

# Pointers II

```

#include <iostream>
using namespace std;

int main() {
    int a = 10;
    int *y = &a;    // y is a pointer storing the address of a
    int &x = *y;    // x is a reference to the value at y (which is a)

    cout << "a: " << a << endl;
    cout << "x: " << x << endl;
    cout << "y: " << *y << endl;

    x = 20;    // Changing x also changes a
    cout << "Updated a: " << a << endl;

    return 0;
}

```

## Arrays And Pointers

# Arrays are themselves pointers

- The name of an array variable in C++, without the use of the [ ] operator, represents the starting address of the array. This address can be stored in a pointer variable
- Since array values are guaranteed to be in contiguous memory, you can access array values using this one pointer

Examples of this is the "pointer arithmetic"

Ex: `int A[ 3] = { 2, 4, 6 };`

`int *iPtr;`

`iPtr = &A;` *//error, A is actually is the address of the first element in the array*

`iPtr = A;` *// it is equivalent to iPtr=&A[0]; //ok*

`cout << "value: " << *iPtr << endl;` *// cout<< A[0];*

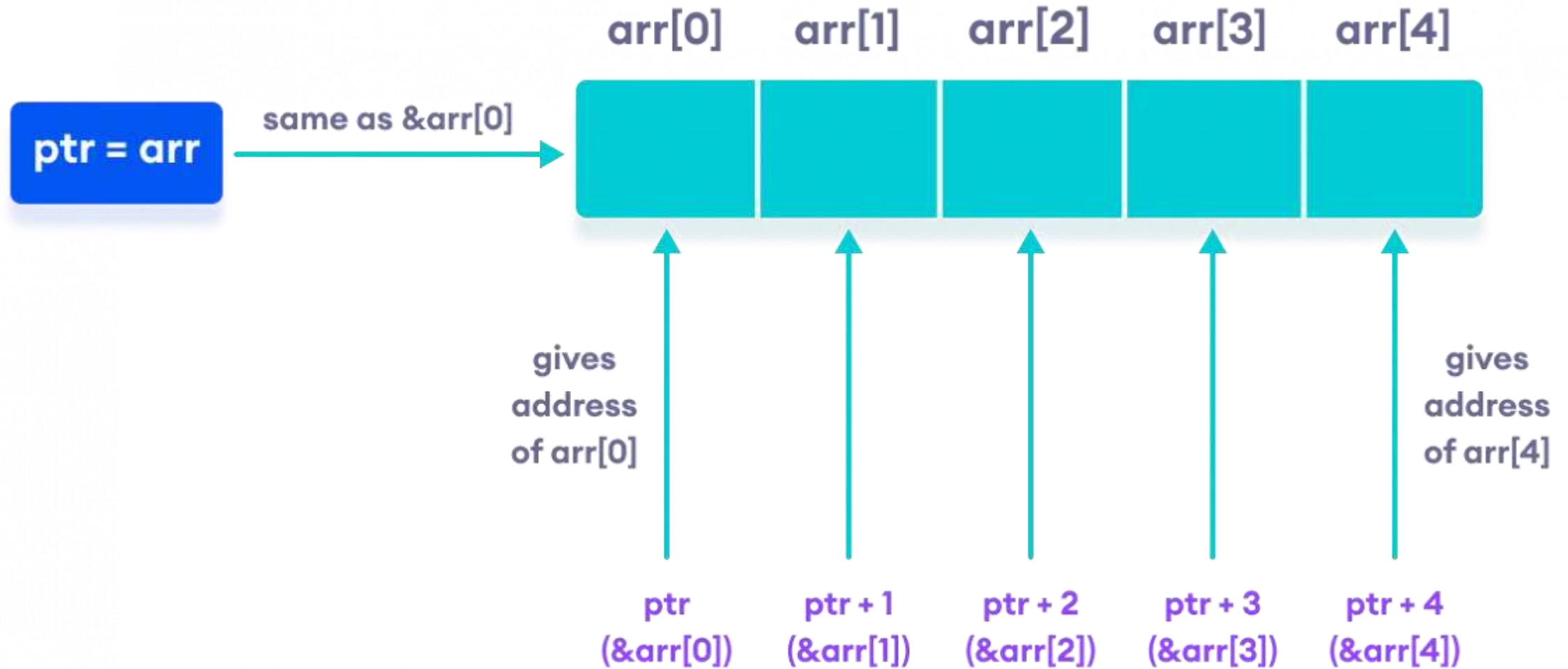
`cout<< " address of first element in the array "<<iPtr<<endl;`

Comments:

- Assigns iPtr to point to the first integer in the iAry array
- The program prints the value stored in the location iPtr that points to the first element in the array A, in this case

## Output

*Value: 2 address of first element in the array 0x28fed0*



# Arrays and pointers

```
int *ptr; int arr[5]; ptr = arr;
```

ptr + 1 is equivalent to &arr[1];

ptr + 2 is equivalent to &arr[2];

ptr + 3 is equivalent to &arr[3];

ptr + 4 is equivalent to &arr[4];

Similarly, we can access the elements using the single pointer. For example,

```
// use dereference operator
```

```
*ptr == arr[0];
```

\*(ptr + 1) is equivalent to arr[1];

\*(ptr + 2) is equivalent to arr[2];

\*(ptr + 3) is equivalent to arr[3];

\*(ptr + 4) is equivalent to arr[4];

# Accessing arrays through pointers

We can access arrays using array name or pointer to the array as follows

## Example 1:

1- `{ int b[5]={10,20,30,40,50};`

2- `int *pt=b;            // pt is pointer to array b`

`// *pt=&b    error`

3- `cout<<" array through indices "<<endl;`

4- `for(int i=0;i<5;i++)`

5- `cout<<b[i]<<" "; cout<<endl<<endl;`

6- `cout<<" array as pointer "<<endl;`

7- `for(int i=0;i<5;i++)cout<<*(b+i)<<" ";`

8- `cout<<endl<<endl;    cout<<" values of array from pointer "<<endl;`

9- `for(int i=0;i<5;i++)cout<<*(pt+i)<<" "; }`

**Example 1 (cont.)**

***10- cout<<endl<<endl; cout<<" addresses of    of array "<<endl;***

***11- for(int i=0;i<5;i++)     cout<<pt+i<<"   ";***

***12- cout<<" addreses of array from array name   "<<endl;***

***13- for(int i=0;i<5;i++)cout<<(b+i)<<"   "; }***

We can access the elements of the array using `b[i]` as in statement 5, or using the name of the array itself (`b+i`), statement 7 or from its pointer `*(pt+i)` statement 11. `Cout<<Ptr+i`, will print the addresses of the array (each element stored in 4 bytes)

## Output

array through indices

10 20 30 40 50

array as pointer

10 20 30 40 50

values of array from pointer

10 20 30 40 50

addresses of array

0x0018ff3c 0x0018ff40 0x0018ff44 0x0018ff48 0x0018ff4c

addresses of array from array name

0x0018ff3c 0x0018ff40 0x0018ff44 0x0018ff48 0x0018ff4c



## Example: assign values to array by its name as pointer

### Example 2:

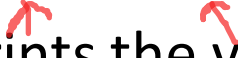
```
1-  {  int i, num[10];
2-  cout<< " assign values to array "<<endl;
3-  for(i=0;i<6;i++)
4-  { *num=i;          cout<< *num <<"   "; }
5-  //num++  error
6-  cout<<endl<<" first element after loop = "<<num[0]<<endl;
7-  cout<< " last element after loop = "<<num[5]<<endl;  //error, num[5] has no value
8-  cout<<" correct assign of array values "<<endl<<endl;
9-          *num= 80;   cout<<" first value of array=  "<<num[0]<<endl;
10-         *(num+1)=100; cout<<" second value of array=  "<<num[1]<<endl;
11-         *(num+9)=55;   cout<<" last value of array=  "<<num[9]<<endl;
12-         cout<<" assign values to array by its name as pointer "<<endl;
13-         for(i=0;i<10;i++)
14-         { *(num+i)= 2*i; cout<< " value "<<(*num+i) <<" pointer "<<(num+i)<<" actual
values "<< *(num+i)<<endl;} }
```

array name is a constant pointer

can change value that address hold  
but

cant change the address

## Comments

- In statement 4, `*num= i`; assign the value of `i` to the first element in the array, so after exit from loop, we find that `num[0]= 5`
- While array name is pointer itself, **we can't increment it as in pointers**, so `num++` is an error
- In statements 9, 10 and 11, we assign values to some elements of the array  
`*( num+i)= value`, assign value to the element `num[i]` in the array.
- In statement 14, **`*(num+i)`** = `i`; assign the values `i` to the elements of the array
- `cout<<*num+ i`, prints the value of `num[0]+value of i`.  

- To get the actual values of the array, we must use `*(num+i)` or `*(pointer of array + i)`.

## Output

assign values to array

0    1    2    3    4    5

first element after loop = 5

last element after loop = 1703748 // any random value, error

correct assign of array values

first value of array= 80

second value of array= 100

last value of array= 55

assign values to array by its name as pointer

value 0 pointer 0x0019ff04 actual values 0

value 1 pointer 0x0019ff08 actual values 2

value 2 pointer 0x0019ff0c actual values 4

value 3 pointer 0x0019ff10 actual values 6

value 4 pointer 0x0019ff14 actual values 8

value 5 pointer 0x0019ff18 actual values 10

assign values to array

# Arrays and pointers

هالام

`int *x [10]` -> array of pointers

`int (*x)[10]` -> pointer to array of 10 elements

## Syntax to Declare Pointer to 2D Array

`data_type (*array_name)[column_size];`

For example,

`int (*ptr)[4] = arr;`

Or `int *ptr[row size]`

pointer for array = array name != & array name

Here, if array `arr[3][4]` declares an array with 3 rows (outer arrays) and 4 columns (inner arrays) of integers. If we declare an array as `int arr[n][m]` then `n` is the number of rows or we can say number of arrays it stores and `m` denotes the number of element each array(row) have in integers as `int` data type is declared.

## Syntax to Declare Pointer to 3D Array

`data_type (*array_name)[col_size][depth_size];`

For example, for array `int arr[1][2][3]`, we can declare a pointer to it as:

`int (*ptr)[2][3] = arr;`

pointer to 2d array

1) `int (*pointer) [columns]`

2) `int *pointer [rows]`

# Arrays and pointers

```
int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

We know that the **name of an array is a constant pointer** that points to 0th 1-D array and if it starts at address 5000. Since arr is a 'pointer to an array of 4 integers', according to pointer arithmetic the expression arr + 1 will represent the address 5016 and expression arr + 2 will represent address 5032.

هالام

for 2D array

arr	-	Points to 0 <sup>th</sup> element of arr	-	Points to 0 <sup>th</sup> 1-D array	-	5000
arr + 1	-	Points to 1 <sup>th</sup> element of arr	-	Points to 1 <sup>st</sup> 1-D array	-	5016
arr + 2	-	Points to 2 <sup>th</sup> element of arr	-	Points to 2 <sup>nd</sup> 1-D array	-	5032

addresse of rows(arrays)

\* or [] -> addrese  
\*\* or [][] -> value

$$[i][j] == * (* (\overset{\text{pointer}[i]}{\text{pointer}} + i) + j)$$

## Ex: 3 Example of 2D array with pointer

If we want to get the address of each element in the 2D array, (\* (pointer name+i)+j) as follows:

```
int main()
{ int(*p)[4];
  cout<<" addresses of each row in the Array from its pointer "<<endl;
    for(int i=0;i< 5; i++)
      cout<<(p+i)<<" ";
      cout<<endl<<endl;

  cout<<" address of each element in the array"<<endl<<endl;
  for(int i=0; i< 5;i++) {for (int j=0; j< 4; j++)
    cout<<(* (p+i)+j)<<" ";
    cout<<endl; } return 0;}
```

## Output

addresses of each row in the Array from its pointer // (p+i)

0x61fdb0 0x61fdc0 0x61fdd0 0x61fde0 0x61fdf0

// \*(p)+i gives the addresses of first row

address of each element in the array // from pointer (\*(p+i)+j)

0x61fdb0 0x61fdb4 0x61fdb8 0x61fdbc

0x61fdc0 0x61fdc4 0x61fdc8 0x61fdcc

0x61fdd0 0x61fdd4 0x61fdd8 0x61fddc

0x61fde0 0x61fde4 0x61fde8 0x61fdec

0x61fdf0 0x61fdf4 0x61fdf8 0x61fdfc

## Ex 4: Another example of 2D array with pointer

```
int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };  
int (*ptr)[4]; ptr = arr;  
cout<<" addresses of rows of the array using increments of ptr "<<endl  
<< ptr<<" "<< ptr + 1<<" "<< ptr + 2<<endl<<endl;  
cout<<" addresses of first row using increments of *ptr "<<endl  
<<* ptr<<" "<< *(ptr) + 1<<" "<< *(ptr) + 2<<" "<<*(ptr)+3<<endl<<endl;  
  
cout<<" values of first row from pointer "<<endl;  
cout<<** ptr<<" "<<** (ptr) + 1<<" "<< ** (ptr) + 2<<" "<<** (ptr)+3<<endl<<endl;  
  
cout<<" another way of rows addresses of array "<<endl<<endl;  
cout<<* ptr<<" "<< *(ptr + 1)<<" "<< *(ptr + 2)<<endl<<endl;
```



## Example of 2D array with pointer (cont.)

```
cout<<" values of array of first column "<<**ptr<<" "<<* *(ptr + 1)<<" "<<** (ptr + 2)<<endl;
```

```
cout<<" addresses of second row "<<*(ptr+1 )<<" "<<*(ptr + 1)+1<<" "<< *(ptr + 1) +  
2<<endl<<endl;
```

```
cout<<" values of second row"<< ** (ptr+1) <<" "<<* (*(ptr + 1)+1)<<" "<< *( *(ptr + 2) +  
1)<<endl<<endl;
```

```
cout<<" values of some elements (increment row and column "  
<<* (*(ptr+1)+2)<<" "<<* (*(ptr+2)+1)<<endl<<endl;
```

```
cout<<" another way to get values from pointer "<<endl;
```

```
cout<< ptr[0][0]<<" "<<ptr[1][2]<<" "<< ptr[2][3]<<endl;
```

```
cout<<" some values from array name"<<endl<<endl;
```

```
cout<<arr[0][0]<<" "<<arr[1][2]<<" "<<arr[2][3]; return 0;}
```

addresses of rows of the array using increments of ptr  
0x61fdf0 0x61fe00 0x61fe10

**addresses** of first row using increments of **\*ptr**  
0x61fdf0 0x61fdf4 0x61fdf8 0x61fdfc

**values** of first row from pointer **\*\*ptr**  
1 2 3 4

another way of rows addresses of array \*(ptr+i)  
0x61fdf0 0x61fe00 0x61fe10

**values** of array of first column 1 5 9 **\*\*\*(ptr+i)**  
**addresses** of second row 0x61fe00 0x61fe04 0x61fe08 **\*(ptr + 1)+i**  
**values** of second row 5 6 10 **\*\*\*(ptr + 1)+i**

Values of some elements (increment row and column 7 10

another way to get values from pointer ptr[i][j] 1 7 12  
some values from array name arr[i][j] 1 7 12

# Array of pointers , another method to declare 2D array

We can create array of pointers as shown:

## Example 5

```
{int A[3][4]; int *p[3]; // p is an array of integer pointers
```

```
for(int i=0; i<3; i++)
```

```
{ p[i]=A[i]; //p points to the two dimensional array A
```

```
cout<<" row no. "<<i<<" through pointer p[ "<<i<<" ] "<<endl;
```

```
for(int j=0; j<4; j++)
```

```
{ A[i][j]=2*i+ 3*j; cout<<* (p[i]+j)<<" "; } cout<<endl; }
```

```
// the above statement is equivalent to cout<<A[i][j]
```

A[3][4]



0	3	6	9
2	5	8	11
4	7	10	13

## Output

row no. 0 through pointer p[ 0]

0 3 6 9

row no. 1 through pointer p[ 1]

2 5 8 11

row no. 2 through pointer p[ 2]

4 7 10 13

## Arrays and pointers

We know, the pointer expression  $*(arr + i)$  is equivalent to the subscript expression  $arr[i]$ . So  $*(arr + i)$  which is same as  $arr[i]$  gives us the base address of  $i$ th 1-D array.

To access an individual element of our 2-D array, we should be able to access any  $j$ th element of  $i$ th 1-D array.

we can get the addresses of subsequent elements in the  $i$ th 1-D array by adding integer values to  $*(arr + i)$ .

For example  $*(arr + i) + 1$  will represent the address of 1st element of 1st element of  $i$ th 1-D array and

$*(arr+i)+2$  will represent the address of 2nd element of  $i$ th 1-D array.

Similarly  $*(arr + i) + j$  will represent the address of  $j$ th element of  $i$ th 1-D array. On dereferencing this expression we can get the  $j$ th element of the  $i$ th 1-D array.

Thus the pointer expression  $*( (*(arr + i) + j ) + k)$  is equivalent to the subscript

expression  $arr[i][j][k]$ .

## Ex: 6 Example of pointer to 3D array

// C program to print the elements of 3-D

// array using pointer notation

```
int arr[2][3][3] = { 5, 10, 22, 6, 11, 44, 7, 12, 55, 20, 30, 66, 21, 31, 77, 22, 32, 88 };
```

```
int i, j, k;
```

```
for (i = 0; i < 2; i++)
```

```
{ cout<<" first dimension "<<endl;
```

```
  for (j = 0; j < 3; j++)
```

```
  {      for (k = 0; k < 3; k++)
```

```
    cout<< *(*(*arr + i) + j) + k) <<"  ";
```

```
    cout<<" second dimension "<<endl;
```

```
    cout<<endl;  } } return 0;}
```

# Output

**Elements of the 3D array through pointer**

**first dimension**

**5 10 22 second dimension**

**6 11 44 second dimension**

**7 12 55 second dimension**

**first dimension**

**20 30 66 second dimension**

**21 31 77 second dimension**

**22 32 88 second dimension**

## Strings and pointers

### Example 7:

*// strings with pointers*

```
1- int main() {int i;
2-     char *c, st[20]="C++ exam";
3-     c=st; // c is a pointer to string st
4-     // c=&st Error
5-     cout << *st<<endl; // prints character st[0]
6-     cout<<*(st+2)<<endl; // prints character st[2]
7-     cout<<" string st = "<<st<<endl<<endl; print while string
8-     cout<<" string through pointer c ="<<c<<endl<<endl;
9-     for(i=0; i<7;i++)
10-    { cout <<" character "<<st[i] <<" pointer "<<(c+i)<<" character from
    pointer "<<*(c+i)<<" string "<<(st+i)<<endl; }
    return 0;}
```

## Comments

- In statement 3, the pointer c points to the string st.
- `cout<<*st` ; prints the first character in the string `st`, while `cout<<*(st+2)` prints the character `st[2]`.
- We can print the string using the pointer c of the string as in statement 8
- We can print the characters of the string using `cout<<st[i]`; or `cout<<*(c+i)` as in statement 10

quiz 2025

- When we use `cout<< (c+i)` or `cout<<st+i`, the string will be printed starting from character no. *i*.

هالام



## Solution

C

+

string st = C++ exam

string through pointer c =C++ exam

character C pointer	C++ exam	character from pointer C	string	C++ exam
character + pointer	++ exam	character from pointer +	string	++ exam
character + pointer	+ exam	character from pointer +	string	+ exam
character pointer	exam	character from pointer	string	exam
character e pointer	exam	character from pointer e	string	exam
character x pointer	xam	character from pointer x	string	xam
character a pointer	am	character from pointer a	string	am

If we want to see the address of the string as array of characters, we should cast the pointer to another pointer type, such as `int *`. Thus, `c` displays as the string "C++ exam", but `(int *)c` displays as the address where the string is located.

### Example to print addresses of string as array of characters through pointers

```
char *c, st[20]="C++ exam";  
    c = st; // c is a pointer to string st  
    cout<<c<<" "<<" address "<<(int *)c<<endl<<endl;  
    for(int i=0;i<6;i++)  
        cout << (c+i)<<"   address "<<(int *)(c+i)<<endl;
```

### Output

C++ exam address 0x0019ff24

C++ exam address 0x0019ff24

++ exam address 0x0019ff25

+ exam address 0x0019ff26

exam address 0x0019ff27

exam address 0x0019ff28

xam address 0x0019ff29 //each character has one byte

## Pointers and strings that declared by data type string

Example 8:

```
{ // pointers and class string
```

```
    {string st, *sptr; st="mohammed"; // pointer to string  
that has been declared as string data type
```

```
    sptr=&st;
```

although string is 1d array but its treated as variable so it needs &

```
    // sptr=st; // error
```

```
    cout<<"string " <<st<<endl<<"address of string st  
    "<<sptr<<endl;
```

```
        for(int i=0;i<8; i++)cout<<"char. no. " <<i<<" is " <<st[i]  
        <<endl;        sptr++; cout<<"address after incrementing  
pointer of string " <<sptr; return 0;}
```

هالام



`*(*(arr+i)+j)` cant be used with array of array of string so use `[][]`

### Output

string mohammed  
address of string st  
char. no. 0 is m  
char. no. 1 is o  
char. no. 2 is h  
char. no. 3 is a  
char. no. 4 is m  
char. no. 5 is m  
char. no. 6 is e  
char. no. 7 is d

0x61fde0

address after incrementing pointer of string 0x61fe00

```
string arr [2]= {"omar","ahmed"};
```

```
cout<< *(arr+1)<<endl;    -> ahmed
```

```
cout<<*arr[0] <<endl;      -> o
```

## Example 9

```
1- main( ) { string s2=" new string";    string *ss;  
2- cout<<" address of string s2 "<<&s2<<endl;  
   cout<<" string s2 = "<<s2<<endl;  
3- // ss=s2; error  
4- ss=&s2; cout<<" address after ss is pointing to string s2 "<<ss<<endl;  
5- for(int i=0; i<7;i++) cout<< s2[i]<<endl; }
```

- In statement 1, pointer ss points to string
- In statement 2, cout<<&s2, prints the address of the string s2
- To point to the string, we use ss=&s2 as in statement 4, so statement 3 is wrong
- In statement 5 cout<<s2[i]; prints the characters of string s2

## Output of example 9

address of string s2 0x28fed0

string s2 = new string

address after ss is pointing to string s2 0x28fed0

n

e

w

s

t

R

## Array of pointers (cont.)

### Example 10 : array of pointers to strings

```
int main()
{
    *s[0] *s[1] *s[2] *s[3]
    const char *s[4]={"logic","design","c++","computer"};

    for(int i=0;i<4;i++) cout char pointer == cout content
        cout<<" value "<< s[i]<<" address "<< casting to print address (int *)s[i]<<endl;

    string p[4]={" test_1"," test_2"," test_3"," test_4"};
    string *ps=p; // pointer to array of strings

    cout<<endl<<" print strings as data type string from its array of pointers
    "<<endl;

    for(int i=0;i<4;i++)
        cout<<"address "<<(ps+i)<<" value "<<ps[i]<<endl; } // it is
    equivalent to cout<<p[i]
```

## Output

```
value logic      address 0x405001 // logic takes 5 bytes
value design    address 0x405007 // design takes 6 bytes and so on
value c++       address 0x40500e
value computer  address 0x405012
```

print strings as data type string from its array of pointers

```
address 0x7ffc7d9f40f0 value test_1
address 0x7ffc7d9f4110 value test_2
address 0x7ffc7d9f4130 value test_3
address 0x7ffc7d9f4150 value test_4 // each string needs 20 bytes
```



Addresses of each character in the two strings "logic", "design"

```
const char *s[4]={a[0]"logic",a[1]"design",a[2]"c++",a[3]"computer"};  
for(int i=0;i<2;i++)  
    for(int j=0;j<strlen(s[i]);j++)  
        cout<<" address "<<(int *)(&s[i][j])<<" value "<< s[i][j]<<endl;
```

### Output

```
address 0x405001 value logic  
address 0x405002 value ogic  
address 0x405003 value gic  
address 0x405004 value ic  
address 0x405005 value c  
address 0x405007 value design  
address 0x405008 value esign  
address 0x405009 value sign  
address 0x40500a value ign  
address 0x40500b value gn  
address 0x40500c value n
```

$s[i]+n$   $\Rightarrow$  string start from nth char  
 $*s[i]+n$   $\Rightarrow$  nth char

## Example of new and delete with arrays

### Example 11

```
int main()
{int *p;      p= new int[10];
    cout<<" pointers and values of the array  assigned by new  " <<endl;
        for(int i=0;i<4;i++){ *(p+i)=2*i;
    cout<<" element "<<i<<" pointer "<<(p+i)<<" value "<< *(p+i)<<endl;  }
    delete []p;
    for(int i=0;i<4;i++)
        cout<<*(p+i)<<endl;
        int x[ ]= {10,20,30,40,50};

    p=x;
    cout<<endl<<endl;
    cout<<" pointers of array x and  values of array x " <<endl;
    for(int i=0;i<5;i++) cout<<" pointer "<<(p+i)<<" value " <<*(x+i) <<endl;
    delete p;
    cout<<" value of array x after delete pointer p " <<endl;
    for(int i=0;i<5;i++)
        cout<<" values of array "<<*(x+i) <<" " <<endl; cout<<endl;
    cout<<" pointer after delete ";cout<<p;}
```

## Output

pointers and values of the array assigned by new

element 0 pointer 0xda1910 value 0

element 1 pointer 0xda1914 value 2

element 2 pointer 0xda1918 value 4

element 3 pointer 0xda191c value 6

14292880 // error addresses

0

14287184

0

pointers of array x and values of array x

pointer 0x61fdf0 value 10

pointer 0x61fdf4 value 20

pointer 0x61fdf8 value 30

pointer 0x61fdfc value 40

pointer 0x61fe00 value 50

هالام

cant delete dynamic allocated  
pointer pointing to a variable or  
array

Deleting pointer to static array and then trying to print it will give run time error

## Pointers to Structure Objects

We can use pointers to **point to an object** of structure. When dealing with pointer objects, its a standard to use **arrow operator -> instead of '.' operator**

### Exampel 12 :

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

```
struct st { int a; char ch;};
```

```
int main{   st obj;           // object of structure st
            st *stobj = &obj; // stobi is a pointer to object obj
            stobj->a = 5;     stobj->ch = '#'; // operator -> used with member of object
                               // stobi ->a is equivalent to obi.a
            cout<< stobj->a<<" , "<< stobj->ch;   return 0; }
```

**OUTPUT**    5 , #

- In the above code, we have declared a pointer stobj of type 'struct st'. Now since the pointer type is a structure, so the address it points to has to be of a 'struct st' type variable(which in this case is 'obj').
- Structure elements are accessed using pointer variable 'stobj' with -> operator.
- We can also use 'obj' to access the structure elements.

Structure elements can be accessed using 'obj' as follows

```
Struct st{ int a; char ch;};  
main ( )  
{ st obj;  
  obj.a = 10;  obj.ch = '&';  
  cout<< obj .a<<" , "<< stobj. ch;  
  return 0; }
```

### **Comments**

- obj is an object of structure st
- We use members of the object obj using dot (.) operator, while in pointers we use -> operator

### **Output**

10 &

### Ex: 13 Another example of pointers and structures

```
struct Coordinate {    int x;    int y;};
```

```
float getDistance(struct Coordinate *X, struct Coordinate *Y) {
```

```
    int x_diff = X->x - Y->x;
```

```
    int y_diff = X->y - Y->y;
```

```
    float distance = sqrt((x_diff * x_diff) + (y_diff * y_diff));    return distance; }
```

```
int main() {
```

```
    struct Coordinate a,b;
```

```
    a.x = 5, a.y = 6;
```

```
    b.x = 4, b.y = 7;
```

```
    float distance = getDistance(&a, &b);
```

```
    cout<<"Distance between points <<endl<<"( " a.x<<" , "<< a.y<<" ) "<<" ("<< b.x<<" , "<< b.y<<" ) "<<endl;
```

```
<<" distance ="<< distance;    return 0; }
```

pointers of type coordinate

optional

address of members

## Output

Distance between points

(5,6) ( 4,7)

Distance= 1.414

## **Example of structures and pointers**

Write a program that uses the structure student which contains the student number, name, his scores in m subjects and the average score. Then the program creates n objects of the structure student.

The program uses the functions:

Fun\_input( ) to read data for certain student

Average( ) to get average score of certain student.

The main function access the above two functions through pointers to the structure, read data of n students and get their average score.



## Functions and pointers

We can **pass pointers to the function** or **return a pointer from functions** as the following examples:

### **Example 14**

```
float value(float *p1,float *p2)
{ return *p1**p2;  }

main( )
{ float k1,k2; float *p1=&k1; float *p2=&k2;
  k1=10; k2=20;
  cout<<" output of function = "<<value(p1,p2);}
```

**Here we send two pointers p1 and p2 to the function**

**output:**

**Output of function = 200**

## **Example 15**

```
void cub_1(int &n) // call by reference function  
{n= n*n*n;}
```

```
void cub_2(int *n) // function argument is an integer pointer  
{ *n=(*n)*(*n)*(*n); }
```

```
main()  
{ int num=5; // call by reference  
  cub_1(num); cout<<" value of num after cub_1( ) "<<num<<endl;  
  // send pointer to the function  
int k=10; cub_2(&k);cout<<" value of k after cub_2( ) "<<k<<endl;
```

In function cub\_1, it was call by reference, so value of num will be changed after calling it.

In function cub\_2, we send pointer to the function, so the value stored in this location will also be changed

## **Output**

value of num after cub\_1( ) 125

value of k after cub\_2( ) 1000

## // function has pointer type (bad pointers)

### Example 16

// function returns pointer

```
int *fun( )
```

```
{int *x; int y=10; x=&y; return x; }
```

// function needs a pointer as its argument

```
int fun_2(int *x ) { *x=30; return *x; }
```

```
int main()
```

```
{int *pp= new int;
```

```
*pp=5; cout<<" value at address pp= "<<*pp<<" stored at address = "<<pp<<endl;
```

// fun( ) returns the address of a local variable

```
pp=fun(); cout<<" new address from fun( ) = "<<pp<<endl<<endl;
```

// send pointer to fun\_2( )

```
*pp=80; *pp=fun_2(pp);
```

```
cout<<" pointer pp will not be changed, address after calling fun_2( )= "<<pp<<endl;
```

```
cout<<" returned value will be changed "<<*pp<<endl; }
```

## Output

value at address pp= 5 stored at address = 0xeb1910  
new address from fun( ) = 0x61fdd4

pointer pp will not be changed, address after calling fun\_2( )= 0x61fdd4  
returned value will be changed 0

//output of \*pp=0 because pp was a pointer to returned address from fun( )

Let's the code will be

```
int y, *p1; p1=&y; *p1= fun_2(p1);
```

Lecture\_8

Prof. Neamat Abdel Kader

### The output:

pointer p1 after calling fun\_2( )= 0x7ffc30ae183c  
returned value will be 30

## Example 17

```
int * fun( )  
{int y, *p;  y=100; p=&y;  
  return p;  }
```

static -> save value stored in function even if function terminated

```
int * fun_2( int n)  
{ static int x[10]; // if not use static, warning: address of local variable 'x' returned  
  cout<<"array in the function fun_2"<<endl;  
  for(int i=0; i<n;i++)  
  {  x[i]=2*i;  cout<<" array "<<x[i]<<endl;}  
  return x;  }
```

```
main( )  
{  int *p; p=fun( );  
  cout<<" pointer "<<p<<" value "<<*p<<endl;  
  int n; cin>>n;  
  p=fun_2(n);  
  cout<<" array after calling function fun_2 "<<endl;  
  for(int i=0; i<n;i++)  
  cout<<" array value "<<*(p+i)<<endl;}
```

## The output if n= 6

supposed give garbage value bec its not static

pointer 0x28feb0 value 100

array in the function fun\_2

array 0

array 2

array 4

array 6

array 8

array 10

array after calling function fun\_2

array value 0

array value 2

array value 4

array value 6

array value 8

array value 10

## Return a pointer from function

C++ allows a function to return a pointer to local variable, static variable and dynamically allocated memory as well. The following example shows that we can return a pointer from function

In the first function `fun( )`, a pointer of an integer is returned from the function, while the second function `fun_2( )` returns a pointer to an array