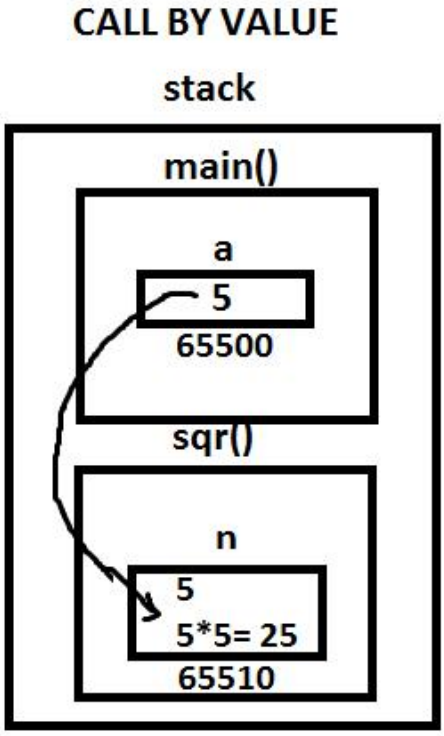
Passing parameters to the functions: [parameter passing techniques]

In C-Language, we can send the arguments to the functions in 2 ways.

1. **Call by value / pass by value.**
2. **Call by address / pass by address. [call by reference]**

Call by value / pass by value:

In call by value we are sending actual parameter value to the formal parameter. Later there is no relation is maintained in between actual and formal parameters. Due to this any change in formal parameter doesn't effects the value of actual parameter.



Eg: 1

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

void  sqr(int n)

{

n = n * n;

}  /* n  deleted after the function execution
*/

void main()

{

int a=5;
```

```
clrscr();
printf("Before function call a = %d\n" ,a);
sqr(a);  /* fun calling */
printf("After function call a = %d", a):
getch();
}
```

Output:

Before function call a = 5

After function call a = 5

Eg: 2 swapping of two integers

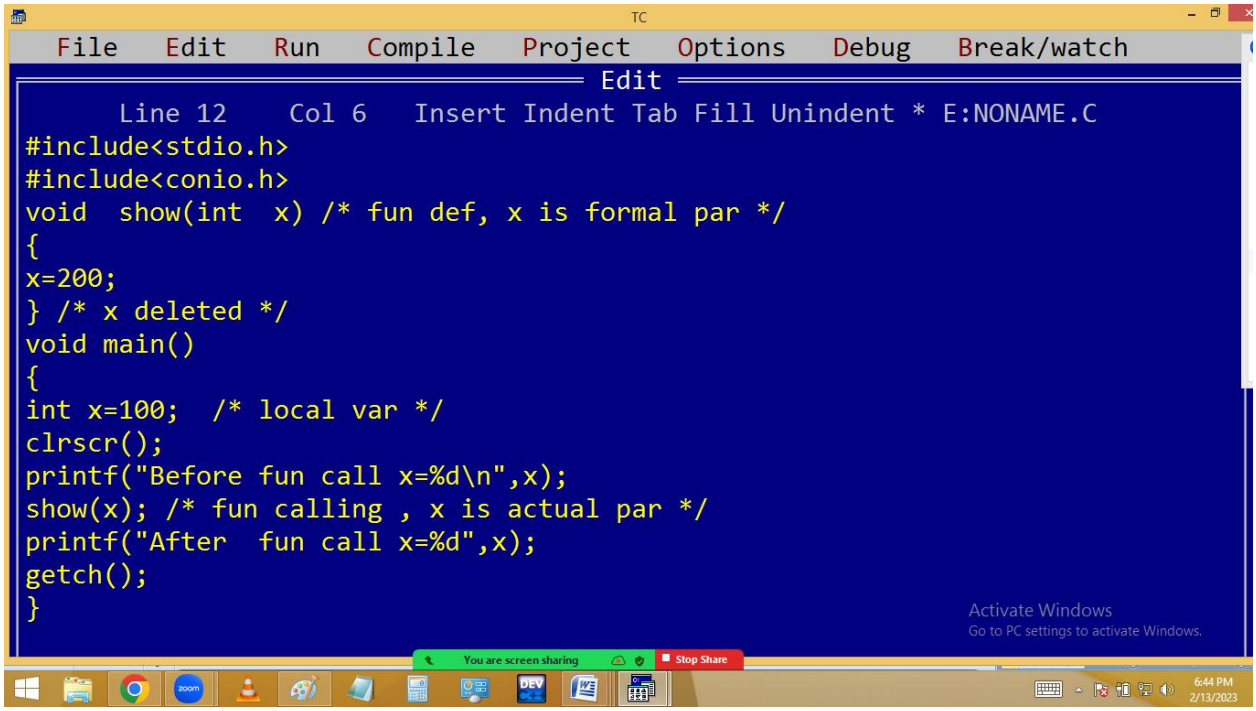
```
#include<stdio.h>
#include<conio.h>
void  swap(int  a, int  b)
{
int  temp=a;
a=b;
b=temp;
}
void  main()
```

```
{
int  a=5, b=7;
clrscr();
printf("Before fun call a=%d, b=%d\n" , a , b);
swap(a, b);
printf("After fun call a=%d, b=%d", a , b);
getch();
}
```

Output:

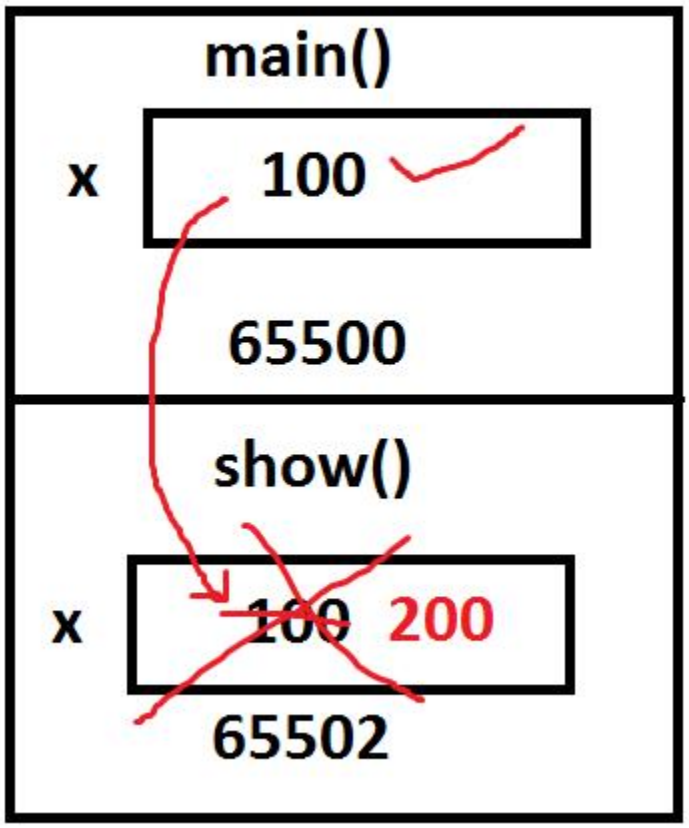
Before fun call a=5, b=7

After fun call a=5, b=7



```
Before fun call x=100
After  fun call x=100
```

Call by /pass By value



Call by address [Reference]:

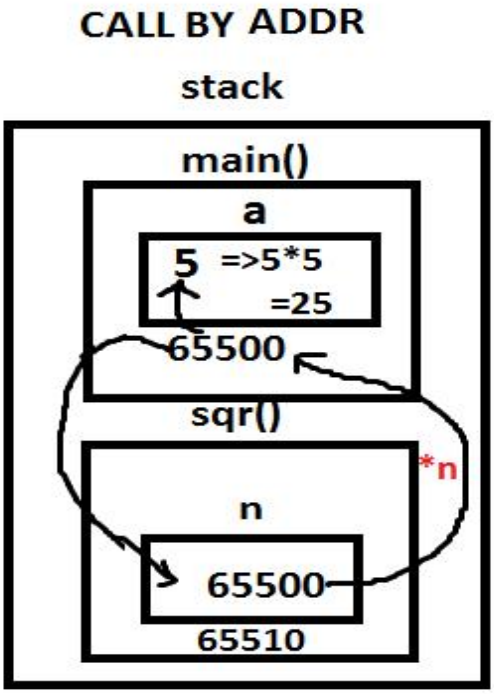
In call by address, the address of actual parameter is passed to formal parameter. Due to this the formal parameter should be declared as a pointer. Then only the formal parameter receives the actual parameter address. Due to this any changes in formal parameter effects in actual parameter address i.e. actual parameter value.

Hence pointers allows the local variables to access outside the functions and this process is called call by address / reference.

It is very much useful in handling the strings, arrays etc outside the functions.

Eg: 1

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

void  sqr(int *n)

{

*n = *n   *   *n;

}

void main()

{

int a=5;

clrscr();

printf("Before function call a = %d\n " ,
```



```
a);
sqr(&a);  /* fun calling with address */
printf("After function call a = %d " , a);
getch();
}
```

Output:

Before function call a = 5

After function call a = 25

Eg: 2 Swap of two integers

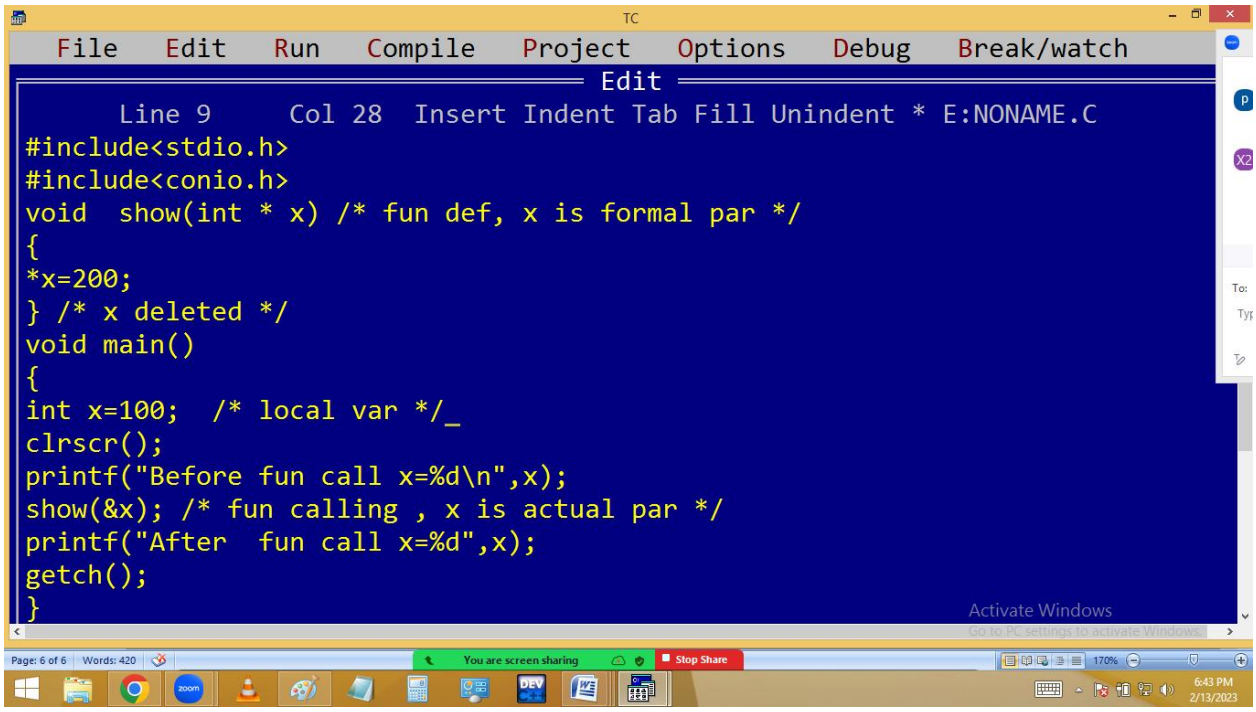
```
#include<stdio.h>
#include<conio.h>
void  swap(int  *a, int  *b)
{
int  temp=*a;   *a = *b;  *b=temp;
}
void  main()
{
int  a=5, b=7;
clrscr();
```

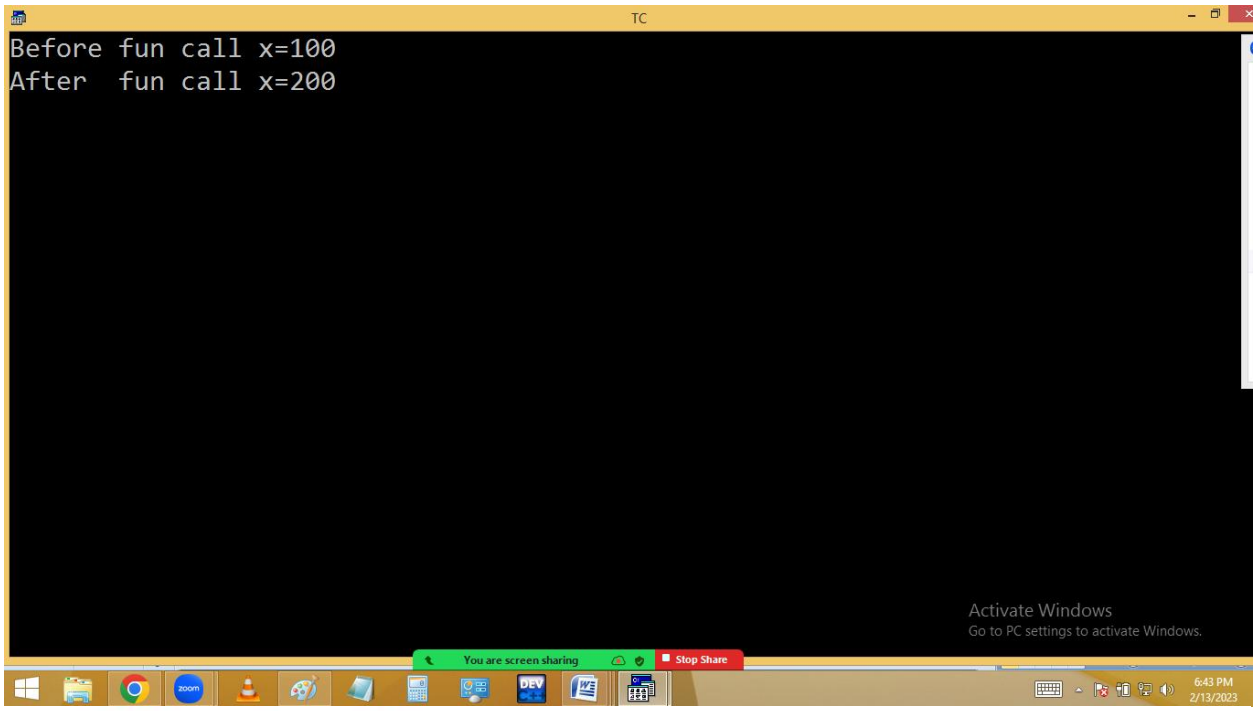
```
printf("Before fun call a=%d, b=%d\n", a ,b);
swap(&a, &b);
printf("After fun call a=%d, b=%d", a ,b);
getch();
}
```

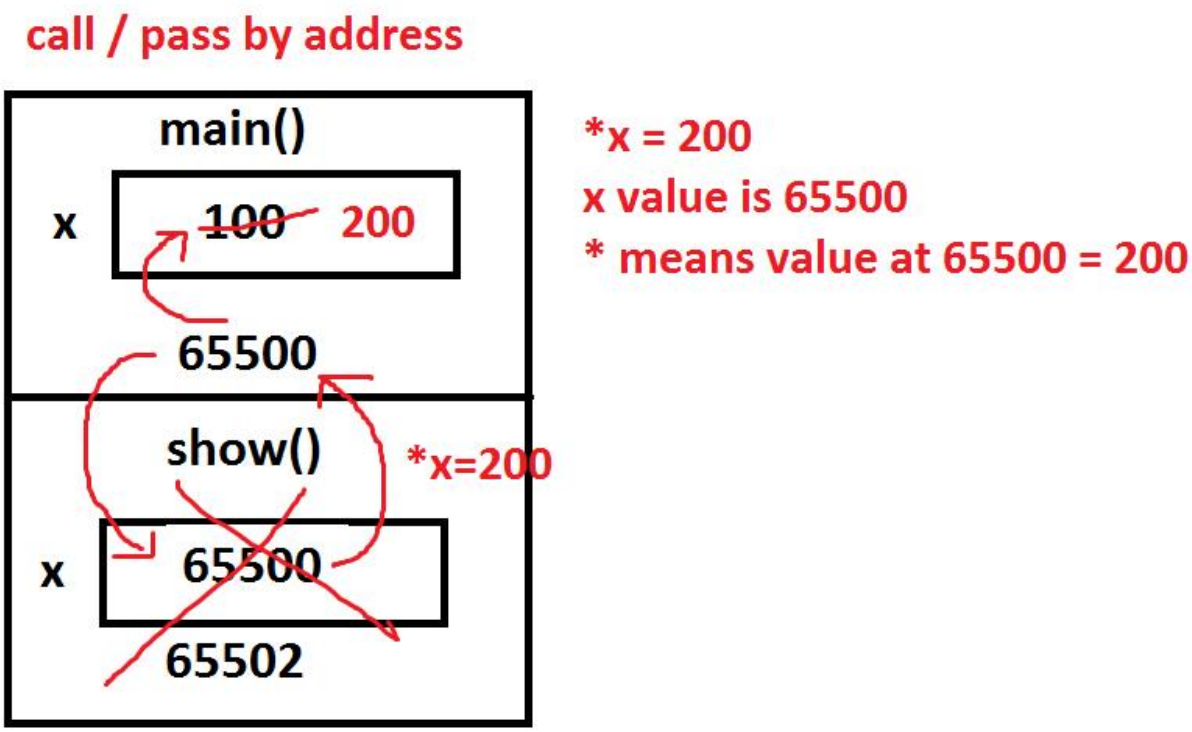
Output:

Before function call a=5, b=7

After function call a=7, b=5







Call by value:

```
File   Edit   Run   Compile   Project   Options   Debug   Br
Edit
      Line 3     Col 48  Insert Indent Tab Fill Unindent * E:9
#include<stdio.h>
#include<conio.h>
void sqr(int n)  /* local var, formal parameter_*/
{
n=n*n;
} /* n deleted */
void main()
{
int n=10; /* local var */
clrscr();
printf("Before fun n value %d\n",n);
sqr(n); /* call by value, n is actual parameter */
printf("After  fun n value %d",n);
getch();
}
```

```
Before fun n value 10
After  fun n value 10_
```

Call by address:

TC

File Edit Run Compile Project Options Debug Br

Edit

Line 5 Col 5 Insert Indent Tab Fill Unindent * E:9

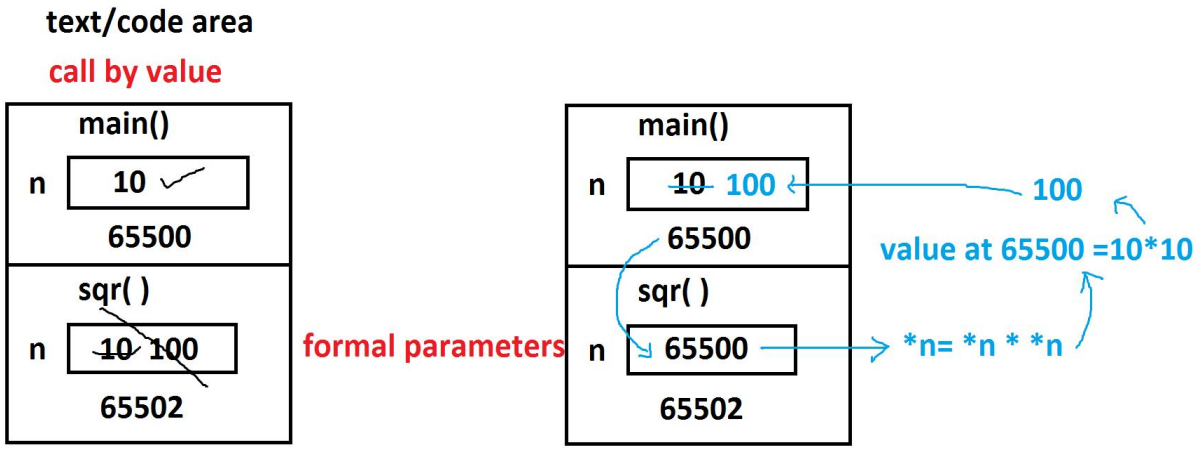
```
#include<stdio.h>
#include<conio.h>
void sqr(int *n)  /* local var, formal parameter */
{
*n=*n * *n;
} /* n deleted */
void main()
{
int n=10; /* local var */
clrscr();
printf("Before fun n value %d\n",n);
sqr(&n); /* call by value, n is actual parameter */
printf("After  fun n value %d",n);
getch();
}
```

9:40 AM

TC

```
Before fun n value 10
After  fun n value 100_
```

9:40 AM


text/code area
call by value
main()
n
10
65500
sqr( )
n
10 100
65502
formal parameters
main()
n
10 100
65500
sqr( )
n
65500
65502
100
value at 65500 =10*10
*n= *n * *n

TC

File Edit Run Compile Project Options Debug Break/wat

Edit

Line 12 Col 31 Insert Indent Tab Fill Unindent * E:9AM.C

```
#include<stdio.h>
#include<conio.h>
void swap(int a, int b)  /* local var, formal parameter */
{
int c=a; a=b; b=c;
}
void main()
{
int a=10, b=20; /* local var */
clrscr();
printf("Before fun a=%d, b=%d\n",a,b);
swap(a,b); /* call by value */
printf("After  fun a=%d, b=%d",a,b);
getch();
}
```

Watch

9:44 AM

TC

```
Before fun a=10, b=20
After  fun a=10, b=20
```

9:44 AM

TC

File Edit Run Compile Project Options Debug Break/wat

Edit

Line 5 Col 19 Insert Indent Tab Fill Unindent * E:9AM.C

```
#include<stdio.h>
#include<conio.h>
void swap(int *a, int *b)  /* local var, formal parameters */
{
int c=*a; *a=*b; *b=c;
}
void main()
{
int a=10, b=20; /* local var */
clrscr();
printf("Before fun a=%d, b=%d\n",a,b);
swap(&a,&b); /* call by address */
printf("After  fun a=%d, b=%d",a,b);
getch();
}
```

Watch

9:45 AM

TC

```
Before fun a=10, b=20
After  fun a=20, b=10_
```

9:46 AM

Passing array / string to function:

String/array is implicit pointer i.e. string / array variable stores base address. Due to this when string/array is passed to a function, implicitly base address is passed and formal parameter becomes pointer and it receives this address. Hence any change occurred in formal parameter, effects on actual parameter value also.

We can declare string / array formal parameter in 3 ways.

1. With size eg: char st[50] / int a[3]
2. Without size eg: char st[ ] / int a[ ]
3. As a pointer eg: char * st / int *a

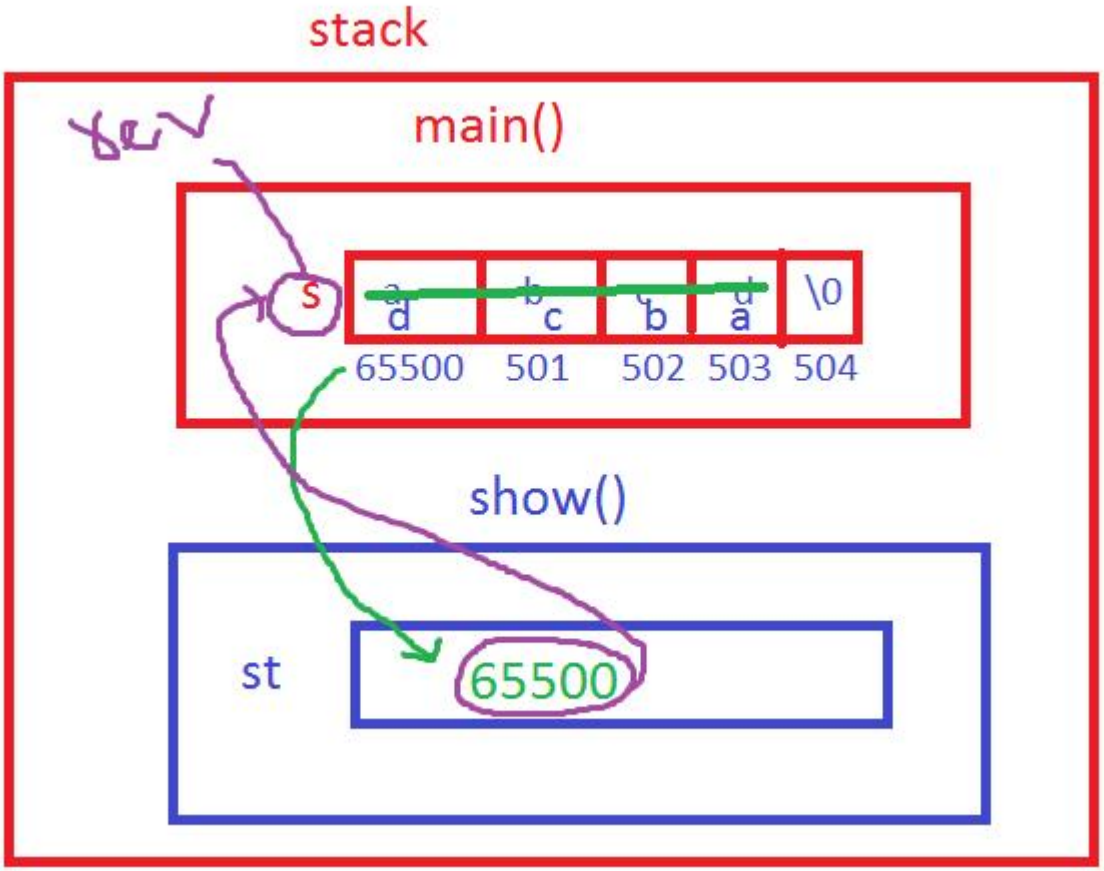
We can pass string / array actual parameter with or without address.

Eg:

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

void  reverse( char st[10] )    or  st[ ]  or  *st

{

strrev(st);

}

void  main()

{

char  s[10]="abcd”;

clrscr();

reverse(s);  or  reverse(&s);
```

```
printf("String = %s", s);
getch();
}
```

O/P: String = dcba



Passing array to function:

```
#include<stdio.h>
#include<conio.h>
void  show(int a[3])   or  a[ ]   or  *a
{
a[0]=100; a[1]=200; a[2]=300;
}
void  main()
{
int  a[3]={10,20,30};
clrscr();
show(a);   or  show(&a);
printf("Array elements %d %d %d",a[0],a[1],a[2]);
getch();
}
```

O/P: Array elements 100 200 300

Passing two – dimensional array to function.

```
#include<stdio.h>
#include<conio.h>
void  show( int (*a)[3] ) or a[2][3] or a[ ][3]
{
a[0][0]=10;  a[1][2]=60;
}
void  main()
{
int a[2][3]={1,2,3,4,5,6};
show(a); /* fun calling */
printf("a[0][0]=%d, a[1][2]=%d",a[0][0],a[1][2]);
getch();
}
```

Output: a[0][0]=10, a[1][2]=60;

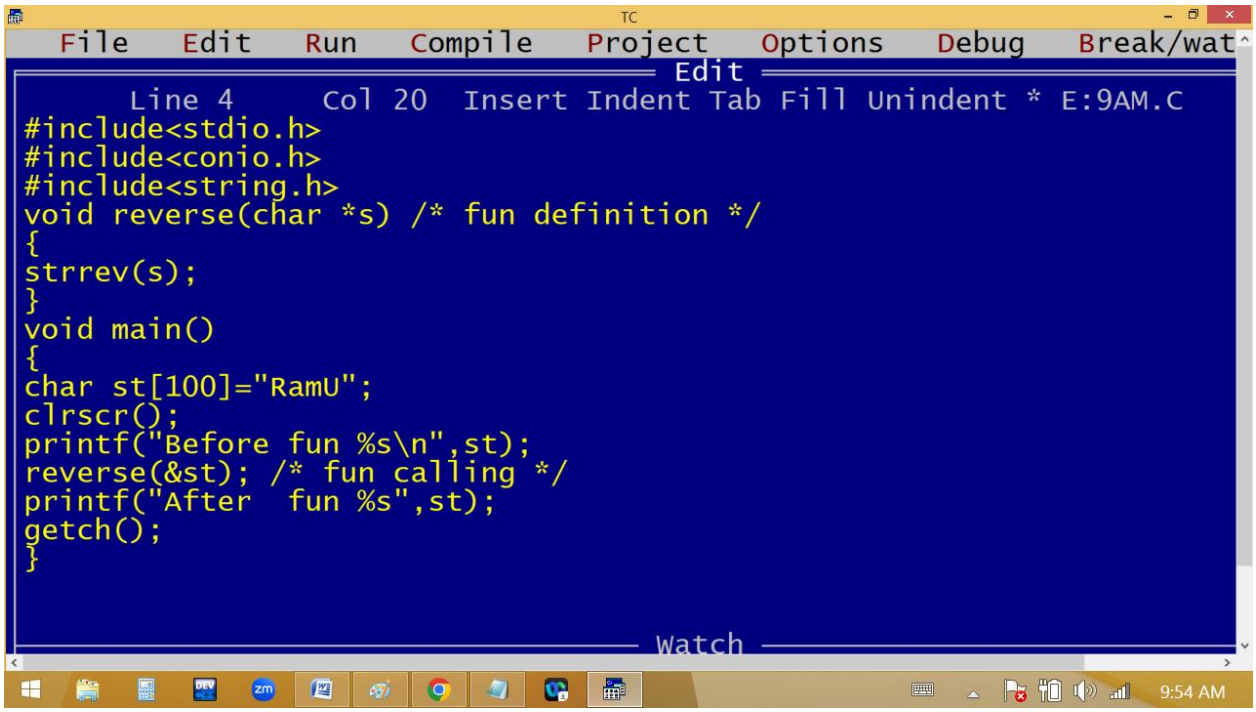
```
#include<stdio.h>
#include<conio.h>
void  show(int (*a)[3]) /* or a[2][3] or a[][3]*/
{
int  r,c;
printf("Elements are\n");
for(r=0;r<2;r++)
{
for(c=0;c<3;c++)
{
printf("%4d",*(*(a+r)+c)); /*or a[r][c]*/
}
printf("\n");
}
}
void  main()
{
int  a[2][3]={1,2,3,4,5,6};
```

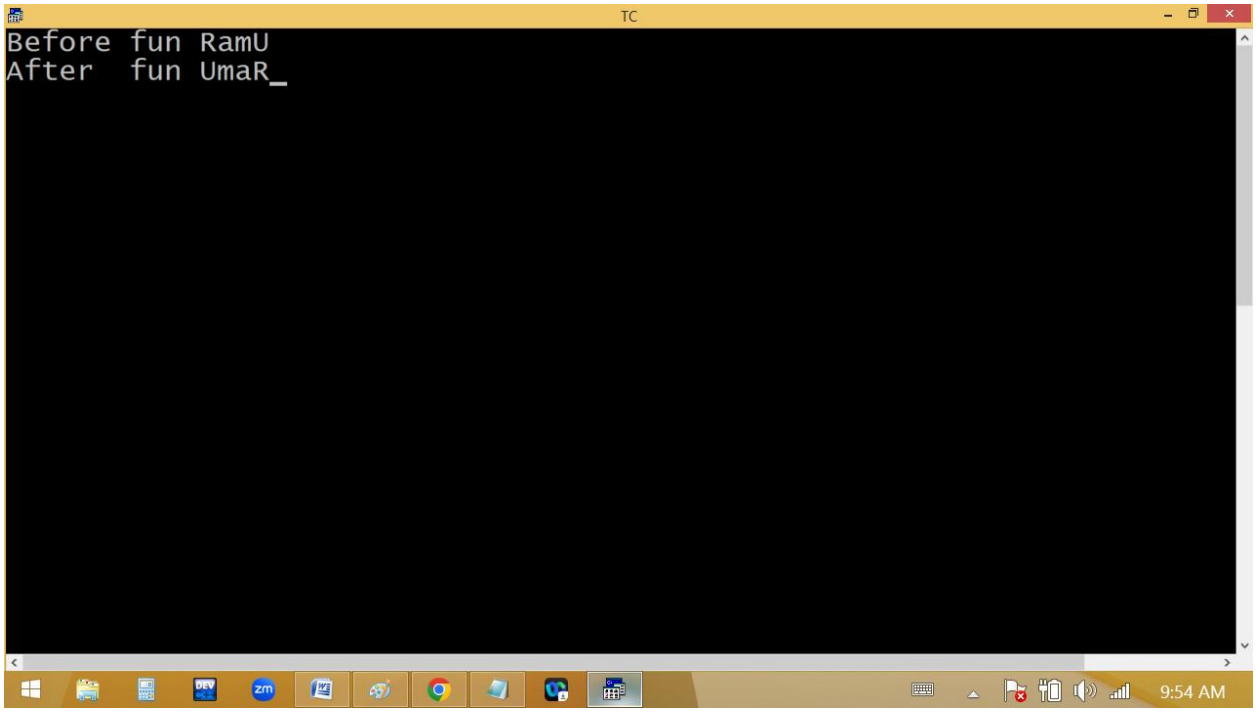
clrscr();

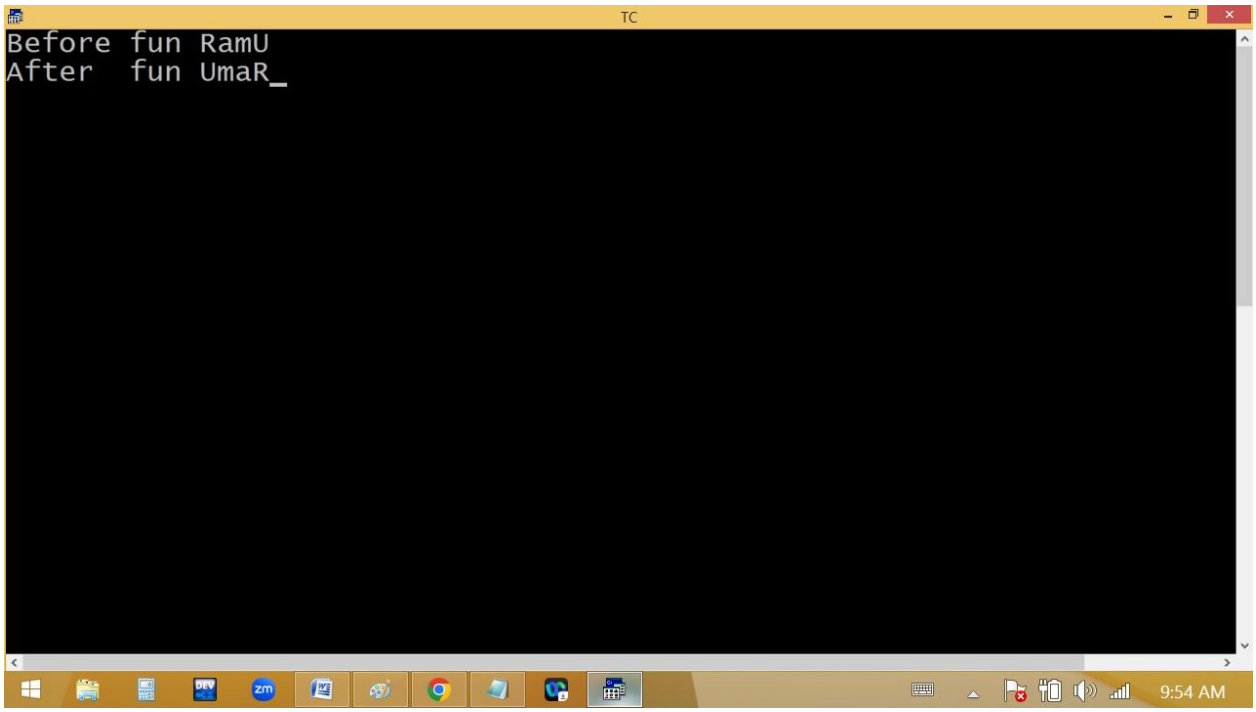
show(a); /* fun calling */

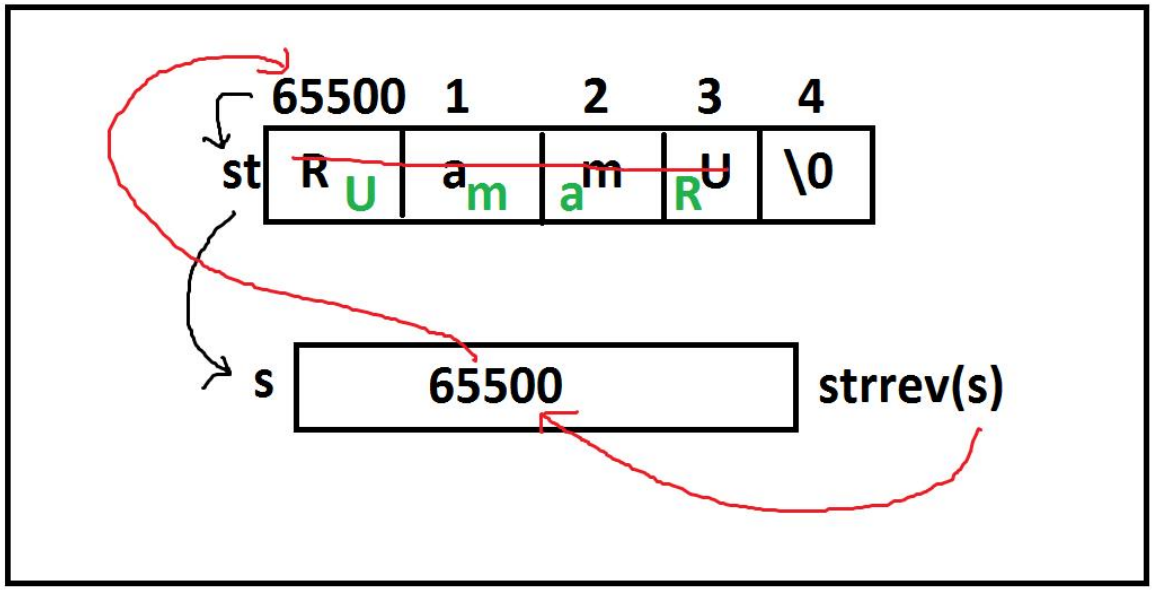
getch();

}




TC
Before fun RamU
After fun UmaR_
9:54 AM


TC
Before fun RamU
After fun UmaR_
9:54 AM


65500
1
2
3
4
st
R
U
a
m
a
m
R
U
\0
s
65500
strrev(s)

TC

File Edit Run Compile Project Options Debug Break/wat

Edit

Line 3 Col 17 Insert Indent Tab Fill Unindent * E:9AM.C

```
#include<stdio.h>
#include<conio.h>
void array(int *a) /* fun definition */
{
a[0]=100;
}
void main()
{
int a[3]={10,20,30};
clrscr();
printf("Before fun a[0] %d\n",a[0]);
array(&a); /* fun calling */
printf("After  fun a[0] %d",a[0]);
getch();
}
```

Watch

9:56 AM

TC

```
Before fun a[0] 10
After  fun a[0] 100_
```

9:56 AM

TC

File Edit Run Compile Project Options Debug Break/wat

Edit

Line 3 Col 20 Insert Indent Tab Fill Unindent * E:9AM.C

```
#include<stdio.h>
#include<conio.h>
void array(int (*a)[3]) /* fun definition */
{
a[0][0]=100;
}
void main()
{
int a[2][3]={10,20,30};
clrscr();
printf("Before fun a[0][0] %d\n",a[0][0]);
array(a); /* fun calling */
printf("After  fun a[0][0] %d",a[0][0]);
getch();
}
```

Watch

10:00 AM

TC

```
Before fun a[0][0] 10
After  fun a[0][0] 100
```

10:00 AM

Function returning address [pointer]:

```
File   Edit   Run   Compile   Project   Options   Debug   Break/wat
                              Edit
     Line 3     Col 44  Insert Indent Tab Fill Unindent * E:9AM.C
#include<stdio.h>
#include<conio.h>
int  max(int a, int b) /* fun definition */_
{
return a>b?a:b;
}
void main()
{
int a,b;
clrscr();
printf("Enter a, b values "); scanf("%d %d",&a, &b);
printf("%d is big",max(a,b)); /* fun calling */
getch();
}
                              Watch
```

```
Enter a, b values 2 7
7 is big
```

TC

File Edit Run Compile Project Options Debug Break/wat

Edit

Line 13 Col 22 Insert Indent Tab Fill Unindent * E:9AM.C

```
#include<stdio.h>
#include<conio.h>
int * max(int *a, int *b) /* fun definition */
{
return *a>*b?a:b;
}
void main()
{
int a,b, *p;
clrscr();
printf("Enter a, b values "); scanf("%d %d",&a, &b);
p = max(&a, &b); /* fun calling */
printf("%d is big",*p); /* fun calling */
getch();
}
```

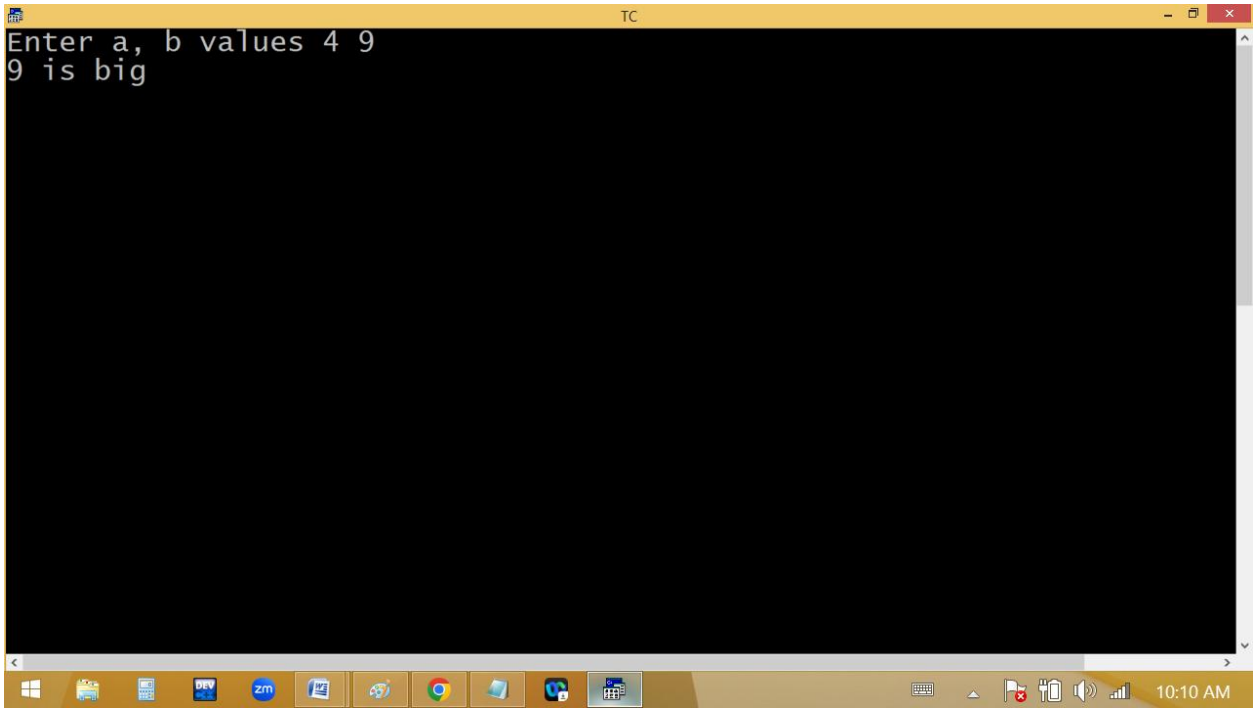
Watch

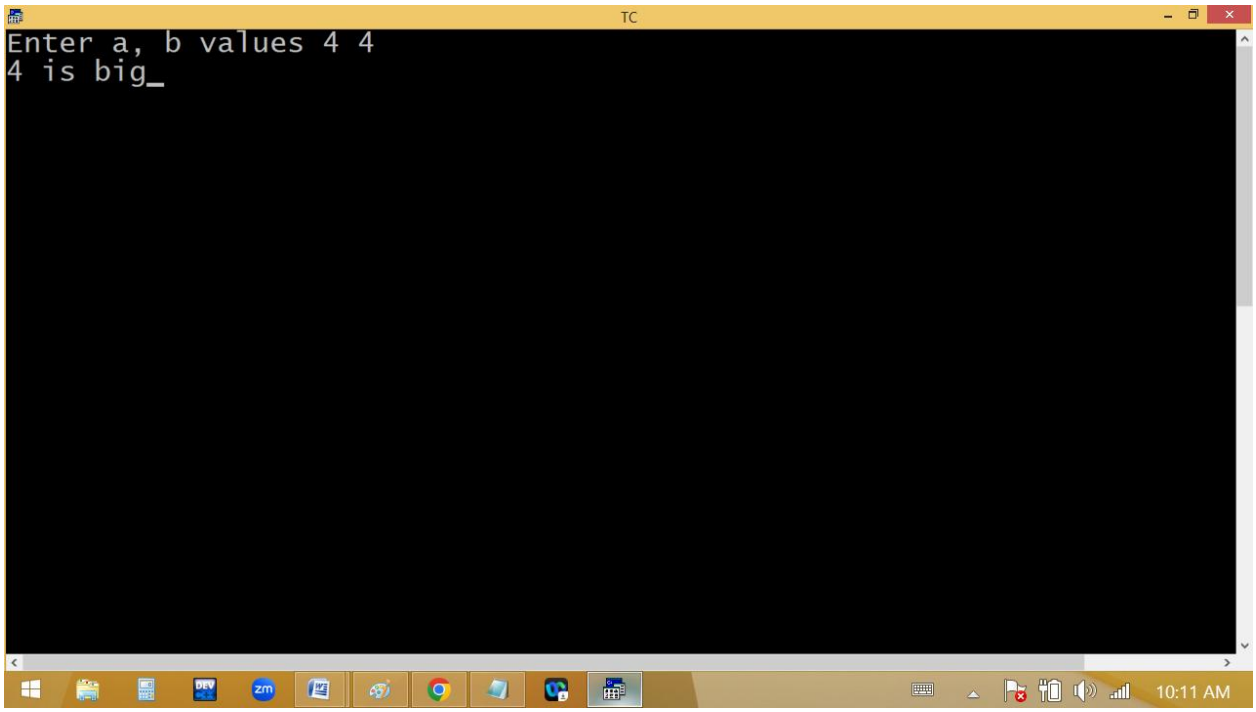
10:10 AM

TC

```
Enter a, b values 5 3
5 is big_
```

10:10 AM


TC
Enter a, b values 4 9
9 is big
10:10 AM


TC
Enter a, b values 4 4
4 is big_
10:11 AM

TC

File Edit Run Compile Project Options Debug Break/wat

Edit

Line 5 Col 24 Insert Indent Tab Fill Unindent * E:9AM.C

```
#include<stdio.h>
#include<conio.h>
char * max(int *a, int *b) /* fun definition */
{
return *a>*b?"a is big":*b>*a?"b is big":"Both are equal";
}
void main()
{
int a,b; char *p;
clrscr();
printf("Enter a, b values "); scanf("%d %d",&a, &b);
p = max(&a, &b); /* fun calling */
printf(p); /* fun calling */
getch();
}
```

Watch

10:13 AM

TC

```
Enter a, b values 4 4
Both are equal_
```

10:13 AM

COMMAND LINE ARGUMENTS

It is the process of sending arguments to main() from command prompt.

It is useful, in designing applications for dos/command based environment.

It allows to create new commands for CUI[Character User Interface] environments.

main() by default having 2 arguments.

1. argc
2. argv

argc is an integer, which counts the no of arguments entered at command prompt, including filename.

argv is a string array(pointer), which stores the arguments entered at command prompt.

The Default no of argument counter is 1.

Eg: Finding n numbers sum using command line arguments:

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void   main(int  argc,  char  *argv[])

{

int i, s=0;

clrscr();

for(i=1;i< argc ;i++) s = s + atoi(argv[i]);

printf("Sum = %d", s);

getch();

}
```

Save file [eg: sum.c] , compile and ctrl+f9

Goto command prompt.

C:\TC> sum 10 20 30

Sum = 60

Eg:

Finding factorial using command line arguments

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void   main(int  argc,  char  *argv[])

{

long int   f=1;

int  n=atoi(argv[1]);

clrscr();

while(n>=1) f=f*n--;

printf("Factorial = %ld", f);
```

```
getch();
}
```

Save file [Eg: fact.c], ctrl+f9

C:\TC> fact 5

Factorial = 120

Eg:

```
/*Finding string length using command line argument*/
#include<stdio.h>
#include<conio.h>
void  main(int  argc,  char *argv[ ])
{
int  i;
for(i=0;argv[1][ i ]!='\0';i++)
{
```

```
printf("%c   ",argv[1][ i ]);
}
printf(" Length = %d", i);
}
```

Save the file [eg: len.c]

compile [alt+F9], run [ctrl+f9]

goto command prompt.

```
C:\TC>len   kishore
k  i  s  h  o  r  e  length = 7
```