**Understanding pointers in c**

Pointer is a variable just like other variables of c but only difference is unlike the other variable it stores the memory address of any other variables of c. This variable may be type of int, char, array, structure, function or any other pointers. For examples:

(1) Pointer p which is storing memory address of a int type variable:

int i=50;

int \*p=&i;

(2) Pointer p which is storing memory address of an array:

int arr[20];

int (\*p)[20]=&arr;

(3) Pointer p which is storing memory address of a function:

char display(void);

char(\*p)(void)=&display;

(4) Pointer p which is storing memory address of struct type variable:

struct abc{

int a;

float b;

}var;

struct abc \*p=&var;

**What is pointer in c programming?**

Pointer is a user defined data type which creates special types of variables which can hold the address of primitive data type like **char**, **int**, **float**, **double** or user defined data type like function, pointer etc. or derived data type like array, structure, **union**, **enum**.

Examples:

**int** \*ptr;

**int** (\*ptr)();

**int** (\*ptr)[2];

**In c programming every variable keeps two type of value.**

1. Contain of variable or value of variable.

2. Address of variable where it has stored in the memory.

**(1) Meaning of following simple pointer declaration and definition:**

**int** a=5;

**int** \* ptr;

ptr=&a;

Explanation:

**About variable a:**

1. Name of variable : a

2. Value of variable which it keeps: 5

3. Address where it has stored in memory : 1025 (assume)

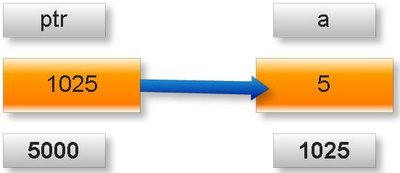
**About variable ptr:**

4. Name of variable : ptr

5. Value of variable which it keeps: 1025

6. Address where it has stored in memory : 5000 (assume)

Pictorial representation:



**Note:** A variable where it will be stored in memory is decided by operating system. We cannot guess at which location a particular variable will be stored in memory.

**(2) Meaning of following pointer declaration and definition:**

**int** a=50;

**int** \*ptr1;

**int** \*\*ptr2;

ptr1=&a;

ptr2=&pt1;

Explanation:

**About variable a:**

1. Name of variable : a

2. Value of variable which it keeps: 50

3. Address where it has stored in memory : 5000 (assume)

**About variable ptr1:**

4. Name of variable : ptr1

5. Value of variable which it keeps: 5000

6. Address where it has stored in memory : 9000 (assume)

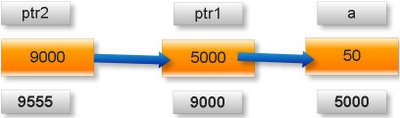
**About variable ptr2:**

7. Name of variable : ptr2

8. Value of variable which it keeps: 9000

9. Address where it has stored in memory : 9555 (assume)

Pictorial representation of above pointer declaration and definition:



**Note:**

\* is know as indirection operator which gives content of any variable.

& is know as reference operator which gives address where variable has stored in memory.

**Cancellation rule of above two operators:**

\* and & operators always cancel to each other. i.e.

**\*&p=p**

But it is not right to write:

&\*p=p

Simple example:

**What will be output of following c program?**

**void** main(){

**int** x=25;

**int** \*ptr=&x; //statement one

**int** \*\*temp=&ptr; //statement two

printf(“%d %d %d”,x,\*ptr,\*\*temp);

}

Output: 25 25 25

Explanation:

As we know value of variable x is 25.

\*ptr= \*(&x) //from statement one

=\*&x

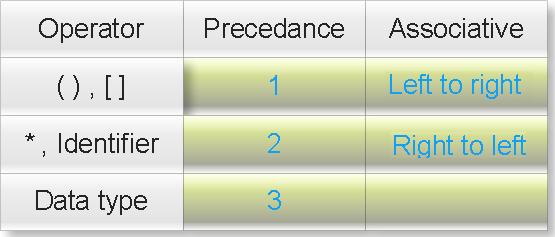
=x //using cancellation rule

=25

\*\*temp= \*\*(&ptr)=\*(\*&ptr)=\*ptr=\*(&x)=\*&x=x=25

**How to read complex pointers in C Programming?**

**Rule 1.** Assign the priority to the pointer declaration considering precedence and associative according to following table.



Where

**():** This operator behaves as bracket operator or function operator.

**[]:** This operator behaves as array subscription operator.

**\***: This operator behaves as pointer operator not as multiplication operator.

**Identifier**: It is not an operator but it is name of pointer variable. You will always find the first priority will be assigned to the name of pointer.

**Data type**: It is also not an operator. Data types also includes modifier (like signed int, long double etc.)

What is meaning of priority of operator? Click me.

What is meaning of associative of operator? Click me.

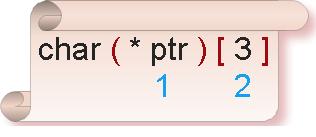
You will understand it better by examples:

**(1) How to read following pointer?**

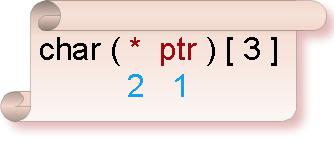
char (\* ptr)[3]

Answer:

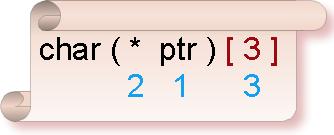
**Step 1:** () and [] enjoys equal precedence. So rule of associative will decide the priority. Its associative is left to right So first priority goes to ().



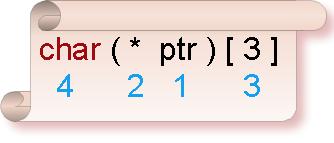
**Step 2:** Inside the bracket \* and ptr enjoy equal precedence. From rule of associative (right to left) first priority goes to ptr and second priority goes to \*.



**Step3:** Assign third priority to [].



**Step4:** Since data type enjoys least priority so assign fourth priority to char.



Now read it following manner:

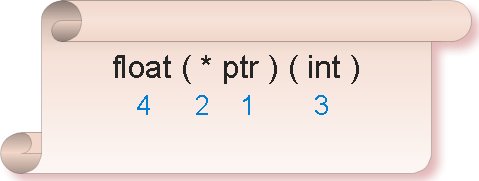
**ptr** is **pointer** to such one dimensional **array** of size three which content **char** type data.

**(2) How to read following pointer?**

float (\* ptr)(int)

Answer:

Assign the priority considering precedence and associative.



Now read it following manner:

**ptr** is **pointer** to such **function** whose parameter is int type data and return type is **float** type data.

**Rule 2:** Assign the priority of each function parameter separately and read it also separately.

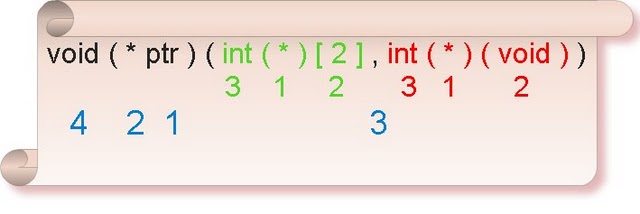
Understand it through following example.

**(3) How to read following pointer?**

void (\*ptr)(int (\*)[2],int (\*) void))

Answer:

Assign the priority considering rule of precedence and associative.



Now read it following manner:

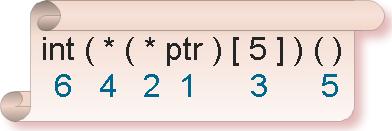
**ptr** is **pointer** to such **function** which first parameter is **pointer** to one dimensional **array** of size two which content **int** type data and second parameter is **pointer** to such **function** which parameter is void and return type is **int** data type and return type is **void**.

**(4) How to read following pointer?**

int ( \* ( \* ptr ) [ 5 ] ) ( )

Answer:

Assign the priority considering rule of precedence and associative.



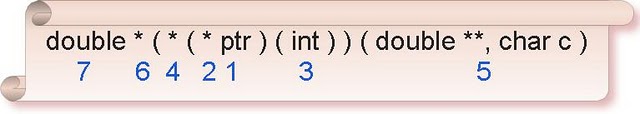
Now read it following manner:

**ptr** is **pointer** to such **array** of size five which content are **pointer** to such **function** which parameter is void and return type is **int** type data.

**(5) How to read following pointer?**

double\*(\*(\*ptr)(int))(double \*\*,char c)

Answer:



Assign the priority considering rule of precedence and associative.

Now read it following manner:

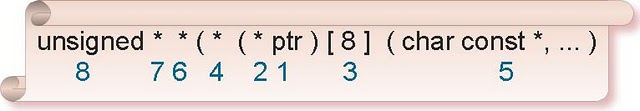
**ptr** is **pointer** to **function** which parameter is int type data and return type is **pointer** to **function** which first parameter is pointer to pointer of double data type and second parameter is char type data type and return type is **pointer** to **double** data type.

**(6) How to read following pointer?**

unsigned int\*\*(\*(\*ptr)[8](char const \*, ...)

Answer:

Assign the priority considering rule of precedence and associative.



Now read it following manner:

**ptr** is **pointer** to **array** of size eight and content of array is **pointer** to **function** which first parameter is pointer to character constant and second parameter is variable number of arguments and return type is **pointer** to **pointer** of **unsigned** int data type.

**Arithmetic operation with pointer in c programming**

**Rule 1:**

Address + Number= Address

Address - Number= Address

Address++ = Address

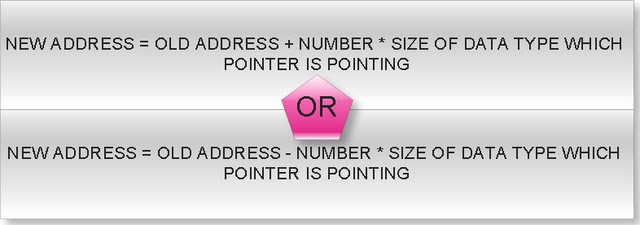
Address-- = Address

++Address = Address

--Address = Address

If we will add or subtract a number from an address result will also be an address.

New address will be:



**(1)What will be output of following c program?**

**void** **main**(){

**int** \*ptr=( **int** \*)1000;

ptr=ptr+1;

printf(" %u",ptr);

}

Output: 1002

**(2)What will be output of following c program?**

**void** **main**(){

**double** \*p=(**double** \*)1000;

p=p+3;

printf(" %u",p);

}

Output: 1024

**(3)What will be output of following c program?**

**void** **main**(){

**float** array[5]={1.1f,2.2f,3.3f};

**float**(\*ptr)[5];

ptr=&array;

printf("%u",ptr);

ptr=ptr+1;

printf(" %u",ptr);

}

Output: 1000 1020

**(4)What will be output of following c program?**

**typedef** **struct** abc{

**int** far\*a;

**double** b;

**unsigned** **char** c;

}ABC;

**void** **main**(){

ABC \*ptr=(ABC \*)1000;

ptr=ptr+2;

printf(" %u",ptr);

}

Output: 1026

**(5)What will be output of following c program?**

**typedef** **union** abc{

**char** near\*a;

**long** **double** d;

**unsigned** **int** i;

}ABC;

**void** **main**(){

ABC \*ptr=(ABC \*)1000;

ptr=ptr-4;

printf(" %u",ptr);

}

Output: 960

**(6)What will be output of following c program?**

**float** \* **display**(**int**,**int**);

**int** max=5;

**void** **main**(){

**float** \*(\*ptr)(**int**,**int**);

ptr=display;

(\*ptr)(2,2);

printf("%u",ptr);

ptr=ptr+1;

printf(" %u",ptr);

}

**float** \* **display**(**int** x,**int** y){

**float** f;

f=x+y+max;

**return** &f;

}

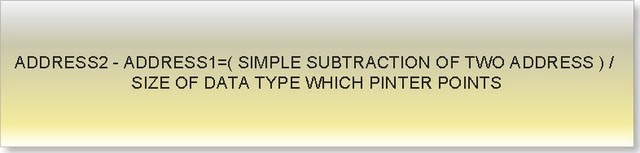
Output: Compiler error

**Rule 2:**

Address - Address=Number

If you will subtract two pointers result will be a number but number will not simple mathematical subtraction of two addresses but it follow following rule:

If two pointers are of same type then:



**Consider following example:**

**void** **main**(){

**int** \*p=(**int** \*)1000;

**int** \*temp;

temp=p;

p=p+2;

printf("%u %u\n",temp,p);

printf("difference= %d",p-temp);

}

Output: 1000 1004

Difference= 2

Explanation:

Here two pointer p and temp are of same type and both are pointing to **int** data type varaible.

p-temp = (1004-1000)/**sizeof**(**int**)

=4/2

=2

**(1)What will be output of following c program?**

**void** main(){

**float** \*p=(**float** \*)1000;

**float** \*q=(**float** \*)2000;

printf("Difference= %d",q-p);

}

Output: Difference= 250

Explanation:

q-p=(2000-1000)/sizeof(float)

=1000/4

=250

**(2)What will be output of following c program?**

**struct** abc{

**signed** **char** c;

**short** **int** i;

**long** **double** l;

};

**void** **main**(){

**struct** abc \*p,\*q;

p=(**struct** abc \*)1000;

q=(**struct** abc \*)2000;

printf("Difference= %d",q-p);

}

Output: Difference= 76

Explanation:

q-p=(2000-1000)/**sizeof**(**struct** abc)

=1000/(1+2+10)

=1000/13

=76

**(3)What will be output of following c program?**

**typedef** **union** xxx{

**char** far \* c;

**const** **volatile** i;

**long** **int** l;

}XXX;

**void** **main**(){

XXX \*p,\*q;

p=(XXX \*)1000;

q=(XXX \*)2000;

printf("Difference= %d",q-p);

}

Output: Difference= 250

Explanation:

q-p=(2000-100)/max(4,2,4)

=1000/4

=250

**(4)What will be output of following c program?**

**void** main(){

**const** **volatile** array[4]={0};

**const** **volatile**(\*p)[4]=&array;

**const** **volatile**(\*q)[4]=&array;

q++;

q++;

printf("%u %u\n",p,q);

printf("Difference= %d",q-p);

}

Output: 1000 1016 (assume)

Difference= 2

Explanation:

q-p=(1016-1000)/**sizeof**(**const** **volatile**)

= 16/ (2\*4)

=2

**Rule 3:**

Address + Address=Illegal

Address \* Address=Illegal

Address / Address=Illegal

Address % Address=Illegal

**(q)What will be output of following c program?**

**void** main( )

{

**int** i=5;

**int** \*p=&i;

**int** \*q=(**int** \*)2;

printf("%d",p+q);

}

**Output:** Compiler error

**Rule 4:** We can use relation operator and condition operator between two pointers.

a. If two pointers are near pointer it will compare only its offset address.

**(q)What will be output of following c program?**

**void** main( )

{

**int** near\*p=(**int** near\*)0x0A0005555;

**int** near\*q=(**int** near\*)0x0A2115555;

**if**(p==q)

printf("Equql");

**else**

printf("Not equal");

}

Output: Equal

b. If two pointers are far pointer it will compare both offset and segment address.

**(q)What will be output of following c program?**

**void** main(){

**int** far\*p=(**int** far\*)0x0A0005555;

**int** far\*q=(**int** far\*)0x0A2115555;

**if**(p==q)

printf("Equal");

**else**

printf("Not equal");

}

Output: Not equal

c. If two pointers are huge pointer it will first normalize into the 20 bit actual physical address and compare to its physical address.

**(q)What will be output of following c program?**

**void** main(){

**int** huge\*p=(**int** huge\*)0x0A0005555;

**int** huge\*q=(**int** huge\*)0x0A2113445;

**if**(p==q)

printf("Equql");

**else**

printf("Not equal");

}

Output: Equal

**Rule 5:** We can perform bitwise operation between two pointers like

Address & Address=Illegal

Address | Address=Illegal

Address ^ Address=Illegal

~Address=Illegal

**(q)What will be output of following c program?**

**void** main(){

**int** i=5,j=10;

**int** \*p=&i;

**int** \*q=&j;

printf("%d",p|q);

}

Output: Compiler error

**Rule 6:** We can find size of a pointer using sizeof operator.

**(q)What will be output of following c program?**

**void** main(){

**int** near\*far\*huge\* p;

printf("%d",**sizeof**(p));

printf(" %d",**sizeof**(\*p));

printf(" %d",**sizeof**(\*\*p));

}

Output: 4 4 2

**Complex pointers in c programming**

**(1) Pointer to function**

**(2) Pointer to array**

**a. Pointer to array of integer**

**a. Pointer to array of function**

**b. Pointer to array of string**

**c. Pointer to array of character**

**d. Pointer to array of structure**

**e. Pointer to array of union**

**f. Pointer to array of array**

**g. Pointer to two dimensional array**

**h. Pointer to three dimensional array**

**i.**

**j. Pointer to array of pointer to string**

**(3) Pointer to structure**

**(4) Pointer to union**

**(5) Multilevel pointers**

**Pointer to array of function in c**

**Note:** array of function means array which content is address of function.

**(q) What will be output if you will execute following code?**

**#include**"conio.h"

**int** display();

**int**(\*array[3])();

**int**(\*(\*ptr)[3])();

**void** **main**(){

array[0]=display();

array[1]=getch();

ptr=&array;

printf("%d",(\*\*ptr)());

(\*(\*ptr+1))();

}

**int** **display**(){

**int** x=5;

**return** x++;

}

Output: 5

Explanation:

In this example:

array []: It is array of pointer to such function which parameter is void and return type is int data type.

ptr: It is pointer to array which contents are pointer to such function which parameter is void and return type is int type data.

**(\*\*ptr)()** = (\*\* (&array)) () //ptr=&array

= (\*array) () // from rule \*&p=p

=array [0] () //from rule \*(p+i)=p[i]

=display () //array[0]=display

**(\*(\*ptr+1))()** =(\*(\*&array+1))() //ptr=&array

=\*(array+1) () // from rule \*&p=p

=array [1] () //from rule \*(p+i)=p[i]

=getch () //array[1]=getch

**Pointer to array of string in c programming**

**(q) What will be output if you will execute following code?**

**void** main(){

**char** \*array[4]={"c","c++","java","sql"};

**char** \*(\*ptr)[4]=&array;

printf("%s ",++(\*ptr)[2]);

}

Output: ava

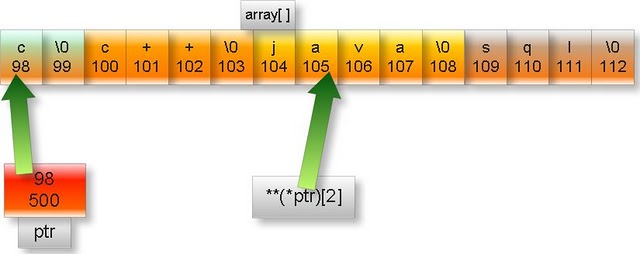
Explanation:

In this example

ptr: It is pointer to array of string of size 4.

array[4]: It is an array and its content are string.

Pictorial representation:



**Note:** In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

**++(\*ptr)[2]**

=++(\*&array)[2] //ptr=&array

=++array[2]

=++”java”

=”ava” //Since ptr is character pointer so it

// will increment only one byte

**Note:** %s is used to print stream of characters up to null (\0) character.

**Pointer to structure in c programming**

**(q) What will be output if you will execute following code?**

**struct** address{

**char** \*name;

**char** street[10];

**int** pin;

}cus={"A.Kumar","H-2",456003},\*p=&cus;

**void** **main**()

{

printf("%s %s",p->name,(\*p).street);

}

Output: A.Kumar H-2

Explanation:

p is pointer to structure address.

-> and (\*). Both are same thing. These operators are used to access data member of structure by using structure’s pointer.

**Pointer to union in c programming**

**(q) What will be output if you will execute following code?**

**union** address{

**char** \*name;

**char** street[10];

**int** pin;

};

**void** **main**(){

**union** address emp,\*p;

emp.name="ja\0pan";

p=&emp;

printf("%s %s",p->name,(\*p).name);

}

Output: ja ja

Explanation:

p is pointer to union address.

-> and (\*). Both are same thing. These operators are used to access data member of union by using union’s pointer.

%s is used to print the string up to null character i.e. ‘\0’

**Multilevel pointers in c programming**

**(1) What will be output if you will execute following code?**

**void** main(){

**int** s=2,\*r=&s,\*\*q=&r,\*\*\*p=&q;

printf("%d",p[0][0][0]);

}

Output: 2

Explanation:

As we know p[i] =\*(p+i)

So,

**P[0][0][0]**=\*(p[0][0]+0)=\*\*p[0]=\*\*\*p

Another rule is: \*&i=i

So,

\*\*\*p=\*\*\* (&q) =\*\*q=\*\* (&r) =\*r=\*(&s) =s=2

**(2) What will be output if you will execute following code?**

**#define** **int** **int**\*

**void** **main**(){

**int** \*p,q;

p=(**int** \*)5;

q=10;

printf("%d",q+p);

}

Output: 25

Explanation: If you will see intermediate file you will find following code:

**void** **main**(){

**int** \*\*p,q;

p=(**int** \*\*)5;

q=10;

printf("%d",q+p);

}

Here q pointer and p is a number.

In c

Address + number = Address

So,

New address = old address + number \* Size of data type to which pointer is pointing.

= 5 + 10 \* sizeof (\*int)

= 5+10\*2 = 25.

**Note.** We are assuming default pointer is near. Actually it depends upon memory model.

**Pointer to array of pointer to string in c programming**

**(q) What will be output if you will execute following code?**

**void** main(){

**static** **char** \*s[3]={"math","phy","che"};

**typedef** **char** \*( \*ppp)[3];

**static** ppp p1=&s,p2=&s,p3=&s;

**char** \* (\*(\*array[3]))[3]={&p1,&p2,&p3};

**char** \* (\*(\*(\*ptr)[3]))[3]=&array;

p2+=1;

p3+=2;

printf("%s",(\*\*\*ptr[0])[2]);

}

Output: che

Explanation:

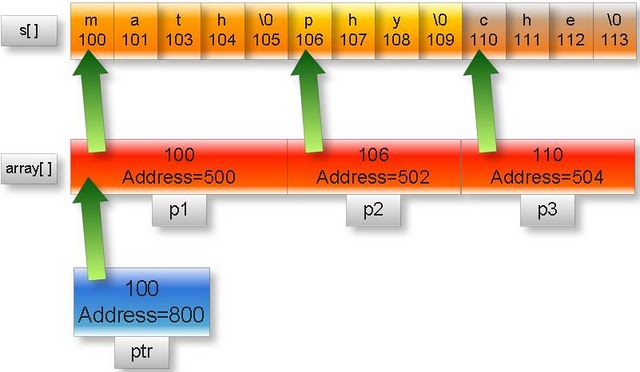
Here

ptr: is pointer to array of pointer to string.

P1, p2, p3: are pointers to array of string.

array[3]: is array which contain pointer to array of string.

Pictorial representation:



**Note:** In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

As we know p[i]=\*(p+i)

**(\*\*\*ptr[0])[2]**=(\*(\*\*\*ptr+0))[2]=(\*\*\*ptr)[2]

=(\*\*\*(&array))[2] //ptr=&array

=(\*\*array)[2] //From rule \*&p=p

=(\*\*(&p1))[2] //array=&p1

=(\*p1)[2]

=(\*&s)[2] //p1=&s

=s[2]=”che”

**Pointer to three dimensional array in c programming**

**void** main(){

**const** array[2][3][3]={0,1,2,3,4,5,6,7,8,9,10,11,12};

**int** **const** (\*ptr)[2][3][3]=&array;

printf("%d ",\*(\*(\*ptr)[1]+2));

}

Output: 11

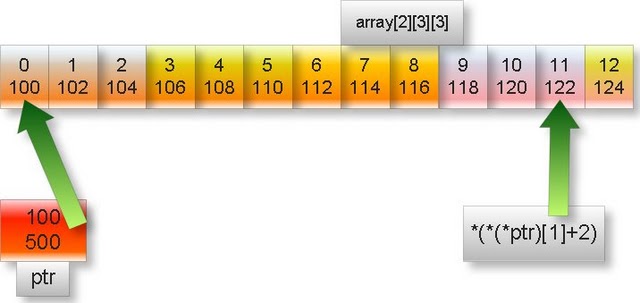
Explanation:

In this example:

array [2][3][3]:It is three dimensional array and its content are constant integers.

ptr: It is pointer to such three dimensional array whose content are constant integer.

Pictorial representation:



**Note:** In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

**\*(\*(\*ptr) [1] +2)**

=\*(\*(\*&array) [1] +2)

=\*(\*array [1] +2)

=\*(array [1] [0] +2)

=array [1] [0] [2]

I.e. array element at the 1\*(3\*3) +0(3) + 2=11th position starting from zero which is 11.

**Pointer to two dimensional array in c programming**

**(q) What will be output if you will execute following code?**

**void** main(){

**long** array[][3]={7l,14l,21l,28l,35l,42l};

**long** **int** (\*ptr)[2][3]=&array;

printf("%li ",-0[1[0[ptr]]]);

}

Output: -28

Explanation:

**-0[1[0[ptr]]]**

=-1[0[ptr]][0] //From rule array[i]=i[array]

=-0[ptr][1][0]

=-ptr [0] [1] [0]

=-\*ptr [0] [1] //From rule array[i]=\*(array+i)

=-\*(&array) [0] [1]

=-(&array) [0] [1][0]

=-(\*&array)[1][0] //From rule \*&p=p

=-array[1][0]

array[1][0] means 1\*(3)+ 0 = 3rd element of array starting from zero i.e. 28

**sorting of array using pointer in c**

void main()

{

int  i,j,temp1,temp2;

int arr[8]={5,3,0,2,12,1,33,2};

int \*ptr;

for(i=0;i<7;i++)

{

 for(j=0;j<7-i;j++)

{

if(\*(arr+j)>\*(arr+j+1))

{

 ptr=arr+j;

 temp1=\*ptr++;

 temp2=\*ptr;

 \*ptr--=temp1;

 \*ptr=temp2;

}

}

}

clrscr();

for(i=0;i<8;i++)

 printf(" %d",arr[i]);

getch();

}

Output: 0 1 2 2 3 5 12 33

**Pointer to array of array**

**(q) What will be output if you will execute following code?**

**void** main(){

**static** **float** farray[][3]={0.0f,1.0f,2.0f,3.0f,4.0f,5.0f,6.0f,7.0f,8.0f};

**float** (\*array[3])[3]={&farray[0],&farray[1],&farray[2]};

**float** (\*(\*ptr)[])[3]=&array;

printf("%f ",2[(\*(\*\*ptr+1))]);

}

Output: 5.000000

Explanation:

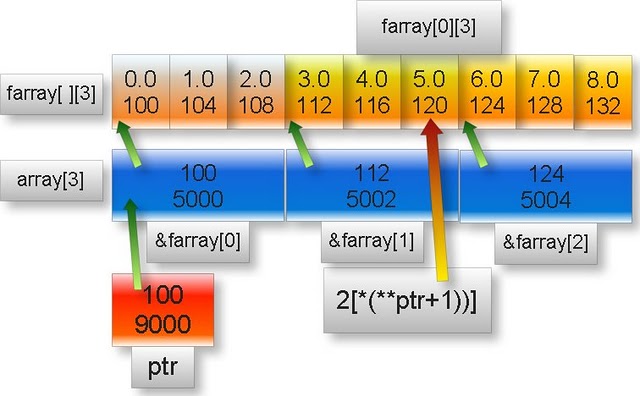
In this example:

farray [][3]: It is two dimension array and its content are float constants.

array [3]:It is one dimension array and its content are address of such one dimension array which content are float constant.

ptr: It is pointer to one dimension array which content are address of such one dimension array which content are float constant.

Pictorial representation:



**Note:** In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

**2[(\*(\*\*ptr+1))]**

= (\*(\*\*ptr+1)) [2]

= (\*(\*\*&array+1)) [2]

= (\*(\*array+1)) [2]

= (\*(array [0] +1)) [2]

= (\*(&farray [0] +1)) [2]

=&farray [0] [1] [2]

=\*&farray [1] [2]

=farray [1] [2]

It is 1\*(3) +2=5th element of farray starting from zero which is 5.0f

**Pointer to array of union in c programming**

**(q) What will be output if you will execute following code?**

**union** emp{

**char** \*name;

**int** id;

};

**void** **main**(){

**static** **union** emp e1={"A"},e2={"B"},e3={"C"};

**union** emp(\*array[])={&e1,&e2,&e3};

**union** emp(\*(\*ptr)[3])=&array;

printf("%s ",(\*(\*ptr+2))->name);

}

Output: C

Explanation:

In this example:

e1, e2, e3: They are variables of union emp.

array []:It is one dimensional array of size thee and its content are address of union emp.

ptr: It is pointer to array of union.

**(\*(\*ptr+2))->name**

=(\*(\*&array+2))->name //ptr=&array

=(\*(array+2))->name //from rule \*&p=p

=array[2]->name //from rule \*(p+i)=p[i]

=(&e3)->name //array[2]=&e3

=\*(&e3).name //from rule ->= (\*).

=e3.name //from rule \*&p=p

=”C”

**Pointer to array of structure in c programming**

**(q) What will be output if you will execute following code?**

**struct** emp{

**char** \*name;

**int** id;

};

**void** **main**(){

**static** **struct** emp e1={"A",1},e2={"B",2},e3={"C",3};

**struct** emp(\*array[])={&e1,&e2,&e3};

**struct** emp(\*(\*ptr)[3])=&array;

printf("%s %d",(\*\*(\*ptr+1)).name,(\*(\*ptr+1))->id);

}

Output: B 2

Explanation:

**(\*\*(\*ptr+1)).name**

=(\*\*(\*&array+1)).name //ptr=&array

=(\*\*(array+1)).name //from rule \*&p =p

=(\*array[1]).name //from rule \*(p+i)=p[i]

=(\*&e2).name //array[1]=&e2

=e2.name=”B” //from rule \*&p =p

**(\*(\*ptr+1))->id**

=(\*\*(\*ptr+1)).id //from rule -> = (\*).

=e2.id=2

**Pointer to array of character in c**

**(q) What will be output if you will execute following code?**

**char** display(**char** (\*)[]);

**void** **main**(){

**char** c;

**char** character[]={65,66,67,68};

**char** (\*ptr)[]=&character;

c=display(ptr);

printf("%c",c);

}

**char** **display**(**char** (\*s)[]){

\*\*s+=2;

**return** \*\*s;

}

Output: C

Explanation: Here function display is passing pointer to array of characters and returning **char** data type.

**\*\*s+=2**

=>\*\*s=\*\*s+2

=>\*\*ptr=\*\*ptr+2 //s=ptr

=>\*\*&character= \*\*&character+2 //ptr=&character

=>\*character=\*character+2 //from rule \*&p =p

=>character[0]=character[0]+2 //from rule \*(p+i)=p[i]

=>character [0] =67

\*\*s=character [0] =67

**Note:** ASCII value of ‘C’ is 67

**Pointer to array of integer in c**

**(2) Pointer to array**

**(1) What will be output if you will execute following code?**

**void** **main**(){

**static** **int** i,j,k;

**int** \*(\*ptr)[];

**int** \*array[3]={&i,&j,&k};

ptr=&array;

j=i+++k+10;

++(\*\*ptr);

printf("%d",\*\*\*ptr);

}

Output: 10

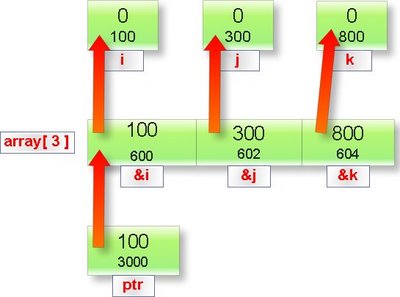
Explanation:

In this example:

array []: It is array of size three and its content are address of integer.

ptr: It is pointer to array which content are address of integer.

Pictorial representation above declaration:



**Note:** In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

**j**=i+++k+10

=i++ + k+10

=0 +0 +10=10

**\*\*\*ptr** = \*\*\* (&array) //ptr=&array

= \*\*array //From rule \*&p=p

//From rule array [0] =\*(array+0) and ++ (\*\*ptr)

=\*array [1]

=\*&j

=j

=10

**(2) What will be output if you will execute following code?**

**void** **main**(){

**int** i,j,k;

**int** \*(\*ptr)[];

**int** \*array[3]={&i,&j,&k};

ptr=&array;

j=i+++k+10;

++(\*\*ptr);

printf("%d",\*\*\*ptr);

}

Output: Compiler error

Explanation: Address of auto variable cannot be member of an array.

**Generic pointer in c programming**

**5. Generic pointer:**

void pointer in c is known as generic pointer. Literal meaning of generic pointer is a pointer which can point type of data.

Example:

**void** \*ptr;

Here ptr is generic pointer.

**Important points about generic pointer in c?**

1. We cannot dereference generic pointer.

**#include** "malloc.h"

**void** **main**(){

**void** \*ptr;

printf("%d",\*ptr);

}

Output: Compiler error

2. We can find the size of generic pointer using sizeof operator.

**#include** "string.h"

**void** **main**(){

**void** \*ptr;

printf("%d",**sizeof**(ptr));

}

Output: 2

Explanation: Size of any type of near pointer in c is two byte.

3. Generic pointer can hold any type of pointers like char pointer, struct pointer, array of pointer etc without any typecasting.

Example:

**void** **main**(){

**char** c='A';

**int** i=4;

**void** \*p;

**char** \*q=&c;

**int** \*r=&i;

p=q;

printf("%c",\*(**char** \*)p);

p=r;

printf("%d",\*(**int** \*)p);

}

Output: A4

4. Any type of pointer can hold generic pointer without any typecasting.

5. Generic pointers are used when we want to return such pointer which is applicable to all types of pointers. For example return type of malloc function is generic pointer because it can dynamically allocate the memory space to stores integer, float, structure etc. hence we type cast its return type to appropriate pointer type.

Examples:

1.

**char** \*c;

c=(**char** \*)malloc(**sizeof**(**char**));

2.

**double** \*d;

d=(**double** \*)malloc(**sizeof**(**double**));

3.

Struct student{

**char** \*name;

**int** roll;

};

Struct student \*stu;

Stu=(**struct** student \*)malloc(**sizeof**(**struct** student));

**NULL pointer in c programming**

**4. NULL pointer:**

Literal meaning of NULL pointer is a pointer which is pointing to nothing. NULL pointer points the base address of segment.

Examples of NULL pointer:

1. **int** \*ptr=(**char** \*)0;

2. **float** \*ptr=(**float** \*)0;

3. **char** \*ptr=(**char** \*)0;

4. **double** \*ptr=(**double** \*)0;

5. **char** \*ptr=’\0’;

6. **int** \*ptr=NULL;

**(q) What is meaning of NULL?**

Answer:

NULL is macro constant which has been defined in the heard file stdio.h, alloc.h, mem.h, stddef.h and stdlib.h as

**#define** NULL 0

Examples:

**(1)What will be output of following c program?**

**#include** "stdio.h"

**void** main(){

**if**(!NULL)

printf("I know preprocessor");

**else**

printf("I don't know preprocessor");

}

Output: I know preprocessor

Explanation:

!NULL = !0 = 1

In if condition any non zero number mean true.

**(2)What will be output of following c program?**

**#include** "stdio.h"

**void** main(){

**int** i;

**static** **int** count;

**for**(i=NULL;i<=5;){

count++;

i+=2;

}

printf("%d",count);

}

Output: 3

**(3)What will be output of following c program?**

**#include** "stdio.h"

**void** **main**(){

**#ifndef** NULL

**#define** NULL 5

**#endif**

printf("%d",NULL+**sizeof**(NULL));

}

Output: 2

Explanation:

NULL+sizeof(NULL)

=0+sizeoof(0)

=0+2 //size of int data type is two byte.

**We cannot copy any thing in the NULL pointer.**

Example:

**(q)What will be output of following c program?**

**#include** "string.h"

**void** **main**(){

**char** \*str=NULL;

strcpy(str,"understanding pointer in c");

printf("%s",str);

}

Output: (null)

**Wild pointer in c programming language**

**2. Wild pointer:**

A pointer in c which has not been initialized is known as wild pointer.

Example:

**(q)What will be output of following c program?**

**void** **main**(){

**int** \*ptr;

printf("%u\n",ptr);

printf("%d",\*ptr);

}

Output: Any address

Garbage value

Here ptr is wild pointer because it has not been initialized.

There is difference between the NULL pointer and wild pointer. Null pointer points the base address of segment while wild pointer doesn’t point any specific memory location.

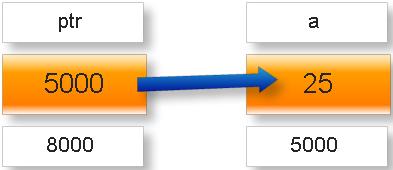
**Dangling pointer problem in c programming**

**Different types of pointers:**

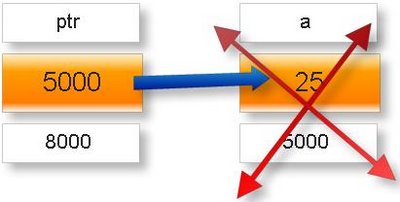
**1. Dangling pointer:**

If any pointer is pointing the memory address of any variable but after some variable has deleted from that memory location while pointer is still pointing such memory location. Such pointer is known as dangling pointer and this problem is known as dangling pointer problem.

Initially:



Later:



For example:

**(q)What will be output of following c program?**

**int** \***call**();

**void** **main**(){

**int** \*ptr;

ptr=call();

clrscr();

printf("%d",\*ptr);

}

**int** \* **call**(){

**int** x=25;

++x;

**return** &x;

}

Output: Garbage value

Explanation: variable x is local variable. Its scope and lifetime is within the function call hence after returning address of x variable x became dead and pointer is still pointing ptr is still pointing to that location.

**Solution of this problem:** Make the variable x is as static variable.

In other word we can say a pointer whose pointing object has been deleted is called dangling pointer.

**Near pointer in C programming**

In TURBO C there are three types of pointers. TURBO C works under DOS operating system which is based on 8085 microprocessor.

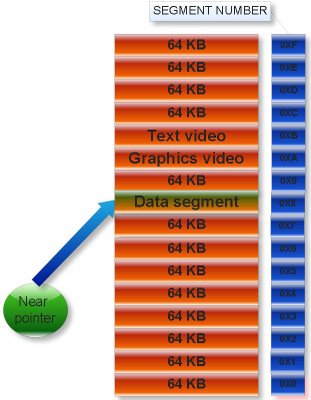
1. Near pointer

2. Far pointer

3. Huge pointer

**Near pointer:**

The pointer which can points only 64KB data segment or segment number 8 is known as near pointer.



That is near pointer cannot access beyond the data segment like graphics video memory, text video memory etc. Size of near pointer is two byte. With help keyword near, we can make any pointer as near pointer.

Examples:

(1)

**void** main(){

**int** x=25;

**int** near\* ptr;

ptr=&x;

printf(“%d”,**sizeof** ptr);

}

Output: 2

(2)

**void** main(){

**int** near\* near \* ptr;

printf(“%d”,**sizeof**(ptr),**sizeof**(\*ptr));

}

Output: 2 2

Explanation: Size of any type of near pointer is two byte.

Near pointer only hold 16 bit offset address. Offset address varies from 0000 to FFFF (in hexadecimal).

**Note:** In printf statement to print the offset address in hexadecimal, %p is used.

Example:

**void** **main**(){

**int** i=10;

**int** \*ptr=&i;

printf("%p",ptr);

}

Output: Offset address in hexadecimal number format.

%p is also used to print any number in hexadecimal number format.

Example:

**void** **main**(){

**int** a=12;

printf("%p",a);

}

Output: 000C

Explanation: Hexadecimal value of 12 is C.

**Consider the following two c program and analyze its output:**

(1)

**void** main(){

**int** near \* ptr=( **int** \*)0XFFFF;

ptr++;

ptr++;

printf(“%p”,ptr);

}

Output: 0003

(2)

**void** **main**(){

**int** i;

**char** near \*ptr=(**char** \*)0xFFFA;

**for**(i=0;i<=10;i++){

printf("%p \n",ptr);

ptr++;

}

}

Output:

FFFA

FFFB  
FFFC  
FFFD  
FFFE  
FFFF  
0000

0001

0002

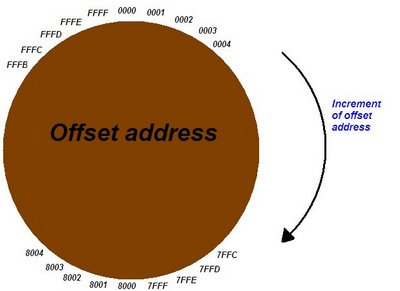
0003

0004

Explanation: When we increment or decrement the offset address from maximum and minimum value respectively then it repeats the same value in cyclic order. This property is known as cyclic nature of offset address.

**Cyclic property of offset address.**

If you increment the near pointer variable then move clockwise direction. If you decrement the near pointer then move anti clockwise direction.



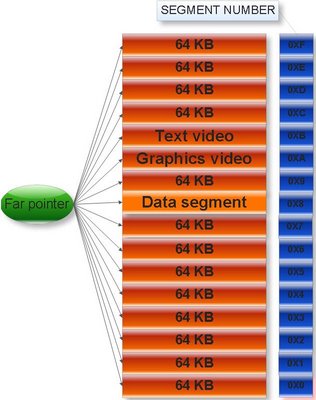
**What is default type of pointer in C?**

Answer: It depends upon memory model.

What is memory model in C?

**Far pointer in c programming**

The pointer which can point or access whole the residence memory of RAM i.e. which can access all 16 segments is known as far pointer.



**Far pointer:**

Size of far pointer is 4 byte or 32 bit.

Examples:

**(1) What will be output of following c program?**

**void** **main**(){

**int** x=10;

**int** far \*ptr;

ptr=&x;

printf("%d",**sizeof** ptr);

}

Output: 4

**(2)What will be output of following c program?**

**void** **main**(){

**int** far \*near\*ptr;

printf("%d %d",**sizeof**(ptr) ,**sizeof**(\*ptr));

}

Output: 4 2

Explanation: ptr is far pointer while \*ptr is near pointer.

**(3)What will be output of following c program?**

**void** **main**(){

**int** far \*p,far \*q;

printf("%d %d",**sizeof**(p) ,**sizeof**(q));

}

Output: 4 4

**First 16 bit stores:** Segment number

**Next 16 bit stores:** Offset address

What is segment number and offset address?

Example:

**void** **main**(){

**int** x=100;

**int** far \*ptr;

ptr=&x;

printf("%Fp",ptr);

}

Output: 8FD8:FFF4

Here 8FD8 is segment address and FFF4 is offset address in hexadecimal number format.

**Note:** %Fp is used for print offset and segment address of pointer in printf function in hexadecimal number format.

In the header file dos.h there are three macro functions to get the offset address and segment address from far pointer and vice versa.

1. **FP\_OFF():** To get offset address from far address.

2. **FP\_SEG():** To get segment address from far address.

3. **MK\_FP():** To make far address from segment and offset address.

Examples:

**(1)What will be output of following c program?**

**#include** "dos.h"

**void** **main**(){

**int** i=25;

**int** far\*ptr=&i;

printf("%X %X",FP\_SEG(ptr),FP\_OFF(ptr));

}

Output: Any segment and offset address in hexadecimal number format respectively.

**(2)What will be output of following c program?**

**#include** "dos.h"

**void** **main**(){

**int** i=25;

**int** far\*ptr=&i;

**unsigned** **int** s,o;

s=FP\_SEG(ptr);

o=FP\_OFF(ptr);

printf("%Fp",MK\_FP(s,o));

}

Output: 8FD9:FFF4 (Assume)

**Note:** We cannot guess what will be offset address, segment address and far address of any far pointer .These address are decided by operating system.

**Limitation of far pointer:**

We cannot change or modify the segment address of given far address by applying any arithmetic operation on it. That is by using arithmetic operator we cannot jump from one segment to other segment. If you will increment the far address beyond the maximum value of its offset address instead of incrementing segment address it will repeat its offset address in cyclic order.

Example:

**(q)What will be output of following c program?**

**void** **main**(){

**int** i;

**char** far \*ptr=(**char** \*)0xB800FFFA;

**for**(i=0;i<=10;i++){

printf("%Fp \n",ptr);

ptr++;

}

}

Output:

B800:FFFA

B800:FFFB

B800:FFFC

B800:FFFD

B800:FFFE

B800:FFFF

B800:0000

B800:0001

B800:0002

B800:0003

B800:0004

This property of far pointer is called cyclic nature of far pointer within same segment.

**Important points about far pointer:**

1. Far pointer compares both offset address and segment address with relational operators.

Examples:

**(1)What will be output of following c program?**

**void** **main**(){

**int** far \*p=(**int** \*)0X70230000;

**int** far \*q=(**int** \*)0XB0210000;

**if**(p==q)

printf("Both pointers are equal");

**else**

printf("Both pointers are not equal");

}

Output: Both pointers are not equal

**(2)What will be output of following c program?**

**void** **main**(){

**int** far \*p=(**int** \*)0X70230000;

**int** far \*q=(**int** \*)0XB0210000;

**int** near \*x,near\*y;

x=(**int** near \*)p;

y=(**int** near \*)q;

**if**(x==y)

printf("Both pointer are equal");

**else**

printf("Both pointer are not equal");

}

Output: Both pointers are equal

2. Far pointer doesn’t normalize.

What is normalization of pointer?