ECE 220: Computer Systems & Programming

Lecture 18: Dynamic Memory Allocation

Thomas Moon

March 19, 2024

```
typedef struct flightType{
        …
}Flight;

int main()
{
    Flight planes[100];
```

What if we need > 100 planes?
→ Increase the size of array.

But, sometimes we only need 2-3 planes
→ Wasting the rest of unused memory.

Dynamic Memory Allocation (this lecture)

Planes take off & land,
i.e. a new data can come and go

Adding or removing an item in the middle.
→ Not efficient in array

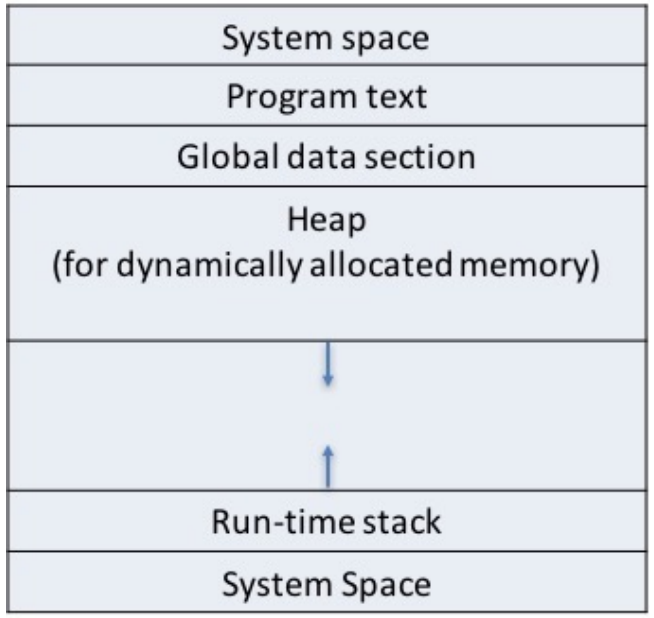
Linked List (next lecture)

Dynamic Allocation

- Ideally, we want to *allocate as much memory as needed* rather than a pre-set amount.
- *Memory allocation manager* manages an area of memory called *heap*.
- During the execution, a program makes a request to the memory allocator for a contiguous piece of memory of a particular size.
- The allocator reserves the memory and returns a pointer to it.

```
malloc
calloc          free
realloc
```

Automatic vs. Dynamic Memory Allocation



	Automatic	Dynamic
Mechanism of allocation	automatic	use malloc() family
Lifetime of memory	programmer has no control - it ends when exit function/block	programmer has control - use free() to deallocate
Location of memory	stack or global data area	heap
Size of memory	fixed	adjustable

(optional) VLA vs Dynamic Array

(variable-length array)

Example code of VLA

```
int main(){
    int n;
    printf("Enter the size: ");
    scanf("%d", &n);

    Flight planes[n];
```

In both VLA and dynamic arrays,
the array size is determined during the run-time.

VLA
fixed
run-time stack

Dynamic array
can grow or shrink
heap

`malloc`

- `malloc` function can be used to allocate some number of bytes of memory in the heap.

```
void *malloc(size_t NumBytes)
```

- The size of allocated memory block (in byte) is indicated by the argument.
- It returns a generic pointer (of type void*) to the memory, or NULL in case of failure.
- The allocated memory is not initialized (there could be left of junk data).
- It is defined in `stdlib.h`

Using `malloc`

- To use malloc, we need to know how many bytes to allocate. The `sizeof` operator asks the compiler to calculate the size of a particular type.
- (optional) We also need to change the type of the return value to the proper kind of pointer- this is called "casting".

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

Flight *ptr = (Flight *) malloc(numFlight*sizeof(Flight));
```

Automatic vs Dynamic

```
int x;
int *ptr;
ptr = &x;
```

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

Example: Student Record Management

```
int main(){
    student *s;

    // allocate 2 students
    s = ____________________________________________;

    //add Bob
    ____________________;
    ____________________;
    ____________________;

    //add Alice
    ____________________;
    ____________________;
    ____________________;
```

```
int main(){
    student s[100];

    s[0].UIN = 1;
    strcpy(s[0].netid, "Bob");
    s[0].GPA = 3.0;

    s[1].UIN = 2;
    strcpy(s[1].netid, "Alice");
    s[1].GPA = 3.5;
```

Valgrind - Check Memory

```
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    double GPA;
}student;
```

sizeof(student) → 112

- Use “valgrind” to check memory leakage

```
valgrind ./a.out
```

```
==150247== HEAP SUMMARY:
==150247==     in use at exit: 224 bytes in 1 blocks
==150247==   total heap usage: 1 allocs, 0 frees, 224 bytes allocated
==150247==
==150247== LEAK SUMMARY:
==150247==    definitely lost: 224 bytes in 1 blocks
==150247==    indirectly lost: 0 bytes in 0 blocks
==150247==      possibly lost: 0 bytes in 0 blocks
==150247==    still reachable: 0 bytes in 0 blocks
==150247==         suppressed: 0 bytes in 0 blocks
```

`malloc` does not de-allocate on its own!

Programmer should manually `free` them to **avoid memory leakage**.

side notes

- I/O stream implementation will also allocate heap memory.
- How many heap alloc and free?

```
int *iptr = (int*) malloc(4*sizeof(int));
char *cptr = (char*) malloc(sizeof(char));

printf("Hello\n");
scanf("%d");

free(iptr);
free(cptr);
```

```
==225141==   total heap usage: 4 allocs, 4 frees, 2,065 bytes allocated
==225141==
==225141== All heap blocks were freed -- no leaks are possible
```

`free`

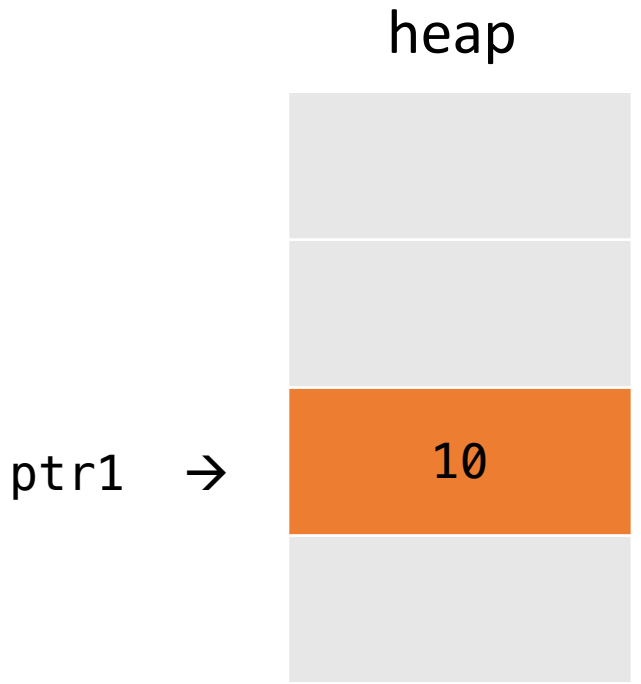
- `free` function frees the memory space pointed by ptr.

```
void free(void *ptr)
```

- ptr should point any Heap memeory location.

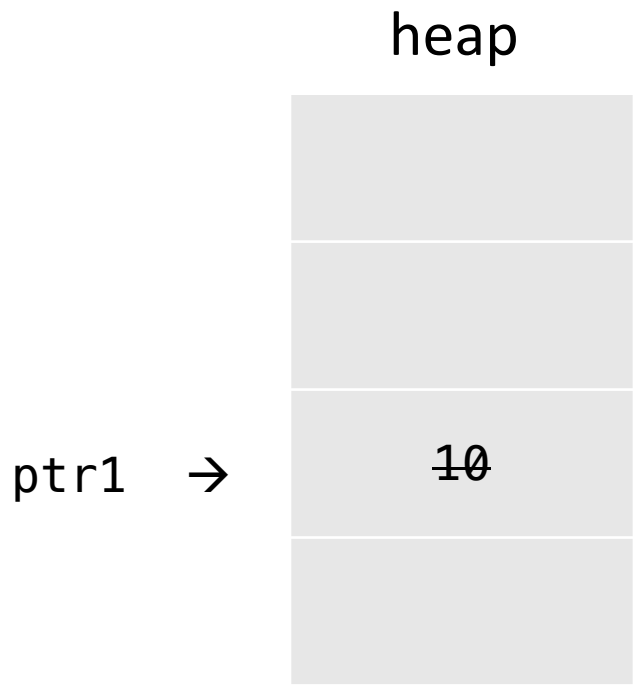
Example of `malloc` and `free`

```c
int main()
{
    int *ptr1 = (int *) malloc(sizeof(int));
    if(ptr1 == NULL){
        printf("Error - malloc failure\n");
        return -1;
    }

    *ptr1 = 10;

    free(ptr1);
```



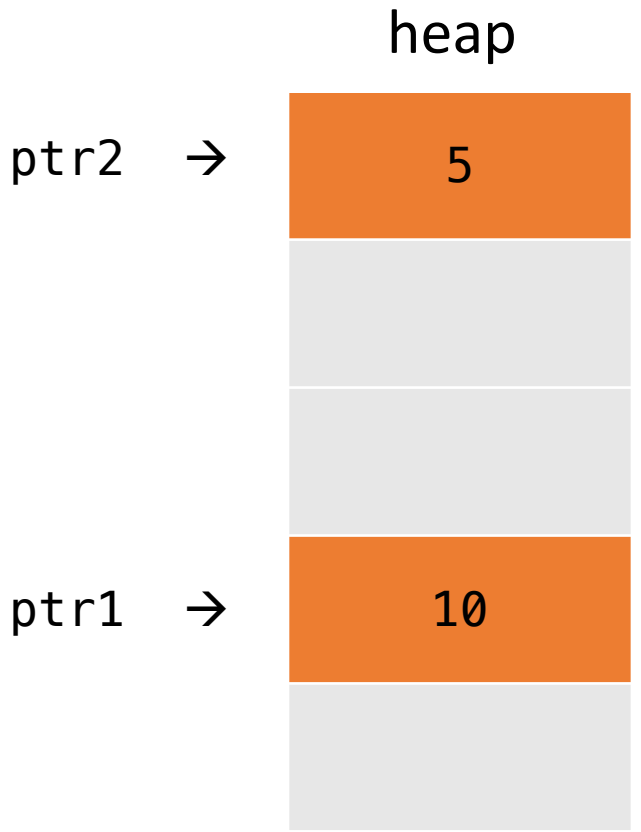
Example of `malloc` and `free`

```c
int main()
{
    int *ptr1 = (int *) malloc(sizeof(int));
    if(ptr1 == NULL){
        printf("Error - malloc failure\n");
        return -1;
    }

    *ptr1 = 10;

    free(ptr1);
```



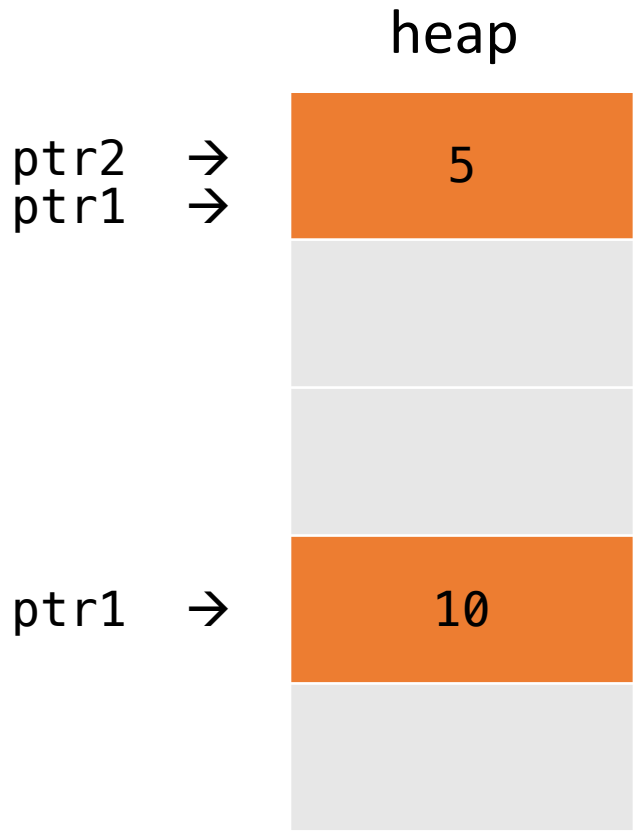
Example of `malloc` and `free`

```c
int main()
{
    int *ptr1 = (int *) malloc(sizeof(int));
    if(ptr1 == NULL){
        printf("Error - malloc failure\n");
        return -1;
    }
    *ptr1 = 10;

    int *ptr2 = (int *) malloc(sizeof(int));
    *ptr2 = 5;
```



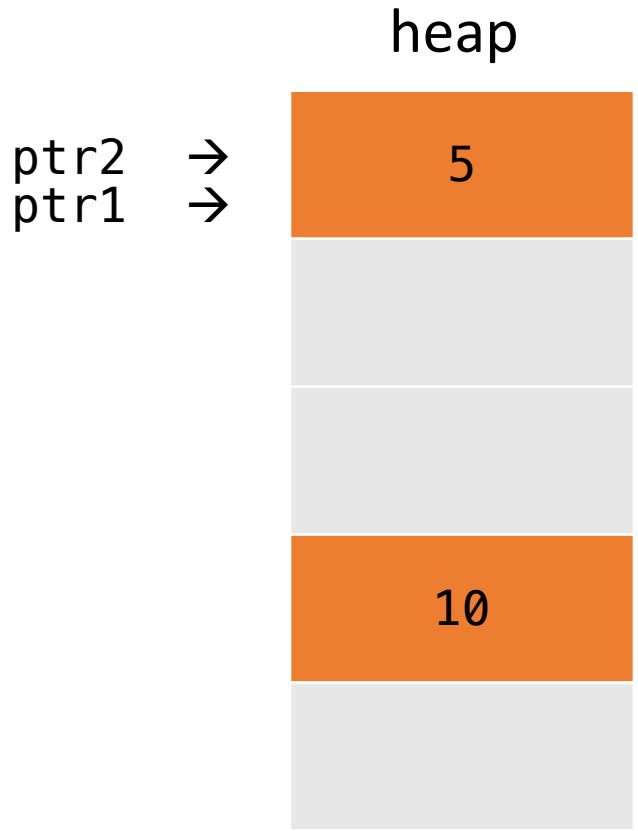
Example of `malloc` and `free`

```c
int main()
{
    int *ptr1 = (int *) malloc(sizeof(int));
    if(ptr1 == NULL){
        printf("Error - malloc failure\n");
        return -1;
    }
    *ptr1 = 10;

    int *ptr2 = (int *) malloc(sizeof(int));
    *ptr2 = 5;

    // What will happen?
    ptr1 = ptr2;
```



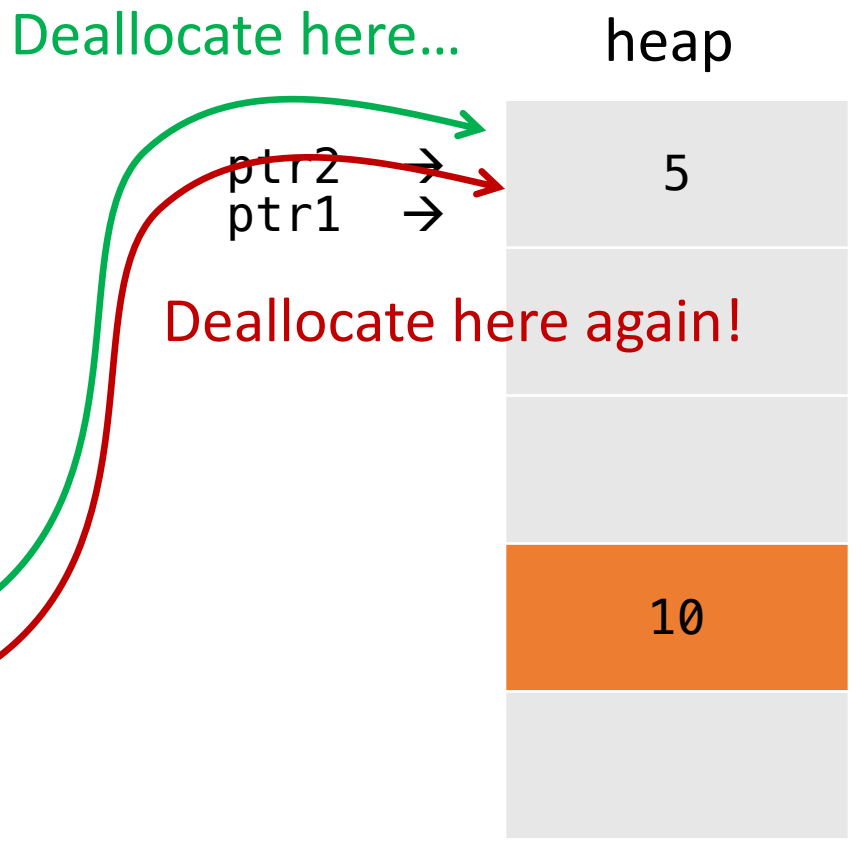
Example of `malloc` and `free`

```c
int main()
{
    int *ptr1 = (int *) malloc(sizeof(int));
    if(ptr1 == NULL){
        printf("Error - malloc failure\n");
        return -1;
    }
    *ptr1 = 10;

    int *ptr2 = (int *) malloc(sizeof(int));
    *ptr2 = 5;

    // What will happen?
    ptr1 = ptr2;

    free(ptr2);
    free(ptr1);  <- Run-time error, why?
```



Example of `malloc` and `free`

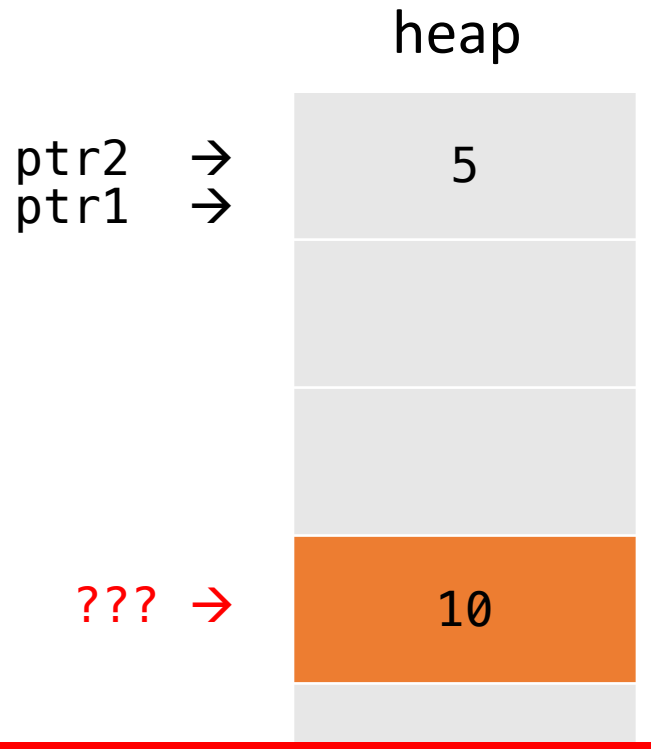
```
int main()
{
    int *ptr1 = (int *) malloc(sizeof(int));
    if(ptr1 == NULL){
        printf("Error – malloc failure\n");
        return -1;
    }
    *ptr1 = 10;

    int *ptr2 = (int *) malloc(sizeof(int));
    *ptr2 = 5;

    // What will happen?
    ptr1 = ptr2;

    free(ptr2);
    free(ptr1);   <- Run-time erro
```



Double-free (free on already free memory)
→ Compiler does NOT check for you

Example of `malloc` and `free`

```c
int main()
{
    int *ptr1 = (int *) malloc(sizeof(int));
    if(ptr1 == NULL){
        printf("Error – malloc failure\n");
        return -1;
    }
    *ptr1 = 10;

    int *ptr2 = (int *) malloc(sizeof(int));
    *ptr2 = 5;

    // What will happen?
    ptr1 = ptr2;

    free(ptr2);
    // free(ptr1);
```

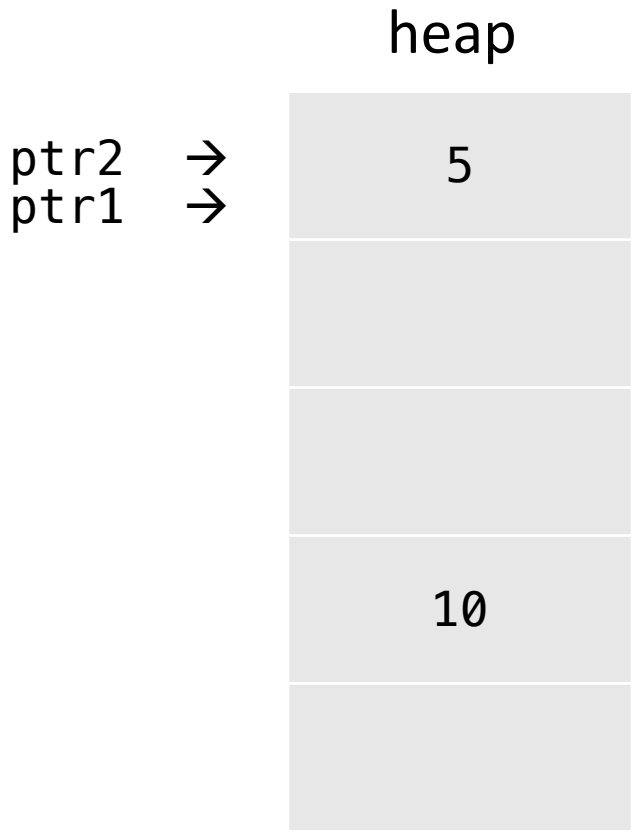


Memory leakage

→ Fail to free all the dynamic-allocated memory

Example of `malloc` and `free`

```c
int main()
{
    int *ptr1 = (int *) malloc(sizeof(int));
    if(ptr1 == NULL){
        printf("Error – malloc failure\n");
        return -1;
    }
    *ptr1 = 10;

    int *ptr2 = (int *) malloc(sizeof(int));
    *ptr2 = 5;

    free(ptr1); // Free before change

    ptr1 = ptr2;

    free(ptr2); // or free(ptr1);
```



calloc

```
void *calloc(size_t n_items, size_t item_size)
```

- `n_items`: the number of items to be allocated
- `item_size`: the size of each item
- Total size of allocated memory = `n_items * item_size`
- Identical to malloc, except calloc initializes allocated memory to zero.

```
int *ptr_malloc = (int*) malloc(4*sizeof(int));
int *ptr_calloc = (int*) calloc(4,sizeof(int));
```

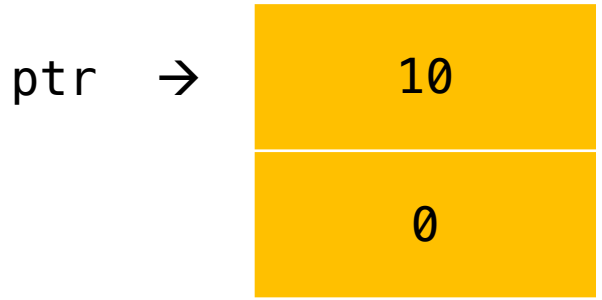
realloc

```
void *realloc(void *ptr, size_t size)
```

- Re-allocate memory block to a different size
  (change the size of memory block pointed by ptr)
- Returns a pointer to the newly allocated memory block
  - the location may be changed
  - if changed, the old memory block will be automatically de-allocated
- The content of the memory block is preserved, even if the block is moved to a new location (if the new size is larger than the old size, the added memory will not be initialized).
  - If ptr is NULL, it is same as malloc.
  - If size is 0 and ptr is not NULL, it is same as free.
  - ptr must have been returned by the malloc family (except ptr is NULL).

Example of `calloc` and `realloc`

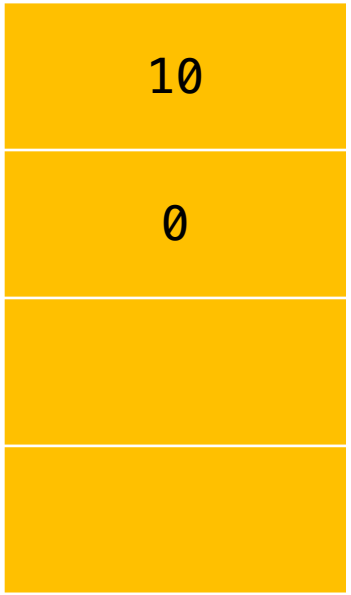
```c
int *ptr;
int *ptr_new;

ptr = (int *) calloc(2, sizeof(int));
*ptr = 10;

ptr_new = (int *) realloc(ptr, 4*sizeof(int));
```



Example of `calloc` and `realloc`

```
int *ptr;
int *ptr_new;

ptr = (int *) calloc(2, sizeof(int));
*ptr = 10;

ptr_new = (int *) realloc(ptr, 4*sizeof(int));
```

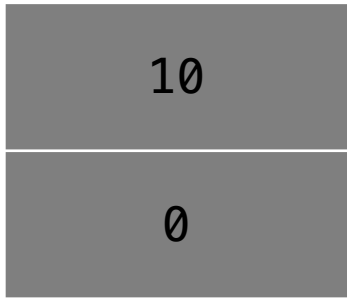
case 1

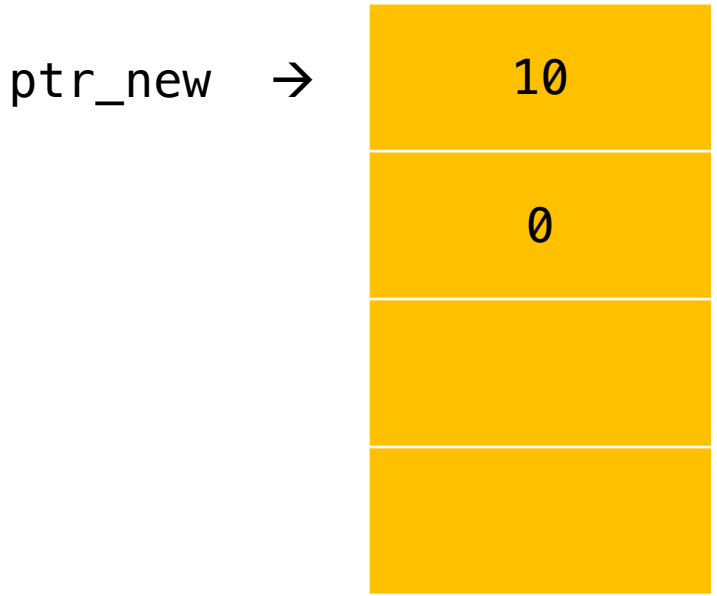


Example of `calloc` and `realloc`

```
int *ptr;
int *ptr_new;

ptr = (int *) calloc(2, sizeof(int));
*ptr = 10;

ptr_new = (int *) realloc(ptr, 4*sizeof(int));
```

case 2





Either case 1 or case 2, we only need one free for ptr_new.

```
free(ptr_new);
```

Example: Student Record Management

Add students from keyboard (unknown number of students)

```
(1. Add a new student, 0. Exit) : 1
Enter netid & GPA: aaa 3.0
(1. Add a new student, 0. Exit) : 1
Enter netid & GPA: bbb 3.0
(1. Add a new student, 0. Exit) : 1
Enter netid & GPA: ccc 4.0
(1. Add a new student, 0. Exit) : 0
```

Output

```
UIN netid GPA
1 aaa 3.000000
2 bbb 3.000000
3 ccc 4.000000
```

Use **dynamic allocation** to store the student record.

```
student *s;
int opt, cnt = 0;
do{
    printf("(1. Add a new student, 0. Exit) : ");
    scanf("%d", &opt);
    if(opt==1){
        cnt++;
        _____________________________________________
        s[cnt-1].UIN = cnt;
        printf("Enter netid & GPA: ");
        scanf("%s %lf", _______, _________);
    }
}while(opt);
printStudent(s, cnt);
_____________
```