```
/*
Problem 1: List Operations
Description:
Write a program that uses the std::list container to manage a collection of integers. Your
program should perform the following operations:

Insert elements at the front and back of the list.
Remove elements from the front and back of the list.
Sort the list in ascending and descending order.
Reverse the list.
Display the elements of the list.

*/
#include <bits/stdc++.h>
using namespace std;

void displayList(list<int>& lst) {
    for (int elem : lst) {
        cout << elem << " ";
    }
    cout << endl;
}

int main() {
    list<int> lst;

    lst.push_back(10);
    lst.push_back(20);
    lst.push_front(5);
    lst.push_front(1);

    cout << "List after inserting elements at the front and back: ";
    displayList(lst);

    lst.pop_front();
    lst.pop_back();

    cout << "List after removing elements from the front and back: ";
    displayList(lst);

    lst.sort();
    cout << "List after sorting in ascending order: ";
    displayList(lst);
    lst.sort(greater<int>());
    cout << "List after sorting in descending order: ";
    displayList(lst);

    lst.reverse();
    cout << "List after reversing: ";
    displayList(lst);

    return 0;
}

#include <iostream>
#include <vector>
#include <algorithm> // For std::max_element and std::min_element
#include <numeric> // For std::accumulate
using namespace std;

// Function to display the elements of the vector
void displayVector(const vector<float>& vec) {
    for (float elem : vec) {
        cout << elem << " ";
    }
    cout << endl;
}

// Function to find the maximum element in the vector
float findMax(const vector<float>& vec) {
    return *max_element(vec.begin(), vec.end());
}

// Function to find the minimum element in the vector
float findMin(const vector<float>& vec) {
    return *min_element(vec.begin(), vec.end());
}

// Function to calculate the average of the elements in the vector
float calculateAverage(const vector<float>& vec) {
    float sum = accumulate(vec.begin(), vec.end(), 0.0f);
    return sum / vec.size();
}

int main() {
    vector<float> vec;
    // Add elements to the vector
    vec.push_back(1.5);
    vec.push_back(2.3);
    vec.push_back(3.7);
    vec.push_back(4.1);
    vec.push_back(5.9);

    cout << "Vector after adding elements: ";
    displayVector(vec);

    // Remove element from a specified position (e.g., position 2)
    int pos = 2;
    if (pos >= 0 && pos < vec.size()) {
        vec.erase(vec.begin() + pos);
    }

    cout << "Vector after removing element at position " << pos << ": ";
    displayVector(vec);

    // Find the maximum and minimum elements in the vector
    float maxElem = findMax(vec);
    float minElem = findMin(vec);

    cout << "Maximum element in the vector: " << maxElem << endl;
    cout << "Minimum element in the vector: " << minElem << endl;

    // Calculate the average of the elements in the vector
    float average = calculateAverage(vec);
    cout << "Average of elements in the vector: " << average << endl;

    // Display the elements of the vector
    cout << "Final elements in the vector: ";
    displayVector(vec);

    return 0;
}

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

void displayqueue(queue<int> q) {
    cout << "Current queue is: ";
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    cout << endl;
}

int main() {
    queue<int> ticketqueue;
    int servedCustomers = 0;

    ticketqueue.push(1);
    ticketqueue.push(2);
    ticketqueue.push(3);
    ticketqueue.push(4);
    ticketqueue.push(5);

    cout << "After adding customers:" << endl;
    displayqueue(ticketqueue);

    while (!ticketqueue.empty()) {
        cout << "Serving customer  is: " << ticketqueue.front() << endl;
        ticketqueue.pop();
        servedCustomers++;
    }

    cout << "All customers served." << endl;

    cout << "Number of customers served: " << servedCustomers << endl;

    return 0;
}

/******************************************************************************

Welcome to GDB Online.
GDB online is an online compiler and debugger tool for C, C++, Python, Java, PHP, Ruby, Perl,
C#, OCaml, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly, HTML, CSS, JS,
SQLite, Prolog.
Code, Compile, Run and Debug online from anywhere in world.

*******************************************************************************/
#include <bits/stdc++.h>
using namespace std;
```

```
int evaluatePostfix(const string& expression) {
    stack<int> s;
    stringstream ss(expression);
    string token;

    while (ss >> token) {
        if (isdigit(token[0])) {
            s.push(stoi(token));
        } else {
            int val2 = s.top(); s.pop();
            int val1 = s.top(); s.pop();

            if (token == "+") s.push(val1 + val2);
            else if (token == "-") s.push(val1 - val2);
            else if (token == "*") s.push(val1 * val2);
            else if (token == "/") s.push(val1 / val2);
        }
    }

    return s.top();
}

int main() {
    string expression;
    cout << "Enter a postfix expression: ";
    getline(cin, expression);

    int result = evaluatePostfix(expression);
    cout << "Result of the evaluation: " << result << endl;

    return 0;
}

/******************************************************************************

Welcome to GDB Online.
GDB online is an online compiler and debugger tool for C, C++, Python, Java, PHP, Ruby, Perl,
C#, OCaml, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly, HTML, CSS, JS,
SQLite, Prolog.
Code, Compile, Run and Debug online from anywhere in world.

*******************************************************************************/
#include <bits/stdc++.h>
```

```
using namespace std;


int evaluatePostfix(const string& expression) {
    stack<int> s;
    stringstream ss(expression);
    string token;

    while (ss >> token) {
        if (isdigit(token[0])) {
            s.push(stoi(token));
        } else {
            int val2 = s.top(); s.pop();
            int val1 = s.top(); s.pop();

            if (token == "+") s.push(val1 + val2);
            else if (token == "-") s.push(val1 - val2);
            else if (token == "*") s.push(val1 * val2);
            else if (token == "/") s.push(val1 / val2);
        }
    }

    return s.top();
}

int main() {
    string expression;
    cout << "Enter a postfix expression: ";
    getline(cin, expression);

    int result = evaluatePostfix(expression);
    cout << "Result of the evaluation: " << result << endl;

    return 0;
}

/*
Classes:

Shape: Base class representing a generic shape.
Rectangle: Derived class representing a rectangle with length and width.
Circle: Derived class representing a circle with radius.
Concepts:
```

Constructors and Destructors:

Define a default constructor for Shape to initialize common properties.
Overload constructors for Rectangle and Circle to take specific dimensions as input during object creation.
Implement destructors for all classes to handle memory cleanup (if applicable).
Overriding:

Override the area() function in Rectangle and Circle to calculate their respective areas using appropriate formulas. The base class Shape can have a pure virtual area() function to enforce implementation in derived classes.
Operator Overloading:

Overload the == operator for Shape to compare shapes based on a chosen criterion (e.g., area for simplicity).
Consider overloading other operators (like +) for specific shapes if applicable (e.g., combining rectangles).
Friend Function:

Define a friend function totalArea outside the class hierarchy that takes an array of Shape pointers and calculates the total area of all shapes. This function needs access to private member variables of Shape and its derived classes.
Template (Optional):

(Optional) Create a template class Point to represent a point in 2D space with x and y coordinates. Use this template class within the Shape hierarchy if needed.
Implementation:

Design the Shape class with appropriate member variables and functions, including a pure virtual area() function.
Implement derived classes Rectangle and Circle with constructors, destructors, overridden area() functions, and potentially overloaded operators.
Define a friend function totalArea that takes an array of Shape pointers and calculates the total area.
(Optional) Implement a template class Point for representing points.
Testing:

Create objects of different shapes (rectangle, circle) and test their constructors, destructors, and overridden area() functions.
Use the overloaded == operator to compare shapes.
Call the totalArea friend function to calculate the total area of an array of shapes.
(Optional) Test the functionality of the Point template class (if implemented).

*/

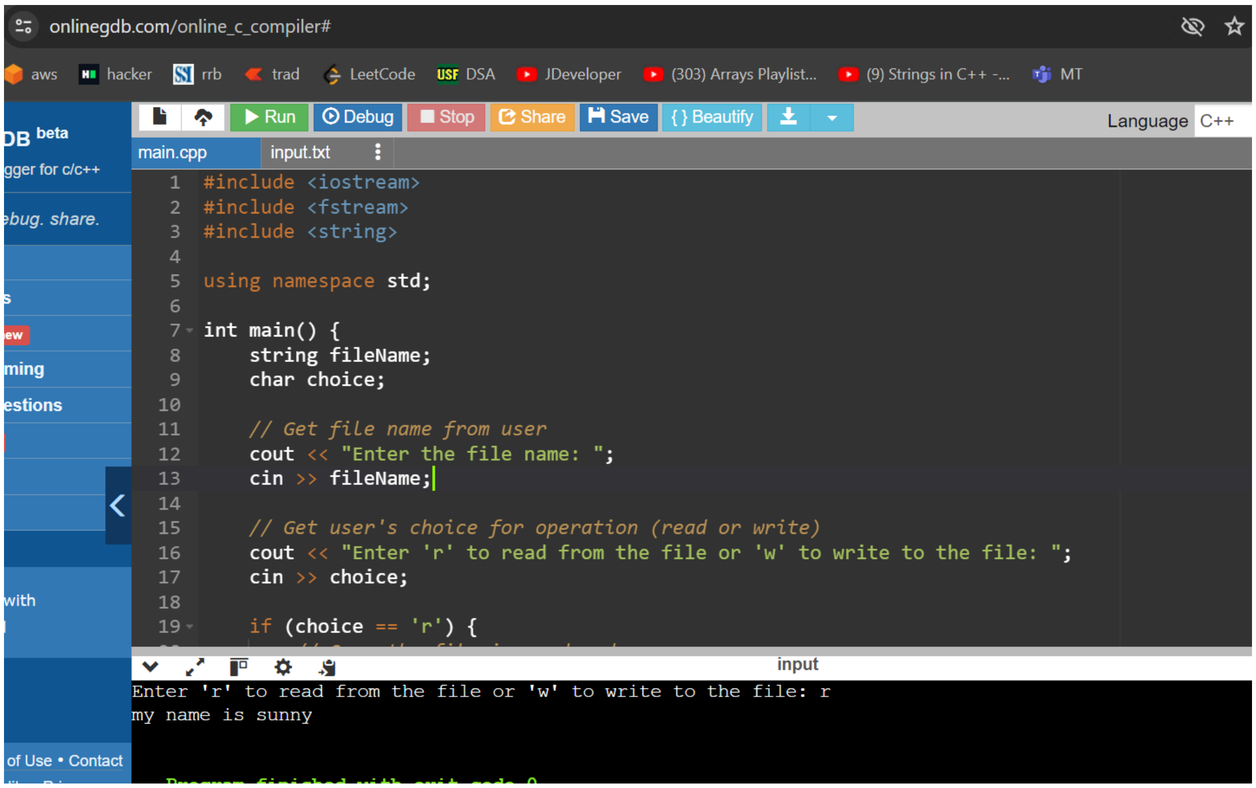
```
#include <bits/stdc++.h>
using namespace std;

class Shape {
public:
   Shape() {                  //define a base class
      cout<<"Shape" << endl;         //print the shape
   }
   virtual double area() const = 0;       //this is a pure virtual function

   bool operator==(const Shape& other) const {      //this is used for comparing based on area
      return area() == other.area();
   }
};

class Rectangle : public Shape {           //derived class Rectangle
private:
   double length;                          //initialize length and width for Rectangle
   double width;

public:
   Rectangle(double l, double w) : length(l), width(w) {        //constructor
      cout<<"Rectangle" << endl;
   }


   double area() const override {                  //override the area function
      return length * width;                     //return area of rectangle
   }

   Rectangle operator+(const Rectangle& other) const {         //combine two Rectangle by
adding their length and width
      return Rectangle(length + other.length, width + other.width);
   }

   friend double totalArea(const vector<Shape*>& shapes);  //this is our friend function
};

class Circle : public Shape {              //derived class circle
private:
   double radius;                          //initialize the radius

public:
   Circle(double r) : radius(r) {          //constructor
```

```
        cout<<"Circle" << endl;
    }


    double area() const override {          //override the area function
        return M_PI * radius * radius;      //return the area of the circle
    }

    friend double totalArea(const vector<Shape*>& shapes);      //friend function
};

double totalArea(const vector<Shape*>& shapes) {        //Defines a friend function totalArea that
takes a vector of Shape pointers
    double total = 0;
    for (size_t i = 0; i < shapes.size(); ++i) {
        total += shapes[i]->area();
    }                                           //return totalArea;
    return total;
}

int main() {                            //here is the main function
    Rectangle rect1(3.0, 4.0);          //Rectangle and Circle instance
    Rectangle rect2(5.0, 6.0);
    Circle circ1(2.0);

    cout<<"Area of rect1 is: " << rect1.area() << endl;
    cout<<"Area of rect2 is: " << rect2.area() << endl;
    cout<<"Area of circ1 is: " << circ1.area() << endl;



    Rectangle rect3 = rect1 + rect2;
    cout<<"Area of rect3 is: " << rect3.area() << endl;

    vector<Shape*> shapes = { &rect1, &rect2, &circ1 };
    cout<<"Total area of all shapes: " << totalArea(shapes) << endl;

    return 0;
}
```

Q.Write a C++ program that reads a text file named input.txt and prints its content to the console.

```
#include <iostream>
```

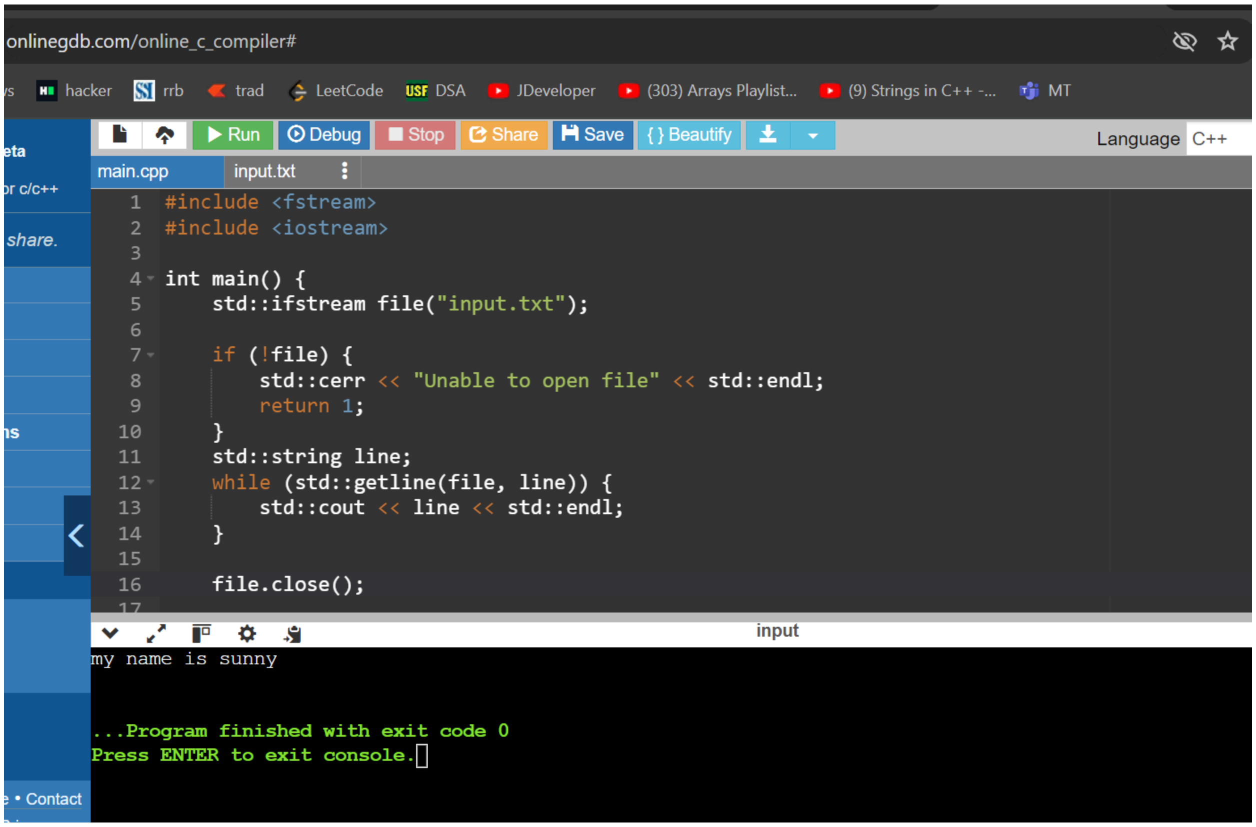
```
#include <fstream>
#include <string>

using namespace std;

int main() {
    string fileName;
    char choice;

    // Get file name from user
    cout << "Enter the file name: ";
    cin >> fileName;

    // Get user's choice for operation (read or write)
    cout << "Enter 'r' to read from the file or 'w' to write to the file: ";
    cin >> choice;

    if (choice == 'r') {
        // Open the file in read mode
        ifstream inputFile(fileName);

        if (inputFile.is_open()) {
            string line;

            // Read data from the file and print line by line
            while (getline(inputFile, line)) {
                cout << line << endl;
            }

            inputFile.close();
        } else {
            cout << "Error opening file for reading." << endl;
        }
    } else if (choice == 'w') {
        // Open the file in write mode (truncates existing content)
        ofstream outputFile(fileName);

        if (outputFile.is_open()) {
            string content;

            // Get content from user to write to the file
            cout << "Enter the content to write to the file: ";
            getline(cin, content, '\n'); // Include newline character
            outputFile << content << endl;

            outputFile.close();
            cout << "Content written to the file successfully." << endl;
        } else {
            cout << "Error opening file for writing." << endl;
        }
    } else {
        cout << "Invalid choice. Please enter 'r' or 'w'." << endl;
    }

    return 0;
}
```


onlinegdb.com/online_c_compiler#
aws hacker rrb trad LeetCode DSA JDeveloper (303) Arrays Playlist... (9) Strings in C++ -... MT
Run Debug Stop Share Save { } Beautify
Language C++
main.cpp input.txt
input

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
    string fileName;
    char choice;

    // Get file name from user
    cout << "Enter the file name: ";
    cin >> fileName;

    // Get user's choice for operation (read or write)
    cout << "Enter 'r' to read from the file or 'w' to write to the file: ";
    cin >> choice;

    if (choice == 'r') {
```

```
Enter 'r' to read from the file or 'w' to write to the file: r
my name is sunny


...Program finished with exit code 0
```

Q.How do you open a file for reading in C++?
Ans: using ifstream class



```cpp
#include <fstream>
#include <iostream>

int main() {
    std::ifstream file("input.txt");

    if (!file) {
        std::cerr << "Unable to open file" << std::endl;
        return 1;
    }
    std::string line;
    while (std::getline(file, line)) {
        std::cout << line << std::endl;
    }

    file.close();
```

```
my name is sunny


...Program finished with exit code 0
Press ENTER to exit console.
```

Q. What is the purpose of the ifstream class in C++?
Ans: some key purpose of ifstream class in c++:-

1. Opening files for reading
2. Reading data from file
3. Checking file status

/*
Write a C++ program that writes the following lines to a file named output.txt:

bash
Copy code
Hello, world!
This is a test file.
*/
#include <fstream>
#include <iostream>

int main() {
    std::ofstream file("output.txt");

    if (!file) {

```
        std::cerr << "Unable to open file for writing" << std::endl;
        return 1;
    }

    file << "Hello, world!" << std::endl;
    file << "This is a test file." << std::endl;

    file.close();

    return 0;
}
```

Q. How do you properly close a file after reading?
Ans: using file.close();

Q.What function do you use to read a line from a file?
use the std::getline function. This function is part of the string header and works with input streams like `std::ifstream`.