

Pointers Review for Understanding MEX Codes

Phys494 Applied Fourier Analysis Lab, by İsmail Arı

March 5, 2008

1 Pointer Basics

Pointers always became the most difficult part to understand in C programming language but let us review it and see how easy it is.

We will only go through the basics so we can understand the code of MEX files.

Now download the 3-minute funny pointer video from the course site and watch it before we go on.

1.1 Pointers and Pointees

A **pointer** stores a reference to something. There are numerous types or things that a pointer can point to (integers, floats, classes, functions, etc.). The pointed thing is called **pointee**.

If we are interested in vectors or matrices (i.e. 1D and 2D arrays) the pointer of these will be pointing to the start of the array. Arrays are also pointers which we use with a simpler syntax.

Allocating a pointer and allocating its pointee are two separate steps. We should think of the pointer/pointee structure as operating at two levels. Both of them must be set up for things to work.

Example: Let us define a pointer which is told to point an integer data.

```
int *x;
```

Notice that we set up the pointer but there's not a pointee yet, so if we write

```
*x = 17;
```

which means "Assign the pointee of x to 17", the program will crash! Because we did not accomplished the memory allocation part of the pointee. So, we need to write

```
x = malloc(sizeof(int));  
*x = 17;
```

Here we allocated a memory region with **malloc** function where we told it to allocate bytes equal to **sizeof(int)**.

1.2 Dereferencing

The dereference operation starts at the pointer and follows its arrow over to access its pointee. The goal may be to look at the pointee state or to change the pointee state. This operation only works when there is the pointee of the pointer. Remind that, the program crashes if we forget to allocate the space for the pointee.

1.3 Pointer Assignment

Pointer assignment between two pointers makes them point to the same pointee.

```
double *x;
double *y;
x = malloc(sizeof(double));
*x = 45.2;
```

We created a space for x and stored a double value in this pointee, now let us make the pointer assignment

```
y = x;
*y = 12.7;
```

Finally, the pointee is assigned to 12.7.

2 Pointers as Arrays

We can use pointers as arrays in our programs. This is used when the length of the array is not a constant value. Remind that the arrays must be created with constant lengths.

```
double x[5]; /* Defined an array of length 5 */
int len = 10;
double y[len]; /* Not Allowed in C !!! */
```

This should be done by using pointers as

```
int len = 10;
double *y;
y = malloc(sizeof(double) * len);
```

We created a space that has the byte number equal to 10 times of a double size.

Let us look at the equal representation in arrays and pointers

```
y[0] = 5;
*y = 5;
```

```
y[4] = 2.7;
*(y+4) = 2.7;
```

Both representations are correct but the former ones are more readable.

Now, study the next C example and try to code the exercise on your own.

2.1 Example

Write a C function that takes a pointer to an array and returns the an array involving only the even elements of the input array. Assume the index starts from 0 and the input array has even number of elements

The function name will be **evens**, it will take two parameters (one is the pointer to the array and the other is the length of the array) and it will return the evens array.

Solution: attached as *evensAndOdds.c*

2.2 Exercise

Modify the example code such that the returning array includes the odds of the input array. You will only complete the code for **odds** function and uncomment the related parts of the code.