          //return square of
   {                              //this Distance
float fltfeet = d.feet + d.inches/12;  //convert to float
float feetsqrd = fltfeet * fltfeet;    //find the square
return feetsqrd;                //return square feet
}
int main()
   {
   Distance dist(3, 6.0);         //two-arg constructor (3'-6")
   float sqft;

   sqft = square(dist);           //return square of dist
                                  //display distance and square
   cout << "\nDistance = "; dist.showdist();
   cout << "\nSquare = " << sqft << " square feet\n";
   return 0;
   }
```

مثال٢ :

```
#include<iostream.h>
class ff
{
private:
int x;
public:
ff() {x=0;}
void set(int y) {x=y;}
friend void plus(ff);
};
void plus(ff f)  {cout<<f.x+5;}
void main()
{
ff f1;
f1.set(50);
plus(f1);
}
```



الأصناف الصديقة (Friend Classes)

- يمكن تحويل جميع دوال الصنف الى دوال صديقة في خطوة واحدة بدلا من كتابة كلمة friend مع كل دالة على حده وذلك بجعل الصنف بأكمله صنف صديق باستخدام كلمة friend ايضاً.

-وللتصريح عن صنف صديق : مثال لجعل الصنف tow صديق للصنف oneنكتب العبارة التالية:

```
Friend class tow          //داخل الصنف one
```

(كائنات tow تتعامل مباشرة مع متغيرات one الخاصة والعامة والمحمية).