Structures
&
POINTERS

structures

التركيبة (السجل):

- نوع بيان يعرفه المستخدم user defined data type.
- تجمع بيانات من نوع مختلف.

```
struct [structure tag]
{
   member definition;
    ...
   member definition;
} [one or more structure variables];
```

```
struct Books
{
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
}book;
```

- يمكن تعريف struct داخل main او خارجها.
- نستخدم access operator (معامل الوصول) للتعامل مع أي عنصر في التركيبة.

```
#include <iostream>
struct Books
{
  char  title[50]  ,  author[50];
  int   book_id;
};
Void main( )
{
  struct Books Book1;    // Declare Book1 of type Book
  // book 1 specification
  strcpy( Book1.title, "Learn C++ Programming");
  strcpy( Book1.author, "Chand Miyan");
  Book1.book_id = 6495407;
  …..
}
```

Structures as Function Arguments: التركيبة كمعامل للدالة

```
#include <iostream>
 void printBook( struct Books book );
struct Books
{
   …….
};
Void main( )
{
   struct Books Book1;
   strcpy( Book1.title, "Learn C++ Programming");
   strcpy( Book1.author, "Chand Miyan");
   Book1.book_id = 6495407;

   // Print Book1 info
   printBook( Book1 );
}
```

```
void printBook( struct Books book )
{
   cout << "Book title : " << book.title <<endl;
   cout << "Book author : " << book.author <<endl;
   cout << "Book id : " << book.book_id <<endl;
}
```

Book title : Learn C++ Programming

Book author : Chand Miyan

Book id : 6495407

مؤشر لتركيبة :Pointers to Structures

```
struct Books *sp;
```

اسناد عنوان التركيبة للمؤشر

```
sp = &Book1;
```

الوصول لاى عنصر من عناصر التركيبة باستخدام المؤشر

```
sp->title;
```

تمرير تركيبة للدالة

```
#include <iostream>
 void printBook( struct Books *k );

struct Books
{   ……   };

Void main( )
{
   ……
   printBook( &Book1 );
 }
// This function accept pointer to structure as parameter.
void printBook( struct Books *k )
{
   cout << "Book title : " << k->title <<endl;
   cout << "Book author : " << k->author <<endl;
   cout << "Book id : " << k->book_id <<endl;
}
```

The typedef Keyword مصطلح (typedef)

```
typedef struct
{
  char  title[50];
  char  author[50];
  int   book_id;
}Books;
```

يمكن استخدام الكلمة BOOK مباشرة لتعريف متغيرات من نوع BOOK.

```
Books Book1, Book2;
```

Example: (struct contains function) تركيبة محتوية علي دالة

```
struct c
 {
   int x,y;
   int sum()
   {
     int z=x+y;
     return z;
   }
 }c1;

Void main ( )
{

   c1.x=10;
   c1.y=20;
   cout<<c1.sum();
}
```

Array of Structures: Example

```
#define MAX 100

struct addr {
    char name[30];
    char street[40];
    char city[20];
    char state[3];
    unsigned long int zip;
}addr_list[MAX];
```

```
Main ()
{

Cin>>addr_list[0].name;
    ……
}
```

pointers

Simple Variables

Consider the declaration in C

int i = 3 ;

This declaration tells the C compiler to:

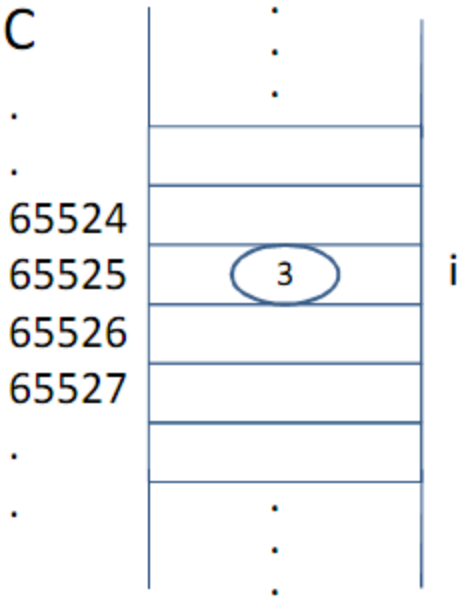
- Reserve space in memory to hold the integer value.
- Associate the name i with this memory location.
- Store the value 3 at this location.

Simple Variables

Consider the declaration in C

int i = 3 ;

Name of a simple variable, which will store a value



Simple Variables

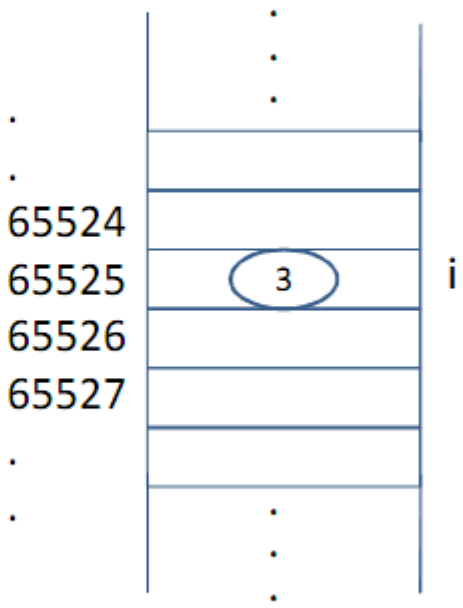
Operator '&'

```
main( )
{
    int i = 3 ;

   Cout<<&i<<"    "<<i<<endl;

}
```

Output?



Simple Variables

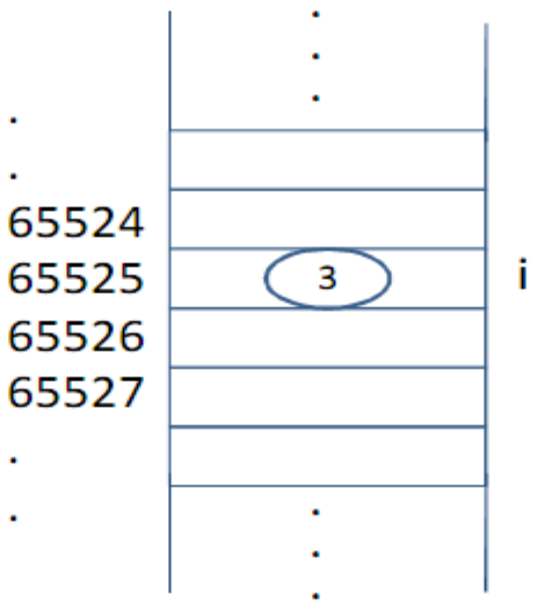
Operator '*'

```
main( )
{
    int i = 3 ;

    cout<<&i<<"    "<<i<<"    "<<*(&i);

}
```

Output?



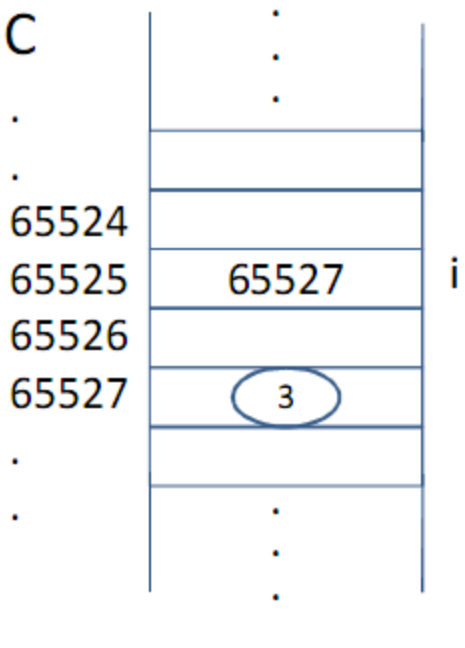
Pointer Variables

Consider the declaration in C

int *i = 3 ;

Name of another variable, but this time it's a pointer variable

This variable will store address rather than value



ما هو المؤشر ؟

متغير يحمل عنوان متغير آخر.

```
type *var-name;
```

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to character
```

```
#include <iostream>
 void main ()
{
   int  var = 20;   // actual variable declaration.
   int  *ip;        // pointer variable

   ip = &var;       // store address of var in pointer variable

   cout << "Value of var variable: " << var << endl;

   // print the address stored in ip pointer variable
   cout << "Address stored in ip variable: "<< ip << endl;

   // access the value at the address available in pointer
   cout << "Value of *ip variable: "<< *ip << endl;
}
```

الناتج

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

Concept	Description
C++ Null Pointers	C++ supports null pointer, which is a constant with a value of zero defined in several standard libraries.
C++ pointer arithmetic	There are four arithmetic operators that can be used on pointers: ++, --, +, -
C++ pointers vs arrays	There is a close relationship between pointers and arrays. Let us check how?

Concept	Description
C++ array of pointers	You can define arrays to hold a number of pointers.
C++ pointer to pointer	C++ allows you to have pointer on a pointer and so on.
Passing pointers to functions	Passing an argument by reference or by address both enable the passed argument to be changed in the calling function by the called function.
Return pointer from functions	C++ allows a function to return a pointer to local variable, static variable and dynamically allocated memory as well.

C++ Null Pointer

يأخذ المؤشر قيمة NULL اذا لم يحمل أي عنوان = صفر.
المؤشر الذي يحمل القيمة NULL يسمي بـ NULL POINTER.

```
#include <iostream>
int main ()
{
  int  *ptr = NULL;
  cout << "The value of ptr is " << ptr ;
  return 0;
}
```

الناتج

```
The value of ptr is 0
```

```
if(ptr)     // succeeds if p is not null
if(!ptr)    // succeeds if p is null
```

فحص قيمة المؤشر

Pointer Comparisons

```
 const int MAX = 3;
Void main ()
{
  int  var[MAX] = {10, 100, 200};
  int  *ptr= var;
  int i = 0;
  while ( ptr <= &var[MAX - 1] )
  {
    cout << "Address of var[" << i << "] = "<< ptr << endl;
    cout << "Value of var[" << i << "] = " << *ptr << endl;
    ptr++;     i++;
  }
}
```

الناتج

```
Address of var[0] = 0xbfce42d0
Value of var[0] = 10
Address of var[1] = 0xbfce42d4
Value of var[1] = 100
Address of var[2] = 0xbfce42d8
Value of var[2] = 200
```

An Example

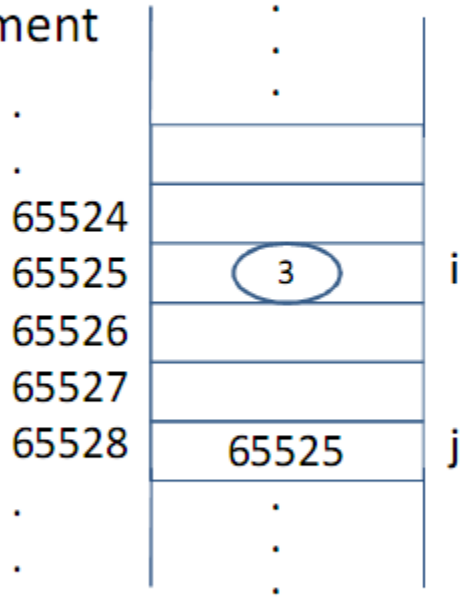
Consider the following C code fragment

```
main( )
{
    int i = 3 ;
    int *j ;

    j = &i ;

    Cout<<“Address of I  = “<<&i<<endl;
    Cout<<“Address of I  = “<<j<<endl;
    Cout<<“Address of j  = “<<&j<<endl;
    Cout<<“value of j = “<<j<<endl;
    Cout<<“value of I = “<<i<<endl;
    Cout<<“value of i = “<<*(&i)<<endl;
    Cout<<“value of i = “<<*j<<endl;

}
```



Parameter Passing

In function calls, parameters are passed by

- Value (of the arguments)
- Reference (addresses of the arguments)

Parameter Passing

- Pass by value

```
main( )
{
    int a = 10, b = 20 ;
    swapv ( a, b ) ;
    printf ( "\na = %d b = %d", a, b ) ;
}
```

Output?

```
swapv ( int x, int y )
{
    int t ;

    t = x ;
    x = y ;
    y = t ;

    printf ( "\nx = %d y = %d", x, y ) ;
}
```

Parameter Passing

- Pass by Reference

```
main( )
{
     int a = 10, b = 20 ;

     swapr ( &a, &b ) ;
     printf ( "\na = %d b = %d", a, b ) ;
}
```

```
swapr( int   *x, int  *y )
{
     int t ;

     t = *x ;
     *x = *y ;
     *y = t ;
}
```

Output?

Pointer Arithmetic

```
int i = 3, *x ;
float j = 1.5, *y ;
char   k = 'c', *z ;

Cout<<"value of i= "<<i<<endl;
Cout<<"value of j = "<<j<<endl;
Cout<<"value of k = "<<k<<endl;

x = &i ;
y = &j ;
z = &k ;

Cout<<"Address in  x  = "<<x<<endl;
Cout<<"Address in y  = "<<y<<endl;
Cout<<"Address in z  = "<<z<<endl;

x++ ;
y++ ;
z++ ;

Cout<<"Address in  x  = "<<x<<endl;
Cout<<"Address in y  = "<<y<<endl;
Cout<<"Address in z  = "<<z<<endl;

}
```

Output?

Incrementing a Pointer:

```
#include <iostream>
 const int MAX = 3;
int main ()
{
  int  var[MAX] = {10, 100, 200};
  int  *ptr;

  ptr = var;
  for (int i = 0; i < MAX; i++)
  {
    cout << "Address of var[" << i << "] = "<< ptr << endl;
    cout << "Value of var[" << i << "] = "<< *ptr << endl;
    ptr++;
  }
  return 0;
}
```

الناتج

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

Decrementing a Pointer:

```
Void main ()
{
   int  var[3] = {10, 100, 200};
   int  *ptr;

   // let us have address of the last element in pointer.
   ptr = &var[2];
   for (int i = 3; i > 0; i--)
   {
      cout << "Address of var[" << i << "] = "<< ptr << endl;
      cout << "Value of var[" << i << "] = "<< *ptr << endl;

      // point to the previous location
      ptr--;
   }
}
```

الناتج

```
Address of var[3] = 0xbfdb70f8
Value of var[3] = 200
Address of var[2] = 0xbfdb70f4
Value of var[2] = 100
Address of var[1] = 0xbfdb70f0
Value of var[1] = 10
```

```
int main ()
{
   int  var[3] = {10, 100, 200};
   int  *ptr;

   // let us have array address in pointer.
   ptr = var;
   for (int i = 0; i < 3; i++)
   {
       cout << ptr << endl;
       cout<< *ptr << endl;

      // point to the next location
      ptr++;
   }
   return 0;
}
```