

Page _____

C-136 \Rightarrow Deallocating the Dynamically Allocated Memory using free()

* free() function will release the dynamically allocated memory.

* It is built-in function defined inside stdlib.h

Syntax:-

void free(pointer)



It does not
return anything.

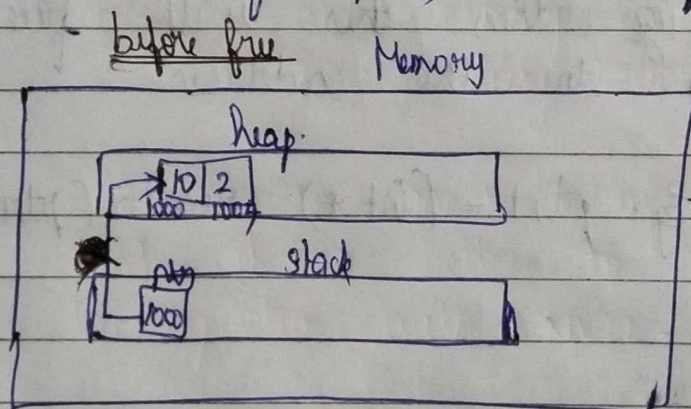
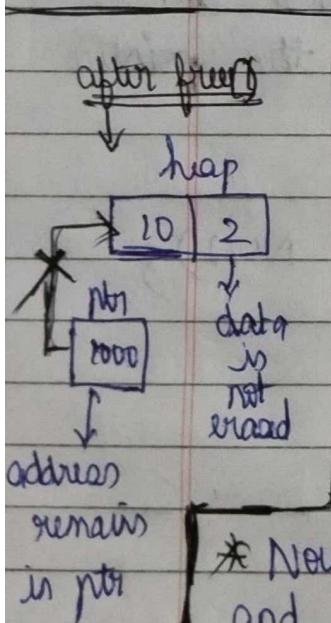
Eg:- int *ptr;

ptr = (int *) malloc(2 * sizeof(int));

free(ptr);

~~if (ptr == NULL)~~

printf("Memory not allocated");



* Now we have allocated memory in heap and we access those memory to store and retrieve the values using pointers which store the base address of allocated heap memory.

Date: _____
Page: _____

* After this we use free function to free the allocated memory. This heap which can be used by some other programs.

NOTE

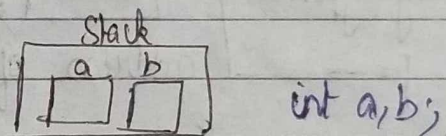
* Now the question is, once we free the memory will the contents stored in the heap memory will be erased? The answer is no.

* New second question is, can you access the data again? The answer is it will show undefined behaviour.

→ undefined behaviour means first thing its not allowed to use the pointer after freeing that pointer. May be you will get any garbage value or same value or any value.

→ So it is not safe to use the pointer after it is freed. (dangerous)

NOTE:



* In stack if you allocate memory for two variables, say a & b then the allocated memory will be released automatically after program execution.

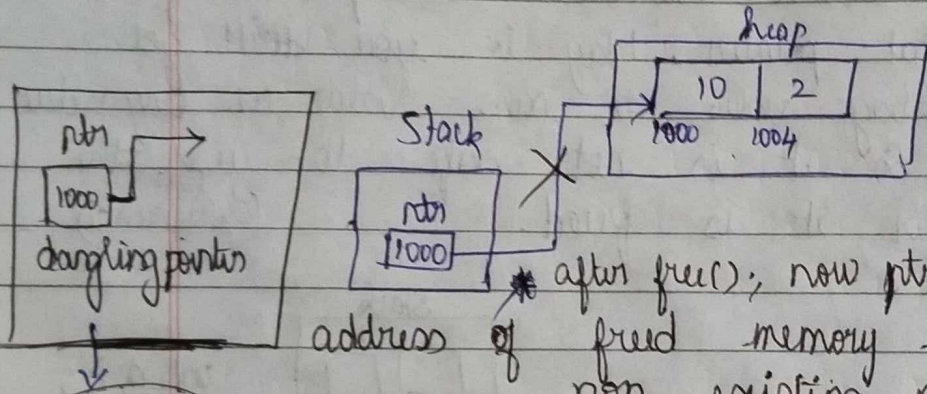
* But in dynamic Memory allocation it is our responsibility to free the allocated memory in heap section since it do not release memory automatically in heap section and hence we use free() function to free the allocated memory.

* If you don't free the memory; at some point of time memory will be exhausted or system crashes.

* To not that within the program we can write `free()` function only at the end, we can also write it between and we can also use more than one free function but to handle this you should know how to use it while implementing it is a program.

NOTE:

* After freeing the ~~pointer~~ memory still the pointer contains the address of the freed memory location; so now this pointer acts as a dangling pointer.



* after `free()`; now ptr still contains address of freed memory location (i.e) non existing memory location.

* So we cannot de-reference this pointer, and it is better to re-initialize this pointer with `NULL`.

```
free(ptr);
ptr = NULL;
free(ptr);
```

// printf("%d", (*ptr + 0));
↓
Shows undefined behaviour
Double free error / or Core dump error

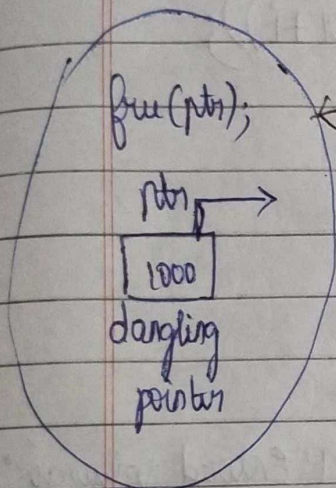
Program 1:-

```
int main()
{
```

```
    int i, *ptr;
    ptr = (int *) malloc(3 * sizeof(int));
    if (ptr == NULL)
```

```
    {
        printf("Memory not allocated");
    }
```

```
    printf("Enter the values:");
    for (i = 0; i < 3; i++)
```



```
        scanf("%d", *(ptr + i));
    }
```

```
    free(ptr); // ptr == NULL
    printf("The entered values are:");
    for (i = 0; i < 3; i++)
```

```
        printf("%d", *(ptr + i));
    }
```

no need again. // free(ptr) → because we have already freed this pointer, so comment it.

```
    return 0;
}
```

O/P

Enter the values: 1 2 3 .

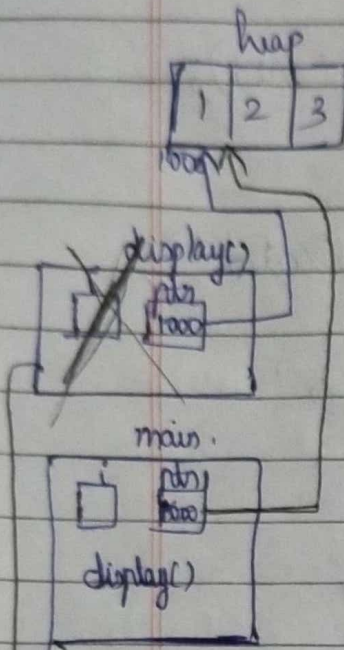
Entered values : 414141 131213 3

↓ ↓
garbage values

↳ (i) It is undefined behavior because we are ~~accessing~~ accessing the dangling pointer.

Program 2

~~int * display~~



```
int* display()
```

```
{
    int i, *ptr;
    ptr = (int*) malloc(3 * sizeof(int));
    printf("Enter the values");
    for(i=0; i<3; i++)
```

```
scanf("%d", (ptr+i));
```

```
return ptr;
```

```
int main()
```

```
{
    int i, *ptr1;
```

```
ptr1 = ptr; printf("Entered values are:");
for(i=0; i<3; i++)
```

```
printf("%d\t", *(ptr1+i));
```

```
free(ptr1);
```

```
return 0;
```

```
}
```

→ Once gets out from display(),

memory for

display is

vanished

and we cannot

access 'ptr' from

main. So we store

the address in

main

O/p:-

Enter Values: 4 5 6

Entered Values: 4 5 6

* In case of static memory allocation; if we try to access the values which is defined within display() function from

Date _____
Page _____

main() function we cannot do that and the pointer becomes dangling pointer since the memory allocated for variables within the display() will get vanished once comes out of the display().

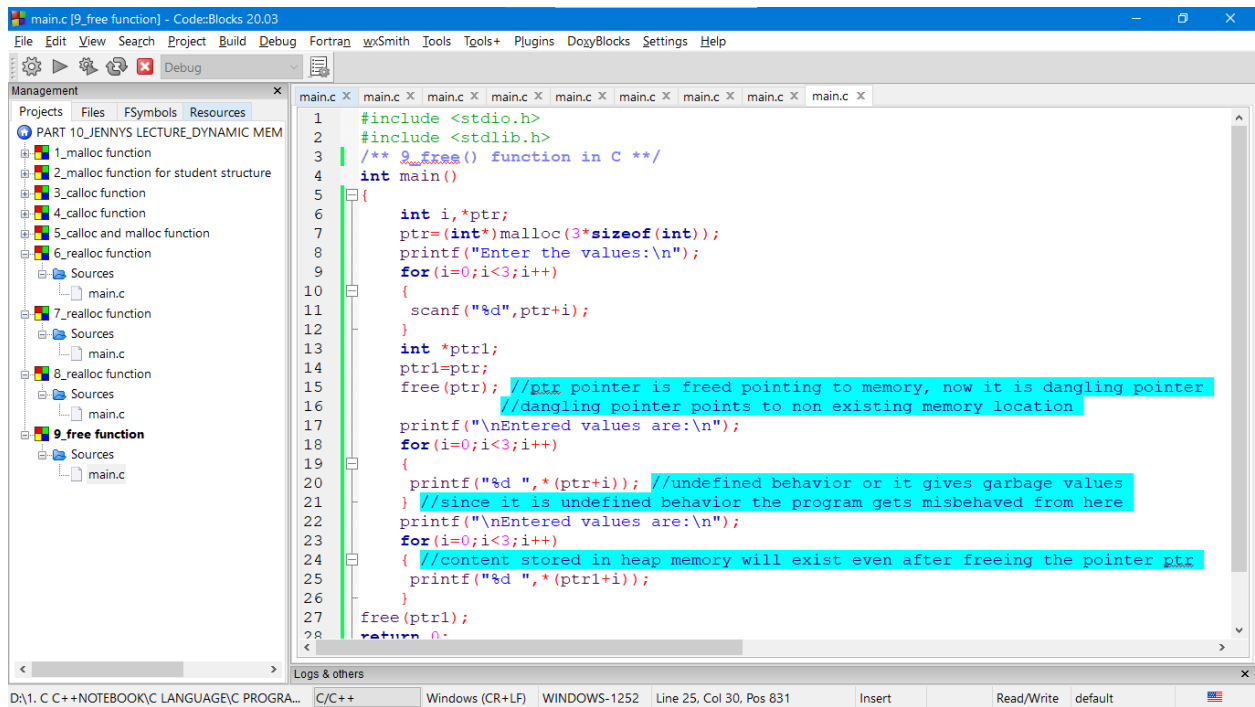
* But in Dynamic Memory allocation since we return our ~~pointer~~ address of the allocated heap section to display function we can access the values of heap section even though the display() function is vanished once it comes out of display after execution.

NOTE:-

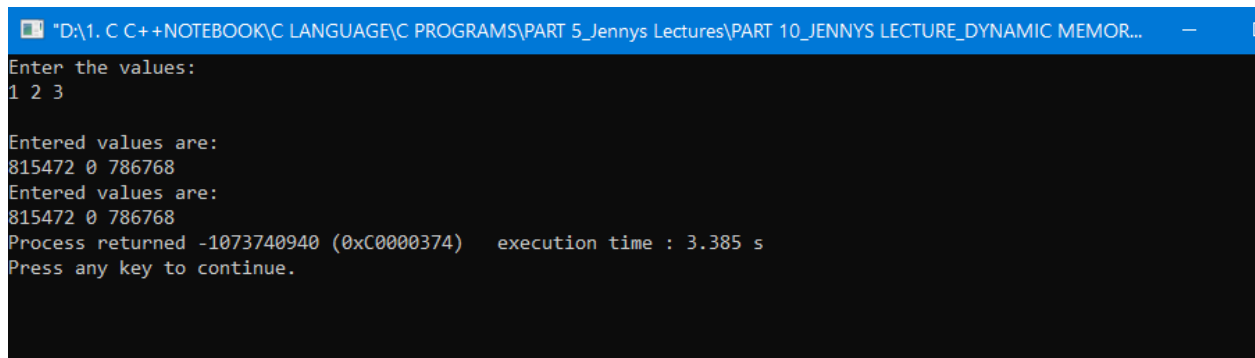
* This programs, says that even though we free a pointer pointing to a allocated heap section memory; then the contents stored in those allocated memory is not going to get vanished.

* Next thing we have to free the pointer pointing to heap memory which is done using DMA Concept; since DMA will not automatically free the memory which can be used by other programs or else the memory gets exhausted and as a result the system crashes or shut down.

* But SMA will automatically de-allocate the memory.



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 /** 9 free() function in C **/
4 int main()
5 {
6     int i,*ptr;
7     ptr=(int*)malloc(3*sizeof(int));
8     printf("Enter the values:\n");
9     for(i=0;i<3;i++)
10     {
11         scanf("%d",ptr+i);
12     }
13     int *ptr1;
14     ptr1=ptr;
15     free(ptr); //ptr pointer is freed pointing to memory, now it is dangling pointer
16               //dangling pointer points to non existing memory location
17     printf("\nEntered values are:\n");
18     for(i=0;i<3;i++)
19     {
20         printf("%d ",*(ptr+i)); //undefined behavior or it gives garbage values
21     } //since it is undefined behavior the program gets misbehaved from here
22     printf("\nEntered values are:\n");
23     for(i=0;i<3;i++)
24     { //content stored in heap memory will exist even after freeing the pointer ptr
25         printf("%d ",*(ptr1+i));
26     }
27     free(ptr1);
28     return 0;
29 }
```



```
"D:\1. C C++NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 10_JENNYS LECTURE_DYNAMIC MEMOR...
Enter the values:
1 2 3

Entered values are:
815472 0 786768
Entered values are:
815472 0 786768
Process returned -1073740940 (0xC0000374)   execution time : 3.385 s
Press any key to continue.
```

main.c [10_free function] - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Management

Projects Files FSymbols Resources

PART 10_JENNYNS LECTURE_DYNAMIC

- 1_malloc function
- 2_malloc function for student stru
- 3_calloc function
- 4_calloc function
- 5_calloc and malloc function
- 6_realloc function
 - Sources
 - main.c
- 7_realloc function
 - Sources
 - main.c
- 8_realloc function
 - Sources
 - main.c
- 9_free function
 - Sources
 - main.c
- 10_free function**
 - Sources
 - main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 /** 10-free() function in C */
4 int* returnPointer()
5 {
6     int i,*ptr;
7     ptr=(int*)malloc(3*sizeof(int));
8     printf("Enter the values:\n");
9     for(i=0;i<3;i++)
10     {
11         scanf("%d",ptr+i);
12     }
13     return ptr;
14     //ptr is local to this function and it gets deallocated once it comes out this function
15 }
16 int main()
17 {
18     int i,*ptr1;
19     ptr1=returnPointer();
20     printf("Entered values are:\n");
21     for(i=0;i<3;i++)
22     {
23         //even though heap memory is freed from pointer ptr, contents stored in heap still exist
24         printf("%d ",*(ptr1+i));
25     }
26     return 0;
27 }
28
```

Logs & others

D:\1. C C++\NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 10_JENNYNS LECTURE_DYNAMIC MEMOR... C/C++ Windows (CR+LF) WINDOWS-1252 Line 24, Col 16, Pos 592 Insert Read/Write default

"D:\1. C C++\NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 10_JENNYNS LECTURE_DYNAMIC MEMOR..."

Enter the values:

1 2 3

Entered values are:

1 2 3

Process returned 0 (0x0) execution time : 2.397 s

Press any key to continue.

The screenshot shows the Code::Blocks IDE with a C program titled "main.c [9_free function] - Code::Blocks 20.03". The program is a demonstration of memory management, specifically focusing on the free function and dangling pointers. The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 /** 9-free() function in C */
4 int main()
5 {
6     int i,*ptr;
7     ptr=(int*)malloc(3*sizeof(int));
8     printf("Enter the values:\n");
9     for(i=0;i<3;i++)
10     {
11         scanf("%d",ptr+i);
12     }
13     int *ptr1;
14     ptr1=ptr;
15     free(ptr); //ptr pointer is freed pointing to memory, now it is dangling pointer
16               //dangling pointer points to non existing memory location
17     /** printf("\nEntered values are:\n");
18         for(i=0;i<3;i++)
19         {
20             printf("%d ",*(ptr+i)); //undefined behavior or it gives garbage values
21         } //since it is undefined behavior the program gets misbehaved from here */
22     printf("\nEntered values are:\n");
23     for(i=0;i<3;i++)
24     { //content stored in heap memory will exist even after freeing the pointer ptr
25         printf("%d ",*(ptr1+i));
26     }
27     free(ptr1);
28     return 0;
29 }
```

The program's output is shown in the terminal window below the IDE:

The terminal window shows the execution of the C program. It prompts the user to "Enter the values:" and the user enters "1 2 3". The program then prints "Entered values are:" followed by "1 2 3". The process returns 0 (0x0) and the execution time is 3.131 s. The prompt "Press any key to continue." is displayed at the bottom.