**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **26-June-2020** | **Name:** | **Raziya Banu** |
| **Course:** | **C++ Programming** | **USN:** | **4AL16EC058** |
| **Topic:** | **Function Templates** | **Semester & Section:** | **8th sem & ‘B’ section** |
| **Github Repository:** |  |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**  In my first session today I have studied about – **Function Templates** **Templates in C++** A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types. For example, a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.  C++ adds two new keywords to support templates: ‘template’ and ‘typename’. The second keyword can always be replaced by keyword ‘class’.  **How templates work?**  Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class. [templates-cpp](https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2014/06/templates-cpp.jpg)    **Function Templates**   We write a generic function that can be used for different data types. Examples of function templates are sort(), max(), min(), printArray(). Know more on [Generics in C++](https://www.geeksforgeeks.org/generics-in-c/)  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | #include <iostream>  using namespace std;  // One function works for all data types.  This would work  // even for user defined types if operator '>' is overloaded  template <typename T>  T myMax(T x, T y)  {     return (x > y)? x: y;  }  int main()  {    cout << myMax<int>(3, 7) << endl;  // Call myMax for int    cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double    cout << myMax<char>('g', 'e') << endl;   // call myMax for char      return 0;  } |   Output:  7  7  g |

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **26-June-2020** | **Name:** | **Raziya Banu** |
| **Course:** | **C++ Programming** | **USN:** | **4AL16EC058** |
| **Topic:** | **Working Files** | **Semester & Section:** | **8th sem & ‘B’ section** |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| To perform file processing in C++, header files <iostream> and <fstream> must be included in your C++ source file. Opening a File A file must be opened before you can read from it or write to it. Either **ofstream** or **fstream** object may be used to open a file for writing. And ifstream object is used to open a file for reading purpose only.  Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.  void open(const char \*filename, ios::openmode mode);  Here, the first argument specifies the name and location of the file to be opened and the second argument of the **open()** member function defines the mode in which the file should be opened.   |  |  | | --- | --- | | **Sr.No** | **Mode Flag & Description** | | 1 | **ios::app**  Append mode. All output to that file to be appended to the end. | | 2 | **ios::ate**  Open a file for output and move the read/write control to the end of the file. | | 3 | **ios::in**  Open a file for reading. | | 4 | **ios::out**  Open a file for writing. | | 5 | **ios::trunc**  If the file already exists, its contents will be truncated before opening the file. |   You can combine two or more of these values by **OR**ing them together. For example if you want to open a file in write mode and want to truncate it in case that already exists, following will be the **syntax −**  ofstream outfile;  outfile.open("file.dat", ios::out | ios::trunc );  Similar way, you can open a file for reading and writing purpose as follows −  fstream afile;  afile.open("file.dat", ios::out | ios::in ); Closing a File When a C++ program terminates it automatically flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.  Following is the standard syntax for close() function, which is a member of fstream, ifstream, and ofstream objects.  void close(); Writing to a File While doing C++ programming, you write information to a file from your program using the stream insertion operator (<<) just as you use that operator to output information to the screen. The only difference is that you use an **ofstream** or **fstream** object instead of the **cout** object. Reading from a File You read information from a file into your program using the stream extraction operator (>>) just as you use that operator to input information from the keyboard. The only difference is that you use an **ifstream** or **fstream** object instead of the **cin** object. Read and Write Example Following is the C++ program which opens a file in reading and writing mode. After writing information entered by the user to a file named afile.dat, the program reads information from the file and outputs it onto the screen −  #include <fstream>  #include <iostream>  using namespace std;  int main () {  char data[100];  // open a file in write mode.  ofstream outfile;  outfile.open("afile.dat");  cout << "Writing to the file" << endl;  cout << "Enter your name: ";  cin.getline(data, 100);  // write inputted data into the file.  outfile << data << endl;  cout << "Enter your age: ";  cin >> data;  cin.ignore();    // again write inputted data into the file.  outfile << data << endl;  // close the opened file.  outfile.close();  // open a file in read mode.  ifstream infile;  infile.open("afile.dat");    cout << "Reading from the file" << endl;  infile >> data;  // write the data at the screen.  cout << data << endl;    // again read the data from the file and display it.  infile >> data;  cout << data << endl;  // close the opened file.  infile.close();  return 0;  } | | | |