PW SKILLS

Lesson Plan

Loops - 2

PW SKILLS

Today's checklist:

- Break statement
- Continue statement
- Problems on operators (+,-,*,/)

Break Statement:

The break in C++ is a loop control statement that is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there and control returns from the loop immediately to the first statement after the loop.

Syntax:

break;

Basically, break statements are used in situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.

We will see here the usage of break statements with three different types of loops:

- Simple loops
- Nested loops
- Infinite loops

Let us now look at the examples for each of the above three types of loops using a break statement.

Break with Simple loops

Consider the situation where we want to search for an element in an array. To do this, use a loop to traverse the array starting from the first index and compare the array elements with the given key.

Example:

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

void findElement(int arr[], int size, int key)
{
    // loop to traverse array and search for key
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            cout << "Element found at position: " << (i + 1);
        }
    }
}

int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6 };
```

```
    int n = 6; // no of elements
    int key = 3; // key to be searched

    // Calling function to find the key
    findElement(arr, n, key);

    return 0;
}
```

Output:
Element found at position: 3

The above code runs fine with no errors. But the above code is not efficient. The above code completes all the iterations even after the element is found. Suppose there are 1000 elements in the array and the key to be searched is present at 1st position so the above approach will execute 999 iterations which are of no purpose and are useless. To avoid these useless iterations, we can use the break statement in our program. Once the break statement is encountered the control from the loop will return immediately after the condition gets satisfied. So will use the break statement with the if condition which compares the key with array elements as shown below:

Example:

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

void findElement(int arr[], int size, int key)
{
    // loop to traverse array and search for key
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            cout << "Element found at position: "
                 << (i + 1);

            // using break to terminate loop execution
            break;
        }
    }
}
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6 };
    int n = 6;
    // key to be searched
    int key = 3;

    findElement(arr, n, key);

    return 0;
}
```

Output:
Element found at position: 3

```
    int n = 6; // no of elements
    int key = 3; // key to be searched

    // Calling function to find the key
    findElement(arr, n, key);

    return 0;
}
```

Break with Nested Loops

We can also use break statements while working with nested loops. If the break statement is used in the innermost loop. The control will come out only from the innermost loop.

Example:

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

int main()
{

    // nested for loops with break statement
    // at inner loop
    for (int i = 0; i < 5; i++) {
        for (int j = 1; j <= 10; j++) {
            if (j > 3)
                break;
            else
                cout << "*";
        }
        cout << endl;
    }

    return 0;
}
```

Output

```
***
***
***
***
***
```

In the above code, we can clearly see that the inner loop is programmed to execute for 10 iterations. But as soon as the value of j becomes greater than 3 the inner loop stops executing which restricts the number of iterations of the inner loop to 3 iterations only. However, the iteration of the outer loop remains unaffected. Therefore, break applies to only the loop within which it is present.

Break with Infinite Loops

The break statement can be included in an infinite loop with a condition in order to terminate the execution of the infinite loop.

Example:

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

int main()
{
    // loop initialization expression
    int i = 0;

    // infinite while loop
    while (1) {
        cout << i << " ";
        i++;
    }

    return 0;
}
```

Output:
Execution timed out

Note: Please do not run the above program in your compiler as it is an infinite loop so you may have to forcefully exit the compiler to terminate the program.

In the above program, the loop condition based on which the loop terminates is always true. So, the loop executes an infinite number of times. We can correct this by using the break statement as shown below:

Example:

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

int main()
{
    // loop initialization expression
    int i = 1;

    // infinite while loop
```

```
    while (1) {
        if (i > 10)
            break;

        cout << i << " ";
        i++;
    }

    return 0;
}
```

Output:
1 2 3 4 5 6 7 8 9 10

The above code restricts the number of loop iterations to 10. Apart from this, the break can be used in Switch case statements too.

Q1. Check if a number is prime or not.

```
#include <iostream>

using namespace std;

int main() {
    int num;

    // Input a number
    cout << "Enter a number: ";
    cin >> num;

    // Check if the number is prime
    if (num <= 1) {
        cout << num << " is not a prime number." << endl;
    } else {
        bool isPrime = true;

        for (int i = 2; i <= num / 2; ++i) {
            if (num % i == 0) {
                isPrime = false;
                break;  // Break the loop if a divisor is found
            }
        }

        if (isPrime) {
            cout << num << " is a prime number." << endl;
        } else {
            cout << num << " is not a prime number." << endl;
        }
    }

    return 0;
}
```

Continue Statement

The continue statement in C++ is used to skip the rest of the code inside a loop and move to the next iteration of the loop. It is often used to skip specific iterations based on certain conditions. Here's an example of using the continue statement in a loop:

Code:

```
#include <iostream>

using namespace std;

int main() {
 // Print even numbers from 1 to 10 using continue statement
 for (int i = 1; i ⩽ 10; ++i) {
 // Skip odd numbers
 if (i % 2 =≠ 0) {
 continue;
 }
 // The code below will be skipped for odd numbers
 cout << i << " is an even number." << endl;
 }

 return 0;
}
```

In this example, the loop iterates from 1 to 10. The if statement checks if the current value of i is an odd number (i % 2 != 0). If it is, the continue statement is executed, and the remaining code inside the loop is skipped for that iteration. As a result, only even numbers are printed in the output.

Output:
2 is an even number.
4 is an even number.
6 is an even number.
8 is an even number.
10 is an even number.

The continue statement is particularly useful when you want to skip certain iterations based on a condition without terminating the entire loop.

Q2. Print odd numbers from 1 to 100.

Code:

```
#include <iostream>

using namespace std;

int main() {
    // Print odd numbers from 1 to 100 using continue statement
    for (int i = 1; i ⩽ 100; ++i) {
        // Skip even numbers
```

```
        if (i % 2 == 0) {
            continue;
        }

        // The code below will be skipped for even numbers
        cout << i << " is an odd number." << endl;
    }

    return 0;
}
```

Q3. Count digits of a given number.

Code:

```
#include <iostream>

using namespace std;

int main() {
    int number, count = 0;

    // Input a number
    cout << "Enter a number: ";
    cin >> number;

    // Handle negative numbers by taking the absolute value
    number = abs(number);

    // Count the digits
    while (number > 0) {
        number = number / 10;
        ++count;
    }

    cout << "Number of digits: " << count << endl;

    return 0;
}
```

Q4. Print sum of digits of a given number.

Code:

```
#include <iostream>

using namespace std;

int main() {
    int number, sum = 0;
```

```
    // Input a number
    cout << "Enter a number: ";
    cin >> number;

    // Handle negative numbers by taking the absolute value
    number = abs(number);

    // Calculate the sum of digits
    while (number > 0) {
        sum += number % 10;  // Extract the last digit and add it
to the sum
        number = number / 10; // Remove the last digit
    }

    cout << "Sum of digits: " << sum << endl;

    return 0;
}
```

Q5. Print reverse of a given number.

Code:

```
#include <iostream>

using namespace std;

int main() {
    int number, reversedNumber = 0;

    // Input a number
    cout << "Enter a number: ";
    cin >> number;

    // Reverse the number
    while (number > 0) {
        int digit = number % 10;  // Extract the last digit
        reversedNumber = reversedNumber * 10 + digit;  // Append
the digit to the reversed number
        number = number / 10;  // Remove the last digit
    }

    cout << "Reversed number: " << reversedNumber << endl;

    return 0;
}
```

Q6. Print the sum of this series :
1 - 2 + 3 - 4 + 5 - 6... upto 'n'.

Code:

```
#include <iostream>

using namespace std;

int main() {
    int n;

    // Input the value of 'n'
    cout << "Enter the value of 'n': ";
    cin >> n;

    // Calculate the sum of the series
    int sum = 0;
    for (int i = 1; i <= n; ++i) {
        if (i % 2 == 0) {
            // Subtract for even terms
            sum -= i;
        } else {
            // Add for odd terms
            sum += i;
        }
    }

    // Print the result
    cout << "Sum of the series: " << sum << endl;

    return 0;
}
```

Q7. Print the factorial of a given number ‘n’.

Code:

```
#include <iostream>

using namespace std;

int main() {
    int n;

    // Input a number
    cout << "Enter a number: ";
    cin >> n;

    // Calculate the factorial
    if (n < 0) {
        cout << "Factorial is not defined for negative numbers."
<< endl;
```

```
    } else {
        long long factorial = 1;

        for (int i = 1; i <= n; ++i) {
            factorial *= i;
        }

        cout << "Factorial of " << n << " is: " << factorial <<
endl;
    }

    return 0;
}
```

Q8. Print the nth fibonacci number.

Code:

```
#include <iostream>

using namespace std;

int main() {
    int n;

    // Input the value of 'n'
    cout << "Enter the value of 'n': ";
    cin >> n;

    // Calculate the nth Fibonacci number
    if (n < 0) {
        cout << "Fibonacci number is not defined for negative
values of 'n'." << endl;
    } else {
        int fibN, fibNMinus1 = 0, fibNMinus2 = 1;

        if (n == 0) {
            fibN = 0;
        } else if (n == 1) {
            fibN = 1;
        } else {
            for (int i = 2; i <= n; ++i) {
                fibN = fibNMinus1 + fibNMinus2;
                fibNMinus2 = fibNMinus1;
                fibNMinus1 = fibN;
            }
        }
        cout << "The " << n << "th Fibonacci number is: " << fibN
<< endl;
    }

    return 0;
}
```

Q9. Two numbers are entered through the keyboard. Write a program to find the value of one number raised to the power of another.

Code:

```
#include <iostream>

using namespace std;

// Function to calculate power without using pow function
double calculatePower(double base, int exponent) {
    double result = 1.0;

    if (exponent > 0) {
        for (int i = 1; i ⩽ exponent; ++i) {
            result *= base;
        }
    } else if (exponent < 0) {
        for (int i = 1; i ⩽ -exponent; ++i) {
            result /= base;
        }
    }

    return result;
}

int main() {
    double base;
    int exponent;

    // Input base and exponent
    cout << "Enter the base: ";
    cin >> base;

    cout << "Enter the exponent: ";
    cin >> exponent;

    // Calculate the result using the custom function
    double result = calculatePower(base, exponent);

    // Print the result
    cout << base << " raised to the power of " << exponent << "
is: " << result << endl;

    return 0;
}
```

PW SKILLS

THANK YOU !

PW SKILLS