C_136 ⇒ Deallocating the Dynamically Allocated Memory using free()

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

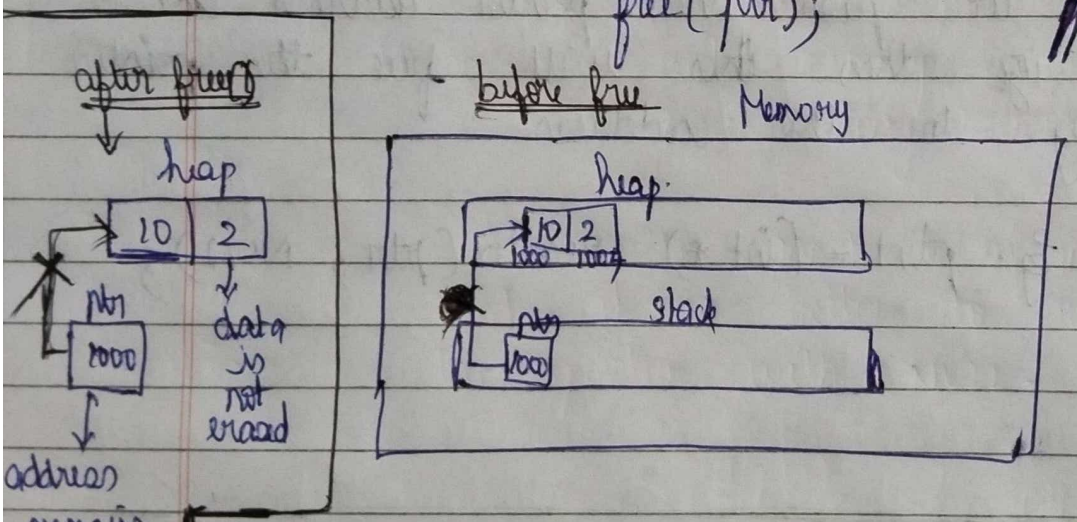
* It is built-in function defined inside stdlib.h

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 is heap and we access those memory to store and retrieve the values using pointers which stores the base address of allocated heap memory.

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

NOTE

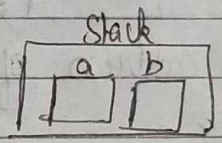
* Now the question is, once we free the memory will the contents stored in the heap memory will be erased? The answer is no.

* Now 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. (dereference)

NOTE:



int a,b;

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

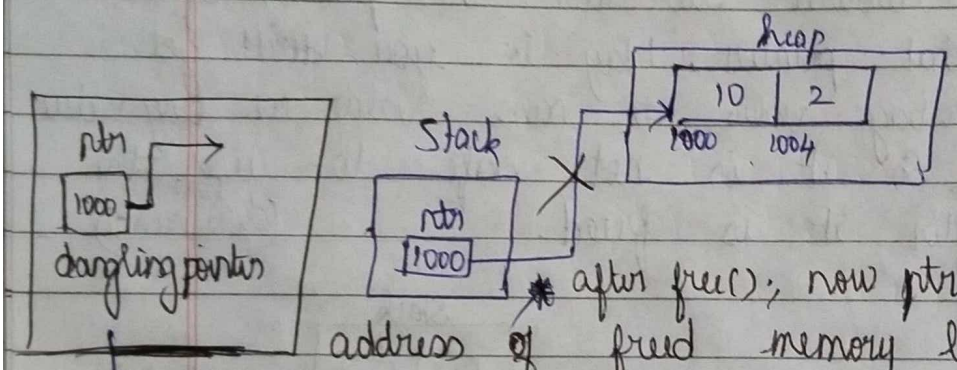
* But in dynamic Memory allocation it is our responsibility to free the allocated memory in heap section since it do not release memory automatically in heap section and hence we use free() function to free the allocated memory.

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

* Its not that within the program we can write free() function only at the end, we can also write in 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 in 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 containing address of freed memory location (i.e) non existing memory location.

pointing to 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);
```

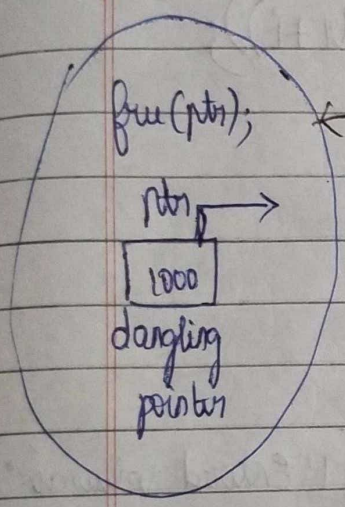
// printf("%d", (*ptr+0));
↓ Shows undefined behaviour

↓ Double free error (or) Core dump error

Date
Page

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));
    }
    free(ptr) -> because we have already freed this pointer; so comment it.
    return 0;
}
```



no need again.

O/P

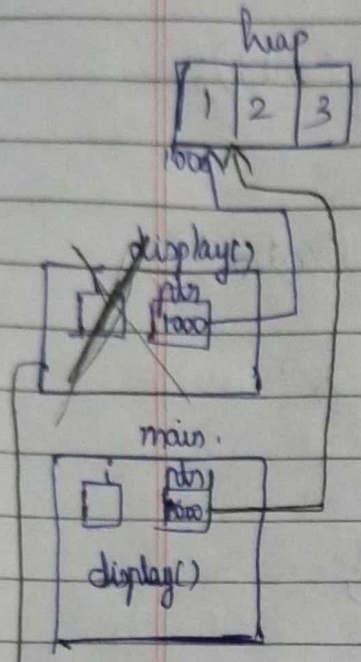
Enter the values: 1 2 3.

Entered values : 4141112 1312113 3

garbage values

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

Program 2



→ once gets out from display(), memory for display is vanished and we cannot access 'ptr' from main. So we store the address in main

```
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", *(ptr+i));
    }
    free(ptr1);
    return 0;
}
```

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

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 downs.

* But SMA will automatically de-allocate the memory.

main.c [9_free function] - Code::Blocks 20.03

```c
#include <stdio.h>
#include <stdlib.h>
/** 9_free() function in C **/
int main()
{
    int i,*ptr;
    ptr=(int*)malloc(3*sizeof(int));
    printf("Enter the values:\n");
    for(i=0;i<3;i++)
    {
     scanf("%d",ptr+i);
    }
    int *ptr1;
    ptr1=ptr;
    free(ptr); //ptr pointer is freed pointing to memory, now it is dangling pointer
               //dangling pointer points to non existing memory location
    printf("\nEntered values are:\n");
    for(i=0;i<3;i++)
    {
     printf("%d ",*(ptr+i)); //undefined behavior or it gives garbage values
    } //since it is undefined behavior the program gets misbehaved from here
    printf("\nEntered values are:\n");
    for(i=0;i<3;i++)
    { //content stored in heap memory will exist even after freeing the pointer ptr
     printf("%d ",*(ptr1+i));
    }
free(ptr1);
return 0;
```

"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

```c
#include <stdio.h>
#include <stdlib.h>
/** 10-free() function in C **/
int* returnPointer()
{
    int i,*ptr;
    ptr=(int*)malloc(3*sizeof(int));
    printf("Enter the values:\n");
    for(i=0;i<3;i++)
    {
     scanf("%d",ptr+i);
    }
    return ptr;
 //ptr is local to this function and it gets deallocated once it comes out this function
}
int main()
{
    int i,*ptr1;
    ptr1=returnPointer();
    printf("Entered values are:\n");
    for(i=0;i<3;i++)
   {
//even though heap memory is freed from pointer ptr, contents stored in heap still exist
    printf("%d ",*(ptr1+i));
   }
   return 0;
}
```

```
"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:
1 2 3
Process returned 0 (0x0)   execution time : 2.397 s
Press any key to continue.
```

main.c [9_free function] - Code::Blocks 20.03

```c
#include <stdio.h>
#include <stdlib.h>
/** 9-free() function in C **/
int main()
{
    int i,*ptr;
    ptr=(int*)malloc(3*sizeof(int));
    printf("Enter the values:\n");
    for(i=0;i<3;i++)
    {
     scanf("%d",ptr+i);
    }
    int *ptr1;
    ptr1=ptr;
   free(ptr); //ptr pointer is freed pointing to memory, now it is dangling pointer
              //dangling pointer points to non existing memory location
  /** printf("\nEntered values are:\n");
    for(i=0;i<3;i++)
    {
     printf("%d ",*(ptr+i)); //undefined behavior or it gives garbage values
    } //since it is undefined behavior the program gets misbehaved from here **/
    printf("\nEntered values are:\n");
    for(i=0;i<3;i++)
    { //content stored in heap memory will exist even after freeing the pointer ptr
     printf("%d ",*(ptr1+i));
    }
free(ptr1);
return 0;
```

"D:\1. C C++NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 10_JENNYS LECTURE_DYNAMI

```
Enter the values:
1 2 3

Entered values are:
1 2 3
Process returned 0 (0x0)   execution time : 3.131 s
Press any key to continue.
```