Initialization Errors

Most C programs use **`malloc()`** to allocate blocks of memory.

A common error is assuming that **`malloc()`** zeros memory.

Initializing large blocks of memory can impact performance and is not always necessary.

Programmers have to initialize memory using **`memset()`** or by calling **`calloc()`**, which zeros the memory.

Initialization Error

```
/* return y = Ax */
int *matvec(int **A, int *x, int n) {
  int *y = malloc(n * sizeof(int));
  int i, j;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      y[i] += A[i][j] * x[j];
  return y;
}
```

Incorrectly assumes `y[i]` is initialized to zero

Memory Management continued…

Problem 1

Once memory has been freed, it is still possible to read or write from its location if the memory pointer has not been set to null.

An example of this programming error:

```
for (p = head; p != NULL; p = p->next)
  free(p);
```

The correct way to perform this operation is to save the required pointer before freeing:

```
for (p = head; p != NULL; p = q) {
  q = p->next;
  free(p);
}
```

Problem 2

realloc(0)

The `realloc()` function deallocates the old object and returns a pointer to a new object of a specified size.

If memory for the new object cannot be allocated, the `realloc()` function does not deallocate the old object and its value is unchanged.

If the `realloc()` function returns a null pointer, failing to free the original memory will result in a memory leak.

The memory allocation may fail due to fragmentation… Internal/ External (OS refresher)

Standard Idiom Using `realloc()`

```
char *p2;
char *p = malloc(100);
...
if ((p2=realloc(p, nsize)) == NULL) {
  if (p) free(p);
  p = NULL;
  return NULL;
}
p = p2;
```

A return value of `NULL` indicates that `realloc()` did not free the memory referenced by `p`

Re-Allocating Zero Bytes

If the value of `nsize` in this example is 0, the standard allows the option of either returning a null pointer or returning a pointer to an invalid (e.g., zero-length) object.

The `realloc()` function for

- gcc 3.4.6 with libc 2.3.4 returns a non-null pointer to a zero-sized object (the same as `malloc(0)`)
- both Microsoft Visual Studio Version 7.1 and gcc version 4.1.0 returns a null pointer

Standard Idiom Using `realloc()`

```
char *p2;
char *p = malloc(100);
...
if ((p2=realloc(p, 0)) == NULL) {
  if (p) free(p);
  p = NULL;
  return NULL;
}
p = p2;
```

In cases where `realloc()` frees the memory but returns a null pointer, execution of the code in this example results in a double-free.

Don't Allocate Zero Bytes

```
char *p2;
char *p = malloc(100);
...
if ((nsize == 0) ||
    (p2=realloc(p, nsize)) == NULL) {
  if (p) free(p);
  p = NULL;
  return NULL;
}
p = p2;
```

Exercise:

Write a program to create a linked list with 10 elements, then deallocate the memory. (without any bugs)