5.1 Introduction to Pointers:

- ✓ Pointe is a variable which stores the address of another variable, since pointer is also kind of a variable.
- ✓ Pointer itself will be stored at different memory location.
- ✓ Pointer variable prefix with *(Asterisk) symbol.

Syntax:

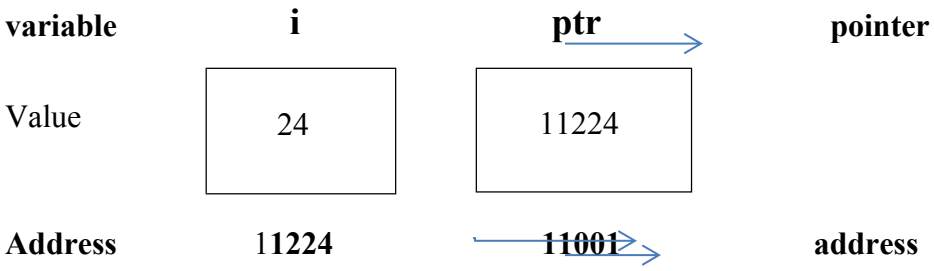
data_type * variable_name.

Example:

int *a; **or** int* a;

- ✓ In above declaration * placed in two different places ,the first one the * prefix with variable and second one * followed by datatype both are right you will follow any one these.

```
int i=24;
int *ptr; // declaration of pointer
ptr = &i;  // assigning address into ptr
```



- ✓ In above fig, i is a variable it holds the value 24 and address of I is 11224(let assume).
- ✓ Pointer always stores the address of variable so, ptr is a pointer , it stores the address of I (&i) i.e 11224.
- ✓ Pointer variable also stored at different location i.e 11001.

Variable name	Value of variable	Address of variable
i	24	11224
Ptr	11224	11001

Types of Pointers:

- ✓ According to data types generally pointer are two types :
  1) Typed pointers
  2) Un-typed pointer
- ✓ Typed pointers are nothing but integer pointers, character pointers, float pointers and double pointers.

char *	character pointer	→
int *	integer pointers	
float *	float pointer	one type of pointer cannot
double *	double pointer	points another type of data.

- ✓ Here one type of pointer cannot points another type of data that means integer pointer can points only integer data only (integer pointer stores the int variable address only). Character pointer points character data only, same as float and double.
- ✓ Un-typed pointer is nothing but generic pointers (void pointer). void pointer can point any type of data.

void * void pointers →

Size of pointers:

- ✓ The pointer occupy the same amount of memory for the all the data types, but how much space they occupy will depends on the compiler where the code is going to run.(like integer , integer size also differ from compiler to compiler.).
- ✓ .If we use 16bit compiler it occupy 2 bytes of memory for all data types , if we use 32 bit compiler the pointer occupies 4 bytes of memory .
- ✓ We will see the following table because the integer and pointer sizes are same based on compilers.

Compiler Type	int size	Pointer size
16 bit	2 bytes	2 bytes
32 bit	4 bytes	4 bytes

Example :

Write a program to find the size of pointers with different data types

```
#include<stdio.h>
int main()
{
        char *cptr;
        int *iptr;
        float *fptr;
        double *dptr;
        printf("\nThe size of character pointeris: %d ",sizeof(*cptr));
        printf("\n The size of integer pointer is: %d ",sizeof(* iptr));
        printf("\n The size of float pointer is: %d ",sizeof(* fptr));
        printf("\n The size of double pointer is: %d ",sizeof(* dptr));
        return 0;
}
```

OUTPUT: (on 32 bit compiler)

```
The size of character pointer is: 1
The size of integer pointer is: 4
The size of float  pointer is: 4
The size of double pointer is: 8
```

Declaring and initializing pointer variables:

- ✓ To declare a pointer variable like normal variable declaration with *.

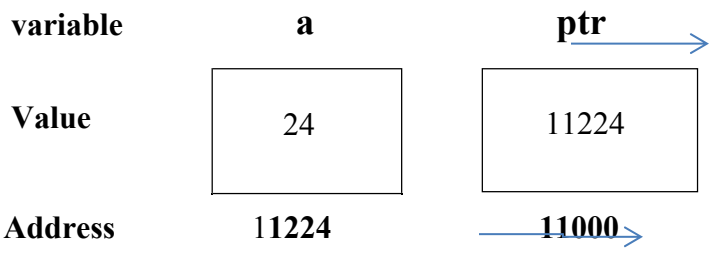
```
int a=24;
int *ptr; // pointer declaration
```

- ✓ Here ptr is a pointer variable it stores the address of the variable, if we are not assign any address to the pointer variable it stores the garbage value.

```
ptr= &a; // pointer initialization
```

- ✓ In this stores the address of the a is stored in ptr. When we will print the ptr then prints the address of the a.
- ✓ To access the value inside a particular address, then we will prefix * with pointer name or address i.e *ptr.

```
int a=24;
int  *ptr; // declaration of pointer
ptr = &a; // assigning address into ptr
```



printf("%d ",ptr) ; prints address i.e 11224

printf("%d ",*ptr) ; prints value inside the address i.e 24

printf("%d",&ptr); prints address of the ptr is 11000

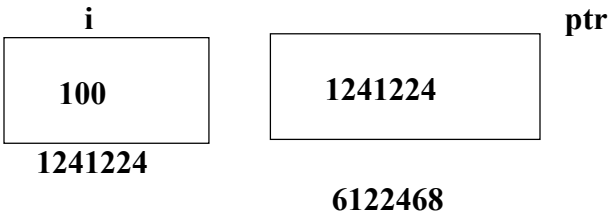
Example:

```
#include<stdio.h>
int main()
{
        int i=100;
        int *ptr;
        ptr=&i;
        printf("\n%d",i);
        printf("\n%d",&i);
        printf("\n%d",ptr);
        printf("\n%d",&ptr);
        printf("\n%d",*ptr);
        printf("\n%d",*(&i));
        return 0;
}
```

OUTPUT:

```
100
1241224
1241224
6122468
100
100
```

Explanation :



i=100
&i=1241224
ptr= 1241224
&ptr=6122468
*ptr=100
*(&i)=100

Difference between * and & operators :

Address operator (&)	Indirection operator (*)
Pointer address operator is denoted by '&' symbol When we use ampersand symbol as a prefix to a variable name, it gives the address of that variable. lets take an example – &n - It gives an address on variable n Working of address operator #include<stdio.h>	Indirection operator is denoted by '*' symbol when we use asterisk symbol as a prefix with variable name, it gives the value which is inside specified address .it is also called as value operator or dereferencing operator. #include<stdio.h>

```
 void main()
{
int n = 10;
 printf("\nValue of n is : %d",n);
printf("\nValue of &n is : %u",&n);
 }
```

Output :

Value of n is : 10
Value of &n is : 1002

Consider the above example, where we have used to print the address of the variable using ampersand operator. In order to print the variable we simply use name of variable while to print the address of the variable we use ampersand along with %u

printf("\nValue of &n is : %u",&n);

Understanding address operator Consider the following program –

```
#include<stdio.h>
int main()
{
int i = 5;
int  *ptr;
ptr = &i;
 printf("\nAddress of i : %u",&i);
printf("\nValue of ptr is : %u",ptr);
return(0);
 }
```

```
 void main()
{
int n = 10;
 printf("\nValue of n is : %d",n);
printf("\nValue of *n is : %u",*n);
 }
```

Output :

Value of n is : 10
Value of &n is : 10

5.2 Pointer Arithmetic:

Incrementing Pointer: Incrementing Pointer is generally used in array because we have contiguous memory in array and we know the contents of next memory location. Incrementing Pointer Variable Depends Upon data type of the Pointer variable

Formula:

new value = current address + i * size_of(data type)

Three Rules should be used to increment pointer –

Address + 1 = Address

Address++ = Address

++Address = Address

Pictorial representation :

i

1224

ptr

1508

After increment

ptr

1508

Data Type	Older address stored in pointer	Next address stored in pointer after increment (Ptr++)
Int	1224	1226
char	1224	1225
Float	1224	1228

Explanation :

- ✓ Incrementing a pointer to an integer data will cause its value to be incremented by 2 .
- ✓ This differs from compiler to compiler as memory required to store integer vary compiler to compiler.
- ✓ **Note to Remember :** Increment and Decrement Operations on pointer should be used when we have Continues memory (in Array).

Example 1 : Increment Integer Pointer

#include<stdio.h>

```
int main()

{
 int *ptr=(int *)1000;
ptr=ptr+1;
printf("New Value of ptr : %u",ptr);
return 0;
 }
```

Output :
New Value of ptr : 1002

Example 2 : Increment Double Pointer

```
#include<stdio.h>
int main()
{

 double *ptr=(double *)1000;

 ptr=ptr+1;
 printf("New Value of ptr : %u",ptr);
 return 0;


}
```

Output :
New Value of ptr : 1008

Note :

- ✓ In C, the programmer may add or subtract integers from pointers, we can also subtract one pointer from the other.
- ✓ An integer value can be added or subtracted from pointer variables but one pointer variable cannot be added to another pointer variables. That mean only values can be added.

Example :

```
int *ptr1=10;
int *ptr2=20;
*ptr1+*ptr2        it is possible i.e values can be added so the value is 30;
 ptr+ptr2          Not possible i.e addresses cannot be added.
```

Example 3:

```
#include<stdio.h>
int main()
{
        int x=10;
        int y=25;
        int *ptr1=&x;
        int *ptr2=&y;
        printf("x+y=%d",*ptr1+*ptr2);
        return 0;
}
```

Output:
x+y=35

Example 4:

```
#include<stdio.h>
int main()
{
        int n *p1,*p2;
        p1=&n;
        p2=p1+2;
        printf("the difference between p1 & p2 is %d memory segments",p2-p1);
        return 0;
}
```

Output: the difference between p1 & p2 is 2 memory segments

Explanation:

Statement	Value in p1	Value in p2
int n,*p1,*p2	Garbage value	Garbage value
p1=&n	1000	Garbage value
p2=p1+2	1000	1008

p2-p1 1008-1000=8 bytes (int occupies 4 bytes for each element, so, 8 bytes for 2 elements.)

Interview Problem: find the output for following program

```
#include<stdio.h>

int main()

{

        int arr[5]={10,20,30,40,50};

        int *ptr;

        ptr=arr;

        printf("%u\n",*++ptr+3);                    statement1
```

```
printf("%u\n",*(ptr--+2)+5);          statement2
printf("%u\n",*(ptr+3)-10);           statement3
return 0;
}
```

OUTPUT:
23
45
30

Explanation:

arr[5]={10,20,30,40,50}

10	20	30	40	50
1000	**1002**	**1004**	**1006**	**1008**

arr

ptr **1000**

ptr=arr or ptr=&arr or ptr=arr[0];

1000

Statement 1:

*++ptr+3 (R to L)

*1002+3

20+3 => 23

Statement 2:

*(ptr - - +2)+5

*(1002 +2)+5

*1006+5

40+5 => 45

Statement 3:

*(ptr+3)-10

*(1000+3)-10

*1006-10

40-10 => 30

5.3 NULL Pointer:

- ✓ Null pointer which is a special values that does not point anywhere, this means that NULL pointer does not point to any valid memory address .
- ✓ To declare a null pointer you may use the predefined constant NULL, which is defined in several standard header files i.e <stdio.h> , <stdlib.h> and <string.h> .
- ✓ The NULL is a constant in c, it returns value is 0.
- ✓ If any pointer does not contain valid memory address or pointer is uninitialized, then the pointer is a null.
- ✓ We can also assign 0 or NULL to make a pointer as "NULL pointer" .

Example : int *ptr==NULL

Example 1:

```
#include<stdlib.h>
int main()
{
   int * ptr=NULL;
   printf("the value of ptr is :%x",ptr);
   if(ptr)
    printf("\nPointer is not a NULL pointer");
  if(!ptr)
    printf("\nPointer is a NULL pointer");
 return 0;

}
```

OUTPUT:
pointer is a NULL pointer

Example 2:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
      int num=24;
      int *ptr1=&num;
      int *ptr2;
      int *ptr3=0;
      if(ptr1==0)
         printf("\nPointer 1 : NULL");
      else
         printf("\nPointer 1 : NOT NULL");
       if(ptr2==0)
          printf("\nPointer 2 : NULL");
       else
          printf("\nPointer 2 : NOT NULL");
       if(ptr3==0)
         printf("\nPointer 3 : NULL");
       else
```

```
        printf("\nPointer 3 : NOT NULL");
    return 0;

}
```

OUTPUT:
Pointer 1: NOT NULL
Pointer 2: NULL
Pointer 3:NULL

- ✓ ptr1 is initialized with address of num. Thus,ptr1 contains valid memory address .
- ✓ ptr2 is uninitialized, it contains garbage value i.e not valid memory address and ptr3 assigned 0. Thus , ptr2 and ptr3 are the NULL pointers,

Note:

- ✓ A runtime error is generated if you try to dereference a null pointer.

Example:

```
int *ptr=NULL;
printf(" Address  %d",ptr);          it gives the output  0
printf("  Value  %d",*ptr);          Generate logical error at run time.
```

5.4 Generic pointers (Void Pointers):

- ✓ Generic pointer is a pointer variable that has void as its data type
- ✓ It is also called as void pointer.
- ✓ It can be used to point to variable of any data type. It declare like pointer variable but using 'void' keyword as pointer data type.

Ex: void *ptr

- ✓ In c, since we cannot have a variable of type void.
- ✓ A pointer variable of type void * cannot be referenced. We need to type cast a void pointer to another kind of pointer before using it.

Example :

```
#include<stdio.h>
int main()
{
    int x=10;
    char ch='R';
    void *gp;          // generic pointer
    gp=&x;             // assigning integer address to void pointer
    printf("\ngeneric pointer points to the integer %d",*(int *)gp);
    gp=&ch;            // assigning character address to void pointer.
    printf("\n generic pointer points to the character %c",*(char *)gp);
    return 0;
}
```

Note : A compilation error will be generated if we assign a pointer of one type to a pointer of another type without a cast.

5.5 Pointers as function Arguments:

- ✓ We know that, it is impossible to modify the actual arguments when we pass them to a function, the incoming arguments to a function are treated as local variables in the function and those local variables get a copy of values passes from their calling functions. This technique is known as call by value.
- ✓ Pointers provide a mechanism to modify data declared in one function using code written in another function.
- ✓ The calling function sends the addresses of the variables and the called function must declare those incoming arguments as pointers. In order to modify the variables sent by the calling function, the called function must dereference the pointer that we are passed it. This technique is known as call by reference.
- ✓ To use pointers for passing arguments to a function, the programmer must do the following steps:
    - ➢ Declare the function parameters as pointer
    - ➢ Use the dereferenced pointer in function body
    - ➢ Pass the address as the actual argument when the function is called.

Example:

```
#include<stdio.h>
void sum(int*,int*,int*);
int main()
{
        int num1,num2,total;
        printf("\n Enter two values num1 & num2:");
        scanf("%d%d",&num1,&num2);
        sum(&num1,&num2,&total);
        printf("\Total =%d",total);
        return 0;
}
void sum(int *a,int *b,int *t)
{
        *t=*a+*b;
}
```

OUTPUT:
Enter two value num1 & num2:
10
20
Total = 30

5.5.1 Parameter passing mechanism:

There are two ways that a C function can be called from a program. They are,

1. Call by value
2. Call by reference

Call by Value

In call by value method, the value of the variable is passed to the function as parameter. The value of the actual parameter cannot be modified by formal parameter. Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

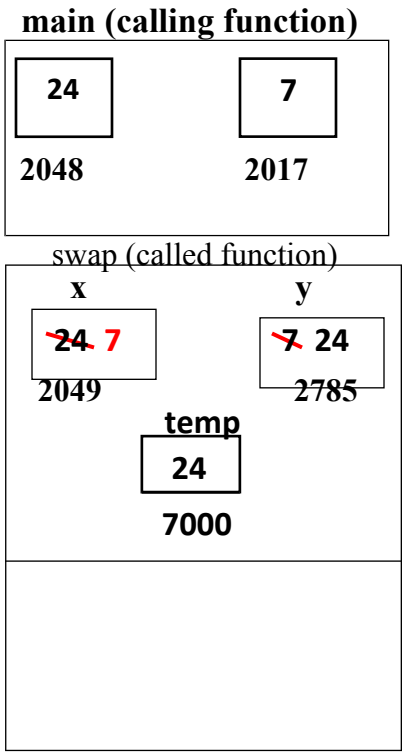
Example

```
#include<stdio.h>
void swap(int , int );
int main()
{
  int a = 24, b = 7;
  printf("\nValues before swap \n a = %d and b = %d", a, b);
  swap(a,b);
   printf(" \nValues after swap \n a = %d and b = %d", a, b);
  return 0;
}
void swap(int x, int y)
{
            int  temp;
            temp = x;
            x = y;
            y = temp;
}
```

OUTPUT

Values before swap
a =24 and b = 7
Values after swap
a =24 and b = 7

Explanation:



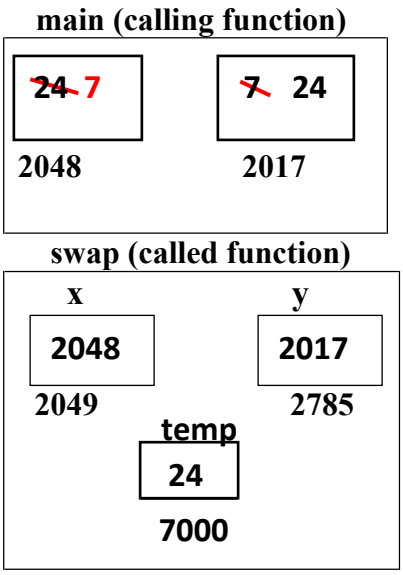
Call by reference

In *call by reference* method, the address of the variable is passed to the function as parameter. The value of the actual parameter can be modified by formal parameter. Same memory is used for both actual and formal parameters since only address is used by both parameters.

Example:

```
#include<stdio.h>
void swap(int *, int *);
int main()
{
   int a = 22, b = 44;
  printf("\nValues before swap \n a = %d and b = %d", a, b);
  swap(&a,&b);
  printf(" \nValues after swap \n a = %d and b = %d", a, b);
  return 0;
}
void swap(int *x, int *y)
{
            int temp;
            temp = *x;
            *x = *y;
            *y = temp;
}
```

OUTPUT:
Values before swap
a =22 and b = 44
Values after swap
a =44 and b = 22



5.6 POINTERS & ARRAYS :

- ✓ An array occupies consecutive memory locations.

```
int arr[ ] = {1,2,3,4,5};
```

a[0]	a[1]	a[2]	a[3]	a[4]
1	2	3	4	5
100	102	104	106	108

- ✓ The name of the array is the starting address of array in memory. It is also known as "**Base address** ".

```
int *ptr;

ptr = &arr[0];
```

- ✓ Ptr is made to point the first element of the array.

Ex:-

```
main( )
{
    int arr[ ]={1,2,3,4,5};
    printf("Address of array=%p %p %p",arr,&arr[0],&arr);
    ptr =&arr[2];
```

1	2	3	4	5
100	102	104	106	108

```
    ptr =arr[3];
```

✓ If pointer variable ptr holds the address of the first element in the array.

```
    int *ptr =&arr[0];
    ptr++;
    printf("The value of the second element of array is %d",*ptr);
}
```

Note:-

An error is generated if an attempt is made to change the address of the array.

Example 1:

```
#include <stdio.h>
int  main( )

{
    int a[ ]={1,2,3,4,5,6,7,8,9};
    int *p1,p2;
    p1=a;
    p2=&a[8];
    while(p1<=p2)
    {
```

OUTPUT:
1 2 3 4 5 6 7 8 9

```
            printf("%d\t",*p1);
            p1++;
        }
  return 0;
}
```

Example 2:

```
int main( )
{
        int arr[ ]={1,2,3,4,5};
        int *ptr,i;
        ptr=&arr[2];
        *ptr= 24;
        *(ptr+1)=0;
        *(ptr-1)=1;
        printf("\n array is");
        for(i=0;i<5;i++)
        {
            printf("%d\t",*(arr+1));
        }
}
```

OUTPUT:
1 1 24 0 5

5.7 Pointers to constant:

- The value of the variable to which the pointer is pointing is constant variable.
- That means, a pointer through which one **cannot change the value** of the variable to which is known as "pointer to constant".

Syntax-1

const <type of pointer> *<pointer name>

Ex:-

const int *ptr;

(OR)

Syntax-2

<type of pointer> const *<pointer name>

Ex:-

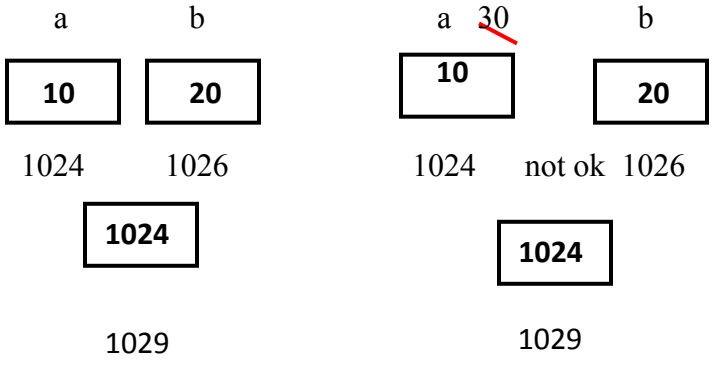
int const *ptr;

- These pointer can change the address the point to but cannot change the value at the address they are pointing to.

Ex:-

```
main( )
{
    int a=10;
    cons tint *ptr=&a;        ptr to const value.
    *ptr=30;                  error because value we can't change
    return 0;
}
```

Example:2

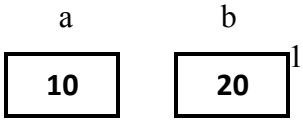


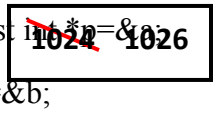

```
main()
{
//Definition of the variable.
    int a=10;
    int b=20;

    const int*ptr=&a;

    ptr=&b;                   It works because address is not constant.
    return 0;
}
```


1) const int *ptr=&a;





5.8 Constant pointers:

- ✓ A constant pointer is one that cannot change address it contains.
- ✓ In other words, we can say that once a constant pointer points to a variable it cannot point to any other variable.

Note:- However ,these pointer can change the value of the variable they point to but cannot change the address they are holding.

Syntax:- <type of pointer>*const <pointer name>

Ex: int *const ptr

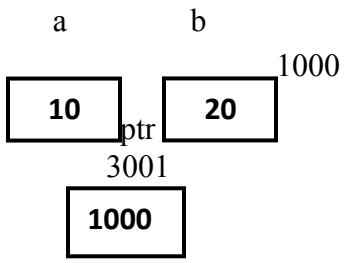
Constant pointer	Value change	Address change
int * const ptr	Possible	Not possible

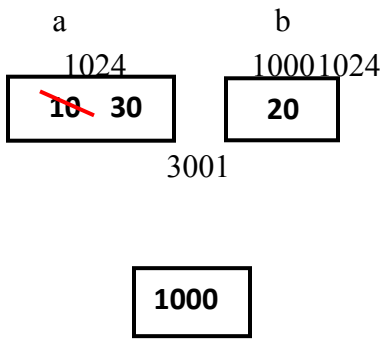
Example	Value constant	Pointer constant
char *ptr	No	No
const char *ptr	Yes	No
char const *ptr	Yes	No
char *const ptr	No	Yes
const char *const ptr	Yes	Yes

Example:

```
int main( )
{
        int a=10;
        int b=20;
        const int *ptr=&a;
        *ptr=30;        possible, we can change the value of ptr
        ptr=&b;         Error because we can't change the address of ptr.
        return 0;
}
```

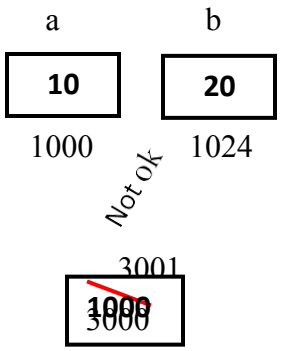
1) int *const ptr=&a; *ptr=30;





2) int *const ptr=&a;

ptr=&b; error we can't change the adrees



Trick to remember constant pointer and pointer to constant:

- ✓ Easy to identify the pointer constant and constant pointers observe the following table

Example	before asterisk(*)	Part after asterisk(*)	Description
const char *ptr	const char	Ptr	const is associated with data type so , value is constant.
char const *ptr	char const	Ptr	const is associated with data type so , value is constant.
char *const ptr	Char	const ptr	const is associated with pointer so , pointer is constant.
const char *const ptr	const char	const ptr	const is associated with both so ,both are constant.

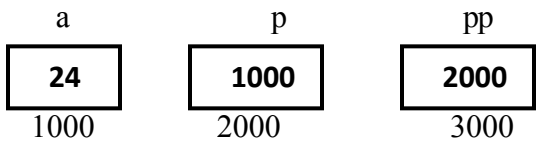
Note : In above table we observe 3 things

1. const keyword placed before the asterisk (*) then that pointer is known as pointer to constant (value is constant).
2. const keyword placed after the asterisk (*) then that pointer is known as constant pointer (address is constant).
3. const keyword placed both sides i.e before and after asterisk (*) then the pointer is both pointer to constant and pointer to constant.

5.9 Pointers to pointers:-

- ✓ As we know , a pointer is used to store the address of a variable in C. Pointer reduce the access time of variable.
- ✓ We can also defined a pointer to store the address of another pointer. Such pointer is known as double pointer (pointer to pointers)
- ✓ The first pointer is used to store the address of a variable whenever the second pointer is used to store the address of the first pointer.

1)int a=24;
int *p=&a;
int **pp=&p;



- ✓ **a** is a variable stored at 1000 location. **p** is pointer which contains the address or a variable and pp is double pointer which contains the address of **p** pointer.

****pp Explanation:**

- ✓ It will give value at location 2000 ,which is 1000.
- ✓ **p will give the value at location 1000 , which is 24.
  - ➢ Value of **pp=24
  - ➢ Value of *p=24
  - ➢ Value of a=24
  - ➢ Value of p=1000
  - ➢ Value of pp=3000
  - ➢ Value of &a=1000

Example :

```
int  main( )

{

        int num=7;

        int *p,**pp;

        p=&num;

        pp=&p;

        printf("\n%d address of number variable",&num);

        printf("\nAddress in p variable-%d",p);

        printf("\nValue of *p is %d",*p);

        printf("\nAddress of pp is %d",pp);

        printf("\nValue of *pp is %d",*pp);
```

OUTPUT:
Adreess of number variable 644523
Adreess in p variable 644523
Value of *p is 7
Adreess in pp is 521342
Value of *pp is 644523
Value of **pp is 7

```
printf("\nValue of **pp is %d",**pp);
return 0;   }
```

Note : In double pointer (pointer to pointer) ,we can also prefix the more than two ** to pointer variable. If we declare more pointers what will happen ? observe the following.

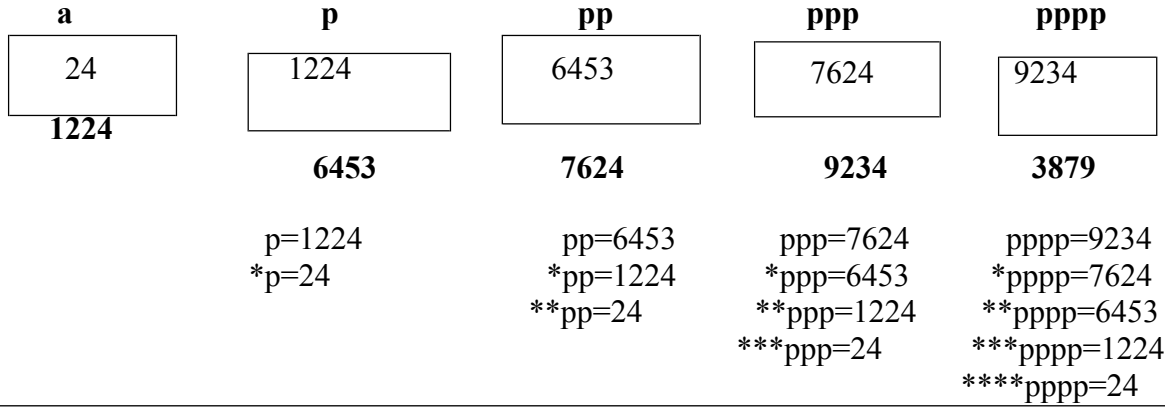
Example 2 :

```
#include<stdio.h>
int main( )
{
        int a=24;
        int *p,**pp,***ppp,****pppp;
        p=&a;
        pp=&p;
        ppp=&pp;
        pppp=&ppp;
        printf("%d\n address of variable a : ",p);
        printf("\n value of variable a :%d",*p);
        printf("adress of pp is: %d",pp);
        printf("\n Address inside pp is %d",*pp);
        printf("\n Value of **pp is: %d",**pp);
        printf("adress of ppp is: %d",ppp);
        printf("\n Address inside ppp is %d",*ppp);
        printf("\n adresss of *ppp is: %d",**ppp);
        printf("\n value inside ***ppp is:%d",***ppp);
        printf("adress of pppp is: %d",pppp);
        printf("\n Address inside pppp is %d",*pppp);
        printf("\n adresss of *pppp is: %d",**pppp);
        printf("\n adress inside ***ppppis:%d",***pppp)
        printf("\n value inside the ****pppp is
:%d",****pppp);
        return 0;
}
```

OUTPUT:

```
address of variable a: 1224
value of variable *p : 24
adress of pp is: 6453
Address * pp is : 1224
Value of **pp is: 24
adress of ppp is 7624
Address in * ppp is:6453
adresss of**ppp is:1224
 value inside ***ppp is 24
adress of pppp is: 9234
Address * pppp is: 7624
adresss of **pppp is 6453
adress inside ***pppp is: 1224
value inside the ****pppp: 24
```

Explanation :



5.10 Pointers & strings:-

String is nothing but array of character terminated with '\0'.

1. char str[10];
   str[0]='c';
   str[1]='s';
   str[2]='e';
   str[3]='\0';
2. char str[10]={'c','s','e','\0'};
3. char str[10]="cse";

- ✓ when the double quotes are used ,NULL character ('\0') is automatically appends to the end of string.
- ✓ When a string is declared like this the compiler sets avoid a contiguous block or memory 10 bytes long to hold characters and initiate its first four characters cse\0 .

Example:

```
#include <stdio.h>

int main( )

{
        char str[ ]="Hello Cse";

        char *pstr;

        pstr=str;

        printf("\n the string is:");

        while(*pstr!='\0')

        {

                printf("%c",*pstr);

                 pstr++;

        }

        return 0;

}
```

OUTPUT:
Hello Cse

- ✓ In this program, we declare pointer *pstr to show the string on the screen. We then point the pointer pstr at str. Then we print each character of the string in the while loop. Instead of using while loop, we could have directly use the function puts

puts(pstr); → **it prints entire string**

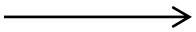
The function prototype for puts :

int puts(const char *s);

- ✓ Here the const modifier is used to assure the user that the function will not modify the contents pointed to by the source pointer. The address of the string is passed to the function as an arguments.

- ✓ The parameter passed to puts() is a pointer which is the address to which it points to, an address .thus writing puts(str) i.e means passing the address of str[0].

By writing puts(str) str means address or str[0].



Similarly when we write puts(pstr)
Pstr=str;

```
#include <stdio.h>
void main( )
{
        char str[100],*pstr;
         int upper=0,lower=0;
         printf("\n Enter the string:");
         gets(str);
         pstr=str;
          while(*pstr!='\0')
         {
                  if(*pstr>='A' && *pstr <='Z')
                           upper++;
                  else if(*pstr>='a' && *pstr<='z')
                            lower++;
                  pstr++;
```

```
        }
        printf("The total number of capitals is %d",upper);
        printf("The total number of lower letters is %d",lower);
}
```

Difference between array name & pointer:-

1. Array address cannot be changed but pointer address will be changed.

```
#include <stdio.h>
void main( )
{
      int arr[5],i;
      for(i=0;i<5;i++)
      {
            *arr=0;
             arr++;//Error
      }
      for(i=0;i<5;i++)
      {
          Printf("%d",*(arr+i));
      }
}
```

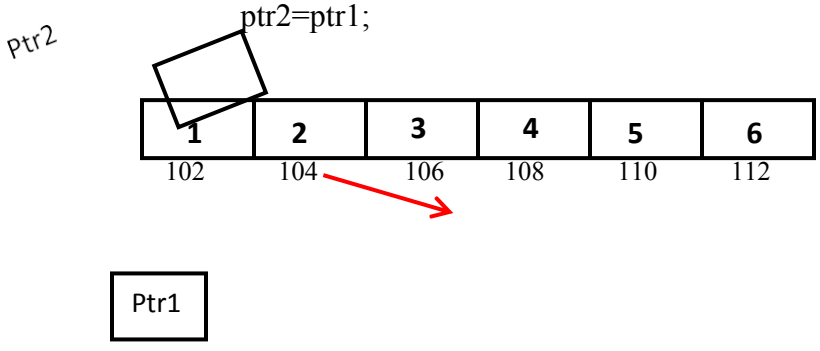
```
#include <stdio.h>
void main( )
{
      int arr[5],i,*parr;
      parr=arr;
      for(i=0;i<5;i++)
      {
            *parr=i;
             parr++;
      }
      for(i=0;i<5;i++)
      {
          Printf("%d",*(parr+i));
      }
}
```

2. array cannot be assigned to another array.

```
int arr[ ]={1,2,3,4,5,6};
int arr2[6];
arr2=arr;  //Error
```

But one pointer variable can be assigned to another pointer variable of the same type

```
int arr[ ]={1,2,3,4,5,6},*ptr1,*ptr2;
ptr1=arr1;
ptr2=ptr1;
```



3. The address operator returns the address the operand .But when an address operator is applied to an array name. If gives the same value without the operator.

4. **Size of operator**

```
#include <stdio.h>
void main( )
{
        int arr[ ]={1,2,3,4,5};
        int *ptr;
        ptr=arr;
        printf("The size of array=%d",sizeof(arr)); → it give size of array i.e 20
        printf("The size of pointer=%d",sizeof(ptr));→ it gives size of pointer i.e 2
}
```

5.11 Function pointers (pointers to functions):-

- ✓ Every function code along with its variable is allocated some space in the memory.
- ✓ Function pointers are pointer variable that point to the address of a function.
- ✓ Like other pointer variable ,function pointer can be declared, assigned value and used to access the function they point to.
- ✓ This is a useful technique for passing a function as argument to another function.
- ✓ In order to declare a pointer to a function we have to declare its prototype of the except that the name of a function enclosed between parentheses. ( ) and an (*) is inserted before name.
- ✓ It is possible to declare pointers to functions . But ,if you want to create a pointer to function the declaration is completely depends on the function prototype.

Declaration of function pointer:

Syntax :

```
return_type(*pointer_name) (arguments list);
```

Example:

```
int (*ptr) (int,int);
```

It can point to any function ,Which is taking integer argument & return int data type.

Note :

- ✓ ptr-name we must place inside the parentheses along with pointer(*). **Why we declare pointer name within the parenthesis**.
- ✓ If we declare pointer name without parenthesis ,it changes the total format, to clear understand observe the following .

Declaration of function pointer is

int (*ptr) (int, int)

Now, We re –write the above statement without using parenthesis for function pointers

int * ptr (int,int)

In above function, ptr is a function which takes two integer arguments and it returns address of integer variable (int*) .

When we declare ptr is a normal function, it returns the integer address . its not function pointer. so, In function pointer we must declare pointer name within parenthesis otherwise compiler consider as normal function it returns integer address.

Initialization of function pointer:

- ✓ if we have declared a pointer to a function, then that pointer can be assigned the address of the correct function jusy by its name.
- ✓ function name always holds the starting address (base address) of the function, it is optional to use the address operator (&) in front of function name.
- ✓ if fp is a function pointer and we have a function add() then,

int (*fp)(int,int) → function pointer declaration

int add (int,int) → function declaration

fp=add; or fp= &add → function pointer points to the add function

- ✓ When we assign the address of function to pointer , the arguments size,type and return type should be match, otherwise it gives error i.e incompatible error.

int mul(int,int,int)

fp=mul or &mul →error, incompatible error because function pointer having two arguments but mul function having three arguments.

Calling functional pointer :

- ✓ When a pointer to a function is declared ,it can be called in two ways

(*fp)(10,20) **OR** fp(10,20)

Example 1:

```
#include <stdio.h>
int mult(int x,int y,int z);
int add(int x,int y);
int  main( )
{
        int res1,res2,res3;
        int (*ptr) (int,int);→ function pointer declaration, it is also declare above the main
        also res1=add(10,20);     function call without using function pointers.
        res2=mult(2,3,4);
         printf("Before using function pointers %d,%d",res1,res2);
         ptr=&add;     → assign the function address to function pointers
         res3=ptr(30,50); or (*ptr) → calling functional pointer
         printf("After using function pointers %d",res3);
       //   ptr=&mult;              Error , incompatible pointer assignment.
      return 0;
}
int add(int x,int y)
{
         int z=x+y;
         return z;
}
```

```
int mult(int x,int y,int z)
{
        int n=x*y*z;
        return n;
}
```

OUTPUT:

Example 2:

```
 Before using function  pointers  30     24
#include<stdio.h>
 After using function pointer       30
void display(int n);

void (*fp) (int);
int main()
{
        fp=display; // assigning function address into function pointer
        if(fp>0)        // Comparing function pointers
        {
                if(fp==display)
                        printf("\n The pointer points to the display function");
                else
                    printf("\n the pointer points to the display function");
            }
         fp(24);       // function pointer calling
        (*fp)(10);     // function pointer calling again with different value
        return 0;
```

```
}
void display(int n)
{
        printf("\n %d",n);
}
```

OUTPUT:
The pointer points to the display function
24
10