**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | 22-6-2020 | **Name:** | Rasika Patil |
| **Course:** | **C++** | **USN:** | 4AL16EC057 |
| **Topic:** | * Basic Concepts * Conditionals and Loops | **Semester & Section:** | 8th B |
| **Github Repository:** | Rasika B Patil |  |  |

|  |  |  |
| --- | --- | --- |
| **MODULE 1**  **DATA TYPES, ARRAYS, POINTERS**  **About data types, arrays, more on pointers, dynamic memory**  **C:\Users\User\Downloads\WhatsApp Image 2020-06-23 at 8.51.21 PM (2).jpeg**C:\Users\User\Downloads\WhatsApp Image 2020-06-23 at 8.51.21 PM (3).jpeg  C:\Users\User\Downloads\WhatsApp Image 2020-06-23 at 8.51.21 PM (1).jpeg**C:\Users\User\Downloads\WhatsApp Image 2020-06-23 at 8.51.21 PM.jpeg**  **DATA TYPE**  All [variables](https://www.geeksforgeeks.org/variables-and-keywords-in-c/) use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared. Every data type requires a different amount of memory.  This article discusses primitive data types available in C++.   * Integer: Keyword used for integer data types is int. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647. * Character: Character data type is used for storing characters. Keyword used for character data type is char. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255. * Boolean: Boolean data type is used for storing boolean or logical values. A boolean variable can store either *true*or *false*. Keyword used for boolean data type is bool. * Floating Point: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is float. Float variables typically requires 4 byte of memory space. * Double Floating Point: Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is double. Double variables typically requires 8 byte of memory space. * void: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value. * [Wide Character](https://www.geeksforgeeks.org/wide-char-and-library-functions-in-c/): Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype. Represented by wchar\_t. It is generally 2 or 4 bytes long.  1. C++ int  * The int keyword is used to indicate integers. * Its size is usually 4 bytes. Meaning, it can store values from **-2147483648 to 214748647**. * For example,   int salary = 85000; 2. C++ float and double  * float and double are used to store floating-point numbers (decimals and exponentials). * The size of float is 4 bytes and the size of double is 8 bytes. Hence, double has two times the precision of float. To learn more, visit C++ float and double. * For example,   float area = 64.74;  double volume = 134.64534;  As mentioned above, these two data types are also used for exponentials. For example,  double distance = 45E12 // 45E12 is equal to 45\*10^12 3. C++ char  * Keyword char is used for characters. * Its size is 1 byte. * Characters in C++ are enclosed inside single quotes ' '. * For example,   char test = 'h';  **Note:** In C++, an integer value is stored in a char variable rather than the character itself. To learn more, visit [C++ characters](https://www.programiz.com/cpp-programming/char). 4. C++ wchar\_t  * Wide character wchar\_t is similar to the char data type, except its size is 2 bytes instead of 1. * It is used to represent characters that require more memory to represent them than a single char. * For example,   wchar\_t test = L'ם' // storing Hebrew character;  Notice the letter L before the quotation marks.  **Note:** There are also two other fixed-size character types char16\_t and char32\_t introduced in C++11. 5. C++ bool  * The bool data type has one of two possible values: true or false. * Booleans are used in conditional statements and loops (which we will learn in later chapters). * For example,   bool cond = false; 6. C++ void  * The void keyword indicates an absence of data. It means "nothing" or "no value". * We will use void when we learn about functions and pointers.   **Note:** We cannot declare variables of the void type.  **ARRAYS**  C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.  Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.  All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element. Declaring Arrays To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows −  type arrayName [ arraySize ];  This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid C++ data type. For example, to declare a 10-element array called balance of type double, use this statement −  double balance[10]; Initializing Arrays You can initialize C++ array elements either one by one or using a single statement as follows −  double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};  The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array −  If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −  double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};  You will create exactly the same array as you did in the previous example.  balance[4] = 50.0;  The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th, i.e., last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representaion of the same array we discussed above −  Array Presentation Accessing Array Elements An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −  double salary = balance[9];  The above statement will take 10th element from the array and assign the value to salary variable. Following is an example, which will use all the above-mentioned three concepts viz. declaration, assignment and accessing arrays −  #include <iostream>  using namespace std;    #include <iomanip>  using std::setw;    int main () {  int n[ 10 ]; // n is an array of 10 integers    // initialize elements of array n to 0  for ( int i = 0; i < 10; i++ ) {  n[ i ] = i + 100; // set element at location i to i + 100  }  cout << "Element" << setw( 13 ) << "Value" << endl;    // output each array element's value  for ( int j = 0; j < 10; j++ ) {  cout << setw( 7 )<< j << setw( 13 ) << n[ j ] << endl;  }    return 0;  } POINTERS IN C++ Pointer is a variable in C++ that holds the address of another variable. They have [data type](https://beginnersbook.com/2017/08/cpp-data-types/) just like variables, for example an integer type pointer can hold the address of an integer variable and an character type pointer can hold the address of char variable. Syntax of pointer data\_type \*pointer\_name;  **How to declare a pointer?**  /\* This pointer p can hold the address of an integer  \* variable, here p is a pointer and var is just a  \* simple integer variable  \*/  int \*p, var  **Assignment** As I mentioned above, an integer type pointer can hold the address of another int variable. Here we have an integer variable var and pointer p holds the address of var. To assign the address of variable to pointer we use **ampersand symbol** (&).  /\* This is how you assign the address of another variable  \* to the pointer  \*/  p = &var; DYNAMIC MEMORY A good understanding of how dynamic memory really works in C++ is essential to becoming a good C++ programmer. Memory in your C++ program is divided into two parts −   * The stack − All variables declared inside the function will take up memory from the stack. * The heap − This is unused memory of the program and can be used to allocate the memory dynamically when program runs.   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 a special operator in C++ which returns the address of the space allocated. This operator is called new operator.  If you are not in need of dynamically allocated memory anymore, you can use delete operator, which de-allocates memory that was previously allocated by new operator. new and delete Operators There is following generic syntax to use new operator to allocate memory dynamically for any data-type.  new data-type;  Here, data-type could be any built-in data type including an array or any user defined data types include class or structure. Let us start with built-in data types. For example we can define a pointer to type double and then request that the memory be allocated at execution time. We can do this using the new operator with the following statements −  double\* pvalue = NULL; // Pointer initialized with null  pvalue = new double; // Request memory for the variable  The memory may not have been allocated successfully, if the free store had been used up. So it is good practice to check if new operator is returning NULL pointer and take appropriate action as below −  double\* pvalue = NULL;  if( !(pvalue = new double )) {  cout << "Error: out of memory." <<endl;  exit(1);  }  **MODULE 2**  **FUNCTION**  **About Function, Function Parameters, Default Argument, Recursion,**  **C:\Users\User\Downloads\WhatsApp Image 2020-06-23 at 8.51.21 PM (4).jpegC:\Users\User\Downloads\WhatsApp Image 2020-06-23 at 8.51.22 PM.jpeg**  **C:\Users\User\Downloads\WhatsApp Image 2020-06-23 at 8.51.22 PM (1).jpeg**C:\Users\User\Downloads\WhatsApp Image 2020-06-23 at 8.52.50 PM.jpeg FUNCTIONA function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function.  Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times. Create a Function C++ provides some pre-defined functions, such as main(), which is used to execute code. But you can also create your own functions to perform certain actions.  To create (often referred to as declare) a function, specify the name of the function, followed by parentheses **()**: Syntax void myFunction() {   // code to be executed } Example Explained  * myFunction() is the name of the function * void means that the function does not have a return value. You will learn more about return values later in the next chapter * inside the function (the body), add code that defines what the function should do  Call a Function Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called.  To call a function, write the function's name followed by two parentheses () and a semicolon ;  In the following example, myFunction() is used to print a text (the action), when it is called: Example Inside main, call myFunction():  // Create a function void myFunction() {   cout << "I just got executed!"; }  int main() {   **myFunction();** // call the function   return 0; }  // Outputs "I just got executed!" FUNCTION PARAMETERSParameters and Arguments Information can be passed to functions as a parameter. Parameters act as variables inside the function.  Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma: Syntax void functionName(parameter1, parameter2, parameter3) {   // code to be executed }  The following example has a function that takes a string called fname as parameter. When the function is called, we pass along a first name, which is used inside the function to print the full name: Example void myFunction(**string fname**) {   cout << fname << " Refsnes\n"; }  int main() {   myFunction("Liam");   myFunction("Jenny");   myFunction("Anja");   return 0; }  // Liam Refsnes // Jenny Refsnes // Anja Refsnes DEFAULT ARGUMENTS A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn’t provide a value for the argument with a default value.  Following is a simple C++ example to demonstrate the use of default arguments. We don’t have to write 3 sum functions, only one function works by using default values for 3rd and 4th arguments.  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | #include<iostream>  using namespace std;    // A function with default arguments, it can be called with  // 2 arguments or 3 arguments or 4 arguments.  int sum(int x, int y, int z=0, int w=0)  {      return (x + y + z + w);  }    /\* Driver program to test above function\*/  int main()  {      cout << sum(10, 15) << endl;      cout << sum(10, 15, 25) << endl;      cout << sum(10, 15, 25, 30) << endl;      return 0;  } |  RECURSIONIn this tutorial, we will learn about recursive function in C++ and its working with the help of examples. A [function](https://www.programiz.com/cpp-programming/function) that calls itself is known as a recursive function. And, this technique is known as recursion. Working of Recursion in C++ void recurse()  {  ... .. ...  recurse();  ... .. ...  }  int main()  {  ... .. ...  recurse();  ... .. ...  } PASSING REFERENCE TO A POINTER **Prerequisite**: [Pointers vs References in C++](https://www.geeksforgeeks.org/pointers-vs-references-cpp/).  For clear understanding, let’s compare the usage of a “pointer to pointer” VS “Reference to pointer” in some cases.  **Note:** It is allowed to use “pointer to pointer” in both C and C++, but we can use “Reference to pointer” only in C++.  **Passing pointer to a function**  If a pointer is passed to a function as a parameter and tried to be modified then the changes made to the pointer does not reflects back outside that function. This is because only a copy of the pointer is passed to the function. It can be said that “pass by pointer” is [passing a pointer by value](https://www.geeksforgeeks.org/passing-by-pointer-vs-passing-by-reference-in-c/). In most cases, this does not present a problem. But the problem comes when you modify the pointer inside the function. Instead of modifying the variable, you are only modifying a copy of the pointer and the original pointer remains unmodified.  Below program illustrate this:  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | #include <iostream>    using namespace std;    int global\_Var = 42;    // function to change pointer value  void changePointerValue(int\* pp)  {      pp = &global\_Var;  }    int main()  {      int var = 23;      int\* ptr\_to\_var = &var;        cout << "Passing Pointer to function:" << endl;        cout << "Before :" << \*ptr\_to\_var << endl; // display 23        changePointerValue(ptr\_to\_var);        cout << "After :" << \*ptr\_to\_var << endl; // display 23        return 0;  } | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |