Dynamic Memory Allocation and Structs in C

March 19, 2014

malloc

It is often the case that you don't know how much memory you will need at compile time; only once the user tells you.

malloc/calloc

In C we can ask the operating system to reserve memory for us by using the `malloc` or `calloc` commands.

- `void* malloc(size_t size)` Returns a pointer to a block of memory that is `size` bytes long
- Example: To dynamically allocate a string that has room for 1000 characters do `char* mystring=(char*) malloc(1000*sizeof(char));`
- `malloc` does not initialize anything for you – who knows what is sitting around in that memory!

calloc

- Does the same job as malloc, but initializes all the data to be 0.
- `calloc(size_t typesize, size_t numtypes)` Allocates enough memory for `numtypes` variables with size `typesize`
- Allocate enough room for 500 `ints`

```
int* intarr=calloc(500,sizeof(int));
```

free

- Any time you `malloc` or `calloc` memory, the operating system reserves it for you until you explicitly return it to the OS using the `free` command.
- **Failure to do this results in a MEMORY LEAK**.
- `free(void* ptr)`

Look at `dyn_mem_example.c`

Look at `memory_leak.c`

Structs

A `struct` in the C language is a complex objects that groups logically related pieces of data. For instance, a message struct might contain information about the sender of the message, the data of the message, time recieved, length, etc.

Structs

The logically related pieces of information are called **members** of the struct.

Here is an example of declaring a struct that is a message.

```
struct message{
    char* data;
    int   origin;
    int   dest;
    int   length;
}
```

```
#include<stdio.h>
#include<stdlib.h>

struct message{
    char* data;
    int   origin;
    int   dest;
    int   length;
};

  void main(){
    struct message m;
    m.data=(char*) malloc(10*sizeof(char));
    m.data[0]='H'; m.data[1]='i'; m.data[2]='\n';
    m.origin=1;
    m.dest=2;
    m.length=15;
    printf("%s",m.data);
  }
```

You can `malloc` structs if you want to:

```
void main(){
   struct message* m = malloc(sizeof(struct message));
   m->length=15;
}
```

A cleaner way to use structs is to typedef

```
typedef struct{
   char*  data;
   int    origin;
   int    dest;
   int    length;
}message;
```

Look at `message_example_2.c`