Week 12 Lecture 1

Memory Allocation

Allocating Memory in C

Requesting Memory

- To get memory, use malloc()
- Two ways to create a string
  - Request at compile time:
    - char x[80];
  - Request at run time
    - char *s;
    - s = malloc(80 * sizeof(char));

malloc

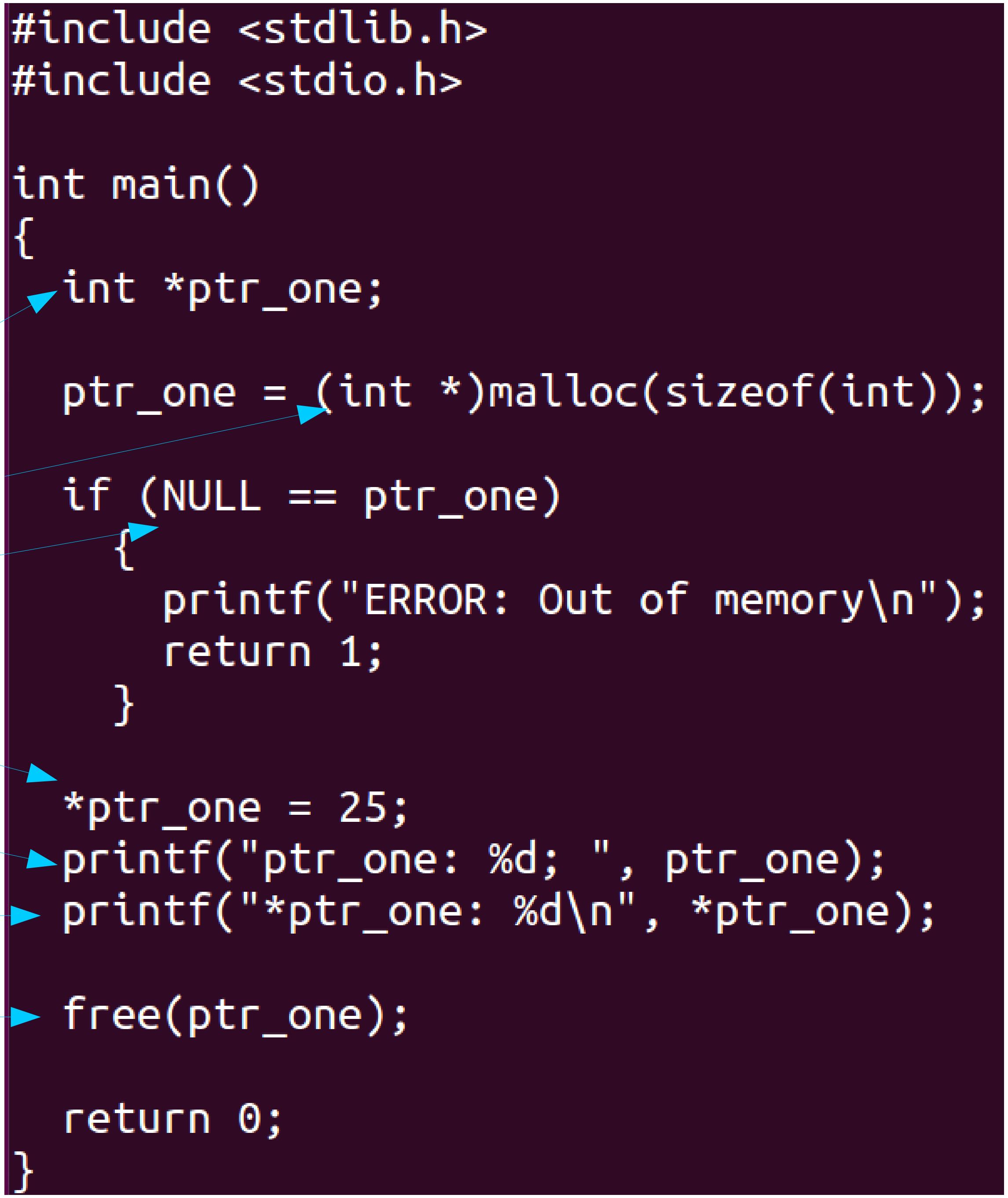
- Defined in <stdlib.h>
- Declaration
  - void *malloc(size_t size);
    - A pointer to anything: void *
    - An unsigned integer: size_t
  - The type size_t is used so that different sized numbers can be used for different sized memories.

free

- To release memory after you are done with it use free()
- Failure to free allocated memory causes a *memory leak*.
  - The computer can allocate all of its memory to variables that are no longer used.

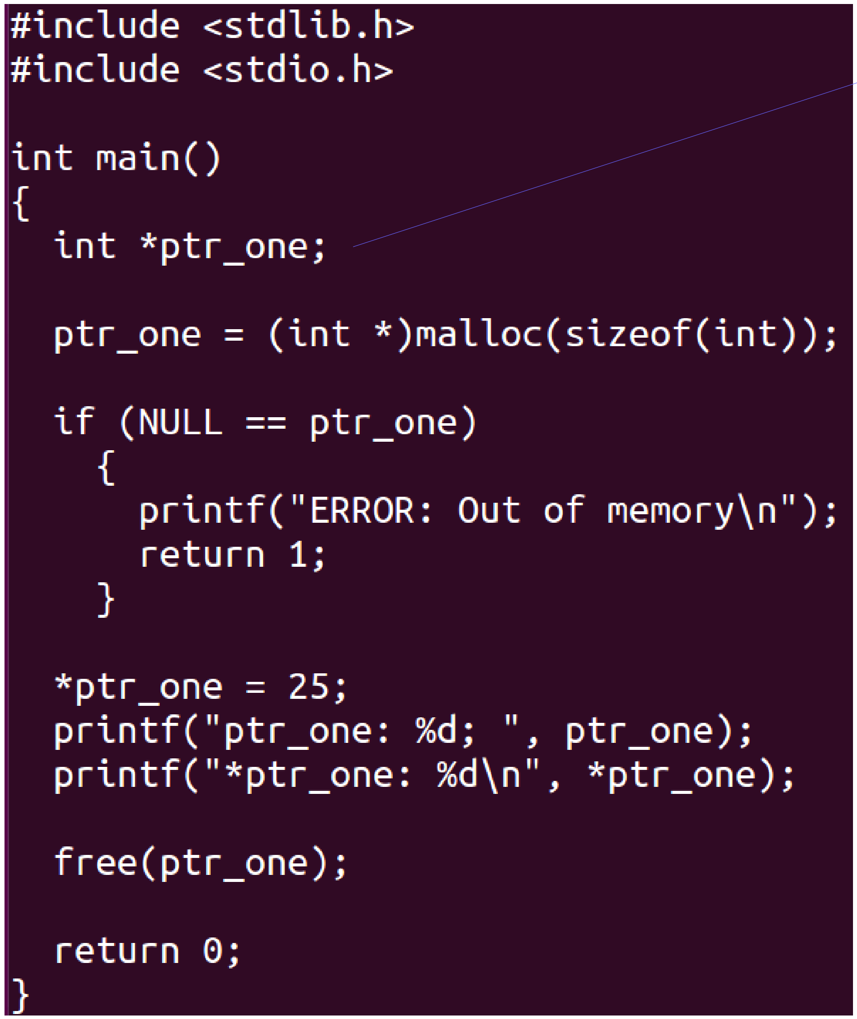
Example

- Pointer created
- Memory allocated
- Test allocation
- Set value
- Print address
- Print value
- Free memory

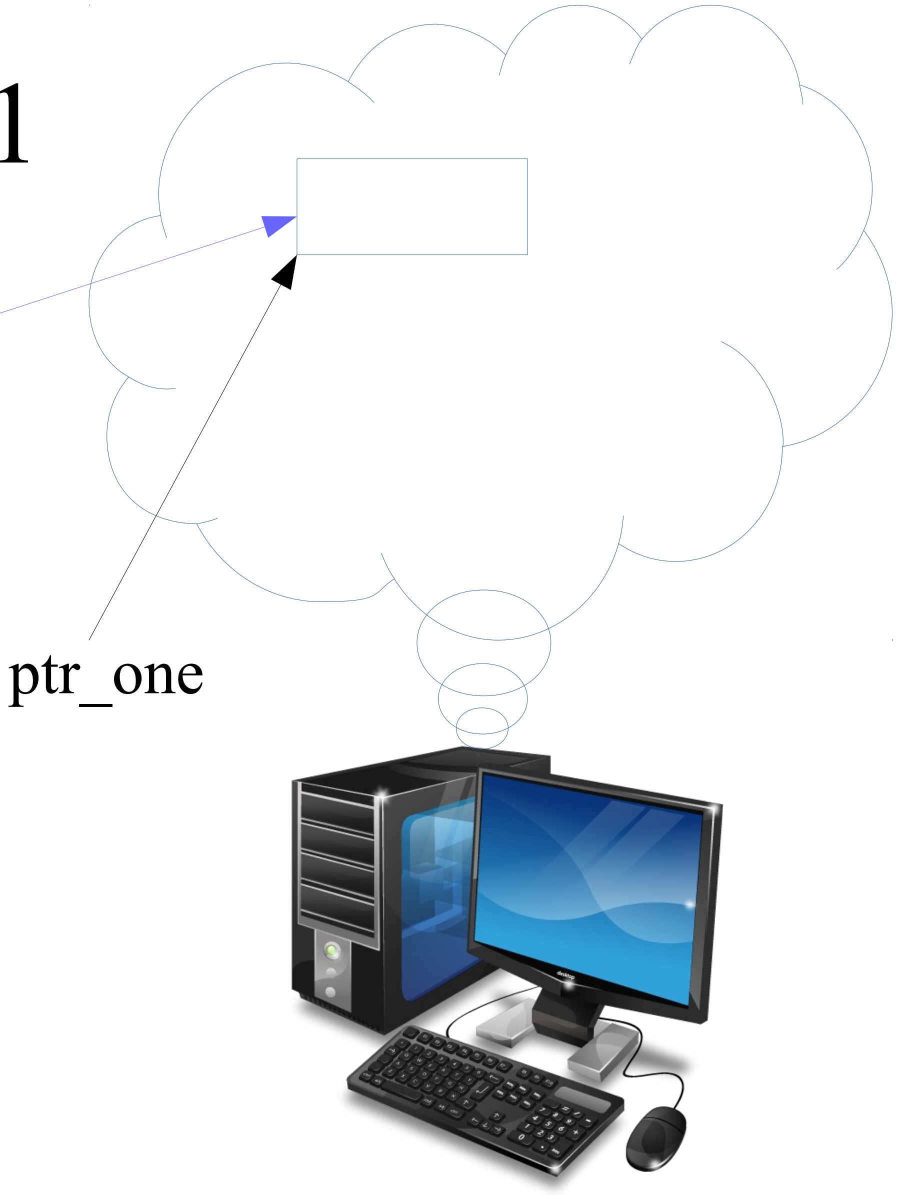


```
#include <stdlib.h>
#include <stdio.h>

int main()
{
  int *ptr_one;

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

  if (NULL == ptr_one)
    {
      printf("ERROR: Out of memory\n");
      return 1;
    }

  *ptr_one = 25;
  printf("ptr_one: %d; ", ptr_one);
  printf("*ptr_one: %d\n", *ptr_one);

  free(ptr_one);

  return 0;
}
```

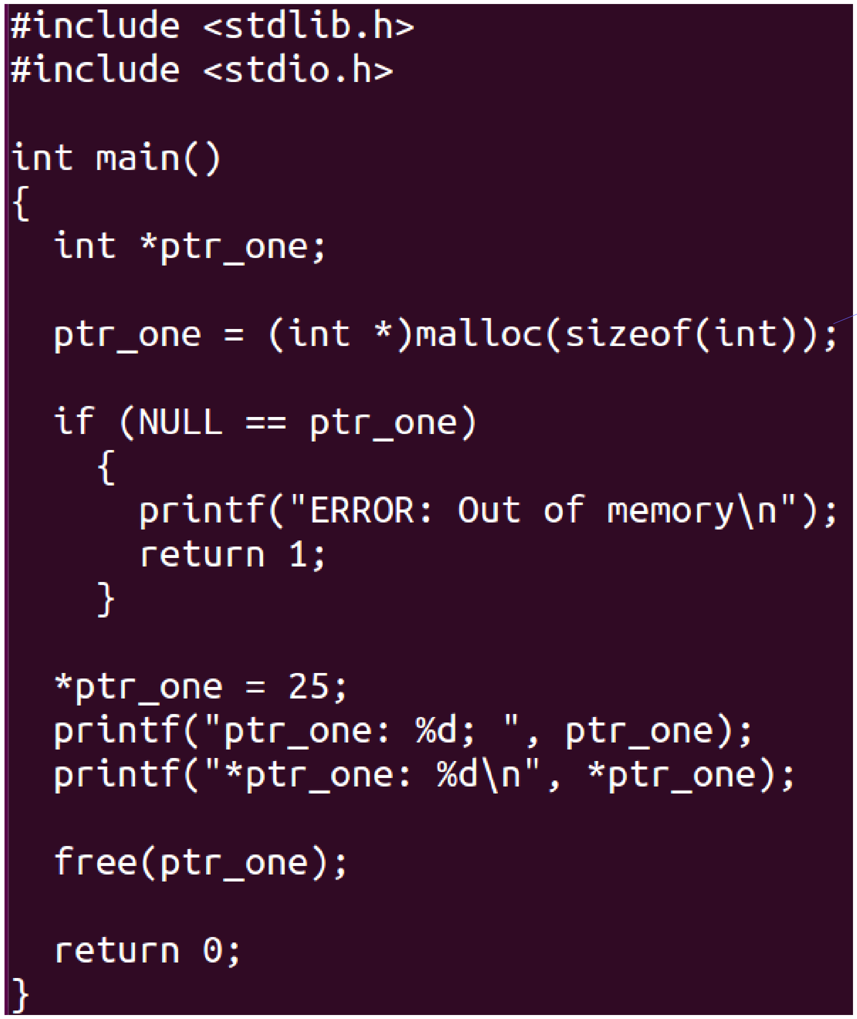
Example Effects 1



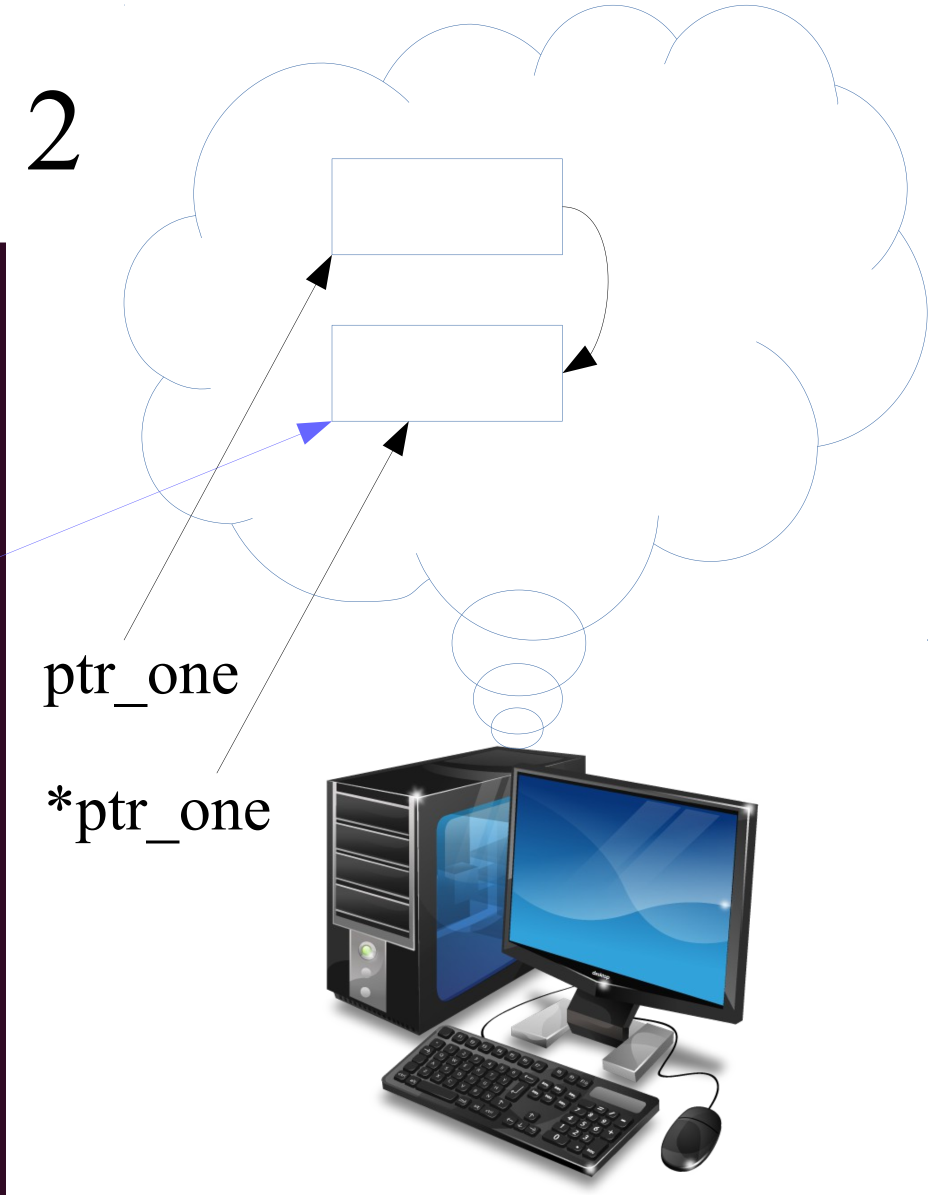
```
#include <stdlib.h>
#include <stdio.h>

int main()
{
  int *ptr_one;

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

  if (NULL == ptr_one)
    {
      printf("ERROR: Out of memory\n");
      return 1;
    }

  *ptr_one = 25;
  printf("ptr_one: %d; ", ptr_one);
  printf("*ptr_one: %d\n", *ptr_one);

  free(ptr_one);

  return 0;
}
```



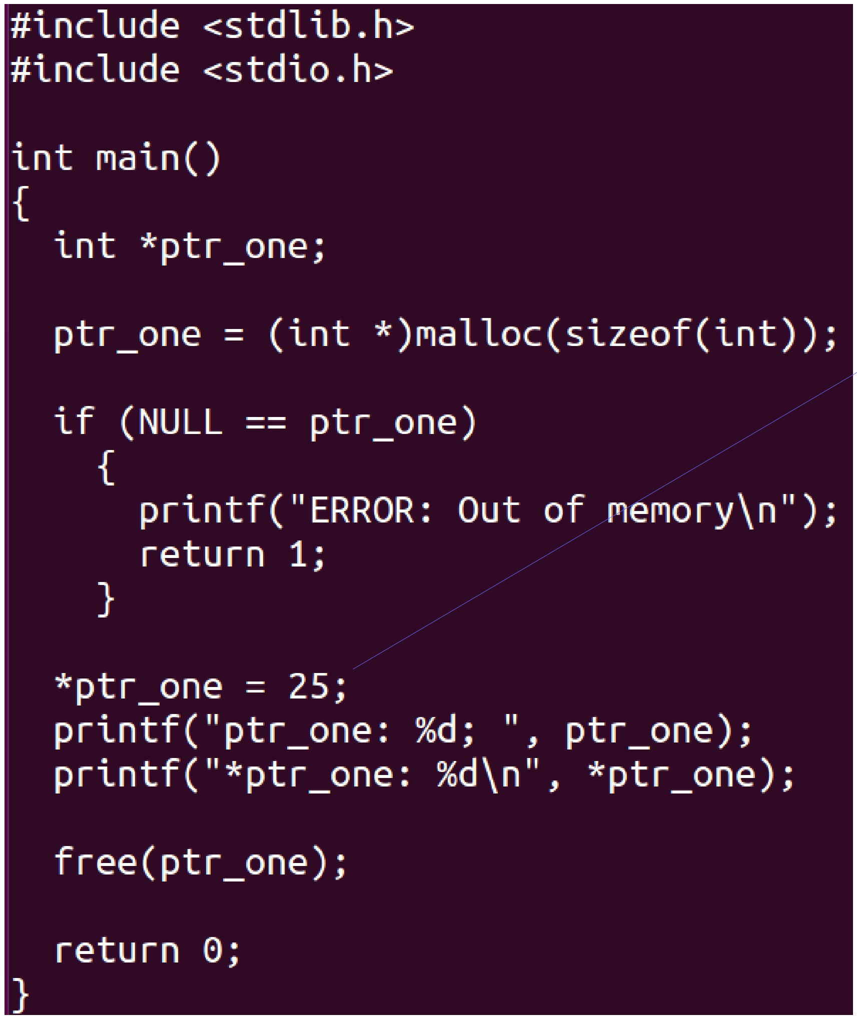
Example Effects 2



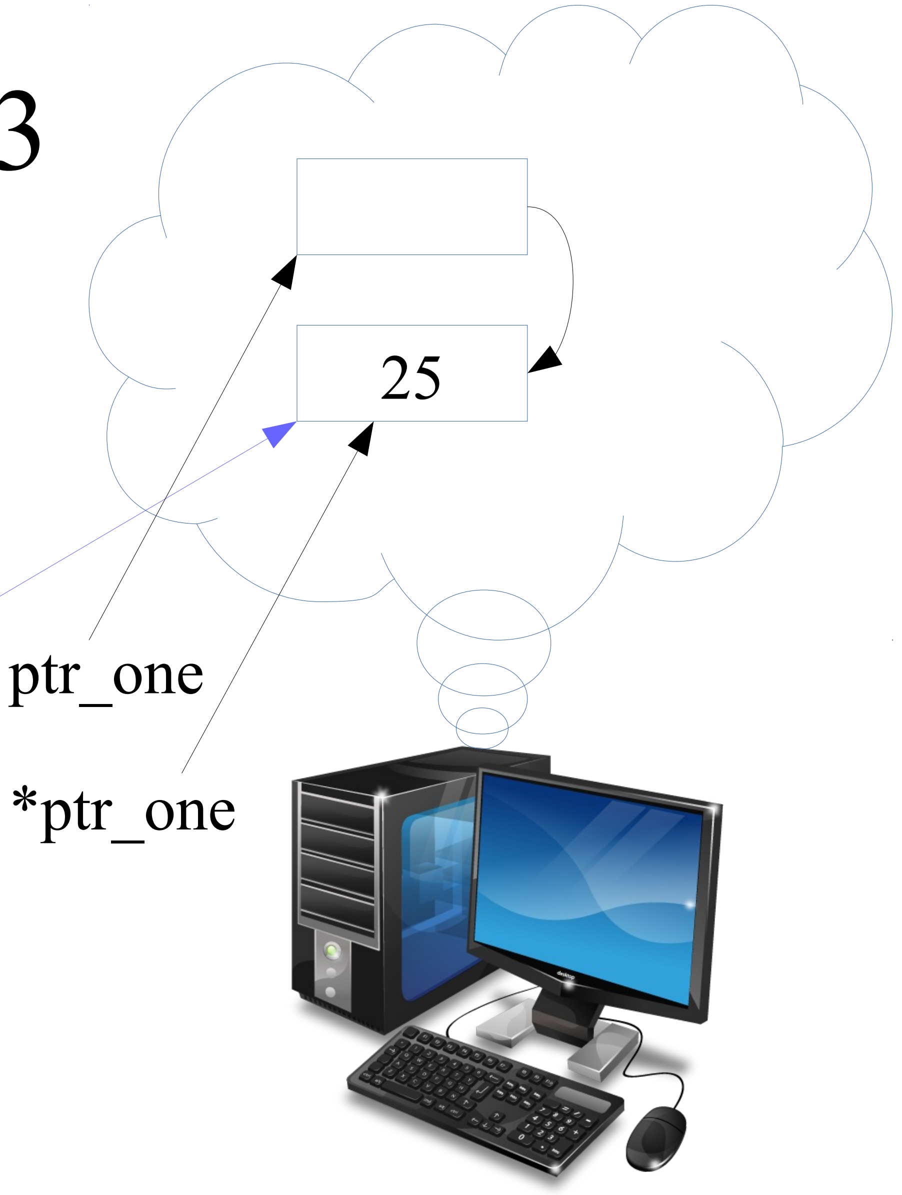
```
#include <stdlib.h>
#include <stdio.h>

int main()
{
  int *ptr_one;

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

  if (NULL == ptr_one)
    {
      printf("ERROR: Out of memory\n");
      return 1;
    }

  *ptr_one = 25;
  printf("ptr_one: %d; ", ptr_one);
  printf("*ptr_one: %d\n", *ptr_one);

  free(ptr_one);

  return 0;
}
```



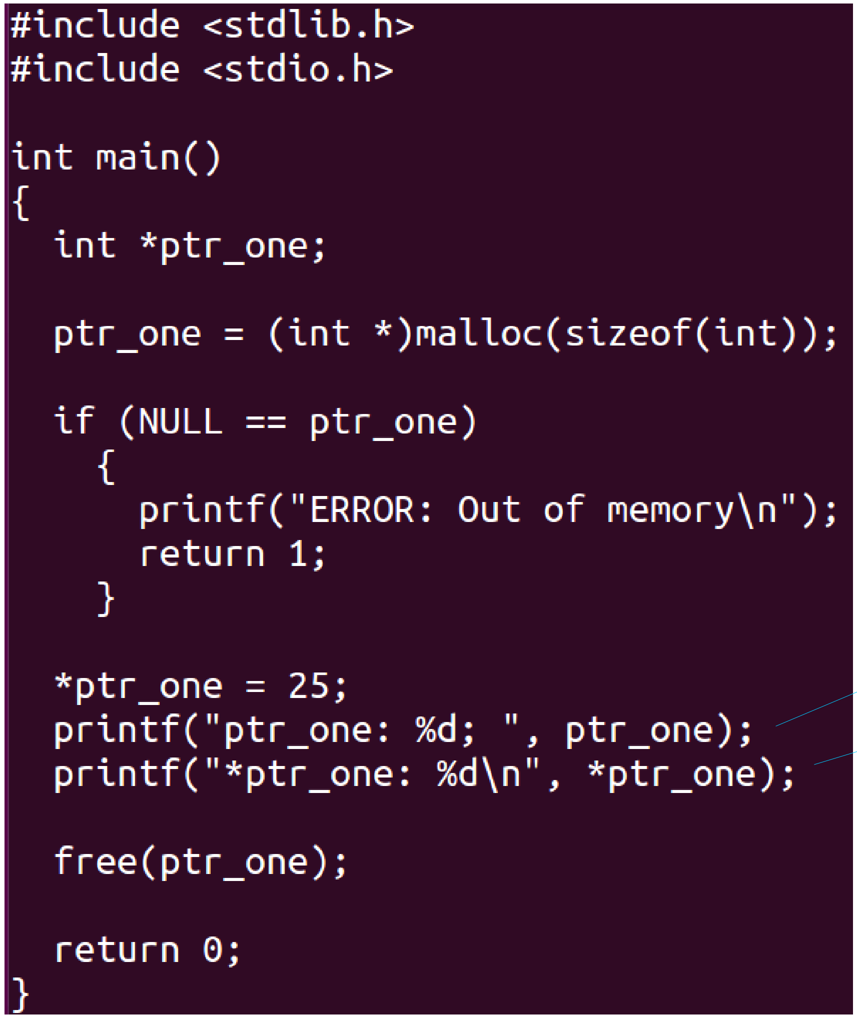
Example Effects 3



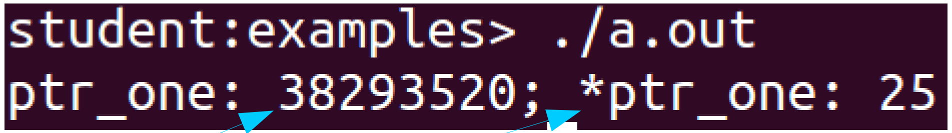
```
#include <stdlib.h>
#include <stdio.h>

int main()
{
  int *ptr_one;

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

  if (NULL == ptr_one)
    {
      printf("ERROR: Out of memory\n");
      return 1;
    }

  *ptr_one = 25;
  printf("ptr_one: %d; ", ptr_one);
  printf("*ptr_one: %d\n", *ptr_one);

  free(ptr_one);

  return 0;
}
```



Example Output

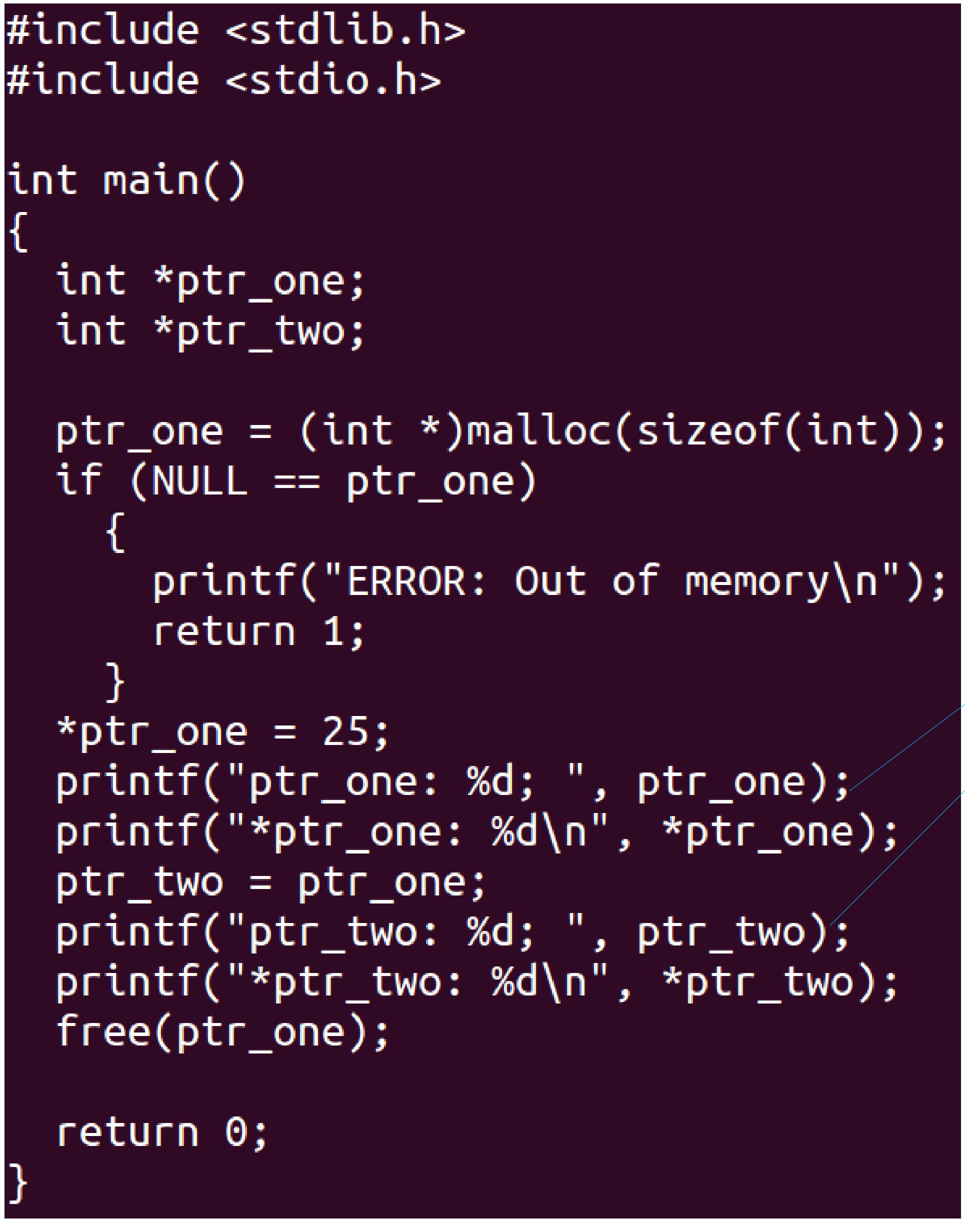


```c
#include <stdlib.h>
#include <stdio.h>

int main()
{
  int *ptr_one;

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

  if (NULL == ptr_one)
    {
      printf("ERROR: Out of memory\n");
      return 1;
    }

  *ptr_one = 25;
  printf("ptr_one: %d; ", ptr_one);
  printf("*ptr_one: %d\n", *ptr_one);

  free(ptr_one);

  return 0;
}
```

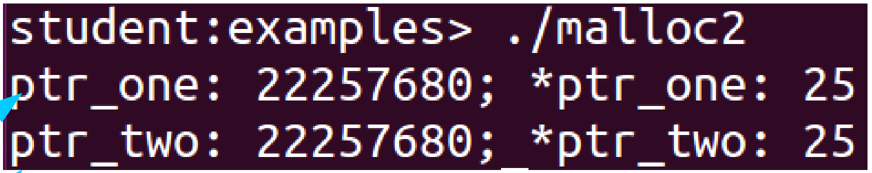


```
student:examples> ./a.out
ptr_one: 38293520; *ptr_one: 25
```

Another Example



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

int main()
{
  int *ptr_one;
  int *ptr_two;

  ptr_one = (int *)malloc(sizeof(int));
  if (NULL == ptr_one)
    {
      printf("ERROR: Out of memory\n");
      return 1;
    }
  *ptr_one = 25;
  printf("ptr_one: %d; ", ptr_one);
  printf("*ptr_one: %d\n", *ptr_one);
  ptr_two = ptr_one;
  printf("ptr_two: %d; ", ptr_two);
  printf("*ptr_two: %d\n", *ptr_two);
  free(ptr_one);

  return 0;
}
```

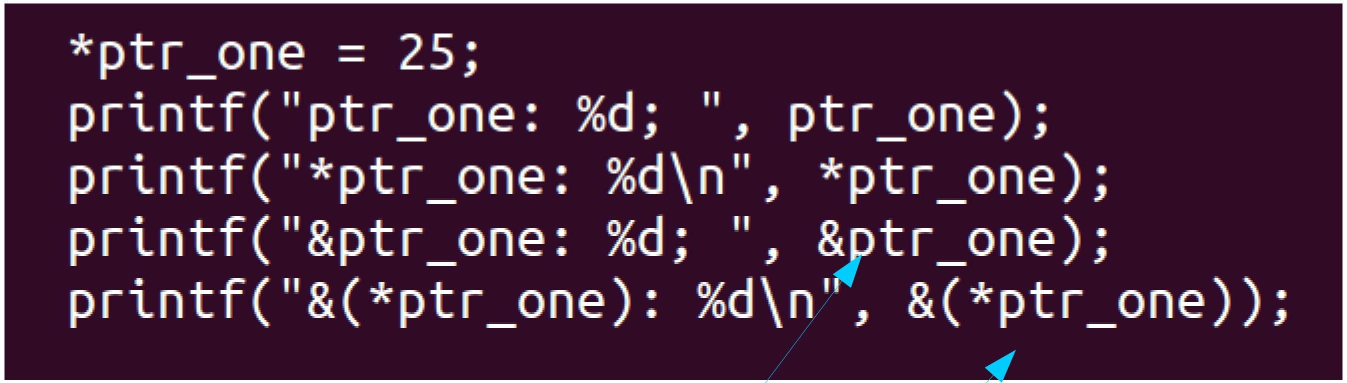


```
student:examples> ./malloc2
ptr_one: 22257680; *ptr_one: 25
ptr_two: 22257680; *ptr_two: 25
```

Using & operator

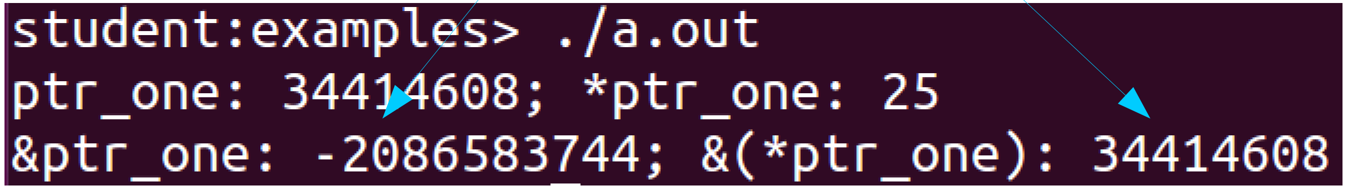
- The & operator gets the address of a variable

& Example



```
*ptr_one = 25;
printf("ptr_one: %d; ", ptr_one);
printf("*ptr_one: %d\n", *ptr_one);
printf("&ptr_one: %d; ", &ptr_one);
printf("&(*ptr_one): %d\n", &(*ptr_one));
```

- Address of the pointer variable
- Address of the variable pointed to



```
student:examples> ./a.out
ptr_one: 34414608; *ptr_one: 25
&ptr_one: -2086583744; &(*ptr_one): 34414608
```