**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **22/06/2020** | **Name:** | **Dhavala** |
| **Course:** | **C++ Programming** | **USN:** | **4AL17EC027** |
| **Topic:** | * **Module 1: Basic Concepts** | **Semester & Section:** | **6TH SEM & A Section** |
| **Github Repository:** | **Dhavala27** |  |  |

|  |
| --- |
| **SESSION DETAILS** |
|  |

|  |
| --- |
| **Report** C++ C++ is a general-purpose programming language. C++ is used to create computer programs. Anything from art applications, music players and even video games! **First C++ Program**  C++ program is a collection of commands or statements. Below is a simple code that has "Hello world!" as its output.  #include <iostream> using namespace std; int main() { cout << "Hello world!"; return 0; }  **Main** Program execution begins with the main function, **int main()**.  #include <iostream> using namespace std; **int main() { cout << "Hello world!"; return 0; }**  **Statements** A **block**is a set of logically connected statements, surrounded by opening and closing curly braces. For example:{ cout << "Hello world!";  return 0; }  **Return** The last instruction in the program is the **return**statement. The line **return 0;** terminates the **main()** function and causes it to return the value 0 to the calling process. A non-zero value (usually of 1) signals abnormal termination.  #include <iostream> using namespace std; int main() { cout << "Hello world!"; **return 0;** }  **Arithmetic Operators** C++ supports these arithmetic operators.  **Assignment Operators** The simple **assignment**operator (=) assigns the right side to the left side. C++ provides shorthand operators that have the capability of performing an operation and an assignment at the same time. **For example:**int x = 10; **x += 4**; // equivalent to x = x + 4 **x -= 5**; // equivalent to x = x – 5  **Assignment Operators** The same shorthand syntax applies to the multiplication, division, and modulus operators.  x \*= 3; // equivalent to x = x \* 3 x /= 2; // equivalent to x = x / 2 x %= 4; // equivalent to x = x % 4  **Increment Operator** The **increment**operator is used to increase an integer's value by one, and is a commonly used C++ operator. **x++;** //equivalent to x = x + 1 |

**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **22/06/2020** | **Name:** | **Dhavala** |
| **Course:** | **C++ Programming** | **USN:** | **4AL17EC027** |
| **Topic:** | * **Module 2: Conditionals and loops** | **Semester & Section:** | **6TH SEM & A Section** |
| **Github Repository:** | **Dhavala27** |  |  |

|  |
| --- |
| **SESSION DETAILS** |
|  |

|  |
| --- |
| **Report** Decision Making The if statement is used to execute some code if a condition is true. Syntax:if (condition) { statements } The condition specifies which expression is to be evaluated. If the condition is true, the statements in the curly brackets are executed. **The if Statement** Use relational operators to evaluate conditions. For example:  if (7 > 4) { cout << "Yes";  } // Outputs "Yes"  **Relational Operators** Additional relational operators:  **The for loop** A for loop is a repetition control structure that allows you to efficiently write a loop that executes a specific number of times. Syntax:  for ( init; condition; increment ) { statement(s); } The init step is executed first, and does not repeat. Next, the condition is evaluated, and the body of the loop is executed if the condition is true. In the next step, the increment statement updates the loop control variable. Then, the loop's body repeats itself, only stopping when the condition becomes false. For example:  for (int x = 1; x < 10; x++) { // some code }  **The do...while Loop** Unlike for and while loops, which test the loop condition at the top of the loop, the do...while loop checks its condition at the bottom of the loop. A do...while loop is similar to a while loop. The one difference is that the do...while loop is guaranteed to execute at least one time. Syntax:  do { statement(s); } while (condition);  **while vs. do...while** If the condition evaluated to false, the statements in the do would still run once:  int a = 42; do { cout << a << endl; a++; } while(a < 5); // Outputs 42  **The do...while Loop** As with other loops, if the condition in the loop never evaluates to false, the loop will run forever. For example:  int a = 42; do { cout << a << endl; } while (a > 0);  **Multiple Conditions** Sometimes there is a need to test a variable for equality against multiple values. That can be achieved using multiple if statements. **For example:**  int age = 42; if (age == 16) { cout <<"Too young"; } if (age == 42) { cout << "Adult"; } if (age == 70) { cout << "Senior"; }  **The break Statement**  The break statement's role is to terminate the switch statement. In instances in which the variable is equal to a case, the statements that come after the case continue to execute until they encounter a break statement. In other words, leaving out a break statement results in the execution of all of the statements in the following cases, even those that don't match the expression. For example:  int age = 42; switch (age) { case 16: cout << "Too young" << endl; case 42: cout << "Adult" << endl; case 70: cout << "Senior" << endl; default: cout <<"This is the default case" << endl; } /\* Outputs Adult Senior This is the default case \*/ |