|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Data Structures Laboratory (using C/C++)** | **Semester:** | **III** |
| **Date of Performance:** |  | **Batch No:** | **A - 3** |
| **Faculty Name:** | **Prof. Om Goswami** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 3**

**Title: Application of Stack**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| **To understand stack operation.**  **Write a program for stack using arrays. Given A [] = {21,34,45,21,60}, perform Push, Pop operations and Display Stack contents.** |

|  |
| --- |
| **COs to be achieved:** |
| **CO1:** Understand and implement the different data structures used in problem solving  **CO2:** Apply linear and non-linear data structure in application development |

|  |
| --- |
| **Books/Journals/Websites referred:** |
| 1. Data Structures by Reema Thareja 2. [Array Implementation of Stack in Data Structure - javatpoint](https://www.javatpoint.com/ds-array-implementation-of-stack) |

|  |
| --- |
| **Tools required:** |
| DEV C/C++ compiler/ Code blocks C compiler/VS Code Python Compiler |

|  |
| --- |
| **Theory:** |
| A stack is a linear data structure in which insertion and deletion of elements are done at only one end, which is known as the top of the stack. Stack is called a last-in, first-out (LIFO) structure because the last element which is added to the stack is the first element which is deleted from the stack. |

|  |
| --- |
| **Implementation details:** |
| 1. **Enlist all the Steps followed and various options explored.**  * The program defines an array-based stack with a fixed maximum size. * It provides three main operations: push, pop, and display. * The program uses a menu-driven approach to allow the user to interact with the stack. * The program handles stack overflow and underflow conditions. * The logic of the program is implemented using switch-case statements. * The program includes a loop to continuously prompt the user for their choice until they choose to exit.  1. **Explain your program logic and methods used.**  * The program uses an array stack to store the elements of the stack. * The variable top is used to keep track of the index of the top element in the stack. * The push function adds an element to the stack by incrementing top and assigning the value to stack[top]. * The pop function removes the top element from the stack by decrementing top. * The display function prints the contents of the stack by iterating from top to 0 and printing stack[i]. * The main function implements the menu-driven approach using a while loop and switch-case statements to handle user choices.  1. **Explain the Importance of the approach followed by you.**  * The array-based implementation of a stack is simple and efficient. * The use of a menu-driven approach allows users to interact with the stack easily. * The program handles stack overflow and underflow conditions, providing informative messages to the user. * The program uses a loop to continuously prompt the user for their choice until they choose to exit, ensuring a user-friendly experience. * The code is well-structured and easy to understand, making it suitable for learning and educational purposes. |

|  |
| --- |
| **C/C++ Code implemented:** |
| #include <stdio.h>  #define MAX\_SIZE 5  int stack[MAX\_SIZE];  int top = -1;  // push  void push(int element) {      if (top >= MAX\_SIZE - 1) {          printf("\nStack Overflow! Cannot push element %d.\n", element);      } else {          stack[++top] = element;          printf("Pushed %d onto the stack.\n", element);      }  }  // pop  void pop() {      if (top == -1) {          printf("Stack is empty. Cannot pop.\n");      } else {          printf("Popped %d from the stack.\n", stack[top--]);      }  }  // display  void display() {      if (top == -1) {          printf("\nStack is empty.\n\n");      } else {          printf("\nStack contents: ");          for (int i = 0; i <= top; i++) {              printf("%d ", stack[i]);          }          printf("\n");      }  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050\n\n");        push(21);      push(34);      push(45);      push(21);      push(60);      push(100);        display();        pop();      pop();      pop();      pop();      pop();      pop();        display();        push(21);      push(34);      push(45);      push(21);      push(60);        display();      return 0;  } |

|  |
| --- |
| **Output/ program results after execution:** |
|  |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| Write a program to evaluate postfix expressions using stack in C/C++.  #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 100  int stack[MAX\_SIZE];  int top = -1;  // push  void push(int item) {      if (top >= MAX\_SIZE - 1) {          printf("Stack Overflow\n");          return;      }      top++;      stack[top] = item;  }  // pop  int pop() {      if (top < 0) {          printf("Stack Underflow\n");          return -1;      }      int item = stack[top];      top--;      return item;  }  // check if character is operator  int is\_operator(char symbol) {      return (symbol == '+' || symbol == '-' || symbol == '\*' || symbol == '/');  }  // evaluate expression  int evaluate(char\* expression) {      int i = 0;      char symbol = expression[i];      int operand1, operand2, result;      while (symbol != '\0') {          if (symbol >= '0' && symbol <= '9') {              int num = symbol - '0';              push(num);          } else if (is\_operator(symbol)) {              operand2 = pop();              operand1 = pop();              switch (symbol) {                  case '+':                      result = operand1 + operand2;                      break;                  case '-':                      result = operand1 - operand2;                      break;                  case '\*':                      result = operand1 \* operand2;                      break;                  case '/':                      if (operand2 != 0) {                          result = operand1 / operand2;                      } else {                          printf("Division by zero\n");                          return -1;                      }                      break;              }              push(result);          }          i++;          symbol = expression[i];      }      result = pop();      return result;  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050\n");      char expression[] = "5 6 7 + \* 8 -";      printf("\nExpression: %s\n", expression);      int result = evaluate(expression);      printf("Result = %d\n", result);      return 0;  } |

|  |
| --- |
| **Conclusion:** |
| In conclusion, the experiment involved understanding stack operations and implementing a program for a stack using arrays. The implementation followed the LIFO (Last In First Out) approach, where the last element added is the first one to be removed. The program handled stack overflow and underflow conditions, providing informative messages. This approach provided a simple and efficient way to implement a stack using arrays in C, allowing for easy manipulation and display of stack elements. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |