is it clear what's going on : ~~III~~ II

2510 : Lecture 19 || March 06/17

▷ typedef continued

We can use typedef with the structure definition ex.

```
typedef struct Name {        you don't need this
    char first [20];
    char last [20];
} Name;
```

same as

```
typedef struct {
    char first [20];
    char last [20];
} Name;
```

OK ✓ Name n;
OK ✓ struct Name n2;

- we can define a structure types and create variables of that type all at once. ex.

```
struct Name {
    char first [20];
    char last [20];
} n, a [10], *p;
```

n: a struct Name
a: array of struct Name
p: pointer to struct Name

▷ Lifetime of objects

- the lifetime of a global (external) variable is the duration of the program

- lifetime of a local variable is the duration of the block in which it is created.

... Lecture 19

```
int n = 1; /* external variable : lifetime of duration
int main (void){ of the program */
  ...
  if (...){
    int x = 2;
    ...
  } /* x is destroyed here */
  ...
}
```

- dynamic memory allows us to have finer control of lifetimes

▷ 4 functions to handle DYNAMIC MEMORY #include <stdlib.h>

1) malloc] for allocating dynamic memory
2) calloc
3) realloc typically used to resize dynamic memory
4) free for deallocating dynamic memory

will put 0s in memory

ex. allocate dynamic memory to store 100 ints

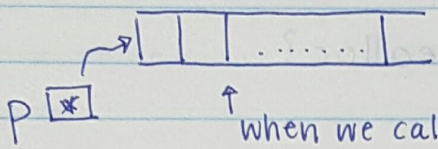
```
#include <stdlib.h>
int *p; size_t i;
p = malloc (100 * sizeof (int));
if (p == 0){   null pointer
  fprintf (stderr, "unable to allocate memory");
  exit (1); /* returns # upon exit & terminates program */
}
for (i = 0; i < 100; i++)
    p [i] = 3 * i; /* using dynamic memory */
free (p); /* deallocate memory after we are done */
```

using pointer like an array

There is a special pointer value called the NULL POINTER. It can be denoted as NULL or simply as ∅.

NULL is a macro, which is defined as ∅.

Note: Never dereference the null pointer! Program will crash



when we call free(p), this block of memory is deallocated. But the address is unchanged!

← continued from above code

- memory allocated by malloc is random values
- calloc: this allocates memory and zeros them out.

```
int *p;
p = calloc (100, sizeof(int));
if (p == 0){
   ...
}
```

(100 = # of elements; sizeof(int) = size of each element)

- realloc

```
int *temp;
int *p;
p = malloc (100 * sizeof (int));
if(p == 0) {
   ...
   exit(1);
}
temp = realloc (p, 200 * sizeof (int));
if (temp == 0) {
   fprintf (stderr, "reallocation failed ");
```

/* We want to resize it to 200 int */

(p = pointer to original block of memory; 200 * sizeof(int) = new size)

cont...

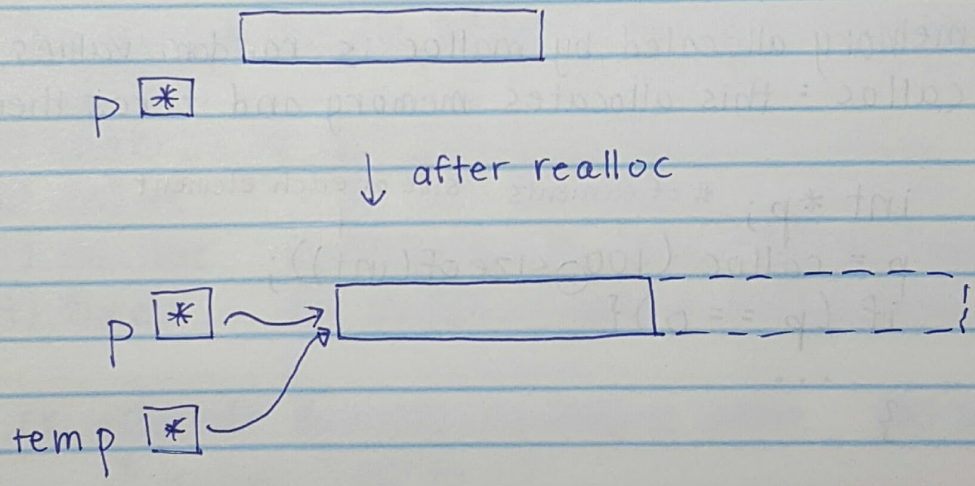
cont.

```
/* We can continue to use p (original block of 100 ints) */
} else {
    p = temp;
    /* use p which now points to a block of 200 ints */
}
```

What happens when we call realloc?

3 cases:

1) realloc fails
2) the system can find sufficient memory next to the original block



3) the system cannot find sufficient memory next to the original block