Australian National University

Memory

Eric McCreath

Memory

When conceptualizing *c* programs we think of the program as executing in a large chuck of contiguous memory.

This memory holds:

- local variable (on the stack),
- global variables,
- the program executable binary or text,
- malloc(ed) sections of memory on the heap, and
- memory mapped parts of memory.

All this memory is much the same so care is needed such that we don't use it in a way it was not intended (e.g. executing your global variables, or writing to your text).

Getting Hold of Memory

We can get hold of some memory for use in out program by:

- declaring variables,
- using malloc, or
- using mmap.

malloc and free

malloc is a library call that returns a pointer to the memory you requested. The parameter for malloc is the number of bytes you request.

Once your program has finished using this block of memory you should use **free** to return it for later use.

If your program doesn't free memory it no longer uses then the amount of memory your program uses can grow over time. This will continue until your system runs out of memory. This is known as a *memory leak* .

Every malloc should be matched up with a free in your code.

```
#include <stdlib.h>
....
 int *data;
 data = (int *) malloc(sizeof(int) * 100); // allocate enough memory for 100 ints
 data[1] = 6;
 free(data);
```

Exercises

- Write a program with a memory leak. See how long it will run for. (note you may need to use this memory i.e. write some data to it and read the data out of it)
- Write a program that reads from standard input a list of floats. Store these floats in an array that you malloced.