Section 1

C Programming
C Basics

- Data Types
  - void
  - char
  - int
  - float

- Assignment
- Arithmetic Operations
- Logical Operations
- Relational Operations
Control Structures

- Branch Structures
  - if/else
  - switch

- Loops
  - while
  - for
  - break
  - continue
Input/Output

- printf
- fgets
Organizing Data

- Structures (struct)
- Arrays
- Strings
  - strlen
  - strncpy
  - strcmp
Functions

- Declaring Functions
- Using Functions
- The main Function
Memory Basics

- Pointer Reference/Dereference
- Type Casting
- Arrays v. Pointers
- void Pointers
Dynamic Memory

- Allocating Memory
- Freeing Memory
Section 2

Formal Programming Languages
Regular Expressions

Expressions that are constructed with character (sets) and the following operations

- Alternation (’|’)
- Iteration - zero or more (’*’)
- Iteration - one or more (’+’)
- Option (’?’)
- Concatenation

are called regular expressions

Which of the following strings matches (a|b)*(c|d)?

- aaaaa
- abca
- abc
- d
Extended Backus-Naur Form (EBNF)

EBNF can be used to formally express a language (context-free grammar)

- Definition (‘:=’, ‘::=’)
- Alternation (‘|’)
- Iteration - zero or more (‘{…}’)
- Option (‘[…]’)
- Concatenation

What does the following EBNF express?

```
bin_digit ::= 0|1
bin_number ::= [+|−]{bin_digit}
```
This is a tree representation based on an EBNF definition.

\[
\text{unop} ::= - \\
\text{binop} ::= -|+|*|/|% \\
\text{expr} ::= \text{number}|\text{name}|(\text{expr})| \\
= \quad \text{unop} \ \text{expr}|\text{expr} \ \text{binop} \ \text{expr}
\]

\[
2 \ast (\text{var}_x + 25)
\]
int x, y;
x = read();
y = read();
while (x != y)
    if (x < y)
        y = y - x;
    else
        x = x - y;
write(x);

Start

x = read();
y = read();

x != y

no

x < y

no

x=x-y;

yes

Stop

y=y-x;
Section 3

Data Structures
Linked Lists

Singly Linked List

Doubly Linked List
Stacks and Queues

**Stack**
- push()
- pop()

**Queue**
- enqueue()
- dequeue()
A *binary tree* is a tree in which every node has a maximum of two children.

*Remember:*

- Representation (pointers & structs v. adjacency matrices)
- Terminology
Section 4

Algorithms
Recursion is the process of solving a problem by dividing the problem and solving (multiple) smaller sub-problem(s) of the same kind.

\[ f(x) = \begin{cases} 
1 & \text{if } x = 1, \\
x \times f(x - 1) & \text{if } x > 1 
\end{cases} \]

**Remember:**
- Define base case
- Define recursive case
Tree Traversal

When we consider a tree, there are many possible ways to traverse the nodes in that tree. The most general classes of tree traversal are:

- Breadth-first Traversal
- Depth-first Traversal
  - Pre-order Traversal
  - In-order Traversal
  - Post-order Traversal
A binary search is a method of searching a sorted space (e.g., array) by comparing our search value to the middle value and cutting our search space in half. Assume we want to determine whether the number 33 is in our array.

```
  2  5  7  12  17  20  33  34  50  65
  2  5  7  12  17  20  33  34  50  65
  2  5  7  12  17  20  33  34  50  65
  2  5  7  12  17  20  33  34  50  65
```
Bubblesort is a simple sorting algorithm that performs multiple iterations of comparing each element with its neighbor and determining whether a switch is necessary. This continues until no more switches are necessary.

Here a single iteration...

```
2  12  34  5  20  17  50  7  33  65  no switch
2  12  34  5  20  17  50  7  33  65  no switch
2  12  34  5  20  17  50  7  33  65  switch
2  12  34  5  20  17  50  7  33  65  switch
2  12  34  5  20  17  50  7  33  65  no switch
2  12  34  5  20  17  50  7  33  65  switch
2  12  34  5  20  17  50  7  33  65  no switch
2  12  34  5  20  17  50  7  33  65  switch
2  12  34  5  20  17  50  7  33  65  switch
2  12  34  5  20  17  50  7  33  65  no switch
```
Mergesort is a 'divide and conquer' approach to sorting in which the general idea is that we will split the array into two equal halves, sort the halves, then merge the two sorted halves together to create a sorted array.

```
Mergesort([12,9,1,17])
  /     \
[12,9]  [1,17]
  |       |
[12]    [9]
  |       |
[12]    [9]

mergesort([12,9])
  /     \
[12]  [9]
  |    |
12    9

mergesort([12])
  / \
[12] 
12

mergesort([9])
  / \
[9] 
9

mergesort([1,17])
  /    \
[1]  [17]
  |    |
1    17

mergesort([1])
  / \
[1] 
1

mergesort([17])
  / \
[17]  
17
```
Mergesort (cont.)

merge([12], [9])

merge([1], [17])

merge([9, 12], [1, 17])

merge([12], [9])

merge([1], [17])

merge([9, 12], [1, 17])
Quicksort is a 'divide and conquer' approach to sorting in which the general idea is to choose a pivot element, split the elements into groups less than and greater than the pivot, sorting each group, then concatenating the sorted groups.

```
| 5 | 9 | 3 | 7 |
```

```
| 3 | 5 |
```

```
| 9 | 7 |
```

```
| 7 | 9 |
```

```
| 3 |
```

```
| 5 |
```

```
| 7 |
```

```
| 9 |
```

```
| [] |
```

```
Quicksort (cont.)

cat([3,5],[7,9])

3 5 7 9

3 5
7 9
cat([3,5],[7,9])
cat([7,9],[])