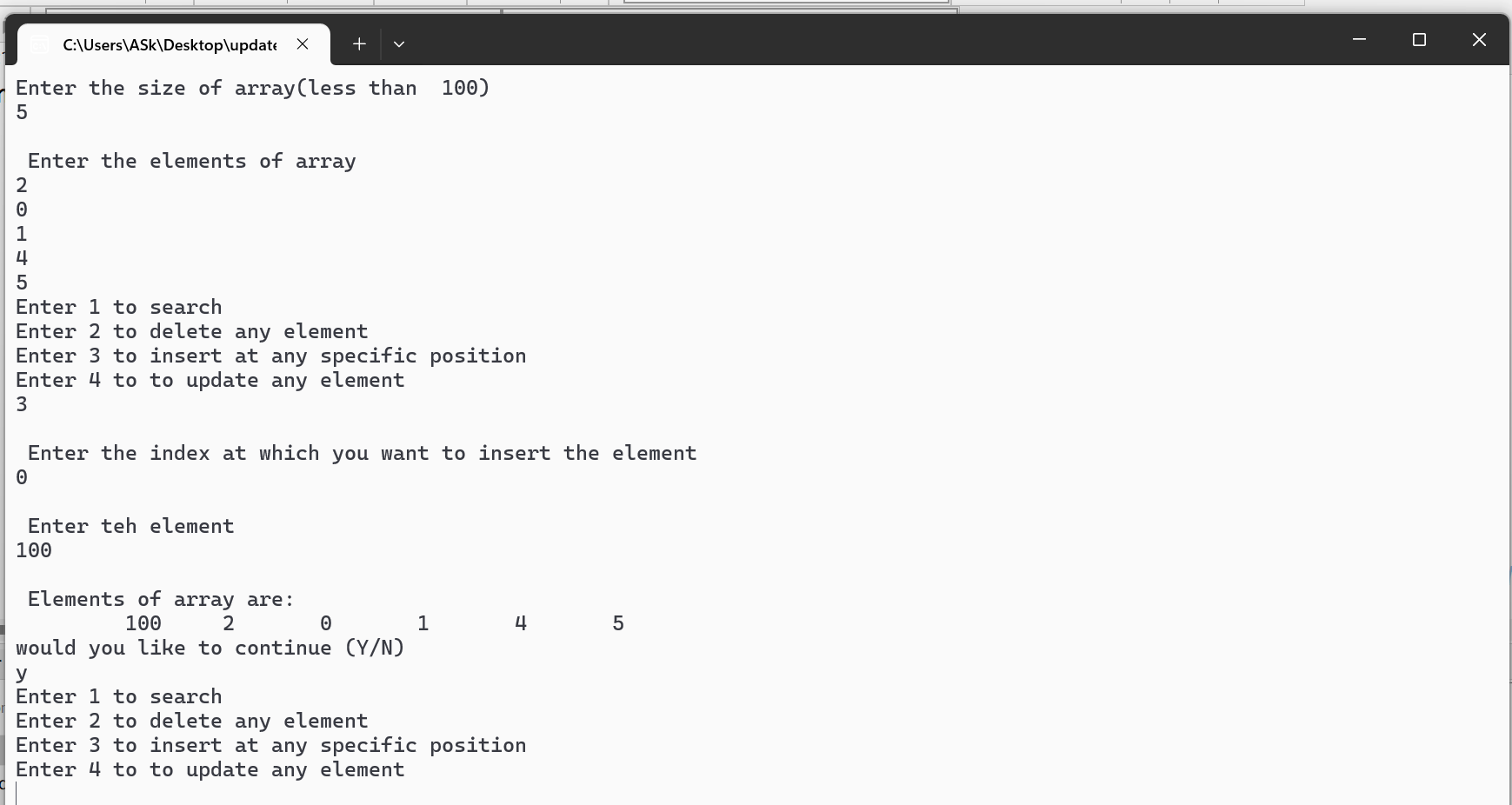
**Week1**

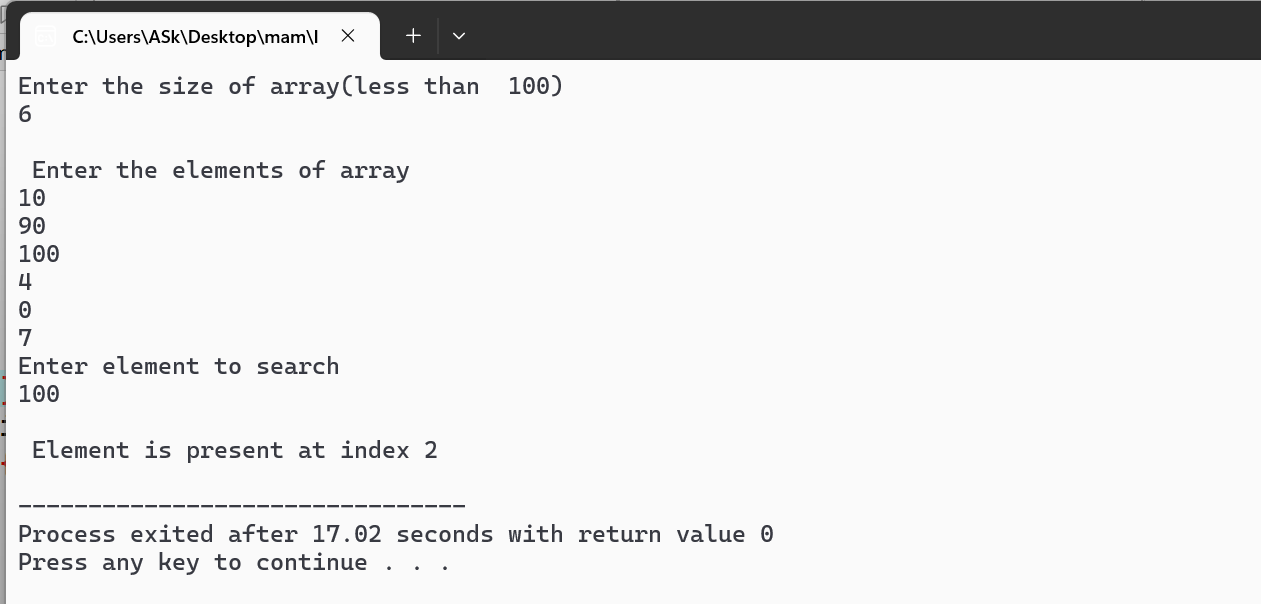
**1. Write a program in C++ to insert, delete, and update the contents of an array.**

|  |  |
| --- | --- |
| 1.  #include<iostream>  using namespace std;  class Array\_Operations  {  public:  void search(int arr[], int size)// function to search for a particular element  {  int index,flag=0;  int key;  cout<<"Enter element to search"<<endl;  cin>>key;  for(int i=0;i<size;i++)  {  if(arr[i]==key)  {  flag=1;  index=i;  break;  }  }  if(flag)  {  cout<<"\n Element is present at index "<<index<<endl;  }  else  {    cout<<"Element not found"<<endl;  }  }  void delete\_element(int arr[], int &size) //function to delete any element  {  if (size == 0) {  cout << "Array is empty!\n";  return;  }  int position;  cout<<"\n Enter the index you want to delete"<<endl;  cin>>position;    int key=arr[position];  if(position<0 || position>size)  3.  cout<<"invalid index, Enter between (0 to "<<n<<")"<<endl;  return;  }  else  {  int key;  cout<<"Enter the element"<<endl;  cin>>key;  arr[index]=key;  }  }  void display(int arr[], int size)  {  cout<<"\n Elements of array are:"<<endl;  for(int i=0;i<size;i++)  {  cout<<"\t "<<arr[i];  }  }  };  int choice()  {  int choice;  cout<<"Enter 1 to search"<<endl;  cout<<"Enter 2 to delete any element"<<endl;  cout<<"Enter 3 to insert at any specific position"<<endl;  cout<<"Enter 4 to to update any element"<<endl;  cin>>choice;  return choice;  }  int main()  {  int n;  char ch;  int arr[100];//maximum size is 100  cout<<"Enter the size of array(less than 100)"<<endl;  cin>>n;  cout<<"\n Enter the elements of array"<<endl;  for(int i=0;i<n;i++)  {  cin>>arr[i];  }  Array\_Operations obj; //creating object of Array\_Operations class (obj)    do{  switch(choice()) // here choice() will be | 2.  {  cout <<"invalid position"<<endl;  }  else  {    for(int i=position;i<size-1;i++)  {  arr[i]=arr[i+1];    }  size--;  cout<<key<<" has been deleted"<<endl;      }    }  void insert\_element(int arr[], int &size)//function to insert at specific index  {  int position,item;  cout<<"\n Enter the index at which you want to insert the element"<<endl;  cin>>position;  cout<<"\n Enter teh element"<<endl;  cin>>item;  if(position<0 || position>size)  {  cout <<"invalid position"<<endl;  }else  {  size++;  for(int i=size;i>=position;i--)  {  arr[i]=arr[i-1];  }  arr[position]=item;  }    }  void update(int arr[], int n)  {  int index;  cout<<"Enter the index"<<endl;  cin>>index;  if(index<0 || index>n)  {  4.  called which will return the operation specified by user  {  case 1: obj.search(arr, n);  break;    case 2: obj.delete\_element(arr,n);  obj.display(arr,n);  break;    case 3: obj.insert\_element(arr,n);  obj.display(arr,n);  break;    case 4: obj.update(arr,n);  obj.display(arr,n);  break;    default: cout<<"invalid operation"<<endl;  break;  }    cout<<"\nwould you like to continue (Y/N)"<<endl;  cin>> ch;      }while(ch=='Y' || ch=='y');          } |



**2. Write a program in C++ to search an element in an array.**

|  |  |
| --- | --- |
| 1. #include<iostream>  using namespace std;  class Search  {  public:  void search(int arr[], int size)// function to search for a particular element  {  int index,flag=0;  int key;  cout<<"Enter element to search"<<endl;  cin>>key;  for(int i=0;i<size;i++)  {  if(arr[i]==key)  {  flag=1;  index=i;  break;  }  }  if(flag)  {  cout<<"\n Element is present at index "<<index<<endl; | 2.  }  else  {    cout<<"Element not found"<<endl;  }  }  };  int main()  {  int n;  char ch;  int arr[100];//maximum size is 100  cout<<"Enter the size of array(less than 100)"<<endl;  cin>>n;  cout<<"\n Enter the elements of array"<<endl;  for(int i=0;i<n;i++)  {  cin>>arr[i];  }  Search obj; //creating object of Search class (obj)  obj.search(arr,n);  } |

****

**3. Write a program in C++ to perform various operations on matrices.**

|  |  |
| --- | --- |
| 1.  #include <iostream>  using namespace std;  class Matrix {  public:  void input(int mat[2][2], string name) {  cout << "Enter elements for " << name << " (2x2 matrix):\n";  for (int i = 0; i < 2; i++)  for (int j = 0; j < 2; j++)  cin >> mat[i][j];  }  void display(int mat[2][2], string name) {  cout << name << ":\n";  for (int i = 0; i < 2; i++) {  for (int j = 0; j < 2; j++)  cout << mat[i][j] << "\t";  cout << endl;  }  }  void add(int mat1[2][2], int mat2[2][2]) {  int result[2][2];  for (int i = 0; i < 2; i++)  for (int j = 0; j < 2; j++)  result[i][j] = mat1[i][j] + mat2[i][j];  display(result, "Result of Addition");  }  3.  int choice;  obj.input(mat1, "Matrix 1");  obj.input(mat2, "Matrix 2");  do {  cout << "\n--------- MENU ---------\n";  cout << "Enter 1 to Display Matrices\n";  cout << "Enter 2 for Addition\n";  cout << "Enter 3 for Subtraction\n";  cout << "Enter 4 for Multiplication\n";  cout << "Enter 5 for Transpose\n";  cout << "Enter 6 to Exit\n";  cout << "------------------------\n";  cout << "Enter your choice: ";  cin >> choice;  switch (choice) {  case 1:  obj.display(mat1, "Matrix 1");  obj.display(mat2, "Matrix 2");  break;  case 2:  obj.add(mat1, mat2);  break;  case 3:  obj.subtract(mat1, mat2);  break;  case 4:  obj.multiply(mat1, mat2);  break;  case 5: {  int tChoice;  cout << "Enter 1 for Transpose of Matrix 1\n";  cout << "Enter 2 for Transpose of Matrix 2\n";  cout << "Your choice: ";  cin >> tChoice;  if (tChoice == 1)  obj.transpose(mat1, "Matrix 1");  else if (tChoice == 2)  obj.transpose(mat2, "Matrix 2");  else  cout << "Invalid choice for transpose!\n";  break;  }  case 6:  cout << "Exiting program. Bye bye 😎\n";  break;  default:  cout << "Invalid choice! Try again.\n";  }  } while (choice != 6);  return 0; } | 2.  void subtract(int mat1[2][2], int mat2[2][2]) {  int result[2][2];  for (int i = 0; i < 2; i++)  for (int j = 0; j < 2; j++)  result[i][j] = mat1[i][j] - mat2[i][j];  display(result, "Result of Subtraction");  }  void multiply(int mat1[2][2], int mat2[2][2]) {  int result[2][2] = {0};  for (int i = 0; i < 2; i++) {  for (int j = 0; j < 2; j++) {  for (int k = 0; k < 2; k++)  result[i][j] += mat1[i][k] \* mat2[k][j];  }  }  display(result, "Result of Multiplication");  }  void transpose(int mat[2][2], string name) {  int result[2][2];  for (int i = 0; i < 2; i++)  for (int j = 0; j < 2; j++)  result[i][j] = mat[j][i];  display(result, "Transpose of " + name);  }  };  int main() {  Matrix obj;  int mat1[2][2], mat2[2][2]; |

**4. Write a program in C++ to implement different string manipulation operations?**

|  |  |
| --- | --- |
| 1.  #include <iostream>  using namespace std;  int stringLength(char str[]) {  int len = 0;  while (str[len] != '\0')  len++;  return len;  }  void stringCopy(char dest[], char src[]) {  int i = 0;  while (src[i] != '\0') {  dest[i] = src[i];  i++;  }  dest[i] = '\0';  }  void stringConcat(char str1[], char str2[]) {  int i = 0;  while (str1[i] != '\0') i++;  int j = 0;  while (str2[j] != '\0') {  str1[i] = str2[j];  i++;  j++;  }  str1[i] = '\0';  }  void stringReverse(char str[]) {  int len = stringLength(str);  for (int i = 0; i < len / 2; i++) {  char temp = str[i];  str[i] = str[len - 1 - i];  str[len - 1 - i] = temp;  }  }  bool stringCompare(char str1[], char str2[]) {  int i = 0;  while (str1[i] != '\0' && str2[i] != '\0') {  if (str1[i] != str2[i])  return false;  i++;  }  return str1[i] == str2[i];  }  3.  stringReverse(str1);  cout << "Reversed string: " << str1 << endl;  break;    case 5:  cout << "Enter another string to compare: ";  cin.getline(str2, 100);  if (stringCompare(str1, str2))  cout << "Strings are equal.\n";  else  cout << "Strings are not equal.\n";  break;  case 6:  toUpperCase(str1);  cout << "Uppercase: " << str1 << endl;  break;  case 7:  toLowerCase(str1);  cout << "Lowercase: " << str1 << endl;  break;  case 8:  cout << "Exiting program...\n";  break;  default:  cout << "Invalid choice! Try again.\n";  }  } while (choice != 8);  return 0;  } | 2.  void toUpperCase(char str[]) {  for (int i = 0; str[i] != '\0'; i++) {  if (str[i] >= 'a' && str[i] <= 'z')  str[i] = str[i] - 32;  }  }  void toLowerCase(char str[]) {  for (int i = 0; str[i] != '\0'; i++) {  if (str[i] >= 'A' && str[i] <= 'Z')  str[i] = str[i] + 32;  }  }  int main() {  char str1[100], str2[100];  int choice;  cout << "Enter first string: ";  cin.getline(str1, 100);  do {  cout << "\n------ STRING OPERATIONS MENU ------\n";  cout << "1. Length of the string\n";  cout << "2. Copy string to another\n";  cout << "3. Concatenate with another string\n";  cout << "4. Reverse the string\n";  cout << "5. Compare with another string\n";  cout << "6. Convert to Uppercase\n";  cout << "7. Convert to Lowercase\n";  cout << "8. Exit\n";  cout << "Enter your choice: ";  cin >> choice;  cin.ignore(); // to consume newline  switch (choice) {  case 1:  cout << "Length: " << stringLength(str1) << endl;  break;  case 2:  stringCopy(str2, str1);  cout << "Copied string: " << str2 << endl;  break;  case 3:  cout << "Enter another string to concatenate: ";  cin.getline(str2, 100);  stringConcat(str1, str2);  cout << "After concatenation: " << str1 << endl;  break;  case 4: |

**5. Write a program to search an element in array using Binary Search.**

**6. Write a program to implement Selection sort**

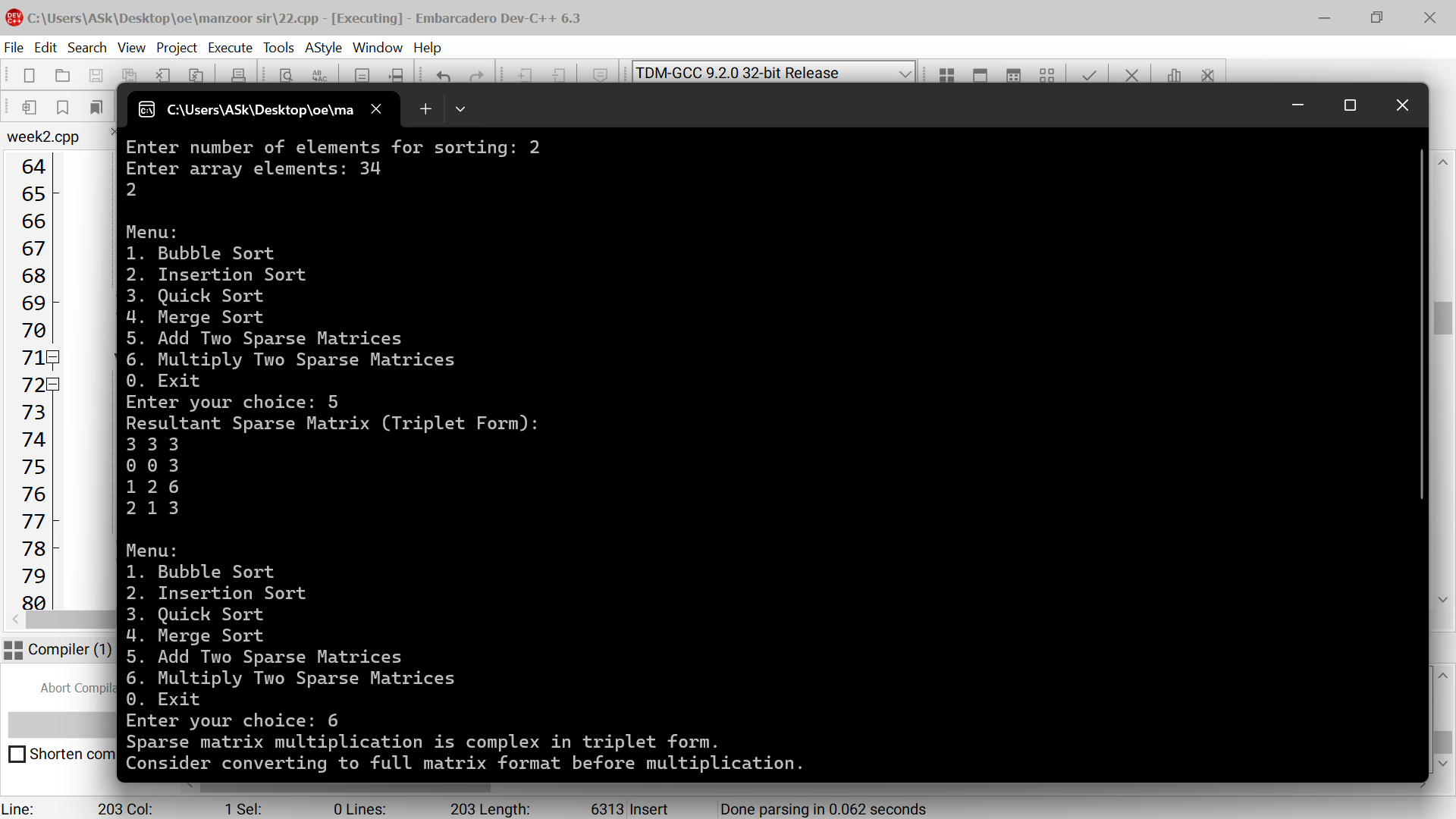
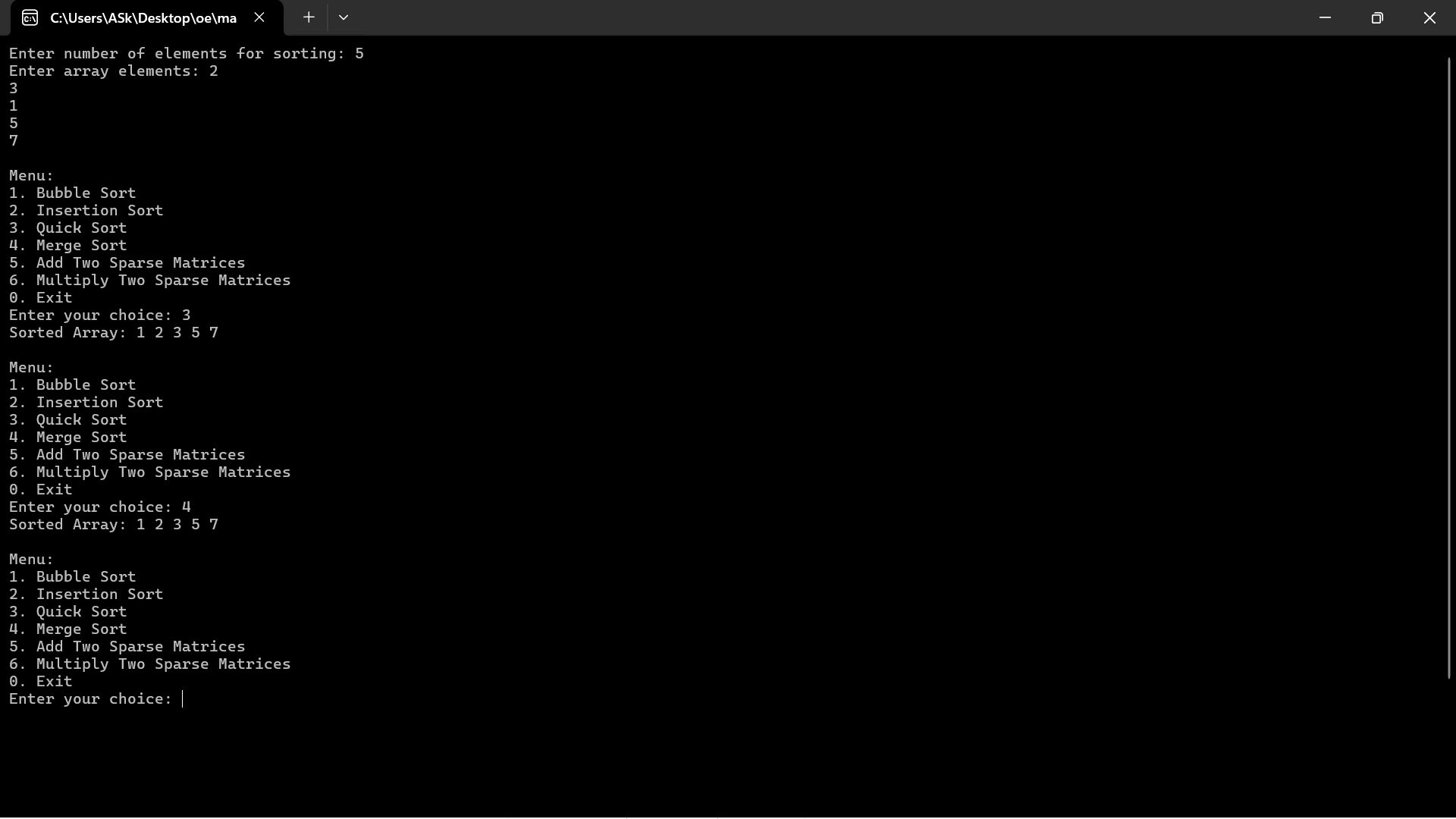
**7. Write a program to implement bubble sort**

|  |  |
| --- | --- |
| 1.  #include <iostream>  using namespace std;  class ArrayOperations {  public:  void bubble\_sort(int arr[], int size) {  bool issorted = true;  for (int i = 0; i < size - 1; i++) {  for (int j = 0; j < size - i - 1; j++) {  if (arr[j] > arr[j + 1]) {  issorted = false;  int temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  }  }  if (issorted) {  cout << "\nArray is already sorted.\n";  return;  }  }  cout << "\nArray after Bubble Sort:\n";  for (int i = 0; i < size; i++)  cout << arr[i] << "\t";  cout << endl;  }  void selection\_sort(int arr[], int size) {  for (int i = 0; i < size - 1; i++) {  int small = i;  for (int j = i + 1; j < size; j++) {  if (arr[j] < arr[small])  small = j;  }  int temp = arr[i];  arr[i] = arr[small];  arr[small] = temp;  }  cout << "\nArray after Selection Sort:\n";  for (int i = 0; i < size; i++)  cout << arr[i] << "\t";  cout << endl;  }  void binary\_search(int arr[], int size) {  int key;    3.  break;  case 2:  obj.selection\_sort(temp, n);  break;  case 3:  obj.selection\_sort(temp, n); // Ensuring sorted before binary search  obj.binary\_search(temp, n);  break;  case 4:  cout << "Current Array:\n";  for (int i = 0; i < n; i++)  cout << arr[i] << "\t";  cout << endl;  break;  case 5:  cout << "Exiting... Khuda Hafiz ";  break;  default:  cout << "Invalid choice! Try again.\n";  }  } while (choice != 5);  return 0;  } | 2.  cout << "Enter the element to search: ";  cin >> key;  int low = 0, high = size - 1, mid;  while (low <= high) {  mid = (low + high) / 2;  if (arr[mid] == key) {  cout << "Element found at index " << mid << endl;  return;  } else if (key > arr[mid]) {  low = mid + 1;  } else {  high = mid - 1;  }  }  cout << "Element not found.\n";  }  };  int main() {  int n, choice;  cout << "Enter size of array: ";  cin >> n;  int arr[n], temp[n];  cout << "Enter elements: ";  for (int i = 0; i < n; i++)  cin >> arr[i];  ArrayOperations obj;  do {  cout << "\n------- MENU -------\n";  cout << "1. Bubble Sort\n";  cout << "2. Selection Sort\n";  cout << "3. Binary Search\n";  cout << "4. Display Array\n";  cout << "5. Exit\n";  cout << "Enter your choice: ";  cin >> choice;  // Copying original array to temp each time  for (int i = 0; i < n; i++) temp[i] = arr[i];  switch (choice) {  case 1:  obj.bubble\_sort(temp, n); |

**Week2**

1. **Write a program to implement insertion sort.**
2. **Write a program to implement quick sort.**
3. **Write a program to implement merge sort.**
4. **Write a program to add two sparse matrices.**
5. **Write a program to multiply two sparse matrices.**

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;  class SortingAndSparseMatrix {  public:  // Bubble Sort  void bubbleSort(int arr[], int n) {  for(int i = 0; i < n - 1; i++) {  for(int j = 0; j < n - i - 1; j++) {  if(arr[j] > arr[j+1]) {  swap(arr[j], arr[j+1]);  }  }  }  }  // Insertion Sort  void insertionSort(int arr[], int n) {  for(int i = 1; i < n; i++) {  int key = arr[i];  int j = i - 1;  while(j >= 0 && arr[j] > key) {  arr[j + 1] = arr[j];  j--;  }  arr[j + 1] = key;  }  }  // Quick Sort Helpers  void quickSort(int arr[], int low, int high) {  if(low < high) {  int pi = partition(arr, low, high);  quickSort(arr, low, pi - 1);  quickSort(arr, pi + 1, high);  }  }  int partition(int arr[], int low, int high) {  int pivot = arr[high];  int i = (low - 1);  for(int j = low; j < high; j++) {  if(arr[j] < pivot) {  i++;  2  swap(arr[i], arr[j]);  }  }  swap(arr[i + 1], arr[high]);  return (i + 1);  }  // Merge Sort Helpers  void merge(int arr[], int l, int m, int r) {  int n1 = m - l + 1;  int n2 = r - m;  int L[n1], R[n2];  for(int i = 0; i < n1; i++) L[i] = arr[l + i];  for(int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];  int i = 0, j = 0, k = l;  while(i < n1 && j < n2) {  if(L[i] <= R[j]) arr[k++] = L[i++];  else arr[k++] = R[j++];  }  while(i < n1) arr[k++] = L[i++];  while(j < n2) arr[k++] = R[j++];  }  void mergeSort(int arr[], int l, int r) {  if(l < r) {  int m = l + (r - l) / 2;  mergeSort(arr, l, m);  mergeSort(arr, m + 1, r);  merge(arr, l, m, r);  }  }  // Display Array  void display(int arr[], int n) {  for(int i = 0; i < n; i++) {  cout << arr[i] << " ";  }  cout << endl;  }  // Sparse Matrix Addition  void addSparse(int mat1[][3], int mat2[][3]) {  int result[100][3], k = 1;  if(mat1[0][0] != mat2[0][0] || mat1[0][1] != mat2[0][1]) {  cout << "Matrix dimensions do not match!\n";  return;  }  4  cout << "Enter number of elements for sorting: ";  cin >> n;  cout << "Enter array elements: ";  for(int i = 0; i < n; i++) cin >> arr[i];  do {  cout << "\nMenu:\n";  cout << "1. Bubble Sort\n";  cout << "2. Insertion Sort\n";  cout << "3. Quick Sort\n";  cout << "4. Merge Sort\n";  cout << "5. Add Two Sparse Matrices\n";  cout << "6. Multiply Two Sparse Matrices\n";  cout << "0. Exit\n";  cout << "Enter your choice: ";  cin >> choice;  switch(choice) {  case 1:  program.bubbleSort(arr, n);  cout << "Sorted Array: ";  program.display(arr, n);  break;  case 2:  program.insertionSort(arr, n);  cout << "Sorted Array: ";  program.display(arr, n);  break;  case 3:  program.quickSort(arr, 0, n - 1);  cout << "Sorted Array: ";  program.display(arr, n);  break; | 3  int i = 1, j = 1;  result[0][0] = mat1[0][0];  result[0][1] = mat1[0][1];  while(i <= mat1[0][2] && j <= mat2[0][2]) {  if(mat1[i][0] == mat2[j][0] && mat1[i][1] == mat2[j][1]) {  result[k][0] = mat1[i][0];  result[k][1] = mat1[i][1];  result[k][2] = mat1[i][2] + mat2[j][2];  i++; j++; k++;  }  else if((mat1[i][0] < mat2[j][0]) ||  (mat1[i][0] == mat2[j][0] && mat1[i][1] < mat2[j][1])) {  result[k][0] = mat1[i][0];  result[k][1] = mat1[i][1];  result[k][2] = mat1[i][2];  i++; k++;  }  else {  result[k][0] = mat2[j][0];  result[k][1] = mat2[j][1];  result[k][2] = mat2[j][2];  j++; k++;  }  }  while(i <= mat1[0][2]) result[k][0] = mat1[i][0], result[k][1] = mat1[i][1], result[k][2] = mat1[i][2], i++, k++;  while(j <= mat2[0][2]) result[k][0] = mat2[j][0], result[k][1] = mat2[j][1], result[k][2] = mat2[j][2], j++, k++;  result[0][2] = k - 1;  cout << "Resultant Sparse Matrix (Triplet Form):\n";  for(int x = 0; x < k; x++) {  cout << result[x][0] << " " << result[x][1] << " " << result[x][2] << endl;  }  }  // Sparse Matrix Multiplication - Placeholder  void multiplySparse(int mat1[][3], int mat2[][3]) {  cout << "Sparse matrix multiplication is complex in triplet form.\n";  cout << "Consider converting to full matrix format before multiplication.\n";  }  };  int main() {  SortingAndSparseMatrix program;  int choice, n, arr[100];  5  case 4:  program.mergeSort(arr, 0, n - 1);  cout << "Sorted Array: ";  program.display(arr, n);  break;  case 5: {  int mat1[10][3] = {{3,3,3},{0,0,1},{1,2,2},{2,1,3}};  int mat2[10][3] = {{3,3,2},{0,0,2},{1,2,4}};  program.addSparse(mat1, mat2);  break;  }  case 6: {  int mat1[10][3] = {{2,2,2},{0,1,3},{1,0,5}};  int mat2[10][3] = {{2,2,2},{0,1,4},{1,0,6}};  program.multiplySparse(mat1, mat2);  break;  }  case 0:  cout << "Exiting...\n";  break;  default:  cout << "Invalid choice!\n";  }  } while(choice != 0);  return 0;  } |
|  |  |



**Week3**

1. **Write a program to implement singly linked list.**
2. **Write a program to implement different operations like adding a node at beginning, end, center, after a certain element, after a certain count of nodes in a linked list.**
3. **Write a program to implement different operations like deleting a node at beginning, end, center, after a certain element, after a certain count of nodes in a linked list.**
4. **Write a program in C++ to reverse a linked list by changing the link in the nodes.**

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  struct Node {  int data;  Node\* next;  };  Node\* head = NULL;  // Function to create a new node  Node\* createNode(int val) {  Node\* newNode = new Node();  newNode->data = val;  newNode->next = NULL;  return newNode;  }  void insertAtBeginning(int val) {  Node\* newNode = createNode(val);  newNode->next = head;  head = newNode;  }  void insertAtEnd(int val) {  Node\* newNode = createNode(val);  if (head == NULL) {  head = newNode;  return;  }  Node\* temp = head;  while (temp->next != NULL)  temp = temp->next;  temp->next = newNode;  }  void insertAfterElement(int key, int val) {  Node\* temp = head;  while (temp != NULL && temp->data != key)  temp = temp->next;  if (temp == NULL) {  cout << "Element not found.\n";  return;  }  Node\* newNode = createNode(val);  newNode->next = temp->next;  temp->next = newNode;  }  void insertAtPosition(int pos, int val) {  if (pos == 0) {  insertAtBeginning(val); | return;  }  Node\* temp = head;  for (int i = 1; i < pos && temp != NULL; i++)  temp = temp->next;  if (temp == NULL) {  cout << "Position out of bounds.\n";  return;  }  Node\* newNode = createNode(val);  newNode->next = temp->next;  temp->next = newNode;  }  void deleteByValue(int key) {  if (head == NULL) return;  if (head->data == key) {  Node\* temp = head;  head = head->next;  delete temp;  return;  }  Node\* temp = head;  while (temp->next != NULL && temp->next->data != key)  temp = temp->next;  if (temp->next == NULL) {  cout << "Element not found.\n";  return;  }  Node\* toDelete = temp->next;  temp->next = temp->next->next;  delete toDelete;  }  void reverseList() {  Node\* prev = NULL;  Node\* current = head;  Node\* next = NULL;  while (current != NULL) {  next = current->next;  current->next = prev;  prev = current;  current = next;  }  head = prev;  }  void display() {  Node\* temp = head;  if (!temp) {  cout << "List is empty.\n";  return;  } |

while (temp != NULL) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL\n";

}

int main() {

int choice, val, pos, key;

do {

cout << "\n--- Singly Linked List Operations ---\n";

cout << "1. Insert at Beginning\n2. Insert at End\n3. Insert After Element\n4. Insert at Position\n";

cout << "5. Delete Node by Value\n6. Reverse List\n7. Display List\n8. Exit\n";

cin >> choice;

switch (choice) {

case 1: cout << "Enter value: "; cin >> val; insertAtBeginning(val); break;

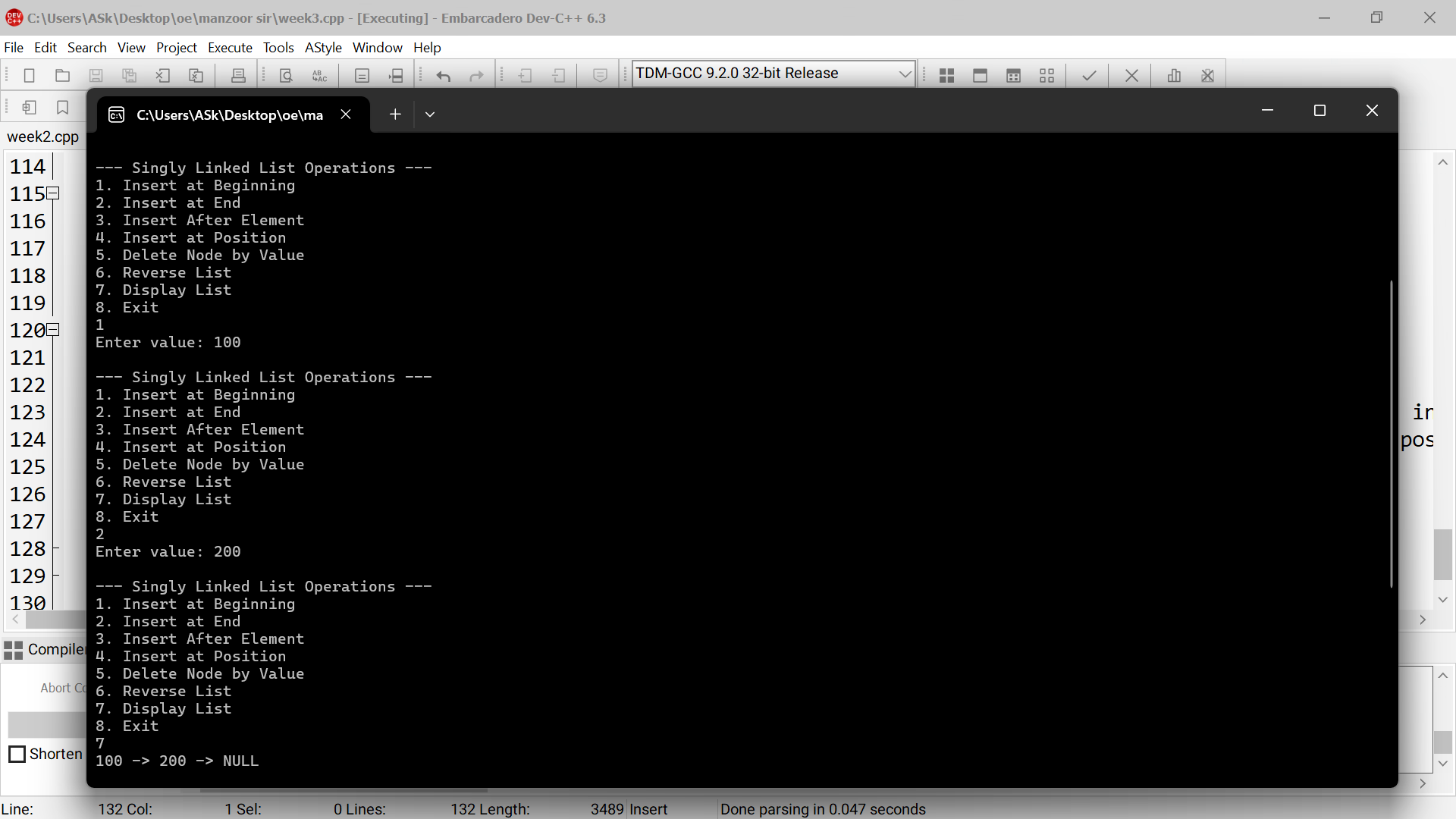
case 2: cout << "Enter value: "; cin >> val; insertAtEnd(val); break;

case 3: cout << "Enter existing element: "; cin >> key; cout << "Enter value to insert: "; cin >> val; insertAfterElement(key, val); break;

case 4: cout << "Enter position: "; cin >> pos; cout << "Enter value: "; cin >> val; insertAtPosition(pos, val); break;

case 5: cout << "Enter value to delete: "; cin >> key; deleteByValue(key); break;

case 6: reverseList(); break;

 case 7: display(); break;

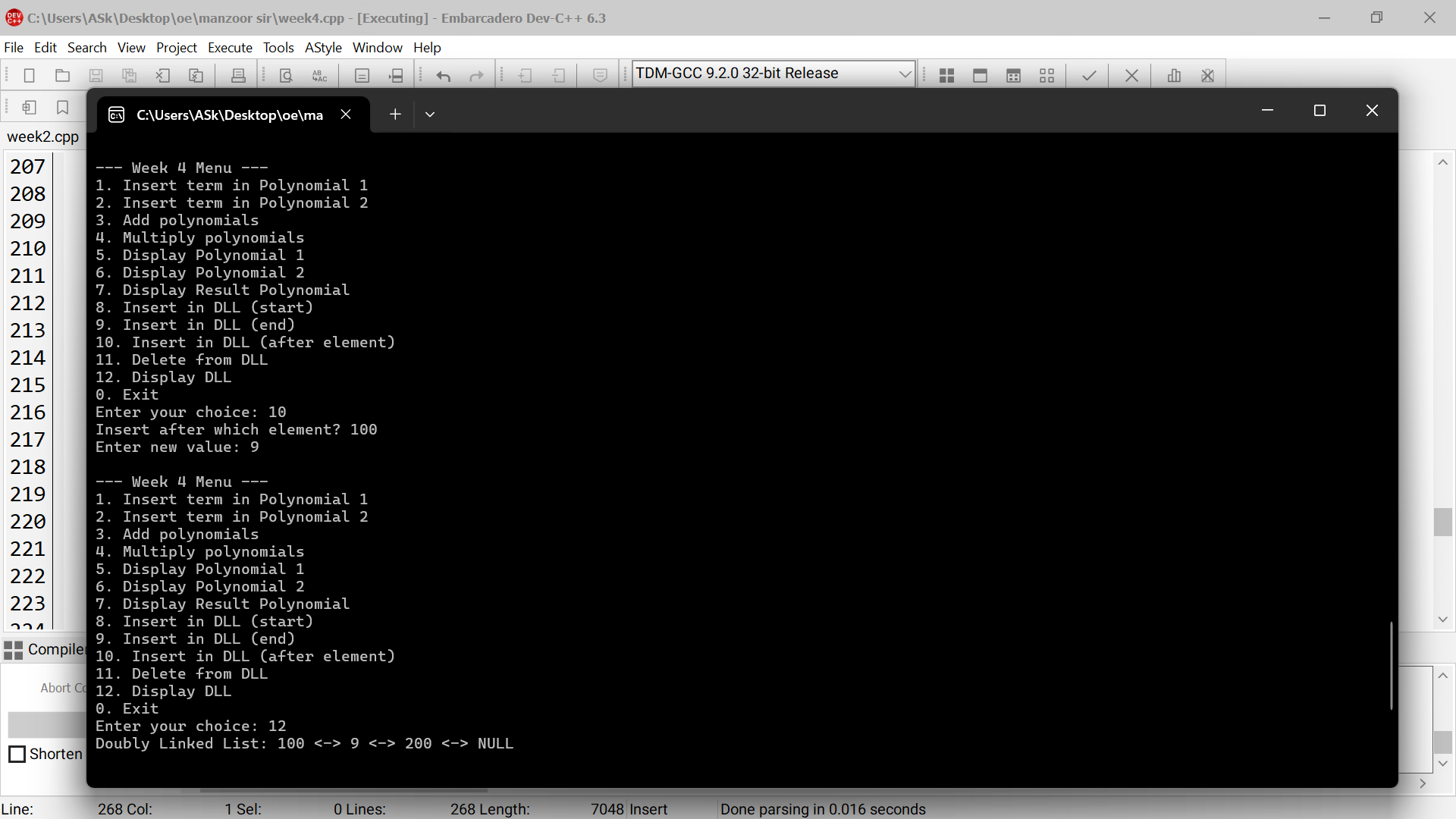
}

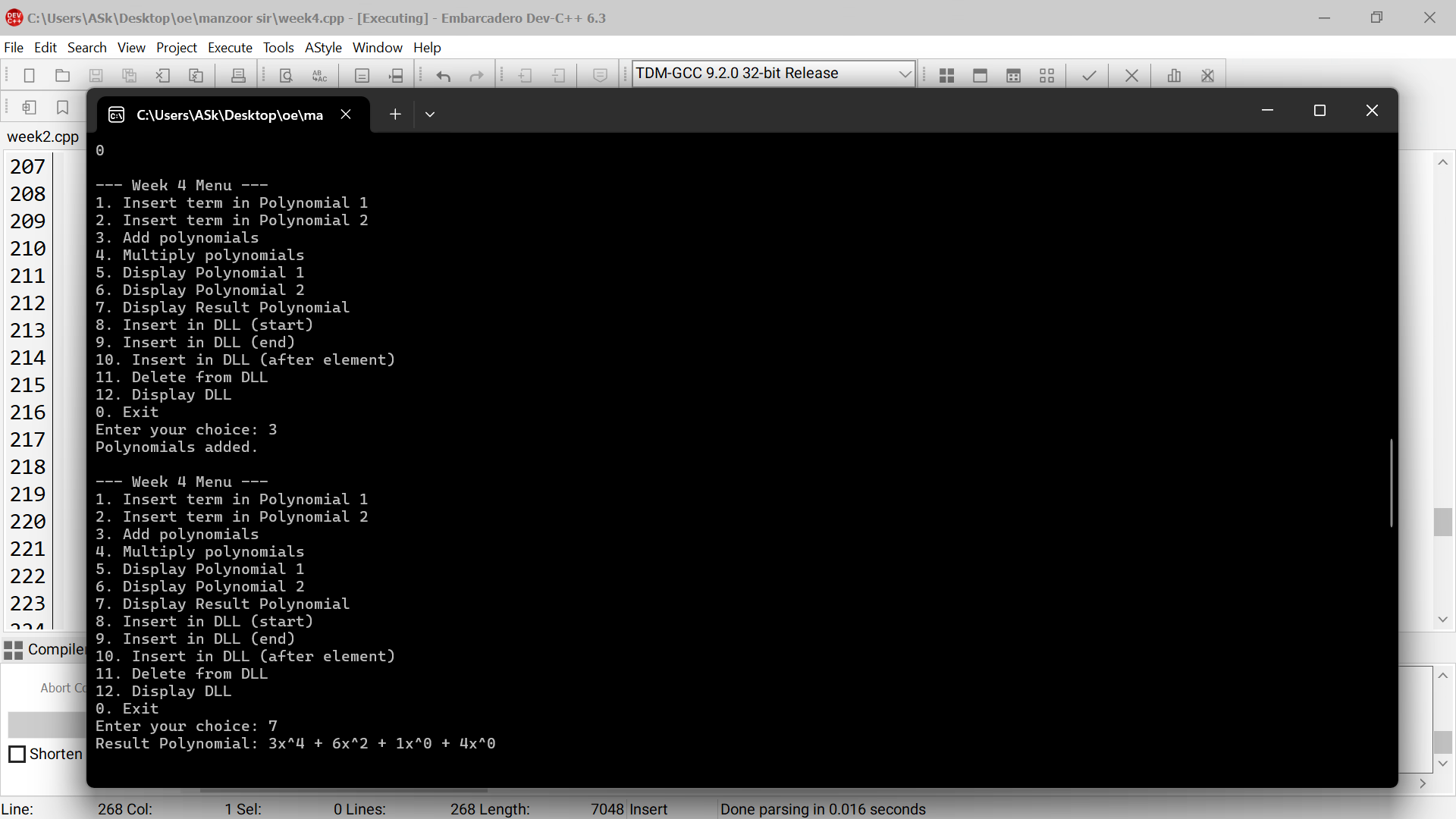
} while (choice != 8);

return 0}

**Week4**

1. **Write a program to add two polynomials represented as linked list.**
2. **Write a program in C++ to multiply two polynomials represented as linked list.**
3. **Write a program in C++ to implement a doubly linked list.**
4. **Write a program to implement different operations like adding a node at beginning, end, center, after a certain element, after a certain count of nodes in a doubly linked list.**
5. **Write a program to implement different operations like deleting a node at beginning, end, center, after a certain element, after a certain count of nodes in a doubly linked list.**



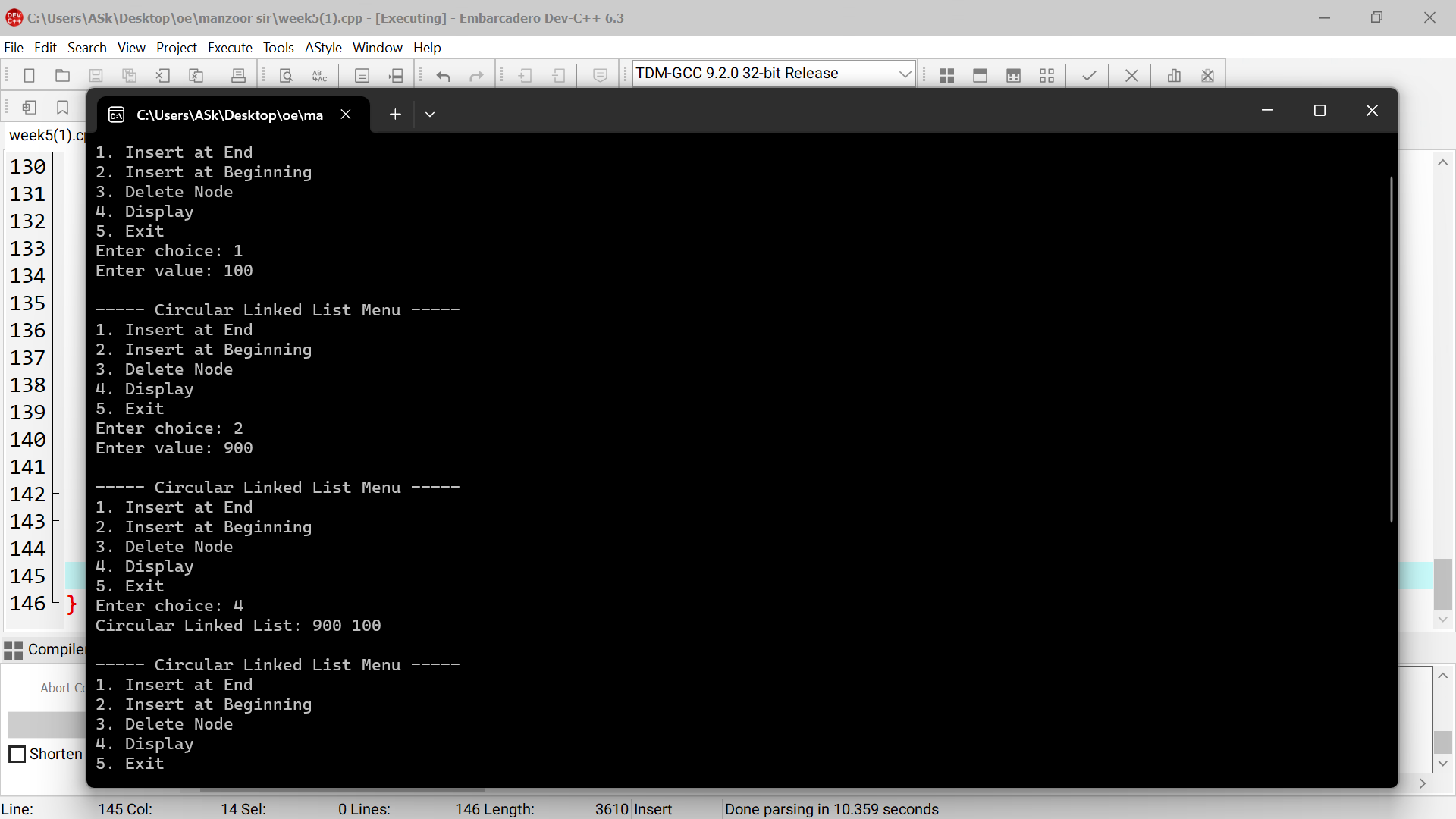


|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  // Polynomial Node  class PolyNode {  public:  int coeff, power;  PolyNode\* next;  PolyNode(int c, int p) {  coeff = c;  power = p;  next = nullptr;  }  };  // Polynomial Class  class Polynomial {  public:  PolyNode\* head;  Polynomial() {  head = nullptr;  }  void insertTerm(int coeff, int power) {  PolyNode\* newNode = new PolyNode(coeff, power);  if (!head || head->power < power) {  newNode->next = head;  head = newNode;  } else {  PolyNode\* temp = head;  while (temp->next && temp->next->power > power)  temp = temp->next;  if (temp->power == power) {  temp->coeff += coeff;  } else {  newNode->next = temp->next;  temp->next = newNode;  }  }  }  void display() {  PolyNode\* temp = head;  3  }  return result;  }  };  // Doubly Linked List Node  class DNode {  public:  int data;  DNode\* prev;  DNode\* next;  DNode(int val) {  data = val;  prev = next = nullptr;  }  };  // Doubly Linked List Class  class DoublyLinkedList {  public:  DNode\* head;  DoublyLinkedList() {  head = nullptr;  }  void insertAtStart(int val) {  DNode\* newNode = new DNode(val);  if (head) head->prev = newNode;  newNode->next = head;  head = newNode;  }  void insertAtEnd(int val) {  DNode\* newNode = new DNode(val);  if (!head) {  head = newNode;  return;  }  DNode\* temp = head;  while (temp->next) temp = temp->next;  temp->next = newNode;  newNode->prev = temp;  }  void insertAfterElement(int key, int val) {  DNode\* temp = head;  5  cout << "5. Display Polynomial 1\n";  cout << "6. Display Polynomial 2\n";  cout << "7. Display Result Polynomial\n";  cout << "8. Insert in DLL (start)\n";  cout << "9. Insert in DLL (end)\n";  cout << "10. Insert in DLL (after element)\n";  cout << "11. Delete from DLL\n";  cout << "12. Display DLL\n";  cout << "0. Exit\n";  cout << "Enter your choice: ";  cin >> choice;  int coeff, power, val, key;  switch (choice) {  case 1:  cout << "Enter coefficient and power: ";  cin >> coeff >> power;  p1.insertTerm(coeff, power);  break;  case 2:  cout << "Enter coefficient and power: ";  cin >> coeff >> power;  p2.insertTerm(coeff, power);  break;  case 3:  result = Polynomial::add(p1, p2);  cout << "Polynomials added.\n";  break;  case 4:  result = Polynomial::multiply(p1, p2);  cout << "Polynomials multiplied.\n";  break;  case 5:  cout << "Polynomial 1: ";  p1.display();  break;  case 6:  cout << "Polynomial 2: ";  p2.display();  break;  case 7:  cout << "Result Polynomial: ";  result.display();  break; | 2  while (temp) {  cout << temp->coeff << "x^" << temp->power;  if (temp->next) cout << " + ";  temp = temp->next;  }  cout << endl;  }  static Polynomial add(Polynomial& p1, Polynomial& p2) {  Polynomial result;  PolyNode \*a = p1.head, \*b = p2.head;  while (a && b) {  if (a->power > b->power) {  result.insertTerm(a->coeff, a->power);  a = a->next;  } else if (a->power < b->power) {  result.insertTerm(b->coeff, b->power);  b = b->next;  } else {  result.insertTerm(a->coeff + b->coeff, a->power);  a = a->next;  b = b->next;  }  }  while (a) {  result.insertTerm(a->coeff, a->power);  a = a->next;  }  while (b) {  result.insertTerm(b->coeff, b->power);  b = b->next;  }  return result;  }  static Polynomial multiply(Polynomial& p1, Polynomial& p2) {  Polynomial result;  for (PolyNode\* a = p1.head; a; a = a->next) {  for (PolyNode\* b = p2.head; b; b = b->next) {  result.insertTerm(a->coeff \* b->coeff, a->power + b->power);  }  4  while (temp && temp->data != key) temp = temp->next;  if (!temp) {  cout << "Element not found.\n";  return;  }  DNode\* newNode = new DNode(val);  newNode->next = temp->next;  if (temp->next) temp->next->prev = newNode;  temp->next = newNode;  newNode->prev = temp;  }  void deleteNode(int key) {  DNode\* temp = head;  while (temp && temp->data != key) temp = temp->next;  if (!temp) {  cout << "Node not found.\n";  return;  }  if (temp->prev) temp->prev->next = temp->next;  else head = temp->next;  if (temp->next) temp->next->prev = temp->prev;  delete temp;  }  void display() {  DNode\* temp = head;  while (temp) {  cout << temp->data << " <-> ";  temp = temp->next;  }  cout << "NULL\n";  }  };  // Main Menu  int main() {  Polynomial p1, p2, result;  DoublyLinkedList dll;  int choice;  do {  cout << "\n--- Week 4 Menu ---\n";  cout << "1. Insert term in Polynomial 1\n";  cout << "2. Insert term in Polynomial 2\n";  cout << "3. Add polynomials\n";  cout << "4. Multiply polynomials\n";  6  case 8:  cout << "Enter value: ";  cin >> val;  dll.insertAtStart(val);  break;  case 9:  cout << "Enter value: ";  cin >> val;  dll.insertAtEnd(val);  break;  case 10:  cout << "Insert after which element? ";  cin >> key;  cout << "Enter new value: ";  cin >> val;  dll.insertAfterElement(key, val);  break;  case 11:  cout << "Enter value to delete: ";  cin >> key;  dll.deleteNode(key);  break;  case 12:  cout << "Doubly Linked List: ";  dll.display();  break;  case 0:  cout << "Exiting program.\n";  break;  default:  cout << "Invalid choice.\n";  }  } while (choice != 0);  return 0;  } |

**Week5**

1. **Write a program to implement different operations of a circular linked list.**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  class Node {  public:  int data;  Node\* next;  Node(int val) {  data = val;  next = nullptr;  }  };  class CircularLinkedList {  private:  Node\* first;  Node\* last;  public:  CircularLinkedList() {  first = last = nullptr;  }  void insertAtEnd(int val) {  Node\* newNode = new Node(val);  if (!first) {  first = last = newNode;  last->next = first;  } else {  last->next = newNode;  last = newNode;  last->next = first;  }  }  void insertAtBeginning(int val) {  Node\* newNode = new Node(val);  if (!first) {  first = last = newNode;  3  prev = curr;  curr = curr->next;  } while (curr != first);  if (!found)  cout << "Node not found\n";  }  void display() {  if (!first) {  cout << "List is empty\n";  return;  }  Node\* temp = first;  cout << "Circular Linked List: ";  do {  cout << temp->data << " ";  temp = temp->next;  } while (temp != first);  cout << endl;  }  };  int main() {  CircularLinkedList cll;  int choice, val;  do {  cout << "\n----- Circular Linked List Menu -----\n";  cout << "1. Insert at End\n";  cout << "2. Insert at Beginning\n";  cout << "3. Delete Node\n";  cout << "4. Display\n";  cout << "5. Exit\n";  cout << "Enter choice: ";  cin >> choice;  switch (choice) {  case 1:  cout << "Enter value: "; | 2  last->next = first;  } else {  newNode->next = first;  first = newNode;  last->next = first;  }  }  void deleteNode(int val) {  if (!first) {  cout << "List is empty\n";  return;  }  Node\* curr = first;  Node\* prev = last;  bool found = false;  do {  if (curr->data == val) {  found = true;  if (first == last && curr == first) { // Only one node  delete curr;  first = last = nullptr;  } else if (curr == first) {  first = first->next;  last->next = first;  delete curr;  } else if (curr == last) {  prev->next = first;  last = prev;  delete curr;  } else {  prev->next = curr->next;  delete curr;  }  cout << "Node deleted\n";  break;  }  4  cin >> val;  cll.insertAtEnd(val);  break;  case 2:  cout << "Enter value: ";  cin >> val;  cll.insertAtBeginning(val);  break;  case 3:  cout << "Enter value to delete: ";  cin >> val;  cll.deleteNode(val);  break;  case 4:  cll.display();  break;  case 5:  cout << "Exiting...\n";  break;  default:  cout << "Invalid choice!\n";  }  } while (choice != 5);  return 0;  } |



1. **Write a program to implement various operations on an array based stack?**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define MAX 100  class Stack {  private:  int arr[MAX];  int top;  public:  Stack() {  top = -1;  }  bool isFull() {  return top == MAX - 1;  }  bool isEmpty() {  return top == -1;  }  void push(int value) {  3  switch (choice) {  case 1:  cout << "Enter value to push: ";  cin >> value;  s.push(value);  break;  case 2:  s.pop();  break;  case 3:  s.peek();  break;  case 4:  s.display();  break;  case 5:  cout << "Exiting program.\n";  break;  default:  cout << "Invalid choice! Please try again.\n";  }  } while (choice != 5);  return 0;  } | 2  for (int i = top; i >= 0; i--) {  cout << arr[i] << " ";  }  cout << endl;  }  };  int main() {  Stack s;  int choice, value;  do {  cout << "\n--- Array-Based Stack Menu ---\n";  cout << "1. Push\n2. Pop\n3. Peek\n4. Display\n5. Exit\n";  cout << "Enter your choice: ";  cin >> choice; |

**3. Write a program to implement various operations on a stack represented using linked list.**

|  |  |
| --- | --- |
| 1  #include<iostream>  using namespace std;  class Node  {  public:  int data;  Node\* next;    Node(int item)  {  next=nullptr;  data=item;  }  };  class stack  {  private:  Node\* top;  public:  stack()  {  top=nullptr;  }    ~stack()  {  while(top!=nullptr)  {  pop();  }  }    void push()  {  int item;  cout<<"Enter the element:"<<endl;  cin>>item;  Node\* newnode=new Node(item);  newnode->next=top;  top=newnode;  3  cout << "\nStack elements are:\n";  Node\* current = top;  while (current != nullptr)  {  cout << current->data << endl;  current = current->next;  }  }    };  int main()  {  stack s;  int option;  do {  cout << "\n----- Stack Menu -----";  cout << "\n1. Push";  cout << "\n2. Pop";  cout << "\n3. Display";  cout << "\n4. Peek";  cout << "\n5. Exit";  cout << "\nEnter your choice: ";  cin >> option;  switch (option) {  case 1:  s.push();  break;  case 2:  s.pop();  break;  case 3:  s.show();  break;  case 4:  s.peek();  break;  case 5:  cout << "Exiting...\n";  break; | 2  cout << item << " pushed onto the stack.\n";      }  void pop()  {  if (top == nullptr)  {  cout << "\nStack is empty. Cannot pop.\n";  return;  }    Node\* temp = top;  cout << "\nDeleted element is: " << temp->data << endl;  top = top->next;  delete temp;    }    void peek()  {  if (top == nullptr)  {  cout << "\nStack is empty. Nothing to peek.\n";  } else  {  cout << "\nTop element is: " << top->data << endl;  }  }  void show()  {  if (top == nullptr)  {  cout << "\nStack is empty.\n";  return;  }  4  default:  cout << "\nInvalid choice. Enter a valid  option.\n";  }  } while (option != 5);  return 0;  } |

**Week7**

**1. Write a program to demonstrate the use of stack in implementing quicksort algorithm to sort an array of integers in ascending order.**

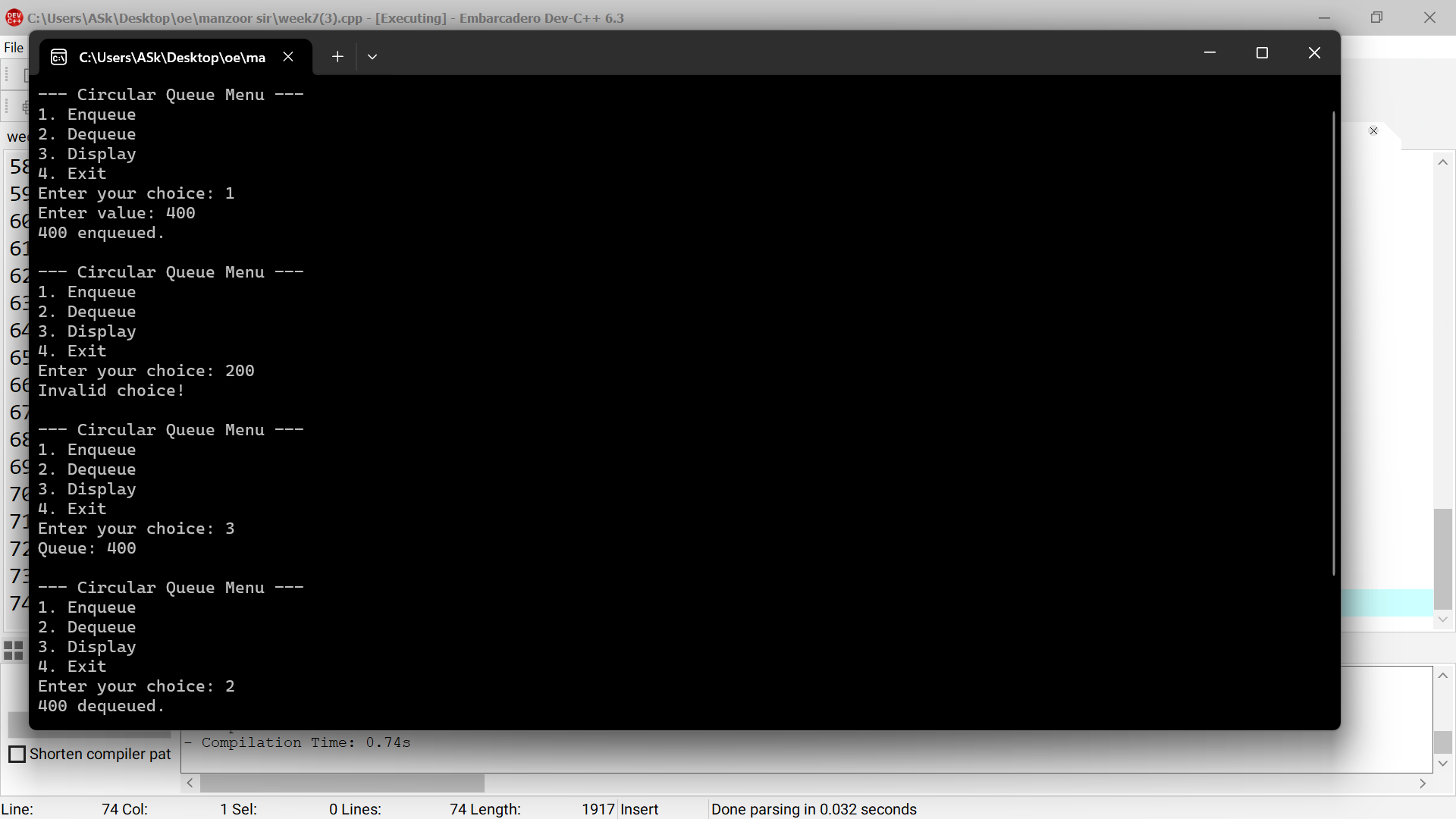
|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define MAX 100  class Stack {  private:  int top;  int data[MAX];  public:  Stack() { top = -1; }  bool isEmpty() { return top == -1; }  void push(int val) {  if (top == MAX - 1) return;  data[++top] = val;  }  int pop() {  if (isEmpty()) return -1;  return data[top--];  }  };  class QuickSortStack {  private:  int arr[MAX], n;  int partition(int low, int high) {  int pivot = arr[high];  int i = low - 1;  for (int j = low; j < high; j++) {  if (arr[j] <= pivot) {  i++;  int t = arr[i]; arr[i] = arr[j]; arr[j] = t;  }  }  int t = arr[i+1]; arr[i+1] = arr[high]; arr[high] = t;  3  cout << "\n--- Stack-Based QuickSort Menu ---\n";  cout << "1. Enter Array\n2. Sort\n3. Exit\nEnter your choice: ";  cin >> choice;  switch (choice) {  case 1: q.inputArray(); break;  case 2: q.sort(); break;  case 3: cout << "Exiting...\n"; break;  default: cout << "Invalid choice!\n";  }  } while (choice != 3);  return 0;  } | 2  return i + 1;  }  public:  void inputArray() {  cout << "Enter number of elements: ";  cin >> n;  cout << "Enter " << n << " elements: ";  for (int i = 0; i < n; i++) {  cin >> arr[i];  }  }  void sort() {  Stack lStack, hStack;  lStack.push(0);  hStack.push(n - 1);  while (!lStack.isEmpty()) {  int high = hStack.pop();  int low = lStack.pop();  int p = partition(low, high);  if (p - 1 > low) {  lStack.push(low);  hStack.push(p - 1);  }  if (p + 1 < high) {  lStack.push(p + 1);  hStack.push(high);  }  }  cout << "Sorted array: ";  for (int i = 0; i < n; i++) cout << arr[i] << " ";  cout << endl;  }  };  int main() {  QuickSortStack q;  int choice;  do {  4 |

**2. Write a program to demonstrate the implementation of various operations on a linear queue represented using a linear array**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define MAX 100  class LinearQueue {  private:  int arr[MAX], front, rear;  public:  LinearQueue() {  front = rear = -1;  }  void enqueue(int val) {  if (rear == MAX - 1) {  cout << "Queue Overflow!\n";  return;  }  if (front == -1) front = 0;  arr[++rear] = val;  cout << val << " enqueued.\n";  }  3  int main() {  LinearQueue q;  int choice, val;  do {  cout << "\n--- Linear Queue Menu ---\n";  cout << "1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter your choice: ";  cin >> choice;  switch (choice) {  case 1: cout << "Enter value: "; cin >> val; q.enqueue(val); break;  case 2: q.dequeue(); break;  case 3: q.display(); break;  case 4: cout << "Exiting...\n"; break;  default: cout << "Invalid choice!\n";  }  } while (choice != 4);  return 0;  } | **2**  void dequeue() {  if (front == -1 || front > rear) {  cout << "Queue Underflow!\n";  return;  }  cout << arr[front++] << " dequeued.\n";  if (front > rear) front = rear = -1;  }  void display() {  if (front == -1) {  cout << "Queue is empty.\n";  return;  }  cout << "Queue: ";  for (int i = front; i <= rear; i++)  cout << arr[i] << " ";  cout << endl;  }  }; |

**3. Write a program to demonstrate the implementation of various operations on a Circular queue represented using a linear array.**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define MAX 5  class CircularQueue {  private:  int arr[MAX], front, rear;  public:  CircularQueue() {  front = rear = -1;  }  void enqueue(int val) {  if ((front == 0 && rear == MAX - 1) || (rear + 1 == front)) {  cout << "Queue Overflow!\n";  return;  }  if (front == -1) front = rear = 0;  else if (rear == MAX - 1) rear = 0;  else rear++;  arr[rear] = val;  cout << val << " enqueued.\n";  }  void dequeue() {  if (front == -1) {  cout << "Queue Underflow!\n";  return;  }  cout << arr[front] << " dequeued.\n";  if (front == rear) front = rear = -1;  else if (front == MAX - 1) front = 0;  else front++;  }  void display() {  if (front == -1) { | 2  cout << "Queue is empty.\n";  return;  }  cout << "Queue: ";  if (front <= rear) {  for (int i = front; i <= rear; i++)  cout << arr[i] << " ";  } else {  for (int i = front; i < MAX; i++) cout << arr[i] << " ";  for (int i = 0; i <= rear; i++) cout << arr[i] << " ";  }  cout << endl;  }  };  int main() {  CircularQueue cq;  int choice, val;  do {  cout << "\n--- Circular Queue Menu ---\n";  cout << "1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter your choice: ";  cin >> choice;  switch (choice) {  case 1: cout << "Enter value: "; cin >> val; cq.enqueue(val); break;  case 2: cq.dequeue(); break;  case 3: cq.display(); break;  case 4: cout << "Exiting...\n"; break;  default: cout << "Invalid choice!\n";  }  } while (choice != 4);  return 0;  } |

****

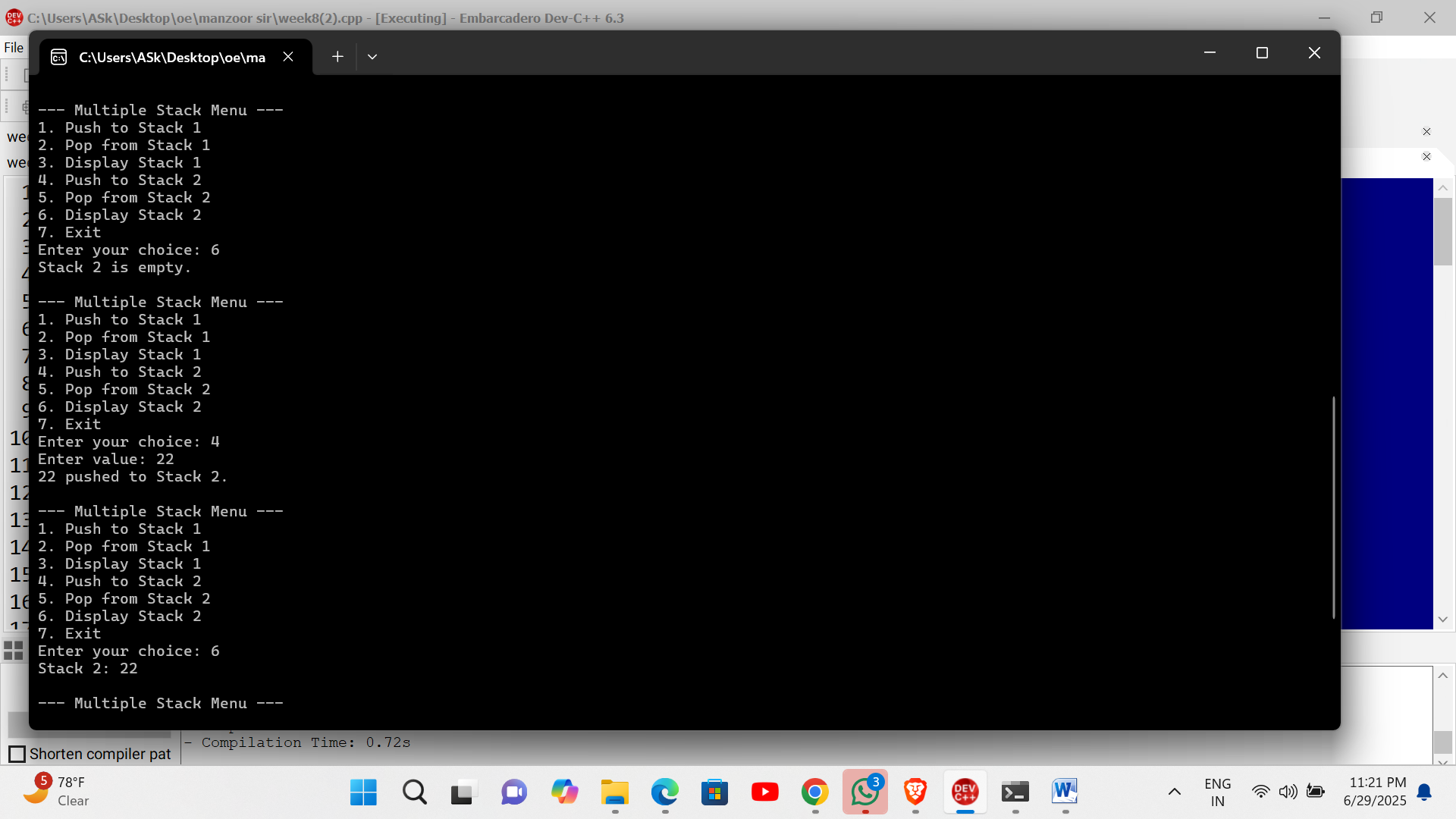
**Week8**

**1. Write a program to demonstrate the implementation of various operations on a queue represented using a linked list?**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  class Node {  public:  int data;  Node\* next;  Node(int val) {  data = val;  next = NULL;  }  };  class LinkedQueue {  private:  Node\* front;  Node\* rear;  public:  LinkedQueue() {  front = rear = NULL;  }  void enqueue(int val) {  Node\* temp = new Node(val);  if (rear == NULL) {  front = rear = temp;  } else {  rear->next = temp;  rear = temp;  }  cout << val << " enqueued.\n";  }  void dequeue() {  if (front == NULL) {  cout << "Queue Underflow!\n";  return;  } while (choice != 4);  return 0;  } | 2  }  Node\* temp = front;  cout << front->data << " dequeued.\n";  front = front->next;  delete temp;  if (front == NULL) rear = NULL;  }  void display() {  if (front == NULL) {  cout << "Queue is empty.\n";  return;  }  cout << "Queue: ";  Node\* temp = front;  while (temp != NULL) {  cout << temp->data << " ";  temp = temp->next;  }  cout << endl;  }  };  int main() {  LinkedQueue q;  int choice, val;  do {  cout << "\n--- Linked List Queue Menu ---\n";  cout << "1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter your choice: ";  cin >> choice;  switch (choice) {  case 1: cout << "Enter value: "; cin >> val; q.enqueue(val); break;  case 2: q.dequeue(); break;  case 3: q.display(); break;  case 4: cout << "Exiting...\n"; break;  default: cout << "Invalid choice!\n";  } |

**2. Write a program to demonstrate the use of multiple stacks?**

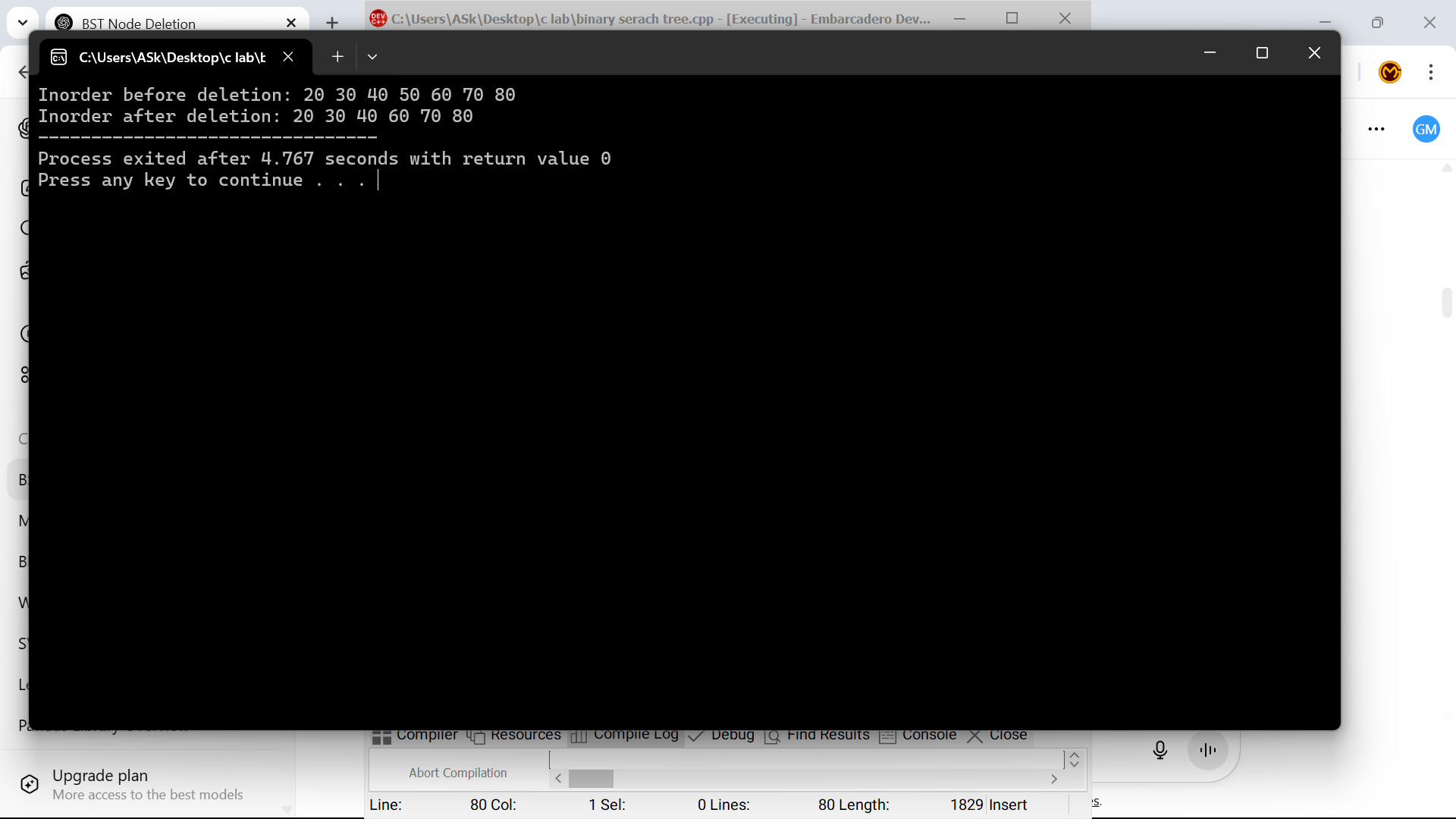
|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define MAX 100  class MultiStack {  private:  int arr[MAX];  int top1, top2;  public:  MultiStack() {  top1 = -1;  top2 = MAX;  }  void push1(int val) {  if (top1 + 1 == top2) {  cout << "Stack Overflow on Stack 1!\n";  return;  }  arr[++top1] = val;  3  void pop2() {  if (top2 == MAX) {  cout << "Stack 2 Underflow!\n";  return;  }  cout << arr[top2++] << " popped from Stack 2.\n";  }  void display1() {  if (top1 == -1) {  cout << "Stack 1 is empty.\n";  return;  }  cout << "Stack 1: ";  for (int i = top1; i >= 0; i--)  cout << arr[i] << " ";  cout << endl;  }  void display2() {  if (top2 == MAX) {  cout << "Stack 2 is empty.\n";  return;  }  cout << "Stack 2: ";  for (int i = top2; i < MAX; i++)  cout << arr[i] << " ";  cout << endl;  }  };  int main() {  MultiStack ms;  int choice, val;  do {  cout << "\n--- Multiple Stack Menu ---\n";  cout << "1. Push to Stack 1\n2. Pop from Stack 1\n";  cout << "3. Display Stack 1\n4. Push to Stack 2\n";  cout << "5. Pop from Stack 2\n6. Display Stack 2\n7. Exit\n";  cout << "Enter your choice: "; | 2  cout << val << " pushed to Stack 1.\n";  }  void push2(int val) {  if (top1 + 1 == top2) {  cout << "Stack Overflow on Stack 2!\n";  return;  }  arr[--top2] = val;  cout << val << " pushed to Stack 2.\n";  }  void pop1() {  if (top1 == -1) {  cout << "Stack 1 Underflow!\n";  return;  }  cout << arr[top1--] << " popped from Stack 1.\n";  }  4  cin >> choice;  switch (choice) {  case 1: cout << "Enter value: "; cin >> val; ms.push1(val); break;  case 2: ms.pop1(); break;  case 3: ms.display1(); break;  case 4: cout << "Enter value: "; cin >> val; ms.push2(val); break;  case 5: ms.pop2(); break;  case 6: ms.display2(); break;  case 7: cout << "Exiting...\n"; break;  default: cout << "Invalid choice!\n";  }  } while (choice != 7);  return 0;  } |

****

**Week9**

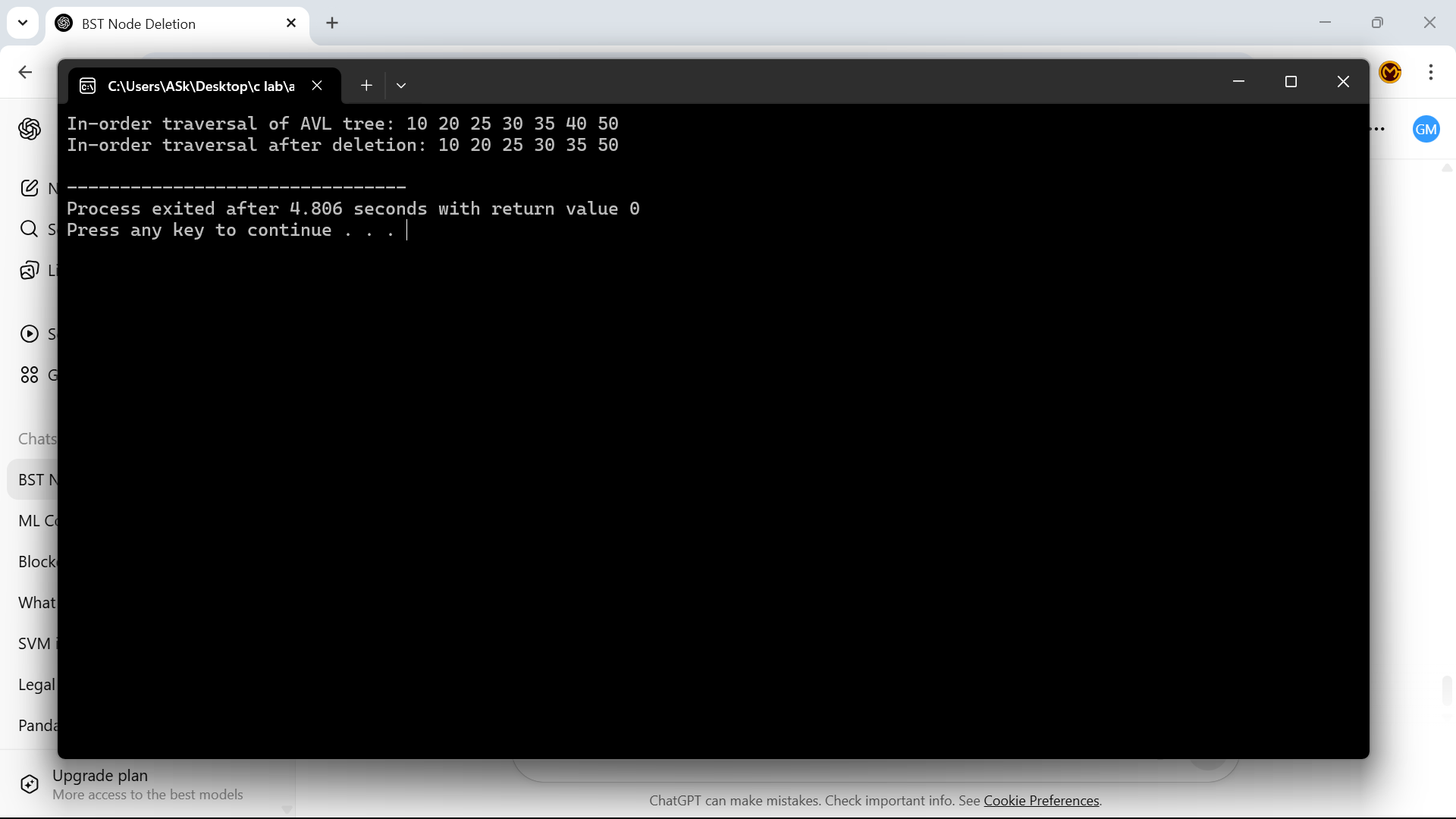
**1. Write a program to delete a node in a binary search tree (BST)**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  struct Node {  int data;  Node\* left;  Node\* right;  };  Node\* newNode(int data) {  Node\* temp = new Node();  temp->data = data;  temp->left = temp->right = nullptr;  return temp;  }  Node\* insert(Node\* root, int key) {  if (!root) return newNode(key);  if (key < root->data) root->left = insert(root->left, key);  else root->right = insert(root->right, key);  return root;  }  Node\* minValueNode(Node\* node) {  Node\* current = node;  while (current && current->left) current = current->left;  return current;  }  Node\* deleteNode(Node\* root, int key) {  if (!root) return root;  if (key < root->data) root->left = deleteNode(root->left, key);  else if (key > root->data) root->right = deleteNode(root->right, key);  else {  if (!root->left) {  Node\* temp = root->right; | **2**  delete root;  return temp;  } else if (!root->right) {  Node\* temp = root->left;  delete root;  return temp;  }  Node\* temp = minValueNode(root->right);  root->data = temp->data;  root->right = deleteNode(root->right, temp->data);  }  return root;  }  void inorder(Node\* root) {  if (root) {  inorder(root->left);  cout << root->data << " ";  inorder(root->right);  }  }  int main() {  Node\* root = nullptr;  root = insert(root, 50);  insert(root, 30);  insert(root, 20);  insert(root, 40);  insert(root, 70);  insert(root, 60);  insert(root, 80);  cout << "Inorder before deletion: ";  inorder(root);  root = deleteNode(root, 50);  cout << "\nInorder after deletion: ";  inorder(root);  return 0;  } |

****

**2. Write a program to implement the different operations of an AVL tree**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  struct Node {  int key, height;  Node\* left;  Node\* right;  };  int height(Node\* n) {  return n ? n->height : 0;  }  int getBalance(Node\* n) {  return n ? height(n->left) - height(n->right) : 0;  }  int max(int a, int b) {  return (a > b) ? a : b;  }  Node\* newNode(int key) {  Node\* node = new Node();  node->key = key;  node->left = node->right = nullptr;  node->height = 1;  return node;  }  Node\* rightRotate(Node\* y) {  Node\* x = y->left;  Node\* T2 = x->right;  // Rotate  x->right = y;  y->left = T2;  // Update heights  y->height = max(height(y->left), height(y->right)) + 1;  x->height = max(height(x->left), height(x->right)) + 1;  return x; // New root  }  Node\* leftRotate(Node\* x) {  Node\* y = x->right;  Node\* T2 = y->left;  // Rotate  y->left = x;  x->right = T2;  3  if (key < root->key)  root->left = deleteNode(root->left, key);  else if (key > root->key)  root->right = deleteNode(root->right, key);  else {  if (!root->left || !root->right) {  Node\* temp = root->left ? root->left : root->right;  if (!temp) {  temp = root;  root = nullptr;  } else  \*root = \*temp;  delete temp;  } else {  Node\* temp = minValueNode(root->right);  root->key = temp->key;  root->right = deleteNode(root->right, temp->key);  }  }  if (!root)  return root;  root->height = 1 + max(height(root->left), height(root->right));  int balance = getBalance(root);  // Balance the tree  if (balance > 1 && getBalance(root->left) >= 0)  return rightRotate(root);  if (balance > 1 && getBalance(root->left) < 0) {  root->left = leftRotate(root->left);  return rightRotate(root);  }  if (balance < -1 && getBalance(root->right) <= 0)  return leftRotate(root);  if (balance < -1 && getBalance(root->right) > 0) {  root->right = rightRotate(root->right);  return leftRotate(root);  }  return root;  }  void inorder(Node\* root) {  if (root) {  inorder(root->left);  cout << root->key << " "; | 2  // Update heights  x->height = max(height(x->left), height(x->right)) + 1;  y->height = max(height(y->left), height(y->right)) + 1;  return y; // New root  }  Node\* insert(Node\* node, int key) {  if (!node)  return newNode(key);  if (key < node->key)  node->left = insert(node->left, key);  else if (key > node->key)  node->right = insert(node->right, key);  else  return node; // No duplicates  node->height = 1 + max(height(node->left), height(node->right));  int balance = getBalance(node);  // Perform rotations  if (balance > 1 && key < node->left->key)  return rightRotate(node);  if (balance < -1 && key > node->right->key)  return leftRotate(node);  if (balance > 1 && key > node->left->key) {  node->left = leftRotate(node->left);  return rightRotate(node);  }  if (balance < -1 && key < node->right->key) {  node->right = rightRotate(node->right);  return leftRotate(node);  }  return node;  }  Node\* minValueNode(Node\* node) {  Node\* current = node;  while (current->left)  current = current->left;  return current;  }  Node\* deleteNode(Node\* root, int key) {  if (!root)  return root;  4  inorder(root->right);  }  }  int main() {  Node\* root = nullptr;  root = insert(root, 30);  root = insert(root, 20);  root = insert(root, 40);  root = insert(root, 10);  root = insert(root, 25);  root = insert(root, 35);  root = insert(root, 50);  cout << "In-order traversal of AVL tree: ";  inorder(root);  cout << "\n";  root = deleteNode(root, 40);  cout << "In-order traversal after deletion: ";  inorder(root);  cout << "\n";  return 0;  } |



**3. C++ Program: In-order Threaded Binary Tree**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  struct Node {  int data;  Node \*left, \*right;  bool lthread, rthread;  };  // Create a new node  Node\* createNode(int key) {  Node\* temp = new Node;  temp->data = key;  temp->lthread = temp->rthread = true;  temp->left = temp->right = nullptr;  return temp;  }  // Insert a node in threaded binary tree  Node\* insert(Node\* root, int key) {  Node\* ptr = root;  Node\* parent = nullptr;  while (ptr != nullptr) {  if (key == ptr->data) {  cout << "Duplicate key!" << endl;  return root;  }  parent = ptr;  if (key < ptr->data) {  if (!ptr->lthread)  ptr = ptr->left;  else  break;  } else {  if (!ptr->rthread)  3  // In-order traversal without recursion or stack  void inorder(Node\* root) {  if (!root) return;  Node\* ptr = root;  while (!ptr->lthread)  ptr = ptr->left;  while (ptr != nullptr) {  cout << ptr->data << " ";  if (ptr->rthread)  ptr = ptr->right;  else {  ptr = ptr->right;  while (ptr && !ptr->lthread)  ptr = ptr->left;  }  }  }  int main() {  Node\* root = nullptr;  root = insert(root, 50);  root = insert(root, 30);  root = insert(root, 70);  root = insert(root, 20);  root = insert(root, 40);  root = insert(root, 60);  root = insert(root, 80);  cout << "In-order traversal of threaded binary tree: ";  inorder(root);  cout << endl;  return 0;  } | 2  ptr = ptr->right;  else  break;  }  }  Node\* newNode = createNode(key);  if (parent == nullptr) {  root = newNode;  } else if (key < parent->data) {  newNode->left = parent->left;  newNode->right = parent;  parent->lthread = false;  parent->left = newNode;  } else {  newNode->left = parent;  newNode->right = parent->right;  parent->rthread = false;  parent->right = newNode;  }  return root;  }  4 |

**4. C++ Program: M-Way Search Tree (M = 4)**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  const int M = 4; // Max children per node  const int MAX\_KEYS = M - 1;  struct MWayNode {  int keys[MAX\_KEYS]; // Up to M-1 keys  MWayNode\* children[M]; // M children  int numKeys; // Current number of keys  bool isLeaf;  MWayNode() {  numKeys = 0;  isLeaf = true;  for (int i = 0; i < M; ++i)  children[i] = nullptr;  }  };  class MWayTree {  MWayNode\* root;  public:  MWayTree() {  root = new MWayNode();  }  void insert(int key) {  root = insertRec(root, key);  }  void inorder() {  inorderRec(root);  cout << endl;  }  bool search(int key) {  return searchRec(root, key);  }  private:  MWayNode\* insertRec(MWayNode\* node, int key) {  int i;  for (i = 0; i < node->numKeys && key > node->keys[i]; i++);  // If key already exists  if (i < node->numKeys && node->keys[i] == key) {  3  int main() {  MWayTree tree;  tree.insert(20);  tree.insert(10);  tree.insert(30);  tree.insert(5);  tree.insert(15);  tree.insert(25);  tree.insert(35);  cout << "In-order Traversal of M-way tree: ";  tree.inorder();  int searchKey = 25;  cout << "Search " << searchKey << ": "  << (tree.search(searchKey) ? "Found" : "Not Found") << endl;  return 0;  } | 2  cout << "Duplicate key not allowed!\n";  return node;  }  if (node->isLeaf) {  if (node->numKeys < MAX\_KEYS) {  for (int j = node->numKeys; j > i; j--)  node->keys[j] = node->keys[j - 1];  node->keys[i] = key;  node->numKeys++;  } else {  cout << "Node full! No balancing implemented.\n";  }  } else {  node->children[i] = insertRec(node->children[i], key);  }  return node;  }  void inorderRec(MWayNode\* node) {  if (!node) return;  for (int i = 0; i < node->numKeys; i++) {  inorderRec(node->children[i]);  cout << node->keys[i] << " ";  }  inorderRec(node->children[node->numKeys]);  }  bool searchRec(MWayNode\* node, int key) {  if (!node) return false;  int i;  for (i = 0; i < node->numKeys && key > node->keys[i]; i++);  if (i < node->numKeys && key == node->keys[i])  return true;  return searchRec(node->children[i], key);  }  }; |

**Week10**

1. **Implement the different operations of a B-tree**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  const int T = 3;  class BTreeNode {  public:  int \*keys;  int t;  BTreeNode \*\*C;  int n;  bool leaf;  BTreeNode(int t1, bool leaf1);  void traverse();  BTreeNode \*search(int k);  void insertNonFull(int k);  void splitChild(int i, BTreeNode \*y);  friend class BTree;  };  class BTree {  public:  BTreeNode \*root;  int t;  BTree(int \_t) {  root = nullptr;  t = \_t;  }  void traverse() {  if (root != nullptr) root->traverse();  }  BTreeNode\* search(int k) {  return (root == nullptr) ? nullptr : root->search(k);  }  void insert(int k);  };  3  int i = 0;  if (s->keys[0] < k)  i++;  s->C[i]->insertNonFull(k);  root = s;  } else  root->insertNonFull(k);  }  }  void BTreeNode::insertNonFull(int k) {  int i = n - 1;  if (leaf) {  while (i >= 0 && keys[i] > k) {  keys[i + 1] = keys[i];  i--;  }  keys[i + 1] = k;  n++;  } else {  while (i >= 0 && keys[i] > k)  i--;  if (C[i + 1]->n == 2 \* t - 1) {  splitChild(i + 1, C[i + 1]);  if (keys[i + 1] < k)  i++;  }  C[i + 1]->insertNonFull(k);  }  }  void BTreeNode::splitChild(int i, BTreeNode \*y) {  BTreeNode \*z = new BTreeNode(y->t, y->leaf);  z->n = t - 1;  for (int j = 0; j < t - 1; j++)  z->keys[j] = y->keys[j + t]; | 2  BTreeNode::BTreeNode(int t1, bool leaf1) {  t = t1;  leaf = leaf1;  keys = new int[2 \* t - 1];  C = new BTreeNode \*[2 \* t];  n = 0;  }  void BTreeNode::traverse() {  int i;  for (i = 0; i < n; i++) {  if (!leaf)  C[i]->traverse();  cout << " " << keys[i];  }  if (!leaf)  C[i]->traverse();  }  BTreeNode\* BTreeNode::search(int k) {  int i = 0;  while (i < n && k > keys[i]) i++;  if (keys[i] == k)  return this;  if (leaf)  return nullptr;  return C[i]->search(k);  }  void BTree::insert(int k) {  if (root == nullptr) {  root = new BTreeNode(t, true);  root->keys[0] = k;  root->n = 1;  } else {  if (root->n == 2 \* t - 1) {  BTreeNode \*s = new BTreeNode(t, false);  s->C[0] = root;  s->splitChild(0, root);  4  if (!y->leaf) {  for (int j = 0; j < t; j++)  z->C[j] = y->C[j + t];  }  y->n = t - 1;  for (int j = n; j >= i + 1; j--)  C[j + 1] = C[j];  C[i + 1] = z;  for (int j = n - 1; j >= i; j--)  keys[j + 1] = keys[j];  keys[i] = y->keys[t - 1];  n++;  }  int main() {  BTree t(3);  t.insert(10);  t.insert(20);  t.insert(5);  t.insert(6);  t.insert(12);  t.insert(30);  t.insert(7);  t.insert(17);  cout << "Traversal of B-tree:\n";  t.traverse();  return 0;  } |

2. **Implement the different operations of a B+ Tree in C++**

|  |  |
| --- | --- |
| 1  #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;  class BPlusNode {  public:  bool isLeaf;  vector<int> keys;  vector<BPlusNode\*> children;  BPlusNode\* next;  BPlusNode(bool leaf = true) {  isLeaf = leaf;  next = nullptr;  }  };  class BPlusTree {  BPlusNode\* root;  int t;  public:  BPlusTree(int \_t) {  root = new BPlusNode(true);  t = \_t;  }  void insert(int key);  void display();  };  void BPlusTree::insert(int key) {  root->keys.push\_back(key);  sort(root->keys.begin(), root->keys.end());  }  void BPlusTree::display() {  BPlusNode\* temp = root;  cout << "Leaf nodes: ";  for (int k : temp->keys)  cout << k << " ";  cout << endl;  } | 2  int main() {  BPlusTree bpt(3);  bpt.insert(5);  bpt.insert(15);  bpt.insert(25);  bpt.insert(35);  bpt.insert(45);  bpt.display();  return 0;  } |

3. **Implement the graph using different representations in C++**

|  |  |
| --- | --- |
| 1  #include <iostream>  #include <vector>  using namespace std;  void adjacencyMatrix(int V, vector<pair<int, int>> edges) {  vector<vector<int>> matrix(V, vector<int>(V, 0));  for (auto e : edges) {  matrix[e.first][e.second] = 1;  matrix[e.second][e.first] = 1;  }  cout << "Adjacency Matrix:\n";  for (int i = 0; i < V; i++) {  for (int j = 0; j < V; j++)  cout << matrix[i][j] << " ";  cout << endl;  }  }  void adjacencyList(int V, vector<pair<int, int>> edges) {  vector<vector<int>> adj(V);  for (auto e : edges) {  adj[e.first].push\_back(e.second);  adj[e.second].push\_back(e.first);  }  cout << "\nAdjacency List:\n";  for (int i = 0; i < V; i++) {  cout << i << ": ";  for (int j : adj[i])  cout << j << " ";  cout << endl;  }  }  void edgeList(vector<pair<int, int>> edges) {  cout << "\nEdge List:\n";  for (auto e : edges)  cout << "(" << e.first << ", " << e.second << ")\n";  } | 2  int main() {  int V = 5;  vector<pair<int, int>> edges = {  {0, 1}, {0, 2}, {1, 2}, {1, 3}, {3, 4}  };  adjacencyMatrix(V, edges);  adjacencyList(V, edges);  edgeList(edges);  return 0;  } |

**Week 11**

1. **C++ program to illustrate the traversal of a graph using Breadth First Search (BFS)**

|  |  |
| --- | --- |
| #include <iostream>  #include <vector>  #include <queue>  using namespace std;  void BFS(int start, vector<vector<int>>& adj, int V) {  vector<bool> visited(V, false);  queue<int> q;  visited[start] = true;  q.push(start);  cout << "BFS Traversal: ";  while (!q.empty()) {  int v = q.front();  q.pop();  cout << v << " ";  for (int u : adj[v]) {  if (!visited[u]) {  visited[u] = true;  q.push(u);  }  }  }  cout << endl;  }  int main() {  int V = 5;  vector<vector<int>> adj(V);  adj[0] = {1, 2};  adj[1] = {0, 3};  adj[2] = {0, 3};  adj[3] = {1, 2, 4};  adj[4] = {3};  BFS(0, adj, V);  return 0;  } |  |

2. **C++ program to illustrate the traversal of a graph using Depth First Search (DFS)**

|  |  |
| --- | --- |
| #include <iostream>  #include <vector>  using namespace std;  void DFSUtil(int v, vector<vector<int>>& adj, vector<bool>& visited) {  visited[v] = true;  cout << v << " ";  for (int u : adj[v]) {  if (!visited[u])  DFSUtil(u, adj, visited);  }  }  void DFS(int V, vector<vector<int>>& adj) {  vector<bool> visited(V, false);  cout << "DFS Traversal: ";  for (int i = 0; i < V; i++) {  if (!visited[i])  DFSUtil(i, adj, visited);  }  cout << endl;  }  int main() {  int V = 5;  vector<vector<int>> adj(V);  adj[0] = {1, 2};  adj[1] = {0, 3};  adj[2] = {0, 3};  adj[3] = {1, 2, 4};  adj[4] = {3};  DFS(V, adj);  return 0;  } |  |

3. **C++ program to find the edges of a spanning tree using Prim's Algorithm**

|  |  |
| --- | --- |
| 1  #include <iostream>  #include <vector>  #include <limits.h>  using namespace std;  #define V 5  int minKey(int key[], bool mstSet[]) {  int min = INT\_MAX, min\_index;  for (int v = 0; v < V; v++)  if (!mstSet[v] && key[v] < min)  min = key[v], min\_index = v;  return min\_index;  }  void primMST(int graph[V][V]) {  int parent[V];  int key[V];  bool mstSet[V];  for (int i = 0; i < V; i++)  key[i] = INT\_MAX, mstSet[i] = false;  key[0] = 0;  parent[0] = -1;  for (int count = 0; count < V - 1; count++) {  int u = minKey(key, mstSet);  mstSet[u] = true;  for (int v = 0; v < V; v++)  if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])  parent[v] = u, key[v] = graph[u][v];  } | 2  cout << "Edge \tWeight\n";  for (int i = 1; i < V; i++)  cout << parent[i] << " - " << i << "\t" << graph[i][parent[i]] << endl;  }  int main() {  int graph[V][V] = {  {0, 2, 0, 6, 0},  {2, 0, 3, 8, 5},  {0, 3, 0, 0, 7},  {6, 8, 0, 0, 9},  {0, 5, 7, 9, 0}  };  primMST(graph);  return 0;  } |

4. **C++ program to find the shortest path in a graph using Warshall’s Algorithm (Floyd-Warshall)**

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  #define V 4  #define INF 99999  void floydWarshall(int graph[V][V]) {  int dist[V][V];  for (int i = 0; i < V; i++)  for (int j = 0; j < V; j++)  dist[i][j] = graph[i][j];  for (int k = 0; k < V; k++) {  for (int i = 0; i < V; i++) {  for (int j = 0; j < V; j++) {  if (dist[i][k] + dist[k][j] < dist[i][j])  dist[i][j] = dist[i][k] + dist[k][j];  }  }  }  cout << "Shortest distances between every pair of vertices:\n";  for (int i = 0; i < V; i++) {  for (int j = 0; j < V; j++) {  if (dist[i][j] == INF)  cout << "INF ";  else  cout << dist[i][j] << " ";  }  cout << endl;  }  }  int main() {  int graph[V][V] = {  {0, 5, INF, 10},  {INF, 0, 3, INF},  {INF, INF, 0, 1},  {INF, INF, INF, 0}  };  floydWarshall(graph);  return 0;  } |  |

**Week 12**

**Write a C++ program to find the shortest path in a graph using Dijkstra’s Algorithm.**

|  |  |
| --- | --- |
| 1  #include <iostream>  #define V 5  #define INF 99999  using namespace std;  int minDistance(int dist[], bool visited[]) {  int min = INF, index;  for (int i = 0; i < V; i++)  if (!visited[i] && dist[i] <= min)  min = dist[i], index = i;  return index;  }  void dijkstra(int graph[V][V], int src) {  int dist[V];  bool visited[V];  for (int i = 0; i < V; i++)  dist[i] = INF, visited[i] = false;  dist[src] = 0;  for (int count = 0; count < V - 1; count++) {  int u = minDistance(dist, visited);  visited[u] = true;  for (int v = 0; v < V; v++)  if (!visited[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v])  dist[v] = dist[u] + graph[u][v];  }  cout << "Vertex\tDistance from Source\n";  for (int i = 0; i < V; i++)  cout << i << "\t" << dist[i] << "\n";  } | 2  int main() {  int graph[V][V] = {  {0, 10, 0, 5, 0},  {0, 0, 1, 2, 0},  {0, 0, 0, 0, 4},  {0, 3, 9, 0, 2},  {7, 0, 6, 0, 0}  };  dijkstra(graph, 0);  return 0;  } |

**2. Write a C++ program to implement Euler Graphs**

|  |  |
| --- | --- |
| `1  #include <iostream>  using namespace std;  #define V 5  void addEdge(int graph[V][V], int u, int v) {  graph[u][v] = 1;  graph[v][u] = 1;  }  void DFS(int graph[V][V], bool visited[], int v) {  visited[v] = true;  for (int i = 0; i < V; i++)  if (graph[v][i] && !visited[i])  DFS(graph, visited, i);  }  bool isConnected(int graph[V][V]) {  bool visited[V] = {false};  int i;  for (i = 0; i < V; i++) {  int degree = 0;  for (int j = 0; j < V; j++)  degree += graph[i][j];  if (degree > 0)  break;  }  if (i == V)  return true;  DFS(graph, visited, i);  for (int i = 0; i < V; i++) {  int degree = 0;  for (int j = 0; j < V; j++)  degree += graph[i][j];  if (degree > 0 && !visited[i])  return false;  }  return true;  }  bool isEulerGraph(int graph[V][V]) {  if (!isConnected(graph)) | 2  return false;  for (int i = 0; i < V; i++) {  int degree = 0;  for (int j = 0; j < V; j++)  degree += graph[i][j];  if (degree % 2 != 0)  return false;  }  return true;  }  int main() {  int graph[V][V] = {0};  addEdge(graph, 0, 1);  addEdge(graph, 1, 2);  addEdge(graph, 2, 3);  addEdge(graph, 3, 0);  addEdge(graph, 0, 2);  addEdge(graph, 1, 3);  if (isEulerGraph(graph))  cout << "The graph is an Euler Graph.\n";  else  cout << "The graph is NOT an Euler Graph.\n";  return 0;  } |

**3. Write a C++ program to implement Hamiltonian Graphs.**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define V 5  bool isSafe(int v, int graph[V][V], int path[], int pos) {  if (graph[path[pos - 1]][v] == 0)  return false;  for (int i = 0; i < pos; i++)  if (path[i] == v)  return false;  return true;  }  bool hamCycleUtil(int graph[V][V], int path[], int pos) {  if (pos == V) {  if (graph[path[pos - 1]][path[0]] == 1)  return true;  else  return false;  }  for (int v = 1; v < V; v++) {  if (isSafe(v, graph, path, pos)) {  path[pos] = v;  if (hamCycleUtil(graph, path, pos + 1))  return true;  path[pos] = -1; // backtrack  }  }  return false;  }  bool hamiltonianCycle(int graph[V][V]) {  int path[V];  for (int i = 0; i < V; i++)  path[i] = -1;  path[0] = 0; | 2  if (!hamCycleUtil(graph, path, 1)) {  cout << "No Hamiltonian Cycle exists.\n";  return false;  }  cout << "Hamiltonian Cycle exists: ";  for (int i = 0; i < V; i++)  cout << path[i] << " ";  cout << path[0] << endl; // Complete the cycle  return true;  }  int main() {  int graph1[V][V] = {  {0, 1, 0, 1, 0},  {1, 0, 1, 1, 1},  {0, 1, 0, 0, 1},  {1, 1, 0, 0, 1},  {0, 1, 1, 1, 0}  };  hamiltonianCycle(graph1);  return 0;  } |

**Week 13**

**Write a C++ program to implement Topological Sorting of a Directed Acyclic Graph (DAG).**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define V 6  void topologicalSortUtil(int v, bool visited[], int graph[V][V], int stack[], int &top) {  visited[v] = true;  for (int i = 0; i < V; i++) {  if (graph[v][i] && !visited[i])  topologicalSortUtil(i, visited, graph, stack, top);  }  stack[++top] = v; // Push to stack  }  void topologicalSort(int graph[V][V]) {  bool visited[V] = {false};  int stack[V], top = -1;  for (int i = 0; i < V; i++)  if (!visited[i])  topologicalSortUtil(i, visited, graph, stack, top); | 2  cout << "Topological Sort (Order of tasks): ";  while (top >= 0)  cout << stack[top--] << " ";  cout << endl;  }  int main() {  // Adjacency matrix for a DAG  int graph[V][V] = {  {0, 1, 1, 0, 0, 0},  {0, 0, 0, 1, 1, 0},  {0, 0, 0, 0, 1, 0},  {0, 0, 0, 0, 0, 1},  {0, 0, 0, 0, 0, 1},  {0, 0, 0, 0, 0, 0}  };  topologicalSort(graph);  return 0;  } |

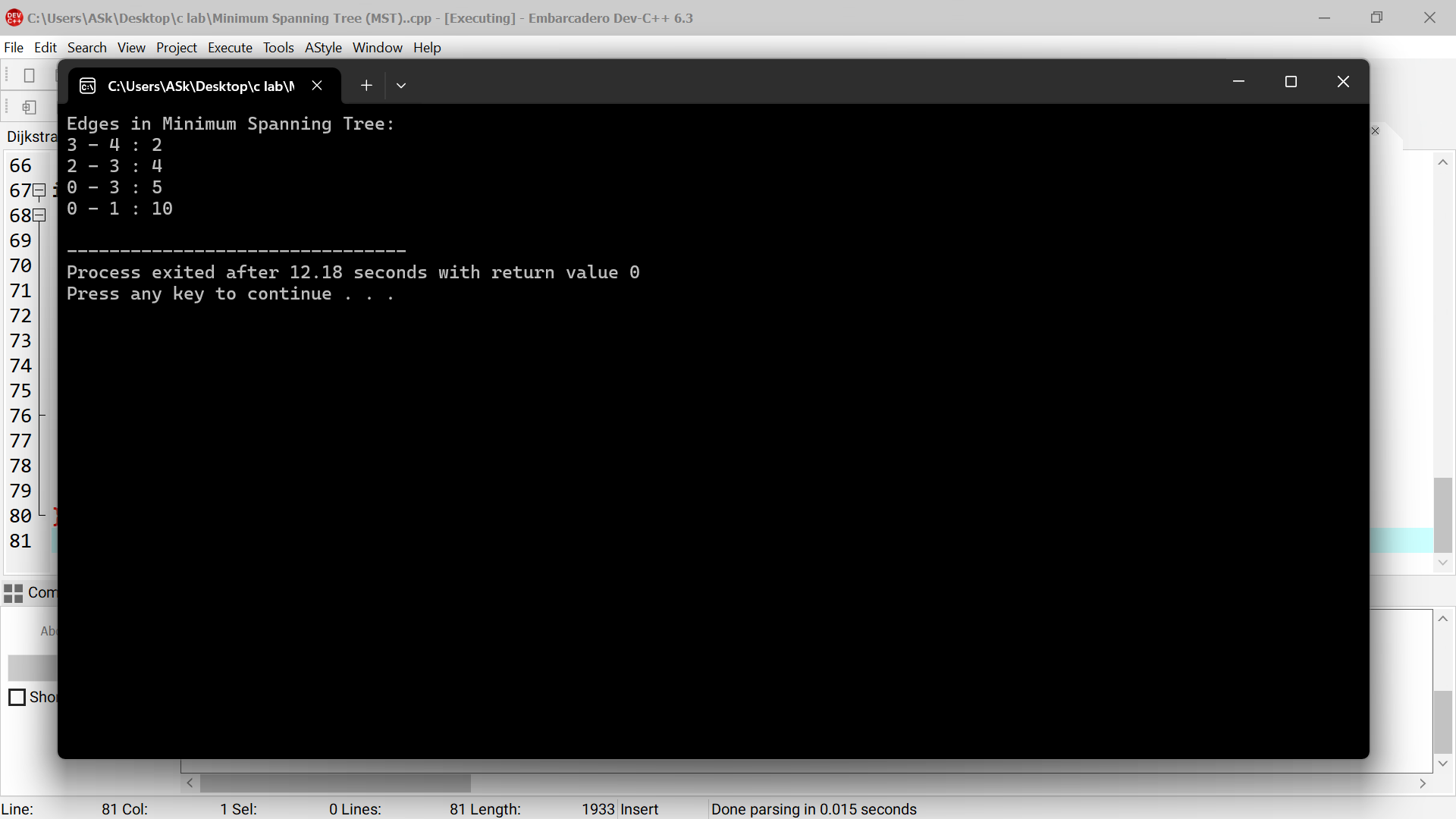
****

**2. Write a C++ program to detect Cycle in a Directed Graph using DFS.**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define V 4  bool dfs(int v, int graph[V][V], bool visited[], bool recStack[]) {  visited[v] = true;  recStack[v] = true;  for (int i = 0; i < V; i++) {  if (graph[v][i]) {  if (!visited[i] && dfs(i, graph, visited, recStack))  return true;  else if (recStack[i])  return true;  }  }  recStack[v] = false; // remove the vertex from recursion stack  return false;  }  bool isCyclic(int graph[V][V]) {  bool visited[V] = {false};  bool recStack[V] = {false};  for (int i = 0; i < V; i++)  if (!visited[i])  if (dfs(i, graph, visited, recStack))  return true;  return false;  } | **2**  int main() {  // Example with a cycle: 0 → 1 → 2 → 0  int graph[V][V] = {  {0, 1, 0, 0},  {0, 0, 1, 0},  {1, 0, 0, 0},  {0, 0, 0, 0}  };  if (isCyclic(graph))  cout << "Cycle detected in the directed graph.\n";  else  cout << "No cycle in the directed graph.\n";  return 0;  } |

**3. Write a C++ program to implement Kruskal’s Algorithm for finding the Minimum Spanning Tree (MST).**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define V 5  #define E 7 // Number of edges  struct Edge {  int src, dest, weight;  };  void sortEdges(Edge edges[], int n) {  for (int i = 0; i < n - 1; i++)  for (int j = 0; j < n - i - 1; j++)  if (edges[j].weight > edges[j + 1].weight) {  Edge temp = edges[j];  edges[j] = edges[j + 1];  edges[j + 1] = temp;  }  }  int find(int parent[], int i) {  if (parent[i] != i)  parent[i] = find(parent, parent[i]);  return parent[i];  }  void unionSets(int parent[], int rank[], int x, int y) {  int xroot = find(parent, x);  int yroot = find(parent, y);  if (rank[xroot] < rank[yroot])  parent[xroot] = yroot;  else if (rank[xroot] > rank[yroot])  parent[yroot] = xroot;  else {  parent[yroot] = xroot;  rank[xroot]++;  }  }  void kruskalMST(Edge edges[], int v, int e) {  sortEdges(edges, e); | 2  Edge result[10]; // Set a reasonable upper limit for result[] size  int parent[V], rank[V] = {0};  for (int i = 0; i < v; i++)  parent[i] = i;  int edgeCount = 0, i = 0;  while (edgeCount < v - 1 && i < e) {  Edge next = edges[i++];  int x = find(parent, next.src);  int y = find(parent, next.dest);  if (x != y) {  result[edgeCount++] = next;  unionSets(parent, rank, x, y);  }  }  cout << "Edges in Minimum Spanning Tree:\n";  for (int i = 0; i < edgeCount; i++)  cout << result[i].src << " - " << result[i].dest << " : " << result[i].weight << endl;  }  int main() {  Edge edges[E] = {  {0, 1, 10},  {0, 2, 6},  {0, 3, 5},  {1, 3, 15},  {2, 3, 4},  {1, 2, 25},  {3, 4, 2}  };  kruskalMST(edges, V, E);  return 0;  } |

****

**Week 14**

**1. Write a C++ program to detect a cycle in an Undirected Graph using DFS.**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define V 5 // Number of vertices  bool dfs(int v, int parent, bool visited[], int graph[V][V]) {  visited[v] = true;  for (int i = 0; i < V; i++) {  if (graph[v][i]) {  if (!visited[i]) {  if (dfs(i, v, visited, graph))  return true;  } else if (i != parent) {  return true; // A back edge found  }  }  }  return false;  }  bool hasCycle(int graph[V][V]) {  bool visited[V] = {false};  for (int i = 0; i < V; i++) {  if (!visited[i]) {  if (dfs(i, -1, visited, graph))  return true;  }  }  return false;  } | 2  int main() {  // Example graph with a cycle  int graph[V][V] = {  {0, 1, 0, 0, 0},  {1, 0, 1, 1, 0},  {0, 1, 0, 0, 0},  {0, 1, 0, 0, 1},  {0, 0, 0, 1, 0}  };  if (hasCycle(graph))  cout << "Cycle detected in the undirected graph.\n";  else  cout << "No cycle in the undirected graph.\n";  return 0;  } |

**2. Write a C++ program to find Strongly Connected Components (SCC) in a Directed Graph using Kosaraju’s Algorithm.**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define V 5 // Number of vertices  void dfs(int v, bool visited[], int graph[V][V], int stack[], int &top) {  visited[v] = true;  for (int i = 0; i < V; i++) {  if (graph[v][i] && !visited[i])  dfs(i, visited, graph, stack, top);  }  stack[++top] = v; // push to stack  }  void dfsUtil(int v, bool visited[], int graph[V][V]) {  visited[v] = true;  cout << v << " ";  for (int i = 0; i < V; i++) {  if (graph[v][i] && !visited[i])  dfsUtil(i, visited, graph);  }  }  void transposeGraph(int graph[V][V], int transpose[V][V]) {  for (int i = 0; i < V; i++)  for (int j = 0; j < V; j++)  transpose[i][j] = graph[j][i];  }  void findSCCs(int graph[V][V]) {  bool visited[V] = {false};  int stack[V], top = -1;  // Step 1: Fill vertices in stack according to their finishing time  for (int i = 0; i < V; i++)  if (!visited[i])  dfs(i, visited, graph, stack, top);  // Step 2: Create the transpose of the graph  int transpose[V][V] = {0}; | 2  transposeGraph(graph, transpose);  // Step 3: Do DFS on transposed graph in order of stack  for (int i = 0; i < V; i++) visited[i] = false;  cout << "Strongly Connected Components:\n";  while (top >= 0) {  int v = stack[top--];  if (!visited[v]) {  dfsUtil(v, visited, transpose);  cout << endl;  }  }  }  int main() {  // Example directed graph with SCCs  int graph[V][V] = {  {0, 1, 0, 0, 0},  {0, 0, 1, 0, 0},  {1, 0, 0, 1, 0},  {0, 0, 0, 0, 1},  {0, 0, 0, 0, 0}  };  findSCCs(graph);  return 0;  } |

**3. Write a C++ program to find Articulation Points (Cut Vertices) in an Undirected Graph using DFS**

|  |  |
| --- | --- |
| 1  #include <iostream>  using namespace std;  #define V 5  int timeCounter = 0;  void dfs(int u, int graph[V][V], bool visited[], int disc[], int low[], int parent[], bool ap[]) {  visited[u] = true;  disc[u] = low[u] = ++timeCounter;  int children = 0;  for (int v = 0; v < V; v++) {  if (graph[u][v]) {  if (!visited[v]) {  children++;  parent[v] = u;  dfs(v, graph, visited, disc, low, parent, ap);  low[u] = (low[u] < low[v]) ? low[u] : low[v];  // Case 1: root with two or more children  if (parent[u] == -1 && children > 1)  ap[u] = true;  // Case 2: non-root, and low[v] >= disc[u]  if (parent[u] != -1 && low[v] >= disc[u])  ap[u] = true;  } else if (v != parent[u]) {  low[u] = (low[u] < disc[v]) ? low[u] : disc[v];  }  }  }  } | 2  void findArticulationPoints(int graph[V][V]) {  bool visited[V] = {false};  int disc[V] = {0}, low[V] = {0}, parent[V];  bool ap[V] = {false};  for (int i = 0; i < V; i++)  parent[i] = -1;  for (int i = 0; i < V; i++)  if (!visited[i])  dfs(i, graph, visited, disc, low, parent, ap);  cout << "Articulation Points:\n";  for (int i = 0; i < V; i++)  if (ap[i])  cout << i << " ";  cout << endl;  }  int main() {  int graph[V][V] = {  {0, 1, 0, 0, 0},  {1, 0, 1, 1, 0},  {0, 1, 0, 0, 0},  {0, 1, 0, 0, 1},  {0, 0, 0, 1, 0}  };  findArticulationPoints(graph);  return 0;  } |