What do I need to Know For My Assignment?

C Pointer Review

- To declare a pointer, we use the * operator. This is similar to but different from using * not in declaration.
- int* ptr;
- This declares a variable that points to an integer

C Pointer Review

- The & or “address of” operator gives us a pointer to the variable.
- Int i;
- The type of &i is *int

C Pointer Review

- The * or “dereferencing” operator, will return the variable that a pointer is pointing to
- Int* i
- *i is of type int

- Remember for structures that (*struct).value == struct->value

C Pointer Review

- A char* can point to a string of any length, because really we are storing the value of the address of the first character.
- In C a string is a set of characters until the termination character ‘/0’
- Arrays are also pointers, only their length is pre-defined

C Pointer Review

- Given double b[5];double *bPtr; bPtr = b;

- bPtr = &b[0]
- bPtr + 1 ==
- *(bPtr + 3) ==
- bPtr[2] ==

C Pointer Review

- Void* ptr;
- This is a void pointer, meaning it can point to anything.
- We can use this to fake polymorphism, i.e. making a linked list using void pointers means we can actually store anything we want in the list

C Pointer Review

- When a pointer points to nothing it is said to have NULL value
- NEVER DEREFERENCE NULL VALUES
- We can check to see if a pointer is null easily
- Int * i;
  if (i==NULL)
  printf(“Obviously True”);

C Pointer Review

- With the exception of arrays and functions, C is always pass by value. This means that the value of your parameter is passed into the function. Therefore changing the value in your function will have no effect
- This is why we must use pass by reference for many functions.

C Pointer Review

- Void foo1(int i) { i = 0; }
- Void foo2(int *i){*i=0;}
- Int main(void) {

```
        int k = 10; foo?(k);
        printf("k = %d",k);
        return 0;
}
```

Data Structures

- Data Structures are, as the name implies, a structure to hold data.
- They all have 3 basic components; constructors, selectors and mutators

Constructors

- Constructors are the part of the structure used to “construct” a variable.

Selectors

- Selectors are procedures that selects and returns the stored data

Mutators

- Mutators are procedures that will mutate the data into an easier more useful form

Our Tools in C

- Typedef lets you rename a type to something else
- Typedef <existingType> <newTypeName>
- Typedef int age;

Our Tools in C

- Enum is a tag that lets us define enumerations
- Enum {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
- !!!!!This does not let you use the type!!!!!
- To use the type, we must typedef
- Typedef Enum {Mon, Tue, Wed, Thu, Fri, Sat, Sun} Day;

Our Tools in C

- Enum looks much like a typedef, because it essentially gives new names for numbers
- It really looks like
  Enum {Mon = 0, Tue = 1, Wed = 2, Thu = 3, Fri = 4, Sat = 5, Sun = 6};
- We can, should we choose, define a type
  Typedef enum {apple = 5, pear = 7} Fruit;

Our Tools in C

- Struct lets us build more complicated types, much like records
- Typedef struct
  { int studentNo
    char* name
  } Student;
- Once again the above is shorthand

Our Tools in C

- Structs can have other structures, arrays and pointers
- Putting a pointer to the same structure allows recursive data types like lists, trees, etc.

Our Tools in C

- To make memory space for our types it is often necessary to use the function malloc, to allocate memory.
- Int *i; automatically sets enough space for an integer, but should we want a pointer to complicated data structures we need malloc
- Malloc takes the number of bytes and returns a pointer to enough space

Our Tools in C

- Sizeof is an operator that returns the size of anything, like the number of bytes necessary for a malloc
- Char c; (sizeof c) == 8
- For arrays remember to find the size of the whole array
- Int i [length]];
  (sizeof i) * length

Date Example

- Lets now create a data type in C for the Date
- We will store, month, day of the month and day of the week

Date Example

- We will do this by creating two enumerations for Month and Day
- typedef enum { JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC } Month;

- typedef enum { MON, TUE, WED, THU, FRI, SAT, SUN } Day;

Date Example

- Now we combine them into the data structure
- Typedef struct { Month month, Day day, int dayNum} Date;

Date Example

- Now we add a constructor
- Date* newDate(Month mo, Day da, int i){
  Date *d = malloc(sizeof(Date));
  if (d = null)
  fprintf(stderr, “Failure Making New Date”);
  else {d->month = mo; d->day = da;
  d->dayNum = I;}
  return d;
  }

Date Example

- Now we add selectors
- Month getMonth(Date *d)
  { return d->month;}
- Day getDay(Date *d)
  {return d->day;}

Date Example

- Now we add a mutator
- Date* tommorow(Date *d){
  return newDate(
  (d->month + ((d->dayNum+1) % 30))% 11,
  (d->day + 1)%6,
  (d->dayNum+1)%30);
  }