**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **23/June/2020** | **Name:** | **Prashantha naik** |
| **Course:** | **C++ (Sololearn)** | **USN:** | **4al17ec074** |
| **Topic:** | **Module 3: Data Types, Arrays, Pointers** | **Semester & Section:** | **6th b** |
| **GitHub Repository:** | **prashanth\_course** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report – Report can be typed or hand written for up to two pages.**  **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**.**  **Integers**  The integer type holds non-fractional numbers, which can be positive or negative. Examples of integers would include 42, -42, and similar numbers.  Use the int keyword to define the integer data type.  int a = 42;  Several of the basic types, including integers, can be modified using one or more of these type modifiers:  signed: A signed integer can hold both negative and positive numbers.  unsigned: An unsigned integer can hold only positive values.  short: Half of the default size.  long: Twice the default size.  **Floating Point Numbers**  A floating point type variable can hold a real number, such as 420.0, -3.33, or 0.03325.  The words floating point refer to the fact that a varying number of digits can appear before and after the decimal point. You could say that the decimal has the ability to "float".  There are three different floating point data types: float, double, and long double.  In most modern architectures, a float is 4 bytes, a double is 8, and a long double can be equivalent to a double (8 bytes), or 16 bytes.  For example:  double temp = 4.21;  **Strings**  A string is an ordered sequence of characters, enclosed in double quotation marks.  It is part of the Standard Library.  You need to include the <string> library to use the string data type. Alternatively, you can use a library that includes the string library.  #include <string>  using namespace std;  int main() {  string a = "I am learning C++";  return 0;  }  **Characters**  A char variable holds a 1-byte integer. However, instead of interpreting the value of the char as an integer, the value of a char variable is typically interpreted as an ASCII character.  A character is enclosed between single quotes (such as 'a', 'b', etc).  For example:  char test = 'S';  **Booleans**  Boolean variables only have two possible values: true (1) and false (0).  To declare a boolean variable, we use the keyword bool.  bool online = false;  bool logged\_in = true;  **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}.  **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**"**  **Static & Dynamic Memory**  To be successful as a C++ programmer, it's essential to have a good understanding of how dynamic memory works.  In a C++ program, memory is divided into two parts:  The stack: All of your local variables take up memory from the stack.  The heap: Unused program memory that can be used when the program runs to dynamically allocate the memory.  Many times, you are not aware in advance how much memory you will need to store particular information in a defined variable and the size of required memory can be determined at run time.  You can allocate memory at run time within the heap for the variable of a given type using the new operator, which returns the address of the space allocated.  new int;  **sizeof** While the size allocated for varying data types depends on the architecture of the computer you use to run your programs, C++ does guarantee a minimum size for the basic data types: |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **23/June/2020** | **Name:** | **Prashantha naik** | |
| **Course:** | **C++ (Sololearn)** | **USN:** | **4al17ec074** | |
| **Topic:** | **Module 4: Functions** | **Semester&Section:** | **6th b** | |
| **Git hub repository** | **prashanth\_couse** |  |  | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Report – Report can be typed or hand written for up to two pages.**  **Functions**  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**  Define a C++ function using the following syntax:return\_type function\_name( parameter list ) { body of the function } return-type: Data type of the value returned by the function. function name: Name of the function. parameters: When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. body of the function: A collection of statements defining what the function does.  **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(); }  **Function 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 demonstrate recursion, let's create a program to calculate a number's factorial. In mathematics, the term factorial refers to the product of all positive integers that are less than or equal to a specific non-negative integer (n). The factorial of n is denoted as n! For example:4! = 4 \* 3 \* 2 \* 1 = 24  Arrays and Functions An array can also be passed to a function as an argument. The parameter should be defined as an array using square brackets, when declaring the function. For example:  void printArray(int arr[], int size) { for(int x=0; x<size; x++) { cout <<arr[x]; } }  **Function Arguments**  There are two ways to pass arguments to a function as the function is being called.  By value: This method copies the argument's actual value into the function's formal parameter. Here, we can make changes to the parameter within the function without having any effect on the argument.  By reference: This method copies the argument's reference into the formal parameter. Within the function, the reference is used to access the actual argument used in the call. This means that any change made to the parameter affects the argument.  **Passing by Reference**  Pass-by-reference copies an argument's address into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument. To pass the value by reference, argument pointers are passed to the functions just like any other value.  void myFunc(int \*x) { \*x = 100; }  int main() { int var = 20; myFunc(&var); cout << var; } // Outputs 100 | | | |