**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **23 June 2020** | **Name:** | **Kavya M M** |
| **Course:** | **C++ PROGRAMMING** | **USN:** | **4AL17EC040** |
| **Topic:** | **Module 3: data types, arrays, pointers**  **Module 4: functions** | **Semester & Section:** | **6TH SEM & ‘A’ SEC** |
| **Github Repository:** | **Kavya\_ECE040** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **C:\Users\cw\Desktop\23 j2.png** |
| **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.  **Expressions**  The examples below show legal and illegal C++ expressions.55+15 // legal C++ expression //Both operands of the + operator are integers  55 + "John" // illegal // The + operator is not defined for integer and string  **Numeric Data Types**  Numeric data types include: Integers (whole numbers), such as -7, 42. Floating point numbers, such as 3.14, -42.67.  **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.  **Booleans**  The Boolean data type returns just two possible values: true (1) and false (0).  **Integers**  The integer type holds non-fractional numbers, which can be positive or negative. Examples of integers would include 42, -42, and similar numbers.  Integers  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.  **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';  **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.  Case-Sensitivity  C++ is case-sensitive, which means that an identifier written in uppercase is not equivalent to another one with the same name in lowercase.  For example, *myvariable* is not the same as *MYVARIABLE* and not the same as *MyVariable*.  These are three different variables.  Variable Naming Rules  C++ keyword (reserved word) cannot be used as variable names. For example, int, float, double, cout cannot be used as a variable name.  **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.  Initializing Arrays  If you omit the size of the array, an array just big enough to hold the initialization is created.  For example: int b[] = {11, 45, 62, 70, 88}; This creates an identical array to the one created in the previous example.  Each element, or member, of the array has an index, which pinpoints the element's specific position. The array's first member has the index of 0, the second has the index of 1.  So, for the array b that we declared above: https://api.sololearn.com/DownloadFile?id=2459 To access array elements, index the array name by placing the element's index in square brackets following the array name. For example:  int b[] = {11, 45, 62, 70, 88};  cout << b[0] << endl; // Outputs 11  cout<< b[3] << endl; // Outputs 70  Arrays in Loops  Let's declare an array, that is going to store 5 integers, and assign a value to each element using the for loop: int myArr[5];  for(int x=0; x<5; x++) { myArr[x] = 42; } Each element in the array is assigned the value of 42. The x variable in the for loop is used as the index for the array.  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]; Visualize this array as a table composed of 3 rows, and 4 columns.https://api.sololearn.com/DownloadFile?id=2422  **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.  Pointer Operations: There are two operators for pointers: Address-of operator (&): returns the memory address of its operand.  Contents-of (or dereference) operator (\*): returns the value of the variable located at the address specified by its operand.  For example:  int var = 50; int \*p; p = &var;  cout << var << endl; // Outputs 50 (the value of var)  cout << p << endl; // Outputs 0x29fee8 (var's memory location)  cout << \*p << endl; /\* Outputs 50 (the value of the variable stored in the pointer p) \*/  **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.  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.  Multiple Parameters  You can define as many parameters as you want for your functions, by separating them with commas.  Let's create a simple function that returns the sum of two parameters.int addNumbers(int x, int y) { // code goes here } As defined, the addNumbers function takes two parameters of type int, and returns int.  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.  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. |