**Project Requirement and Specification**

**on**

**Student Report Card Management in C++**

**(CSE III Semester Mini project PCS-304)**

**2020-2021**

****

**Submitted to: Submitted by:**

Mrs. Astha Gour Mr. Yash Tariyal

(CC-CSE-B-III-Sem) Roll. No.:1918821

CSE-G-III-Sem

Session: 2020-2021

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GRAPHIC ERA HILL UNVERSITY, DEHRADUN.**

* 1. **About Project**

Student report card system project in C++ is a simple console application built without the use of graphics. In this project, we can perform typical report card related functions like adding a new student record and displaying, modifying, editing and deleting it. File handling has been effectively used to perform all these. This project will tell us how to use file handling in C++, add, read, display, search, modify and delete record from file.

* 1. **Requirement of Project**
     1. **Hardware Requirement**

As a minimum, you should have at least 500 Mb for data and 32 Mb RAM. However, this will allow you only a very limited usage of GRASS (although see for GRASS on handhelds). For a serious usage of GRASS, you will need 128MB at the very least. Obviously, a good video card is a must for cartography, ideally with 3D support.

* + 1. **Software Requirement**

C-compiler (cc, gcc, egcs, ...)

gcc: <http://www.gnu.org/software/gcc/gcc.html>

Note pad/Note-pad++

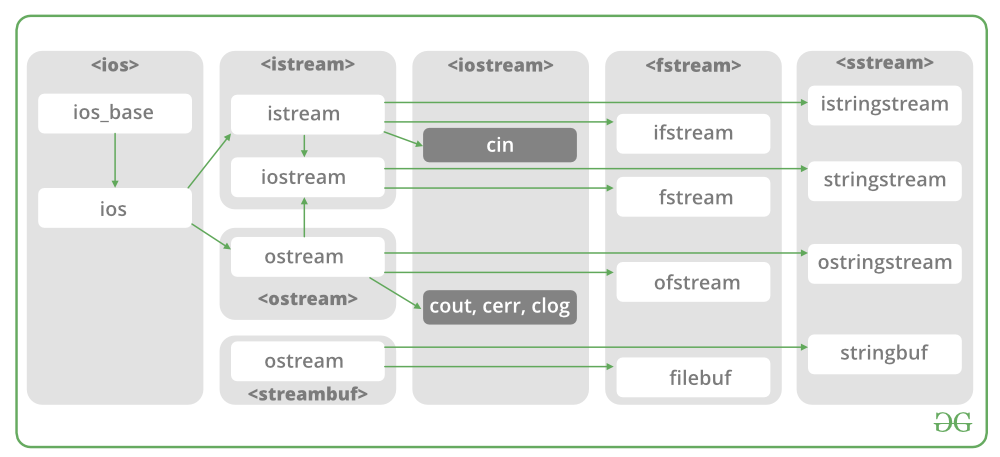
Notepad++: <https://notepad-plus-plus.org/downloads>

**TABLE OF CONTENTS**

**SNO. TITLE**

1. **INTRODUCTION**
   1. File handling
   2. Modes
   3. Default open modes
   4. File Position Pointer
2. **PROJECT**
   1. About Report Card Management
   2. Header file used
   3. Class in file handling
   4. CSV file management
   5. Create and Write to a file
   6. Read a file
   7. Scope Resolution Operator in C++
   8. Type Conversion In C++
3. **CONCLUSION**
4. **REFERENCES**

**Introduction**

In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream available in fstream header file.  
**ofstream:** Stream class to write on files  
**ifstream:** Stream class to read from files  
**fstream:** Stream class to both read and write from/to files.  
[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20191129162746/CPP-File-Handling.png)  
Now the first step to open the particular file for read or write operation. We can open file by  
1. passing file name in constructor at the time of object creation  
2. using the open method**.**

**Modes:**

|  |  |  |
| --- | --- | --- |
| **MEMBER CONSTANT** | **STANDS FOR** | **ACCESS** |
| in \* | input | File open for reading: the internal stream buffer supports input operations. |
|  |  |  |
| out | output | File open for writing: the internal stream buffer supports output operations. |
|  |  |  |
| binary | binary | Operations are performed in binary mode rather than text. |
|  |  |  |
| ate | at end | The output position starts at the end of the file. |
|  |  |  |
| app | append | All output operations happen at the end of the file, appending to its existing contents. |
|  |  |  |
| trunc | truncate | Any contents that existed in the file before it is open are discarded. |
|  |  |  |

**Default Open Modes:**

|  |  |
| --- | --- |
| ifstream | ios:in |
| ofstream | ios:out |
| fstream | ios:in | ios:out |

## File Position Pointers

Both **istream** and **ostream** provide member functions for repositioning the file-position pointer. These member functions are **seekg** ("seek get") for istream and **seekp** ("seek put") for ostream.

The argument to seekg and seekp normally is a long integer. A second argument can be specified to indicate the seek direction. The seek direction can be **ios::beg** (the default) for positioning relative to the beginning of a stream, **ios::cur** for positioning relative to the current position in a stream or **ios::end** for positioning relative to the end of a stream.

The file-position pointer is an integer value that specifies the location in the file as a number of bytes from the file's starting location.

**About Student Report Card System:**

[](https://www.codewithc.com/wp-content/uploads/2014/06/student-report-card-cpp.png)

File handling has been used for the effective implementation of all the typical features of this project. The key features of Student Report Card System are:

1. **Create student report card record**: This feature creates a new student record containing his marks. For this the information to be provided are the name and roll no. of the student, and the marks obtained by him/her in 5 subjects – Physics, Chemistry, Maths, English and Computer Science.

2. **Read all students report card record**: function in this student report card system project in C++ has been used for this feature. It basically shows the progress report of all the students added in file. This feature displays the roll no. and name of all the students, the marks obtained by them in 5 subjects – Physics, Chemistry, Maths, English and Computer Science, along with the percentage and grade of each student.

3. **Read specific student’s report card record**: This feature is same as the one explained above, except it shows the progress report and relevant data related to a particular student.

4. **Display all students’ grade report**: This feature enlists all the students’ record saved in file. The grade report is displayed in a tabular form with roll no. and name of the students, marks achieved in the five subjects, and the grade and percentage obtained by them.

5. **Modify student’s report card record**: In student report card system project in C++, this feature is used to edit the report card record of a particular student. For this, the name and roll no. of the student is sought. Upon successful modification, the program displays the message “Record Updated”. If no record of student is found in file, it displays the message “Record not found”.

6. **Delete student record**: This feature deletes the report card record of a particular student; it first of all asks for the name and roll no. of the student whose record is to be deleted.

**Header Files Used:**

Student report card system is a very simple project that runs with just five header files. The conio.h header file is required to compile the source code in Turbo C++; it is not required in Code: Blocks. Also, in order to make the project simple and easy to understand and analyze, graphics has not been used in this project. Here, are the header files required for this project:

* #include<iostream>
* #include<fstream>
* #include<iomanip>

# **Read/Write Class Objects from/to File in C++:**

Given a file “Input.txt” in which every line has values same as instance variables of a class.   
Read the values into the class’s object and do necessary operations.

**Theory :**

The data transfer is usually done using '>>'

and <<' operators. But if you have

a class with 4 data members and want

to write all 4 data members from its

object directly to a file or vice-versa,

we can do that using following syntax :

**To write object's data members in a file :**

// Here file\_obj is an object of ofstream

file\_obj.write((char \*) & class\_obj, sizeof(class\_obj));

**To read file's data members into an object :**

// Here file\_obj is an object of ifstream

file\_obj.read((char \*) & class\_obj, sizeof(class\_obj));

**Examples:**

**Input :**

Input.txt :

Michael 19 1806

Kemp 24 2114

Terry 21 2400

Operation : Print the name of the highest

rated programmer.

**Output :**

Terry

# **CSV file management using C++:**

**CSV** is a simple file format used to store tabular data such as a spreadsheet or a database. CSV stands for **Comma Separated Values**. The data fields in a CSV file are separated/delimited by a comma **(‘, ‘)** and the individual rows are separated by a newline **(‘\n’)**. CSV File management in C++ is similar to text-type file management, except for a few modifications.

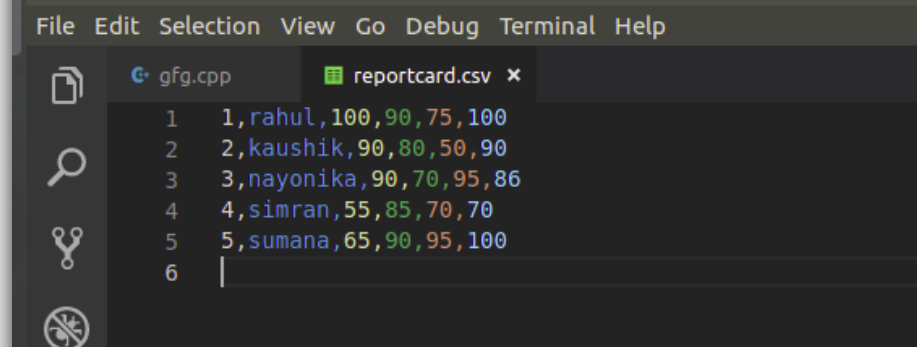
This article discusses about how to **create, update and delete records** in a CSV file:

**Note:** Here, a reportcard.csv file has been created to store the student’s roll number, name and marks in math, physics, chemistry and biology.

1. **Create operation:**

The create operation is similar to creating a text file, i.e. input data from the user and write it to the csv file using the file pointer and appropriate delimiters(‘, ‘) between different columns and ‘\n’ after the end of each row.

|  |
| --- |
| void create()  {  // file pointer  fstream fout;  // opens an existing csv file or creates a new file.  fout.open("reportcard.csv", ios::out | ios::app);  cout << "Enter the details of 5 students:"  << " roll name maths phy chem bio";  << endl;  int i, roll, phy, chem, math, bio;  string name;  // Read the input  for (i = 0; i < 5; i++) {  cin >> roll  >> name  >> math  >> phy  >> chem  >> bio;  // Insert the data to file  fout << roll << ", "  << name << ", "  << math << ", "  << phy << ", "  << chem << ", "  << bio  << "\n";  }  } |

**Output:**  


## Create and Write To a File:

To create a file, use either the ofstream or fstream class, and specify the name of the file.

To write to the file, use the insertion operator (<<).

### **Example**

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main() {  
  // Create and open a text file  
  ofstream MyFile("filename.txt");  
  
  // Write to the file  
  MyFile << "Files can be tricky, but it is fun enough!";  
  
  // Close the file  
  MyFile.close();  
}

## Read a File:

To read from a file, use either the ifstream or fstream class, and the name of the file.

Note that we also use a while loop together with the getline() function (which belongs to the ifstream class) to read the file line by line, and to print the content of the file:

### **Example**

// Create a text string, which is used to output the text file  
string myText;  
  
// Read from the text file  
ifstream MyReadFile("filename.txt");  
  
// Use a while loop together with the getline() function to read the file line by line  
while (getline (MyReadFile, myText)) {  
  // Output the text from the file  
  cout << myText;  
}  
  
// Close the file  
MyReadFile.close();

# **Scope resolution operator in C++:**

In C++, scope resolution operator is **::**. It is used for following purposes.

**1) To access a global variable when there is a local variable with same name:**

|  |
| --- |
| // C++ program to show that we can access a global variable  // using scope resolution operator :: when there is a local  // variable with same name  #include<iostream>  using namespace std;    int x;  // Global x    int main()  {    int x = 10; // Local x    cout << "Value of global x is " << ::x;    cout << "\nValue of local x is " << x;    return 0;  } |

Output:

Value of global x is 0

Value of local x is 10

**2) To define a function outside a class.**

|  |
| --- |
| // C++ program to show that scope resolution operator :: is used  // to define a function outside a class  #include<iostream>  using namespace std;  class A  {  public:  // Only declaration  void fun();  };  // Definition outside class using ::  void A::fun()  {  cout << "fun() called";  }  int main()  {     A a;     a.fun();     return 0;  } |

**Output:**

fun() called

**3) To access a class’s static variables.**

|  |
| --- |
| // C++ program to show that :: can be used to access static  // members when there is a local variable with same name  #include<iostream>  using namespace std;    class Test  {      static int x;  public:      static int y;        // Local parameter 'a' hides class member      // 'a', but we can access it using ::      void func(int x)      {         // We can access class's static variable         // even if there is a local variable    cout << "Value of static x is " << Test::x;    cout << "\nValue of local x is " << x;      }  };    // In C++, static members must be explicitly defined  // like this  int Test::x = 1;  int Test::y = 2;   int main()  {     Test obj;     int x = 3 ;     obj.func(x);     cout << "\nTest::y = " << Test::y;     return 0;  } |

Output:

Value of static x is 1

Value of local x is 3

Test::y = 2;

**4) In case of multiple Inheritance:**  
If same variable name exists in two ancestor classes, we can use scope resolution operator to distinguish.

|  |
| --- |
| // Use of scope resolution operator in multiple inheritance.  #include<iostream>  using namespace std;    class A  {  protected:      int x;  public:      A() { x = 10; }  };    class B  {  protected:      int x;  public:      B() { x = 20; }  };    class C: public A, public B  {  public:     void fun()     {        cout << "A's x is " << A::x;        cout << "\nB's x is " << B::x;     }  };    int main()  {      C c;      c.fun();      return 0;  } |

Output:

A's x is 10

B's x is 20

**5) For namespace**  
If a class having the same name exists inside two namespace we can use the namespace name with the scope resolution operator to refer that class without any conflicts

|  |
| --- |
| // Use of scope resolution operator for namespace.  #include<iostream>   int main(){   std::cout << "Hello" << std::endl;   } |

Here, cout and endl belong to the std namespace.

**6) Refer to a class inside another class:**  
If a class exists inside another class we can use the nesting class to refer the nested class using the scope resolution operator

|  |  |  |  |
| --- | --- | --- | --- |
| // Use of scope resolution class inside another class.  #include<iostream>  using namespace std;  class outside  {  public:        int x;        class inside        {        public:              int x;              static int y;              int foo();   };  };  int outside::inside::y = 5;  int main(){      outside A;      outside::inside B;    } **Type Conversion In C++:** A type cast is basically a conversion from one type to another. There are two types of type conversion:  **1.Implicit Type Conversion** Also known as ‘automatic type conversion’.  Done by the compiler on its own, without any external trigger from the user. Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.  All the data types of the variables are upgraded to the data type of the variable with largest data type.  bool -> char -> short int -> int ->  unsigned int -> long -> unsigned  long long -> float -> double -> long double  It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).  **Example of Type Implicit Conversion:**   |  | | --- | | // An example of implicit conversion  #include <iostream>  using namespace std;  int main()  {  int x = 10; // integer x  char y = 'a'; // character c  // y implicitly converted to int. ASCII  // value of 'a' is 97  x = x + y;  // x is implicitly converted to float  float z = x + 1.0;  cout << "x = " << x << endl  << "y = " << y << endl  << "z = " << z << endl;  return 0;  } |   **Output:**  x = 107  y = a  z = 108  **2.Explicit Type Conversion**: This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.  In C++, it can be done by two ways:  **Converting by assignment:** This is done by explicitly defining the required type  in front of the expression in parenthesis. This can be also considered as forceful casting.  **Syntax:**  (type) expression  where *type* indicates the data type to which the final result is converted.  **Example:**   |  | | --- | | // C++ program to demonstrate  // explicit type casting  #include <iostream>  using namespace std;  int main()  {  double x = 1.2;  // Explicit conversion from double to int  int sum = (int)x + 1;  cout << "Sum = " << sum;  return 0;  } |   **Output:**  Sum = 2  **Conversion using Cast operator:** A Cast operator is an **unary operator** which forces one data type to be  converted into another data type.  C++ supports four types of casting:  1.[Static Cast](https://www.geeksforgeeks.org/static_cast-in-c-type-casting-operators/)  2.Dynamic Cast  3.[Const Cast](https://www.geeksforgeeks.org/casting-operators-in-c-set-1-const_cast/)  4.[Reinterpret Cast](https://www.geeksforgeeks.org/reinterpret_cast-in-cpp/)  **Example:**   |  | | --- | | #include <iostream>  using namespace std;  int main()  {  float f = 3.5;  // using cast operator  int b = static\_cast<int>(f);  cout << b;  } |   **Output:**  3  **Advantages of Type Conversion:**  This is done to take advantage of certain features of type hierarchies or type representations.  It helps to compute expressions containing variables of different data types. |

**CONCLUSION**

Throughout we made a student report card management system project in c++ from end to end and we learned and obtained several insights about file handling in c++ and also about different types of files or syntax used in file handling. In this project we are collecting data about the student like his/her roll no, name, and then marks which he/she obtained in different subjects and calculating percentage of the student and giving the grades according to their percentage and at last we are saving the data in the text file and also user can modify change it any time. So that is all about the project.

**REFERENCES**

1. <https://www.tutorialspoint.com/cplusplus/cpp_file>
2. <https://www.geeksforgeeks.org/c-plus-plus/?ref=lbp>