ज्ञानेन प्रकाशते जगत्
INDUS UNIVERSITY

Data Structure and Algorithm

CALL LOGS USING STACK

Presented by-

Shubhangi Goswami
IU2141230273
CSE- D2 (Sem 4)

designed by freepik

What is Call Logs?

Call logs are a record of phone calls made or received by an individual or organization, and are an important tool for managing communication and keeping track of important information.

The program we'll be looking at allows users to enter call log information, including the caller's name, phone number, and call duration.

This information is then stored in a stack data structure, which allows the most recent call logs to be easily accessed and managed.

EXPLAINING STACK

- A stack is a fundamental data structure in computer science that allows data to be stored and retrieved in a ***Last-In, First-Out (LIFO) order.***

- A stack is a collection of elements that supports two main operations: push and pop.

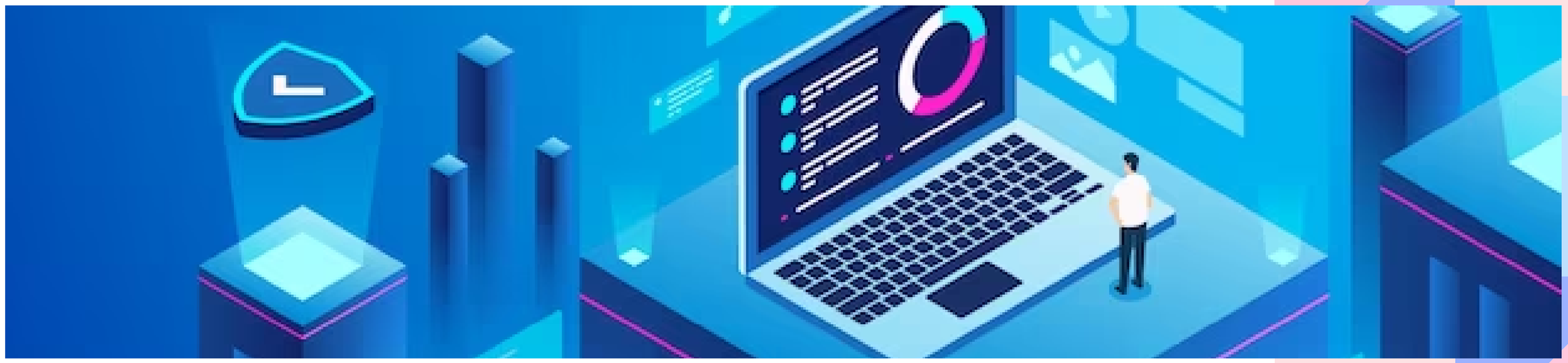
- A stack can be implemented using an array or a linked list. In the array implementation, we use a fixed-size array to store the stack elements.

OPERATION

1. **Peek:** This operation returns the value of the topmost element without removing it from the stack.
2. **Size:** This operation returns the number of elements in the stack.
3. **isEmpty:** This operation checks whether the stack is empty or not.

APPLICATIONS OF STACK:

Stacks are used in various computer algorithms and programming languages. For example, stacks are used in expression evaluation, recursion, and parsing. The call stack in programming languages is an example of a stack that is used to keep track of function calls and returns.



USING CALL LOGS IN STACK

- In computer programming, a stack is a data structure that stores and manages a collection of elements.
- The stack data structure follows the **Last-In-First-Out (LIFO**) principle, which means that the last item added to the stack will be the first one to be removed.

A stack is used in **call logs** to record the order of phone calls that have been placed or received. The specifics of the call (such as the phone number, duration, time, etc.) are added to the top of the stack each time a new call is placed or answered.

When a call is ended or terminated, the details of the **call are removed from the top of the stack**, and the next item in the stack becomes the new top. **This process continues until the stack is empty, meaning that all the calls in the log have been processed.**

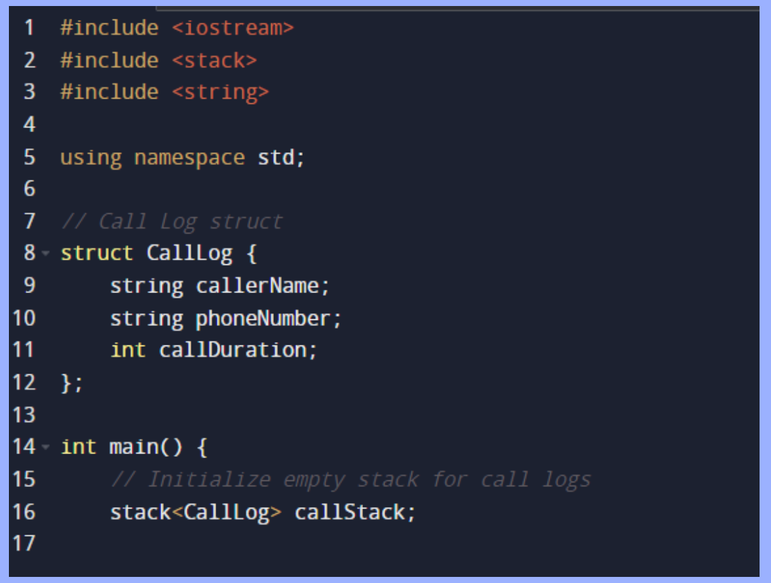
Using a stack to manage call logs allows for efficient tracking and retrieval of call data. The LIFO principle ensures that the most recent calls are always at the top of the stack and can be quickly accessed. Additionally, the stack data structure provides a way to easily add and remove call details as needed, making it a convenient and flexible tool for managing call logs.

CODE

```
#include <iostream>
#include <stack>
#include <string>

using namespace std;
// Call Log struct
struct CallLog {
    string callerName;
    string phoneNumber;
    int callDuration;
};

int main() {
    // Initialize empty stack for call logs
    stack<CallLog> callStack;
```



First, the program initializes an empty stack to hold call log information. Users are then prompted to enter the caller's name, phone number, and call duration for each call log.

CODE

```
// Prompt the user to enter call log information
  while (true) {
      CallLog newCall;
      cout << "Enter caller name (or 'q' to quit): ";
      getline(cin, newCall.callerName);
      if (newCall.callerName == "q") {
          break;
      }
      cout << "Enter phone number: ";
      getline(cin, newCall.phoneNumber);
      cout << "Enter call duration (in minutes): ";
      cin >> newCall.callDuration;
      cin.ignore();
// Ignore newline character after entering call
duration
      callStack.push(newCall);
  }
```

```
18      // Prompt the user to enter call log information
19      while (true) {
20          CallLog newCall;
21          cout << "Enter caller name (or 'q' to quit): ";
22          getline(cin, newCall.callerName);
23          if (newCall.callerName == "q") {
24              break;
25          }
26          cout << "Enter phone number: ";
27          getline(cin, newCall.phoneNumber);
28          cout << "Enter call duration (in minutes): ";
29          cin >> newCall.callDuration;
30          cin.ignore();  // Ignore newline character after
                entering call duration
31          callStack.push(newCall);
32      }
33
```

- **The program uses a while loop to repeatedly prompt the user to enter call log information.**
- **The getline function is used to read the caller name and phone number from the user, rather than cin, to ensure that the entire line of input is captured.**
- **Additionally, the cin.ignore() function is used after entering the call duration to ignore the newline character that is left in the input buffer.**

CODE

```
// Print the current call logs in the stack
   cout << "\nCurrent call logs in stack:\n";
   while (!callStack.empty()) {
      CallLog currentCall = callStack.top();
      cout << "Caller name: " <<
currentCall.callerName << "\n";
      cout << "Phone number: " <<
currentCall.phoneNumber << "\n";
      cout << "Call duration: " <<
currentCall.callDuration << " minutes\n";
      callStack.pop();
   }

   return 0;
}
```

```
34      // Print the current call logs in the stack
35      cout << "\nCurrent call logs in stack:\n";
36      while (!callStack.empty()) {
37          CallLog currentCall = callStack.top();
38          cout << "Caller name: " << currentCall.callerName <<
                "\n";
39          cout << "Phone number: " << currentCall.phoneNumber
                << "\n";
40          cout << "Call duration: " << currentCall
                .callDuration << " minutes\n";
41          callStack.pop();
42      }
43
44      return 0;
45  }
```

After all call logs have been entered, the program uses another while loop to iterate over the stack and print out each call log entry. The top function is used to access the most recent call log entry, and pop is used to remove it from the stack.

OUTPUT

Output

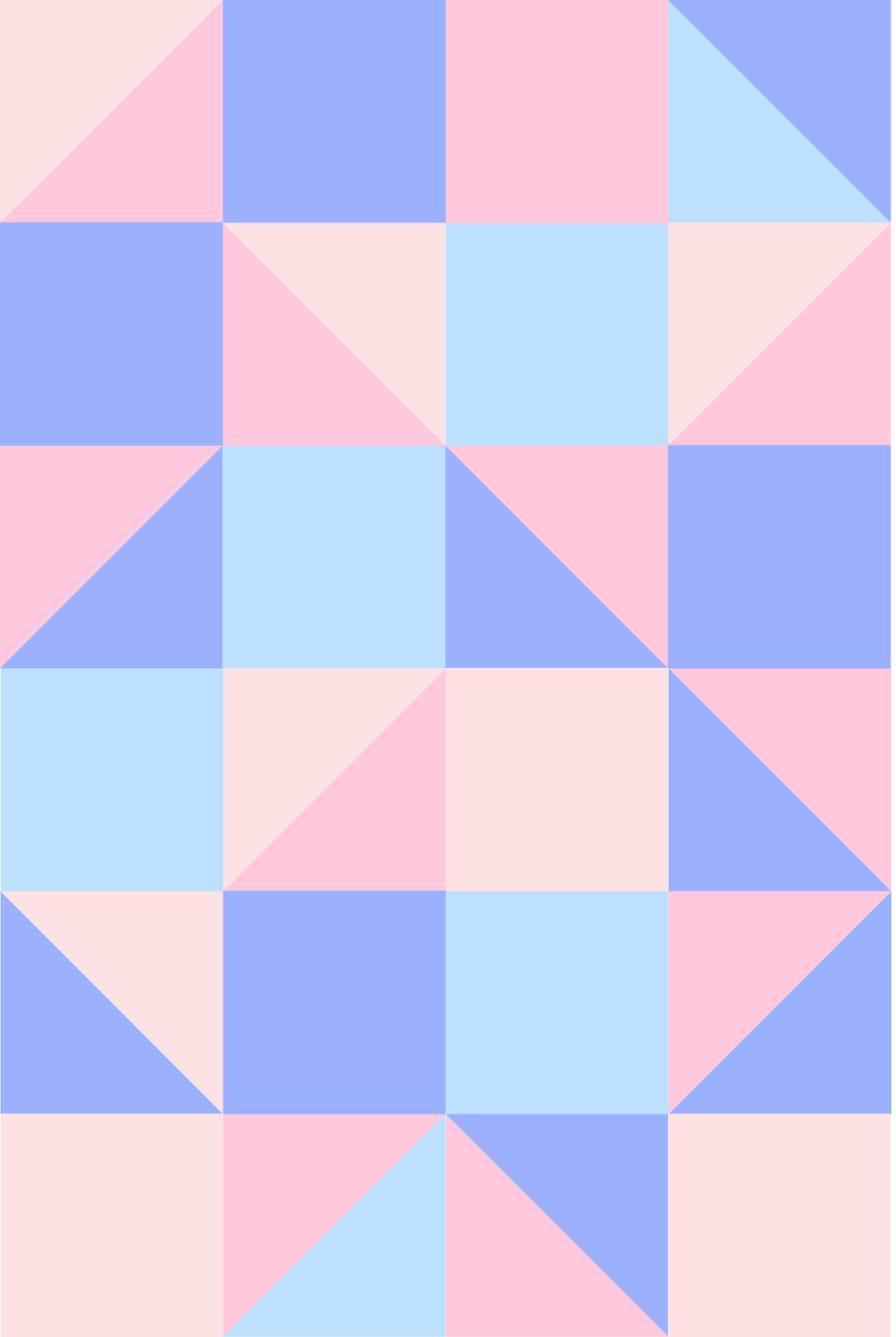
```
/tmp/dziaSWKdBS.o
Enter caller name (or 'q' to quit):
```

```
/tmp/dziaSWKdBS.o
Enter caller name (or 'q' to quit): Shubhangi Goswami
Enter phone number: 9081233357
Enter call duration (in minutes): 50
Enter caller name (or 'q' to quit): Nirali Parikh
Enter phone number: 8787878787
Enter call duration (in minutes): 90
Enter caller name (or 'q' to quit): Eva Mewada
Enter phone number: 9898786578
Enter call duration (in minutes): 10
Enter caller name (or 'q' to quit):
```

```
/tmp/dziaSWKdBS.o
Enter caller name (or 'q' to quit): Shubhangi Goswami
Enter phone number: 9081233357
Enter call duration (in minutes): 50
Enter caller name (or 'q' to quit): Nirali Parikh
Enter phone number: 8787878787
Enter call duration (in minutes): 90
Enter caller name (or 'q' to quit): Eva Mewada
Enter phone number: 9898786578
Enter call duration (in minutes): 10
Enter caller name (or 'q' to quit): q
Current call logs in stack:
Caller name: Eva Mewada
Phone number: 9898786578
Call duration: 10 minutes
Caller name: Nirali Parikh
Phone number: 8787878787
Call duration: 90 minutes
Caller name: Shubhangi Goswami
Phone number: 9081233357
Call duration: 50 minutes
```

CONCLUSION:

- This program provides a simple and effective way to manage call logs using the stack data structure.
- By allowing users to easily enter and access call log information, this program can be a valuable tool for individuals and organizations who need to keep track of their phone communications.
- Its Last-In-First-Out (LIFO) principle makes it an ideal choice for certain algorithmic problems.
- Understanding how to implement a stack and use it effectively can greatly enhance your programming skills.



THANK YOU