|  |  |  |  |
| --- | --- | --- | --- |
| **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: 2**

**Title: Linked List**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| **To understand basics of linked list and its operations.**  **Write a program to -**   1. **Create a singly linked list of integers. 97, 53, 367, 76, 121, 10.** 2. **Insert an integer 32 in the beginning of the linked list.** 3. **Delete any value specified by the user in the above linked list.** 4. **Display the contents of the above list after Deletion.** 5. **Search any value specified by the user in the linked list.** |

|  |
| --- |
| **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. [Linked List C/C++ Programs - GeeksforGeeks](https://www.geeksforgeeks.org/c-programs-gq/linked-list-programs-gq/) |

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

|  |
| --- |
| **Theory:** |
| A linear list is a list where each element has a unique successor. There are four common operations associated with a linear list:   * Insertion * Deletion * Retrieval * Traversal   Linear lists can be divided into two categories:   1. general list 2. restricted list   In a general list the data can be inserted or deleted without any restriction whereas in a restricted list there are restrictions for these operations. Linked lists and arrays are commonly used to implement general linear lists. A linked list is simply a chain of structures which contain a pointer to the next element. It is dynamic in nature. Items may be added to it or deleted from it at will.  400px-C_language_linked_list  A list item has a pointer to the next element, or to NULL if the current element is the tail (end of the list). This pointer points to a structure of the same type as itself. This structure that contains elements and pointers to the next structure is called a Node. |

|  |
| --- |
| **Implementation details:** |
| 1. **Enlist all the Steps followed and various options explored.**   Steps Followed:   * Define a structure for a node in the linked list. * Create a function to create a new node with a given data value. * Create a function to insert a node at the beginning of the linked list. * Create a function to delete a node with a specific value from the linked list. * Create a function to search for a value in the linked list. * Create a function to display the contents of the linked list. * In the main function, create an initial linked list with some values. * Display the initial linked list. * Insert a value at the beginning of the linked list. * Display the linked list after insertion. * Prompt the user to enter a value to delete from the linked list. * Delete the specified value from the linked list. * Display the linked list after deletion. * Prompt the user to enter a value to search in the linked list. * Search for the specified value in the linked list. * Free the memory by deleting all remaining nodes in the linked list.   Options Explored:   * The program uses a singly linked list, but it could be modified to use a doubly linked list if needed. * The program allows the user to insert a value at the beginning of the linked list, but insertion at other positions could be implemented. * Deletion is performed based on the value provided by the user. It could be extended to delete nodes at a specific position.  1. **Explain your program logic and methods used.**   The program implements a basic singly linked list with the following key functions:   * createNode: This function creates a new node with a given data value and returns a pointer to the newly created node. * insertAtBeginning: This function inserts a new node with a specified data value at the beginning of the linked list. It updates the head of the list to point to the new node. * deleteNode: This function deletes a node with a specific value from the linked list. It traverses the list to find the node with the specified value and adjusts the pointers of the previous and next nodes accordingly. * searchValue: This function searches for a value in the linked list and returns whether the value was found or not. It also prints the position of the value if found. * displayList: This function displays the contents of the linked list by traversing and printing each node's data.   In the main function, an initial linked list is created with some values. The program then demonstrates the insertion of a value at the beginning of the list, deletion of a specified value, and searching for a specified value. Finally, it frees the memory by deleting all nodes in the linked list.   1. **Explain the Importance of the approach followed by you.**   The approach followed in this program is important because it demonstrates the fundamental operations that can be performed on a singly linked list. These operations are building blocks for more complex data structures and algorithms. Understanding how to create, manipulate, and traverse linked lists is a fundamental skill for any programmer, especially when dealing with dynamic data structures.  Moreover, the program provides a clear and concise example of memory management in C, showing how to allocate and deallocate memory for nodes. This is crucial to prevent memory leaks and manage resources efficiently in programs.  Overall, this program serves as a foundational example of working with linked lists and can be a starting point for more advanced data structure implementations and algorithmic problem-solving. |

|  |
| --- |
| **C/C++ Code implemented:** |
| #include <stdio.h>  #include <stdlib.h>  // structure for a node in the linked list  struct Node {      int data;      struct Node\* next;  };  // function to create new node  struct Node\* createNode(int data) {      struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));      newNode->data = data;      newNode->next = NULL;      return newNode;  }  // function to insert node at beginning  struct Node\* insertAtBeginning(struct Node\* head, int data) {      struct Node\* newNode = createNode(data);      newNode->next = head;      return newNode;  }  // function to delete node  struct Node\* deleteNode(struct Node\* head, int key) {      struct Node\* current = head;      struct Node\* prev = NULL;      while (current != NULL && current->data != key) {          prev = current;          current = current->next;      }      if (current == NULL) {          printf("Value %d not found in the linked list.\n", key);          return head;      }      if (prev == NULL) {          // key first node --> update head          head = head->next;      } else {          // remove node with specified key          prev->next = current->next;      }      free(current);      printf("Value %d deleted from the linked list.\n", key);      return head;  }  // function to search for value  int searchValue(struct Node\* head, int key) {      struct Node\* current = head;      int position = 0;      while (current != NULL) {          if (current->data == key) {              printf("Value %d found at position %d in the linked list.\n", key, position);              return 1; // value found          }          current = current->next;          position++;      }      printf("Value %d not found in the linked list.\n", key);      return 0; // value not found  }  // function to display contents of list  void displayList(struct Node\* head) {      struct Node\* current = head;      printf("Linked List: ");      while (current != NULL) {          printf("%d ", current->data);          current = current->next;      }      printf("\n");  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050");      struct Node\* head = NULL;      int choice, value, searchValueToDelete, searchValueToSearch;      // adding numbers in linked list      head = insertAtBeginning(head, 97);      head = insertAtBeginning(head, 53);      head = insertAtBeginning(head, 367);      head = insertAtBeginning(head, 76);      head = insertAtBeginning(head, 121);      head = insertAtBeginning(head, 10);      // displaying list      printf("\nInitial ");      displayList(head);      // inserting 32 at beginning      head = insertAtBeginning(head, 32);      // displaying list      printf("Linked List after inserting 32 at the beginning: ");      displayList(head);      // deleting value      printf("Enter a value to delete from the linked list: ");      scanf("%d", &searchValueToDelete);      head = deleteNode(head, searchValueToDelete);      // displaying list      displayList(head);      // searching for value specified      printf("Enter a value to search in the linked list: ");      scanf("%d", &searchValueToSearch);      searchValue(head, searchValueToSearch);      // free memory      while (head != NULL) {          struct Node\* temp = head;          head = head->next;          free(temp);      }      return 0;  } |

|  |
| --- |
| **Output/ program results after execution:** |
| * **Value found in linked list:**      * **Value NOT found in linked list:** |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **Modify the above code for doubly LL/ circular LL/ circular doubly LL (any one).**   **Used doubly linked list.**  #include <stdio.h>  #include <stdlib.h>  // define structure for node in doubly linked list  struct Node {      int data;      struct Node\* prev;      struct Node\* next;  };  // create a new node  struct Node\* createNode(int data) {      struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));      newNode->data = data;      newNode->prev = NULL;      newNode->next = NULL;      return newNode;  }  // function to insert node  struct Node\* insertAtBeginning(struct Node\* head, int data) {      struct Node\* newNode = createNode(data);      newNode->next = head;      if (head != NULL) {          head->prev = newNode;      }      return newNode;  }  // function to delete node with specific value  struct Node\* deleteNode(struct Node\* head, int key) {      struct Node\* current = head;      while (current != NULL && current->data != key) {          current = current->next;      }      if (current == NULL) {          printf("Value %d not found in the linked list.\n", key);          return head;      }      if (current->prev != NULL) {          current->prev->next = current->next;      } else {          head = current->next;      }      if (current->next != NULL) {          current->next->prev = current->prev;      }      free(current);      printf("Value %d deleted from the linked list.\n", key);      return head;  }  // function to search for value  int searchValue(struct Node\* head, int key) {      struct Node\* current = head;      int position = 0;      while (current != NULL) {          if (current->data == key) {              printf("Value %d found at position %d in the linked list.\n", key, position);              return 1; // value found          }          current = current->next;          position++;      }      printf("Value %d not found in the linked list.\n", key);      return 0; // value not found  }  // function to display linked list  void displayList(struct Node\* head) {      struct Node\* current = head;      printf("\nLinked List: ");      while (current != NULL) {          printf("%d ", current->data);          current = current->next;      }      printf("\n");  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050\nUSING DOUBLY LINKED LIST");      struct Node\* head = NULL;      int choice, value, searchValueToDelete, searchValueToSearch;      // Creating the LL      head = insertAtBeginning(head, 97);      head = insertAtBeginning(head, 53);      head = insertAtBeginning(head, 367);      head = insertAtBeginning(head, 76);      head = insertAtBeginning(head, 121);      head = insertAtBeginning(head, 10);      // display initial LL      printf("\n\nInitial ");      displayList(head);      // inserting 32      head = insertAtBeginning(head, 32);      // displaying LL      printf("\nDoubly Linked List after inserting 32 at the beginning: ");      displayList(head);      // deleting value specified by user      printf("\nEnter a value to delete from the linked list: ");      scanf("%d", &searchValueToDelete);      head = deleteNode(head, searchValueToDelete);      // displaying LL      displayList(head);      // searching for value specified by user      printf("Enter a value to search in the linked list: ");      scanf("%d", &searchValueToSearch);      searchValue(head, searchValueToSearch);      // free memory      while (head != NULL) {          struct Node\* temp = head;          head = head->next;          free(temp);      }      return 0;  } |

|  |
| --- |
| **Conclusion:** |
| In conclusion, exploring and understanding these linked list methods has provided valuable insights into the core concepts of data structures and memory management in C. Learning how to create, manipulate, and traverse linked lists forms a strong foundation for tackling more complex data structures and algorithmic challenges. |

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