

Objectives:

- To utilize control structures, functions, and built-in libraries in C++ programming.
- To perform calculations involving quadratic equations using the standard formula.
- To implement logic that checks if a triangle is valid and determines its type.
- To use string manipulation and character analysis for evaluating password strength.

Tools and Libraries Used:

- Programming Language: C++
- IDE: G++
- Libraries: `#include <iostream>`, `include <string>`, `#include <math>`

Theory:

Basics of C++ Programming

C++ is a versatile language used to build efficient programs. Beginners start by learning variables, conditional statements, and loops to solve simple problems.

Structure of a C++ Program

A basic C++ program includes header files (like `<iostream>`) and starts with `main()`, which is the entry point. The program uses `using namespace std;` to access standard features easily. The main function contains the code and ends with `return 0;` to indicate success.

Example:

```
1. #include<iostream>
2. using namespace std;
3. int main() {
4.     cout << "Hello world!";
5.     return 0;
6. }
```

Variables and Data Types

Variables store data. You can declare and assign them like this:

```
1. int age;           // Declaration
2. age = 20;          // Assignment
3. int score = 100;   // Declaration + Initialization
```

Common Data Types:

- int – whole numbers (e.g., int x = 5;)
- float – decimal numbers (e.g., float pi = 3.14;)
- double – more precise decimals (e.g., double d = 2.718;)
- char – single characters (e.g., char c = 'A';)

Variable Naming Rules

- Start with a letter or underscore
- No digits at the beginning
- No space or special characters (except _)
- Case-sensitive (Age ≠ age)

Conditional Statements

Conditional statements control program flow based on conditions.

if statement:

Used when we must check the condition.

Syntax:

```
1. if (condition) {  
2. // Code runs if condition is true  
3. }
```

if...else statement:

Used when we must check the condition and execute true and false condition separately.

Syntax:

```
1. if (condition) {  
2. // Runs if true  
3. } else {  
4. // Runs if false  
5. }
```

else...if ladder:

Used when multiple conditions are to be checked one after another.

Syntax:

```
1. if (condition1) {  
2. // code if condition1 is true  
3. } else if (condition2) {  
4. // code if condition2 is true  
5. } else if (condition3) {  
6. // code if condition3 is true  
7. } else {  
8. // code if none are true  
9. }
```

switch Statement:

Used to select one block of code from many options based on a variable's value.

Syntax:

```
1. switch (expression) {  
2.   case value1:  
3.     // code for case 1  
4.     break;  
5.   case value2:  
6.     // code for case 2  
7.     break;  
8.   ...  
9.   default:  
10.    // code if no cases match  
11. }
```

Loops in C++

for Loop

Used when the number of iterations is known.

Syntax:

```
1. for (initialization; condition; update) {  
2.   // code to repeat  
3. }
```

while Loop

Used when the condition is checked before the loop body and the number of repetitions is not fixed.

Syntax:

```
1. while (condition) {  
2.   // code to repeat  
3. }  
4.
```

do...while Loop

Runs the loop body at least once before checking the condition.

Syntax:

```
1. do {  
2.   // code to repeat  
3. } while (condition);
```

1. WAP to calculate the value of a variable from the given quadratic equation.

```
#include <iostream>

#include <cmath>

using namespace std;

int main() {

    double a, b, c;

    double discriminant, x1, x2;

    cout << "Enter coefficients a, b and c: ";

    cin >> a >> b >> c;

    if (a == 0) {

        cout << "This is not a quadratic equation (a cannot be 0)." << endl;

    }

    discriminant = b * b - 4 * a * c;

    if (discriminant > 0) {

        // Two real and distinct roots

        x1 = (-b + sqrt(discriminant)) / (2 * a);

        x2 = (-b - sqrt(discriminant)) / (2 * a);

        cout << "Roots are real and different." << endl;

        cout << "x1 = " << x1 << endl;

        cout << "x2 = " << x2 << endl;

    }
```

```

}

else if (discriminant == 0) {

    // One real root

    x1 = -b / (2 * a);

    cout << "Roots are real and same." << endl;

    cout << "x = " << x1 << endl;

}

else {

    // Complex roots

    double realPart = -b / (2 * a);

    double imaginaryPart = sqrt(-discriminant) / (2 * a);

    cout << "Roots are complex and imaginary." << endl;

    cout << "x1 = " << realPart << " + " << imaginaryPart << "i" << endl;

    cout << "x2 = " << realPart << " - " << imaginaryPart << "i" << endl;

}

return 0;

}

```

Outputs:

```
C:\Users\User\Documents\file X + v
Enter coefficients a, b and c: 1
2
3
Roots are complex and imaginary.
x1 = -1 + 1.41421i
x2 = -1 - 1.41421i

Process returned 0 (0x0)   execution time : 68.427 s
Press any key to continue.
```

```
C:\Users\User\Documents\file X + v
Enter coefficients a, b and c: 1
5
-6
Roots are real and different.
x1 = 1
x2 = -6

Process returned 0 (0x0)   execution time : 12.495 s
Press any key to continue.
```

```
C:\Users\User\Documents\file X + v
Enter coefficients a, b and c: 1
2
1
Roots are real and same.
x = -1

Process returned 0 (0x0)   execution time : 8.732 s
Press any key to continue.
|
```


2. WAP to check whether the given angles form acute, obtuse or right angled triangle.

```
#include <iostream>

using namespace std;

int main() {

    int angle1, angle2, angle3;

    cout << "Enter three angles of the triangle: ";

    cin >> angle1 >> angle2 >> angle3;

    int sum = angle1 + angle2 + angle3;

    // Check if the angles form a valid triangle

    if (sum != 180 || angle1 <= 0 || angle2 <= 0 || angle3 <= 0) {

        cout << "Invalid triangle. Angles do not sum up to 180 degrees." << endl;

    }

    else if (angle1 == 90 || angle2 == 90 || angle3 == 90) {

        cout << "The triangle is a right-angled triangle." << endl;

    }

    else if (angle1 > 90 || angle2 > 90 || angle3 > 90) {

        cout << "The triangle is an obtuse-angled triangle." << endl;
```

```

    }

    else {

        cout << "The triangle is an acute-angled triangle." << endl;

    }

    return 0;

}

```

Output

```

C:\Users\User\Documents\file X + v
Enter three angles of the triangle: 120
30
30
The triangle is an obtuse-angled triangle.

Process returned 0 (0x0)   execution time : 26.897 s
Press any key to continue.

```

```

C:\Users\User\Documents\file X + v
Enter three angles of the triangle: 120
0
56
Invalid triangle. Angles do not sum up to 180 degrees.

Process returned 0 (0x0)   execution time : 6.935 s
Press any key to continue.

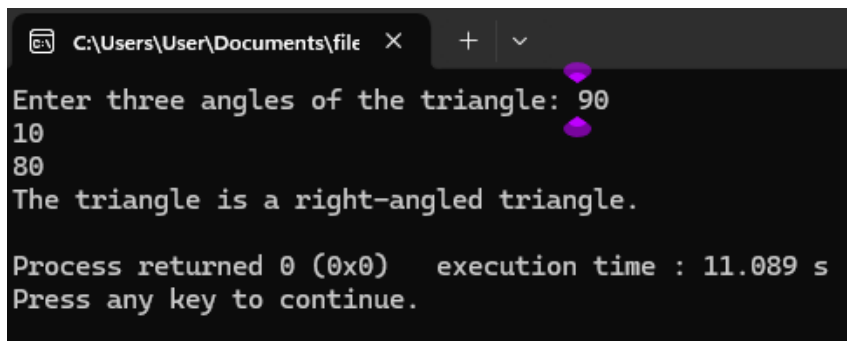
```

```

C:\Users\User\Documents\file X + v
Enter three angles of the triangle: 60
60
60
The triangle is an acute-angled triangle.

Process returned 0 (0x0)   execution time : 15.966 s
Press any key to continue.

```



```
C:\Users\User\Documents\file X + v
Enter three angles of the triangle: 90
10
80
The triangle is a right-angled triangle.

Process returned 0 (0x0)   execution time : 11.089 s
Press any key to continue.
```

3. WAP to check whether the entered password is a strong password or not. For a password to be strong it must have numbers, special keys, alpha keys(both caps and small).

```
#include<iostream>

#include<string>

using namespace std;

//special characters

int scharacter(string password,int len)

{

    int temp1=0;

    while(len>=0)

    {

        if(password[len]>=34&&password[len]<=47||password[len]>=58&&password[len]<=64||password[len]>=91&&password[len]<=96)

        {

            temp1++;

        }

        len--;

    }

    return temp1;

}
```

```

//caps letter

int caps(string password, int len)

{

    int temp2=0;

    while(len>=0)

    {

        if(password[len]>=65&&password[len]<=90)

        {

            temp2++;

        }

        len--;

    }

    return temp2;

}

//small letters

int small(string password, int len)

{

    int temp3=0;

```

```

while(len>=0)

{

if(password[len]>=97&&password[len]<=122)

{

    temp3++;

}

len--;

}

return temp3;

}

//numbers

int num(string password, int len)

{

    int temp4=0;

    while(len>=0)

    {

        if(password[len]>=48&&password[len]<=57)

        {

            temp4++;

        }

        len—

```

```

    }

    return temp4;

}

int main()

{

    int len,a,b,c,d;

    string password;

    cout<<"enter a strong password"<<endl;

    cin>>password;

    len=password.length();

    a=scharacter(password,len);

    b=caps(password, len);

    c=small(password, len);

    d=num(password,len);

    if(a>=1&&b>=1&&c>=1&&d>=1)

    {

        cout<<"**the password is strong**";

    }

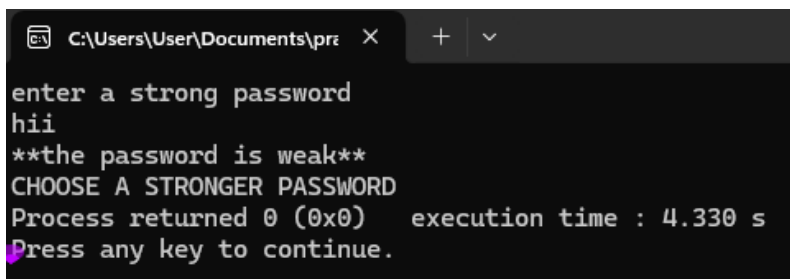
    else{

        cout<<"**the password is weak**"<<endl<<"CHOOSE A STRONGER
PASSWORD";

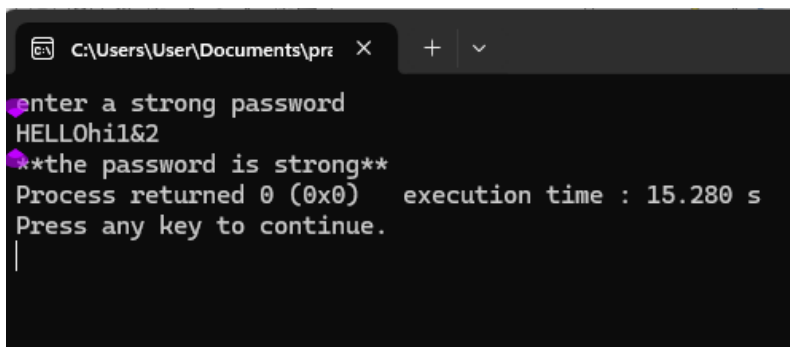
```

```
}  
  
return 0;  
  
}
```

Output:



```
C:\Users\User\Documents\prj X + v  
enter a strong password  
hii  
**the password is weak**  
CHOOSE A STRONGER PASSWORD  
Process returned 0 (0x0) execution time : 4.330 s  
Press any key to continue.
```



```
C:\Users\User\Documents\prj X + v  
enter a strong password  
HELLOhi1&2  
**the password is strong**  
Process returned 0 (0x0) execution time : 15.280 s  
Press any key to continue.  
|
```


Discussion:

We were able to familiarize ourselves with the several kinds of loops and control statements, including do-while, while, and if else, through this lab. Numerous issues that called for the appropriate application of control statements and logical reasoning were resolved. With the assistance of professors, the errors that transpired were resolved through appropriate code analysis..

Conclusion:

This lab examined the basic ideas and real-world uses of control statements in programming, such as loops (for, while, do-while) and conditional expressions (if, if-else, switch). These constructs are necessary for efficiently completing repetitive activities and for controlling a program's flow according to predetermined conditions. We gained practical experience in implementing decision-making logic and managing code block iteration.s