**Chapter 14: Interlude: Memory API**

We will talk about memory allocation interfaces in UNIX systems.

**14.1 Types of Memory**

In a running C program, there are two types of memory that are allocated. The first is called the **stack** memory. Allocations and deallocations of it are managed implicitly by the compiler for you, the programmer. For this reason, it is sometimes called **automatic** memory.

When we declare a piece of memory, like “*int x;*” The compiler will do everything: make sure to make space on the stack and deallocate the memory.

This needs for a long-lived memory, named **heap** memory where all allocations and deallocations are explicitly handled by us, the programmer.

Text

Description automatically generated with low confidence

In the above declaration of heap memory, the compiler also allocates for a pointer in the stack.

**14.2 The malloc() call**

The malloc() call is simple as we just need to pass it a size asking for some room in the heap. If it succeeds and gives you back a pointer to the newly-allocated space, else if it fails and returns NULL. TO use this, include the header file stdlib.h. Example:



We must also be careful with string. When declaring space for a string, use the following idiom: malloc(strlen(s) + 1), which gets the length of the string using the function strlen(), and adds 1 to it in order to make room for the end-of-string character.

Malloc returns a pointer to type void. This is just for the programmer to decide what to do with is using **cast**. In the above example, we use (double \*).

**14.3 The free() Call**

To free memory that is no longer in use in the heap, programmers simply call free(), which takes one argument, which is the pointer returned by malloc().

**14.4 Common Errors**

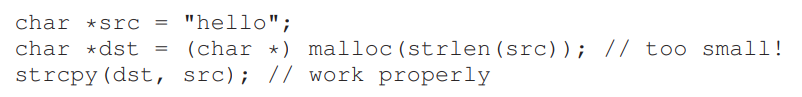
Newer languages now support for **automatic memory management**. In such languages, we never have to free space. Instead, a garbage collector runs and figures out what memory you no longer have references to and frees it for you.

**Forgetting to allocate memory:**

Many routines expect memory to be allocated before we call them. For example, when we use strcpy(dst, src), the dst pointer must be allocated first. On the other hand, if we use strdup(dst, src), we would not have to use it.

**Not allocating enough memory:**

When we allocate not enough memory, we call it buffer overflow. For example:

****