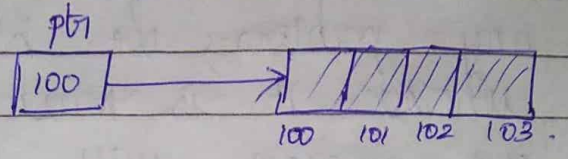
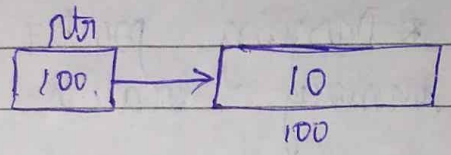
C_82 ⇒ Dangling Pointers in C

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

↳ we allocate memory dynamically from heap of datatype int to pointer ptr.

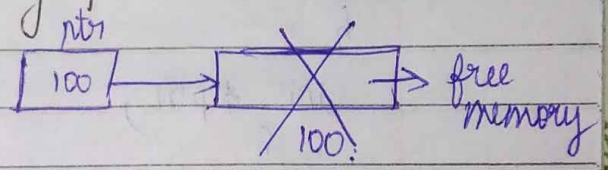


*ptr = 10;

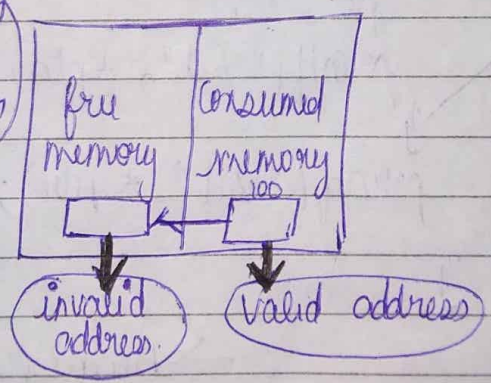


printf("Value at address of ptr: %d", *ptr);

free(ptr); // dangling pointer.



Here we free out the memory space pointing by ptr and acts as (dangling pointer or hanging pointer)

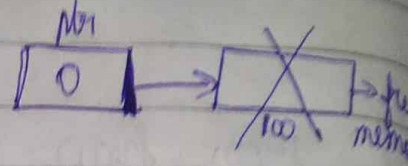


printf("%d", *ptr); ⇒ when we dereference this ptr; then we have any garbage value.

* So; its better don't leave any pointer as dangling or hanging.

* So, to overcome the error handling of this dangling pointer; we can initialize it with NULL.

```
ptr = NULL;
```



Notes:

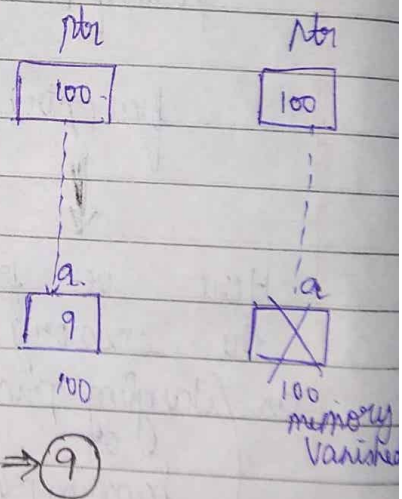
* So before dereferencing or accessing any pointers; we have to check whether it is NULL or not (orelse) our program will be crashed

* Dangling pointer points to a freed memory location.

Problem Example: 1

```
main()
{
  int *ptr;
  {
    int a = 9;
    ptr = &a;
    printf("%d", *ptr);
  }
  printf("%d", *ptr);
}
```

Internal block



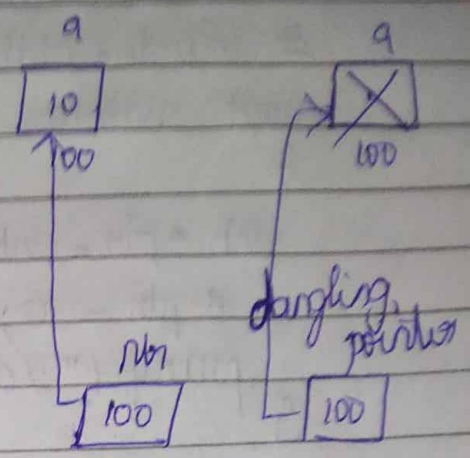
→ 9

→ Here ptr is now dangling pointer

→ Because, 'a' is internal block variable and it can't be accessed outside of the block.

Problem Example ②

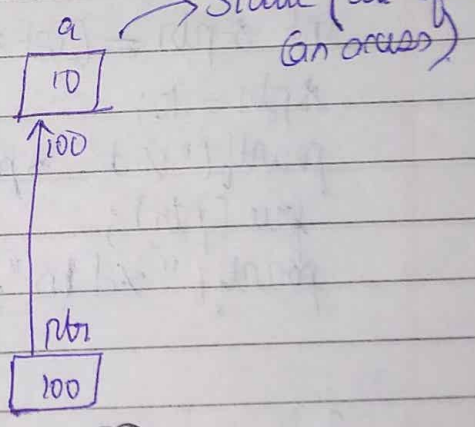
```
int * f()
{
  int a=10;
  return &a;
}
main()
{
  int *ptr = f();
  printf("%d", *ptr);
}
```



Dangling pointer

Problem Example ③ :

```
int *f()
{
static int a=10;
  return &a;
}
main()
{
  int *ptr = f();
  printf("%d", *ptr);
```



Static (all functions can access)

⇒ 10.

A since 'a' is static it is accessed by all functions in the program.

Note:

static variables have a scope throughout the program (ie) static variables are having scope globally.

Program 1

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int *ptr = (int*) malloc (sizeof(int));
    *ptr = 10;
  printf("%d", *ptr);  ⇒ 10
}
```

Program 2

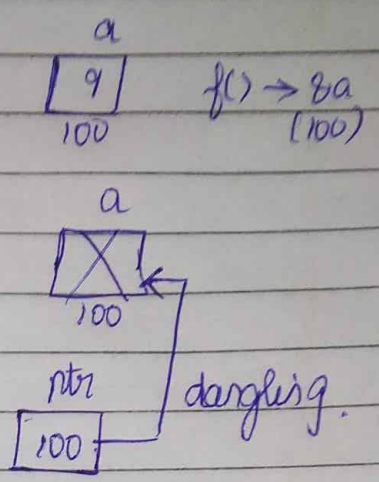
```
#include <stdlib.h>
#include <stdio.h>
void main()
{
  int *ptr = (int *) malloc(sizeof(int));
   *ptr = 10;
   printf("%d", *ptr);  ⇒ 10
    free(ptr);
    printf("%d\n", *ptr);  ⇒ Dangling pointer &
                             we get any garbage value
                             or program crashes.
}
```

Program 3

```
void main()
{
  int *ptr = (int *) malloc (sizeof(int));
  *ptr = 10;
  printf("%d", *ptr);  ⇒ 10
  free(ptr);
   ptr = NULL;
  printf("%d\n", ptr);  ⇒ 0 ptr is NULL we cannot
                          dereference
}
```

Program 4

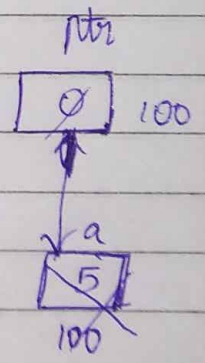
```
# include <stdio.h>
# include <stdlib.h>

int* f()
{
    int a=9;
    return &a;
}
int main()
{
    int *ptr = f();
    printf("%d", *ptr);
}
}
```



⇒ dangling; hence we get garbage value or program crashes.

Program 5

```
# include <stdio.h>
# include <stdlib.h>
void main()
{
    int *ptr = NULL;
    {
        int a=5;
        ptr = &a;
        printf("%d\n", *ptr);
    }
    printf("%d\n", *ptr);
}
```



⇒ (5)

⇒ garbage value but sometimes gives a value 5.

PROBLEM 1:

```c
#include <stdio.h>
#include <stdlib.h>
/** 1-DANGLING POINTER **/
int main()
{
    int *ptr;
    {
     int a=9;
     ptr=&a;
     printf("%d\n",*ptr);
    }
     printf("%d\n",*ptr);//This is dangling pointer
/** This sometimes gives garbage value or same 9 but later value will change**/
}
```

```
"D:\1. C NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 6_JENNYS LECTURE_P
9
9

Process returned 0 (0x0)   execution time : 0.037 s
Press any key to continue.
```

PROBLEM 2:

```c
#include <stdio.h>
#include <stdlib.h>
/** 2-DANGLING POINTER **/
int main()
{
 int *ptr=(int*)malloc(sizeof(int));
 *ptr=10;
 printf("%d\n",*ptr);
 free(ptr);
 printf("%d\n",*ptr);//dangling pointer
 /** this will give garbage value as output **/
getch();
}
```

```
"D:\1. C NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 6_JENNYS LECTURE_POIN
10
12740768
```

PROBLEM 3:

```c
#include <stdio.h>
#include <stdlib.h>
/** 3-DANGLING POINTER **/
int main()
{
 int *ptr=(int*)malloc(sizeof(int));
 *ptr=10;
 printf("%d\n",*ptr);
 free(ptr);
 ptr=NULL;
 printf("%d\n",ptr);//we cant deference Null pointer because it has only zero
 /** Null pointer doest point to any memory location and it has value 0 **/
getch();
}
```

"D:\1. C NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PART 6_JENNYS LECTURE_POINTERS\11_DAN

```
10
0
```

PROBLEM 4:

```c
#include <stdio.h>
#include <stdlib.h>
/** 4-DANGLING POINTER **/
int *f()
{
 int a=9;
 return &a;
 /** scope of variable 'a' is within this block only**/
}
int main()
{
int *ptr=f();
printf("%d\n",*ptr);//dangling pointer
/** memory allocated to variable 'a' vanishes **/
/** so we cant dereference this pointer ptr **/
getch();
}
```

PROBLEM 5:

```c
#include <stdio.h>
#include <stdlib.h>
/** 5-DANGLING POINTER **/
int *f()
{
 static int a=9;
 return &a;
 /** variable 'a' is static hence memory do not vanishes outside the block**/
}
int main()
{
int *ptr=f();
printf("%d\n",*ptr);
/** memory allocated to variable 'a' remains still, since it is static  **/
/** Hence we can access 'a' for entire program **/
getch();
}
```

"D:\1. C NOTEBOOK\C LANGUAGE\C PROGRAMS\PAR

```
9
```

PROBLEM 6:

```c
#include <stdio.h>
#include <stdlib.h>
/** 6-DANGLING POINTER **/
int main()
{
int *ptr=NULL;
{
 int a=9;
 ptr=&a;
 printf("%d\n",*ptr);
}
//dangling pointer
printf("%d\n",ptr); //gives unknown memory address
printf("%d\n",*ptr);//gives garbage value
getch();
}
```

"D:\1. C NOTEBOOK\C LANGUAGE\C PROGRAMS\PART 5_Jennys Lectures\PAR

```
9
6422036
9
```