malloc and free

For example, let's assume we need a block of memory to hold a string of say 100,000,000 ints.

```
int *p;
p = malloc(100000000 * sizeof (int));
if (p == NULL) {
   printf("Error: array could not be allocated.\n");
   exit(1);
}
          // we can now use the pointer
          // ... lots of things to do

free(p);  // free up the memory that was used
```

sizeof

- **sizeof** - C operator yields bytes needed for type or variable
- **sizeof (type)** or **sizeof variable**
- note unusual (badly designed) syntax - brackets indicate argument is a type
- use sizeof for every malloc call

```
printf("%ld", sizeof (char));     // 1
printf("%ld", sizeof (int));      // 4 commonly
printf("%ld", sizeof (double));   // 8 commonly
printf("%ld", sizeof (int[10])); // 40 commonly
printf("%ld", sizeof (int *));    // 4 or 8 commonly
printf("%ld", sizeof "hello");    // 6
```

malloc and sizeof

- **sizeof** - C operator yields bytes needed for type or variable
- note unusual syntax **sizeof (type)** or **sizeof variable**
- use sizeof for every malloc call
- `malloc()` returns pointer to block of memory
- `malloc()` returns a (`void *`) pointer - can be assigned to any pointer type
- `malloc()` returns NULL if insufficient memory available - check for this

free

- `free()` indicates you've finished using the block of memory
- Continuing to use memory after `free()` results in very nasty bugs.
- `free()` memory block twice also cause bad bugs.
- if program keeps calling `malloc()` without corresponding `free()` calls program's memory will grow steadily larger called a **memory leak**.
- Memory leaks major issue for long running programs.
- Operating system recovers memory when program exists.