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 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()**
Block Memory allocation	Contiguous memory blocks 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);

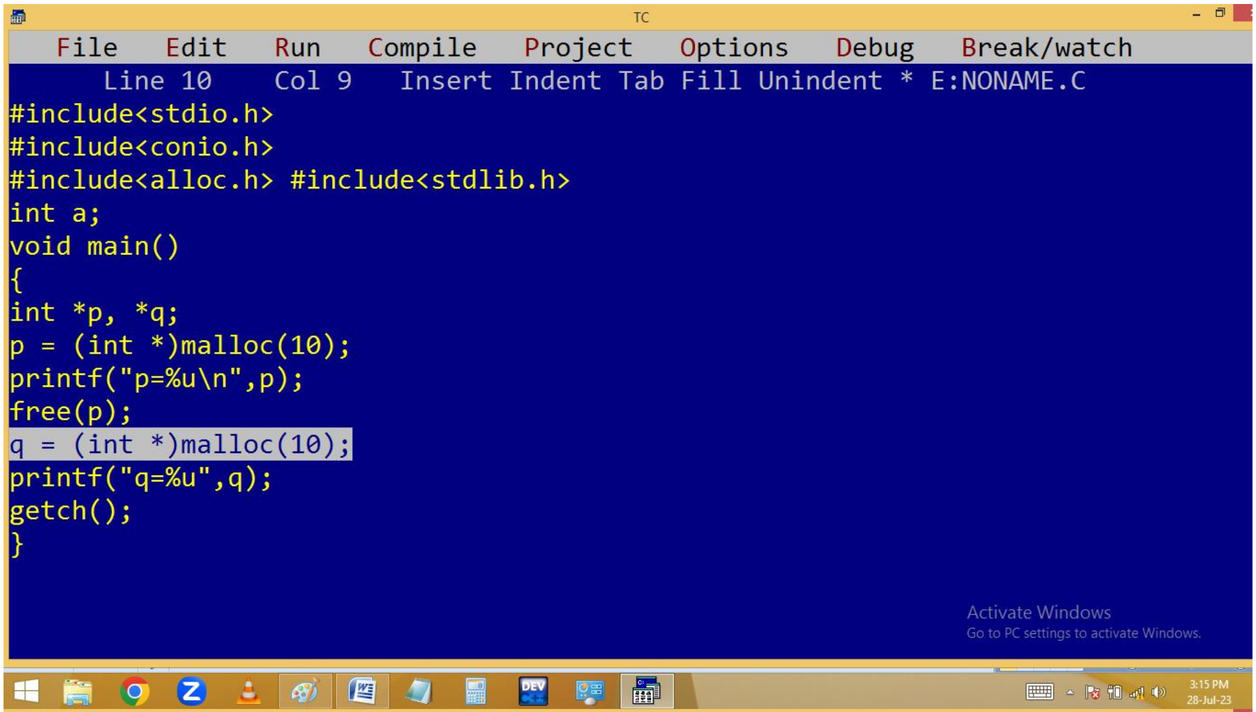
free(): It is used to release the memory allocated by malloc(), calloc() and realloc().

Syntax: void free(pointer);

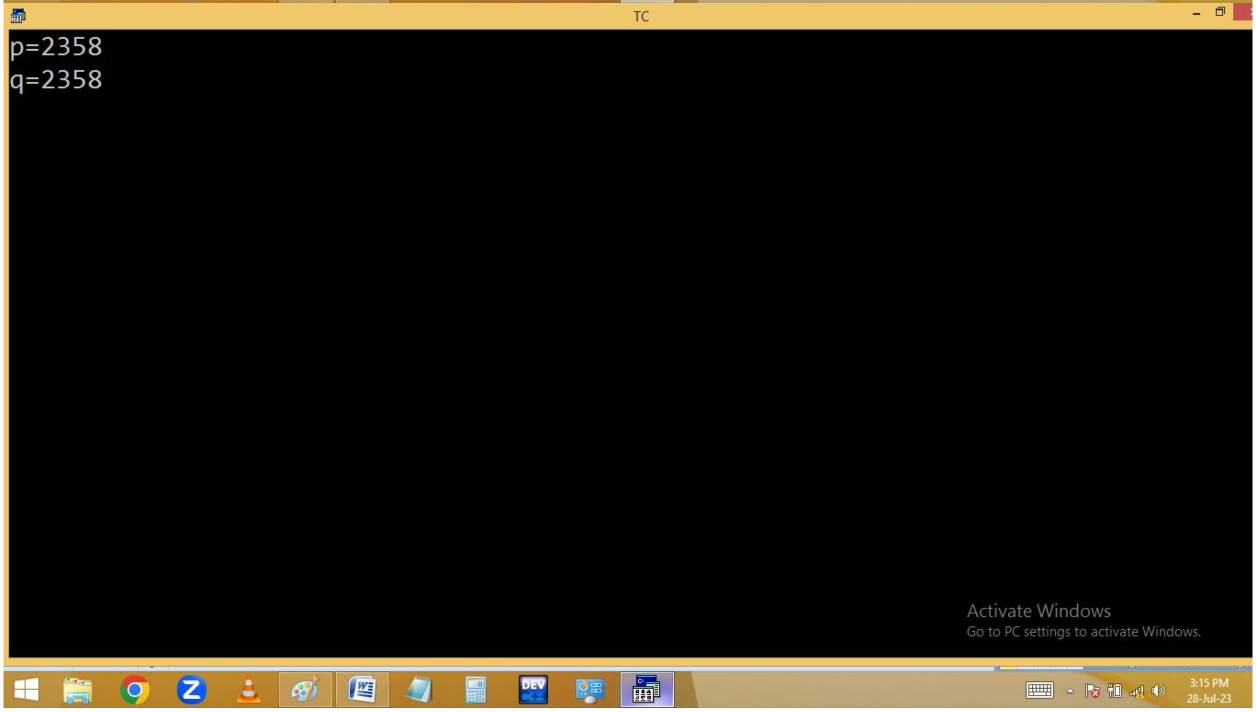
realloc(): It is used to extend the memory allocated by malloc() or calloc() at runtime. Working style is similar to malloc().

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

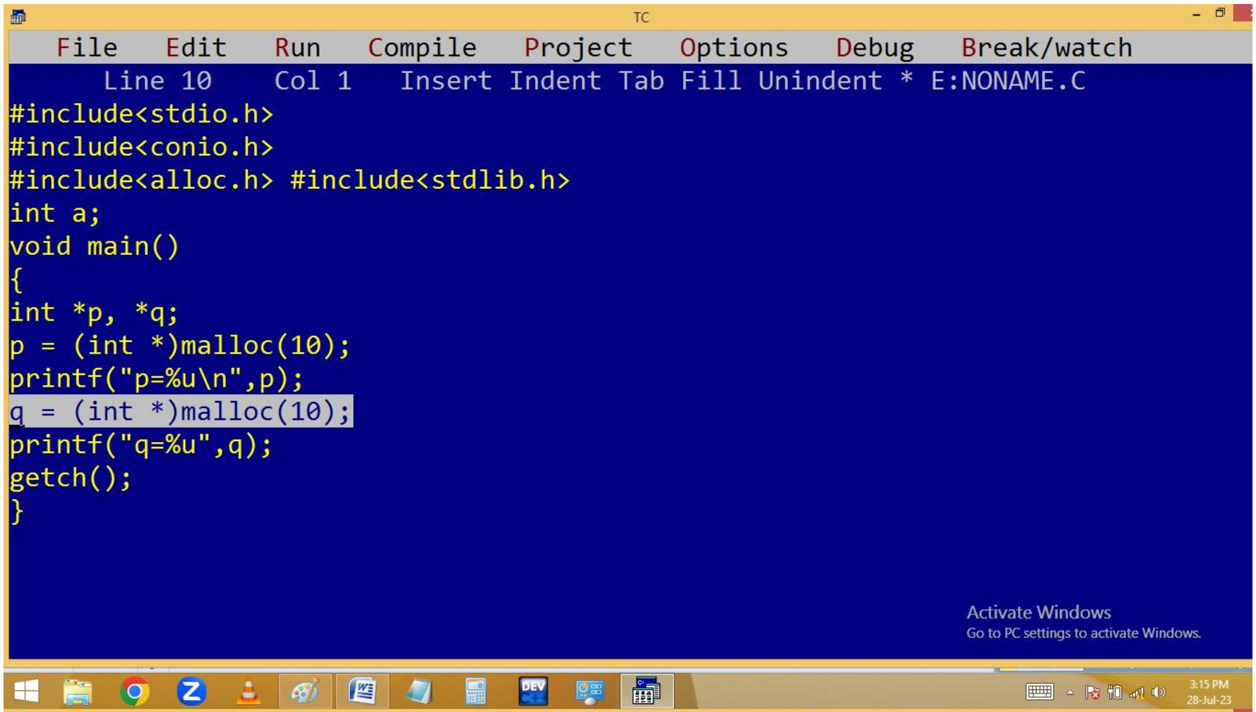
free example:


TC
File Edit Run Compile Project Options Debug Break/watch
Line 10 Col 9 Insert Indent Tab Fill Unindent * E:NONAME.C
Activate Windows
Go to PC settings to activate Windows.
3:15 PM
28-Jul-23

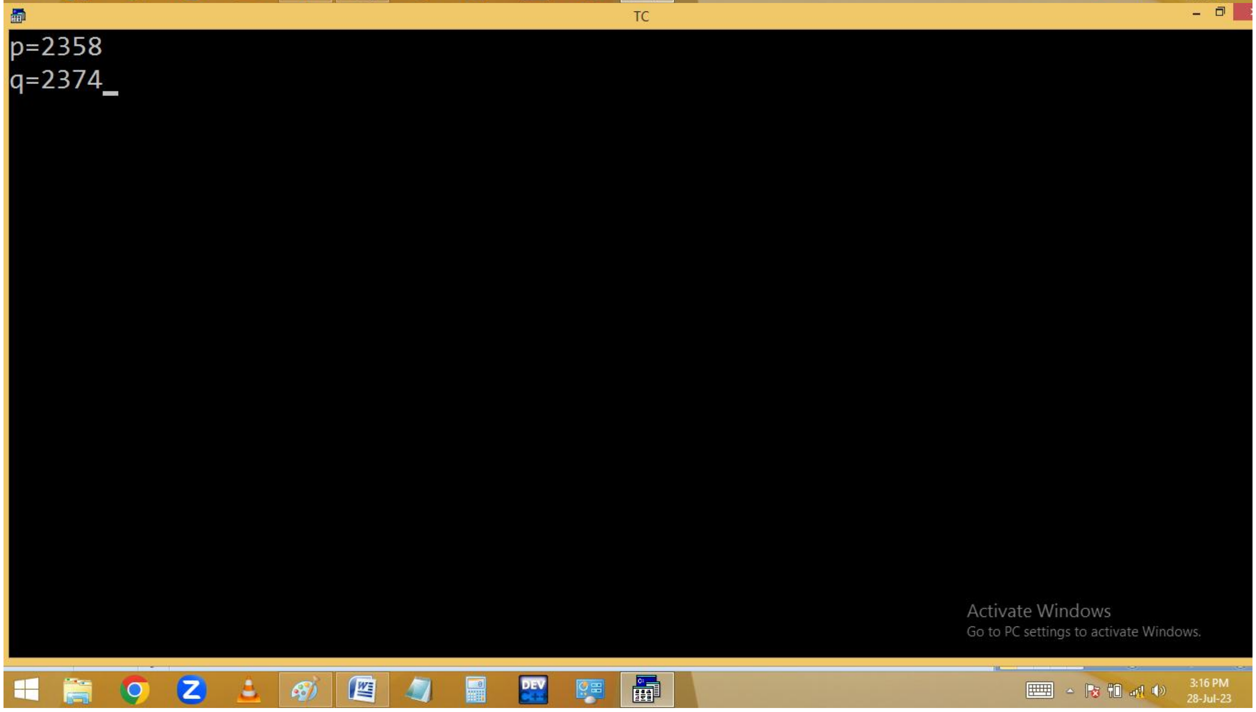
```
#include<stdio.h>
#include<conio.h>
#include<alloc.h> #include<stdlib.h>
int a;
void main()
{
int *p, *q;
p = (int *)malloc(10);
printf("p=%u\n",p);
free(p);
q = (int *)malloc(10);
printf("q=%u",q);
getch();
}
```



```
p=2358
q=2358
```


TC
File Edit Run Compile Project Options Debug Break/watch
Line 10 Col 1 Insert Indent Tab Fill Unindent * E:NONAME.C
Activate Windows
Go to PC settings to activate Windows.
3:15 PM
28-Jul-23

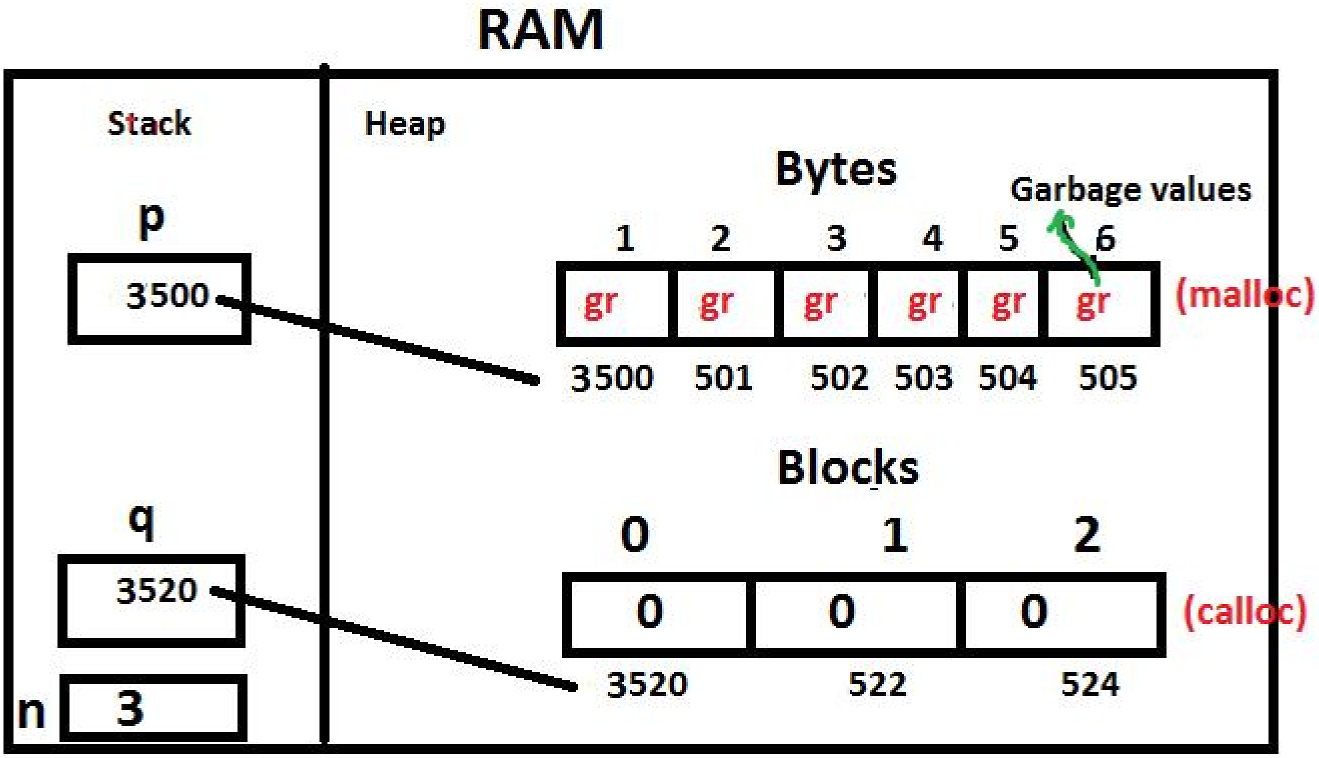
```
#include<stdio.h>
#include<conio.h>
#include<alloc.h> #include<stdlib.h>
int a;
void main()
{
int *p, *q;
p = (int *)malloc(10);
printf("p=%u\n",p);
q = (int *)malloc(10);
printf("q=%u",q);
getch();
}
```



```
p=2358
q=2374_
```

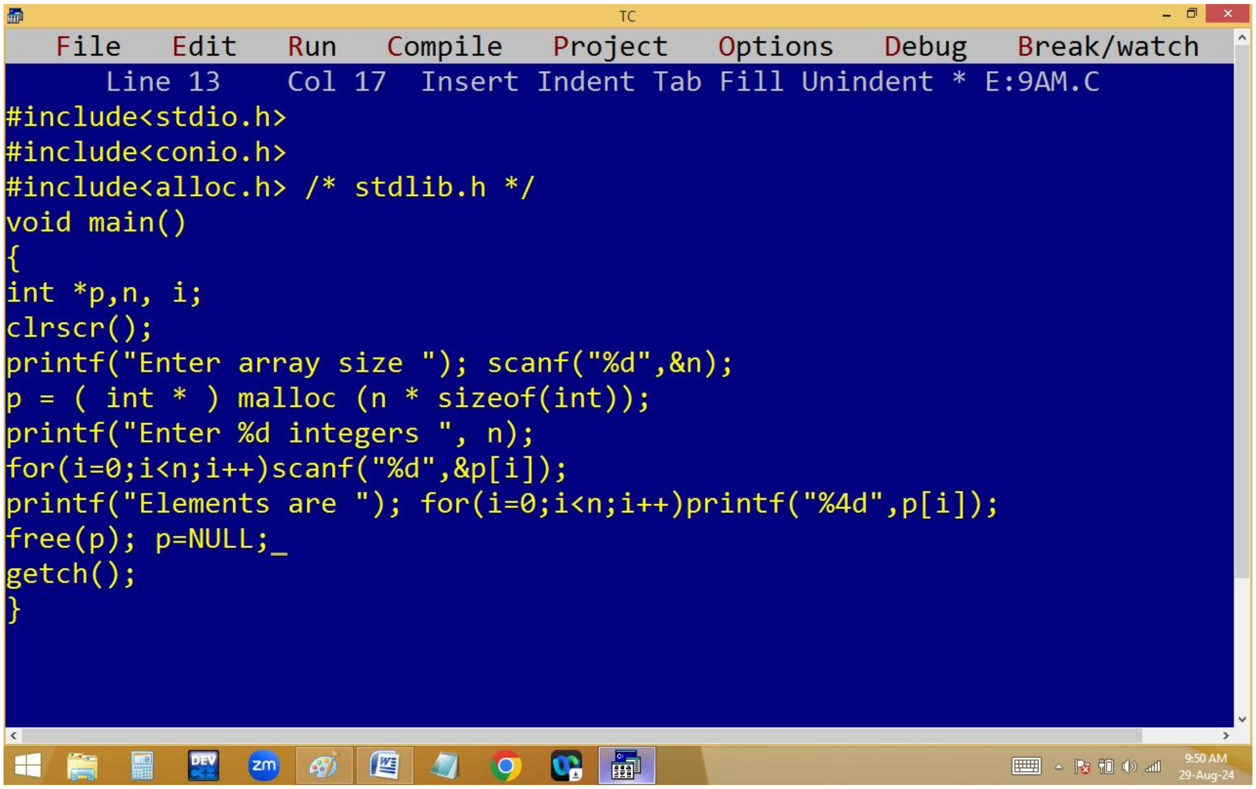
allocating memory for 3 integers using malloc(), calloc().

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

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

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

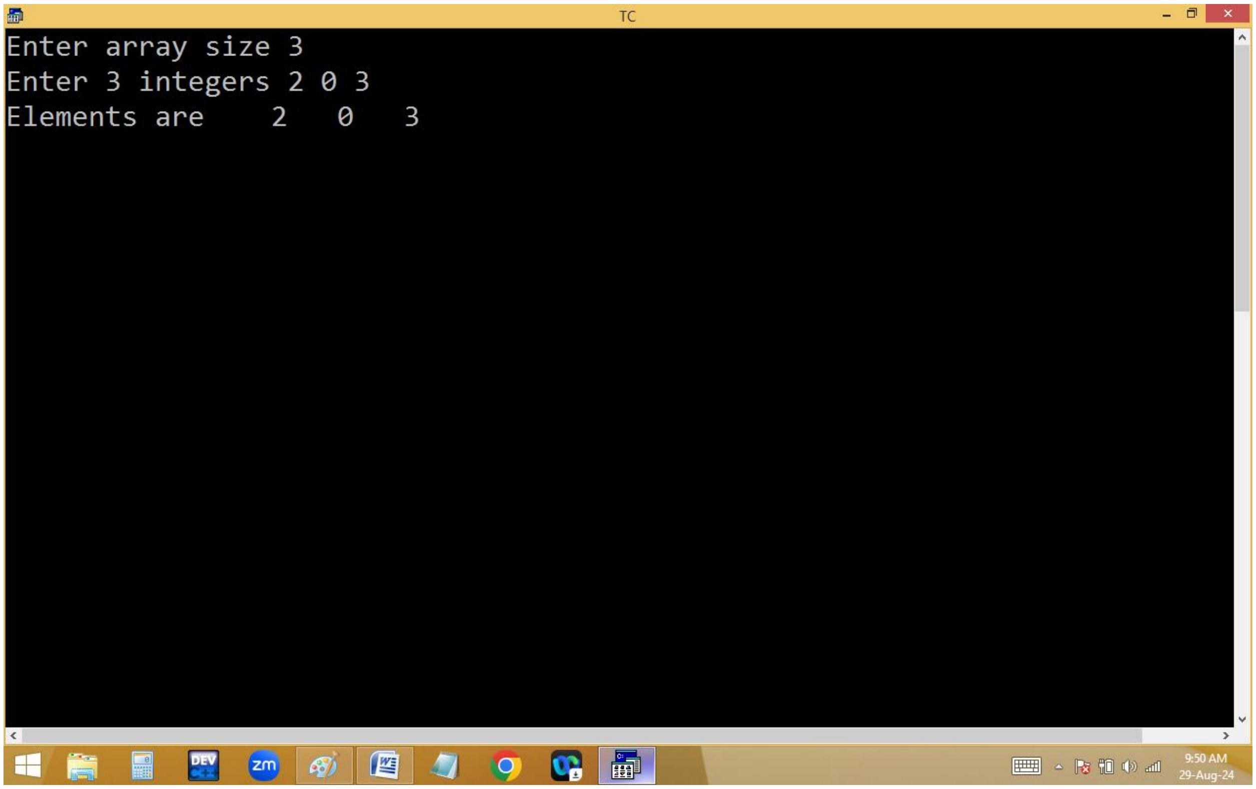


Eg:

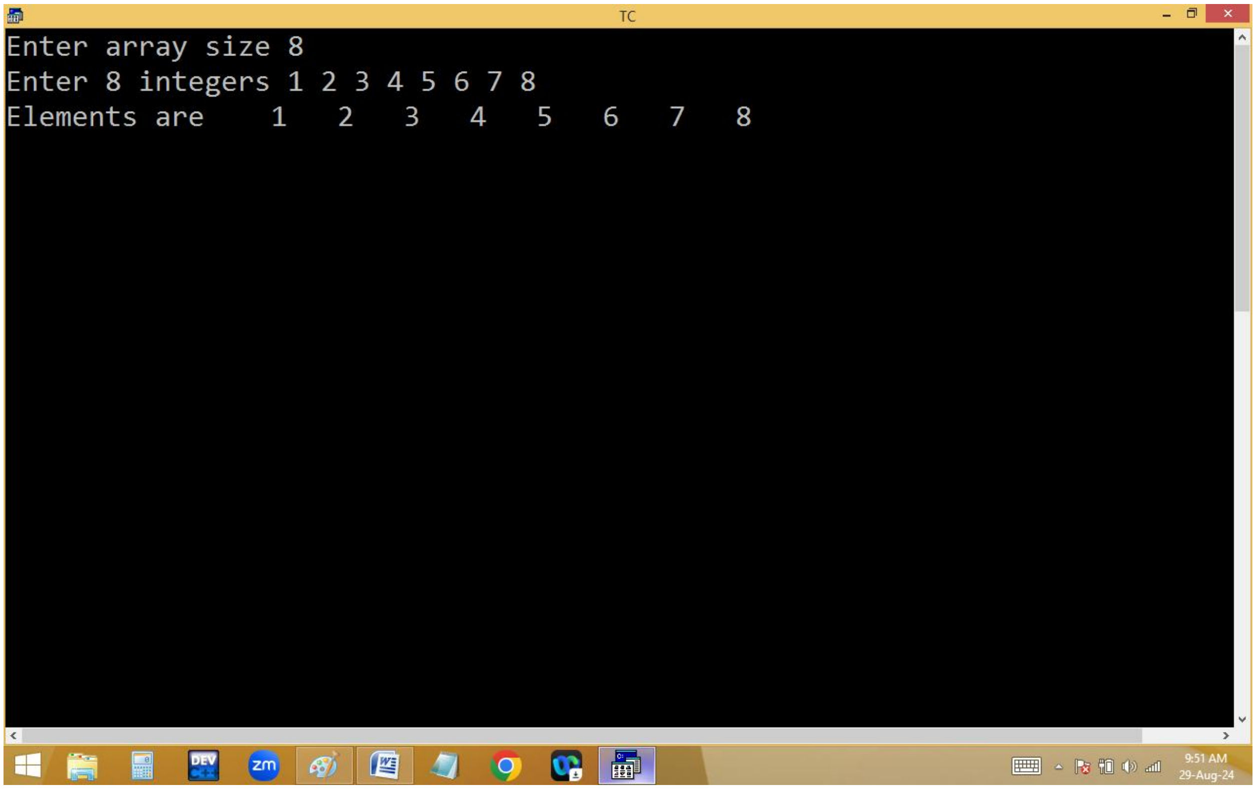
Creating dynamic one-dimensional array:


TC
File Edit Run Compile Project Options Debug Break/watch
Line 13 Col 17 Insert Indent Tab Fill Unindent * E:9AM.C
9:50 AM
29-Aug-24

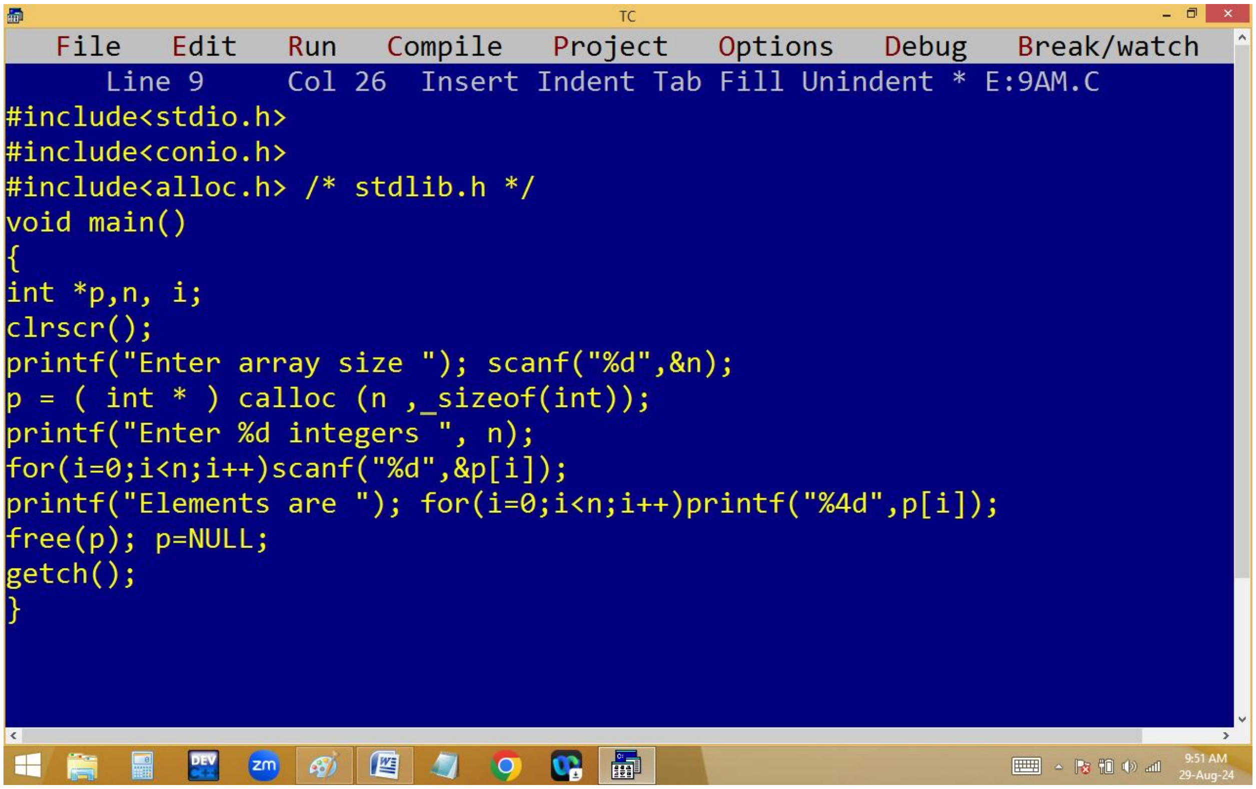
```
#include<stdio.h>
#include<conio.h>
#include<alloc.h> /* stdlib.h */
void main()
{
int *p,n, i;
clrscr();
printf("Enter array size "); scanf("%d",&n);
p = ( int * ) malloc (n * sizeof(int));
printf("Enter %d integers ", n);
for(i=0;i<n;i++)scanf("%d",&p[i]);
printf("Elements are "); for(i=0;i<n;i++)printf("%4d",p[i]);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



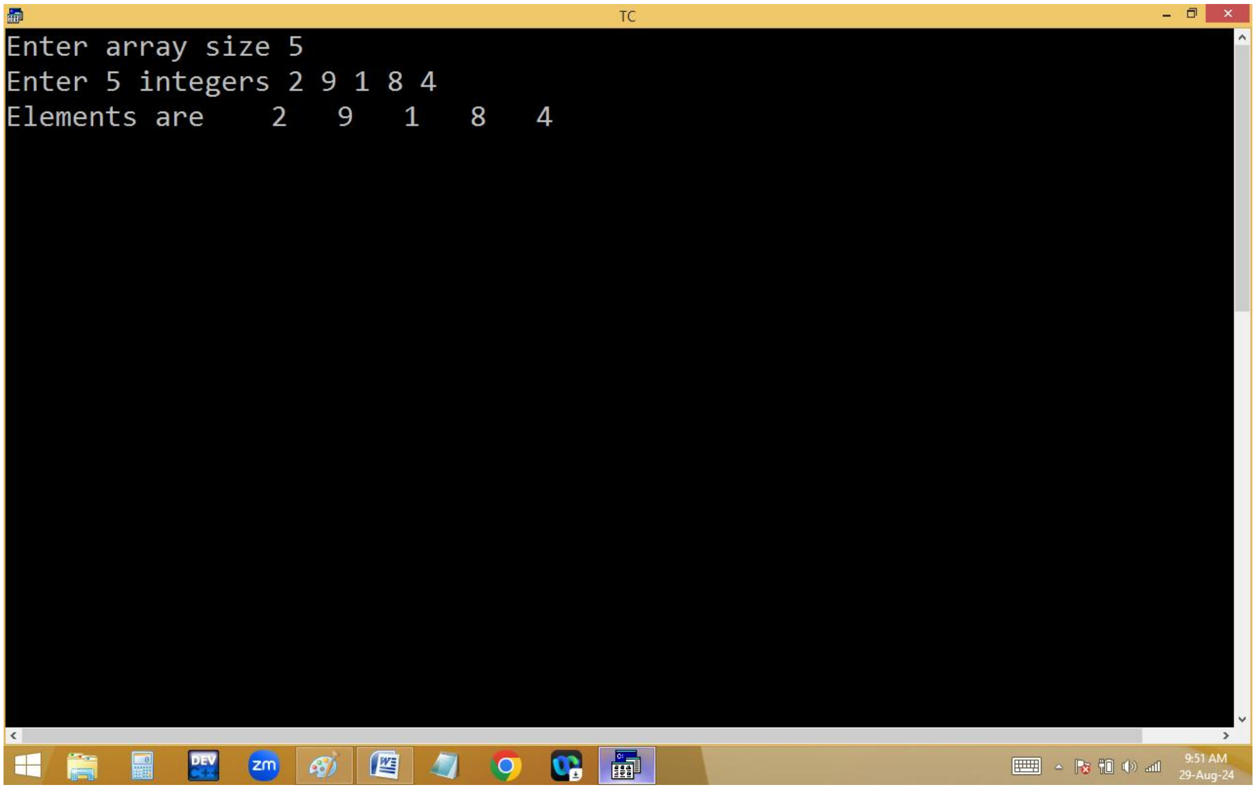
```
Enter array size 3
Enter 3 integers 2 0 3
Elements are    2   0   3
```



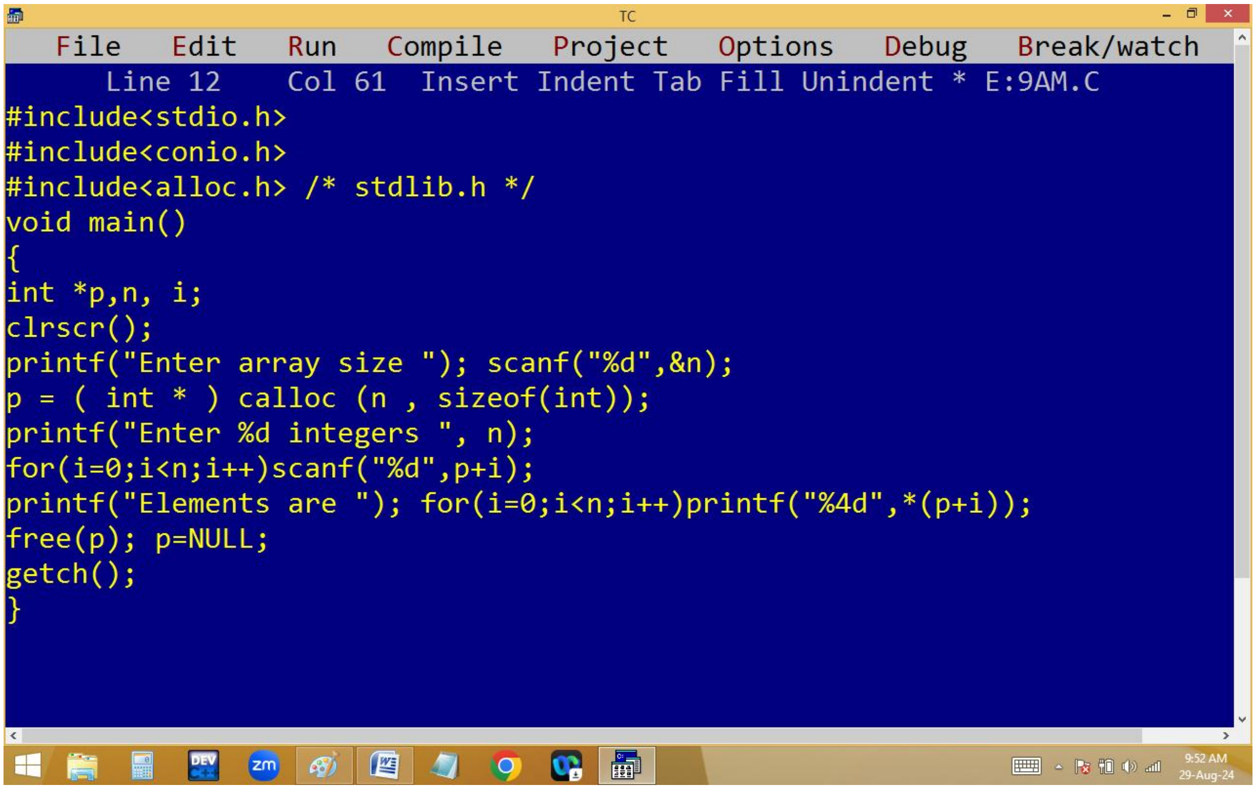
```
Enter array size 8
Enter 8 integers 1 2 3 4 5 6 7 8
Elements are    1   2   3   4   5   6   7   8
```


TC
File Edit Run Compile Project Options Debug Break/watch
Line 9 Col 26 Insert Indent Tab Fill Unindent * E:9AM.C
9:51 AM
29-Aug-24

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

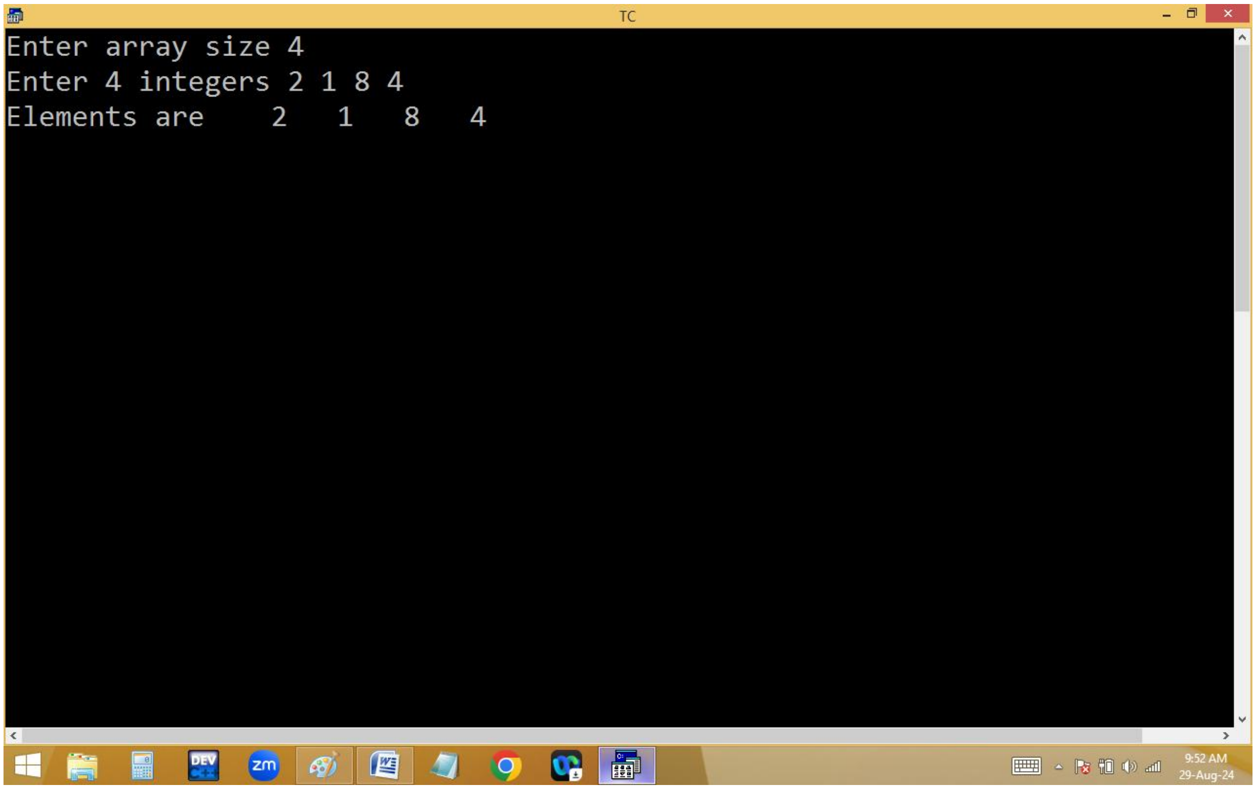


```
Enter array size 5
Enter 5 integers 2 9 1 8 4
Elements are    2   9   1   8   4
```

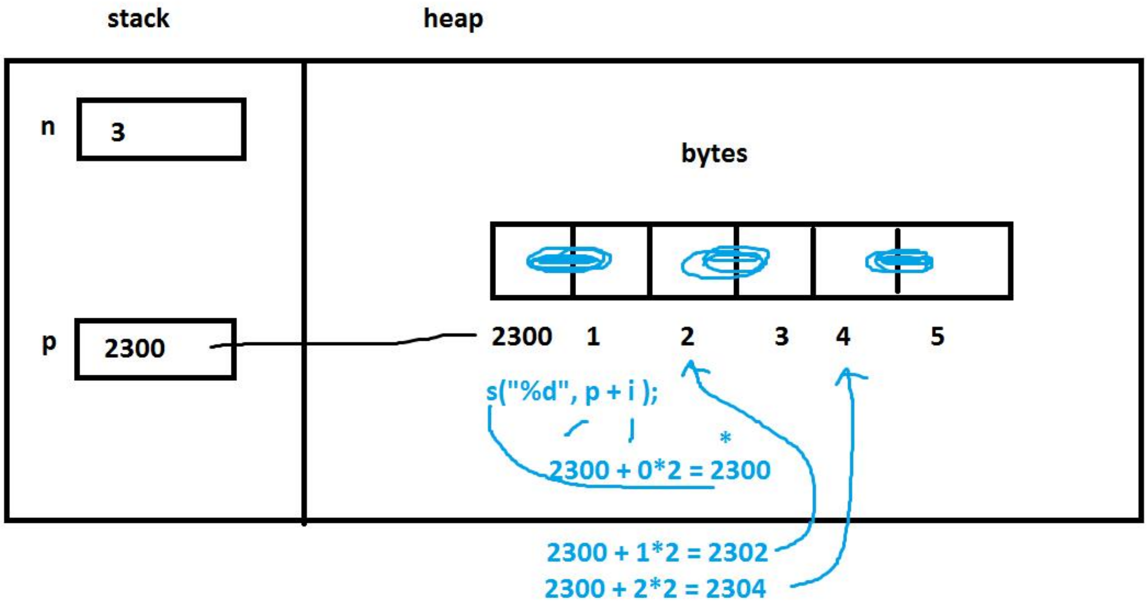


TC
File Edit Run Compile Project Options Debug Break/watch
Line 12 Col 61 Insert Indent Tab Fill Unindent * E:9AM.C
9:52 AM
29-Aug-24

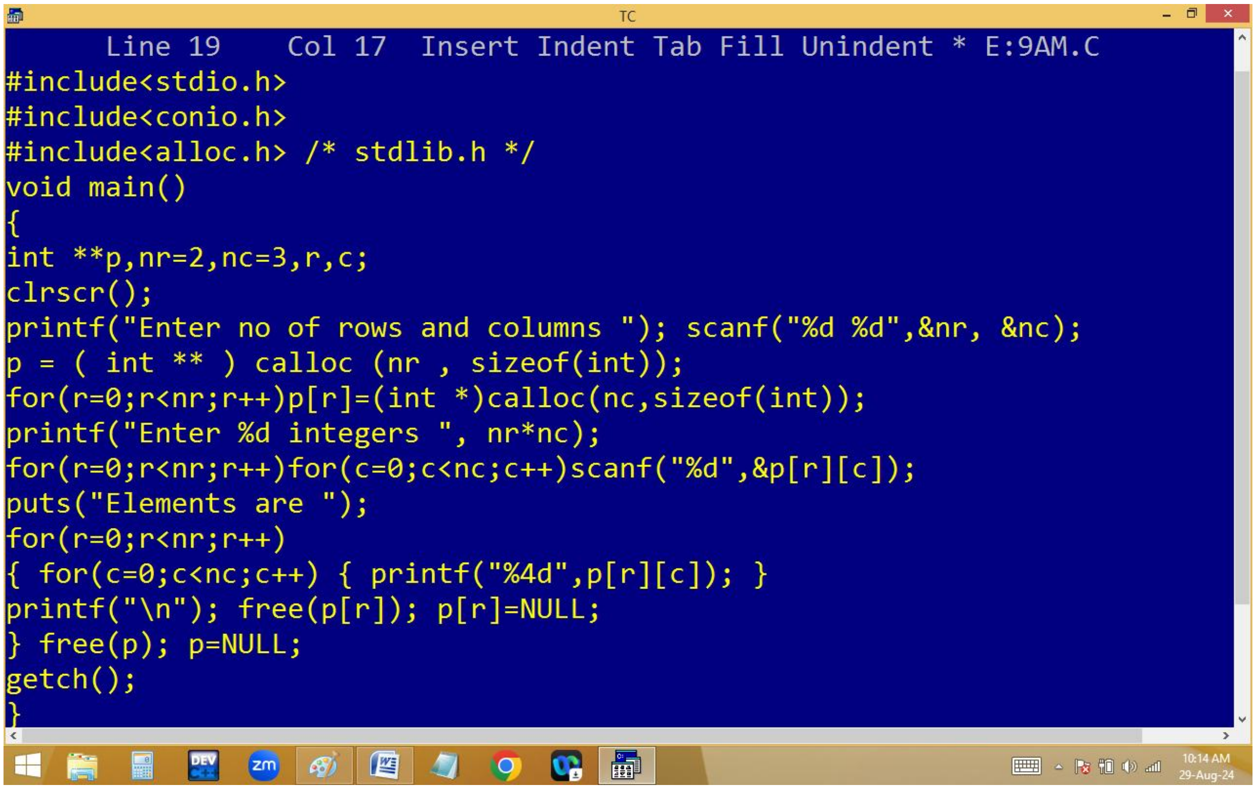
```
#include<stdio.h>
#include<conio.h>
#include<alloc.h> /* stdlib.h */
void main()
{
int *p,n, i;
clrscr();
printf("Enter array size "); scanf("%d",&n);
p = ( int * ) calloc (n , sizeof(int));
printf("Enter %d integers ", n);
for(i=0;i<n;i++)scanf("%d",p+i);
printf("Elements are "); for(i=0;i<n;i++)printf("%4d",*(p+i));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



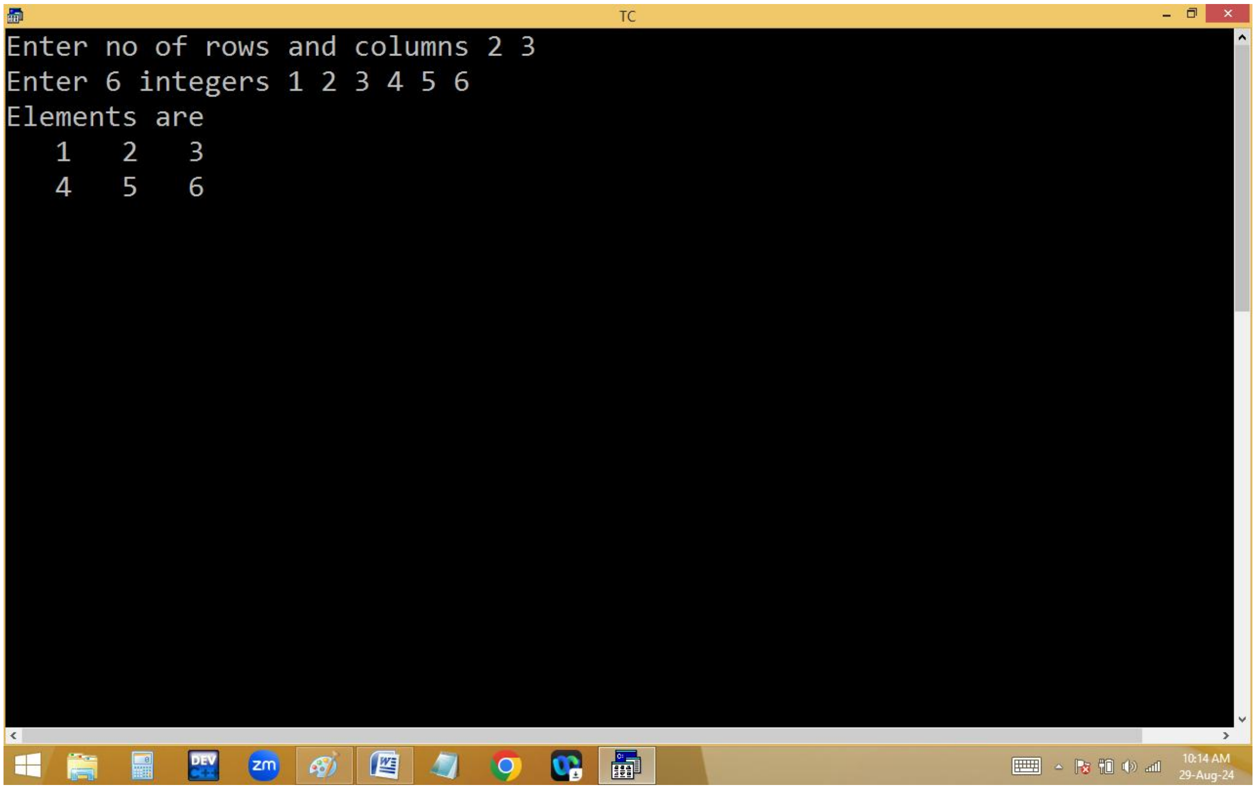
```
Enter array size 4
Enter 4 integers 2 1 8 4
Elements are     2    1    8    4
```



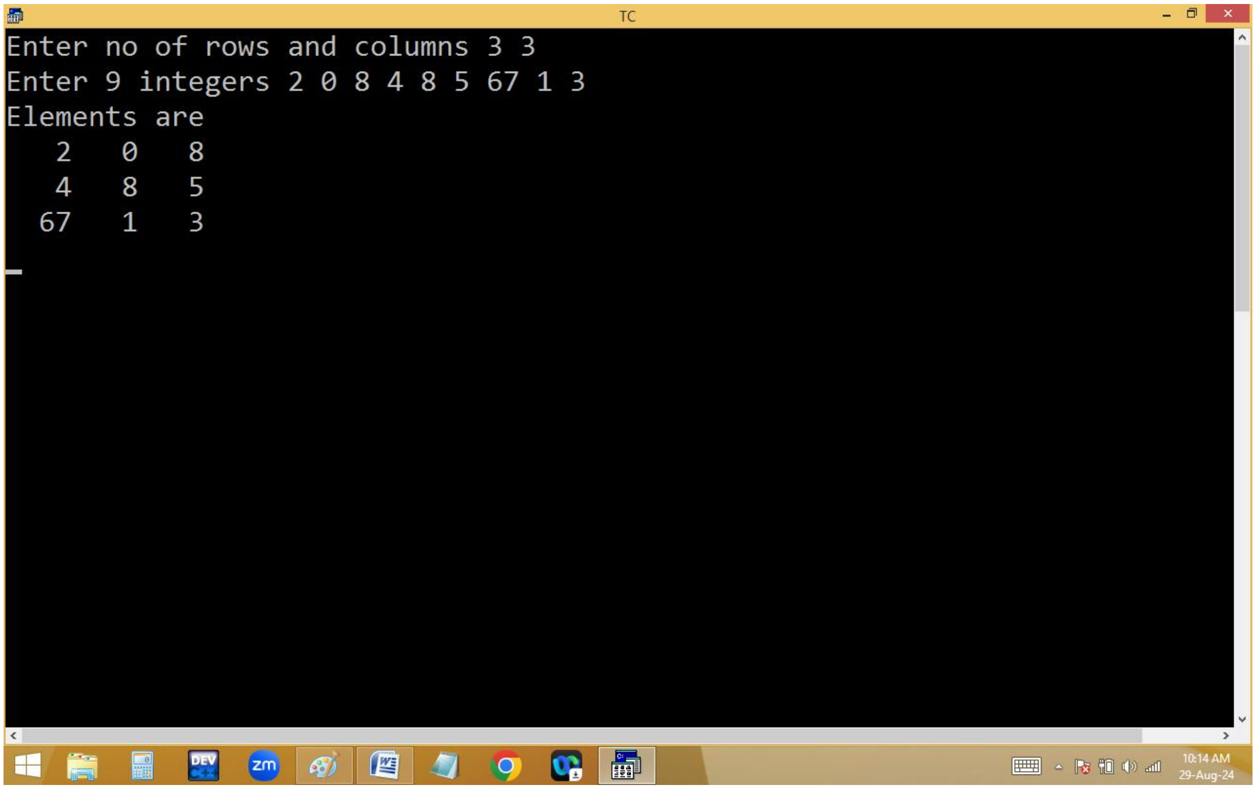
Creating a dynamic multi dimensional array:



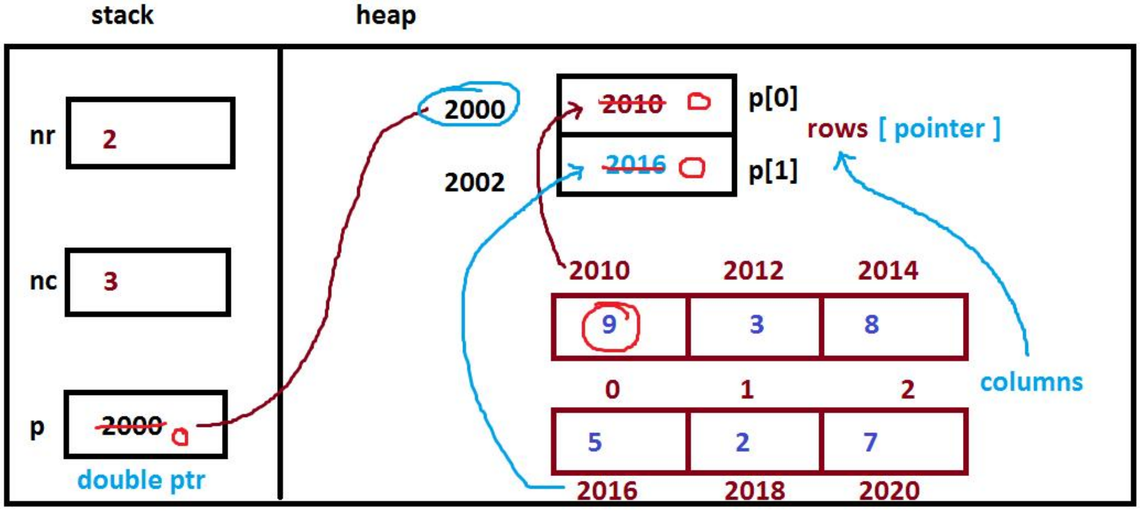
```
      Line 19     Col 17  Insert Indent Tab Fill Unindent * E:9AM.C
#include<stdio.h>
#include<conio.h>
#include<alloc.h> /* stdlib.h */
void main()
{
int **p,nr=2,nc=3,r,c;
clrscr();
printf("Enter no of rows and columns "); scanf("%d %d",&nr, &nc);
p = ( int ** ) calloc (nr , sizeof(int));
for(r=0;r<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 are ");
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 and columns 2 3
Enter 6 integers 1 2 3 4 5 6
Elements are
   1   2   3
   4   5   6
```

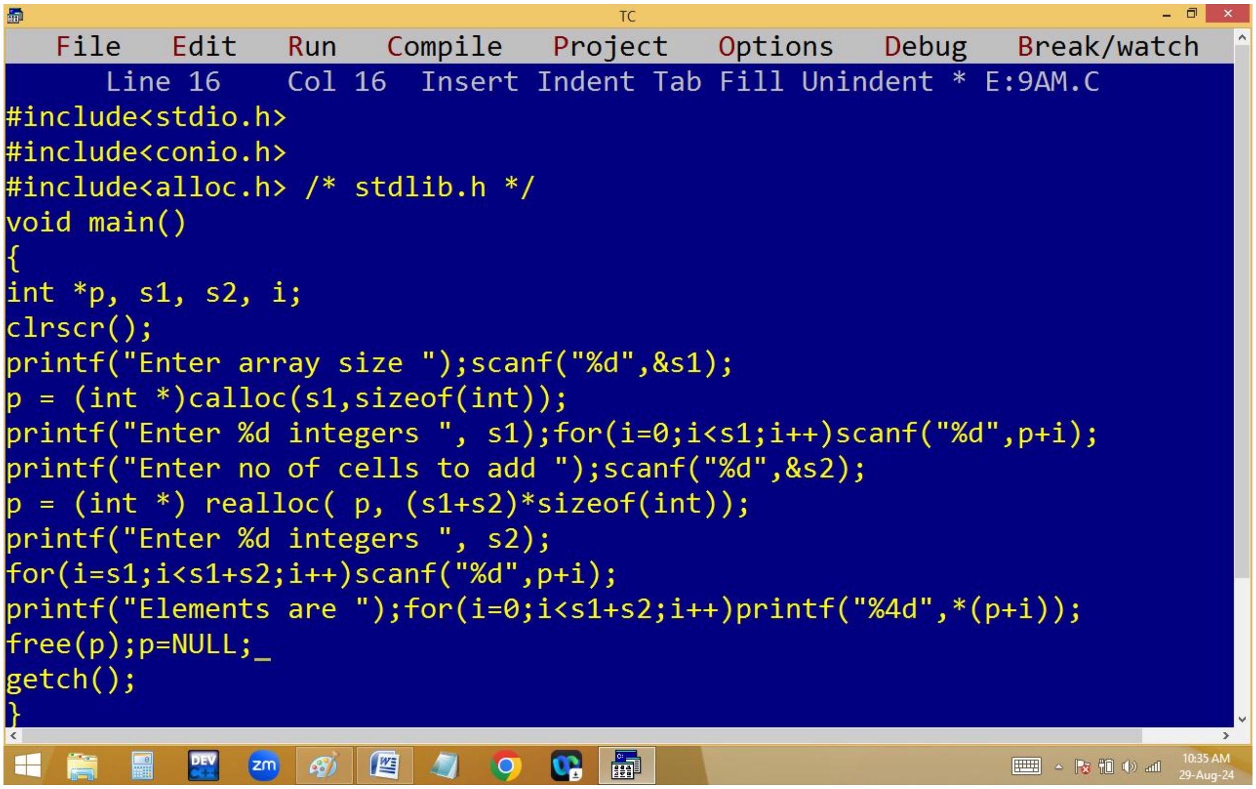


```
Enter no of rows and columns 3 3
Enter 9 integers 2 0 8 4 8 5 67 1 3
Elements are
   2   0   8
   4   8   5
  67   1   3
```

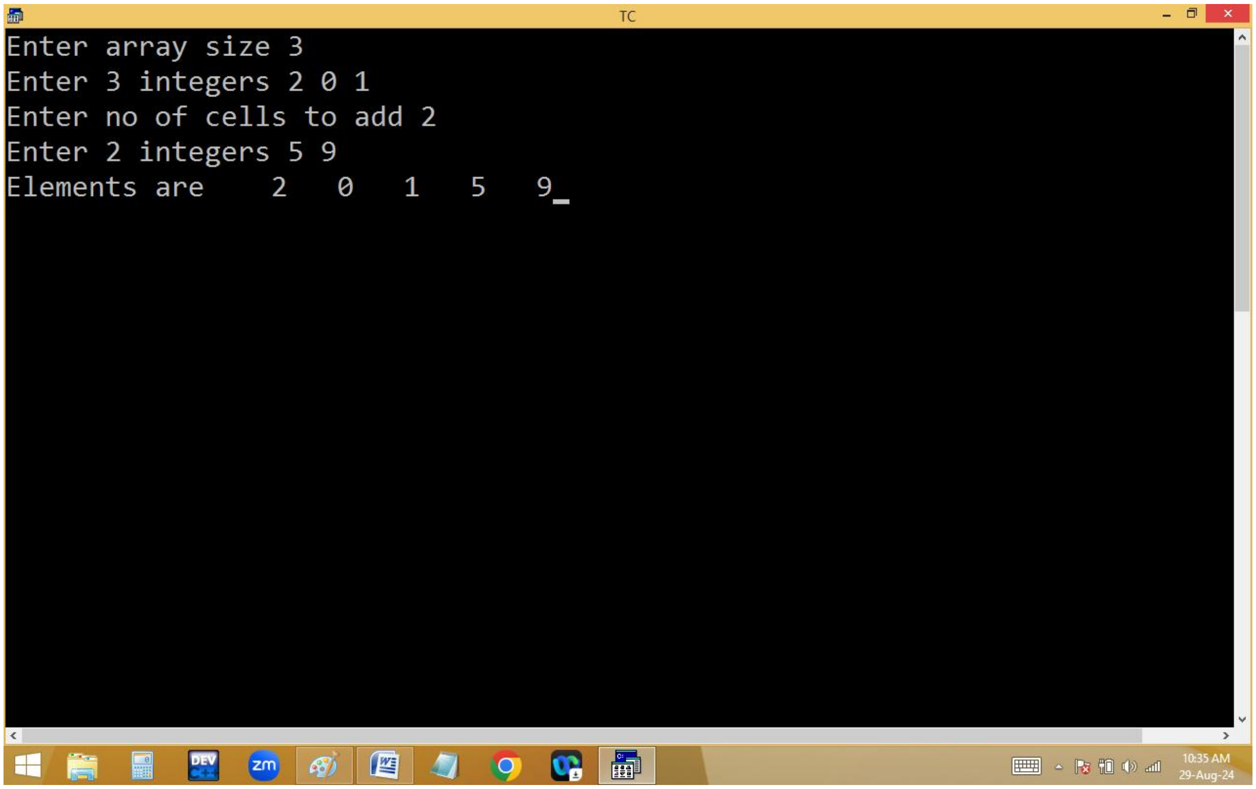


scanf("%d", &p[r][c]); 2010+0*2=2010

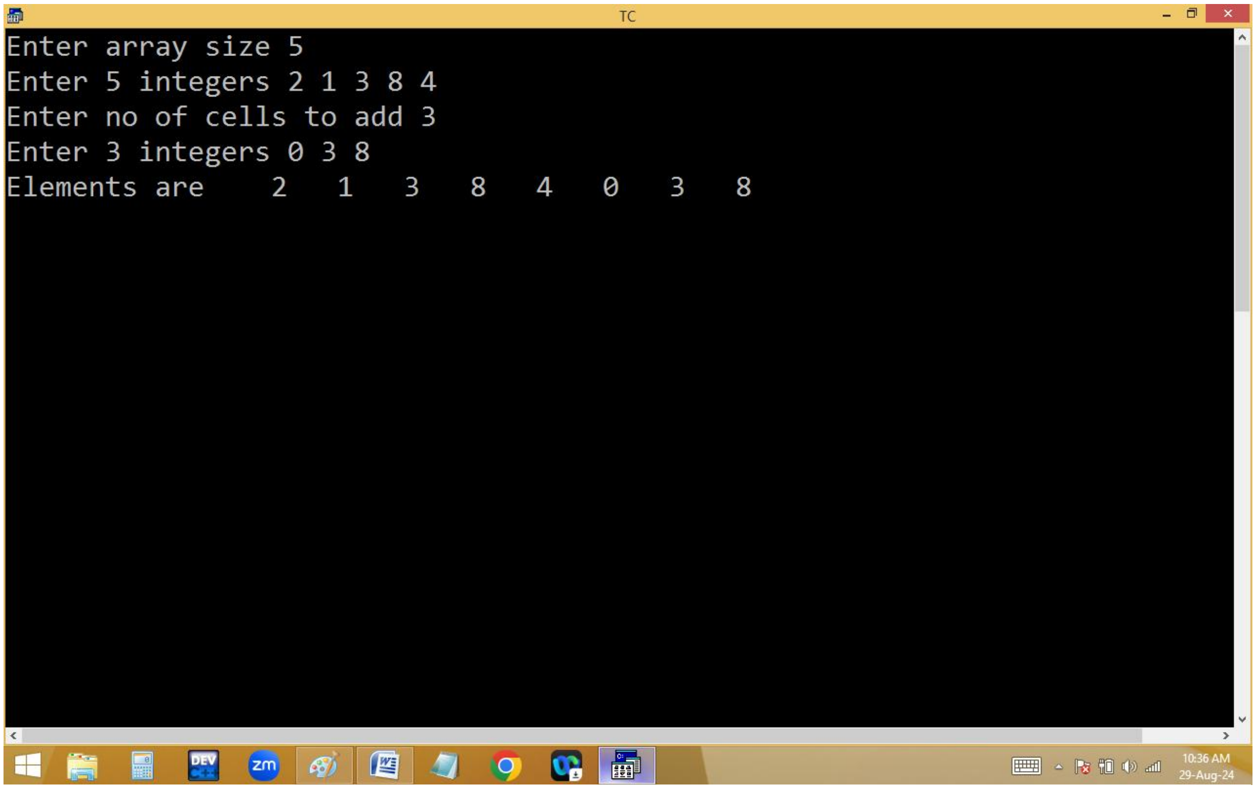
Realloc() example:


TC
File Edit Run Compile Project Options Debug Break/watch
Line 16 Col 16 Insert Indent Tab Fill Unindent * E:9AM.C
10:35 AM
29-Aug-24

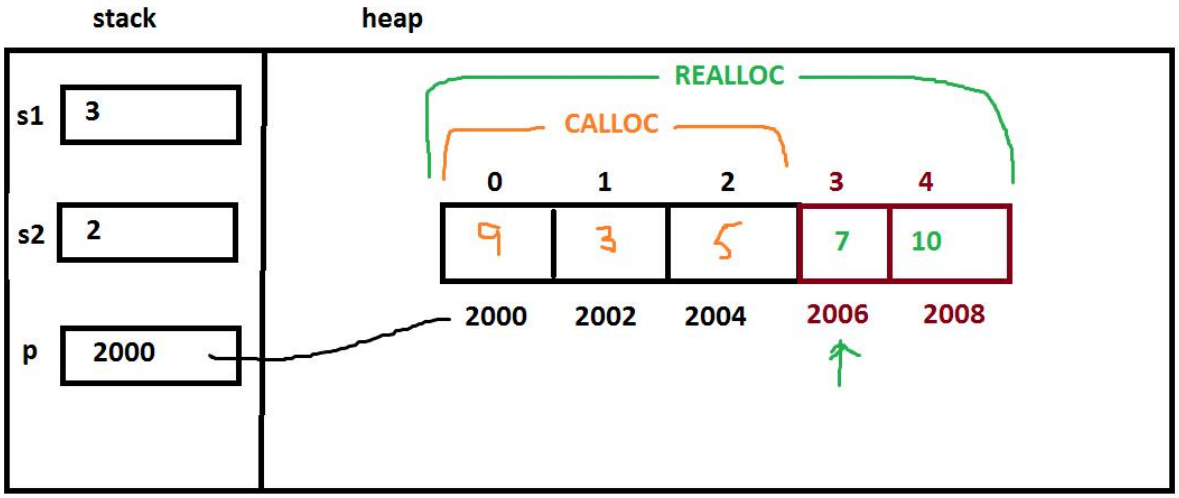
```c
#include<stdio.h>
#include<conio.h>
#include<alloc.h> /* stdlib.h */
void main()
{
int *p, s1, s2, i;
clrscr();
printf("Enter array size ");scanf("%d",&s1);
p = (int *)calloc(s1,sizeof(int));
printf("Enter %d integers ", s1);for(i=0;i<s1;i++)scanf("%d",p+i);
printf("Enter no of cells to add ");scanf("%d",&s2);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



```
Enter array size 3
Enter 3 integers 2 0 1
Enter no of cells to add 2
Enter 2 integers 5 9
Elements are    2   0   1   5   9
```



```
Enter array size 5
Enter 5 integers 2 1 3 8 4
Enter no of cells to add 3
Enter 3 integers 0 3 8
Elements are    2    1    3    8    4    0    3    8
```



USER DEFINED FUNCTIONS

What is a function?

It is a small program is used to do a particular task.

It is a sub program / sub routine / procedure / module / structure.

It is a reusable code component.

It is a self contained block.

It is a small program within another program.

Advantages:

1. modularity: dividing big program into small pieces.
2. Reusability: write once, use many times.
3. Simplicity: easy to read and understand.
4. Efficiency: performance is high.