**ELEC 278 Notes – Course notes for the whole semester**

**Week 1 – Basic Programming Concepts**

* 4 basic components

1. **Processor** – fetch, decode & execute instructions
   1. Has small memory 🡪 registers, one will be NI (Next Instruction)
2. **Memory** – instructions + data kept here (in bits), can’t distinguish between either whilst in memory
3. **I/O** – allow entry/ exit of data
4. **Bus** – connects processor, memory + I/O
   1. 3 wires (Address, data, control)

* Read and load operations (Load/store)
* Write information to, read information from
* 8 bits = 1 byte, bit is state 0/1
* 5 fundamental features: fundamental items of data, basic manipulation, collections of data, control of execution, modularization, and data exchange between modules
* String = array of chars
* Can declare and init in same line
* Scope where name is valid, lifetime, time variable exists

**Week 2 – Pointers & Operators**

* Pointer – variable that points to another
  + Holds location of variable being pointed to
  + Are typed, type of var being pointed to must be known
* Use ‘\*’ to declare pointer
* Use & (address of) to point pointer to var
* Use ‘\*’ to access pointed to vars
* Pointer arithmetic 🡪 +/- of integer from pointer, number used is multiplied by size of thing pointer points to
* Use cast (type \*)
* Void pointers, hold address, what it points to == unknown
* Enum allows for custom var types
* Typedef, add own types to program
* Key operators
  + A = ++b, b incremented, value assigned to a
  + A = b++ value assigned to a, b incremented

**Week 3 – Iteration & Code Modularization**

* **Iteration** 🡪 repeating steps until certain condition met or not
* Definite Iteration 🡪 know # of steps that needs to be done (For)
  + For (start; stop; step)
  + Can have more than one start, stop or step
* Indefinite 🡪 Recognize when done (While)
  + Pre check, if true (execute), else don’t
  + Do while 🡪 do at least once, evaluate outcome (Post check)
    - True 🡪 enter while loop
    - False 🡪 exit, move on to next statement
  + Use break to ‘break’ out of while loop, then go to next statement of code
  + Continue – keeps you within the current block
* C doesn’t distinguish between for and while loops
* **Functions** 🡪 making reusable pieces of code
  + Basic functions, parameters and return values
  + Can be invoked anywhere in program
    - Optimal if functions are less than 60 lines
  + Definition
    - Type of value returned by function
    - Name
    - Parameters and their types
* **Hybrid Storage**
  + Arrays 🡪 collection of identical items
  + Structures 🡪 collection of heterogenous (dissimilar) – **struct**
    - **Sizeof()** – gives # of bytes occupied by instance of type
    - Variables can be declared with definition
    - Can use **typedef()**
    - Use dot notation to access fields
  + Unions 🡪 multiple ways to store same data
    - Might want to treat memory in 2 different ways
      * One type sometimes, another the other times
  + Malloc() – allocates memory at run time
* **Alternate Code Paths**
  + Conditional Expression – Assigns values based on values of Boolean condition
    - X = <condition>?<expression1>:<expression2>
    - Expression 1 if true, expression 2 otherwise
* **Flow Control**
  + Choices
    - 1/ None/ other/ many
  + Repetition (Iteration)
    - Definite (for), Indefinite (while)
      * Indefinite (Pre/ Post check)
  + Switch case
    - ???

**Recursive Functions**

* Recursion 🡪 define something in terms of itself
  + Code 🡪 function using itself, ‘calls itself’
* Must have condition where it **WON’T** recurse
* Can use recursion to reverse order of string
* Infinite recursion can occur when
  + End condition not met, forgotten
  + Will terminate when ‘stack overflows’
* Functions to study – factorial, Fibonacci, ackerman

**Fundamental Information Structures**

* Arrays & Linked Lists = fundamental data structures
* Arrays
  + Correspond to chunk of memory
  + Reference without square brackets is = to pointer to first item in array
  + Outside valid index range
    - Compiler generates bounds-checking code 🡪 verifies if index is valid
    - No bounds checking 🡪 segment fault
  + Has fixed size
* Linked list
  + Text, table

    Description automatically generatedOne part of data structure is pointer to next
  + Some operations
    - Add new node
      * To empty list
      * At beginning
      * In between
      * At end
    - Locate a node
    - Delete node
      * Same as addition

**Week 4**

* Text

  Description automatically generatedNode; consists of key and value
* Use free() to destroy node
  + Free(nodepointer)
* Descriptor – contains no data, let us reference the list
* Structural info – pointers between nodes
* Arrays have fixed size, can have multiple dimensions
  + Rows, columns….
* Doubly linked, points in both directions

**Week 5**

* Stack 🡪 LIFO
  + Only need to keep track of head position
* Queue 🡪 FIFO
  + Can be implemented w/ array or linked list, linked list is preferred
  + Node only needs a pointer to the next item, and a value
  + Need front and end pointer for the queue
  + Can implement as a stuct (for multi – instance)

**Week 6**

Dequeue 🡪 Double ended queue, can add only to front and back

* + Nodes will need pointers that point in front and behind
    - Node
      * 🡪 next, prev, a value
* Recursion
  + When a function calls itself within a function definition
    - Factorial, Fibonacci sequence
  + Must have a base case i.e.
    - If (r == NULL) return 0;

**Week 7** – **Trees**

* Binary trees
  + Each node has either: no child (leaf), 1 child, 2 child)
  + Each tree node structure has 3 fields
    - A data value/ pointer to data value
    - 2 tree node structs pointing to left & right child
  + For a binary tree with all integer values
    - To sum values 🡪 use recursion to find sum the value of root node, left trees and right trees
      * Sum = root-> value + left\_sum + right\_sum;
      * Where left\_sum & right\_sum recursively call the tree total function to figure out the sums of the left & right trees
* Binary **SEARCH** trees
  + Root node used for comparison
    - Left tree less than root
    - Right tree greater than root
  + Can’t have duplicate values
* Recursion vs Iteration

**Quiz 2**

* Out of 50
* 4 questions (MC, short answer)
* Source code, write I/O of function