Southern Technical University
Technical Institute / Qurna
Dep. of Computer Systems Techniques

Second class
Subject : Data Structures
Lecturer : Israa Mahmood Hayder
Lecture no.6,7

المؤشرات(Pointers)

الاسبوع السادس-السابع -

السادس	المؤشرات pointers • تعريف المؤشر • الذاكرة / حجز الذاكرة للمؤشرات وتحريرها • فوائد المؤشرات ومميزاتها المؤشرات والمصفوفات / مصفوفات المؤشرات والمؤشرات للمصفوفات
السابع	• المؤشرات كعناوين • مقارنة المؤشرات، مؤشرات المؤشرات، مؤشرات الدالة

B// Rationale (مبررات الوحدة) :-

The student will learn about dynamic variables (pointers) and how to use them to point to addresses of primitive and non-primitive data structures.

C// Central (الفكرة المركزية):-

- Dynamic storage
- Addresses
- References
- Pointers, Pointer to integers, characters, real data
- Pointer to arrays
- Pointer to functions

D// Objectives (أهداف الوحدة):-

After studying this unit, the student will be able to define pointers and use them in primitive and non primitive DS

3- The Text (عرض الوحدة النمطية):

Pointers : Is dynamic variable is created and destroyed dynamically during the execution of the program, unlike static variables. Dynamic variables are not referenced by user specified name , instead , they are referenced by pointers.

P → V

The dynamic variable "V" is referenced by pointer variable "P" which "points to" V.

int n=20; (static variable)

ox3ffdl8

20

Ex١

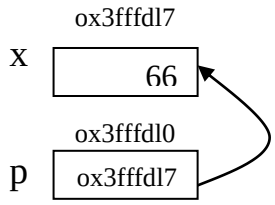
```
Main( )
{
   int   x=66 ;
   int   *p=&x ;
                        // p hold the address of n

   cout<< "n=" << n << ",&n" <<&n<< ",p="<< p << endl;
   cout << "&p=" << &p << endl ;
```

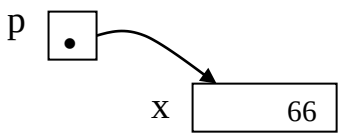
Run

x=66 , &x= ox3fffdl7 , p= ox3fffdl7

&p= ox3fffdl0



and the last pointers can be drown as follows :-



p points to x then *p =66 like n

Ex2

```
main ( )
{
  float  n =3.4;
  float  *p=&n;
  cout << "*p" << *p << endl ;
}
```

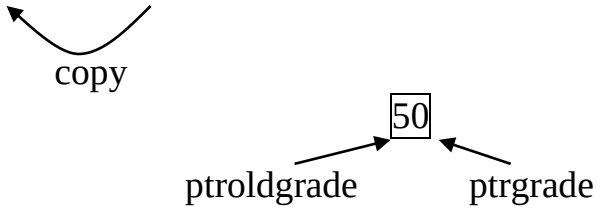
Run : *p=3.4

Ex 3

```
int   stno;
char  grade;
char *ptrgr;                    pointer to grade
int    *ptrst;                  pointer to stno
```

accessing data pointed by pointer :-

```
char  oldgrade;
char  grade = 'A';
char  *ptrgrade;
char  *ptroldgrade;
ptrgrade = & grade;
ptroldgrade = ptrgrade
```



اذا طبعنا ptroldgrade , ptrgrade نجدهما يحويان نفس البيانات

Pointer is used to reduce No. of times data is copied within memory.

- **Pointer arithmetic :-**

Ex1

```
int    stn =1234;
stn ++;
```

run

stn=1235

ex2

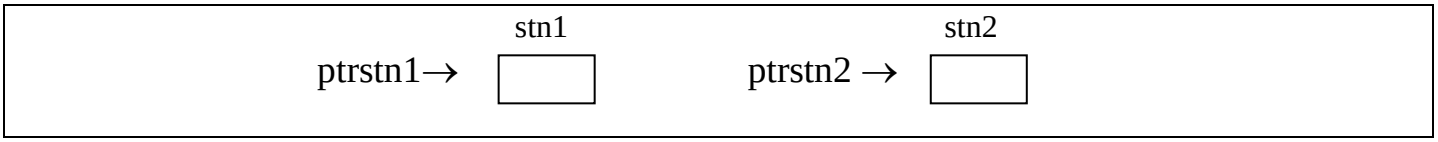
```
int    stn =1234;
stn --;
```

run

stn=1233

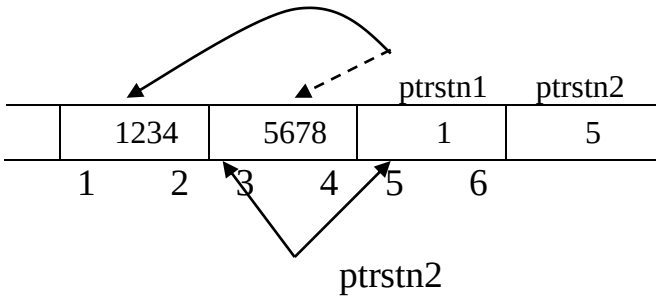
ex3

```
int    stn1 =1234;
int    stn2 =5678;
int    *ptrstn1;
int    *ptrstn2;
```




```
ptrstn1 = &stn1;
ptrstn2 = &ptrstn1;
```

```
ptrstn1 ++;
ptrstn2 --;
delete ptrstn2;
```



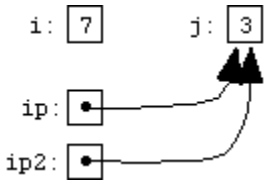
We can also assign pointer values to other pointer variables. If we declare a second pointer variable:

```
int *ip2;
```

then we can say

```
ip2 = ip;
```

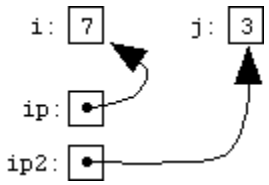
Now `ip2` points where `ip` does; we've essentially made a ``copy'' of the arrow:



Now, if we set `ip` to point back to `i` again:

```
ip = &i;
```

the two arrows point to different places:



We can now see that the two assignments

```
ip2 = ip;
```

and

```
*ip2 = *ip;
```

do two very different things. The first would make `ip2` again point to where `ip` points (in other words, back to `i` again). The second would store, at the location pointed to by `ip2`, a copy of the value pointed to by `ip`; in other words (if `ip` and `ip2` still point to `i` and `j` respectively) it would set `j` to `i`'s value, or 7.

Pointers to Pointers

Since we can have pointers to `int`, and pointers to `char`, and pointers to any structures we've defined, and in fact pointers to any type in C, it shouldn't come as too much of a surprise that we can have pointers to other pointers. If we're used to thinking about simple pointers, and to keeping clear in our minds the distinction between *the pointer itself* and *what it points to*, we should be able to think about pointers to pointers, too, although we'll now have to distinguish between the pointer, what it points to, and what the pointer that it points to points to. (And, of course, we might also end up with pointers to pointers to pointers, or pointers to pointers to pointers to pointers, although these rapidly become too esoteric to have any practical use.)

The declaration of a pointer-to-pointer looks like

```
int **ipp;
```

where the two asterisks indicate that two levels of pointers are involved.

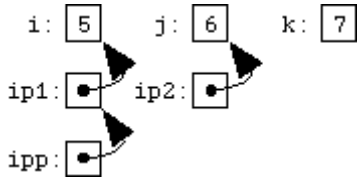
Starting off with the familiar, uninspiring, kindergarten-style examples, we can demonstrate the use of `ipp` by declaring some pointers for it to point to and some `int`s for those pointers to point to:

```
int i = 5, j = 6; k = 7;
int *ip1 = &i, *ip2 = &j;
```

Now we can set

```
ipp = &ip1;
```

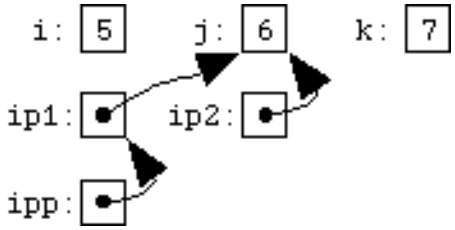
and `ipp` points to `ip1` which points to `i`. `*ipp` is `ip1`, and `**ipp` is `i`, or 5. We can illustrate the situation, with our familiar box-and-arrow notation, like this:



If we say

```
*ipp = ip2;
```

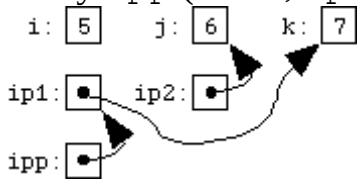
we've changed the pointer pointed to by `ipp` (that is, `ip1`) to contain a copy of `ip2`, so that it (`ip1`) now points at `j`:



If we say

```
*ipp = &k;
```

we've changed the pointer pointed to by `ipp` (that is, `ip1` again) to point to `k`:



Derived types :- الانواع المشتقة

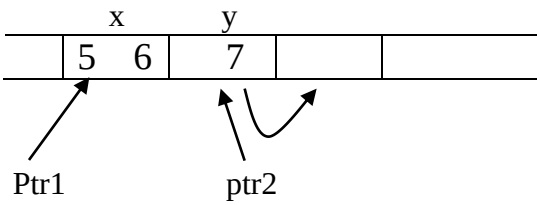
int &r = n ;	reference
int *p = & n ;	pointer
int a[ ] = { 33 , 66 }	array
const int c =33	const int
int f( ) = { return 33 }	function returns int

Notes :- (On Pointers)

- pointer to pointer is variable whose value is an address of another pointer variable .
- use pointer to pointer in program to arrange data without having to move data in memory.
- If we print the content of pointer , the address shown on the screen.

Ex

```
Main ( )
{
  int  x=5;
  int  y=7;
  int *ptr1=&x ;
  int *ptr2=&y ;
  *ptr1 +=1;
  *ptr2 ++;
  ptr1=ptr2;
  *ptr1=*ptr2;
  cout << "*ptr1=" << ptr1 << " ,*ptr2" << ptr2 << endl ;
  delete ptr1;
  delete ptr2;
  return 0;      }
```



Quiz1:

Write program to define pointer to long integer and decrement by 1 then print the pointer value and address.

2- Using pointers to arrays:-

When we use arrays, we can deal with the elements using an index that relative to the base address. In structured languages, pointers are used with linked list.

Ex1:- Print The Array Elements

```
Main ( )
{
  Short   a[3]={22,33,44};
  cout << "a="<< a << endl;
  cout<<"size of short="<< sizof(short) <<endl;
  short*end=a+3;            // convert size to of  list 6
  for(short * p=a;p<end;p++)
  {
    cout<<"p="<<p<<"\t*p="<<*p<<endl;
  }  }
```

Run

```
a=0x3fffla
size of short=2
p=0x3fffla          *p=22
p=0x3ffflc          *p=33
p=0x3fffle          *p=44
```

* p++ إذا المؤشر double يتقدم ٨ بايتات وإذا short يتقدم بايتين وإذا int يتقدم ٤ بايتات.
* إذا أردنا العنصر الخامس نقوم بمايلي:-

```
float* p=a      // a[0]
pt =5           //a[5]
```

ملاحظة :- إذا a[8] تعني a[0-7] وللانتقال للعنصر الأخير

```
float*p=a[7]
```

ex2:-

```
  main ( )
  {
    short a[ ]={22,33,44,55,66};
    for ( short* p=a ; p<a+5; p++)
        cout<<"p="<<p<<"*p="<<*p<<endl;    }
```

Strings as pointers:

Another way of accessing a *contiguous* chunk of memory, instead of with an array, is with a *pointer*.

Since we are talking about *strings*, which are made up of *characters*, we'll be using *pointers to characters*, or rather, `char *`'s.

However, pointers only hold an address, they cannot hold all the characters in a character array. This means that when we use a `char *` to keep track of a string, the character array containing the string must already exist (having been either statically- or dynamically-allocated).

Below is how you might use a *character pointer* to keep track of a string.

```
char label[] = "Single";
char label2[10] = "Married";
char *labelPtr;

labelPtr = label;
```

We would have something like the following in memory (e.g., supposing that the array `label` started at memory address `2000`, etc.):

```
label @2000
-----------------------------
| S | i | n | g | l | e | \0 |
-----------------------------

label2 @3000
-----------------------------------------
| M | a | r | r | i | e | d | \0 |   |   |
-----------------------------------------

labelPtr @4000
--------
| 2000 |
--------
```

Note: Since we assigned the pointer the address of an *array of characters*, the pointer must be a *character pointer*--the types must match.

Also, to assign the address of an array to a pointer, we do not use the *address-of* (`&`) operator since the name of an array (like `label`) behaves like the address of that array in this context. That's also why you don't use an ampersand when you pass a string variable to `scanf()`, e.g,

```
int id;
char name[30];
```

```
cin>> &id >> name;
```

Now, we can use `labelPtr` just like the array name `label`. So, we could access the third character in the string with:

```
Cout<< "Third char is: << labelPtr[2];
```

It's important to remember that the only reason the pointer `labelPtr` allows us to access the `label` array is because we made `labelPtr` point to it. Suppose, we do the following:

```
labelPtr = label2;
```

Now, no longer does the pointer `labelPtr` refer to `label`, but now to `label2` as follows:

```
label2 @3000
-----------------------------------------
| M | a | r | r | i | e | d | \0 |   |   |
-----------------------------------------

labelPtr @4000
--------
| 3000 |
--------
```

So, now when we subscript using `labelPtr`, we are referring to characters in `label2`. The following:

```
printf("Third char is: %c\n", labelPtr[2]);
```

prints out **r**, the third character in the `label2` array.

Ex

```
char   n1='D'   , n2 ='A'  , n3='C' , n4 ='B';
char *p1  , **p2;
p1= & n1;
p2 = &p1;
cout << **p2 ;
```

	n1	n2	n3	n4	p1	p2
	D	A	C	B	1	5
	1	2	3	4	5	6

Resut : D

Pointer to Functions

One use is returning pointers from functions, via pointer arguments rather than as the formal return value. To explain this, let's first step back and consider the case of returning a simple type, such as `int`, from a function via a pointer argument. If we write the function

```
f(int *ip)
{
	*ip = 5;
}
```

and then call it like this:

```
int i;
f(&i);
```

then `f` will ``return'' the value 5 by writing it to the location specified by the pointer passed by the caller; in this case, to the caller's variable `i`. A function might ``return'' values in this way if it had multiple things to return, since a function can only have one formal return value (that is, it can only return one value via the `return` statement.) The important thing to notice is that for the function to return a value of type `int`, it used a parameter of type pointer-to-`int`.

Now, suppose that a function wants to return a *pointer* in this way. The corresponding parameter will then have to be a pointer to a pointer. For example, here is a little function which tries to allocate memory for a string of length `n`, and which returns zero (``false'') if it fails and 1 (nonzero, or ``true'') if it succeeds, returning the actual pointer to the allocated memory via a pointer:

```
#include <stdlib.h>
int allocstr(int len, char **retptr)
{
	char *p = malloc(len + 1);  /* +1 for \0 */
	if(p == NULL)
		return 0;
	*retptr = p;
	return 1;
}
```

The caller can then do something like

```
char *string = "Hello, world!";
```

```
char *copystr;
if(allocstr(strlen(string), &copystr))
      strcpy(copystr, string);
else  fprintf(stderr, "out of memory\n");
```

(This is a fairly crude example; the `allocstr` function is not terribly useful. It would have been just about as easy for the caller to call `malloc` directly. A different, and more useful, approach to writing a ``wrapper'' function around `malloc` is exemplified by the `chkmalloc` function we've been using.)

References:

1- Data Structures Demystified, by Jim Keogh and Ken Davidson, ISBN:0072253592, McGraw-Hill/Osborne © 2004

2- هياكل البيانات / الطبعة الثانية، تاليف د.عصام الصفار،اصدارات السفير للنشر/ بغداد، ٢٠٠١

3- Steve Summit // Copyright, 1996 http://www.eskimo.com/~scs/cclass/notes/sx10a.html

4- Robert I. Pitts , BU CAS CS - Strings as arrays, as pointers, and string.h Copyright © 1993-2000, http://www.cs.bu.edu/teaching/c/string/intro/.

5- الحقيبة التعليمية مادة "هياكل البيانات "، اعداد : نفارت الياس يوسف ،المعهد التقني كركوك-