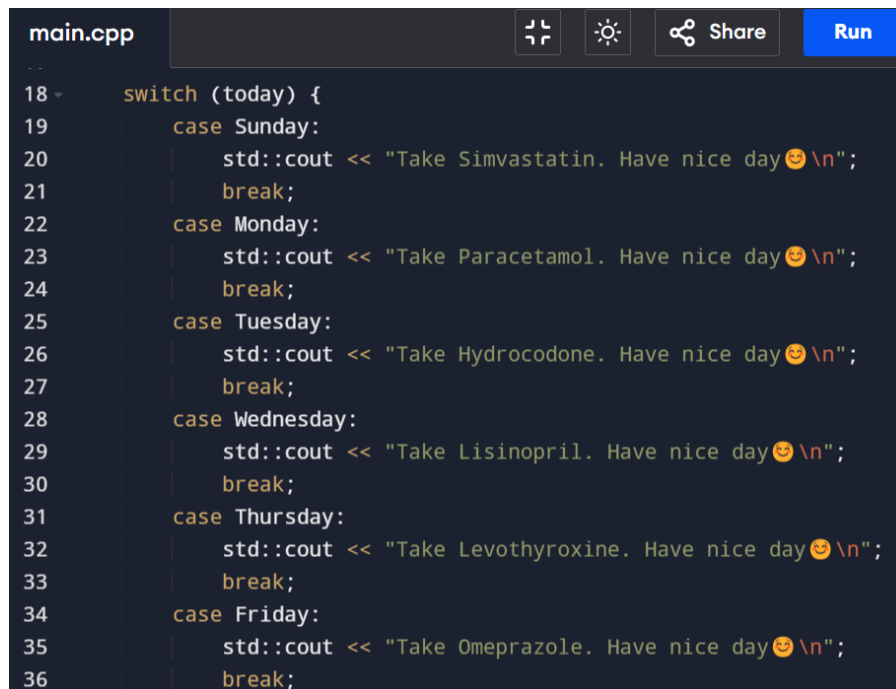


CPP NOTES – Day 05

switch Statement – Multiway Branching

```
switch (expression) {  
  case value_1:  
    // statements_break.  
    break;  
  case value_2:  
    // statements_2;  
    break;  
  default:  
    // default_statements;  
    break;  
}
```

- allows you to execute a block of code among many alternatives. It is an alternative to the long if-else-if ladder, providing a cleaner and more readable syntax.

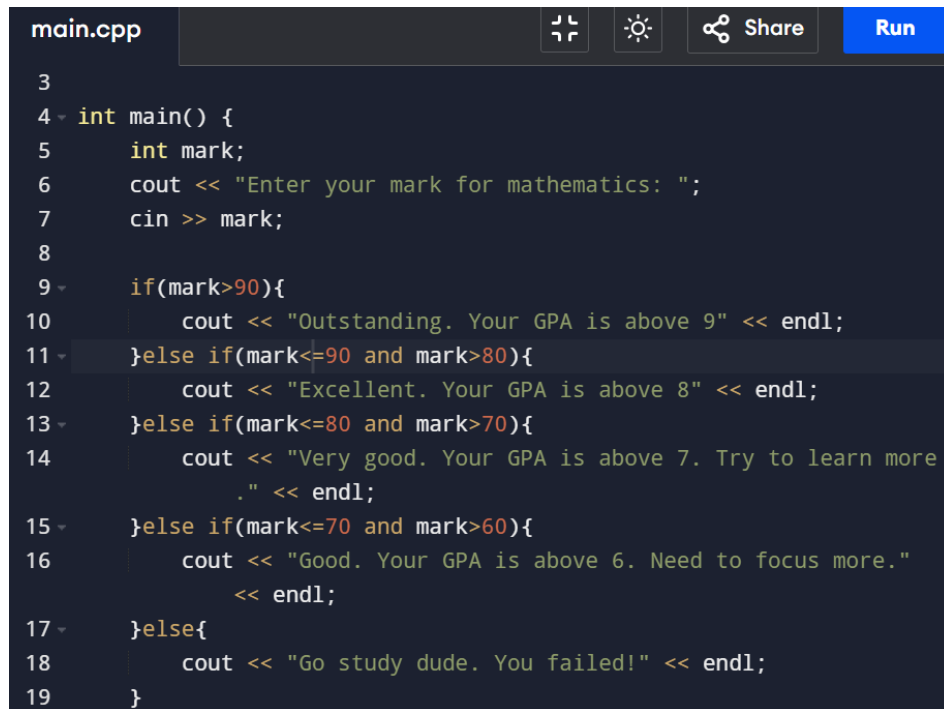


The screenshot shows a code editor window titled 'main.cpp'. The code is a C++ program that uses a switch statement to recommend a medicine based on the day of the week. The code is as follows:

```
18 switch (today) {  
19     case Sunday:  
20         std::cout << "Take Simvastatin. Have nice day 😊\n";  
21         break;  
22     case Monday:  
23         std::cout << "Take Paracetamol. Have nice day 😊\n";  
24         break;  
25     case Tuesday:  
26         std::cout << "Take Hydrocodone. Have nice day 😊\n";  
27         break;  
28     case Wednesday:  
29         std::cout << "Take Lisinopril. Have nice day 😊\n";  
30         break;  
31     case Thursday:  
32         std::cout << "Take Levothyroxine. Have nice day 😊\n";  
33         break;  
34     case Friday:  
35         std::cout << "Take Omeprazole. Have nice day 😊\n";  
36         break;
```

Else-if Statement

The else if statement in C++ is used to specify a new condition if the previous condition is false. This allows for multiple conditions to be checked sequentially.

A screenshot of a code editor window titled 'main.cpp'. The code is written in C++ and uses an else-if chain to evaluate a mark. The code is as follows:

```
3
4 int main() {
5     int mark;
6     cout << "Enter your mark for mathematics: ";
7     cin >> mark;
8
9     if(mark>90){
10        cout << "Outstanding. Your GPA is above 9" << endl;
11    }else if(mark<=90 and mark>80){
12        cout << "Excellent. Your GPA is above 8" << endl;
13    }else if(mark<=80 and mark>70){
14        cout << "Very good. Your GPA is above 7. Try to learn more" << endl;
15    }else if(mark<=70 and mark>60){
16        cout << "Good. Your GPA is above 6. Need to focus more." << endl;
17    }else{
18        cout << "Go study dude. You failed!" << endl;
19    }
```

Developer Test Cases

- Rainy- A rainy test case is a "negative" test case that is designed to test the system's behavior under unusual, invalid, or unexpected conditions. It's often used to check how the system handles errors or failures.

Eg: In the login functionality, a rainy test case could involve a user entering an incorrect password or a non-existent username to see if the system displays the appropriate error message.

- Sunny - A sunny test case is a "happy path" test case where everything works as expected under normal, ideal conditions. It's the scenario where all inputs are valid, and the system behaves as intended without any errors or exceptions.

Eg: In a login functionality, a sunny test case would be when a user enters a valid username and password and successfully logs in.

```
main.cpp  Run  Output
5  string genre;
6  cout << "Enter the genre: ";
7  cin >> genre;
8  if(genre[0] >= '0' && genre[0] <= '9'){
9      cout << "Genre can't start with numeric characters" << endl;
10     return 0;
11 }
12 for(char i=0; i<=genre.length(); i++){
13     if(!isalpha(genre[i])){
14         cout << "Genre can't contain special characters." << endl;
15         return 0;
16     }
17 }
18 if(genre.compare("Action") == 0){
19     cout << "Selected genre is Action" << endl;
20 } else if(genre.compare("Comedy") == 0){
21     cout << "Selected genre is Comedy" << endl;
22 } else if(genre.compare("Drama") == 0){
    ^ Enter the genre: $del
    Genre can't contain special characters.

    === Code Execution Successful ===
```

- This code shows the handling of string validation along with else if statements.

Lambda – a function without a name.

Best practices

- Avoid deep nesting
- Use switch with enum class
- Prefer if const expr for simple cases
- Ternary operator for simple cases
- Don't forget break in switch

Goto Statement

statement in C++ is a jump statement that allows you to jump to another part of the program by specifying a labeled statement. It provides an unconditional jump from the goto statement to a label within the same function.

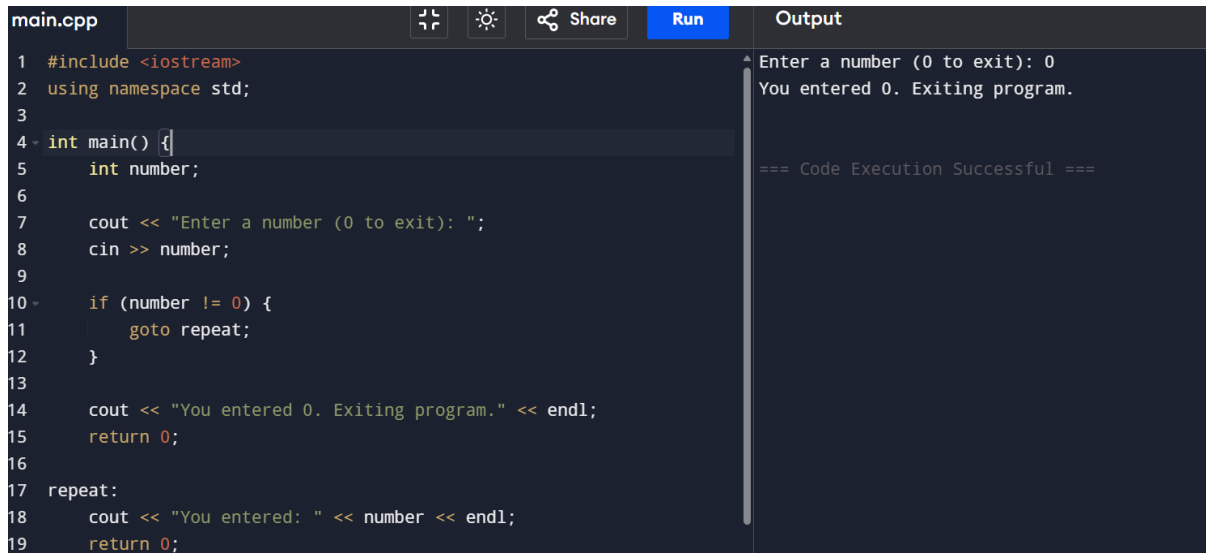
- Can be used to getout of a nested loop

goto label;

// ... some code

label:

// code to jump to



The screenshot shows a C++ IDE with a file named `main.cpp`. The code implements a loop using the `goto` statement. It prompts the user to enter a number, and if the number is not 0, it jumps back to the start of the loop. The output shows the user entering 0, and the program exiting successfully.

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int number;
6
7     cout << "Enter a number (0 to exit): ";
8     cin >> number;
9
10    if (number != 0) {
11        goto repeat;
12    }
13
14    cout << "You entered 0. Exiting program." << endl;
15    return 0;
16
17 repeat:
18    cout << "You entered: " << number << endl;
19    return 0;
```

Output:

```
Enter a number (0 to exit): 0
You entered 0. Exiting program.

=== Code Execution Successful ===
```

Flags

flags are commonly used as indicators that signal whether a certain condition has occurred or if a particular state is active within a program. Flags are typically represented using boolean variables (or sometimes integers, where non-zero values represent "true" and zero represents "false") that can be checked during the flow of the program.

Flags can be used to:

- Control the flow of the program based on specific conditions.
- Indicate whether a certain event or condition has been met, helping the program decide what to do next.