SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade

Kuniamuthur, Coimbatore – 641008

Phone : (0422)-2678001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CYBER SECURITY

23CS201 – DATA STRUCTURES AND ALGORITHMS LAB

CONTINUOUS ASSESSMENT RECORD

Submitted by

Name:……………………….…………….

Register Number:..

Degree & Branch:…………………………….

Class & Semester:…………………………….

Academic Year:…………………………

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade
Kuniamuthur, Coimbatore – 641008
Phone : (0422)-2678001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CYBER SECURITY

23CS201 – DATA STRUCTURES AND ALGORITHMS LAB

Continuous Assessment

Record Submitted by

Name: ……..…….…………………….………… **Register Number:**............……………………

Class/Semester:……………………….….……… **Degree& Branch:**…………….…………………

BONAFIDE CERTIFICATE

This is to certify that this record is the bonafide record of work done by Mr./Ms.

---

during the academic year 2024 - 2025.

Faculty In-charge **Head of the Department**

Submitted for the University practical examination held on______________________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade

Kuniamuthur, Coimbatore – 641008

Phone : (0422)-2678001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

23CS201 – DATA STRUCTURES AND ALGORITHMS

LAB

Record of laboratory work

EVEN SEMESTER (2024 - 2025)

Name of the Faculty	**Ms. Alis**

CONTINUOUS EVALUATION SHEET REFERENCES

RUBRICS TABLE

Rubrics for Evaluating Laboratory

Subject Code : 23CS201

Lab Name : Data Structures and Algorithms

Method: Lab Reports and Observation of

Faculty In charge Outcomes Assessed:

a) Graduates will demonstrate knowledge of mathematical, scientific and multidisciplinary approach for problem solving.

b) Graduates will be able to apply their knowledge in various programming skills to create solutions for product based and application based software.

c) Graduates will possess the ability to create real time solutions for different projects by using modern tools prevailing in the current trends.

d) Graduates attain advanced knowledge in the stream of Computer Science and Engineering to develop and maintain the simple and complex information systems.

INDEX- LIST OF PROGRAMS

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade
Kuniamuthur, Coimbatore – 641008
Phone : (0422)-2678001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CYBER SECURITY

EVEN SEMESTER: 2024-2025

Reg No:

Name of the Student:

Name of the Lab:

Components	**Exp No and Date**											
	Ex-1	**Ex-2**	**Ex-3**	**Ex-4**	**Ex-5**	**Ex-6**	**Ex-7**	**Ex-8**	**Ex-9**	**Ex-10**	**Ex 11**	**Ex 12**
Aim & Algorithm 20 Marks												
Coding 30 Marks												
Compilation & Debugging 30 Marks												
Execution & Results 10 Marks												
Documentation & Viva 10 Marks												
Total												

Staff In-charge

Ex No: 1	Implementation of Singly, Doubly and Circular Linked List.

Write a CPP program to create a singly linked list, insert a node at the corresponding place, delete a node at the given place and display the list.

Aim:

To implement a singly linked list with the following functionalities:

1. Insertion: Insert a new node at a specified position.
2. Deletion: Delete a node at a given position.
3. Display: Traverse and print the elements of the linked list.

Algorithm:

1. **Node Structure:**
   - Define a structure/class to represent a node in the linked list.
   - Include data (e.g., integer) and a pointer to the next node.
2. **Create Linked List:**
   - Read the number of nodes (N) from the input.
   - Create an empty head node.
   - For each node (from 0 to N-1):
     - Read data and position from input.
     - Create a new node with the given data.
     - Insert the new node at the specified position.
3. **Insertion:**
   - Read the data and position for the new node from input.
   - Create a new node with the given data.
   - If the position is 0 (insertion at the beginning):
     - Make the new node the head and update the next pointer.
   - Otherwise:
     - Traverse the list to find the node before the insertion position.

        - Adjust the pointers to insert the new node.

4. **Deletion:**
    - Read the position of the node to be deleted from input.
    - If the position is 0 (deletion at the beginning):
        - Update the head to point to the next node.
    - Otherwise:
        - Traverse the list to find the node before the node to be deleted.
        - Adjust the pointers to bypass the node to be deleted.
5. **Display:**
    - Initialize a temporary pointer to the head.
    - While the temporary pointer is not null:
        - Print the data of the current node.
        - Move the temporary pointer to the next node

PROGRAM:

```
#include <iostream>
using namespace std;
struct Node {
int data;
Node* next;
Node(int val) : data(val), next(nullptr) {}
};
class LinkedList {
public:
Node* head;
LinkedList() : head(nullptr) {}
```

```
void insert(int data, int pos) {

cout << "Inserting " << data << " at position " << pos << endl;

Node* newNode = new Node(data);

if (pos == 0) {

newNode->next = head;

head = newNode;

return;

}

Node* temp = head;for (int i = 0; temp != nullptr && i < pos - 1; i++) {

temp = temp->next;

}

if (temp == nullptr) {

cout << "Invalid position " << pos << endl;

return;

}

newNode->next = temp->next;

temp->next = newNode;

}

void remove(int pos) {

cout << "Removing node at position " << pos << endl;

if (head == nullptr) {

cout << "List is empty. Nothing to remove." << endl;

return;

}

Node* temp = head;

if (pos == 0) {

head = head->next;

cout << "Deleted node with value " << temp->data << endl;

delete temp;
return;
}
Node* prev = nullptr;
for (int i = 0; temp != nullptr && i < pos; i++) {
prev = temp;
temp = temp->next;
}
if (temp == nullptr) {
cout << "Invalid position " << pos << endl;
return;
}
prev->next = temp->next;
cout << "Deleted node with value " << temp->data << endl;
delete temp;
}
void display() {
cout << "Current List: ";
Node* temp = head;
while (temp) {
cout << temp->data << " ";
temp = temp->next;
}
cout << endl;
}
};
int main() {
LinkedList list;
int N, data, pos;
cin >> N;
```

```
cout << "Creating list with " << N << " nodes" << endl;
for (int i = 0; i < N; i++) {
cin >> data >> pos;
list.insert(data, pos);
}list.display();
cout<<"position to remove: ";cin >> pos;
list.remove(pos);
list.display();
return 0;
}
```

OUTPUT:

```
3
Creating list with 3 nodes
10 0
Inserting 10 at position 0
20 1
Inserting 20 at position 1
30 2
Inserting 30 at position 2
Current List: 10 20 30
position to remove: 2
Removing node at position 2
Deleted node with value 30
Current List: 10 20
```

Result:

The program successfully implements a singly linked list with the functionalities of insertion, deletion, and display. It demonstrates the dynamic nature of linked lists by allowing modifications to the structure.

Ex No: 2	Implementation of Stack using Arrays.

Palindrome Using Stack
Write a program to check whether the string is palindrome or not using Stack.
Example
Input
malayalam
Output
malayalam is palindrome
Explanation
Push the string into the stack, then invert the string by popping the characters. Compare the two strings and print a palindrome if they are the same otherwise not a palindrome.

Aim:

To determine whether a given string is a palindrome or not using a stack data structure.

Algorithm:

1. Create an empty stack.
2. Push each character of the input string onto the stack.
3. Create an empty string to store the reversed characters.
4. While the stack is not empty:
    i. 4.1. Pop the top character from the stack.
    ii. 4.2. Append the popped character to the reversed string.
5. Compare the original string with the reversed string.
6. If the original string is equal to the reversed string, it is a palindrome.
7. Otherwise, it is not a palindrome.
8. Print the result.

PROGRAM:

```
#include <iostream>
#include <stack>
using namespace std;
bool isPalindrome(string str) {
stack<char> s;
for (char ch : str) {
s.push(ch);
}
string reversedStr;
while (!s.empty()) {
reversedStr += s.top();
s.pop();
}
cout << "Original String: " << str << endl;
cout << "Reversed String: " << reversedStr << endl;
return str == reversedStr;
}
int main() {
string input;
cin >> input;
if (isPalindrome(input)) {
cout << input << " is palindrome" << endl;
} else {
cout << input << " is not palindrome" << endl;
}
return 0;
}
```

OUTPUT:

```
malayalam
Original String: malayalam
Reversed String: malayalam
malayalam is palindrome
```

RESULT:

The program successfully checks if a given string is a palindrome or not using a stack.
It efficiently reverses the string using the stack's LIFO (Last-In, First-Out) property and compares it with the original string to determine the result.

Ex No: 3	Implementation of Stack using Linked List.

Write a code to implement stack using linked list and perform push and pop using underflow and overflow condition. The pop is done only once after the elements are put into the stack.

Aim:

To implement a stack using a linked list data structure, incorporating push and pop operations with underflow and overflow conditions.

Algorithm:

1. **Node Structure:**
    - Define a structure named Node to represent a node in the linked list.
    - Each node has two members: data (to store the integer value) and next (a pointer to the next node in the list).
2. **Stack Structure:**
    - Define a structure named Stack to represent the stack.
    - The Stack structure has a single member: top (a pointer to the top node of the stack).
3. **Push Operation:**
    - Create a new node with the given data.
    - If the stack is empty (top is null):
        - Set the top of the stack to point to the new node.
    - Otherwise:
        - Set the next pointer of the new node to point to the current top node.
        - Update the top of the stack to point to the new node.
4. **Pop Operation:**
    - If the stack is empty (top is null):
        - Handle underflow condition (e.g., print an error message).
    - Otherwise:
        - Create a temporary pointer to store the current top node.
        - Update the top of the stack to point to the next node.
        - Delete the temporary node.
5. **Display Stack:**
    - Create a temporary pointer to traverse the stack from top to bottom.
    - While the temporary pointer is not null:
        - Print the data of the current node.
        - Move the temporary pointer to the next node.

PROGRAM:

```
#include <iostream>
using namespace std;
struct Node {
int data;
Node* next;
Node(int val) : data(val), next(nullptr) {}
};
struct Stack {
Node* top;
int maxSize;
int currentSize;
};
void initialize(Stack &s, int size) {
cout << "Initializing stack with size " << size << endl;
s.top = nullptr;
s.maxSize = size;
s.currentSize = 0;
}
void push(Stack &s, int data) {
cout << "Attempting to push " << data << endl;
if (s.currentSize >= s.maxSize) {
cout << "Stack Overflow! Cannot push " << data << endl;
return;
}
Node* newNode = new Node(data);
newNode->next = s.top;
s.top = newNode;
s.currentSize++;
cout << "Pushed: " << data << ", Current Stack Size: " << s.currentSize << endl;
}
void pop(Stack &s) {
cout << "Attempting to pop from stack" << endl;
if (s.top == nullptr) {
cout << "Stack Underflow! Cannot pop." << endl;
return;
}
Node* temp = s.top;
s.top = s.top->next;
cout << "Popped: " << temp->data << ", Current Stack Size: " << s.currentSize - 1 << endl;
delete temp;
s.currentSize--;
}
void display(Stack &s) {
cout << "Displaying stack contents" << endl;
if (s.top == nullptr) {
cout << "Stack is empty." << endl;
return;
}
Node* temp = s.top;
cout << "Stack elements: ";
while (temp) {cout << temp->data << " ";
temp = temp->next;
}
cout << endl;
}
int main() {
int size, num;
cin >> size;
Stack s;
initialize(s, size);
for (int i = 0; i < size; i++) {
cin >> num;
push(s, num);
}
display(s);
pop(s);
display(s);
return 0;
}
```

OUTPUT:

```
3
Initializing stack with size 3
10 20 30
Attempting to push 10
Pushed: 10, Current Stack Size: 1
Attempting to push 20
Pushed: 20, Current Stack Size: 2
Attempting to push 30
Pushed: 30, Current Stack Size: 3
Displaying stack contents
Stack elements: 30 20 10
Attempting to pop from stack
Popped: 30, Current Stack Size: 2
Displaying stack contents
Stack elements: 20 10
```

RESULT:

The program successfully implements a stack using a linked list data structure. It includes push and pop operations with underflow and overflow handling. The stack can be used to store and retrieve data in a manner.

EX NO: 4	Implementation of Stack applications

Write a program to check if a given string of parentheses is balanced or not using a stack.

Note :

The input string contains only parentheses ('(', ')', '{', '}', '[', and ']') and no other characters.

Note : Refer the sample output**.**

Sample Input

{()}[]

Sample Output

The string {()}[] is balanced.

Sample Input

{[}]

Sample Output

The string {[}] is not balanced.

Aim:

To determine if a given string of parentheses is balanced using a stack data structure.

Algorithm:

1. Initialize an empty stack.
2. Iterate through each character in the input string:
   - If the character is an opening bracket ((or { or [):
     - Push the character onto the stack.
   - If the character is a closing bracket () or } or]):
     - If the stack is empty:
       - The string is not balanced.
     - Pop the top element from the stack.
     - If the popped element is not the corresponding opening bracket for the current closing bracket:
       - The string is not balanced.
3. After iterating through the entire string:
   - If the stack is empty:
     - The string is balanced.
   - Otherwise:
     - The string is not balanced.
4. Print the result:
   - If the string is balanced, print "The string <string> is balanced."
   - Otherwise, print "The string <string> is not balanced."

PROGRAM:

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
bool isBalanced(const string& str) {
stack<char> s;
for (char ch : str) {
if (ch == '(' || ch == '{' || ch == '[') {
s.push(ch);
} else if (ch == ')' || ch == '}' || ch == ']') {
if (s.empty()) return false;
char top = s.top();
s.pop();
if ((ch == ')' && top != '(') || (ch == '}' && top != '{') || (ch == ']' && top != '[')) {
return false;
}
}
}
return s.empty();
}
int main() {
string input;
cin >> input;
if (isBalanced(input)) {
cout << "The string " << input << " is balanced." << endl;
} else {
cout << "The string " << input << " is not balanced." << endl;
}
return 0;
}
```

OUTPUT 1:

```
{{}
The string {{} is not balanced.
```

OUTPUT 2:

```
{}
The string {} is balanced.
```

RESULT:

The program successfully checks if a given string of parentheses is balanced or not using a stack. It efficiently determines the balanced state of the parentheses string by utilizing the stack's LIFO (Last-In, First-Out) property to match opening and closing brackets.

EX NO: 5	Implementation of Queue using Arrays.

Implement two queues in an array:
Write a program to insert and delete elements from both queues.
Note: n is always 10 (5 each)

Aim:
To implement two queues within a single array, each with a capacity of 5 elements, and perform enqueue and dequeue operations on both queues efficiently.

Algorithm:

1. **Data Structure:**
   - Create an array of size 10 to store the elements of both queues.
   - Define variables to track the front and rear indices of each queue within the array.
2. **Enqueue Operation:**
   - Check for queue overflow.
   - If the queue is not full:
     - Update the rear index of the respective queue.
     - Insert the element at the rear position in the array.
3. **Dequeue Operation:**
   - Check for queue underflow.
   - If the queue is not empty:
     - Retrieve the element at the front position.
     - Update the front index of the respective queue.
4. Input:
   - Read the elements for queue 1 from the input.
   - Enqueue each element into queue 1.
   - Read the elements for queue 2 from the input.
   - Enqueue each element into queue 2.
5. **Output:**
   - Dequeue and print the front element from queue 1.
   - Dequeue and print the front element from queue 2.

PROGRAM:

```
#include <iostream>
using namespace std;

const int n = 10;

struct TwoQueues {
   int arr[n];
   int front1, rear1, front2, rear2;

   TwoQueues() {
      front1 = 0, rear1 = -1;
      front2 = 5, rear2 = 4;
   }

   void enqueue1(int x) {
      if (rear1 == 4) {
         cout << "Queue 1 Overflow\n";
         return;
      }
      arr[++rear1] = x;
   }

   void enqueue2(int x) {
      if (rear2 == 9) {
         cout << "Queue 2 Overflow\n";
         return;
      }
      arr[++rear2] = x;
   }

   void dequeue1() {
      if (front1 > rear1) {
         cout << "Queue 1 Underflow\n";
         return;
      }
      cout << arr[front1++] << endl;
   }

   void dequeue2() {
      if (front2 > rear2) {
         cout << "Queue 2 Underflow\n";
         return;
      }
      cout << arr[front2++] << endl;
   }
};

int main() {
   TwoQueues q;
   int x;

   cout << "Enter 5 elements for Queue 1:\n";
   for (int i = 0; i < 5; i++) {
      cin >> x;
      q.enqueue1(x);
   }
```

```
    cout << "Enter 5 elements for Queue 2:\n";
    for (int i = 0; i < 5; i++) {
        cin >> x;
        q.enqueue2(x);
    }

    cout << "Dequeued from Queue 1: ";
    q.dequeue1();

    cout << "Dequeued from Queue 2: ";
    q.dequeue2();

    return 0;
}
```

OUTPUT:

```
Enter 5 elements for Queue 1:
11 12 13 14 15
Enter 5 elements for Queue 2:
21 22 23 24 25
Dequeued from Queue 1: 11
Dequeued from Queue 2: 21
```

RESULT:

The program successfully implements two queues within a single array, allowing for efficient insertion and deletion of elements from both queues. It correctly handles queue overflow, and underflow conditions and produces the expected output based on the given input.