COMP1917: 10 Dynamic Memory

Sim Mautner

s.mautner@unsw.edu.au

August 16, 2016

Purpose

- Current Problems:
  - Want to create arrays within functions and return them to the function which called them.
- Future Problems:
  - Don't want to limit the amount of data to be loaded as the application is used more over time.
  - Want to use as much memory as needed, no more, no less.

Stack and Heap

- Parameters and local variables are stored on the *stack*.
- Problem: When the current function returns, all variables stored on the stack are lost.
- Demo: What happens when we try to create an array in a function, return it, and then use it?
- Solution: Store values on the *heap* instead.

Aside: `sizeof`

- Function which returns the size (number of bytes) which a type requires.

```
int sizeOfChar = sizeof(char);
printf("The size of a char is %d\n", sizeOfChar);
```

malloc and free

- Used to allocate space on the heap to store a data structure (int, char, array).
- Returns a pointer to (the address in memory of) the allocated space.

```
int *justANumber = malloc(sizeof(int));

int *aWholeArrayOfInts =
        malloc(sizeof(int) * ARRAY_LENGTH);
```

malloc and free

- After mallocing, check that it was successful. (If there is not enough memory, malloc will not work and NULL will be returned.)

```
int *justANumber = malloc(sizeof(int));
if(justANumber == NULL) {
   printf("Out of memory.\n");
   exit(EXIT_FAILURE);
}

int *aWholeArrayOfInts =
          malloc(sizeof(int) * ARRAY_LENGTH);
if(justANumber == NULL) {
   printf("Out of memory.\n");
   exit(EXIT_FAILURE);
}
```

malloc and free

- To release memory that has been malloced, use `free()`.

```
free(justANumber);

free(aWholeArrayOfInts);
```

Exercises

- Ex 1: Write a function which asks the user to enter a length, and creates an array of that length.