**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **22/06/2020** | **Name:** | **Navya R** |
| **Course:** | **C++** | **USN:** | **4AL16EC041** |
| **Topic:** | **Basic concepts**  **Conditions and loops** | **Semester & Section:** | **8 A** |
| **Github Repository:** | **Navya-R** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  **MODULE 1**  C++ is a general-purpose programming language. C++ is used to create computer programs. Anything from art applications, music players and even video games! **New Line** The **cout**operator does not insert a line break at the end of the output. One way to print two lines is to use the **endl**manipulator, which will put in a line break.  #include <iostream> using namespace std;  int main() { cout << "Hello world!" << **endl**; cout << "I love programming!"; return 0; } **Multiple New Lines** Using a single **cout**statement with as many instances of **\n** as your program requires will print out multiple lines of text.  #include <iostream> using namespace std;  int main() { cout << " Hello **\n** world! **\n** I **\n** love **\n** programming!"; return 0; } **Variables** Creating a **variable**reserves a memory location, or a space in memory for storing values. The compiler requires that you provide a **data type** for each variable you declare. C++ offer a rich assortment of built-in as well as user defined **data types**.  **Integer**, a built-in type, represents a whole number value. Define integer using the keyword **int**. C++ requires that you specify the **type**and the **identifier**for each variable defined.An **identifier**is a name for a variable, function, class, module, or any other user-defined item. An identifier starts with a letter (A-Z or a-z) or an underscore (\_), followed by additional letters, underscores, and digits (0 to 9). For example, define a variable called **myVariable**that can hold **integer**values as follows:**int** myVariable = 10;  a program to calculate and print the sum of two integers.  #include <iostream> using namespace std;  int main() { int a = 30; int b = 12; int sum = a + b;  **cout << sum;**  return 0; }  //Outputs 42 **Arithmetic Operators** C++ supports these arithmetic operators. The addition operator adds its operands together.  Int x = 40 + 60; cout << x;  // Outputs **100** 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  **MODULE 2**   **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. If the condition is **false**, the statements are simply ignored, and the program continues to run after the if statements body. **The if Statement** Use **relational operators** to evaluate conditions.  **For example:**  **if (7 > 4)** { cout << "Yes";  }  // Outputs "Yes"  The **if** statement evaluates the condition (7>4), finds it to be **true**, and then executes the **cout**statement. If we change the greater operator to a less than operator (**7<4**), the statement will not be executed and nothing will be printed out. A condition specified in an if statement does not require a semicolon. **The else Statement** An **if** statement can be followed by an optional **else**statement, which executes when the condition is **false**. **Syntax:if** (condition) { //statements } **else** { //statements } The code above will test the condition: - If it evaluates to **true**, then the code inside the **if**statement will be executed. - If it evaluates to **false**, then the code inside the **else**statement will be executed. When only **one**statement is used inside the if/else, then the curly braces can be omitted. **Loops** A **loop**repeatedly executes a set of statements until a particular condition is satisfied.  A **while**loop statement repeatedly executes a target statement as long as a given condition remains **true**. **Syntax:while** (condition) { statement(s); } The loop iterates while the condition is **true**. At the point when the condition becomes **false**, program control is shifted to the line that immediately follows the loop. **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); **The switch Statement** The **switch**statement tests a variable against a list of values, which are called **cases**, to determine whether it is equal to any of them.**switch** (expression) { **case** value1: statement(s); **break;** **case** value2: statement(s); **break;** ... **case** valueN: statement(s); **break;** } Switch evaluates the expression to determine whether it's equal to the value in the case statement. If a match is found, it executes the statements in that case. |