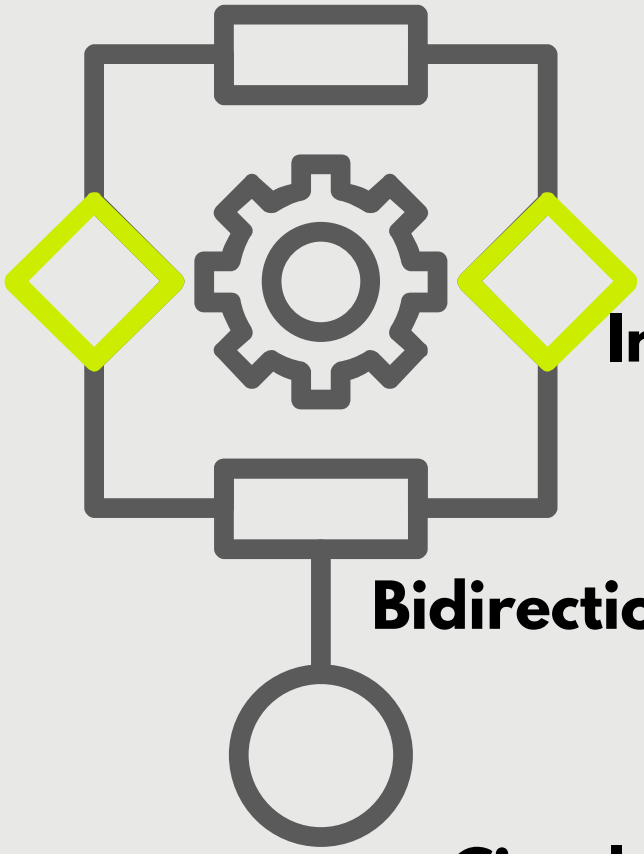
REPORT



01

Introduction

02

Bidirectional Linked Lists

03

Circular Linked Lists

04

Queue Log System

05

stacks

06

Recursion and Mixed Functions

07

Binary Search Tree Log System

01

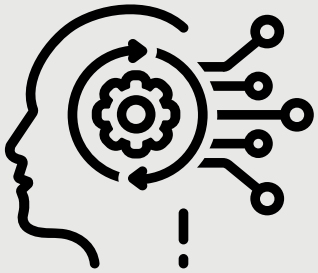
Linked Lists

1. Overview

This C program implements a log management system using linked lists. Each log entry contains an ID, severity level, message, and timestamp. The system allows insertion, deletion, search, sort, reverse, and count operations.

2. Data Structure

```
typedef struct logentry {
    int id, severity;
    char message[256];
    char timestamp[40];
    struct logentry *next;
} logentry, *Plogentry;
```



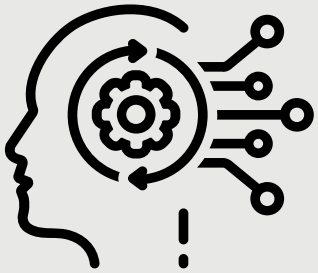
InsertLogEntry

Purpose: Adds a new log entry to the list.

```
InsertLogEntry(&head, &tail, 101, 5, "Network initialized", 0);
InsertLogEntry(&head, &tail, 102, 3, "User login", -1);
InsertLogEntry(&head, &tail, 103, 4, "Firewall alert", 1);
```

Output:

```
ID: 101, Severity: 5, Message: Network initialized, Date: (curren
ID: 103, Severity: 4, Message: Firewall alert, Date: ...
ID: 102, Severity: 3, Message: User login, Date: ...
```



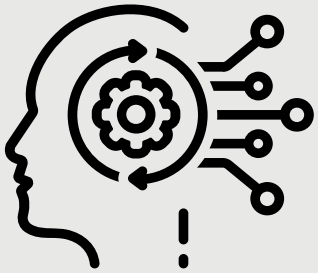
Delete_Log_Entry_ID

Purpose: Deletes a log by ID.

```
Delete_Log_Entry_ID(&head, &tail, 101);
```

Output:

```
Log with ID 101 is removed from the list.
```



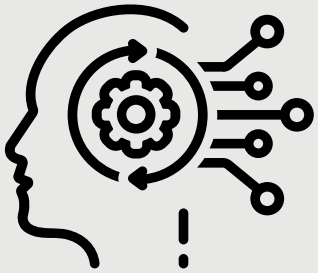
serch_by_id

Purpose: Searches for a log by its ID.

```
logentry *found = serch_by_id(head, tail, 102);
if (found) printf("Found: %s\n", found->message);
```

Output:

```
Found: User login
```



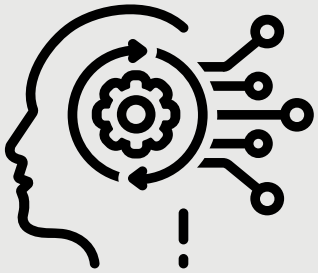
serch_by_date

Purpose: Finds a log with a specific date

```
logentry *result = serch_by_date(head, tail,
```

Output:

```
Returns the node containing the log with the specified date
```



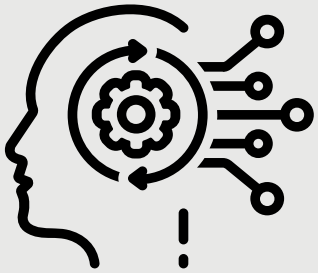
serch_by_keyword

Purpose: Searches for logs containing a keyword.

```
logentry *match = serch_by_keyword(head, tail, "login");
```

Output:

```
Returns the node containing "login" in the message field.
```



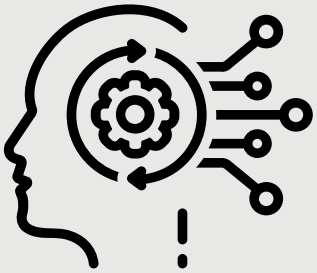
sort_log_by_severity

Purpose: Sorts logs by severity.

```
sort_log_by_severity(&head, &tail);
```

Output:

```
Logs rearranged in descending order of severity.
```



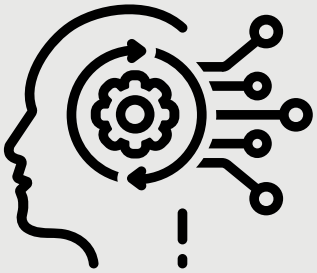
sort_log_by_date

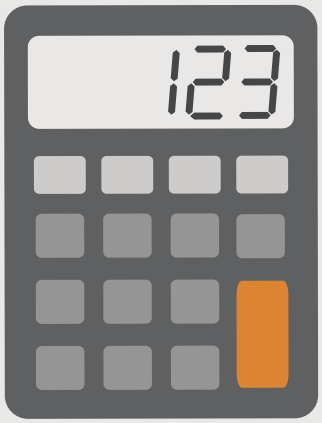
Purpose: Sorts logs by date.

```
sort_log_by_date(&head, &tail);
```

Output:

```
sorted from latest to oldest based on timestamps.
```





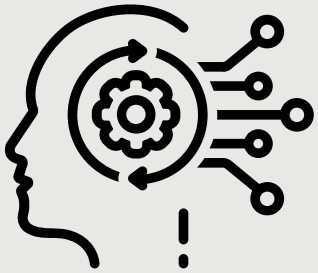
CountTotalLogs

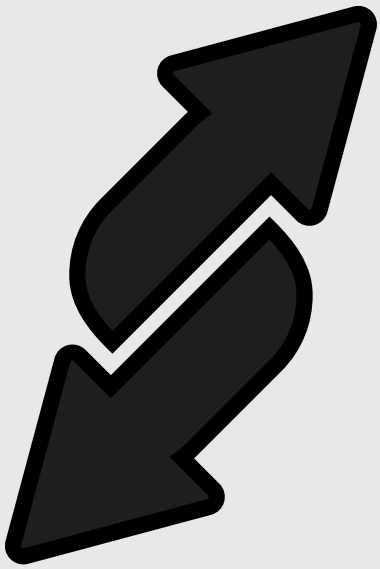
Purpose: Counts total number of logs

```
int total = CountTotalLogs(head, tail);
printf("Total Logs: %d\n", total);
```

Output:

```
Total Logs: 3
```





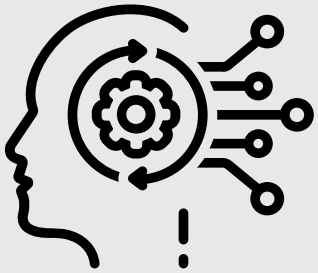
reverse

Purpose: Reverses the list.

```
reverse(&head, &tail);
```

Output:

```
The order of logs is reversed.
```



02

Bidirectional Linked Lists

1. Overview

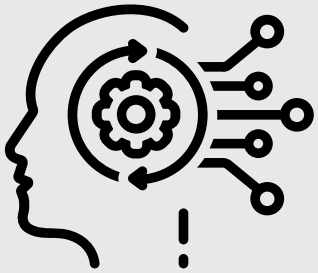
This C program implements a log management system using bidirectional (doubly) linked lists. Each log entry contains an ID, severity level, message, and timestamp. The system allows insertion, deletion, search, sort, reverse, and count operations.

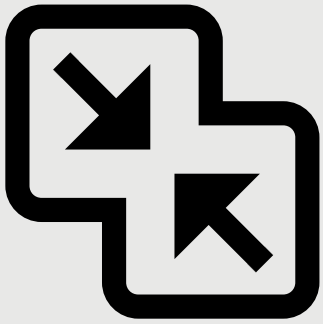
2. Data Structure

```c
typedef struct logentry {
    int id, severity;
    char message[256];
    char timestamp[40];
    struct logentry *next;
} logentry, *Plogentry;
```

note:

we All functions from singly linked list but with bidirectional navigation





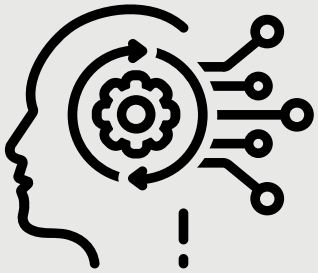
MergeLogLists

Purpose: Merges two log lists by appending the second to the first.

```
MergeLogLists(&head1, &tail1, head2);
```

Output:

```
Log list 2 is merged at the end of log list 1.
```



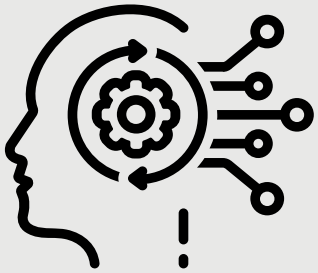
DeleteLogAtMiddle

Purpose: Deletes the log at the middle.

```
DeleteLogAtIndex(&head, &tail, 2);
```

Output:

```
Log at index 2 is removed from the list.
```



03 Circular Linked Lists

1. Overview

This C program implements a log system using a circular linked list. All core functionalities from singly and bidirectional linked list systems are preserved, with a circular structure enabling constant-time cycling through logs. Logs are structured with ID, severity, message, and timestamp fields.

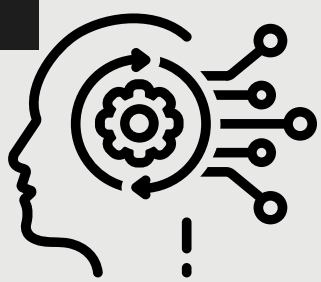
2. Data Structure

```
typedef struct clogentry {
    int id, severity;
    char message[256];
    char timestamp[40];
    struct clogentry *next;
} Clogentry, *PClogentry;

typedef struct logbuffer {
    PClogentry oldest;
    PClogentry recent;
    int size;
    int maxsize;
} Logbuffer, *PLogbuffer;
```

note:

All functions from singly linked list but with circular traversal



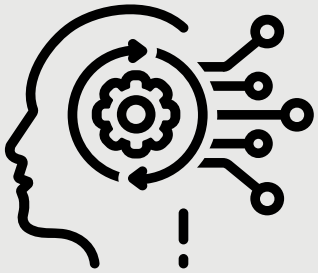
Implement a Fixed-Size Log Buffer

Purpose: Implement a Fixed-Size Log Buffer (overwrite old logs automatically)

```
PLogbuffer logBuffer;
createlogbuffer(&logBuffer, 3);
insertlogbuffer(logBuffer, 1, 2, "First");
insertlogbuffer(logBuffer, 2, 3, "Second");
insertlogbuffer(logBuffer, 3, 1, "Third");
insertlogbuffer(logBuffer, 4, 5, "Fourth");
printClogentry(logBuffer->oldest);
```

Output:

```
[ID: 2] [...] [severity: 3] Second
[ID: 3] [...] [severity: 1] Third
[ID: 4] [...] [severity: 5] Fourth
```



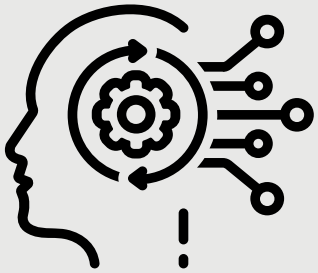
DetectCyclesintheList

Purpose: Detect Cycles in the List (validate log data consistency)

```
bool hasCycle = DetectCyclesintheList(logBuffer->oldest);
if (hasCycle) {
    printf("Circular structure is intact.\n");
}
```

Output:

```
Circular structure is intact.
```



04 Queue Log System

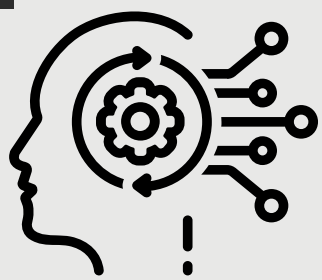
1. Overview

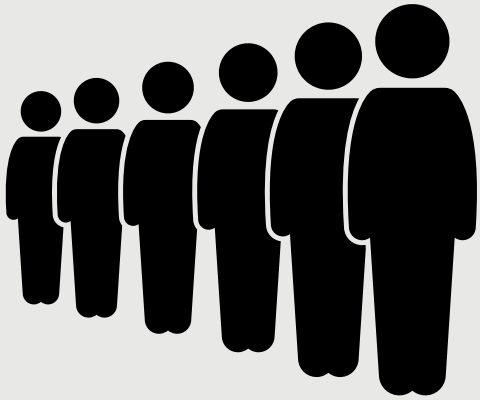
This C program demonstrates basic queue operations to manage log data in First-In-First-Out (FIFO) order. It includes functionality to enqueue new logs, dequeue logs, peek at the front of the queue, and display the full queue.

2. Data Structure

```
typedef struct node{
    int value;
    struct node *next;
} node;

typedef struct queue{
    node *front;
    node *reare;
} queue;
```





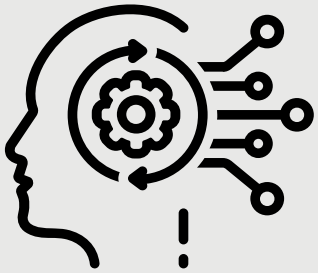
enqueue

Purpose: Inserts a new value at the rear of the queue.

```
enqueue(head, 10);
enqueue(head, 20);
print(head);
```

Output:

```
10-->20
```



dequeue

Purpose: Removes and prints the value at the front of the queue.

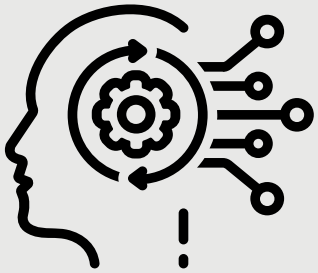
```
dequeue(head);
```

Output:

10

Queue becomes:

20



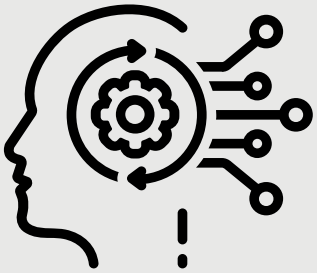
peek

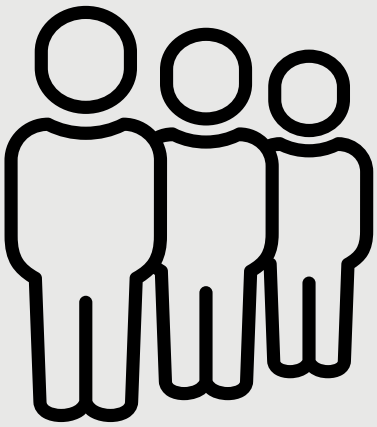
Purpose: Returns the value at the front without removing it.

```
int front = peek(head);
printf("%d", front);
```

Output:

```
20
```





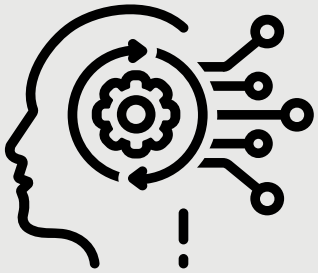
is empty

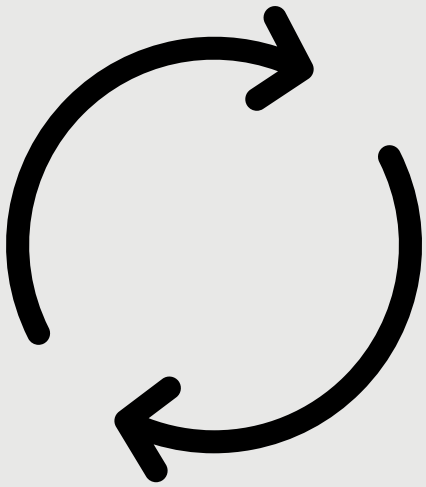
Purpose: Check if Queue is Empty or Full.

```
if (isEmpty(head)) printf("Queue is empty\n");
```

Output:

```
Queue is empty
```





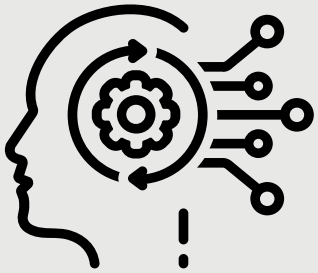
Circular Queue Implementation

Purpose:

To improve space efficiency and continuous buffer behavior, the queue can be extended to a circular queue:

- Instead of setting reare->next = NULL, it can be set to front.
- When reare->next == front, the queue is full.

Circular queues are useful for fixed-size buffering scenarios, such as implementing a ring buffer in embedded systems.



05

stacks

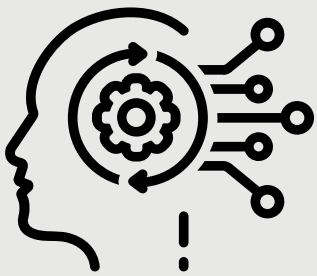
1. Overview

This C program implements a stack-based log system using dynamic memory. It supports pushing new log entries, popping from the stack, peeking at the top log, checking for emptiness, and reversing the entire stack. Each log entry includes a timestamp, ID, severity, and message.

2. Data Structure

```c
typedef struct logentry {
    int id, severity;
    char message[256];
    char timestamp[40];
    struct logentry *next;
} logentry, *Plogentry;

typedef struct STACK {
    Plogentry top;
} *stack;
```



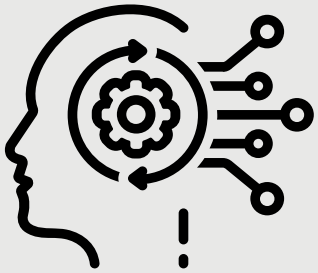
isemptystack

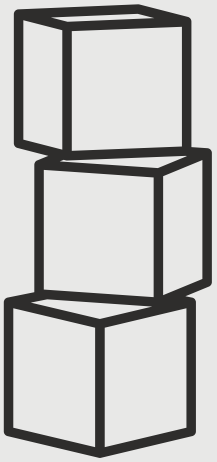
Purpose: Checks if the stack is empty.

```
if (isemptystack(S)) printf("Empty\n");
```

Output:

```
Empty
```





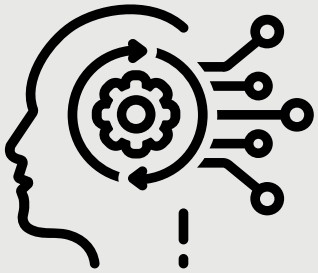
poplogentry

Purpose: Removes and returns the top log from the stack.

```
Plogentry popped = poplogentry(S);
printf("%d", popped->id);
```

Output:

```
5
```



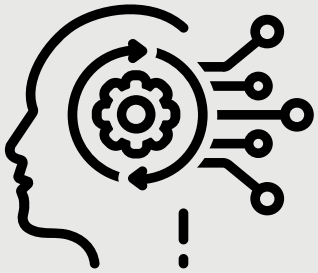
peeklogentry

Purpose: Returns a copy of the top log
without removing it.

```
logentry top = peeklogentry(S);
printf("%d", top.id);
```

Output:

```
5
```



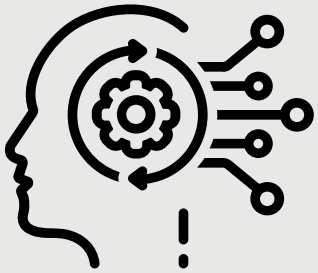
ReverseaStack

Purpose: Reverses the stack order.

```
ReverseaStack(S); 1->2->3->4
```

Output:

```
After 4->3->2->1
```



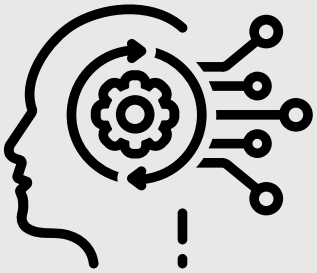
PushNewLogEntry

Purpose: Creates a new log and pushes it onto the stack.

```
PushNewLogEntry(S, 1, 3, "Initialized system");
2->3->4
```

Output:

```
after push (1)  2->3->4->1
```



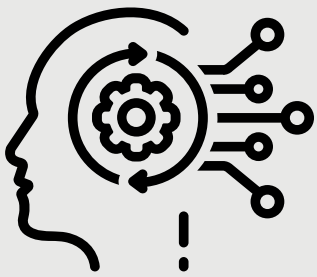
06 Recursion and Mixed Functions

1. Overview

This module demonstrates how recursion and other utility functions can be used for data structure processing. It includes operations on linked lists, arrays, and factorials using recursive techniques.

2. Data Structure

```
typedef struct node {
    int value;
    struct node *next;
} node;
```



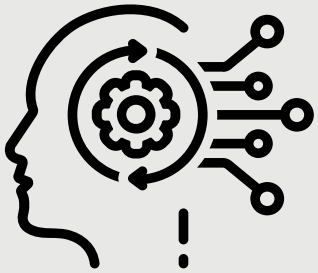
reverse_lls

Purpose: Recursively reverses a linked list.

```
node* head = create_sample_list();
head = reverse_lls(head);
```

Output:

```
Original: 1 -> 2 -> 3 -> NULL
Reversed: 3 -> 2 -> 1 -> NULL
```



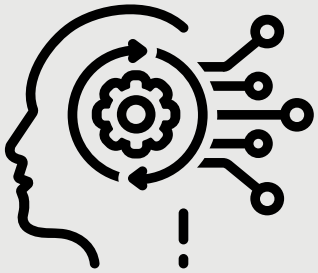
Factorial

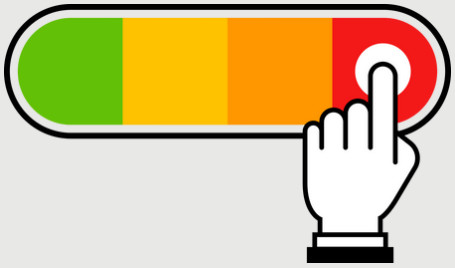
Purpose: Calculates factorial of an integer recursively.

```
int result = fact(5);
printf("%d", result);
```

Output:

```
120
```





find max_ID

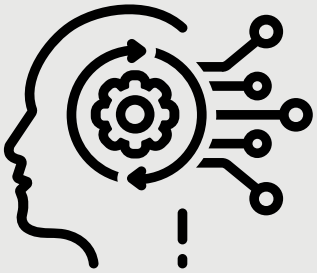
max_id

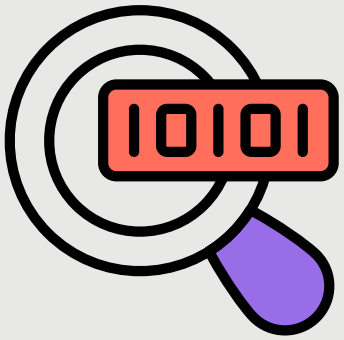
Purpose: Finds the maximum value in a linked list recursively.

```
node* head = create_sample_list();
int max = max_id(head);
printf("%d", max);
```

Output:

```
The largest value in list: 9
```





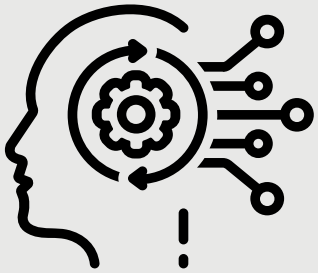
binary_serch(

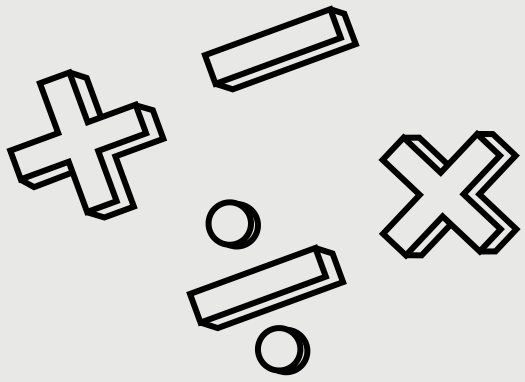
Purpose: Performs recursive binary search on an array.

```
int pos = binary_serch(sorted_arr, 0, 5, 8);
```

Output:

```
Element 8 found at index 3
```





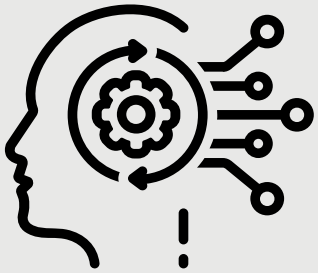
Infix_postfi

Purpose: Converts an infix expression to postfix using stack logic (prototype only).

```
char expr[] = "2+3*4";
infix_postfix(expr);
```

Output:

```
Postfix expression: 2 3 4 * +
```



07

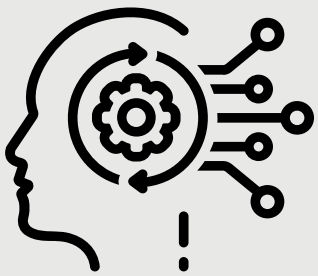
Binary Search Tree Log System

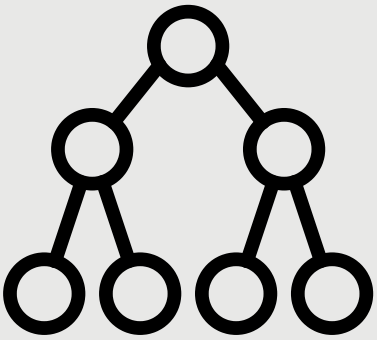
1. Overview

This C program provides a log management system using a Binary Search Tree (BST). It allows for inserting, deleting, searching, and traversing logs based on their timestamp. Logs are inserted in timestamp order, making retrieval and chronological ordering efficient.

2. Data Structure

```
typedef struct TREE {
    int id;
    int severity;
    char timestamp[40];
    char message[100];
    struct TREE *leftchild, *rightchild;
} TREE, *tree;
```





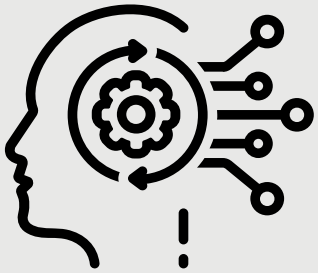
InsertLogintoBST

Purpose: Inserts a log into the BST based on timestamp.

```
tree root = NULL;
root = InsertLogintoBST(root, node);
```

Output:

```
Log inserted at correct timestamp position.
```



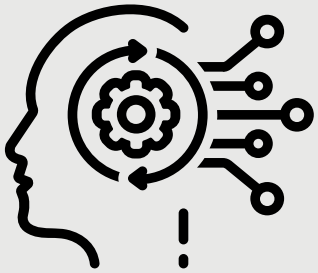
DeleteLogFromBST

Purpose: Removes a log with a given timestamp from the BST.

```
root = DeleteLogFromBST(root, "17_05_2025 14:32:00");
```

Output:

```
Log with timestamp deleted successfully.
```



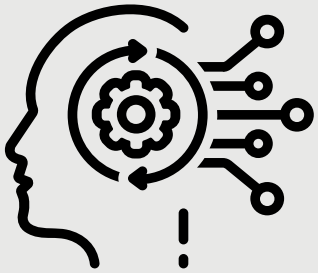
SearchLogInBST

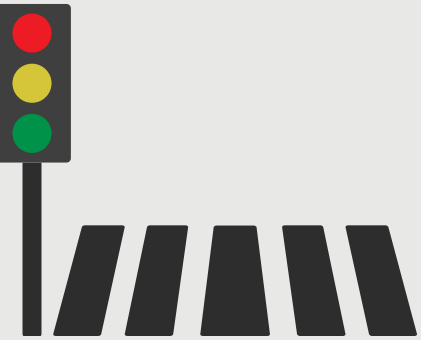
Purpose: Searches for a log by timestamp.

```
found = SearchLogInBST(root, "17_05_2025 14:32:00");
```

Output:

```
[ID: 04] [17_05_2025 14:32:00] [severity : 02] Log message
```





inorder / preorder / postorder Traversal

Purpose: Traversal functions to print the BST.

```
inorder(root);
preorder(root);
postorder(root);
```

Output:

```
[ID: 01] [16_05_2025 10:00:00] ...
[ID: 02] [17_05_2025 14:32:00] ...
```