

1. Access Control and Getters:

Create the User class with private members for username and profile picture (string).

Implement public member functions for the constructor and getters (accessor methods) for username and profile picture.

```
#include <iostream>

#include <string>

using namespace std;

class User {
private:
    string username;
    string profilePicture;
public:
    User(const string& uname, const string& pic) {                // Constructor
        username = uname;
        profilePicture = pic;
    }

    string getUsername() const {                                  // Getter for username
        return username;
    }

    string getProfilePicture() const {                            // Getter for profile picture
        return profilePicture;
    }
};

int main() {
    User user("Ramya", "profile.jpg");                          // Example usage

    cout << "Username: " << user.getUsername() << endl;        // Accessing
```

private members via getter methods

```
    cout << "Profile Picture: " << user.getProfilePicture() << endl;

    return 0;
}
```

2. Post Class and Display:

Create the derived class Post inheriting from User.

Add private members for post content (string) and timestamp (date/time format of your choice).

Implement a public member function getPostInfo that returns a formatted string containing username, profile picture, post content, and timestamp.

```
#include <iostream>

#include <string>

#include <ctime>           // for timestamp handling

#include <sstream>         // for string stream

using namespace std;

class User {
private:
    string username;

    string profilePicture;

public:
    User(const string& uname, const string& profilePic)
        : username(uname), profilePicture(profilePic) {}

    string getUsername() const {
        return username;
    }

    string getProfilePicture() const {
```

```

        return profilePicture;
    }
};

class Post : public User { // Derived class Post inheriting from
    User

private:
    string postContent;

    string timestamp;

public:
    Post(const string& uname, const string& profilePic, const string& content)
        : User(uname, profilePic), postContent(content) {

        time_t now = time(0); // Get current timestamp

        tm* localTime = localtime(&now);

        stringstream ss;

        ss << (localTime->tm_year + 1900) << '-'

            << (localTime->tm_mon + 1) << '-'

            << localTime->tm_mday << ' '

            << localTime->tm_hour << ':'

            << localTime->tm_min << ':'

            << localTime->tm_sec;

        timestamp = ss.str();
    }

    string getPostInfo() const {

        stringstream ss;

        ss << "Username: " << getUsername() << "\n";

        ss << "Profile Picture: " << getProfilePicture() << "\n";
    }
};

```



```

        return username;

    }

    friend void basicInteract(const User& user1, const User& user2);           // Declare
    basicInteract as a friend function

};

void basicInteract(const User& user1, const User& user2) {                     //
    Definition of basicInteract function

        std::cout << user1.getUsername() << " interacts with " << user2.getUsername() << "." << std::endl;

    }

int main() {                                                                    // Main function for testing

    User user1("john");                                                         // Create two User objects

    User user2("ram");

    basicInteract(user1, user2);                                                // Call basicInteract function

    return 0;

}

```

4. Overloaded Interact Functions:

Create overloaded versions of the interact function:

likePost(User& user, Post& post): This function should print a message indicating the user liked the post.

followUser(User& follower, User& followed): This function should print a message indicating the user started following another user.

```
#include <iostream>
```

```
#include <string>
```

```
class User;
```

```
class Post;
```

```
void interact(User& user, Post& post);
```

```
void interact(User& follower, User& followed);
```

```

class User {
public:
    User(const std::string& name) : name(name) {}

    std::string getName() const { return name; }

private:
    std::string name;
};

class Post {
public:
    Post(const std::string& content) : content(content) {}

    std::string getContent() const { return content; }

private:
    std::string content;
};

void interact(User& user, Post& post) {                                // Overloaded interact functions
    std::cout << user.getName() << " liked the post: \"" << post.getContent() << "\"\" << std::endl;
}

void interact(User& follower, User& followed) {
    std::cout << follower.getName() << " started following " << followed.getName() << std::endl;
}

int main() {
    User arjun("Arjun");

    User ram("Ram");

    Post post("Hello, world!");

    interact(arjun, post);                                // Arjun liked the post: "Hello, world!"
}

```

```

        interact(ram, arjun);                // Ram started following Arjun

        return 0;

    }

```

5. Refactoring with Encapsulation:

Revisit the class design. Can you modify the code to reduce reliance on friend functions?

Consider adding public member functions or accessor methods within the User class to provide controlled access to relevant data instead of exposing everything through friend functions.

Bonus Challenge:

Implement a way to store and manage friend connections within the class hierarchy. You could explore a separate Friendship class or a boolean flag within User to track friend status. Modify the interact functions to incorporate this information and display more relevant messages based on the relationship between users.

```

#include <iostream>

#include <string>

class User {

private:

    std::string username;

    bool isFriend;

public:

    User(const std::string& username) : username(username), isFriend(false) {}

    std::string getUsername() const {

        return username;

    }

    void addFriend(User& user) {

        user.isFriend = true;

        isFriend = true;

        std::cout << username << " and " << user.getUsername() << " are now friends.\n";
    }
}

```

```

    }

    void removeFriend(User& user) {

        if (user.isFriend) {

            user.isFriend = false;

            isFriend = false;

            std::cout << username << " and " << user.getUsername() << " are no longer friends.\n";

        } else {

            std::cout << username << " and " << user.getUsername() << " are not friends.\n";

        }

    }

    void interact(const User& user) const {

        if (isFriend && user.isFriend) {

            std::cout << username << " interacts with friend " << user.getUsername() << ".\n";

        } else {

            std::cout << username << " interacts with non-friend " << user.getUsername() << ".\n";

        }

    }

};

int main() {

    User vishwa("Vishwa");

    User kiran("Kiran");

    vishwa.addFriend(kiran);

    vishwa.interact(kiran);

    kiran.interact(vishwa);

    vishwa.removeFriend(kiran);

```



```
vishwa.interact(kiran);

kiran.interact(vishwa);

return 0;

}
```

NUMBER OF OBJECTS CREATED :-

```
#include <iostream>

class MyClass {
    private:

    static int counter;

    public:

    MyClass() {
        counter++;
    }

    static int getCount() {
        return counter;
    }
};

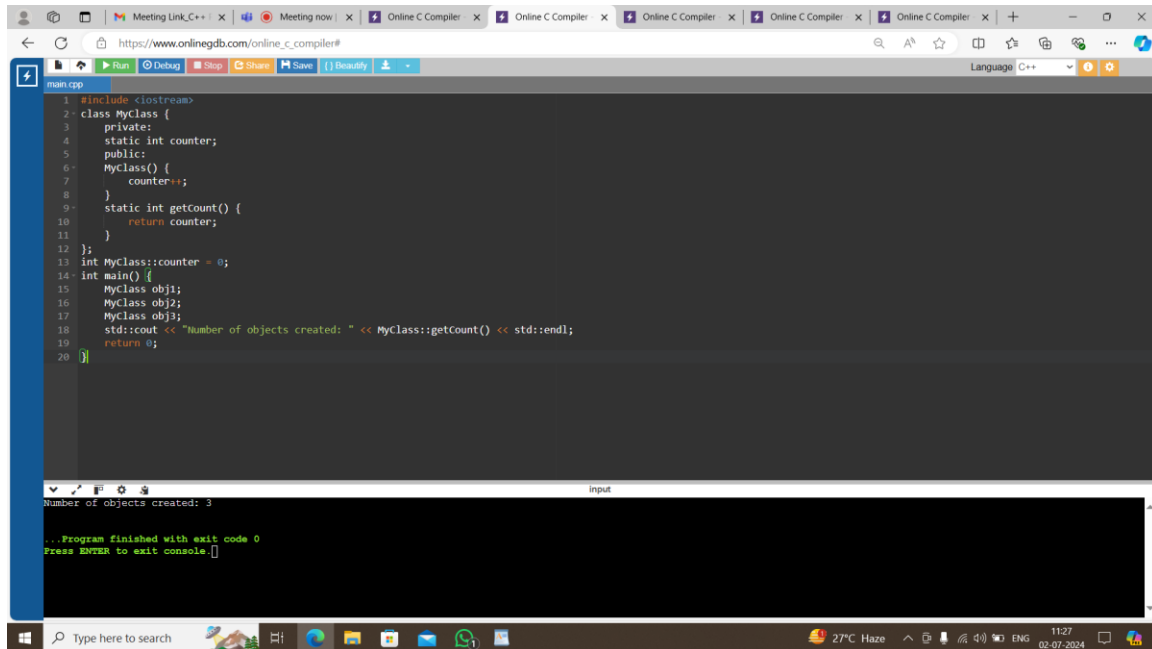
int MyClass::counter = 0;

int main() {
    MyClass obj1;
    MyClass obj2;
    MyClass obj3;

    std::cout << "Number of objects created: " << MyClass::getCount() << std::endl;

    return 0;
}
```

OUTPUT :-



```
1 #include <iostream>
2 class MyClass {
3 private:
4     static int counter;
5 public:
6     MyClass() {
7         counter++;
8     }
9     static int getCount() {
10         return counter;
11     }
12 };
13 int MyClass::counter = 0;
14 int main() {
15     MyClass obj1;
16     MyClass obj2;
17     MyClass obj3;
18     std::cout << "Number of objects created: " << MyClass::getCount() << std::endl;
19     return 0;
20 }
```

Number of objects created: 3

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

NUMBER OF OBJECTS CREATED IN STATIC :-

```
#include<iostream>
```

```
class myclass{
```

```
    private:
```

```
        static int counter;
```

```
        int count;
```

```
    public:
```

```
        myclass(){
```

```
            count++;
```

```
            counter++;
```

```
        }
```

```
        static int getcounter(){
```

```
            return counter;
```

```
        }
```

```
        int getcount(){
```

```

        return count;
    }
};

int myclass::counter =0;

int main(){

    myclass obj1;

    myclass obj2;

    myclass obj3;

    std::cout<<"number of objects created:"<<myclass::getcounter()<<std::endl;

    std::cout<<"objects1 count method:"<<obj1.getcount()<<std::endl;

    std::cout<<"objects2 count method:"<<obj2.getcount()<<std::endl;

    std::cout<<"objects3 count method:"<<obj3.getcount()<<std::endl;

    return 0;

}

```

OUTPUT :-

The screenshot shows a web browser window with the URL https://www.onlinegdb.com/online_c_compiler#. The browser has multiple tabs open, including 'Meeting Link C++', 'Meeting now', and several 'Online C Compiler' tabs. The main content area displays a C++ program in a dark-themed editor. The code defines a class 'myclass' with a static counter, a constructor that increments the counter, and two methods to get the counter value. In the 'main' function, three objects of 'myclass' are created, and their counts are printed. The output window at the bottom shows the execution results: 'number of objects created:3', 'objects1 count method:1', 'objects2 count method:1', and 'objects3 count method:1'. The program finished with exit code 0.

```

main.cpp
1 #include<iostream>
2 class myclass{
3 private:
4     static int counter;
5     int count;
6 public:
7     myclass(){
8         count++;
9         counter++;
10    }
11    static int getcounter(){
12        return counter;
13    }
14    int getcount(){
15        return count;
16    }
17 };
18 int myclass::counter =0;
19 int main(){
20     myclass obj1;
21     myclass obj2;
22     myclass obj3;
23     std::cout<<"number of objects created:"<<myclass::getcounter()<<std::endl;
24     std::cout<<"objects1 count method:"<<obj1.getcount()<<std::endl;
25     std::cout<<"objects2 count method:"<<obj2.getcount()<<std::endl;
26     std::cout<<"objects3 count method:"<<obj3.getcount()<<std::endl;
27     return 0;
28 }

```

number of objects created:3
objects1 count method:1
objects2 count method:1
objects3 count method:1
...Program finished with exit code 0
Press ENTER to exit console.

Distance Converter:

Create a class named `DistanceConverter`. Include the following static methods:

`convertMilesToKm(double miles)`: Converts miles to kilometers (1 mile = 1.60934 kilometers).

`convertKmToMiles(double kilometers)`: Converts kilometers to miles. In your main function, prompt the user for a distance and a unit (miles or kilometers). Use the appropriate static method from the `DistanceConverter` class to perform the conversion and display the result to the user.

```
#include <iostream>
```

```
using namespace std;
```

```
class DistanceConverter {
```

```
public:
```

```
    static float convertMilesToKm(float miles) {
```

```
        return miles * 1.60934;
```

```
    }
```

```
    static float convertKmToMiles(float kms) {
```

```
        return kms / 1.60934;
```

```
    }
```

```
};
```

```
int main() {
```

```
    float distance;
```

```
    char unit;
```

```
    cout << "Enter distance: ";
```

```
    cin >> distance;
```

```
    cout << "Enter unit (m for miles, k for kms): ";
```

```
    cin >> unit;
```

```
    if (unit == 'm' || unit == 'M') {
```

```

        float kmss = DistanceConverter::convertMilesToKm(distance);

        cout << distance << " miles is equal to " << kmss << " kms." << endl;

    } else if (unit == 'k' || unit == 'K') {

        float miles = DistanceConverter::convertKmToMiles(distance);

        cout << distance << " kms is equal to " << miles << " miles." << endl;

    } else {

        cout << "Invalid unit entered. Please enter 'm' for miles or 'k' for kilometers." << endl;

    }

    return 0;

}

```

Math Utility Class:

Design a class named MathUtil. Include static methods for basic mathematical operations:

add(int a, int b): Adds two integers.

subtract(int a, int b): Subtracts two integers.

multiply(int a, int b): Multiplies two integers.

divide(int a, int b) (optional): Divides two integers with error handling for division by zero. In your main function, prompt the user for two numbers and an operation (+, -, *, or /). Use the corresponding static method from the MathUtil class to perform the calculation and display the result.

```
#include <iostream>
```

```
#include <stdexcept>
```

```
class MathUtil {
```

```
public:
```

```
    static int add(int a, int b) {
```

```
        return a + b;
```

```
    }
```

```
    static int sub(int a, int b) {
```

```

        return a - b;
    }

    static int mul(int a, int b) {
        return a * b;
    }

    static double divide(int a, int b) {
        if (b == 0) {
            throw std::invalid_argument("Division by zero!");
        }
        return static_cast<double>(a) / b;
    }
};

int main() {
    int a = 20;
    int b = 32;

    std::cout << "Addition: " << MathUtil::add(a, b) << std::endl;
    std::cout << "Subtraction: " << MathUtil::sub(a, b) << std::endl;
    std::cout << "Multiplication: " << MathUtil::mul(a, b) << std::endl;
    try {
        std::cout << "Division: " << MathUtil::divide(a, b) << std::endl;
    } catch (const std::invalid_argument& e) {
        std::cerr << e.what() << std::endl;
    }

    return 0;
}

```

Simple Currency Converter:

Create a class named `CurrencyConverter`. Define a static variable named `exchangeRate` (e.g., USD to EUR exchange rate). Implement static methods:

`convertToEur(double amount)`: Converts an amount from the base currency (USD) to EUR based on the exchange rate.

`convertFromEur(double amount)`: Converts an amount from EUR to the base currency (USD). In your main function, prompt the user for an amount and a conversion direction (USD to EUR or EUR to USD). Use the appropriate static method from the `CurrencyConverter` class to perform the conversion and display the result.

```
#include <iostream>

using namespace std;

class CurrencyCon {
private:
    static double exchangeRate;                // exchange rate from
    USD to EUR

public:
    static double convertToEur(double amount) {
        return amount * exchangeRate;
    }

    static double convertFromEur(double amount) {
        return amount / exchangeRate;
    }
};

double CurrencyCon::exchangeRate = 0.85;      // 1 USD = 0.85 EUR (example
rate)

int main() {
    double amount;

    int choice;
```

```

    cout << "Enter the amount to convert: ";

    cin >> amount;

    cout << "Choose conversion direction:" << endl;

    cout << "1. USD to EUR" << endl;

    cout << "2. EUR to USD" << endl;

    cout << "Enter your choice (1 or 2): ";

    cin >> choice;

    switch (choice) {

        case 1:

            cout << amount << " USD = " << CurrencyCon::convertToEur(amount) << " EUR" << endl;

            break;

        case 2:

            cout << amount << " EUR = " << CurrencyCon::convertFromEur(amount) << " USD" <<
endl;

            break;

        default:

            cout << "Invalid choice" << endl;

    }

    return 0;

}

```

FUNCTION TEMPLATES :-

```

#include <iostream>

using namespace std;

template<class T> T add(T &a,T &b)

{

```



```

        T result = a+b;

        return result;
    }

int main()
{
    int i = 2;

    int j = 3;

    float m = 2.3;

    float n = 1.2;

    cout << "Addition of i and j is :" <<add(i,j);

    cout << '\n';

    cout << "Addition of m and n is :" <<add(m,n);

    return 0;
}

```

FUNCTION TEMPLATES WITH MULTIPLE PARAMETERS :-

```

#include <iostream>

using namespace std;

template<class X, class Y>void fun(X a,Y b)
{
    std::cout << "Value of a is : " <<a<< std::endl;

    std::cout << "Value of b is : " <<b<< std::endl;
}

int main()
{
    fun(15,12.3);

    return 0;
}

```

OVERLOADING FUNCTION TEMPLATE :-

```
#include <iostream>

using namespace std;

template<class X> void fun(X a)
{
    std::cout << "Value of a is : " <<a<< std::endl;
}

template<class X, class Y>void fun(X a,Y b)
{
    std::cout << "Value of a is : " <<a<< std::endl;
    std::cout << "Value of b is : " <<b<< std::endl;
}

int main()
{
    fun(10);
    fun(20,30.5);
    return 0;
}
```

Design a function template named compare that takes two arguments of the same type and returns a boolean value indicating whether the first argument is greater than, less than, or equal to the second argument. How would you adapt this template to work with custom data types?

```
#include <iostream>

template <typename T>                                // Template function to compare
two values

bool compare(const T& a, const T& b) {
    if (a < b) {
        std::cout << a << " is less than " << b << std::endl;
    }
}
```

```

        return false;
    } else if (a > b) {
        std::cout << a << " is greater than " << b << std::endl;
        return false;
    } else {
        std::cout << a << " is equal to " << b << std::endl;
        return true;
    }
}

class CustomType {                                // Custom data type
public:
    int value;

    CustomType(int v) : value(v) {}

    bool operator<(const CustomType& other) const {    // Overload
comparison operators
        return this->value < other.value;
    }

    bool operator>(const CustomType& other) const {
        return this->value > other.value;
    }

    bool operator==(const CustomType& other) const {
        return this->value == other.value;
    }

    friend std::ostream& operator<<(std::ostream& os, const CustomType& obj) {
        os << obj.value;
        return os;
    }
};

```

```

int main() {

    int x = 10, y = 20;                                // Compare basic data types

    compare(x, y);

    double a = 3.12, b = 3.81;

    compare(a, b);

    CustomType obj1(7), obj2(7);                        // Compare custom data types

    compare(obj1, obj2);

    CustomType obj3(5), obj4(10);

    compare(obj3, obj4);

    return 0;

}

```

OUTPUT :-

The screenshot shows a web browser window with an online C++ compiler. The code is pasted into the editor, and the output is displayed in the console. The output shows the results of the comparison function for various inputs: 10 is less than 20, 3.12 is less than 3.81, 7 is equal to 7, and 5 is less than 10. The program finishes with exit code 0.

```

1 #include <iostream>
2 template <typename T>
3 bool compare(const T& a, const T& b) {
4     if (a < b) {
5         std::cout << a << " is less than " << b << std::endl;
6     } else if (a > b) {
7         std::cout << a << " is greater than " << b << std::endl;
8     } else {
9         std::cout << a << " is equal to " << b << std::endl;
10        return true;
11    }
12    return false;
13 }
14
15 class CustomType {
16 public:
17     int value;
18     CustomType(int v) : value(v) {}
19     bool operator<(const CustomType& other) const {
20         return this->value < other.value;
21     }
22     bool operator>(const CustomType& other) const {
23         return this->value > other.value;
24     }
25     bool operator==(const CustomType& other) const {
26         return this->value == other.value;
27     }
28     friend std::ostream& operator<<(std::ostream& os, const CustomType& obj) {
29         os << obj.value;
30     }
31 };
32
33 int main() {
34     int x = 10, y = 20;
35     compare(x, y);
36     double a = 3.12, b = 3.81;
37     compare(a, b);
38     CustomType obj1(7), obj2(7);
39     compare(obj1, obj2);
40     CustomType obj3(5), obj4(10);
41     compare(obj3, obj4);
42     return 0;
43 }

```

Output:

```

10 is less than 20
3.12 is less than 3.81
7 is equal to 7
5 is less than 10
...Program finished with exit code 0
Press ENTER to exit console.

```

Implement a function template named swap that exchanges the values of two variables of the same type. Discuss the potential limitations of this approach when dealing with complex data structures.

```

#include <iostream>

template <typename T>                                // Template function to swap two values

```

```

void swap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

class CustomType {                                     // Custom data type
public:
    int value;
    CustomType(int v) : value(v) {}

    friend std::ostream& operator<<(std::ostream& os, const CustomType& obj) {
        os << obj.value;
        return os;
    }
};

int main() {
    int x = 8, y = 20;
    std::cout << "Before swap: x = " << x << ", y = " << y << std::endl;
    swap(x, y);
    std::cout << "After swap: x = " << x << ", y = " << y << std::endl;
    double a = 3.14, b = 2.71;
    std::cout << "Before swap: a = " << a << ", b = " << b << std::endl;
    swap(a, b);
    std::cout << "After swap: a = " << a << ", b = " << b << std::endl;
    CustomType obj1(7), obj2(10);
    std::cout << "Before swap: obj1 = " << obj1 << ", obj2 = " << obj2 << std::endl;
    swap(obj1, obj2);
    std::cout << "After swap: obj1 = " << obj1 << ", obj2 = " << obj2 << std::endl;
}

```

```

return 0;

}

```

OUTPUT :-

The screenshot shows a web browser window with the URL https://www.onlinegdb.com/online_c_compiler#. The code editor contains the following C++ code:

```

1 #include <iostream>
2 template<typename T>           // Template function to swap two values
3 void swap(T& a, T& b) {
4     T temp = a;
5     a = b;
6     b = temp;
7 }
8
9 class CustomType {           // Custom data type
10 public:
11     int value;
12     CustomType(int v) : value(v) {}
13
14     friend std::ostream& operator<<(std::ostream& os, const CustomType& obj) {
15         os << obj.value;
16         return os;
17     }
18 };
19
20 int main() {
21     int x = 8, y = 20;
22     std::cout << "Before swap: x = " << x << ", y = " << y << std::endl;
23     swap(x, y);
24     std::cout << "After swap: x = " << x << ", y = " << y << std::endl;
25     double a = 3.14, b = 2.71;
26     std::cout << "Before swap: a = " << a << ", b = " << b << std::endl;
27     swap(a, b);
28     std::cout << "After swap: a = " << a << ", b = " << b << std::endl;
29     CustomType obj1(7), obj2(10);
30     std::cout << "Before swap: obj1 = " << obj1 << ", obj2 = " << obj2 << std::endl;
31     swap(obj1, obj2);
32 }

```

The output window shows the following results:

```

Before swap: x = 8, y = 20
After swap: x = 20, y = 8
Before swap: a = 3.14, b = 2.71
After swap: a = 2.71, b = 3.14
Before swap: obj1 = 7, obj2 = 10
After swap: obj1 = 10, obj2 = 7
...Program finished with exit code 0
Press ENTER to exit console.

```

Consider a scenario where you need to find the minimum value in an array. Create a function template named `findMin` that works with any data type for which the comparison operator (`<`) is defined. Explain how function templates promote code reusability in this case.

```
#include <iostream>
```

```
template<typename T>
```

```
T findMin(T arr[], int size) {
```

```
    T min = arr[0];           // Assume the first element is the minimum initially
```

```
    for (int i = 1; i < size; ++i) {
```

```
        if (arr[i] < min) {
```

```
            min = arr[i];     // Update min if a smaller element is found
```

```
        }
```

```
    }
```

```
    return min;              // Return the minimum value
```

```
}
```

```

int main() {

    int intArray[] = {3, 1, 4, 1, 5, 9, 2, 6};           // Example usage with different
data types

    double doubleArray[] = {3.14, 2.71, 1.62, 6.28, 0.5};

    char charArray[] = {'a', 'b', 'c', 'A', 'B', 'C'};

    int minInt = findMin(intArray, 8);                  // Using the findMin
function template

    double minDouble = findMin(doubleArray, 5);

    char minChar = findMin(charArray, 6);

    std::cout << "Min integer: " << minInt << std::endl;

    std::cout << "Min double: " << minDouble << std::endl;

    std::cout << "Min char: " << minChar << std::endl;

    return 0;

}

```

OUTPUT :-

The screenshot shows a web browser window with an online C++ compiler. The code is pasted into the editor, and the output is displayed in the console. The output shows the minimum values for the integer, double, and character arrays.

```

Min integer: 1
Min double: 0.5
Min char: A

```

Below the output, a message states: "Program finished with exit code 0. Press ENTER to exit console."