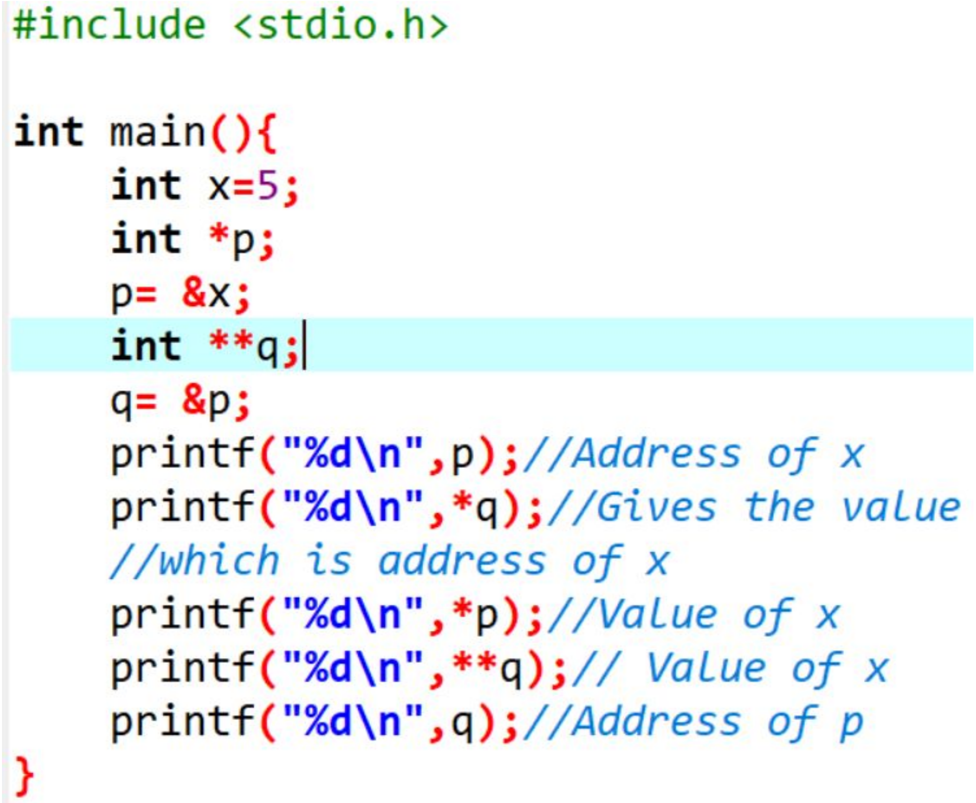
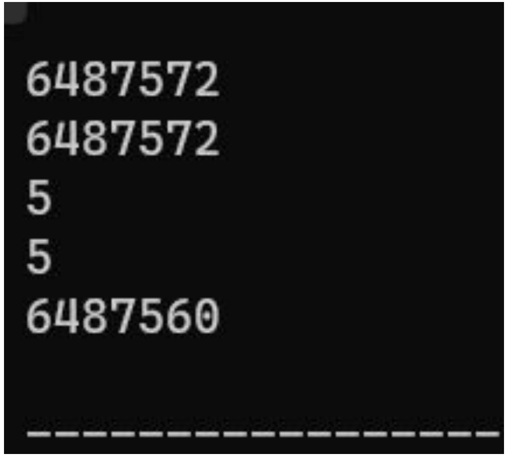
Pointers and Dynamic Memory Allocation

Sulove Bhattarai

Pointers to Pointers

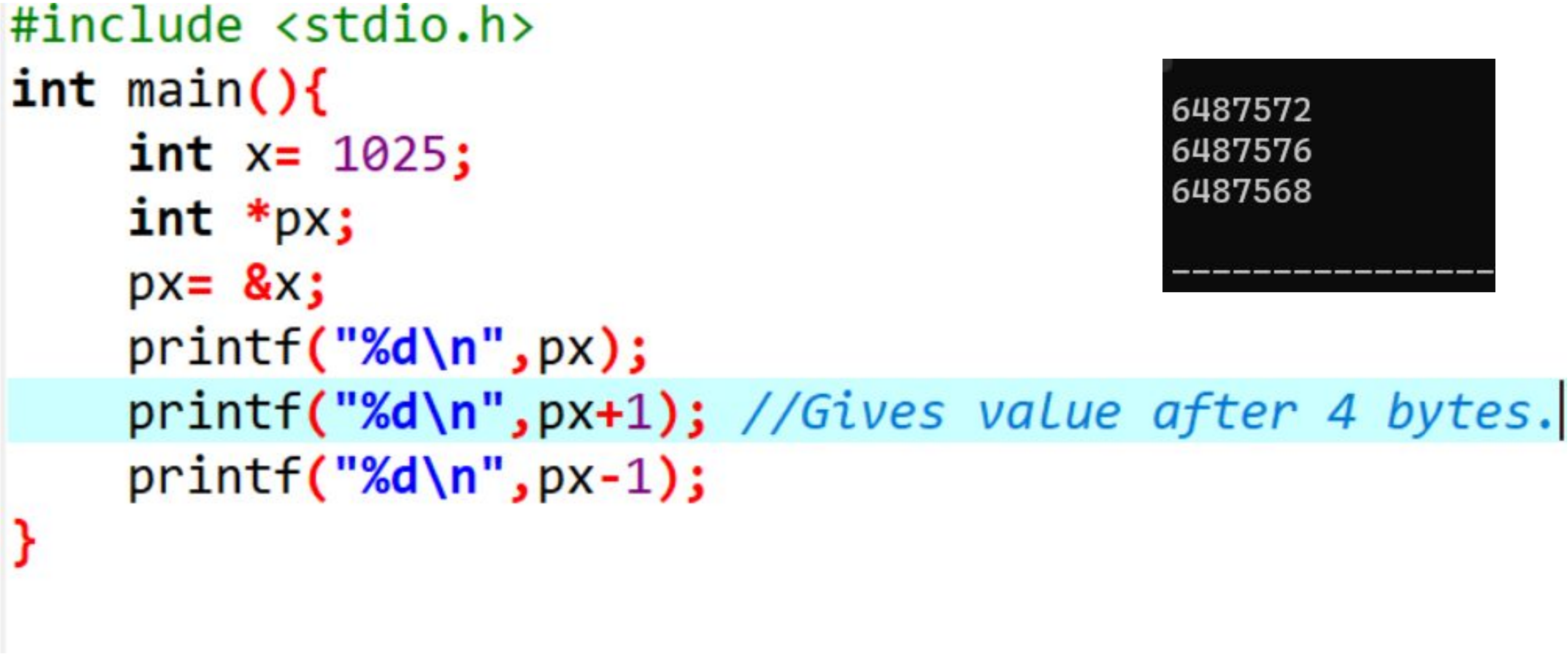


```c
#include <stdio.h>

int main(){
    int x=5;
    int *p;
    p= &x;
    int **q;
    q= &p;
    printf("%d\n",p);//Address of x
    printf("%d\n",*q);//Gives the value
    //which is address of x
    printf("%d\n",*p);//Value of x
    printf("%d\n",**q);// Value of x
    printf("%d\n",q);//Address of p
}
```



```
6487572
6487572
5
5
6487560


-------------------
```

Pointer Arithmetic



```c
#include <stdio.h>
int main(){
    int x= 1025;
    int *px;
    px= &x;
    printf("%d\n",px);
    printf("%d\n",px+1); //Gives value after 4 bytes.
    printf("%d\n",px-1);
}
```

```
6487572
6487576
6487568

---------------
```

Void Pointers

No associated data type

Can point to any data of any data type.

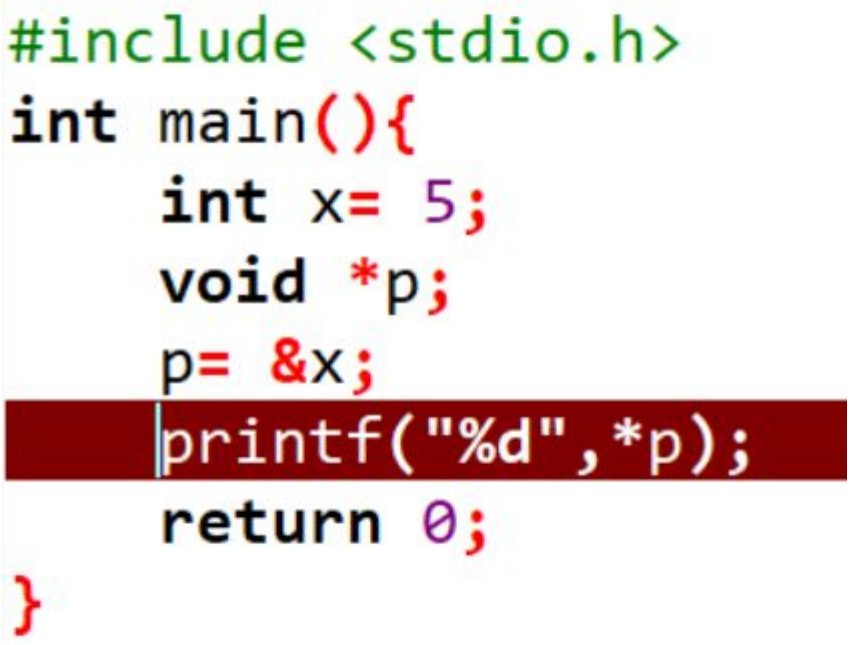
Can be typecasted to any data type.

Cannot dereference a void pointer without typecasting.

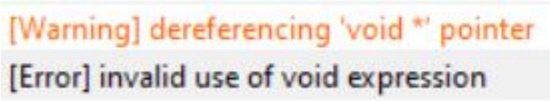
Declaration

```
void  *p;
```

Void pointers

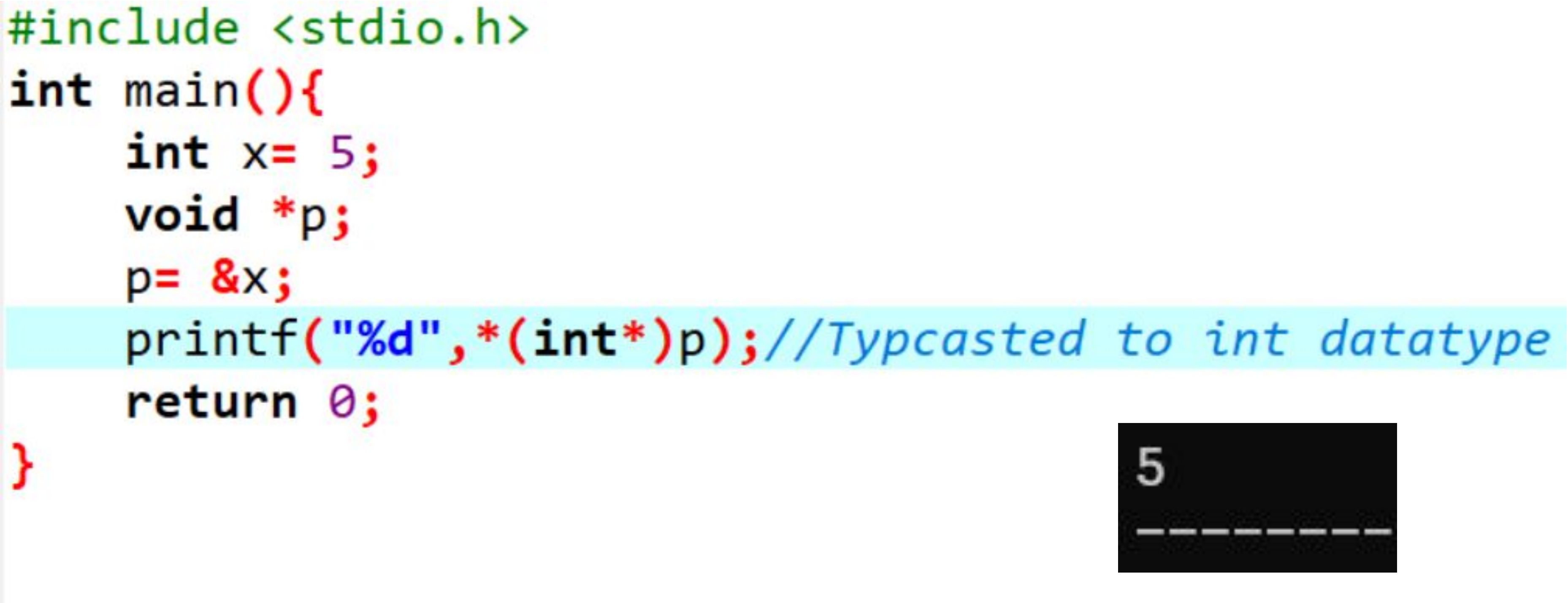


```
#include <stdio.h>
int main(){
    int x= 5;
    void *p;
    p= &x;
    printf("%d",*p);
    return 0;
}
```



```
[Warning] dereferencing 'void *' pointer
[Error] invalid use of void expression
```

Void Pointers



```c
#include <stdio.h>
int main(){
    int x= 5;
    void *p;
    p= &x;
    printf("%d",*(int*)p);//Typcasted to int datatype
    return 0;
}
```

```
5
---------
```

Null Pointer

Does not point to any memory location.

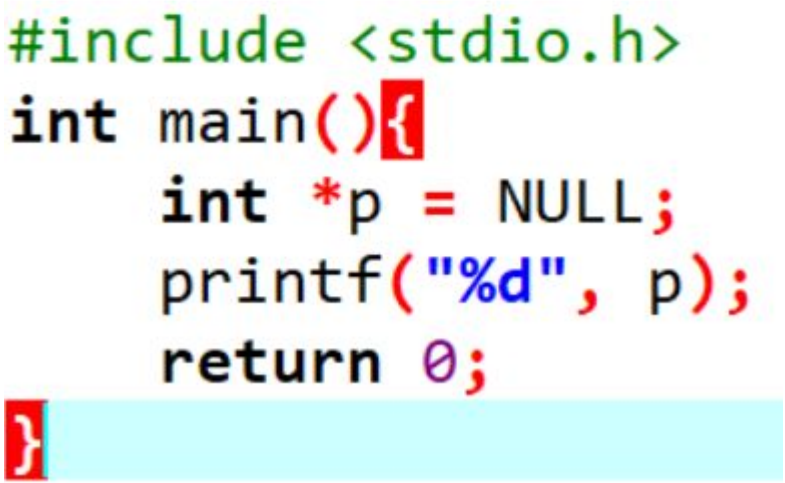
Indication of invalid memory location.

Useful for handling errors

Declaration:

```
int *p= NULL;
```

Null Pointer



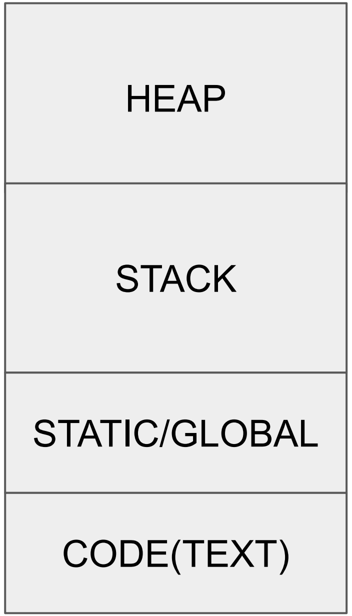
```
#include <stdio.h>
int main(){
    int *p = NULL;
    printf("%d", p);
    return 0;
}
```

Dynamic Memory

Unlike stack, size of the heap can vary during program execution.

Allows to keep variable in the memory till we want.

Using heap= Dynamic memory allocation



HEAP

STACK

STATIC/GLOBAL

CODE(TEXT)

Dynamic Memory Allocation

C programming language provide 4 functions.

malloc

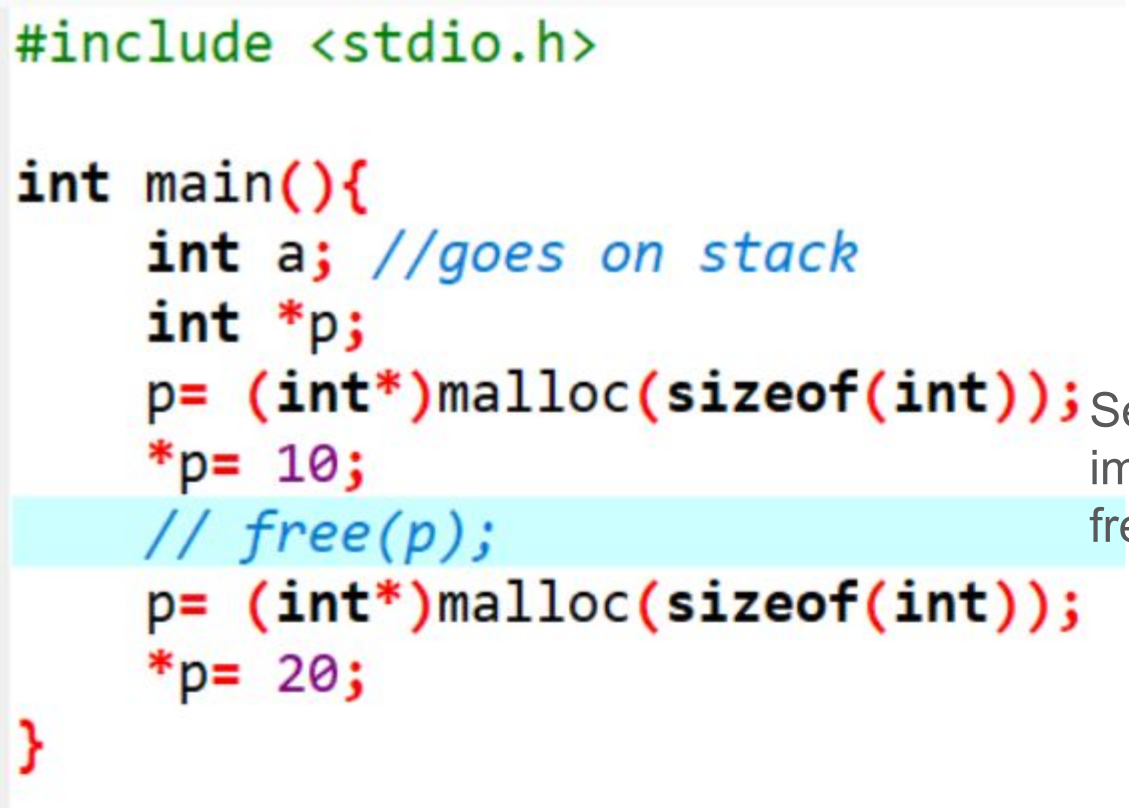
calloc

realloc

free

Dynamic Memory

Only way to access heap memory is through reference.



```c
#include <stdio.h>

int main(){
    int a; //goes on stack
    int *p;
    p= (int*)malloc(sizeof(int));
    *p= 10;
    // free(p);
    p= (int*)malloc(sizeof(int));
    *p= 20;
}
```

See the implementation of free()

malloc()

malloc- void* malloc(size_t size)

Note: size_t is unsigned integer

allocates memory block according to the size specified by the user without knowing the type of data to be stored

returns a void pointer pointing to the first byte of memory

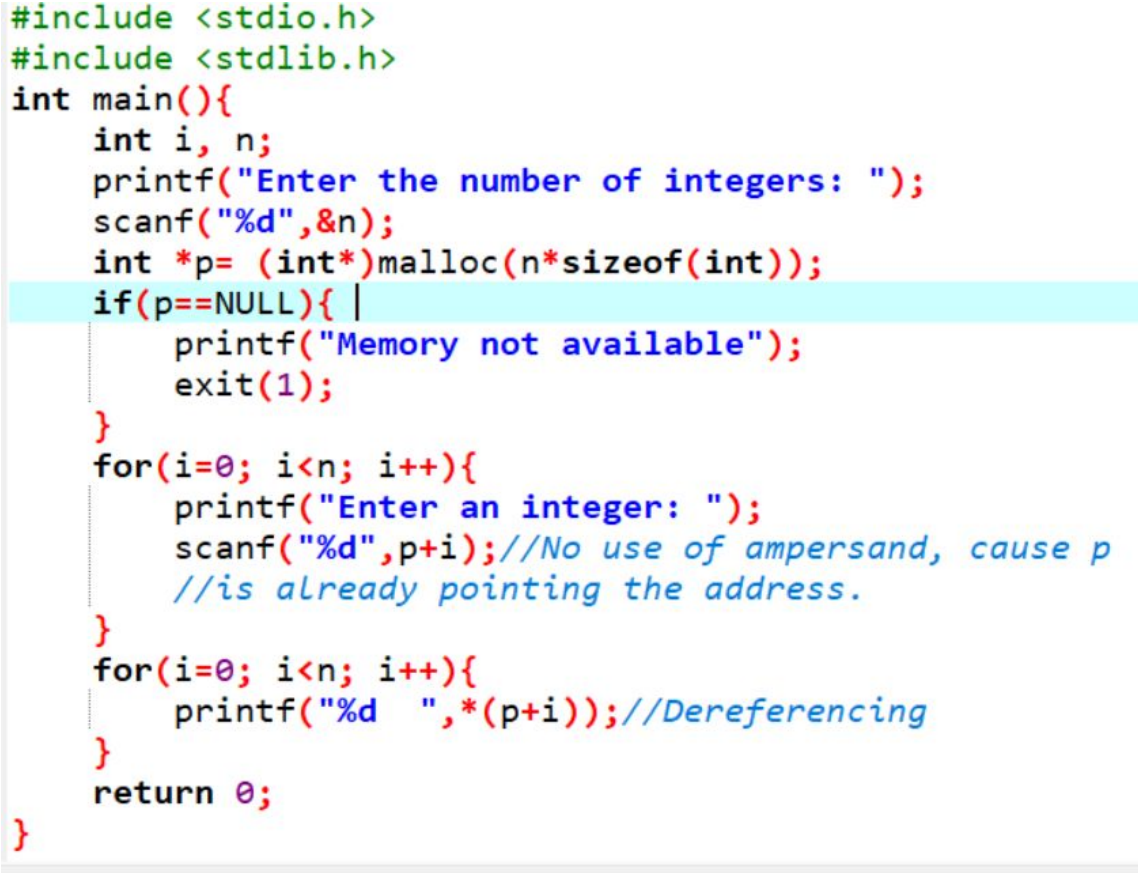
should be typecasted to required datatype.

int *p= (int*) malloc(8)

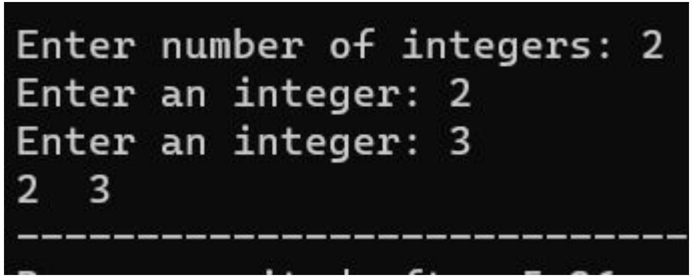
allocates 8 bytes of memory in the heap

Returns NULL when memory is not available in the heap

malloc() (Memory Allocation)

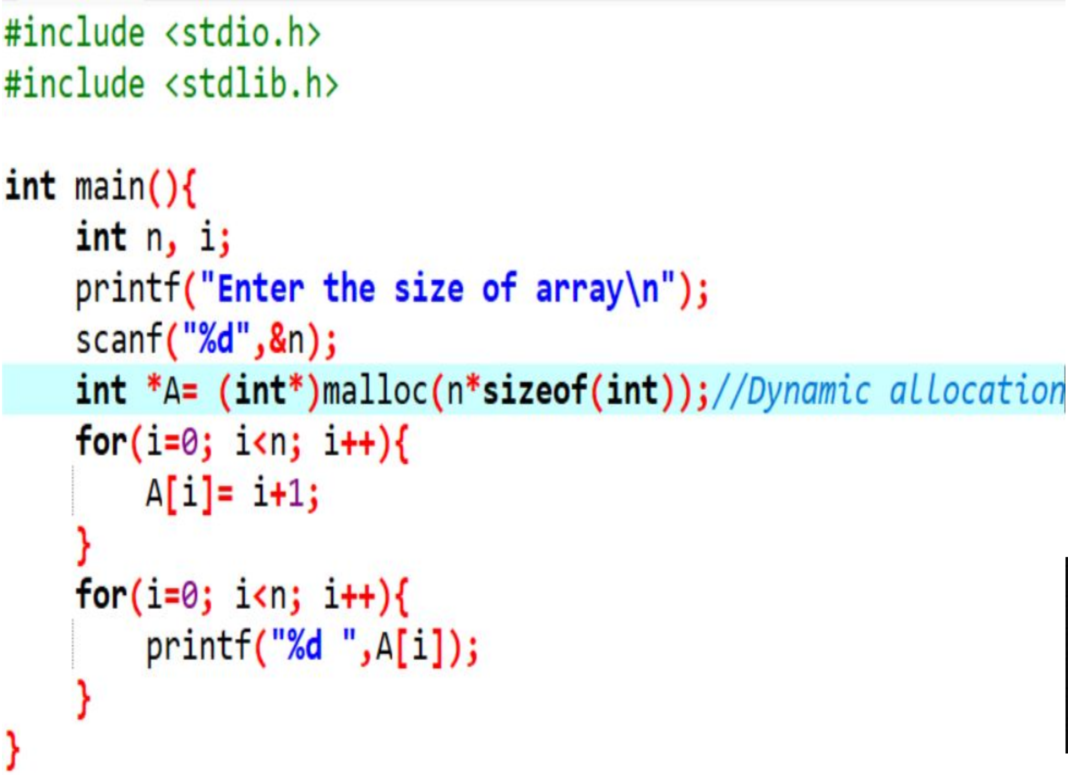


```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i, n;
    printf("Enter the number of integers: ");
    scanf("%d",&n);
    int *p= (int*)malloc(n*sizeof(int));
    if(p==NULL){
        printf("Memory not available");
        exit(1);
    }
    for(i=0; i<n; i++){
        printf("Enter an integer: ");
        scanf("%d",p+i);//No use of ampersand, cause p
        //is already pointing the address.
    }
    for(i=0; i<n; i++){
        printf("%d  ",*(p+i));//Dereferencing
    }
    return 0;
}
```

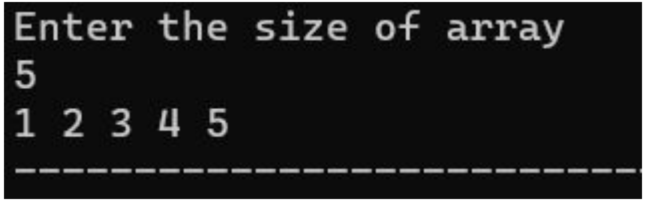


```
Enter number of integers: 2
Enter an integer: 2
Enter an integer: 3
2  3
----------------------------
```

malloc()



```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int n, i;
    printf("Enter the size of array\n");
    scanf("%d",&n);
    int *A= (int*)malloc(n*sizeof(int));//Dynamic allocation
    for(i=0; i<n; i++){
        A[i]= i+1;
    }
    for(i=0; i<n; i++){
        printf("%d ",A[i]);
    }
}
```



```
Enter the size of array
5
1 2 3 4 5
--------------------------
```

calloc()

allocate multiple blocks of memory

needs two arguments

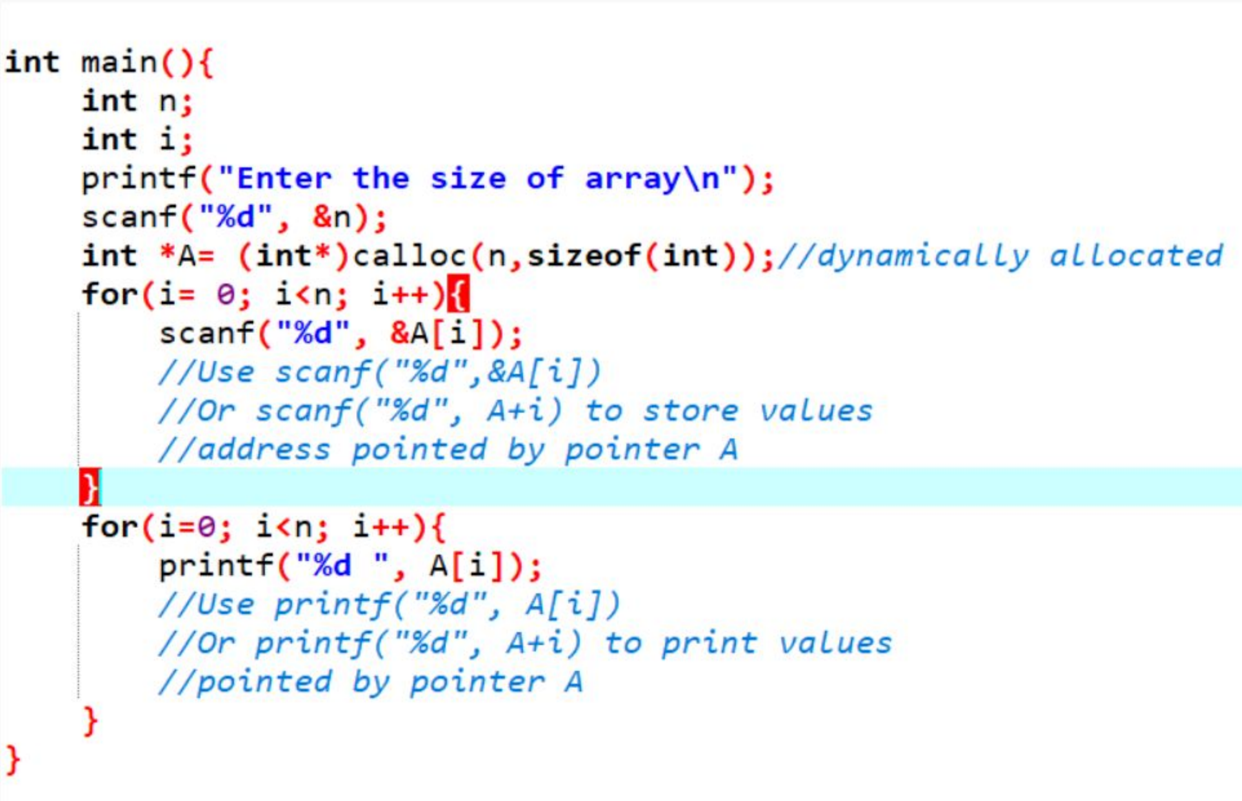
Syntax: void *calloc()(size_t n, size_t size);

size_t n -> Number of blocks

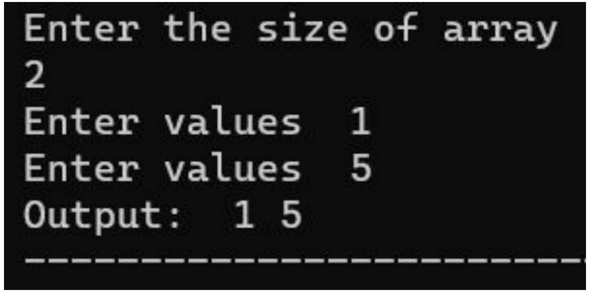
size_t size-> size of each block

Memory initialized to zero if calloc() is used while malloc() has garbage values.

calloc()

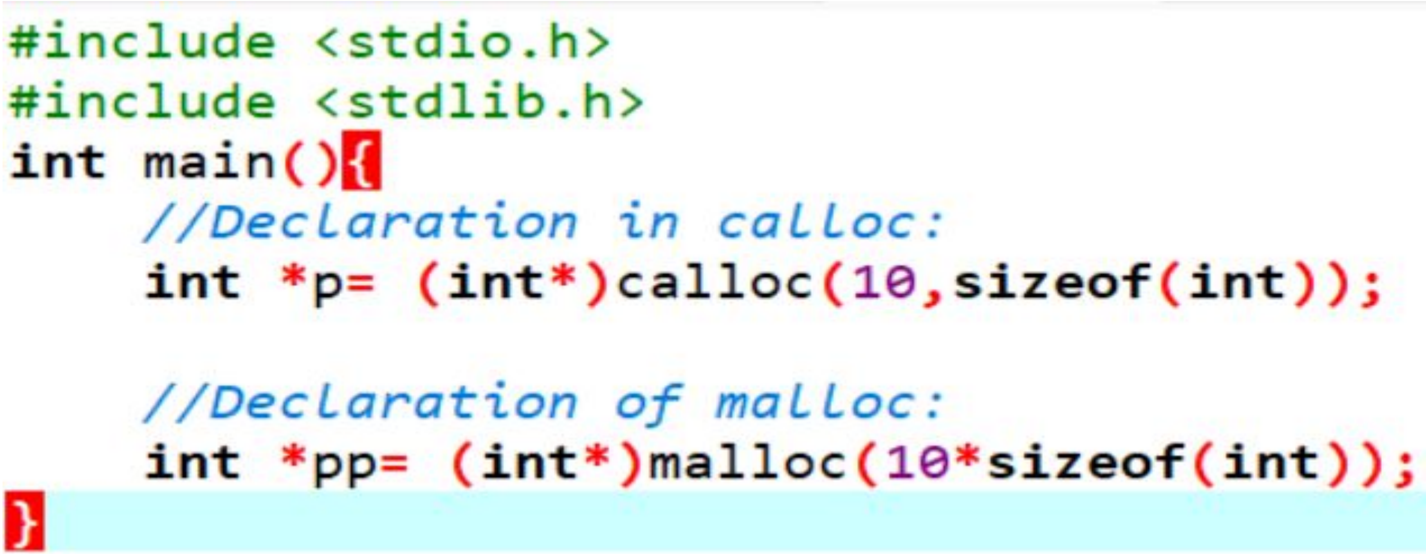


```
int main(){
    int n;
    int i;
    printf("Enter the size of array\n");
    scanf("%d", &n);
    int *A= (int*)calloc(n,sizeof(int));//dynamically allocated
    for(i= 0; i<n; i++){
        scanf("%d", &A[i]);
        //Use scanf("%d",&A[i])
        //Or scanf("%d", A+i) to store values
        //address pointed by pointer A
    }
    for(i=0; i<n; i++){
        printf("%d ", A[i]);
        //Use printf("%d", A[i])
        //Or printf("%d", A+i) to print values
        //pointed by pointer A
    }
}
```



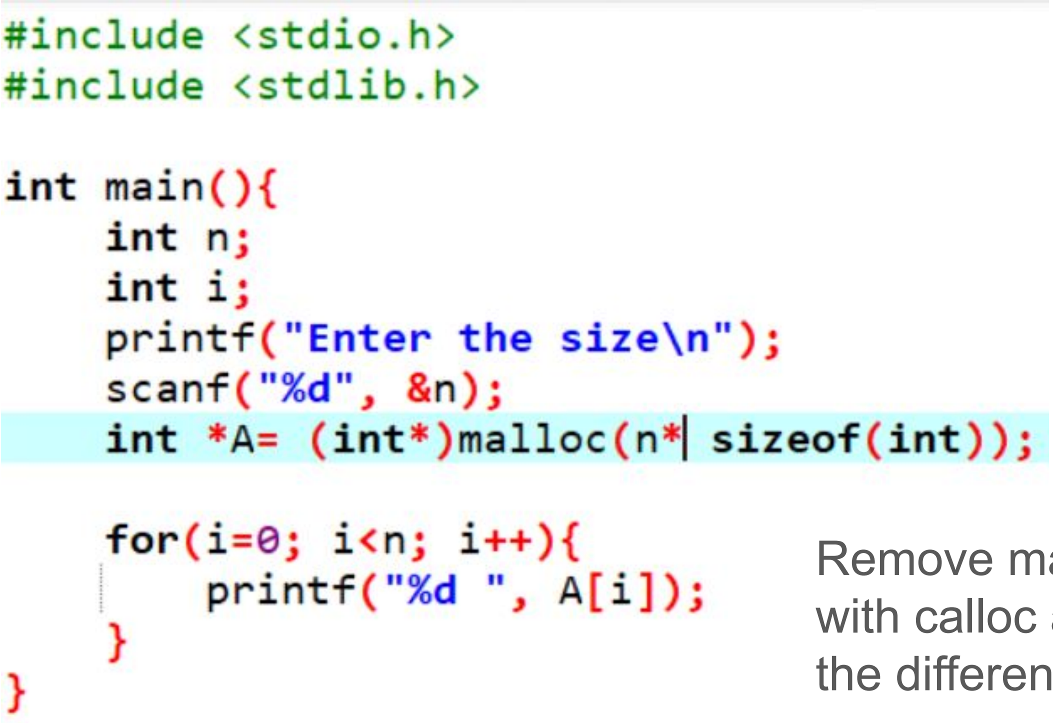
```
Enter the size of array
2
Enter values  1
Enter values  5
Output:  1 5
--------------------------
```

malloc() vs calloc()



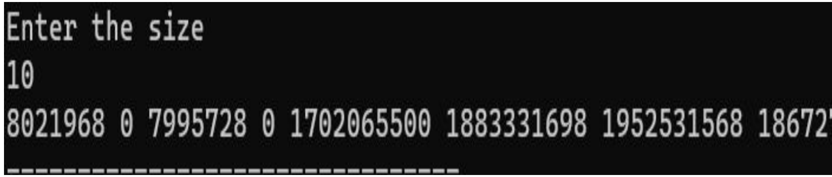
```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    //Declaration in calloc:
    int *p= (int*)calloc(10,sizeof(int));

    //Declaration of malloc:
    int *pp= (int*)malloc(10*sizeof(int));
}
```

malloc() vs calloc()



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

int main(){
    int n;
    int i;
    printf("Enter the size\n");
    scanf("%d", &n);
    int *A= (int*)malloc(n* sizeof(int));

    for(i=0; i<n; i++){
        printf("%d ", A[i]);
    }
}
```

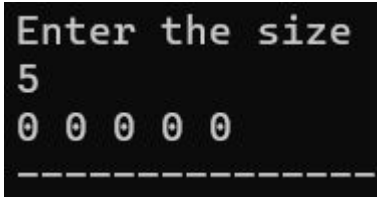
Output of malloc:



```
Enter the size
10
8021968 0 7995728 0 1702065500 1883331698 1952531568 18672
--------------------------------
```

Remove malloc with calloc and see the difference.

Output of calloc:



```
Enter the size
5
0 0 0 0 0
---------------
```

realloc()

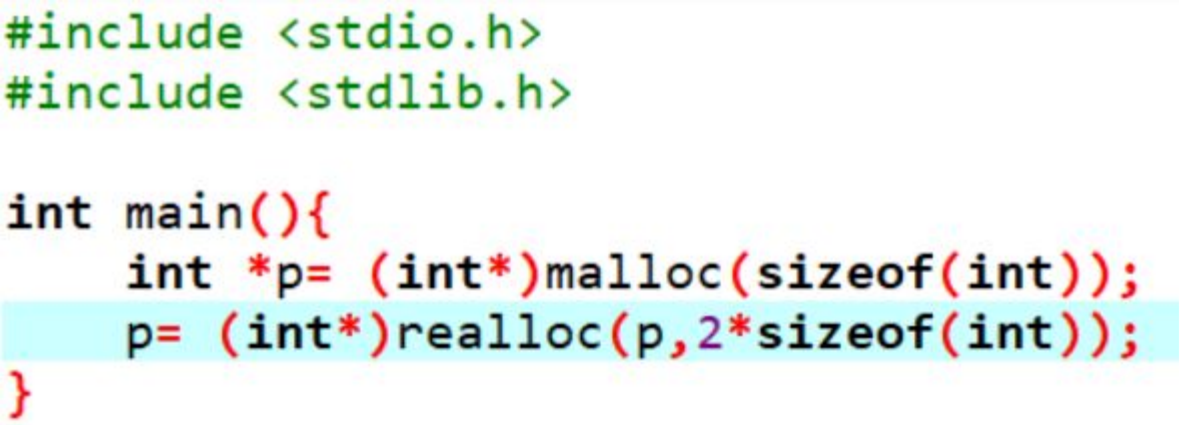
change the size of memory block without losing previous data

Syntax: void *realloc(void *p, size_t size);

void *p-> pointer the previously allocated memory

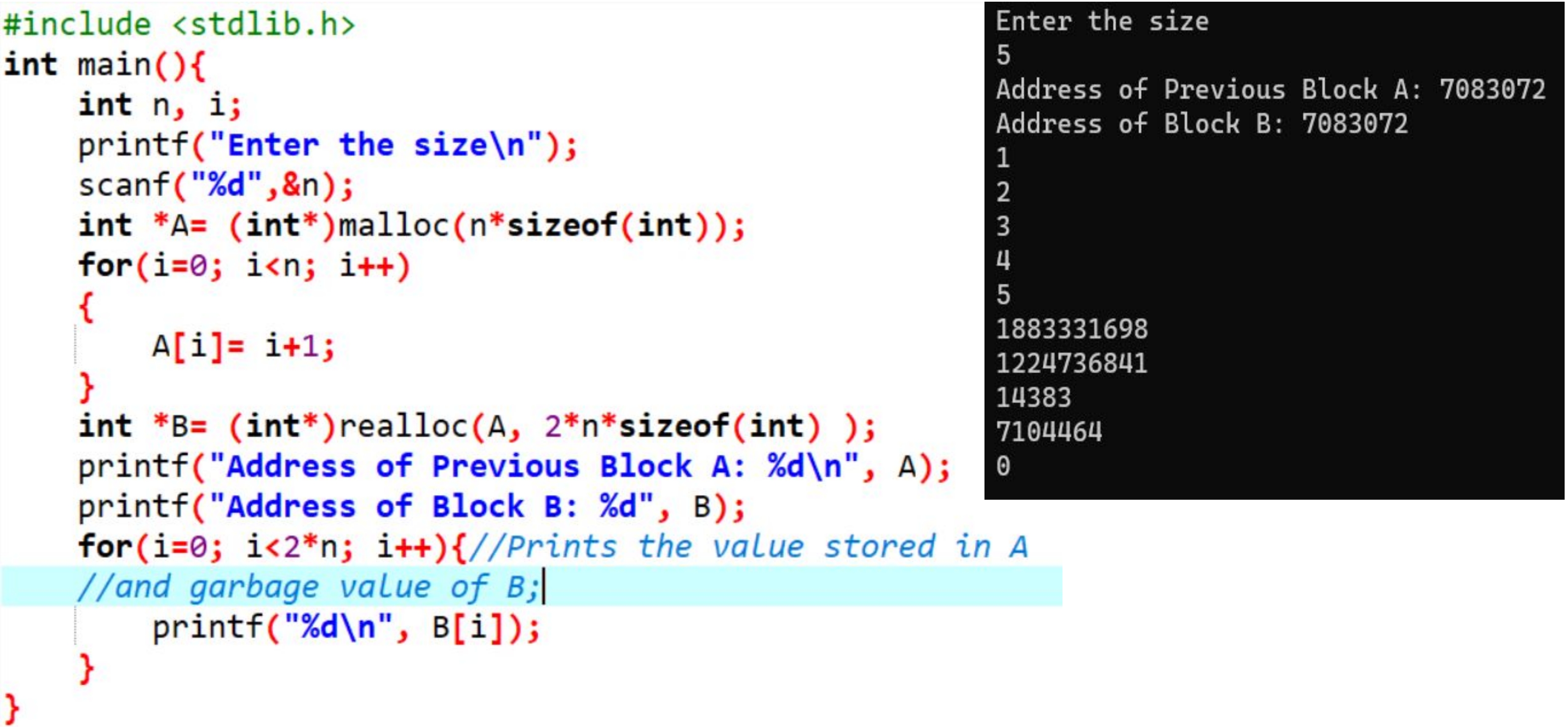
size_t size -> new size for the memory

contents of old block is not lost while creating new block



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

int main(){
    int *p= (int*)malloc(sizeof(int));
    p= (int*)realloc(p,2*sizeof(int));
}
```

realloc()



```c
#include <stdlib.h>
int main(){
    int n, i;
    printf("Enter the size\n");
    scanf("%d",&n);
    int *A= (int*)malloc(n*sizeof(int));
    for(i=0; i<n; i++)
    {
        A[i]= i+1;
    }
    int *B= (int*)realloc(A, 2*n*sizeof(int) );
    printf("Address of Previous Block A: %d\n", A);
    printf("Address of Block B: %d", B);
    for(i=0; i<2*n; i++){//Prints the value stored in A
    //and garbage value of B;
        printf("%d\n", B[i]);
    }
}
```

```
Enter the size
5
Address of Previous Block A: 7083072
Address of Block B: 7083072
1
2
3
4
5
1883331698
1224736841
14383
7104464
0
```

THANK YOU!!