Dynamic Memory Allocation

Intro

- In c so far we have looked at static memory allocation.
- Memory can be allocated inside a stack frame but deletion is difficult.


```c
#include<stdio.h>
int main()
{
  int a[5]; // allocates an array of 5 integers.
  int b[2]; // allocates an array of 2 integers.

  a = b; // attempts to resize a by pointing it to the smaller array.
}
```

```
(base) nawiebe@DESKTOP-ITUK0IR:~/209-shared/notes/20211/w03/demos$ gcc errors.c
errors.c: In function ‘main’:
errors.c:7:5: error: assignment to expression with array type
    7 |   a = b; // attempts to resize a by pointing it to the smaller array.
      |     ^
```

How to actually do it

```c
#include<stdio.h>
#include<stdlib.h> // provide malloc and free
int main()
{
  int *memptr = (int*) malloc(sizeof(int)*5);  //allocates memory from heap of 5 x size of integer and point to it.
  free(memptr); // places memory back on heap (without reinitializing)

  memptr = (int*) malloc(sizeof(int)*2); // re-allocates 2 integers
  free(memptr); // returns memory to heap before exiting
  return 0;
}
```

sizeof

- Returns the size of a datatype in bytes (not nibbles ☹).
- Benefits:
  - Don’t have to remember sizes of data types.
  - Increased portability since not all systems have same size of int, float etc.

OS	Architecture	Size of "long" type
Windows	IA-32	4 bytes
	Intel® 64	4 bytes
Linux	IA-32	4 bytes
	Intel® 64	8 bytes
mac OS	Intel® 64	8 bytes

(2008 data not up to date, but illustrates the point)

Malloc:

- Allocates memory from heap and returns a pointer to it in the heap.
  - Memory is not in stack frame.
  - Tracks size of allocated memory.
- If allocation fails, returns ptr = NULL.
- System does not track memory.
- All allocated memory must be freed!
- Failure to do this causes the memory to be lost
  - This is known as a memory leak.
- Exiting program frees all memory

Free:

- De-allocates memory stored in heap.
  - Uses information stored by malloc to decide how much it needs to free.
- Free should be used on all dynamically allocated memory when leaving the main program.
- There are cases where you may not want to free when you leave a function.

Example

```c
#include<stdio.h>
#include<stdlib.h>
int* allocator(int *ptr)
{
  if(ptr == NULL) //Checks to see if memory hasn't been allocated
    {
      printf("Allocated Memory \n");
      ptr = (int *) malloc(sizeof(int));
      *ptr=0;
      printf("Initial value = %d\n", *ptr);
    }
  else //increments the memory element and prints
    {
      *ptr +=1;
      printf("value of incremented element = %d\n", *ptr);
    }
  return ptr;
}
int main()
{
  int *ptr=NULL; // Makes a new pointer and points it at nothing
  ptr = allocator(ptr); // allocates memory, returns pointer to it.
  ptr = allocator(ptr); // increments memory
  ptr = allocator(ptr);
  free(ptr); //frees memory at address stored in ptr.
}
```



```
Allocated Memory
Initial value = 0
value of incremented element = 1
value of incremented element = 2
```