**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **23/06/2020** | **Name:** | **Nayanashree K S** |
| **Course:** | **C++** | **USN:** | **4AL16EC042** |
| **Topic:** | **Data types, arrays, pointers**  **Functions** | **Semester & Section:** | **8 A** |
| **Github Repository:** | **nayana\_online** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  **MODULE 3** **Data Types** The operating system allocates memory and selects what will be stored in the reserved memory based on the variable's data type. The data type defines the proper use of an identifier, what kind of data can be stored, and which types of operations can be performed. There are a number of built-in types in C++. Tap Continue to learn more! **Strings & Characters** A string is composed of numbers, characters, or symbols. String literals are placed in double quotation marks; some examples are "Hello", "My name is David", and similar.  Characters are single letters or symbols, and must be enclosed between single quotes, like 'a', 'b', etc. In C++, single quotation marks indicate a character; double quotes create a string literal. While 'a' is a single a character literal, "a" is a string literal. **Variable Naming Rules** Use the following rules when naming variables: - All variable names must begin with a letter of the alphabet or an underscore( \_ ). - After the initial letter, variable names can contain additional letters, as well as numbers. Blank spaces or special characters are not allowed in variable names. There are two known naming conventions: Pascal case: The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized. For example: BackColor Camel case: The first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalized. For example: backColor **Arrays** An array is used to store a collection of data, but it may be useful to think of an array as a collection of variables that are all of the same type. Instead of declaring multiple variables and storing individual values, you can declare a single array to store all the values. When declaring an array, specify its element types, as well as the number of elements it will hold. For example:int a[5]; In the example above, variable a was declared as an array of five integer values [specified in square brackets]. You can initialize the array by specifying the values it holds:int b[5] = {11, 45, 62, 70, 88}; The values are provided in a comma separated list, enclosed in {curly braces}. **Multi-Dimensional Arrays** A multi-dimensional array holds one or more arrays. Declare a multidimensional array as follows.type name[size1][size2]...[sizeN]; Here, we've created a two-dimensional 3x4 integer array:int x[3][4]; **Pointers** Every variable is a memory location, which has its address defined. That address can be accessed using the ampersand (&) operator (also called the address-of operator), which denotes an address in memory.  For example:  int score = 5; cout << &score << endl;  //Outputs "0x29fee8"  **MODULE 4**  **Functions**   **Functions** A function is a group of statements that perform a particular task. You may define your own functions in C++.  Using functions can have many advantages, including the following: - You can reuse the code within a function. - You can easily test individual functions. - If it's necessary to make any code modifications, you can make modifications within a single function, without altering the program structure. - You can use the same function for different inputs. **The Return Type** The main function takes the following general form:int main() { // some code return 0; } A function's return type is declared before its name. In the example above, the return type is int, which indicates that the function returns an integer value. Occasionally, a function will perform the desired operations without returning a value. Such functions are defined with the keyword void. **Defining a Function** As an example, let's define a function that does not return a value, and just prints a line of text to the screen.void printSomething()  { cout << "Hi there!"; } Our function, entitled printSomething, returns void, and has no parameters. Now, we can use our function in main().  int main()  { printSomething();  return 0; } **Function Parameters** For a function to use arguments, it must declare formal parameters, which are variables that accept the argument's values.  For example:void printSomething(int x)  { cout << x; } This defines a function that takes one integer parameter and prints its value. **Random Numbers** Being able to generate random numbers is helpful in a number of situations, including when creating games, statistical modeling programs, and similar end products.  In the C++ standard library, you can access a pseudo random number generator function that's called rand(). When used, we are required to include the header <cstdlib>.  #include <iostream> #include <cstdlib> using namespace std;  int main() { cout << rand(); } **Default Values for Parameters** When defining a function, you can specify a default value for each of the last parameters. If the corresponding argument is missing when you call a function, it uses the default value.  To do this, use the assignment operator to assign values to the arguments in the function definition, as shown in this example.int sum(int a, int b=42) { int result = a + b; return (result); } This assigns a default value of 42 to the b parameter. If we call the function without passing the value for the b parameter, the default value will be used.  int main() { int x = 24; int y = 36;  //calling the function with both parameters int result = sum(x, y); cout << result << endl; //Outputs 60  //calling the function without b result = sum(x); cout << result << endl; //Outputs 66  return 0; } **Overloading** Function overloading allows to create multiple functions with the same name, so long as they have different parameters.  For example, you might need a printNumber() function that prints the value of its parameter.void printNumber(int a) {  cout << a; } This is effective with integer arguments only. Overloading it will make it available for other types, such as floats.void printNumber(float a) {  cout << a; } **Recursion** A recursive function in C++ is a function that calls itself.  To avoid having the recursion run indefinitely, you must include a termination condition. |