|  |  |
| --- | --- |
| DatA Structures  Laboratary (LCPIT-101) | |
| Practical file | |
| Submitted to: | Submitted by: | |
| Er. Parminder kaur wadhwa Assistant Professor (IT)  GNDEC, Ludhiana | Brahamjot SIngh CRN: 2221146  Class: D2IT  Section: B2 | |

Experiment-1: Program to insert a new element in an array:

1. **At end:**

* Program:

1. #include <iostream>
2. using namespace std;
3. int main() {
4. int originalArray[100]; // Assuming a maximum array size of 100
5. int n; // Number of elements in the original array
6. int newElement;
7. // Input the number of elements in the original array
8. cout << "Enter the number of elements in the array: ";
9. cin >> n;
10. // Input elements into the original array
11. cout << "Enter " << n << " elements:\n";
12. for (int i = 0; i < n; ++i) {
13. cin >> originalArray[i];
14. }
15. // Input the new element to be inserted
16. cout << "Enter the new element to insert: ";
17. cin >> newElement;
18. // Check if there is enough space in the original array
19. if (n >= 100) {
20. cout << "Error: Array is full. Cannot insert a new element.\n";
21. } else {
22. // Insert the new element at the end of the original array
23. originalArray[n] = newElement;
24. n++; // Increase the number of elements in the array
25. // Print the updated array
26. cout << "Array after insertion:\n";
27. for (int i = 0; i < n; ++i) {
28. cout << originalArray[i] << " ";
29. }
30. cout << "\n";
31. }
32. return 0;
33. }

* Output:

Output for Exp 1 (a)


1. **At a given position:**

* **Program:**

1. include <iostream>
2. using namespace std;
3. int main() {
4. int originalArray[100]; // Assuming a maximum array size of 100
5. int n; // Number of elements in the original array
6. int newPosition, newElement;
7. // Input the number of elements in the original array
8. cout << "Enter the number of elements in the array: ";
9. cin >> n;
10. // Input elements into the original array
11. cout << "Enter " << n << " elements:\n";
12. for (int i = 0; i < n; ++i) {
13. cin >> originalArray[i];
14. }
15. // Input the position at which the new element should be inserted
16. cout << "Enter the position to insert the new element (0-based index): ";
17. cin >> newPosition;
18. // Input the new element to be inserted
19. cout << "Enter the new element to insert: ";
20. cin >> newElement;
21. // Check if the position is valid
22. if (newPosition < 0 || newPosition > n) {
23. cout << "Error: Invalid position. Position should be between 0 and " << n << ".\n";
24. } else {
25. // Create a new array one element larger than the original
26. int newArray[101]; // Assuming a maximum array size of 100
27. // Copy elements from the original array to the new array up to the specified position
28. for (int i = 0; i < newPosition; ++i) {
29. newArray[i] = originalArray[i];
30. }
31. // Insert the new element at the specified position
32. newArray[newPosition] = newElement;
33. // Continue copying the remaining elements from the original array to the new array
34. for (int i = newPosition; i < n; ++i) {
35. newArray[i + 1] = originalArray[i];
36. }
37. n++; // Increase the number of elements in the array
38. // Print the updated array
39. cout << "Array after insertion:\n";
40. for (int i = 0; i < n; ++i) {
41. cout << newArray[i] << " ";
42. }
43. cout << "\n";
44. }
45. return 0;
46. }

* Output:

A screenshot of a computer

Description automatically generated

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Experiment-2: Program to delete an element from a given array whose:

1. **Value is given:**

* **Program:**

1. #include <iostream>
2. using namespace std;
3. int main() {
4. int originalArray[100]; // Assuming a maximum array size of 100
5. int n; // Number of elements in the original array
6. int elementToDelete;
7. // Input the number of elements in the original array
8. cout << "Enter the number of elements in the array: ";
9. cin >> n;
10. // Input elements into the original array
11. cout << "Enter " << n << " elements:\n";
12. for (int i = 0; i < n; ++i) {
13. cin >> originalArray[i];
14. }
15. // Input the element to be deleted
16. cout << "Enter the element to delete: ";
17. cin >> elementToDelete;
18. bool elementFound = false;
19. int indexToDelete = -1;
20. // Find the index of the element to delete
21. for (int i = 0; i < n; ++i) {
22. if (originalArray[i] == elementToDelete) {
23. elementFound = true;
24. indexToDelete = i;
25. break; // Exit the loop as the element has been found
26. }
27. }
28. if (elementFound) {
29. // Shift elements after the deleted element one position to the left
30. for (int i = indexToDelete; i < n - 1; ++i) {
31. originalArray[i] = originalArray[i + 1];
32. }
33. n--; // Reduce the number of elements in the array
34. // Print the updated array
35. cout << "Array after deleting " << elementToDelete << ":\n";
36. for (int i = 0; i < n; ++i) {
37. cout << originalArray[i] << " ";
38. }
39. cout << "\n";
40. } else {
41. cout << "Element not found in the array.\n";
42. }
43. return 0;
44. }

* **Output:**

A screenshot of a computer

Description automatically generated

1. **Position is given:**

* **Program:**

1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5. int originalArray[100]; // Assuming a maximum array size of 100
6. int n;                  // Number of elements in the original array
7. int positionToDelete;
8. // Input the number of elements in the original array
9. cout << "Enter the number of elements in the array: ";
10. cin >> n;
11. // Input elements into the original array
12. cout << "Enter " << n << " elements:\n";
13. for (int i = 0; i < n; ++i)
14. {
15. cin >> originalArray[i];
16. }
17. // Input the position at which the element should be deleted (0-based index)
18. cout << "Enter the position to delete an element (0-based index): ";
19. cin >> positionToDelete;
20. // Check if the position is valid
21. if (positionToDelete >= 0 && positionToDelete < n)
22. {
23. // Shift elements after the deleted position one position to the left
24. for (int i = positionToDelete; i < n - 1; ++i)
25. {
26. originalArray[i] = originalArray[i + 1];
27. }
28. n--; // Reduce the number of elements in the array
29. // Print the updated array
30. cout << "Array after deletion:\n";
31. for (int i = 0; i < n; ++i)
32. {
33. cout << originalArray[i] << " ";
34. }
35. cout << "\n";
36. }
37. else
38. {
39. cout << "Invalid position. Position should be between 0 and " << n - 1 << ".\n";
40. }
41. return 0;
42. }

* **Output:**

A screenshot of a computer

Description automatically generated

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Experiment-3: Program to find the location of a given element using Linear Search

* **Program:**

1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5. int arr[100]; // Assuming a maximum array size of 100
6. int n;        // Number of elements in the array
7. int elementToFind;
8. // Input the number of elements in the array
9. cout << "Enter the number of elements in the array: ";
10. cin >> n;
11. // Input elements into the array
12. cout << "Enter " << n << " elements:\n";
13. for (int i = 0; i < n; ++i)
14. {
15. cin >> arr[i];
16. }
17. // Input the element to find
18. cout << "Enter the element to find: ";
19. cin >> elementToFind;
20. int location = -1; // Initialize the location to -1 (not found)
21. // Perform linear search to find the location of the element
22. for (int i = 0; i < n; ++i)
23. {
24. if (arr[i] == elementToFind)
25. {
26. location = i;
27. break;
28. }
29. }
30. if (location != -1)
31. {
32. cout << "Element " << elementToFind << " found at index " << location << ".\n";
33. }
34. else
35. {
36. cout << "Element " << elementToFind << " not found in the array.\n";
37. }
38. return 0;
39. }

* **Output:**

A screenshot of a computer

Description automatically generated

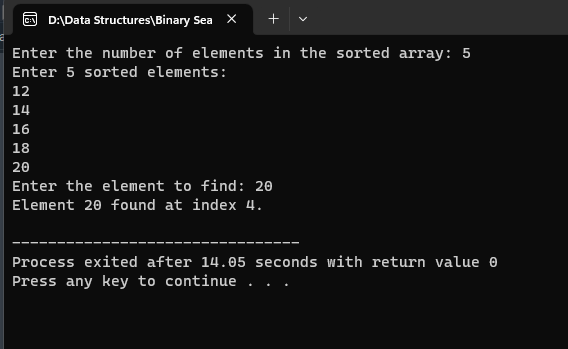
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Experiment-4: Program to find the location of a given element using Binary Search

* **Program:**

1. #include <iostream>
2. using namespace std;
3. // Function to perform binary search and find the location of the element
4. int binarySearch(int arr[], int n, int elementToFind)
5. {
6. int left = 0;
7. int right = n - 1;
8. while (left <= right)
9. {
10. int mid = left + (right - left) / 2;
11. if (arr[mid] == elementToFind)
12. {
13. return mid; // Element found, return its index
14. }
15. else if (arr[mid] < elementToFind)
16. {
17. left = mid + 1; // Search the right half
18. }
19. else
20. {
21. right = mid - 1; // Search the left half
22. }
23. }
24. return -1; // Element not found
25. }
26. int main()
27. {
28. int arr[100]; // Assuming a maximum array size of 100
29. int n;        // Number of elements in the sorted array
30. int elementToFind;
31. // Input the number of elements in the sorted array
32. cout << "Enter the number of elements in the sorted array: ";
33. cin >> n;
34. // Input elements into the sorted array (assuming it's sorted in ascending order)
35. cout << "Enter " << n << " sorted elements:\n";
36. for (int i = 0; i < n; ++i)
37. {
38. cin >> arr[i];
39. }
40. // Input the element to find
41. cout << "Enter the element to find: ";
42. cin >> elementToFind;
43. // Call the binarySearch function to find the location of the element
44. int location = binarySearch(arr, n, elementToFind);
45. if (location != -1)
46. {
47. cout << "Element " << elementToFind << " found at index " << location << ".\n";
48. }
49. else
50. {
51. cout << "Element " << elementToFind << " not found in the array.\n";
52. }
53. return 0;
54. }

* **Output:**



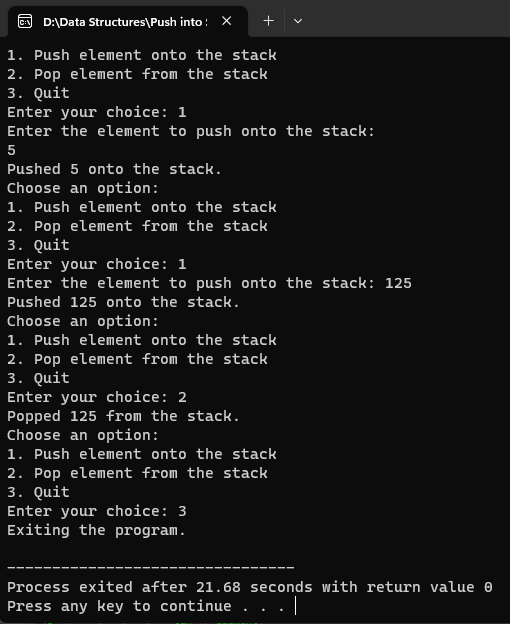
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-5: Program to implement push and pop operations on a stack using linear array.

* **Program:**

1. #include <iostream>
2. using namespace std;
3. const int MAX\_SIZE = 100; // Maximum size of the stack
4. int arr[MAX\_SIZE]; // Linear array to represent the stack
5. int top = -1;      // Variable to keep track of the top of the stack
6. // Function to push an element onto the stack
7. void push(int element)
8. {
9. if (top == MAX\_SIZE - 1)
10. {
11. cout << "Stack Overflow: Cannot push element onto the stack. The stack is full.\n";
12. return;
13. }
14. top++;              // Increment the top of the stack
15. arr[top] = element; // Insert the element at the top position
16. cout << "Pushed " << element << " onto the stack.\n";
17. }
18. // Function to pop an element from the stack
19. void pop()
20. {
21. if (top == -1)
22. {
23. cout << "Stack Underflow: Cannot pop element from the stack. The stack is empty.\n";
24. return;
25. }
26. cout << "Popped " << arr[top] << " from the stack.\n";
27. top--; // Decrement the top to remove the topmost element
28. }
29. int main()
30. {
31. int choice, element;
32. while (true)
33. {
34. cout << "Choose an option:\n";
35. cout << "1. Push element onto the stack\n";
36. cout << "2. Pop element from the stack\n";
37. cout << "3. Quit\n";
38. cout << "Enter your choice: ";
39. cin >> choice;
40. switch (choice)
41. {
42. case 1:
43. cout << "Enter the element to push onto the stack: ";
44. cin >> element;
45. push(element);
46. break;
47. case 2:
48. pop();
49. break;
50. case 3:
51. cout << "Exiting the program.\n";
52. return 0;
53. default:
54. cout << "Invalid choice. Please try again.\n";
55. }
56. }
57. return 0;
58. }

* **Output:**



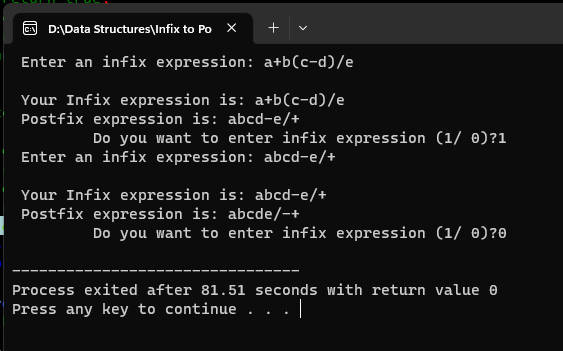
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-6: Program to convert an infix expression to a postfix expression using stack.

* Program:

1. #include <iostream>
2. #include <stack>
3. using namespace std;
4. // defines the Boolean function for operator, operand, equalOrhigher precedence and the string conversion function.
5. bool IsOperator(char);
6. bool IsOperand(char);
7. bool eqlOrhigher(char, char);
8. string convert(string);
9. int main()
10. {
11. string infix\_expression, postfix\_expression;
12. int ch;
13. do
14. {
15. cout << " Enter an infix expression: ";
16. cin >> infix\_expression;
17. postfix\_expression = convert(infix\_expression);
18. cout << "\n Your Infix expression is: " << infix\_expression;
19. cout << "\n Postfix expression is: " << postfix\_expression;
20. cout << "\n \t Do you want to enter infix expression (1/ 0)?";
21. cin >> ch;
22. // cin.ignore();
23. } while (ch == 1);
24. return 0;
25. }
26. // define the IsOperator() function to validate whether any symbol is operator.
27. /\* If the symbol is operator, it returns true, otherwise false. \*/
28. bool IsOperator(char c)
29. {
30. if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^')
31. return true;
32. return false;
33. }
34. // IsOperand() function is used to validate whether the character is operand.
35. bool IsOperand(char c)
36. {
37. if (c >= 'A' && c <= 'Z') /\* Define the character in between A to Z. If not, it returns False.\*/
38. return true;
39. if (c >= 'a' && c <= 'z') // Define the character in between a to z. If not, it returns False. \*/
40. return true;
41. if (c >= '0' && c <= '9') // Define the character in between 0 to 9. If not, it returns False. \*/
42. return true;
43. return false;
44. }
45. // here, precedence() function is used to define the precedence to the operator.
46. int precedence(char op)
47. {
48. if (op == '+' || op == '-') /\* it defines the lowest precedence \*/
49. return 1;
50. if (op == '\*' || op == '/')
51. return 2;
52. if (op == '^') /\* exponent operator has the highest precedence \*/
53. return 3;
54. return 0;
55. }
56. /\* The eqlOrhigher() function is used to check the higher or equal precedence of the two operators in infix expression. \*/
57. bool eqlOrhigher(char op1, char op2)
58. {
59. int p1 = precedence(op1);
60. int p2 = precedence(op2);
61. if (p1 == p2)
62. {
63. if (op1 == '^')
64. return false;
65. return true;
66. }
67. return (p1 > p2 ? true : false);
68. }
69. /\* string convert() function is used to convert the infix expression to the postfix expression of the Stack \*/
70. string convert(string infix)
71. {
72. stack<char> S;
73. string postfix = "";
74. char ch;
75. S.push('(');
76. infix += ')';
77. for (int i = 0; i < infix.length(); i++)
78. {
79. ch = infix[i];
80. if (ch == ' ')
81. continue;
82. else if (ch == '(')
83. S.push(ch);
84. else if (IsOperand(ch))
85. postfix += ch;
86. else if (IsOperator(ch))
87. {
88. while (!S.empty() && eqlOrhigher(S.top(), ch))
89. {
90. postfix += S.top();
91. S.pop();
92. }
93. S.push(ch);
94. }
95. else if (ch == ')')
96. {
97. while (!S.empty() && S.top() != '(')
98. {
99. postfix += S.top();
100. S.pop();
101. }
102. S.pop();
103. }
104. }
105. return postfix;
106. }

* **Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-7: Program to evaluate a postfix expression using stacks.

* Program:

1. #include <bits/stdc++.h>
2. using namespace std;
3. // Stack type
4. class Stack
5. {
6. public:
7. int top;
8. unsigned capacity;
9. int \*array;
10. };
11. // Stack Operations
12. Stack \*createStack(unsigned capacity)
13. {
14. Stack \*stack = new Stack();
15. if (!stack)
16. return NULL;
17. stack->top = -1;
18. stack->capacity = capacity;
19. stack->array = new int[(stack->capacity \* sizeof(int))];
20. if (!stack->array)
21. return NULL;
22. return stack;
23. }
24. int isEmpty(Stack \*stack)
25. {
26. return stack->top == -1;
27. }
28. int peek(Stack \*stack)
29. {
30. return stack->array[stack->top];
31. }
32. int pop(Stack \*stack)
33. {
34. if (!isEmpty(stack))
35. return stack->array[stack->top--];
36. return '$';
37. }
38. void push(Stack \*stack, int op)
39. {
40. stack->array[++stack->top] = op;
41. }
42. // The main function that returns value
43. // of a given postfix expression
44. int evaluatePostfix(char \*exp)
45. {
46. // Create a stack of capacity equal to expression size
47. Stack \*stack = createStack(strlen(exp));
48. int i;
49. // See if stack was created successfully
50. if (!stack)
51. return -1;
52. // Scan all characters one by one
53. for (i = 0; exp[i]; ++i)
54. {
55. // if the character is blank space then continue
56. if (exp[i] == ' ')
57. continue;
58. // If the scanned character is an
59. // operand (number here),extract the full number
60. // Push it to the stack.
61. else if (isdigit(exp[i]))
62. {
63. int num = 0;
64. // extract full number
65. while (isdigit(exp[i]))
66. {
67. num = num \* 10 + (int)(exp[i] - '0');
68. i++;
69. }
70. i--;
71. // push the element in the stack
72. push(stack, num);
73. }
74. // If the scanned character is an operator, pop two
75. // elements from stack apply the operator
76. else
77. {
78. int val1 = pop(stack);
79. int val2 = pop(stack);
80. switch (exp[i])
81. {
82. case '+':
83. push(stack, val2 + val1);
84. break;
85. case '-':
86. push(stack, val2 - val1);
87. break;
88. case '\*':
89. push(stack, val2 \* val1);
90. break;
91. case '/':
92. push(stack, val2 / val1);
93. break;
94. }
95. }
96. return pop(stack);
97. }
98. }
99. // Driver code
100. int main()
101. {
102. char exp[] = "60 6 / 5 2 \* 5 - +";
103. cout << "The result of the evaluated postfix expression is = ";
104. cout << evaluatePostfix(exp);
105. return 0;
106. }

* Output:

A screenshot of a computer

Description automatically generated

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-8: Implement recursive function for Tower of Hanoi problem.

* Program:

1. #include <iostream>
2. using namespace std;
3. class Tower
4. {
5. public:
6. void TOH(int n, char Beg, char Aux, char End)
7. {
8. if (n == 1)
9. {
10. cout << "Moving Disk " << n << " from " << Beg << " to " << End << endl;
11. return;
12. }
13. else
14. {
15. TOH(n - 1, Beg, End, Aux);
16. cout << "Moving Disk " << n << " from " << Beg << " to " << End << endl;
17. TOH(n - 1, Aux, Beg, End);
18. }
19. }
20. };
21. int main()
22. {
23. int n;
24. cout << "Enter number of disks: ";
25. cin >> n;
26. Tower t;
27. t.TOH(n, 'A', 'B', 'C');
28. return 0;
29. }

* Output:

A screenshot of a computer program

Description automatically generated

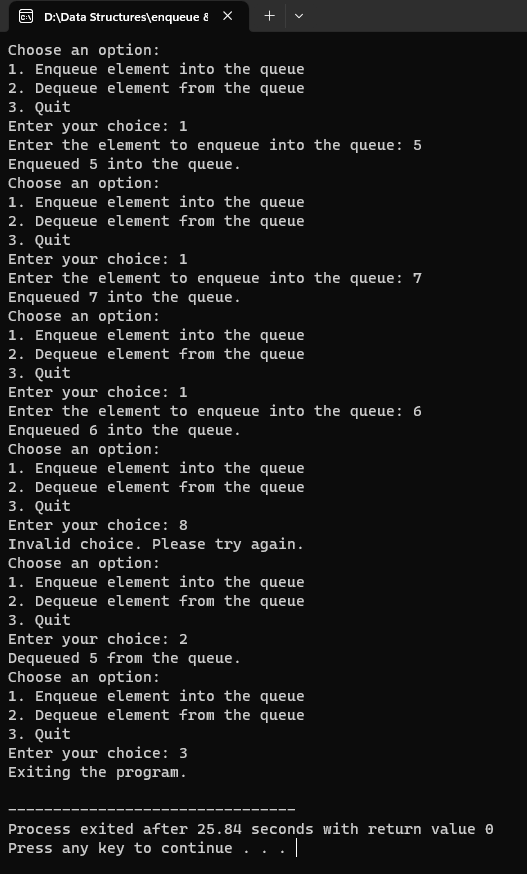
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-9: Program to implement insertion and deletion operations in a queue using linear array.

* **Program:**

1. #include <iostream>
2. using namespace std;
3. const int MAX\_SIZE = 100; // Maximum size of the queue
4. int arr[MAX\_SIZE]; // Linear array to represent the queue
5. int front = -1;    // Pointer to the front of the queue
6. int rear = -1;     // Pointer to the rear of the queue
7. // Function to check if the queue is empty
8. bool isEmpty()
9. {
10. return (front == -1 && rear == -1);
11. }
12. // Function to check if the queue is full
13. bool isFull()
14. {
15. return (rear == MAX\_SIZE - 1);
16. }
17. // Function to enqueue (insert) an element into the queue
18. void enqueue(int element)
19. {
20. if (isFull())
21. {
22. cout << "Queue Overflow: Cannot enqueue element. The queue is full.\n";
23. return;
24. }
25. if (isEmpty())
26. {
27. front = rear = 0; // If the queue was empty, set both front and rear to 0
28. }
29. else
30. {
31. rear++; // Increment the rear pointer
32. }
33. arr[rear] = element; // Insert the element at the rear of the queue
34. cout << "Enqueued " << element << " into the queue.\n";
35. }
36. // Function to dequeue (remove) an element from the queue
37. void dequeue()
38. {
39. if (isEmpty())
40. {
41. cout << "Queue Underflow: Cannot dequeue element. The queue is empty.\n";
42. return;
43. }
44. int removedElement = arr[front];
45. if (front == rear)
46. {
47. front = rear = -1; // If there was only one element, reset both front and rear to -1 (empty queue)
48. }
49. else
50. {
51. front++; // Increment the front pointer
52. }
53. cout << "Dequeued " << removedElement << " from the queue.\n";
54. }
55. int main()
56. {
57. int choice, element;
58. while (true)
59. {
60. cout << "Choose an option:\n";
61. cout << "1. Enqueue element into the queue\n";
62. cout << "2. Dequeue element from the queue\n";
63. cout << "3. Quit\n";
64. cout << "Enter your choice: ";
65. cin >> choice;
66. switch (choice)
67. {
68. case 1:
69. cout << "Enter the element to enqueue into the queue: ";
70. cin >> element;
71. enqueue(element);
72. break;
73. case 2:
74. dequeue();
75. break;
76. case 3:
77. cout << "Exiting the program.\n";
78. return 0;
79. default:
80. cout << "Invalid choice. Please try again.\n";
81. }
82. }
83. return 0;
84. }

* Output:



A screenshot of a computer program

Description automatically generated

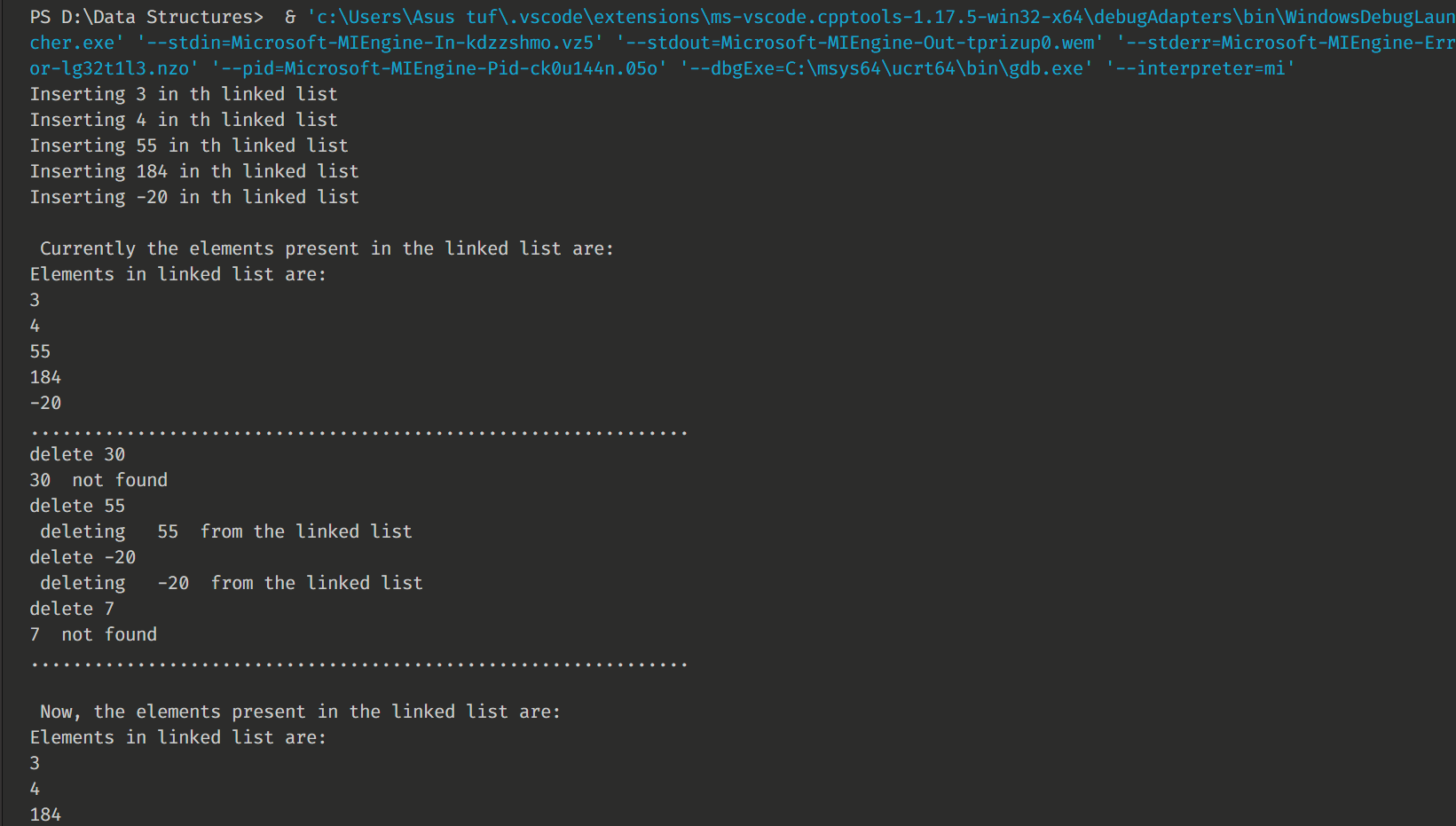
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-10: WAP to implement Linked List.

* **Program:**

1. #include <iostream>
2. #include <stdlib.h>
3. using namespace std;
4. struct Node
5. {
6. int data;
7. Node \*next;
8. };
9. Node \*head = NULL;
10. void insertinLL(int val)
11. {
12. Node \*newnode = (Node \*)malloc(sizeof(Node));
13. newnode->data = val;
14. newnode->next = NULL;
15. Node \*last = head;
16. if (last == NULL)
17. {
18. head = newnode;
19. return;
20. }
21. else
22. {
23. while (last->next != NULL)
24. {
25. last = last->next;
26. }
27. last->next = newnode;
28. return;
29. }
30. }
31. void display()
32. {
33. Node \*ptr = head;
34. if (ptr == NULL)
35. {
36. cout << "No elements are there in Linked List";
37. }
38. else
39. {
40. cout << "Elements in linked list are:\n";
41. while (ptr != NULL)
42. {
43. cout << ptr->data;
44. cout << "\n";
45. ptr = ptr->next;
46. }
47. }
48. }
49. void deletedata(int todel)
50. {
51. Node \*temp = head;
52. Node \*prev;
53. while (temp != NULL && temp->data != todel)
54. {
55. prev = temp;
56. temp = temp->next;
57. }
58. if (temp == NULL)
59. {
60. cout << "\n"
61. << todel << "  not found \n";
62. return;
63. }
64. else
65. {
66. cout << "\n deleting   " << todel << "  from the linked list \n";
67. prev->next = temp->next;
68. free(temp);
69. }
70. }
71. int main()
72. {
73. cout << "Inserting 3 in th linked list \n";
74. insertinLL(3);
75. cout << "Inserting 4 in th linked list \n";
76. insertinLL(4);
77. cout << "Inserting 55 in th linked list \n";
78. insertinLL(55);
79. cout << "Inserting 184 in th linked list \n";
80. insertinLL(184);
81. cout << "Inserting -20 in th linked list \n";
82. insertinLL(-20);
83. cout << "\n Currently the elements present in the linked list are:\n";
84. display();
85. cout << "..............................................................\n";
86. cout << "delete 30 ";
87. deletedata(30);
88. cout << "delete 55 ";
89. deletedata(55);
90. cout << "delete -20";
91. deletedata(-20);
92. cout << "delete 7 ";
93. deletedata(7);
94. cout << "..............................................................\n";
95. cout << "\n Now, the elements present in the linked list are:\n";
96. display();
97. return 0;
98. }

* **Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-11: WAP to implement Stack using linked list.

* **Program:**

1. #include <iostream>
2. using namespace std;
3. struct Node
4. {
5. int data;
6. struct Node \*next;
7. };
8. struct Node \*top = NULL;
9. void push(int val)
10. {
11. struct Node \*newnode = (struct Node \*)malloc(sizeof(struct Node));
12. newnode->data = val;
13. newnode->next = top;
14. top = newnode;
15. }
16. void pop()
17. {
18. if (top == NULL)
19. cout << "Stack Underflow" << endl;
20. else
21. {
22. cout << "The popped element is " << top->data << endl;
23. top = top->next;
24. }
25. }
26. void display()
27. {
28. struct Node \*ptr;
29. if (top == NULL)
30. cout << "stack is empty";
31. else
32. {
33. ptr = top;
34. cout << "Stack elements are: ";
35. while (ptr != NULL)
36. {
37. cout << ptr->data << " ";
38. ptr = ptr->next;
39. }
40. }
41. cout << endl;
42. }
43. int main()
44. {
45. int ch, val;
46. cout << "1) Push in stack" << endl;
47. cout << "2) Pop from stack" << endl;
48. cout << "3) Display stack" << endl;
49. cout << "4) Exit" << endl;
50. do
51. {
52. cout << "Enter choice: ";
53. cin >> ch;
54. switch (ch)
55. {
56. case 1:
57. {
58. cout << "Enter value to be pushed: ";
59. cin >> val;
60. push(val);
61. break;
62. }
63. case 2:
64. {
65. pop();
66. break;
67. }
68. case 3:
69. {
70. display();
71. break;
72. }
73. case 4:
74. {
75. cout << "Exit" << endl;
76. break;
77. }
78. default:
79. {
80. cout << "Invalid Choice" << endl;
81. }
82. }
83. } while (ch != 4);
84. return 0;
85. }

* **Output:**

A black screen with blue text

Description automatically generated

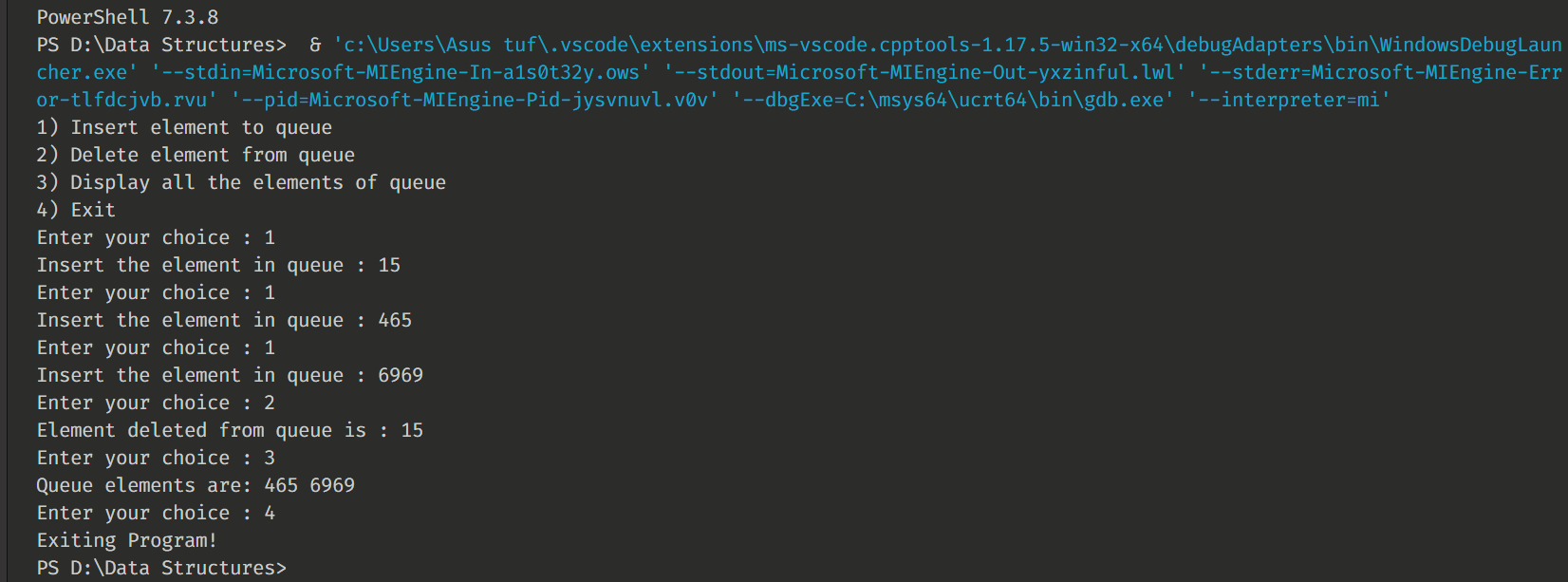
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-12: WAP to implement Queue using Linked List.

* **Program:**

1. #include <iostream>
2. using namespace std;
3. struct node
4. {
5. int data;
6. struct node \*next;
7. };
8. struct node \*front = NULL;
9. struct node \*rear = NULL;
10. struct node \*temp;
11. void Insert()
12. {
13. int val;
14. cout << "Insert the element in queue : ";
15. cin >> val;
16. if (rear == NULL)
17. {
18. rear = (struct node \*)malloc(sizeof(struct node));
19. rear->next = NULL;
20. rear->data = val;
21. front = rear;
22. }
23. else
24. {
25. temp = (struct node \*)malloc(sizeof(struct node));
26. rear->next = temp;
27. temp->data = val;
28. temp->next = NULL;
29. rear = temp;
30. }
31. }
32. void Delete()
33. {
34. temp = front;
35. if (front == NULL)
36. {
37. cout << "Underflow" << endl;
38. return;
39. }
40. else if (temp->next != NULL)
41. {
42. temp = temp->next;
43. cout << "Element deleted from queue is : " << front->data << endl;
44. free(front);
45. front = temp;
46. }
47. else
48. {
49. cout << "Element deleted from queue is : " << front->data << endl;
50. free(front);
51. front = NULL;
52. rear = NULL;
53. }
54. }
55. void Display()
56. {
57. temp = front;
58. if ((front == NULL) && (rear == NULL))
59. {
60. cout << "Queue is empty" << endl;
61. return;
62. }
63. cout << "Queue elements are: ";
64. while (temp != NULL)
65. {
66. cout << temp->data << " ";
67. temp = temp->next;
68. }
69. cout << endl;
70. }
71. int main()
72. {
73. int ch;
74. cout << "1) Insert element to queue" << endl;
75. cout << "2) Delete element from queue" << endl;
76. cout << "3) Display all the elements of queue" << endl;
77. cout << "4) Exit" << endl;
78. do
79. {
80. cout << "Enter your choice : ";
81. cin >> ch;
82. switch (ch)
83. {
84. case 1:
85. Insert();
86. break;
87. case 2:
88. Delete();
89. break;
90. case 3:
91. Display();
92. break;
93. case 4:
94. cout << "Exiting Program!" << endl;
95. break;
96. default:
97. cout << "Invalid choice" << endl;
98. }
99. } while (ch != 4);
100. return 0;
101. }

* **Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-13: WAP to implement Bubble Sort.

* **Program:**

1. #include <iostream>
2. using namespace std;
3. void swapping(int &a, int &b)
4. { // swap the content of a and b
5. int temp;
6. temp = a;
7. a = b;
8. b = temp;
9. }
10. void display(int \*array, int size)
11. {
12. for (int i = 0; i < size; i++)
13. cout << array[i] << " ";
14. cout << endl;
15. }
16. void bubbleSort(int \*array, int size)
17. {
18. for (int i = 0; i < size; i++)
19. {
20. int swaps = 0; // flag to detect any swap is there or not
21. for (int j = 0; j < size - i - 1; j++)
22. {
23. if (array[j] > array[j + 1])
24. { // when the current item is bigger than next
25. swapping(array[j], array[j + 1]);
26. swaps = 1; // set swap flag
27. }
28. }
29. if (!swaps)
30. break; // No swap in this pass, so array is sorted
31. }
32. }
33. int main()
34. {
35. int n;
36. cout << "Enter the number of elements: ";
37. cin >> n;
38. int arr[n]; // create an array with given number of elements
39. cout << "Enter elements:" << endl;
40. for (int i = 0; i < n; i++)
41. {
42. cin >> arr[i];
43. }
44. cout << "Array before Sorting: ";
45. display(arr, n);
46. bubbleSort(arr, n);
47. cout << "Array after Sorting: ";
48. display(arr, n);
49. }

* **Output:**

A screenshot of a computer

Description automatically generated

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-14: WAP to implement Selection Sort.

* **Program:**

1. #include <iostream>
2. using namespace std;
3. void swapping(int &a, int &b)
4. { // swap the content of a and b
5. int temp;
6. temp = a;
7. a = b;
8. b = temp;
9. }
10. void display(int \*array, int size)
11. {
12. for (int i = 0; i < size; i++)
13. cout << array[i] << " ";
14. cout << endl;
15. }
16. void selectionSort(int \*array, int size)
17. {
18. int i, j, imin;
19. for (i = 0; i < size - 1; i++)
20. {
21. imin = i; // get index of minimum data
22. for (j = i + 1; j < size; j++)
23. if (array[j] < array[imin])
24. imin = j;
25. // placing in correct position
26. swap(array[i], array[imin]);
27. }
28. }
29. int main()
30. {
31. int n;
32. cout << "Enter the number of elements: ";
33. cin >> n;
34. int arr[n]; // create an array with given number of elements
35. cout << "Enter elements:" << endl;
36. for (int i = 0; i < n; i++)
37. {
38. cin >> arr[i];
39. }
40. cout << "Array before Sorting: ";
41. display(arr, n);
42. selectionSort(arr, n);
43. cout << "Array after Sorting: ";
44. display(arr, n);
45. }

* **Output:**

A computer screen with blue text

Description automatically generated

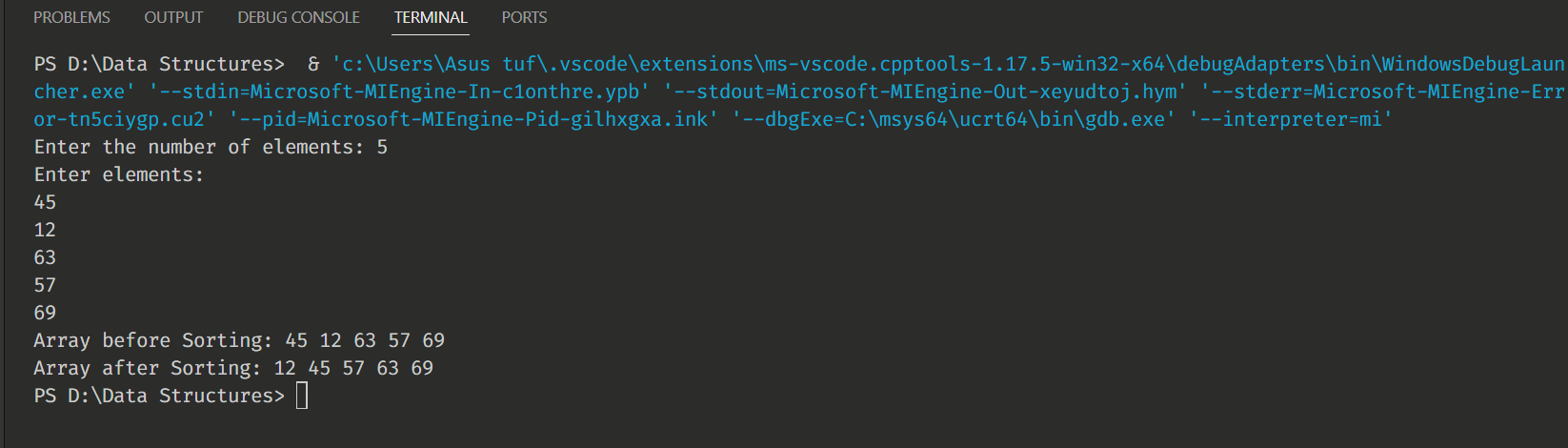
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-15: WAP to implement Insertion Sort

* **Program:**

1. #include <iostream>
2. using namespace std;
3. void display(int \*array, int size)
4. {
5. for (int i = 0; i < size; i++)
6. cout << array[i] << " ";
7. cout << endl;
8. }
9. void insertionSort(int \*array, int size)
10. {
11. int key, j;
12. for (int i = 1; i < size; i++)
13. {
14. key = array[i]; // take value
15. j = i;
16. while (j > 0 && array[j - 1] > key)
17. {
18. array[j] = array[j - 1];
19. j--;
20. }
21. array[j] = key; // insert in right place
22. }
23. }
24. int main()
25. {
26. int n;
27. cout << "Enter the number of elements: ";
28. cin >> n;
29. int arr[n]; // create an array with given number of elements
30. cout << "Enter elements:" << endl;
31. for (int i = 0; i < n; i++)
32. {
33. cin >> arr[i];
34. }
35. cout << "Array before Sorting: ";
36. display(arr, n);
37. insertionSort(arr, n);
38. cout << "Array after Sorting: ";
39. display(arr, n);
40. return 0;
41. }

* **Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-16: WAP to implement Quick Sort.

* **Program:**

1. #include <iostream>
2. using namespace std;
3. // Swap two elements - Utility function
4. void swap(int \*a, int \*b)
5. {
6. int t = \*a;
7. \*a = \*b;
8. \*b = t;
9. }
10. // partition the array using last element as pivot
11. int partition(int arr[], int low, int high)
12. {
13. int pivot = arr[high]; // pivot
14. int i = (low - 1);
15. for (int j = low; j <= high - 1; j++)
16. {
17. // if current element is smaller than pivot, increment the low element
18. // swap elements at i and j
19. if (arr[j] <= pivot)
20. {
21. i++; // increment index of smaller element
22. swap(&arr[i], &arr[j]);
23. }
24. }
25. swap(&arr[i + 1], &arr[high]);
26. return (i + 1);
27. }
28. // quicksort algorithm
29. void quickSort(int arr[], int low, int high)
30. {
31. if (low < high)
32. {
33. // partition the array
34. int pivot = partition(arr, low, high);
35. // sort the sub arrays independently
36. quickSort(arr, low, pivot - 1);
37. quickSort(arr, pivot + 1, high);
38. }
39. }
40. void displayArray(int arr[], int size)
41. {
42. int i;
43. for (i = 0; i < size; i++)
44. {
45. cout << arr[i] << "\t";
46. }
47. }
48. int main()
49. {
50. int arr[] = {12, 23, 3, 43, 51, 35, 19, 45};
51. int n = sizeof(arr) / sizeof(arr[0]);
52. cout << "Input array: " << endl;
53. displayArray(arr, n);
54. cout << endl;
55. quickSort(arr, 0, n - 1);
56. cout << "Array sorted with quick sort" << endl;
57. displayArray(arr, n);
58. return 0;
59. }

* **Output:**

A computer screen shot of a program

Description automatically generated

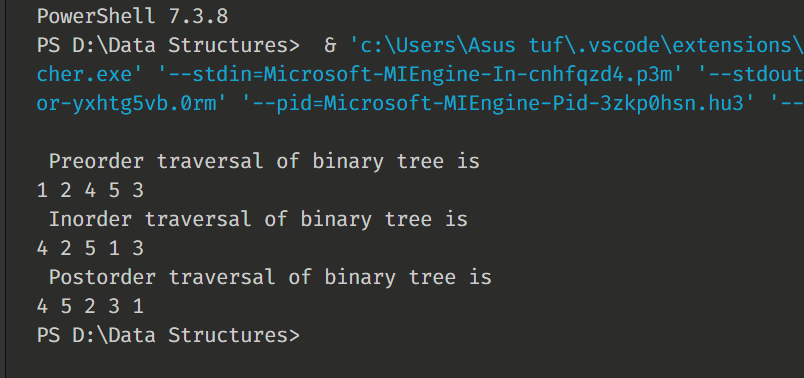
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-17: WAP to implement Preorder, Inorder, Postorder Traversals

* **Program:**

1. #include <iostream>
2. using namespace std;
3. /\* A binary tree node has data, pointer to left child
4. and a pointer to right child \*/
5. struct Node
6. {
7. int data;
8. struct Node \*left, \*right;
9. Node(int data)
10. {
11. this->data = data;
12. left = right = NULL;
13. }
14. };
15. /\* Given a binary tree, print its nodes according to the
16. "bottom-up" postorder traversal. \*/
17. void printPostorder(struct Node \*node)
18. {
19. if (node == NULL)
20. return;
21. // first recur on left subtree
22. printPostorder(node->left);
23. // then recur on right subtree
24. printPostorder(node->right);
25. // now deal with the node
26. cout << node->data << " ";
27. }
28. /\* Given a binary tree, print its nodes in inorder\*/
29. void printInorder(struct Node \*node)
30. {
31. if (node == NULL)
32. return;
33. /\* first recur on left child \*/
34. printInorder(node->left); /\* then print the data of node \*/
35. cout << node->data << " ";
36. /\* now recur on right child \*/
37. printInorder(node->right);
38. }
39. /\* Given a binary tree, print its nodes in preorder\*/
40. void printPreorder(struct Node \*node)
41. {
42. if (node == NULL)
43. return;
44. /\* first print data of node \*/
45. cout << node->data << " ";
46. /\* then recur on left sutree \*/
47. printPreorder(node->left);
48. /\* now recur on right subtree \*/
49. printPreorder(node->right);
50. }
51. /\* Driver program to test above functions\*/
52. int main()
53. {
54. struct Node \*root = new Node(1);
55. root->left = new Node(2);
56. root->right = new Node(3);
57. root->left->left = new Node(4);
58. root->left->right = new Node(5);
59. cout << "\n Preorder traversal of binary tree is \n";
60. printPreorder(root);
61. cout << "\n Inorder traversal of binary tree is \n";
62. printInorder(root);
63. cout << "\n Postorder traversal of binary tree is \n";
64. printPostorder(root);
65. return 0;
66. }

* **Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-18: WAP to implement BFS (Breadth First Search).

* **Program:**

1. #include <iostream>
2. #include <queue>
3. #define NODE 6
4. using namespace std;
5. typedef struct node
6. {
7. int val;
8. int state; // status
9. } node;
10. int graph[NODE][NODE] = {
11. {0, 1, 1, 1, 0, 0},
12. {1, 0, 0, 1, 1, 0},
13. {1, 0, 0, 1, 0, 1},
14. {1, 1, 1, 0, 1, 1},
15. {0, 1, 0, 1, 0, 1},
16. {0, 0, 1, 1, 1, 0}};
17. void bfs(node \*vert, node s)
18. {
19. node u;
20. int i, j;
21. queue<node> que;
22. for (i = 0; i < NODE; i++)
23. {
24. vert[i]
25. .state = 0; // not visited
26. }
27. vert[s.val].state = 1; // visited
28. que.push(s);           // insert starting node
29. while (!que.empty())
30. {
31. u = que.front(); // delete from queue and print
32. que.pop();
33. cout << char(u.val + 'A') << " ";
34. for (i = 0; i < NODE; i++)
35. {
36. if (graph[i][u.val])
37. {
38. // when the node is non-visited
39. if (vert[i].state == 0)
40. {
41. vert[i].state = 1;
42. que.push(vert[i]);
43. }
44. }
45. }
46. u.state = 2; // completed for node u
47. }
48. }
49. int main()
50. {
51. node vertices[NODE];
52. node start;
53. char s;
54. for (int i = 0; i < NODE; i++)
55. {
56. vertices[i].val = i;
57. }
58. s = 'B'; // starting vertex B
59. start.val = s - 'A';
60. cout << "BFS Traversal: ";
61. bfs(vertices, start);
62. cout << endl;
63. }

* **Output:**

A computer screen with white text

Description automatically generated

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Experiment-19: WAP to implement DFS (Depth First Search).

* **Program:**

1. #include <iostream>
2. #include <stack>
3. using namespace std;
4. #define NODE 6
5. typedef struct node
6. {
7. int val;
8. int state; // status
9. } node;
10. int graph[NODE][NODE] =
11. {
12. {0, 1, 1, 1, 0, 0},
13. {1, 0, 0, 1, 1, 0},
14. {1, 0, 0, 1, 0, 1},
15. {1, 1, 1, 0, 1, 1},
16. {0, 1, 0, 1, 0, 1},
17. {0, 0, 1, 1, 1, 0}
18. };
19. void dfs(node \*vertex, node start)
20. {
21. node u;
22. stack<node> myStack;
23. for (int i = 0; i < NODE; i++)
24. {
25. vertex[i].state = 0; // not visited
26. }
27. myStack.push(start);
28. while (!myStack.empty())
29. {
30. // pop and print node
31. u = myStack.top();
32. myStack.pop();
33. cout << char(u.val + 'A') << " ";
34. if (u.state != 1)
35. {
36. // update vertex status to visited
37. u.state = 1;
38. vertex[u.val].state = 1;
39. for (int i = 0; i < NODE; i++)
40. {
41. if (graph[i][u.val])
42. {
43. if (vertex[i].state == 0)
44. {
45. myStack.push(vertex[i]);
46. vertex[i].state = 1;
47. }
48. }
49. }
50. }
51. }
52. }
53. int main()
54. {
55. node vertices[NODE];
56. node start;
57. char s;
58. for (int i = 0; i < NODE; i++)
59. {
60. vertices[i].val = i;
61. }
62. s = 'C'; // starting vertex C
63. start.val = s - 'A';
64. cout << "DFS Traversal: ";
65. dfs(vertices, start);
66. cout << endl;
67. }

* **Output:**

A computer screen with white text

Description automatically generated

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_