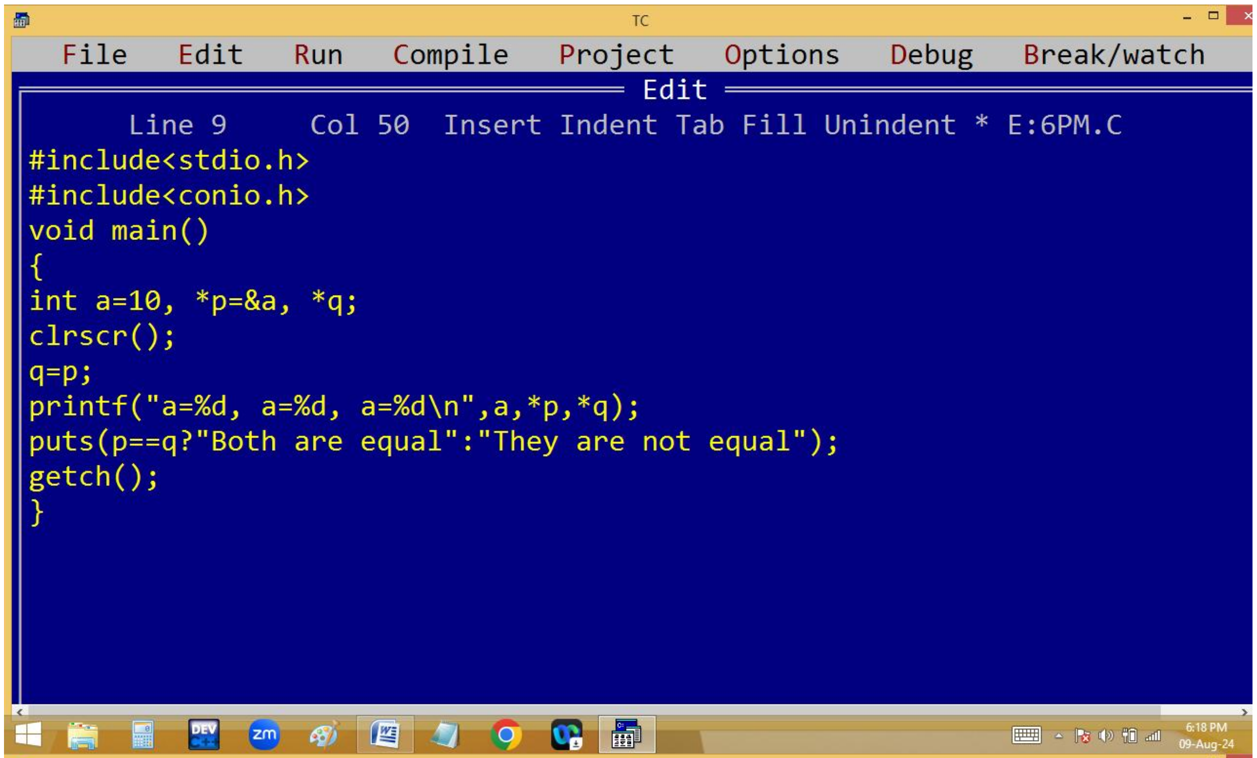
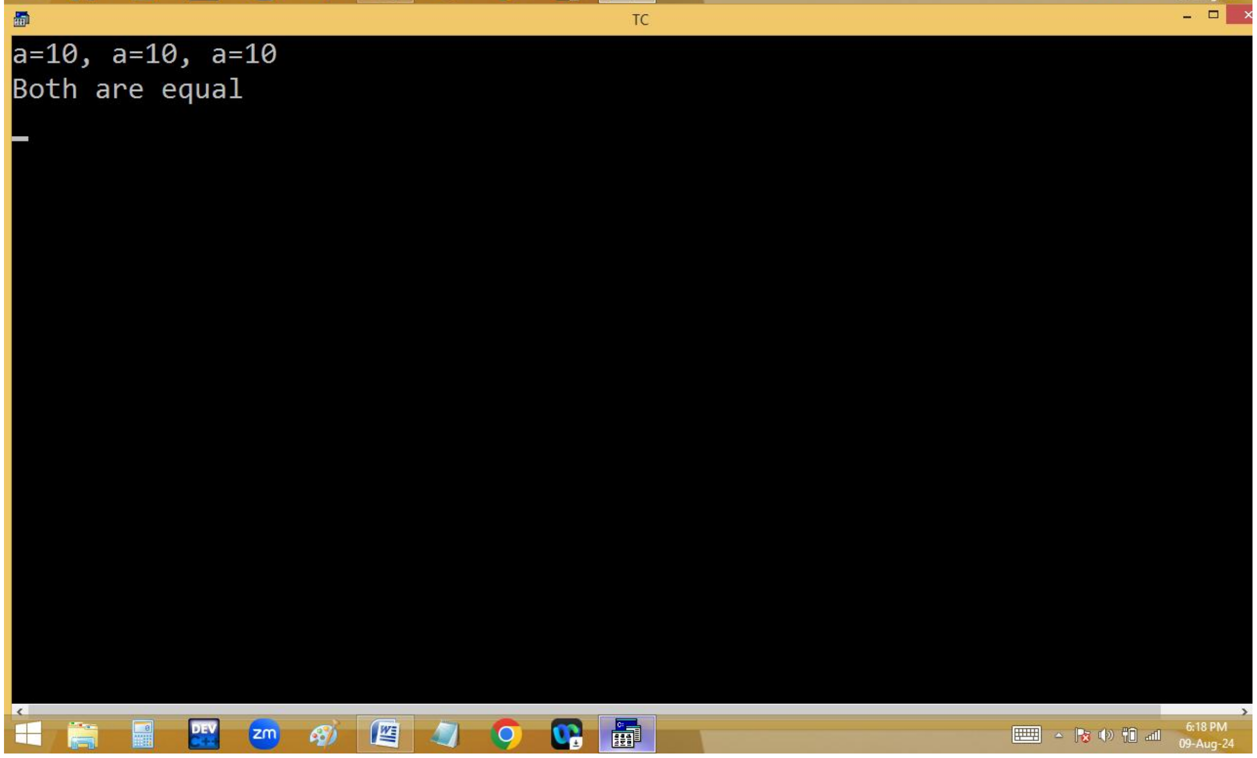
Pointer arithmetic:

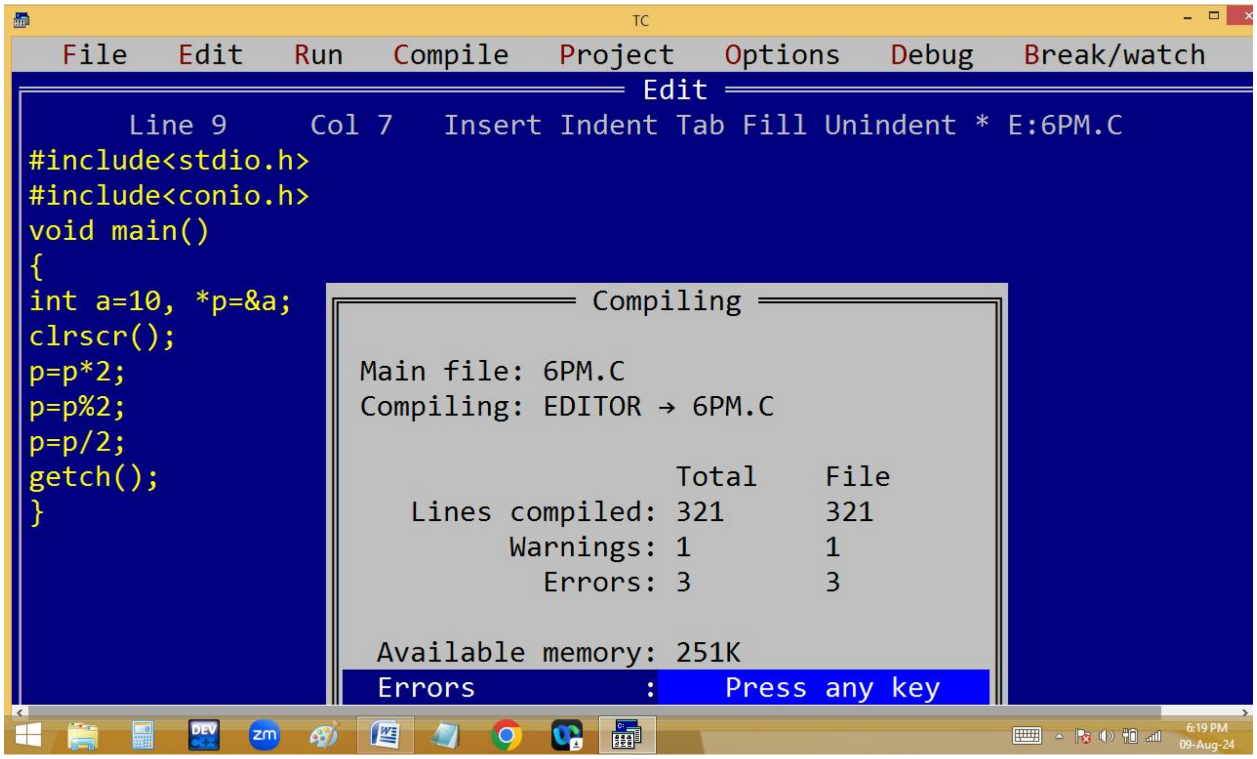
Like normal variables we can do =, ==, +, -, ++, -- on pointers also. But we can't do *, %, / on pointers.


TC
File Edit Run Compile Project Options Debug Break/watch
Edit
Line 9 Col 50 Insert Indent Tab Fill Unindent * E:6PM.C
6:18 PM
09-Aug-24

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=10, *p=&a, *q;
clrscr();
q=p;
printf("a=%d, a=%d, a=%d\n",a,*p,*q);
puts(p==q?"Both are equal":"They are not equal");
getch();
}
```



```
a=10, a=10, a=10
Both are equal
```



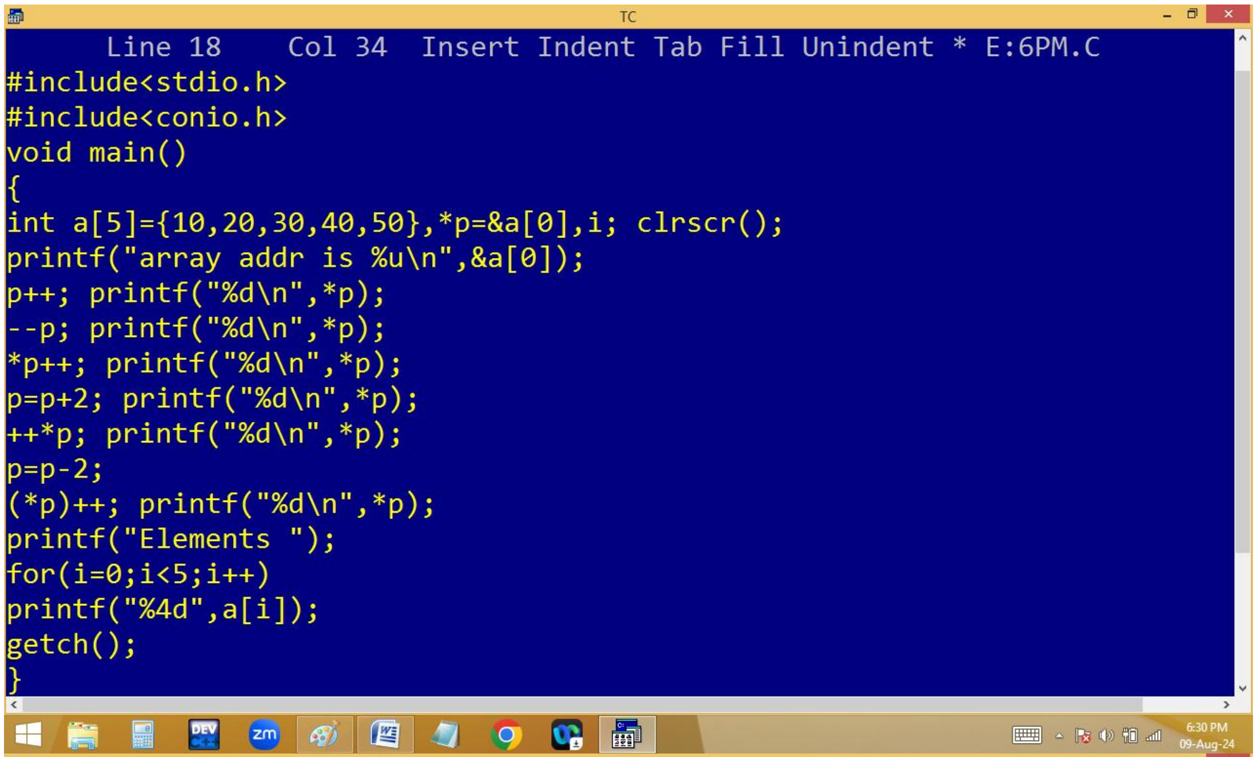
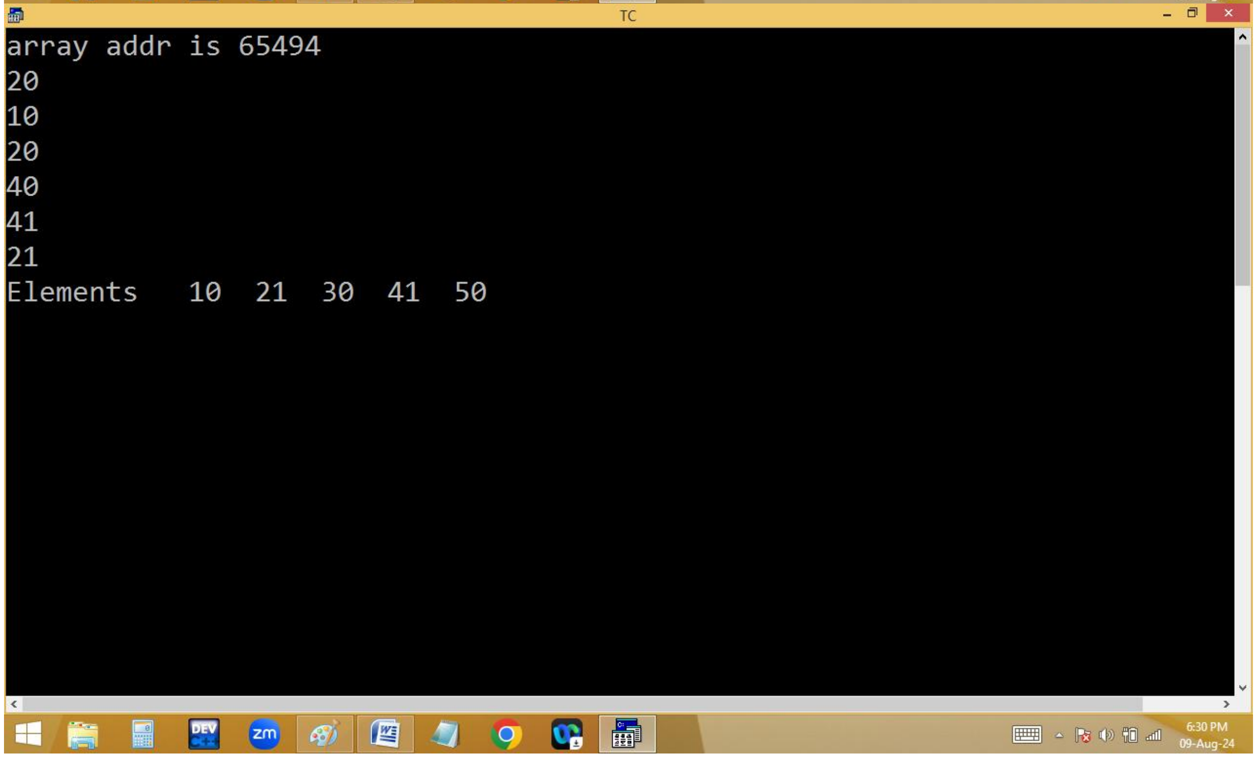
```
 File   Edit   Run   Compile   Project   Options   Debug   Break/watch
                                  Edit
      Line 9     Col 7   Insert Indent Tab Fill Unindent * E:6PM.C
#include<stdio.h>
#include<conio.h>
void main()
{
int a=10, *p=&a;
clrscr();
p=p*2;
p=p%2;
p=p/2;
getch();
}
```

```
                  Compiling

 Main file: 6PM.C
 Compiling: EDITOR → 6PM.C

                      Total     File
    Lines compiled: 321       321
          Warnings: 1         1
            Errors: 3         3

  Available memory: 251K
  Errors          :    Press any key
```


```
      Line 18    Col 34  Insert Indent Tab Fill Unindent * E:6PM.C
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5]={10,20,30,40,50},*p=&a[0],i; clrscr();
printf("array addr is %u\n",&a[0]);
p++; printf("%d\n",*p);
--p; printf("%d\n",*p);
*p++; printf("%d\n",*p);
p=p+2; printf("%d\n",*p);
++*p; printf("%d\n",*p);
p=p-2;
(*p)++; printf("%d\n",*p);
printf("Elements ");
for(i=0;i<5;i++)
printf("%4d",a[i]);
getch();
}
```

```
array addr is 65494
20
10
20
40
41
21
Elements   10  21  30  41  50
```

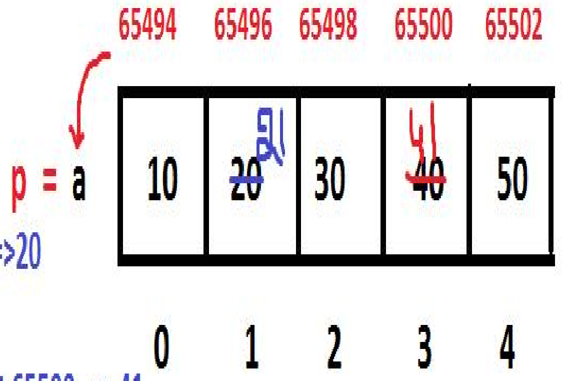
p++ ==> 65494+1*2=65496==>p(*p)==>value at 65496==>20

--p;-->65496-1*2=65494==>p(*p)==>value at 65494=->10

*p++==>p++==>65494+1*2=65496==>*p==>value at 65496 ==>20

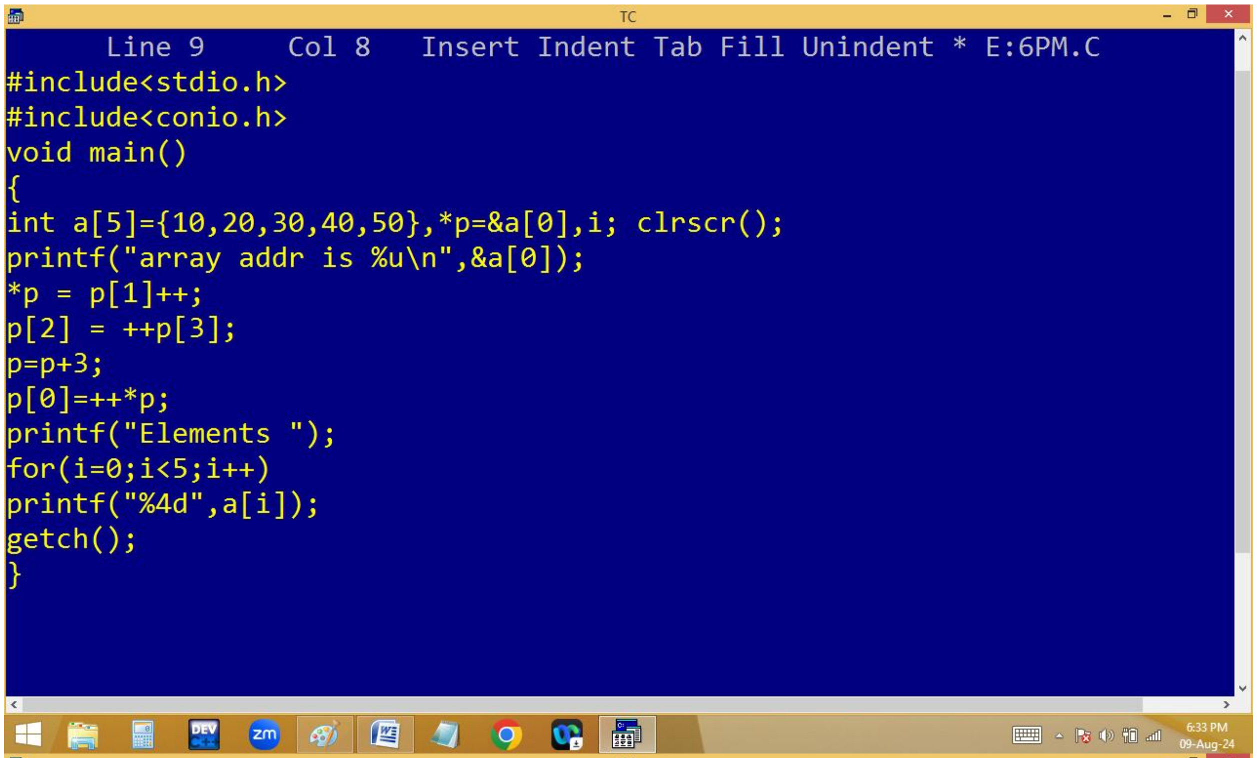
p=p+2==>65496+2*2=65500==>*p==>value at 65500==>40

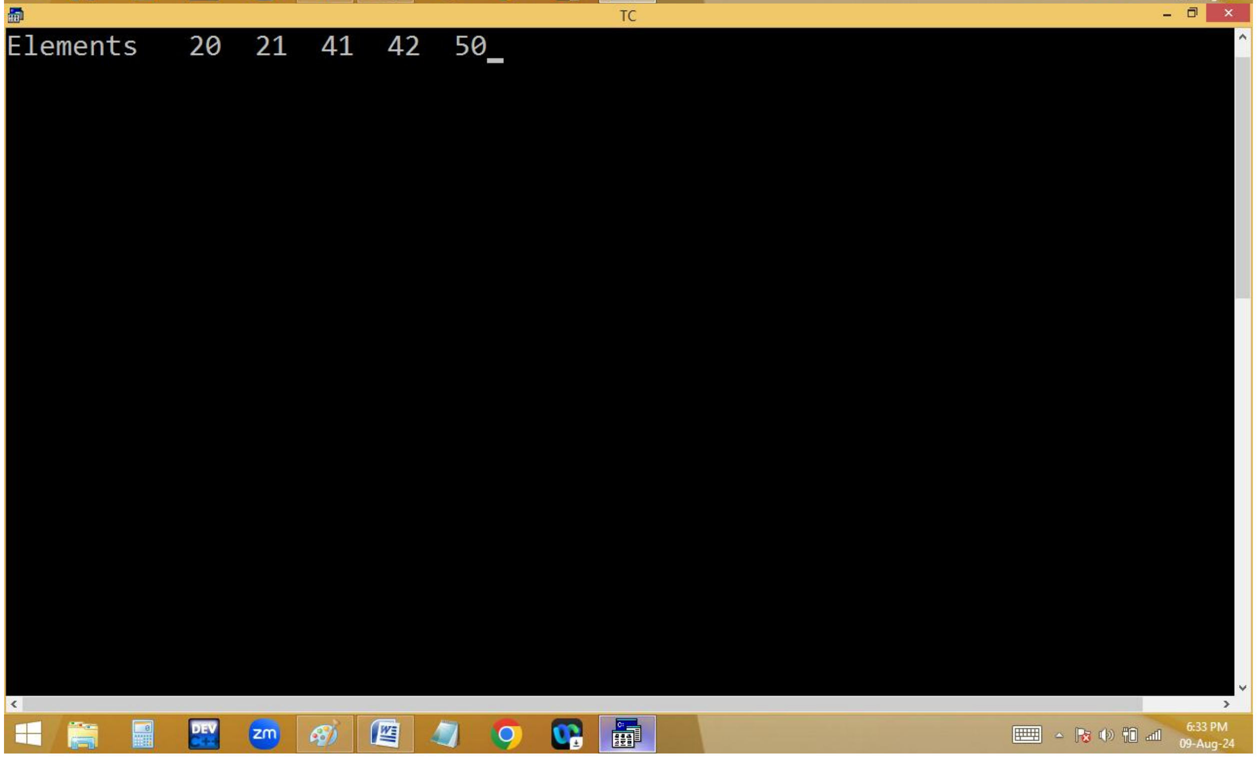
++*p==>++ of value at 65500==>40++==>41 ==>*p==>value at 65500==> 41

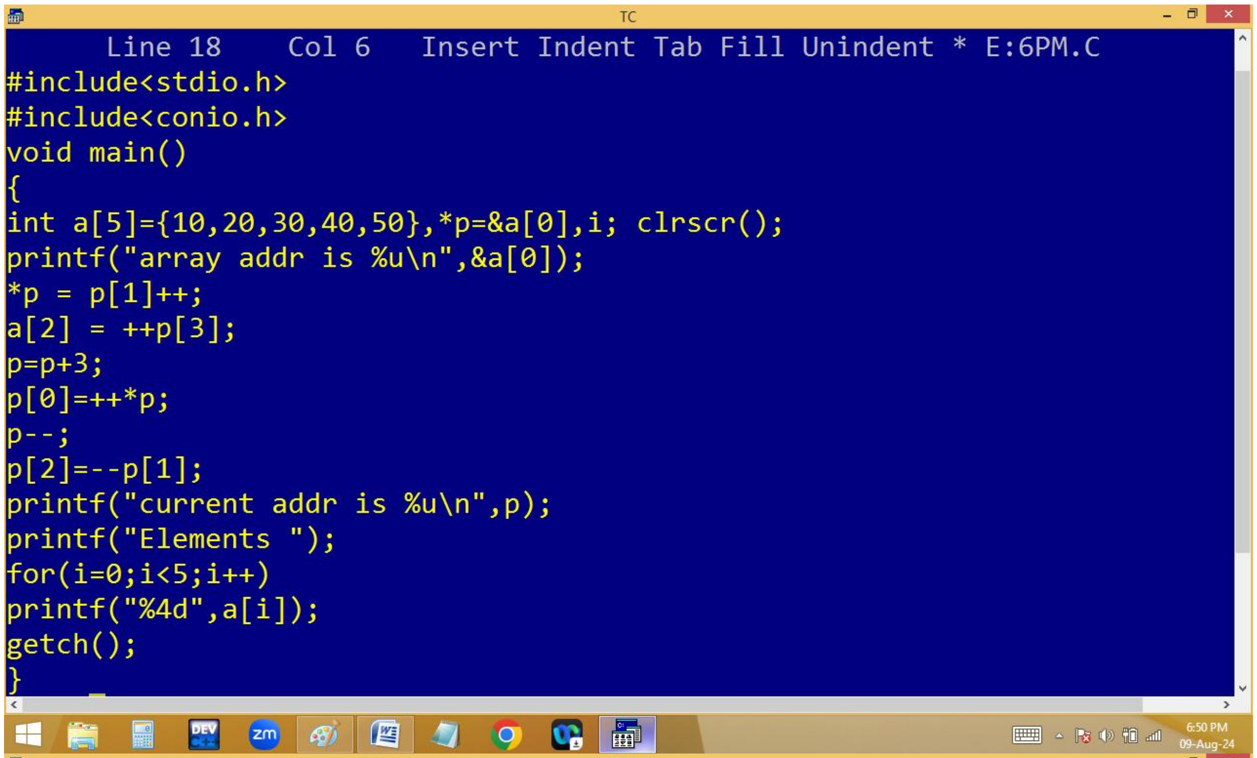


p=p-2==>65500-2*2=65496==> (*p)++ ==> value at 65496++ ==> 20++ ==> 21

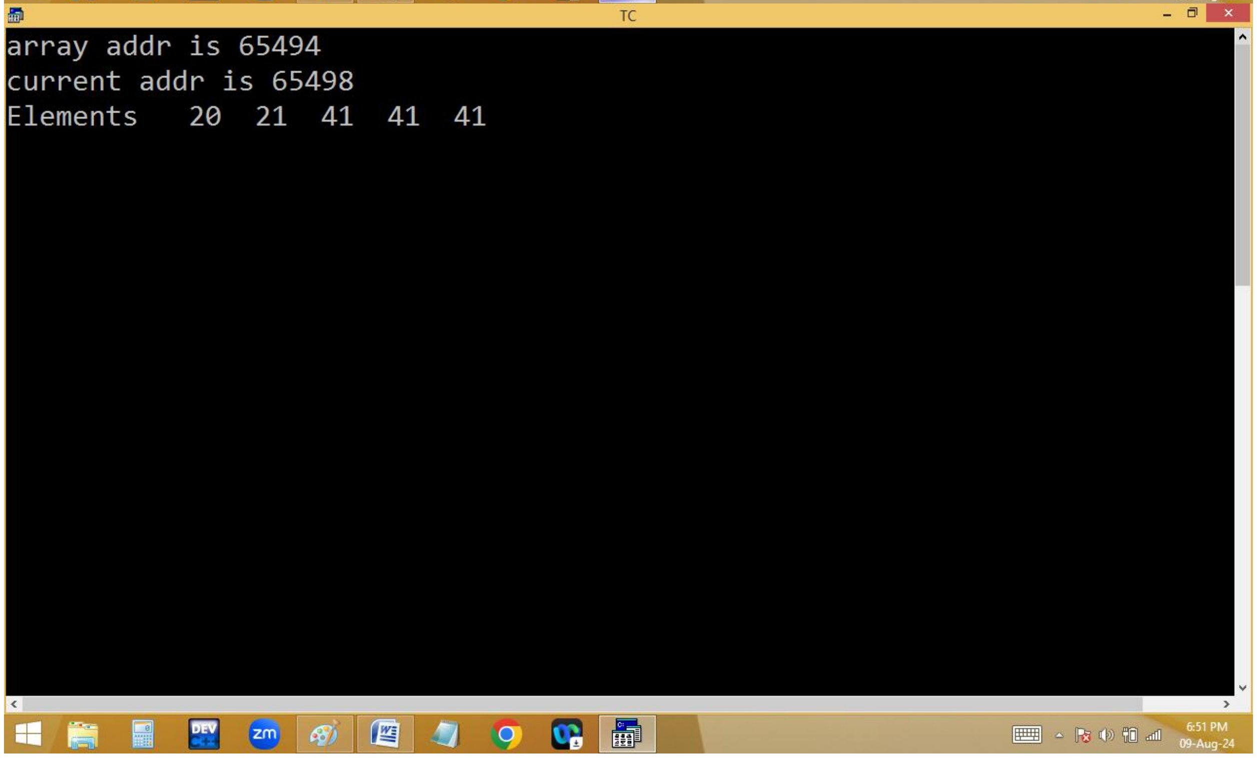
p(*p) ==> value at 65496 ==> 21


```
      Line 9     Col 8   Insert Indent Tab Fill Unindent * E:6PM.C
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5]={10,20,30,40,50},*p=&a[0],i; clrscr();
printf("array addr is %u\n",&a[0]);
*p = p[1]++;
p[2] = ++p[3];
p=p+3;
p[0]=++*p;
printf("Elements ");
for(i=0;i<5;i++)
printf("%4d",a[i]);
getch();
}
```

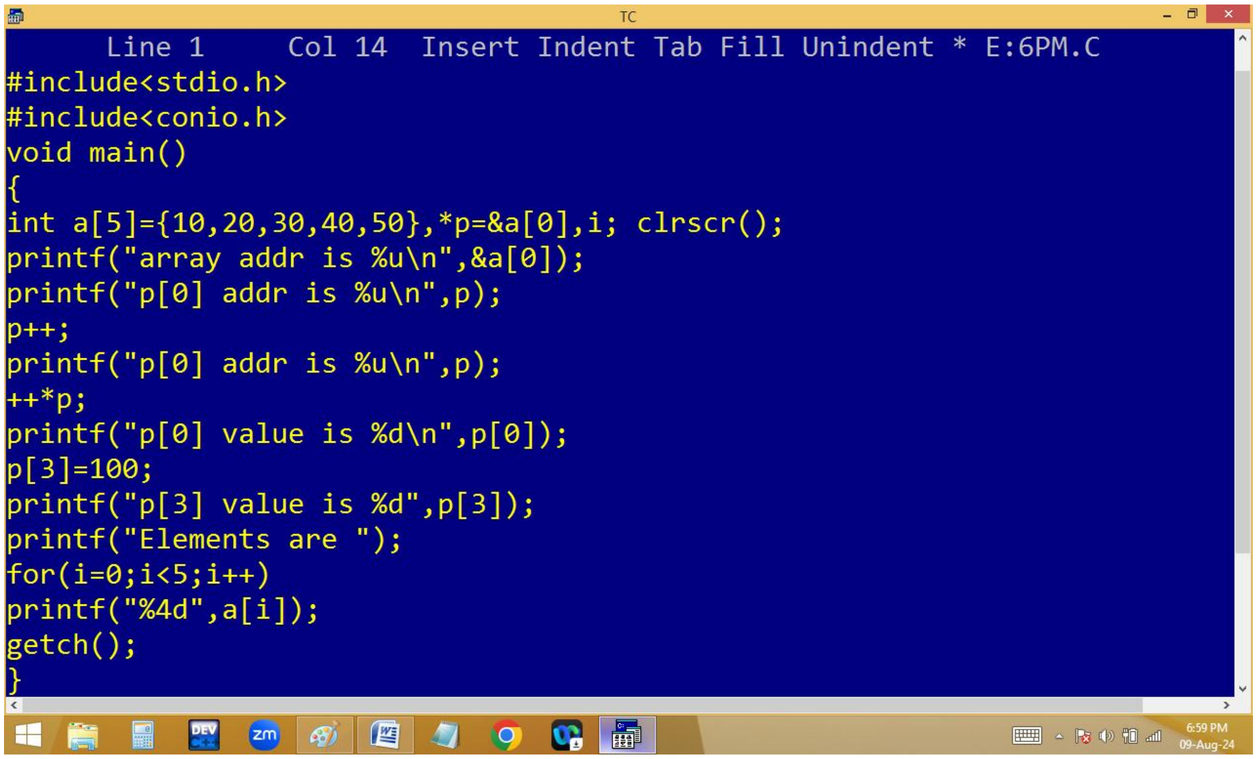

```
Elements   20  21  41  42  50_
```



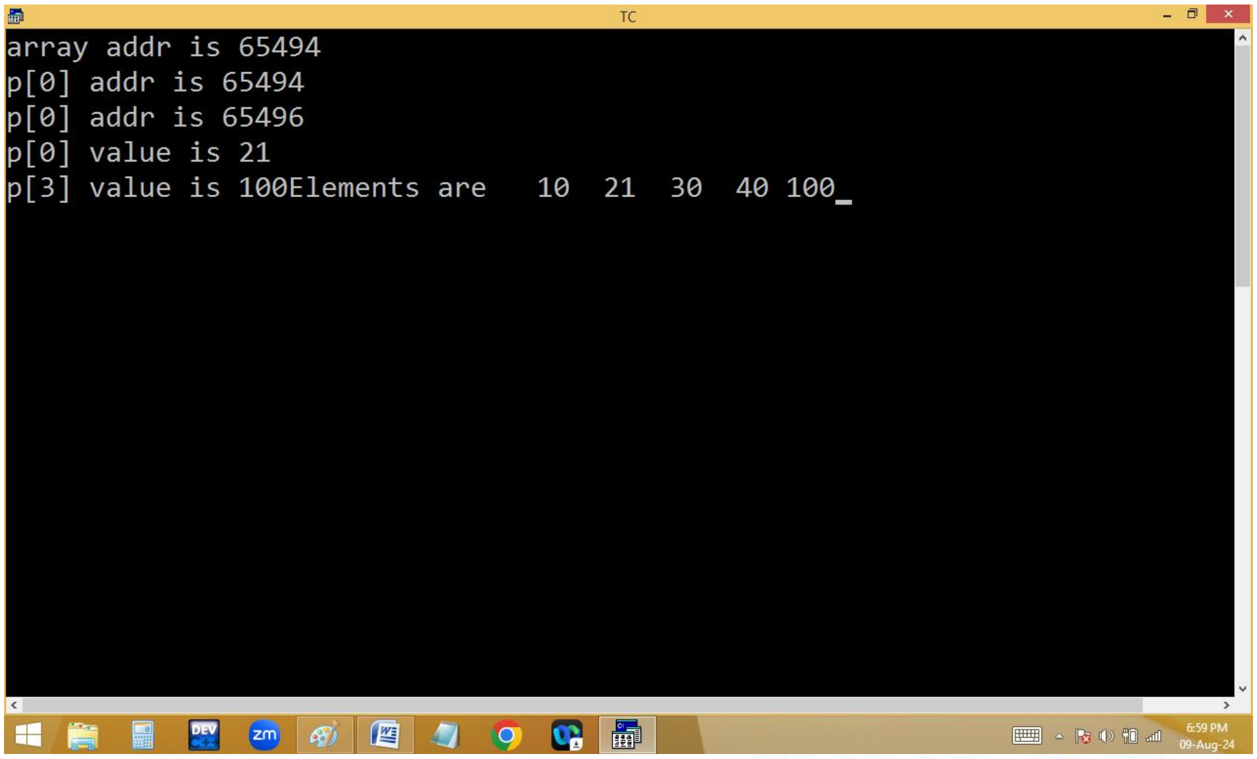
```
      Line 18    Col 6   Insert Indent Tab Fill Unindent * E:6PM.C
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5]={10,20,30,40,50},*p=&a[0],i; clrscr();
printf("array addr is %u\n",&a[0]);
*p = p[1]++;
a[2] = ++p[3];
p=p+3;
p[0]=++*p;
p--;
p[2]=--p[1];
printf("current addr is %u\n",p);
printf("Elements ");
for(i=0;i<5;i++)
printf("%4d",a[i]);
getch();
}
```



```
array addr is 65494
current addr is 65498
Elements   20   21   41   41   41
```

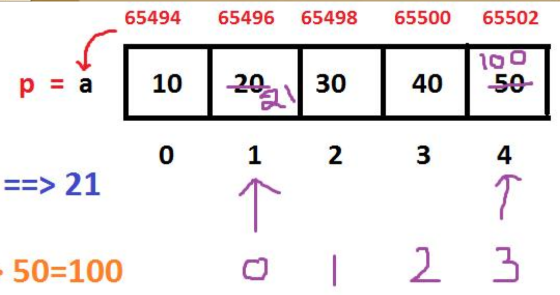


```
      Line 1     Col 14  Insert Indent Tab Fill Unindent * E:6PM.C
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5]={10,20,30,40,50},*p=&a[0],i; clrscr();
printf("array addr is %u\n",&a[0]);
printf("p[0] addr is %u\n",p);
p++;
printf("p[0] addr is %u\n",p);
++*p;
printf("p[0] value is %d\n",p[0]);
p[3]=100;
printf("p[3] value is %d",p[3]);
printf("Elements are ");
for(i=0;i<5;i++)
printf("%4d",a[i]);
getch();
}
```



```
array addr is 65494
p[0] addr is 65494
p[0] addr is 65496
p[0] value is 21
p[3] value is 100Elements are   10  21  30  40 100_
```

p= 65494
p++==> 65494+1*2=p=65496

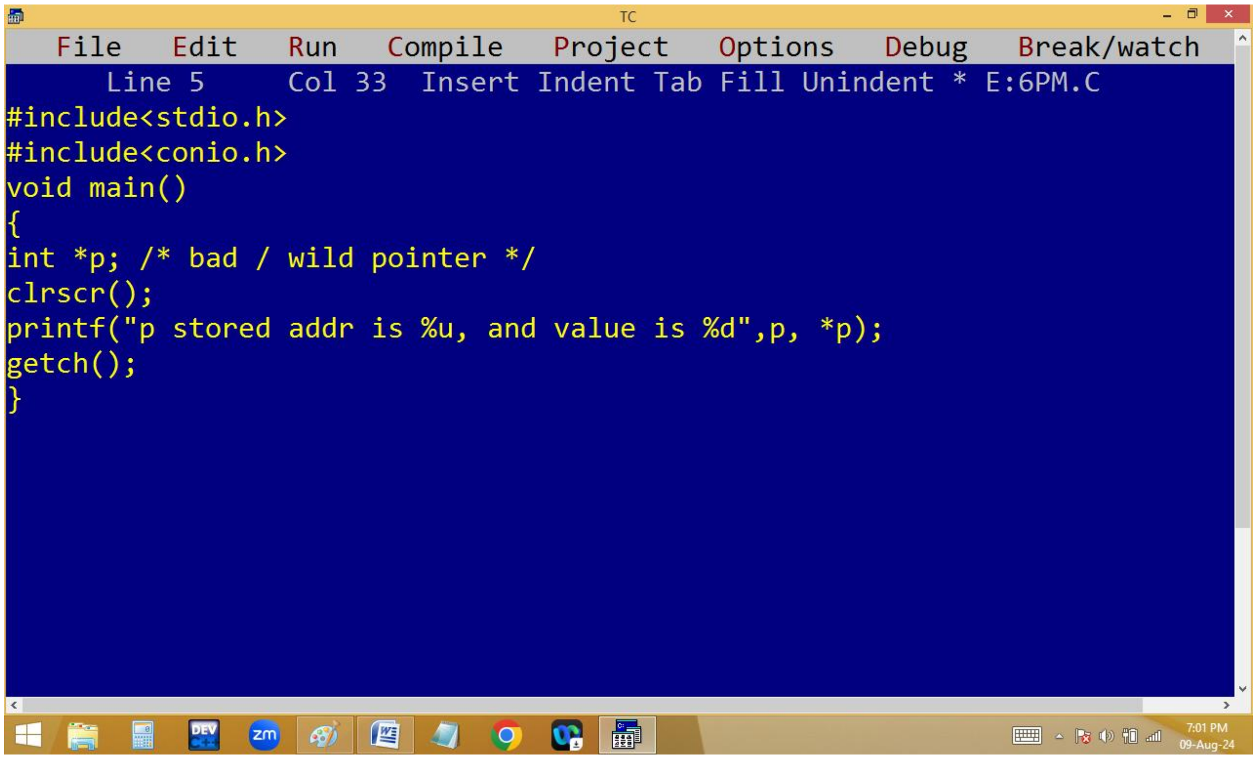


++*p ==> ++ of value at 65496 ==> ++20 ==> 21

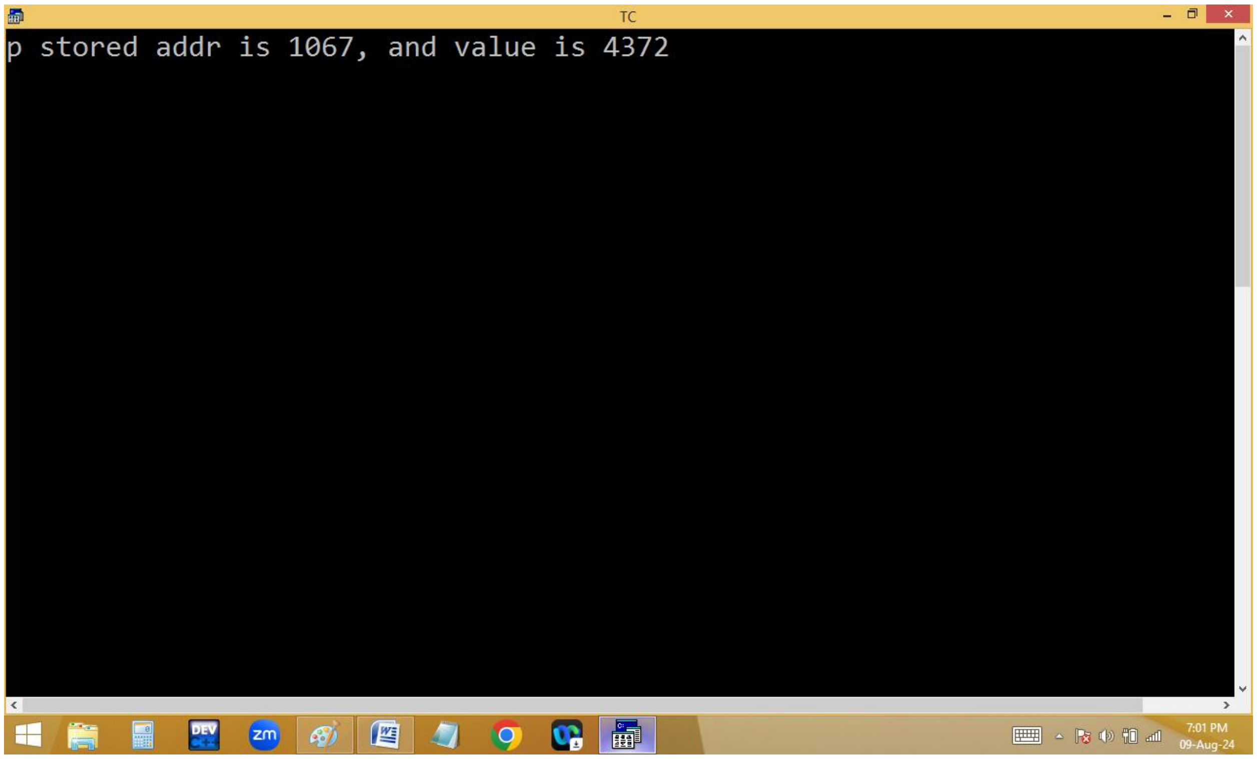
p[3]=100==> 65496+3*2=65502=100==> 50=100

Bad / wild pointer:

A pointer is declared but not initialized. In this situation the pointer is storing some unknown address and value. This kind of pointer is called bad or wild pointer.

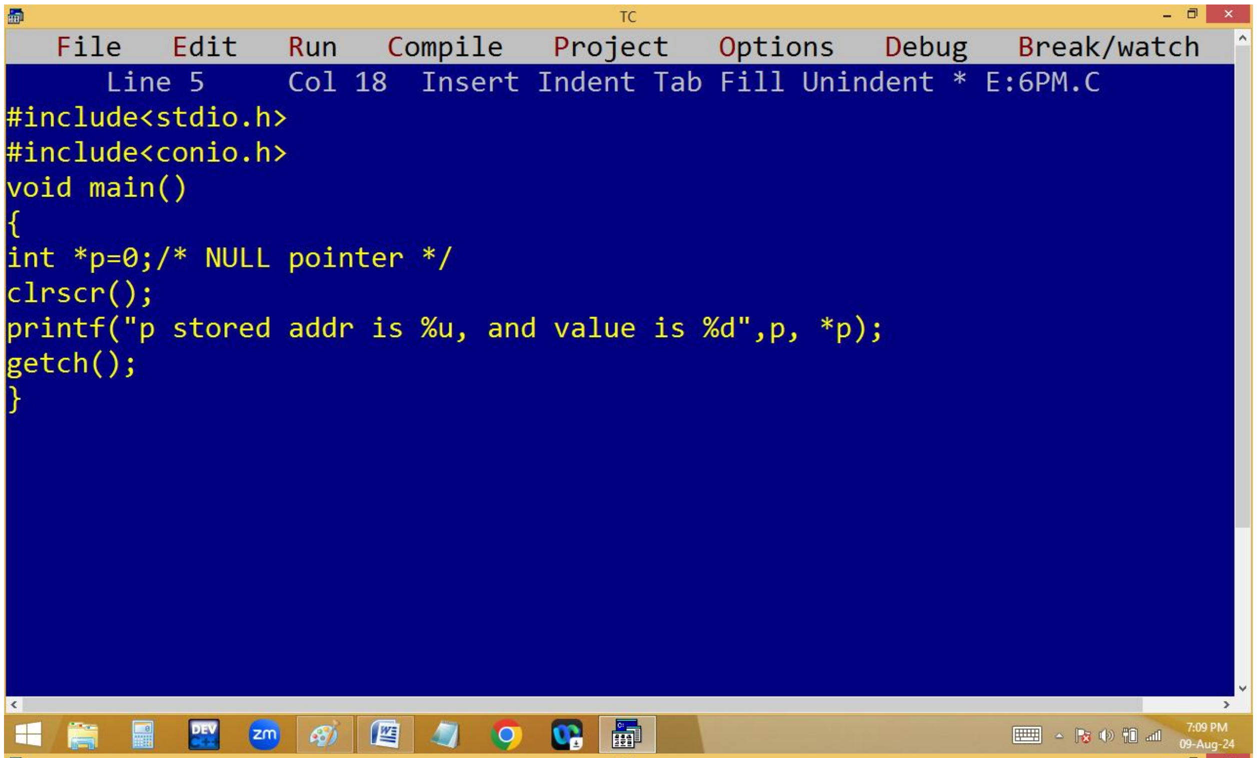

TC
File Edit Run Compile Project Options Debug Break/watch
Line 5 Col 33 Insert Indent Tab Fill Unindent * E:6PM.C
7:01 PM
09-Aug-24

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *p; /* bad / wild pointer */
clrscr();
printf("p stored addr is %u, and value is %d",p, *p);
getch();
}
```

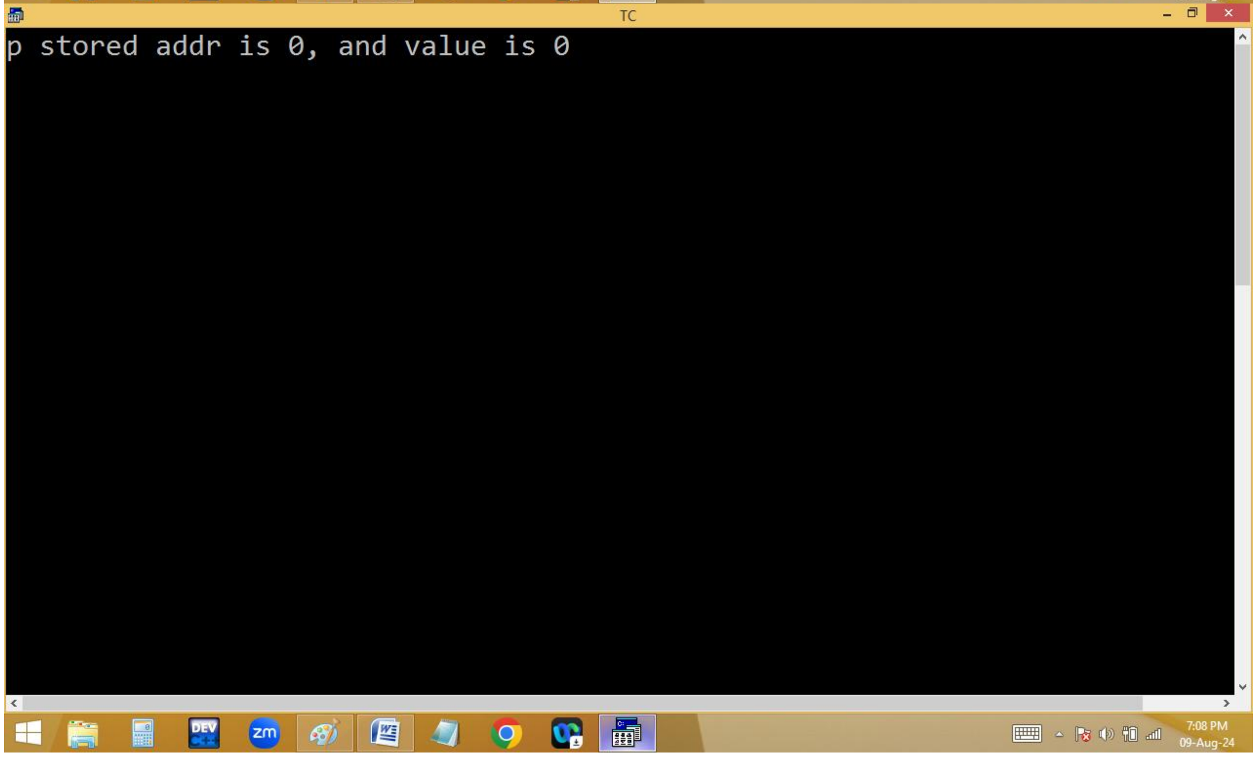


```
p stored addr is 1067, and value is 4372
```

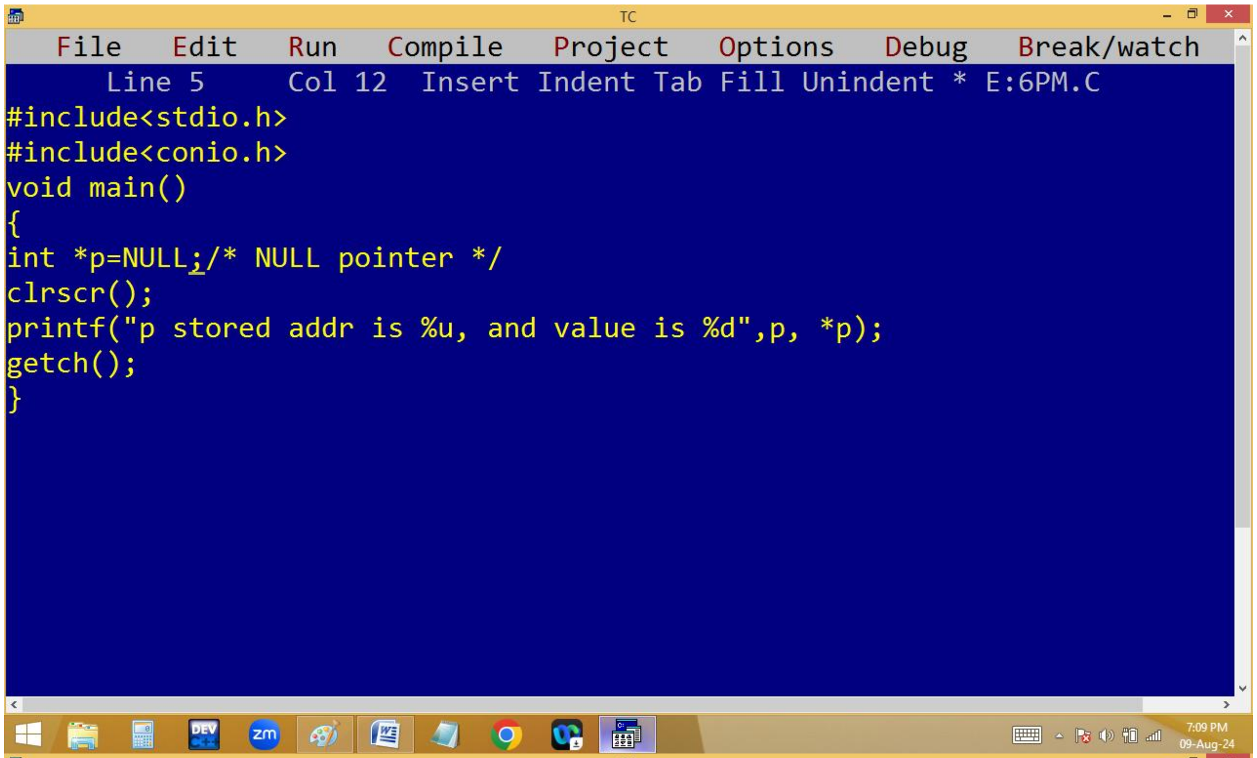
NULL pointer: when a pointer is initialized with NULL / 0 then it is called NULL pointer. To avoid bad and dangling pointers we are using NULL pointer.


TC
File Edit Run Compile Project Options Debug Break/watch
Line 5 Col 18 Insert Indent Tab Fill Unindent * E:6PM.C

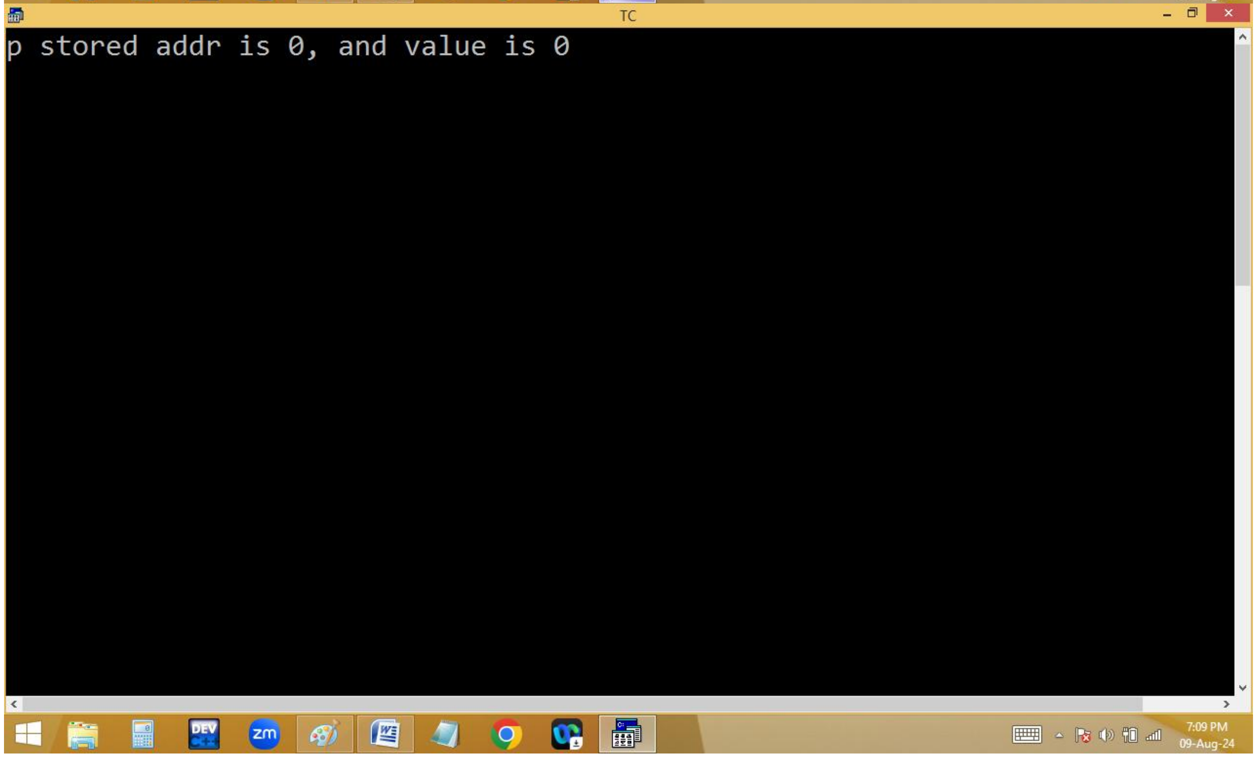
```
#include<stdio.h>
#include<conio.h>
void main()
{
int *p=0;/* NULL pointer */
clrscr();
printf("p stored addr is %u, and value is %d",p, *p);
getch();
}
```



```
p stored addr is 0, and value is 0
```

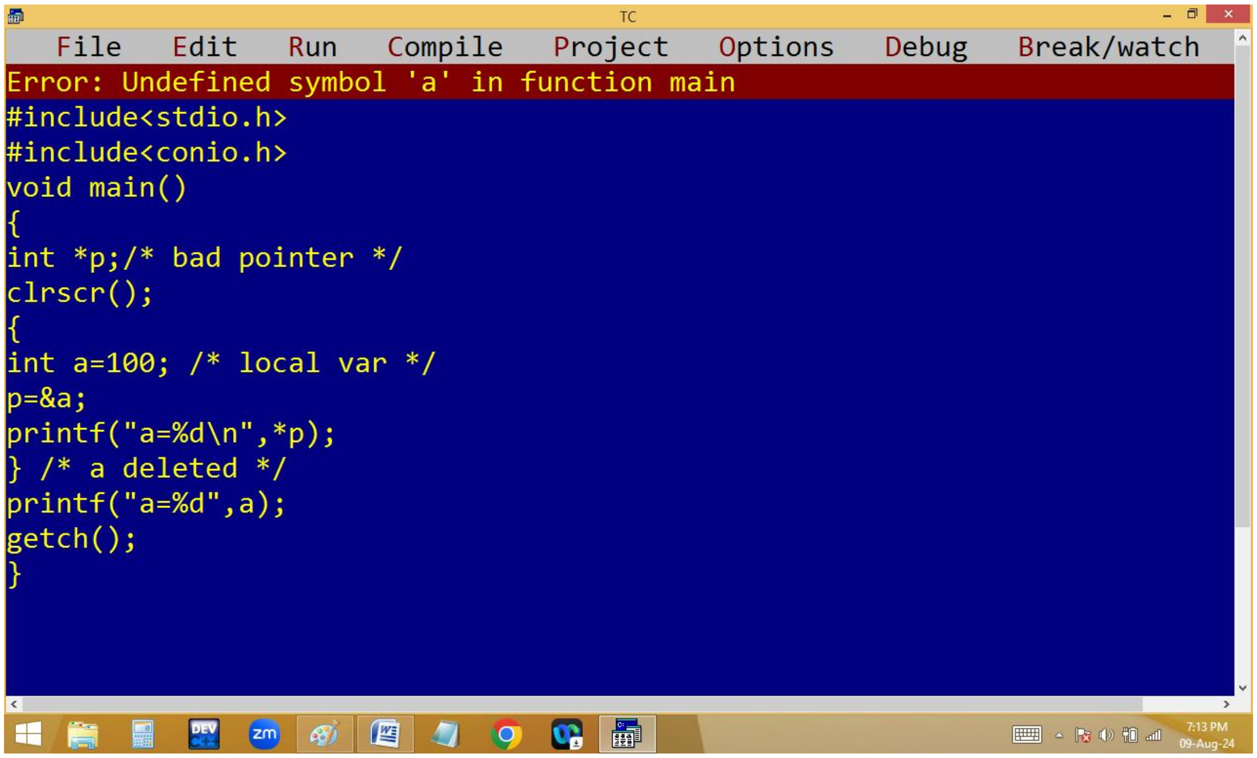

TC
File Edit Run Compile Project Options Debug Break/watch
Line 5 Col 12 Insert Indent Tab Fill Unindent * E:6PM.C
7:09 PM
09-Aug-24

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *p=NULL;/* NULL pointer */
clrscr();
printf("p stored addr is %u, and value is %d",p, *p);
getch();
}
```

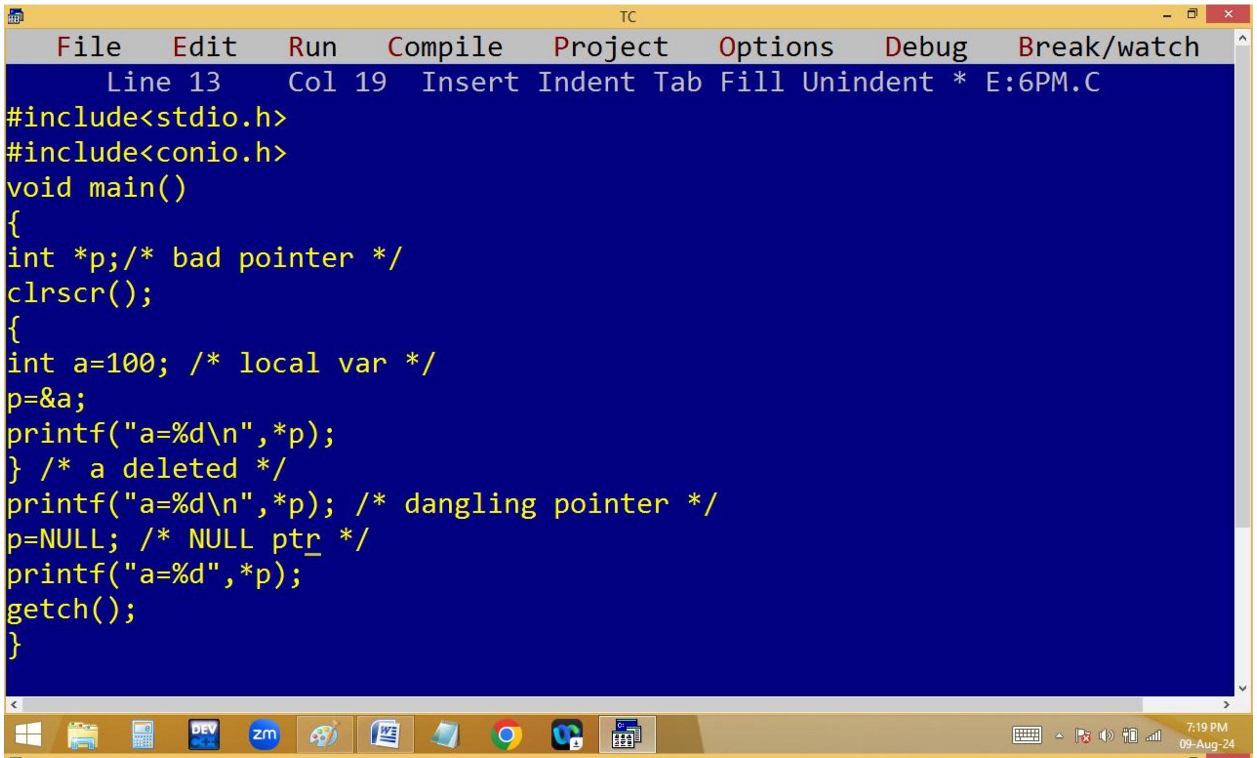


```
p stored addr is 0, and value is 0
```

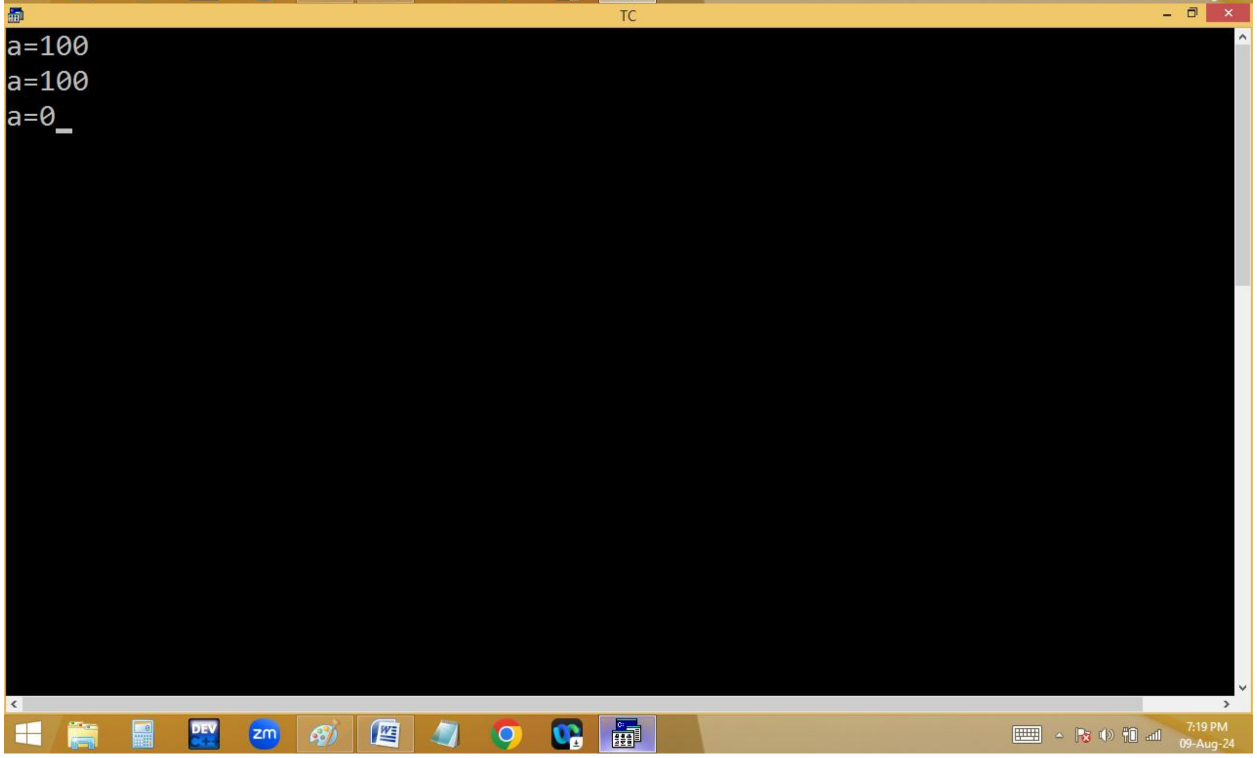
Dangling pointer: A pointer is declared and initialized with a variable address. After some time that variable deleted from memory. But the pointer is still showing the deleted variable value. This kind of pointer is called dangling pointer and to avoid this initialize with NULL pointer.



```
Error: Undefined symbol 'a' in function main
#include<stdio.h>
#include<conio.h>
void main()
{
int *p;/* bad pointer */
clrscr();
{
int a=100; /* local var */
p=&a;
printf("a=%d\n",*p);
} /* a deleted */
printf("a=%d",a);
getch();
}
```



```
File    Edit    Run    Compile    Project    Options    Debug    Break/watch
      Line 13     Col 19   Insert Indent Tab Fill Unindent * E:6PM.C
#include<stdio.h>
#include<conio.h>
void main()
{
int *p;/* bad pointer */
clrscr();
{
int a=100; /* local var */
p=&a;
printf("a=%d\n",*p);
} /* a deleted */
printf("a=%d\n",*p); /* dangling pointer */
p=NULL; /* NULL ptr */
printf("a=%d",*p);
getch();
}
```

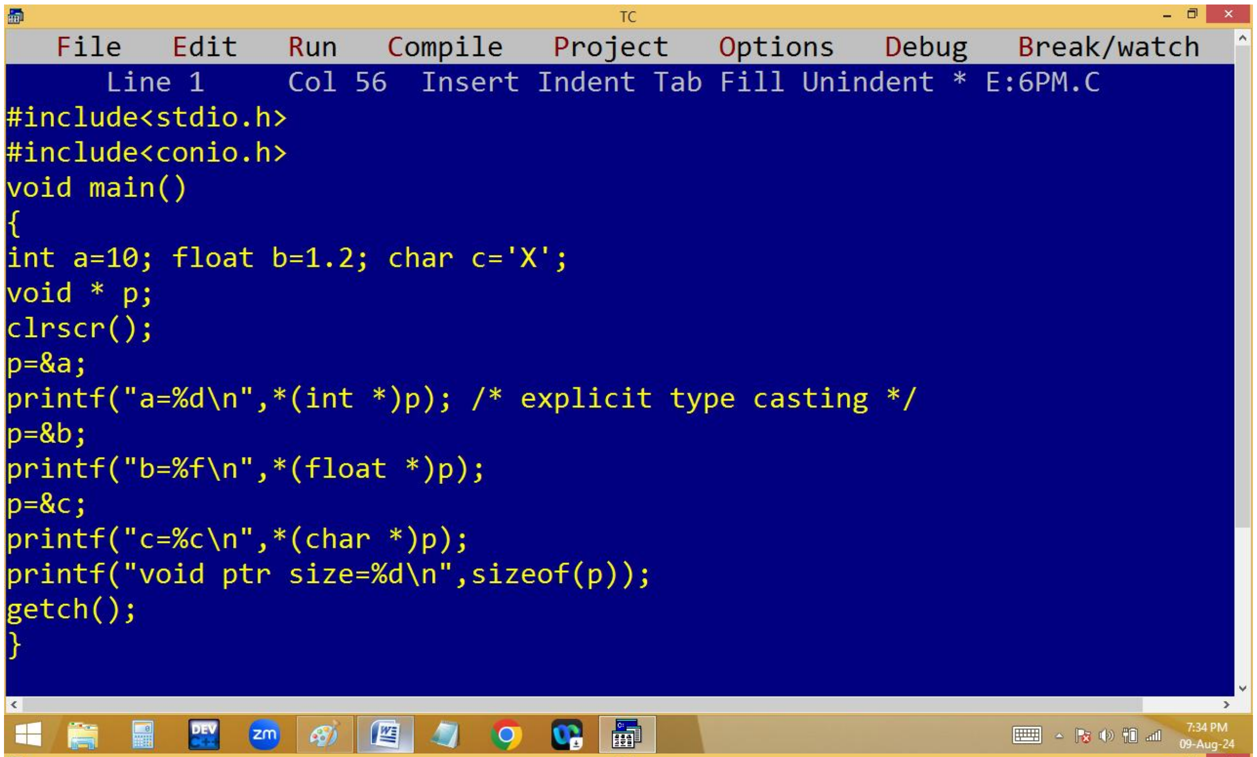


```
a=100
a=100
a=0_
```

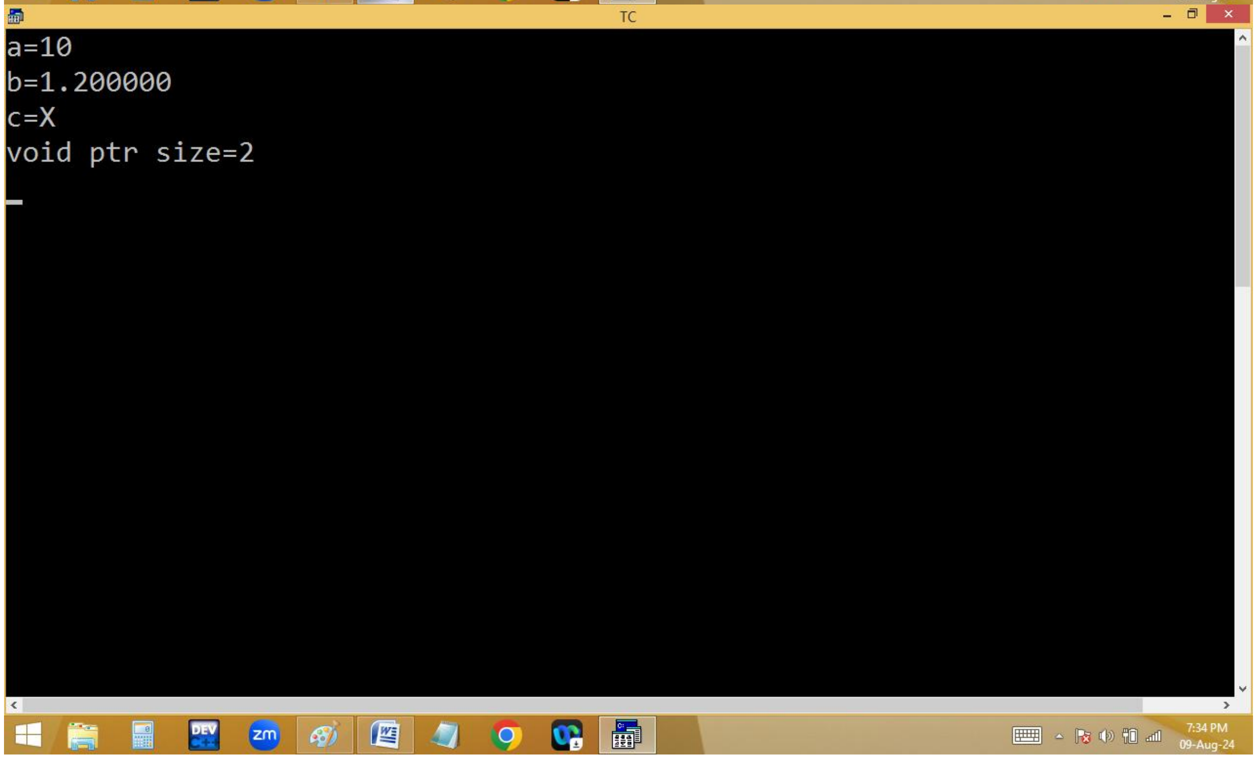
void / generic pointer: pointer can store same type of address only. But when several variables with different data types, we have to define several pointers.

Void pointer can store any type of address. But before going to use void pointer, explicit type casting should be done.

Void pointer takes 2 bytes memory. It is very much used in dynamic memory allocation.


TC
File Edit Run Compile Project Options Debug Break/watch
Line 1 Col 56 Insert Indent Tab Fill Unindent * E:6PM.C
7:34 PM
09-Aug-24

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=10; float b=1.2; char c='X';
void * p;
clrscr();
p=&a;
printf("a=%d\n",*(int *)p); /* explicit type casting */
p=&b;
printf("b=%f\n",*(float *)p);
p=&c;
printf("c=%c\n",*(char *)p);
printf("void ptr size=%d\n",sizeof(p));
getch();
}
```



```
a=10
b=1.200000
c=X
void ptr size=2
```