Memory management

To store anything in our computer, we should have to allocate the memory first.

This memory allocation is conducted in two ways.

1. Static memory allocation.
2. Dynamic memory allocation.

In static memory allocation, the memory is specified at compile/design time, based on the data type or array size. This type of memory management is called compile time memory management [**compiler indicates memory and O.S allocates the memory**].

In static memory allocation, the memory size is fixed at compile time and we can't change this memory size at run time. It causes some times memory wastage / shortage.

To avoid this problem, the only solution is dynamic memory allocation.

In dynamic memory allocation, the memory is allocated at run time, based on the user input, instantly.
This type of memory management is called run time memory management.

To conduct dynamic memory allocation, we should have to use **pointers**.

In dynamic memory allocation the memory is allocated in **HEAP** area.

To manage the dynamic memory, we are using some predefined functions like

- malloc()
- calloc()
- realloc()
- free()

All these functions are available in **<alloc.h>**

malloc(), realloc(), calloc() functions are able to allocate the memory of **64KB** Maximum at a time.

To allocate more than 64KB memory, use the functions

- farmalloc()
- farcalloc()
- farrealloc().

Note:

when we are working with dynamic memory allocation, we have to allocate the memory for any data type. Due to this all these functions return datatype is **void ***, which is a generic type. Due to this we should have to provide **explicit type casting** for all these functions.

malloc()	**calloc()**
Memory allocation	Contiguous memory allocation
Allocates memory in bytes form	Allocates memory in blocks form.
Initial values garbage	Initial values 0
One argument required	Two arguments required
Used for normal variables	Used for array type variables

Syntax:

void * malloc(bytes);

void * calloc(no of blocks, block_size);

free(): It is used to release the memory allocated by malloc(), calloc() and realloc().

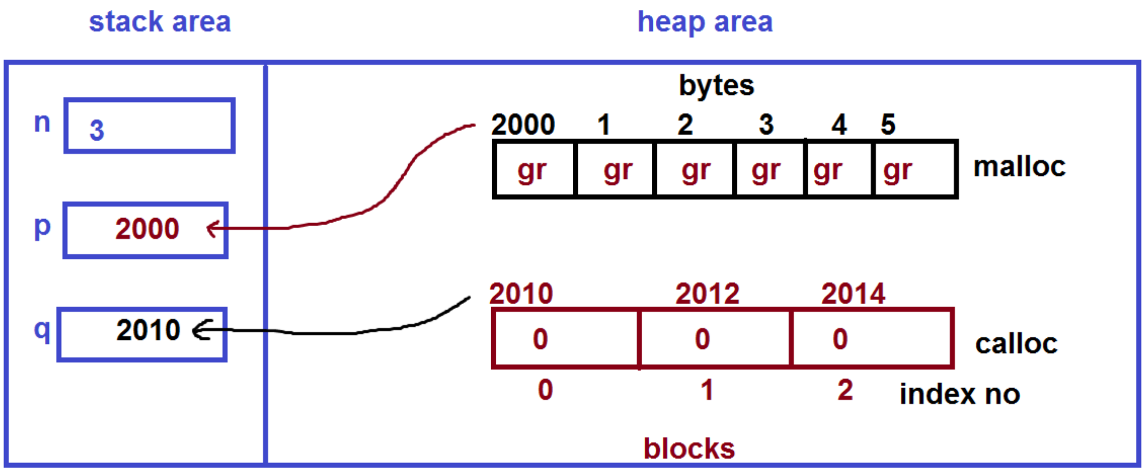
Syntax: void free(pointer);

realloc(): It is used to extend the memory allocated by malloc() or calloc() at runtime. Working style is similar to malloc().

Syntax: **void * realloc(oldptr, newsize);**

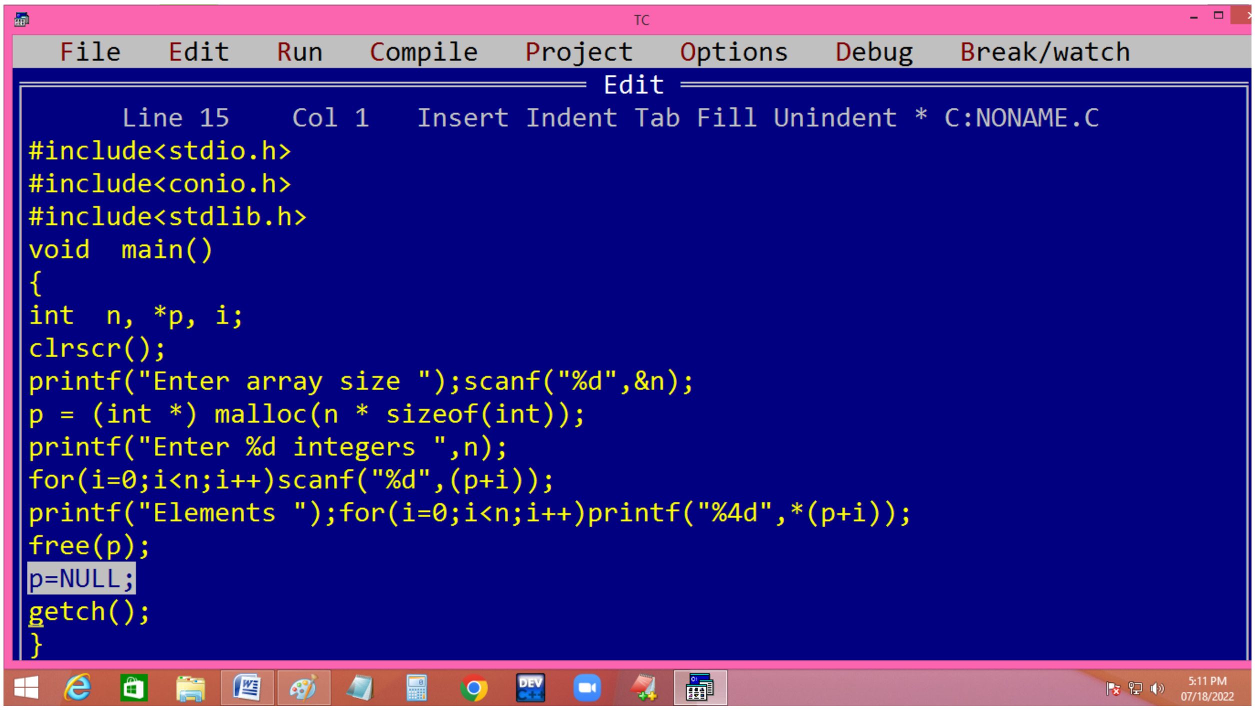
allocating memory for 3 integers using malloc(), calloc().

```
int *p, *q, n=3;

p = (int  *)malloc(n * sizeof(int));

q = (int  *)calloc(n ,  sizeof(int));
```

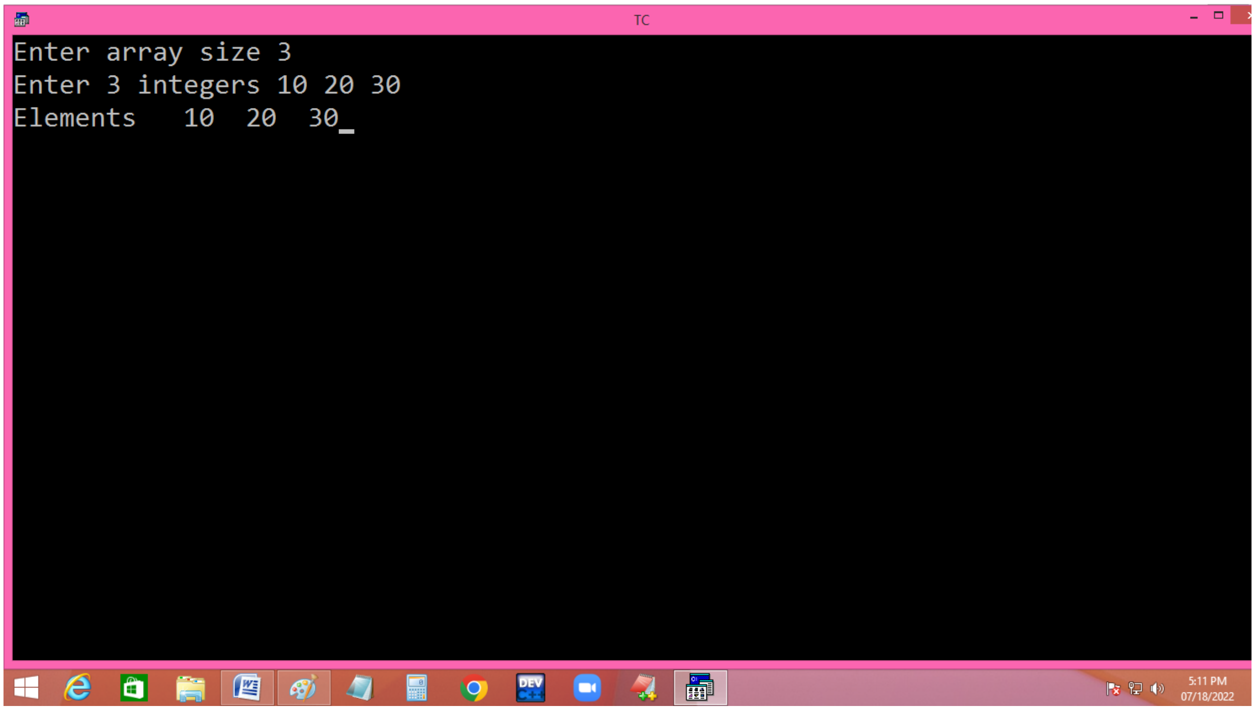


Eg:

Creating dynamic one-dimensional array:



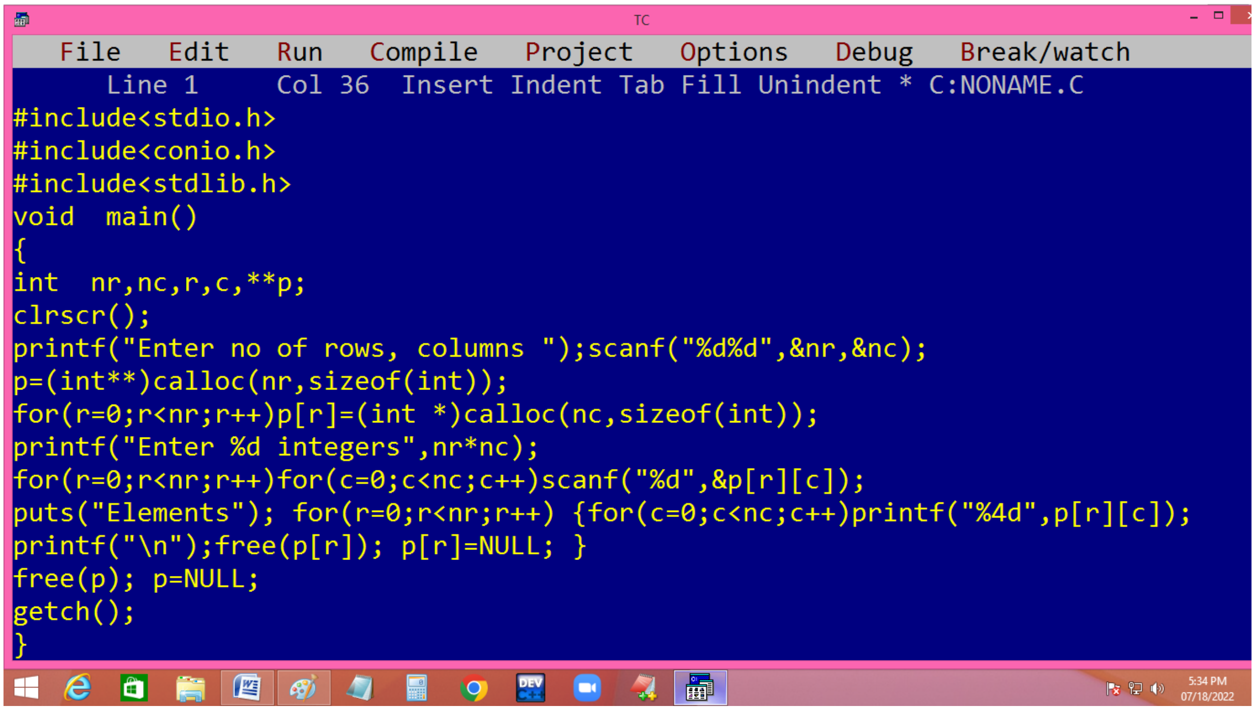
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void  main()
{
int  n, *p, i;
clrscr();
printf("Enter array size ");scanf("%d",&n);
p = (int *) malloc(n * sizeof(int));
printf("Enter %d integers ",n);
for(i=0;i<n;i++)scanf("%d",(p+i));
printf("Elements ");for(i=0;i<n;i++)printf("%4d",*(p+i));
free(p);
p=NULL;
getch();
}
```



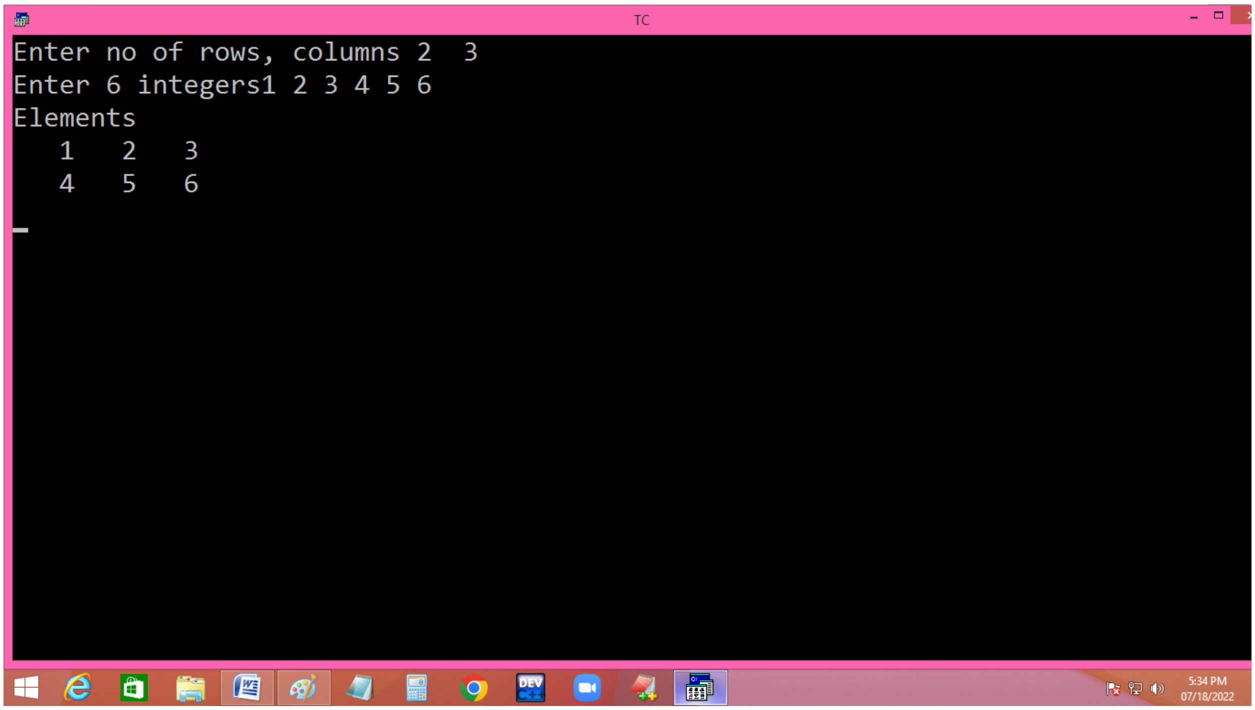
```
Enter array size 3
Enter 3 integers 10 20 30
Elements   10  20  30_
```

p = (int *)calloc(n , sizeof(int));

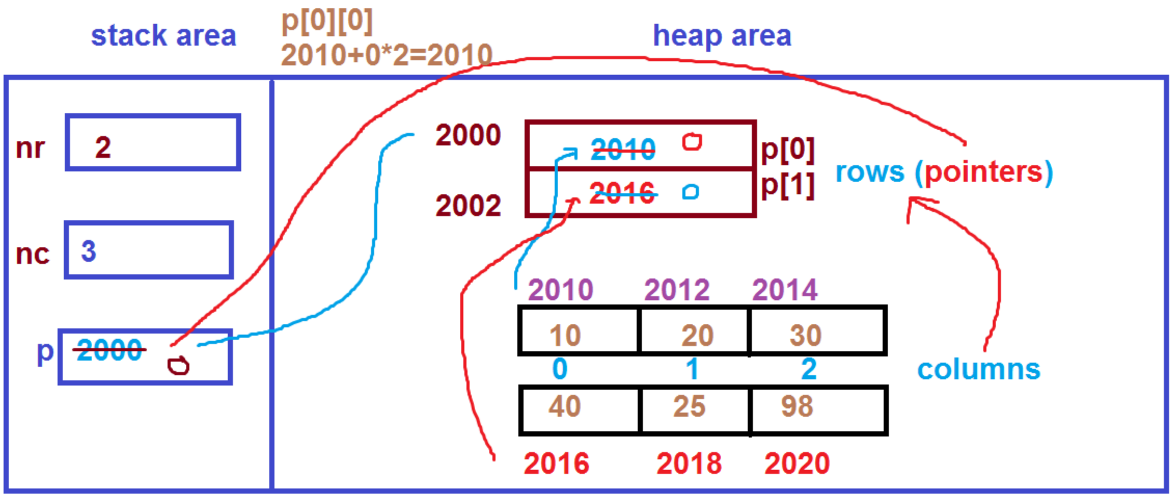
Eg. dynamic multi-dimensional array

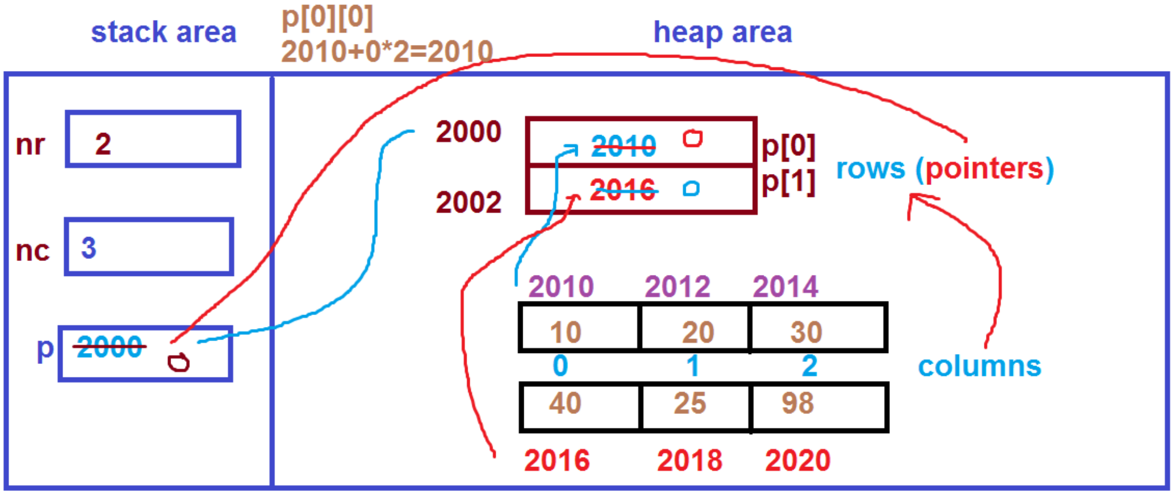


```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void  main()
{
int  nr,nc,r,c,**p;
clrscr();
printf("Enter no of rows, columns ");scanf("%d%d",&nr,&nc);
p=(int**)calloc(nr,sizeof(int));
for(r=0;r<nr;r++)p[r]=(int *)calloc(nc,sizeof(int));
printf("Enter %d integers",nr*nc);
for(r=0;r<nr;r++)for(c=0;c<nc;c++)scanf("%d",&p[r][c]);
puts("Elements"); for(r=0;r<nr;r++) {for(c=0;c<nc;c++)printf("%4d",p[r][c]);
printf("\n");free(p[r]); p[r]=NULL; }
free(p); p=NULL;
getch();
}
```

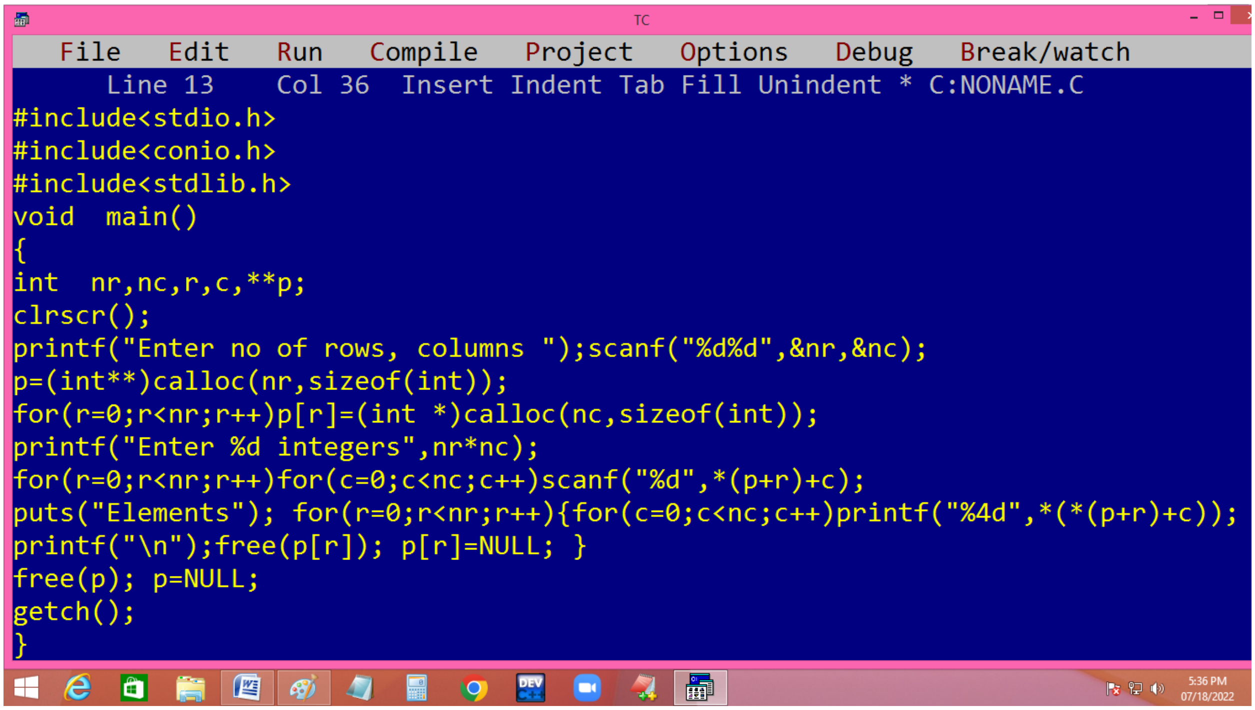


```
Enter no of rows, columns 2  3
Enter 6 integers1 2 3 4 5 6
Elements
   1   2   3
   4   5   6
```

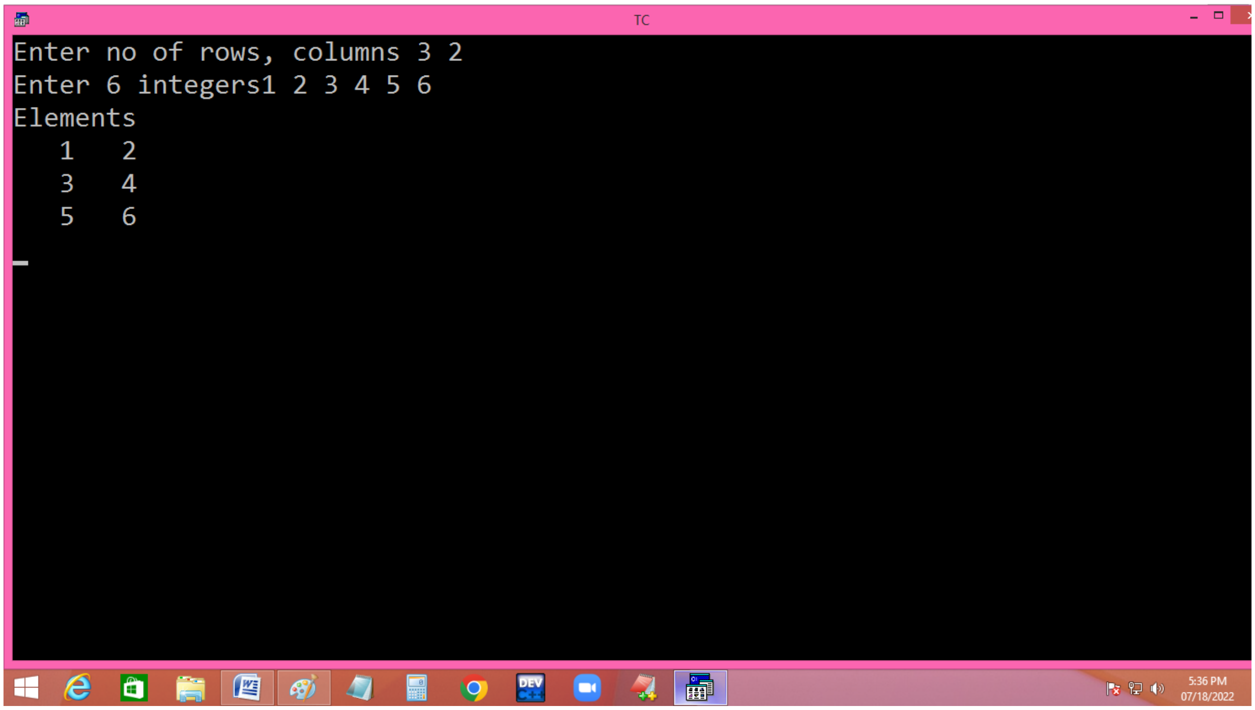




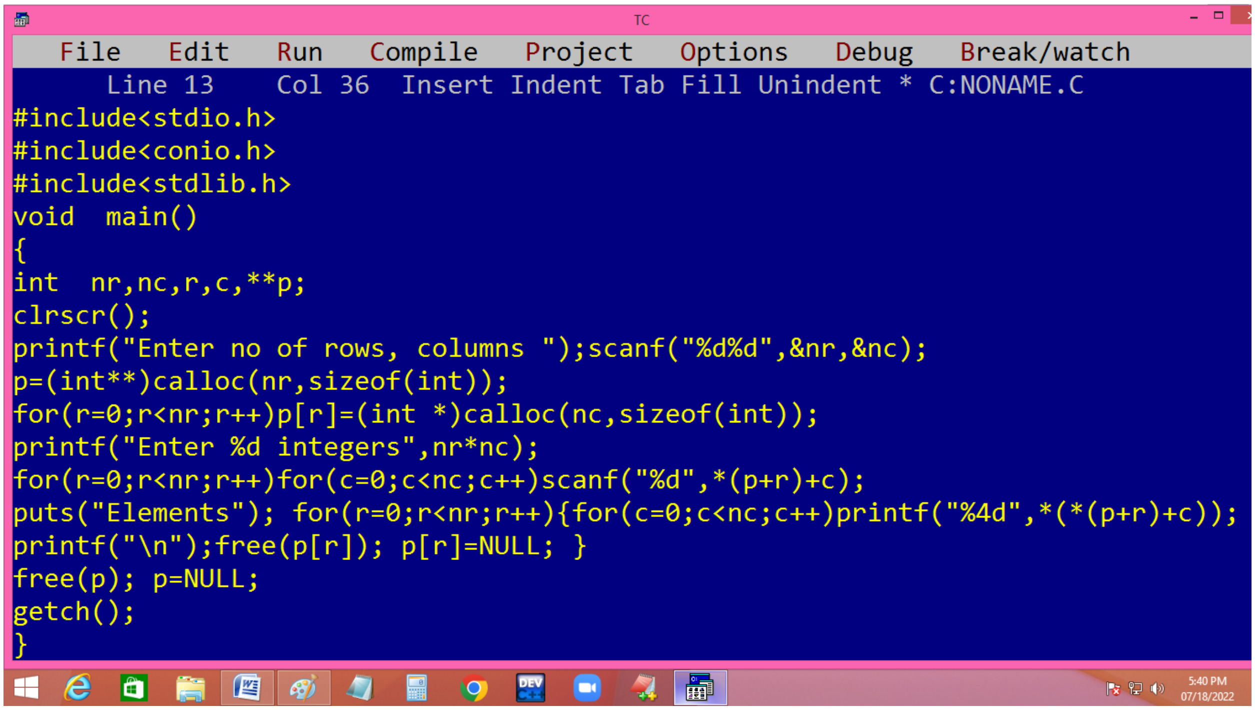
Using pointer notation:



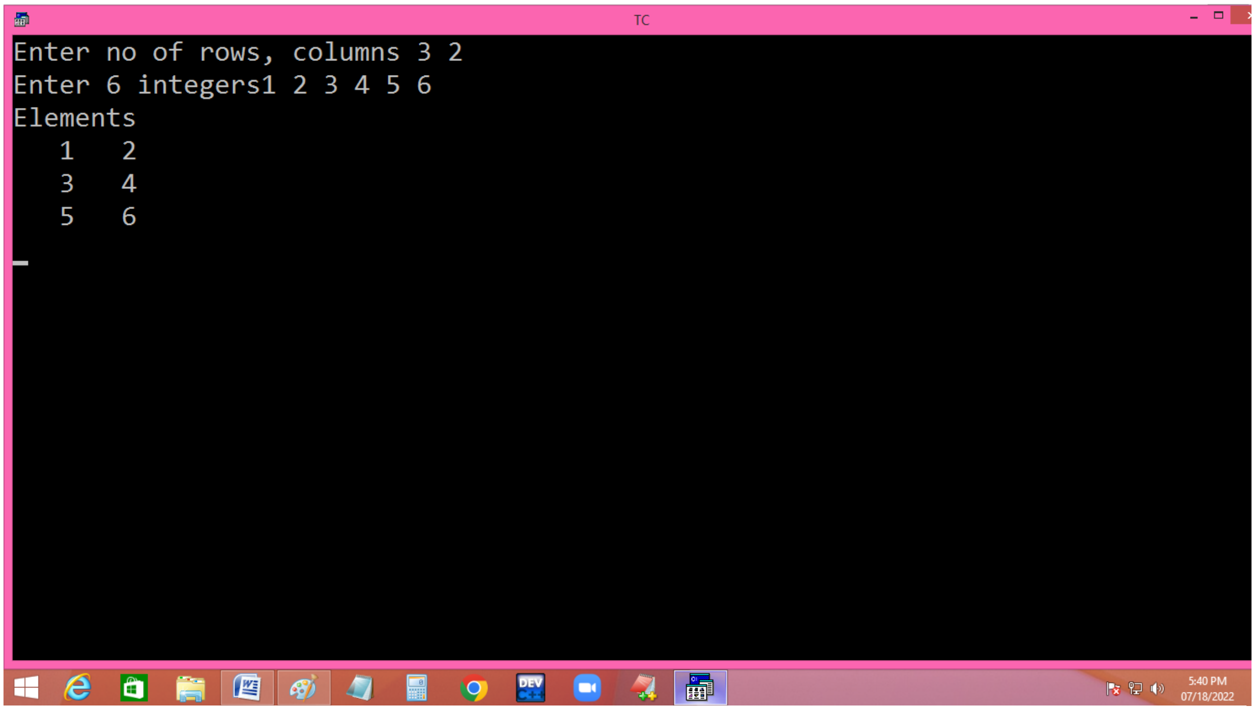
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void  main()
{
int  nr,nc,r,c,**p;
clrscr();
printf("Enter no of rows, columns ");scanf("%d%d",&nr,&nc);
p=(int**)calloc(nr,sizeof(int));
for(r=0;r<nr;r++)p[r]=(int *)calloc(nc,sizeof(int));
printf("Enter %d integers",nr*nc);
for(r=0;r<nr;r++)for(c=0;c<nc;c++)scanf("%d",*(p+r)+c);
puts("Elements"); for(r=0;r<nr;r++){for(c=0;c<nc;c++)printf("%4d",*(*(p+r)+c));
printf("\n");free(p[r]); p[r]=NULL; }
free(p); p=NULL;
getch();
}
```



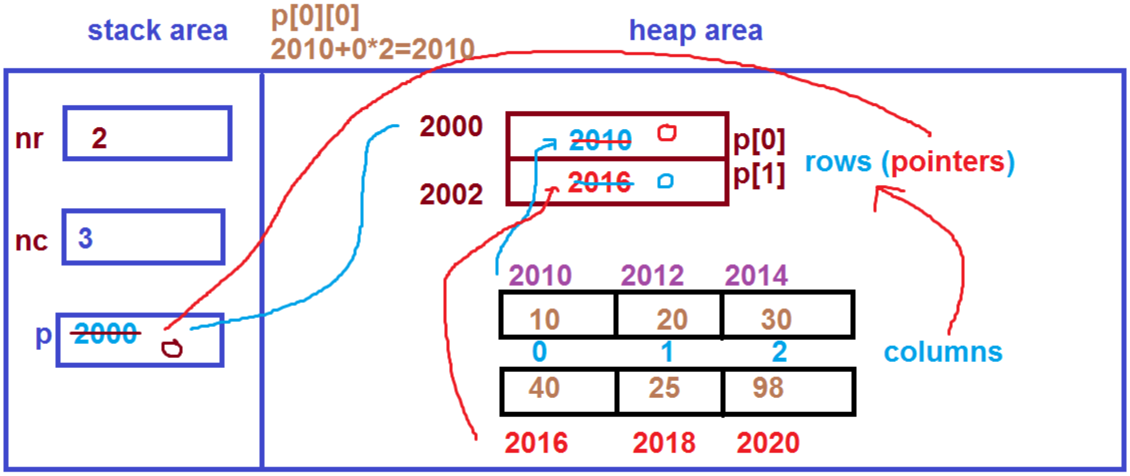
```
Enter no of rows, columns 3 2
Enter 6 integers1 2 3 4 5 6
Elements
   1   2
   3   4
   5   6

_
```



```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void  main()
{
int  nr,nc,r,c,**p;
clrscr();
printf("Enter no of rows, columns ");scanf("%d%d",&nr,&nc);
p=(int**)calloc(nr,sizeof(int));
for(r=0;r<nr;r++)p[r]=(int *)calloc(nc,sizeof(int));
printf("Enter %d integers",nr*nc);
for(r=0;r<nr;r++)for(c=0;c<nc;c++)scanf("%d",*(p+r)+c);
puts("Elements"); for(r=0;r<nr;r++){for(c=0;c<nc;c++)printf("%4d",*(*(p+r)+c));
printf("\n");free(p[r]); p[r]=NULL; }
free(p); p=NULL;
getch();
}
```

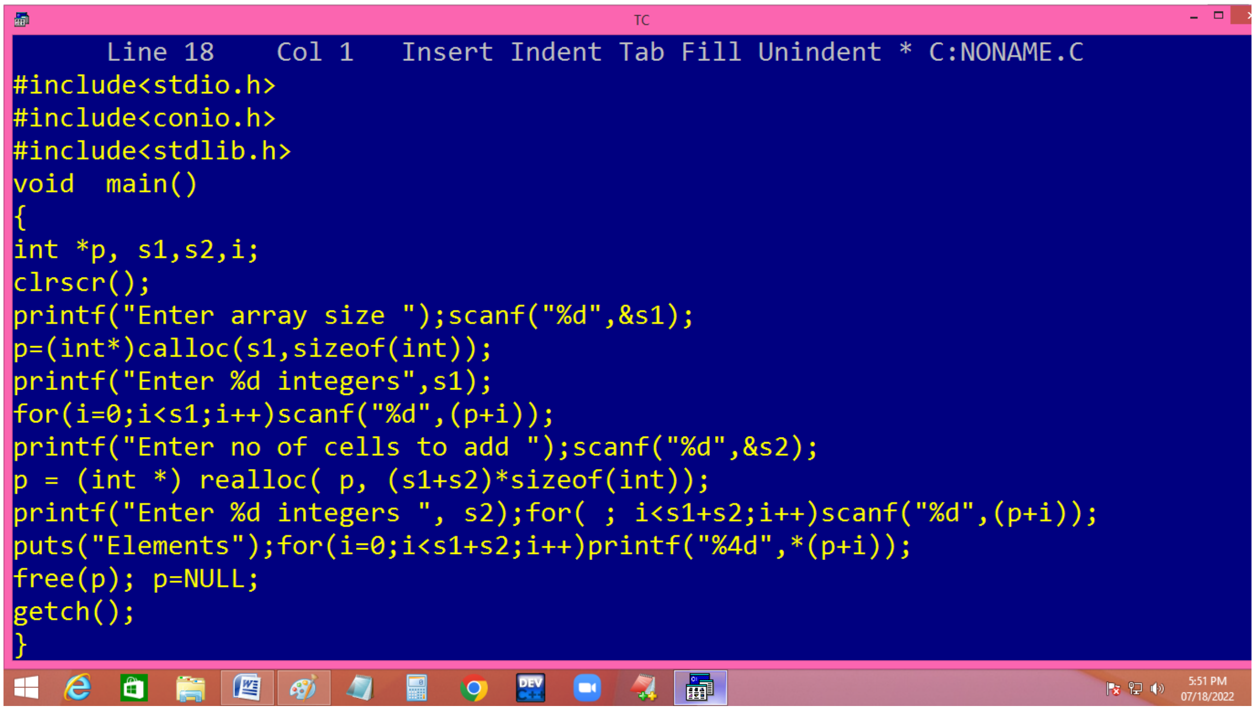


```
Enter no of rows, columns 3 2
Enter 6 integers1 2 3 4 5 6
Elements
   1   2
   3   4
   5   6
```

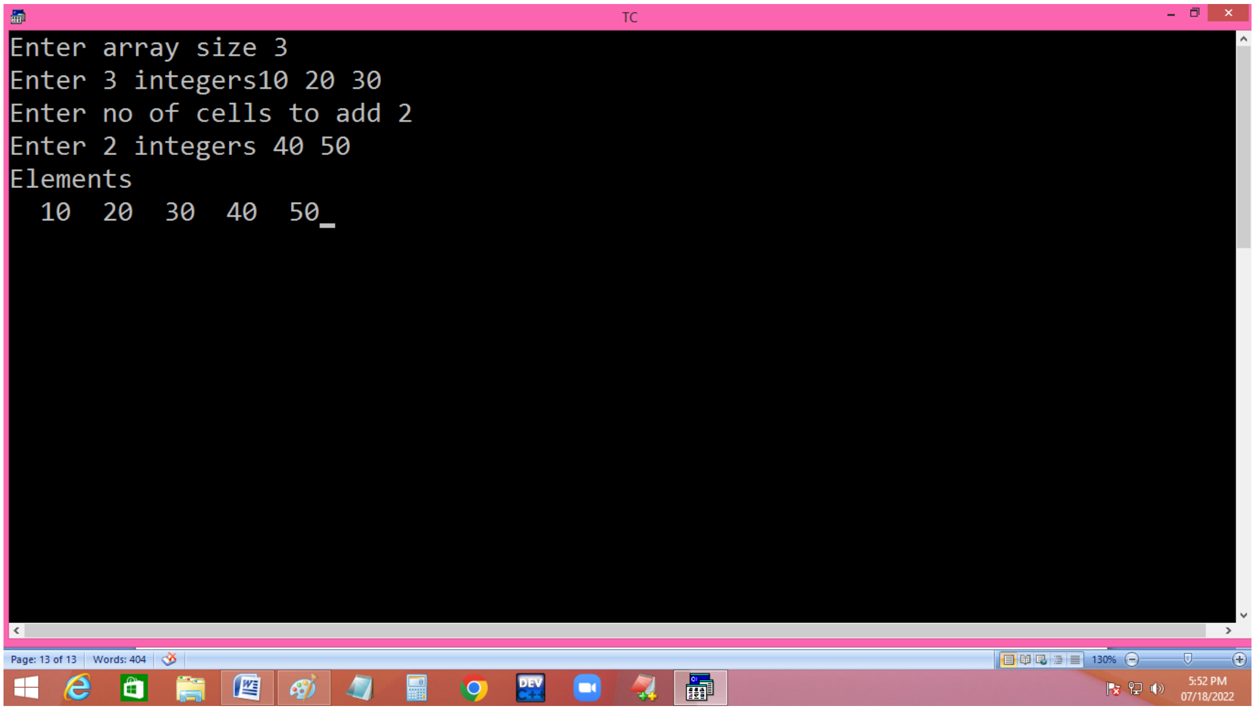


*(p+r) + c ==> *2000==>2010 + 0*2=2010   printf(*(*(p+r)+c)) ==> *2010==>10

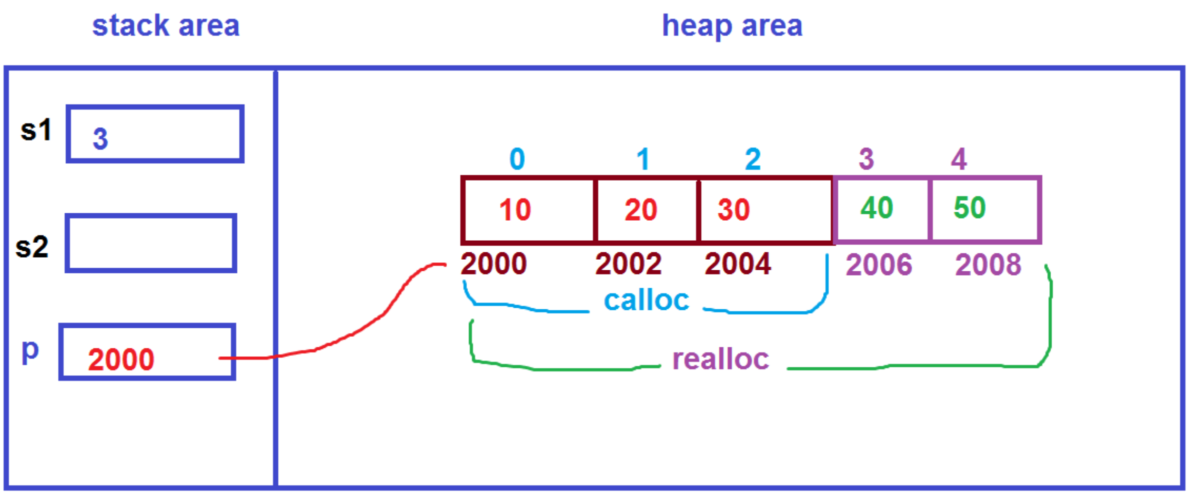
realloc() example:


TC
Line 18   Col 1   Insert Indent Tab Fill Unindent * C:NONAME.C
5:51 PM
07/18/2022

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void  main()
{
int *p, s1,s2,i;
clrscr();
printf("Enter array size ");scanf("%d",&s1);
p=(int*)calloc(s1,sizeof(int));
printf("Enter %d integers",s1);
for(i=0;i<s1;i++)scanf("%d",(p+i));
printf("Enter no of cells to add ");scanf("%d",&s2);
p = (int *) realloc( p, (s1+s2)*sizeof(int));
printf("Enter %d integers ", s2);for( ; i<s1+s2;i++)scanf("%d",(p+i));
puts("Elements");for(i=0;i<s1+s2;i++)printf("%4d",*(p+i));
free(p); p=NULL;
getch();
}
```



```
Enter array size 3
Enter 3 integers10 20 30
Enter no of cells to add 2
Enter 2 integers 40 50
Elements
  10  20  30  40  50_
```


stack area
heap area
s1
3
s2
p
2000
0
1
2
3
4
10
20
30
40
50
2000
2002
2004
2006
2008
calloc
realloc