SLL

```
Define a structure for a Node:
  - Node contains two parts: data and next (a pointer to the next node)

Initialize START as NULL (this represents an empty list)

Function isempty:
  - Check if START is NULL (i.e., the list is empty)
  - Return true if empty, otherwise false

Function createSLL:
  - Set START to NULL to create an empty singly linked list (SLL)

Function insertBegin(newele):
  - Create a new node with the given element (newele)
  - Set the new node's next pointer to START (the current list)
  - Set START to the new node (new node becomes the first element in the list)

Function insertAfter(newele, existingele):
  - Create a new node with the given element (newele)
  - Traverse the list to find the node with data equal to existingele
  - If found:
    - Set the new node's next pointer to point to the next node of the found node
    - Set the found node's next pointer to the new node

Function deleteBefore(existingele):
  - Check if the list is empty; if true, print an error message
  - If there's no element before existingele (i.e., it's at the start or second position), print an error message
  - Traverse the list to find the node before the node containing existingele
  - If found:
    - Remove the previous node and update the links
    - Free the memory and return the deleted element
  - If not found, print an error message

Function display:
  - If the list is empty, print a message
  - Otherwise, traverse the list and print the data of each node until the end

In the main program:
  - Create an empty list
  - Insert several elements at the beginning of the list
  - Display the list
  - Insert a new element after a specific element
  - Display the list again
  - Delete the element before a specific element
  - Display the list one last time
```

Stack

```
Define a structure for a Node:
   - A node has two parts:
      1. data (an integer)
      2. next (pointer to the next node)

Define a structure for the Stack:
   - A stack has one part:
      1. top (points to the top node in the stack)

Function to create a stack (createStack):
   - Allocate memory for a new stack
   - Set the top of the stack to NULL (indicating an empty stack)
   - Return the new stack

Function to push an element onto the stack (push):
   - Create a new node
   - Set the node's data to the given value
   - Link the new node to the current top of the stack
   - Update the top of the stack to the new node
   - Print the pushed element

Function to pop an element from the stack (pop):
   - If the stack is empty, print a message and return -1
   - Otherwise, remove the top node
   - Store its data and free the node
   - Update the top of the stack to the next node
   - Return the popped element

Function to display the top element of the stack (displayTop):
   - If the stack is empty, print a message
   - Otherwise, print the data of the top node

In the main function:
   - Create a new stack
   - Push three elements (10, 20, and 30) onto the stack
   - Display the top element
   - Pop elements one by one from the stack, displaying the top element after each pop
   - If the stack is empty, print a message
```

Infix to Postfix

Define a structure for a Node:

- A node has two parts:
    1. data (a character)
    2. next (pointer to the next node)

Define a structure for the Stack:

- A stack has one part:
    1. top (points to the top node in the stack)

Function to create a stack (createStack):

- Allocate memory for a new stack
- Set the top of the stack to NULL (indicating an empty stack)
- Return the new stack

Function to push an element onto the stack (push):

- Create a new node
- Set the node's data to the given value
- Link the new node to the current top of the stack
- Update the top of the stack to the new node
- Print the pushed element

Function to pop an element from the stack (pop):

- If the stack is empty, print a message and return -1
- Otherwise, remove the top node
- Store its data and free the node
- Update the top of the stack to the next node
- Return the popped element

Function to peek at the top element of the stack (peek):

- If the stack is empty, return -1
- Otherwise, return the data of the top node

Function to check if a character is an operator (isOperator):

- Return true if the character is one of +, -, *, or /

Function to get the precedence of an operator (precedence):

- Return 1 for + and -
- Return 2 for * and /
- Return 0 for anything else

Function to convert an infix expression to a postfix expression (infixToPostfix):

- Create an empty stack
- Loop through each character in the input expression:
    - If the character is an operand (letter or number), add it to the output
    - If the character is '(', push it onto the stack
    - If the character is ')', pop from the stack and add to the output until '(' is found
    - If the character is an operator:

```
            - Pop from the stack and add to the output while the precedence of the operator on the stack is greater than or equal to the current operator
            - Push the current operator onto the stack
    - After the loop, pop all remaining operators from the stack and add them to the output
    - Print the postfix expression

In the main function:
    - Define an infix expression
    - Print the original infix expression
    - Call the function to convert the infix expression to postfix
    - Print the postfix expression
```

Priority Queue

```
Define a structure for PriorityQueue:
    - An array 'data' of fixed size MAX to store elements
    - Two variables 'front' and 'rear' to track the front and rear of the queue

Function to create an empty queue (createQueue):
    - Set both 'front' and 'rear' to -1 (queue is empty)

Function to check if the queue is empty (isEmpty):
    - Return true if 'front' is -1

Function to check if the queue is full (isFull):
    - Return true if 'rear' equals MAX - 1 (queue is full)

Function to insert an element in the queue (insert):
    - If the queue is full, print an error message
    - If the queue is empty:
        - Set 'front' and 'rear' to 0, insert the value at 'rear'
    - If the queue is not empty:
        - Start from the 'rear' and shift elements that are smaller than the new value to the right
        - Insert the new value in its correct position
        - Increment 'rear'

Function to delete an element from the queue (delete):
    - If the queue is empty, print an error message
    - Otherwise:
        - Print and remove the element at the 'front'
        - If 'front' equals 'rear', reset both to -1 (queue is now empty)
        - Otherwise, increment 'front' to remove the element
```

```
Function to display all elements in the queue (display):
    - If the queue is empty, print an error message
    - Otherwise, print all elements from 'front' to 'rear'

In the main function:
    - Create an empty priority queue
    - Display menu options:
        1. Create a new queue
        2. Insert a value into the queue
        3. Delete an element from the queue
        4. Display all elements in the queue
        5. Exit the program
    - Based on user input, perform the appropriate operation
```

Binary Search Tree

```
Define a structure for a Node:
    - A node has four parts:
        1. data (the value)
        2. left (points to the left child)
        3. right (points to the right child)
        4. parent (points to the parent node)

Function to create a new node (createNode):
    - Allocate memory for a new node
    - Set its data to the given value
    - Set its left and right children to NULL (no children yet)
    - Set its parent to the given parent node
    - Return the new node

Function to create an empty BST (createBST):
    - Return NULL to represent an empty tree (no nodes yet)

Function to insert an element into the BST (insert):
    - If the tree is empty, create a new root node with the given value
    - If the tree is not empty:
        - If the new value is smaller than the root's value, insert it in the left subtree
        - If the new value is larger, insert it in the right subtree
        - Repeat this process until the correct position is found
    - Return the root node

Function to search for an element in the BST (search):
    - If the current node is NULL or matches the search value, return the node
```

```
  - If the search value is smaller, search in the left subtree
  - If the search value is larger, search in the right subtree

Function to find the minimum value in a subtree (findMin):
  - Traverse the left subtree until you reach the leftmost node (the minimum value)
  - Return the node with the smallest value

Function to delete an element from the BST (delete):
  - If the tree is empty, return NULL
  - If the value is smaller than the current node's value, delete it from the left subtree
  - If the value is larger, delete it from the right subtree
  - If the value matches the current node:
    - If the node has no left child, return its right child
    - If the node has no right child, return its left child
    - If the node has two children:
      - Find the minimum node in the right subtree
      - Replace the current node's value with the minimum node's value
      - Delete the minimum node from the right subtree
  - Return the modified tree

Function to get the parent of a node (getParent):
  - Search for the node with the given value
  - If the node is found and has a parent, return the parent node
  - If the node is not found or is the root, return NULL (no parent)

Function to display the BST in Inorder traversal (DisplayInorder):
  - Traverse the left subtree, display the root, then traverse the right subtree
  - Print the values in sorted order

In the main function:
  - Create an empty BST
  - Insert several elements into the BST
  - Display the BST in Inorder traversal
  - Search for an element in the BST
  - Delete an element from the BST
  - Display the BST again in Inorder traversal
  - Get and print the parent of a specific element
```