COMP 211: Lab 2
Fall, 2014

1 Commands

1.1 Declaration and Assignment

Task 1.1 Write down the state of memory after executing each of the following commands:

```
int x = 0;

x = 4;

int y = 1;

y = x;

y = 17;
```

Is `x` changed by executing the final command? Why or why not?

1.2 Loops

How many times is the body of the following loop executed? What is the value of `e` each time the loop starts? What is the value of `e` at the end of the loop?

```
int e = 1;
while (e <= 8) {
  e  = 2 * e
}
```

Have someone check your work up to this point!

2 Handout Code

For the rest of the lab, you should edit the file `lab02.c0`, which is available from the course web page. The file currently contains some functions from class from yesterday. To load the file, download it and run

```
coin lab02.c0
```

from the directory where `lab02.c0` is stored.

3 Factorial

Suppose you have n friends coming over for dinner and you also have n chairs around your table. How many different ways are there to seat people?

Label the chairs $a, b, ...$ For chair a, there n choices for who sits there. For chair b, there are $n-1$ choices, because one person already has a chair. For chair c, there are $n-2$ choices, and so on. So in total, there are

$$n * (n-1) * (n-2) \ldots * 2 * 1$$

choices. This is called the *factorial* of n.

Task 3.1 Write a function `fact` that takes one integer argument `n` and returns the factorial of `n` (also an integer).

4 Arrays

An array of length `n` has elements numbered `0,1,2,...,n-1`, so whenever you write `A[i]`, you should be sure that `0 <= i < the length of A`. This is called checking that all array accesses are *in bounds*.

Task 4.1 In coin,

- create an array of length 3 using the command

  ```
  --> int[] A = alloc_array(int,3);
  ```

- what value is in `A[0]` and `A[1]` and `A[2]` at first?
- store the values `1,2,3` in its three cells using the commands

  ```
  --> A[0] = 1;
  ```

  etc.

- What happens if you try to access `A[3]`?

Task 4.2 Write a function

```
int mult_array(int[] A, int length)
```

that multiplies together all of the numbers in `A`, and returns this product. You can assume that the second argument `length` is the length of `A`.

Create an array in `coin` like above and test your function.

Task 4.3 Write a function

```
void addOne(int[] A, int length)
```

that modifies `A` by adding one to each element. The second argument `length` represents the length of `A`. The return type of `void` indicates that the function does not return any interesting value; one calls the function solely for the changes it makes to `A`.

Create an array `A` of length `3` in `coin` like above, run the command

```
--> addOne(A,3);
```

And then check that `A[0]` and `A[1]` and `A[2]` have the right values.

Task 4.4 Write a function

```
int[] increasing(int n)
```

that creates an array containing the numbers `0,1,2,3,...,(n-1)` in the corresponding position. Test your function in `coin`.

Task 4.5 In coin, run the following commands

```
--> int[] a = alloc_array(int,2);
--> a[0] = 1;
--> int[] b = a;
--> b[0] = 17;
--> a[0];
```

Explain why the value of `a[0]` is what it is.

Task 4.6 Define a function `addOneCopy`

```
int[] addOneCopy(int[] arr, int size)
```

that is like `addOne`, except instead of changing `arr`, it creates and returns a *new* array whose elements are the elements of `arr` with 1 added to them. The original array `arr` should be unchanged.

Write a series of commands in `coin` that shows the difference between `addOne` and `addOneCopy`.