Project report on

**SALES ANNOUNCES**

In Computer Science

Session 2012-13

Name of student

**Kunal Das**

Under the guidance of

**S. Chandrashekhar**

For the partial fulfillment of

AISSCE (2012), CBSE, New Delhi

Submitted to

Jawaharlal Nehru School

Habibganj BHEL, Bhopal

**Certificate**

This is to certify that Kunal Das of class 12th ‘B’ has successfully completed his project report in C++ on ‘Sales Announces’ under my guidance for the AISSE as prescribed by CBSE in the year 2012-13.

Date:

|  |  |
| --- | --- |
| Signature of:  Internal examiner | External examiner |
|  |  |
|  | - S. Chandrashekhar |

**Acknowledgement**

I would like to thank my C++ teachers for all that they have taught me and their guidance for making this program. I would also like to thank my sisters for all the extra help.

**Preface**

This file provides interaction about general **OOPs (Object Oriented Programming)** concept and **Database concept** to help understand the structure and mechanism of the program **Sales Announces**.

Each **function** of the program has been provided with **algorithm** and with **screenshots** so that readers can understand better, what is happening during the course of program usage.

The program does not hold well in all place of situation; thus **limitations** of the program have been listed at the end of the file.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| # | Title | Page no. |
| 1. | Certificate |  |
| 2. | Acknowledgement |  |
| 3. | Preface |  |
| 4. | Object Oriented Programming (OOP) |  |
| 5. | Database concept |  |
| 6. | Documentation |  |
| 7. | Introduction |  |
| 8. | Algorithm |  |
| 9. | Coding |  |
| 10. | Screenshots |  |
| 11. | Limitations |  |
| 12. | Bibliography |  |

**Object Oriented Programming**

With the defeat of **procedural** oriented programming due to complexities and errors, object oriented programming was introduced with set of concepts to greatly decrease the complexities and cope with errors.

Object Oriented Programming lays emphasis on **objects** and provides a full set of operations for each **class**. Indeed it serves two most important purposes of a programming language:

|  |  |
| --- | --- |
| 1. | It provides a vehicle for the programmer to specify actions to be executed. |
| 2. | It provides a set of concepts for the programmer to use when thinking about what can be done. |

|  |  |  |
| --- | --- | --- |
| **procedural** | : | The programming approach that focuses on the procedures for the solution of a problem is known as procedural programming **paradigm**. This approach emphasizes on ‘doing’ rather than the ‘data’. |
|  |  |  |
| **object** | : | Object is an identifiable entity with some characteristics and behavior. |
|  |  |  |
| **class** | : | A class is a template, representing a group of objects that share common properties and relationships. |
|  |  |  |
| **paradigm** | : | Paradigm means organizing principle of a program. It is an approach to programming. |
|  |  |  |
| **Modular programming** | : | Breaking down of a large program into smaller units called **functions** (sub-programs). The idea of breaking a program into functions can further be extended by grouping a number of functions together into a larger entity called a **module**. |
|  |  |  |
| **module** | : | A set of related procedure with data they manipulate is called a module, |

Basic concepts of Object Oriented Programming (OOP):

|  |  |  |
| --- | --- | --- |
| **Data Abstraction** | : | Abstraction refers to the act of representing essential features without including the background details or explanations. |
|  |  |  |
| **Data encapsulation** | : | The wrapping up of data and operations (**functions**: that operates on the data) into a single unit (**class**) is known as data encapsulation. |
|  |  |  |
| **Modularity** | : | It is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules. |
|  |  |  |
| **Inheritance** | : | It is the capability of one class of things to inherit capabilities or properties from another class. |
|  |  |  |
| **Polymorphism** | : | It is the ability of a data to be processed in more than one form. |

**Database concept**

The collection of data, arranged in a systematic order in the system through some software, is known as a database.

Whenever a user wants to share some data with more than one place by the multiuser, we have to use database.

Storage of data in a \*.dat file is only a method to store data for a long time. When dealing with huge load of data kept stored in a ‘data file’. The most difficult task is ‘how to display the data’ so as to meet the user demands. It is done by general form of representation called table.

A table consists of rows and columns. In database concept rows are called **tuples**, and columns are called **attributes**.

Consider a list of products in the form of a bill:

**Product**

+------+------+------+----------+--------+

| Code | Name | Rate | Quantity | Amount |

+------+------+------+----------+--------+

| 31 | A | 50 | 5 | 250 |

| 32 | B | 75 | 2 | 150 |

| 33 | C | 20 | 8 | 160 |

| 34 | D | 34 | 4 | 136 |

| 35 | E | 15 | 7 | 105 |

+------+------+------+----------+--------+

Here Product code, Product name, Product rate, Quantity & Total amount, are called **attributes**.

The row representing details of one product is called **tuple**.

Collection of attributes is called **degree of table**.

Collection of tuples is called **cardinality of table**.

In the program available in this file, ‘Sales Announces’ the Data model of the Database is taken to be **Relational data model**.

|  |  |
| --- | --- |
| **Relational data** **model:** | In the given table, the relationship between the attributes is known as relational data model. |
|  |  |
| Example: | When we enter the product code, the screen displays the data connected with that product code. |
|  |  |

Documentation

|  |  |  |
| --- | --- | --- |
| **Name of the project** | : | Sales Announces |
|  |  |  |
| **Purpose of the project** | : | Store records of details of products available in a store and prepare a bill for the bought products, hence calculate percentile of sales made. |
|  |  |  |
| **Name of the source file** | : | sales announces |
|  |  |  |
| **Name of data files** | : | PRODUCT.dat  BILL.dat |
|  |  |  |
| **Size of the source file** | : | 43.8 KB |
|  |  |  |
| **Operating system used** | : | Windows 8 Release Preview |
|  |  |  |
| **C++ version used** | : | Turbo C++ 3.0 |

List of **member functions** and their uses:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Name | parameter | return type | Usage |
| **1.** | last\_code | void | integer | Returns the last greatest product code present in PRODUCT.dat |
| **2.** | product\_code | integer | integer | Returns the given product code if the given product code is found in PRODUCT.dat |
| **3.** | search\_code | integer | integer | Returns truth value if the given product code is found in PRODUCT.dat |
| **4.** | display\_product | integer | void | Searches the given product code in PRODUCT.dat & if found, then puts it onto display |
| **5.** | display\_bill | integer | void | Searches the given product code in BILL.dat & if found, then puts it onto display |
| **6.** | find\_bill | integer | integer | Returns truth value if the given product code is found in BILL.dat |
| **7.** | modify\_entry | integer | void | Accesses to modify the details related to the given product code in BILL.dat |
| **8.** | delete\_entry | integer | void | Accesses to delete the details related to the given product code in BILL.dat |
| **9.** | **write\_product** | void | void | Accesses to write or append a record into PRODUCT.dat |
| **10.** | **read\_product** | void | void | Accesses to read PRODUCT.dat |
| **11.** | **insert\_product** | void | void | Accesses to insert a record in between two records in PRODUCT.dat |
| **12.** | **delete\_product** | void | void | Accesses to delete a record from PRODUCT.dat |
| **13.** | **search\_product** | void | void | Accesses to search a record in PRODUCT.dat |
| **14.** | **modify\_product** | void | void | Accesses to modify a record in PRODUCT.dat |
| **15.** | sort\_product | void | void | Accesses to sort the entire PRODUCT.dat |
| **16.** | **write\_bill** | void | void | Accesses to purchase products, i.e. to decrease quantity from PRODUCT.dat and put it in BILL.dat |
| **17.** | **read\_bill** | void | void | Accesses to read BILL.dat |
| **18.** | **delete\_bill** | void | void | Accesses to delete a record from BILL.dat |
| 19. | **search\_bill** | void | void | Accesses to search a record in BILL.dat |
| 20. | **modify\_bill** | void | void | Accesses to modify a record in PRODUCT.dat |
| 21. | **sort\_bill** | void | void | Accesses to sort the entire BILL.dat |
| 22. | sales | void | void | Calculates and displays percentage of sales made if PRODUCT.dat & BILL.dat are found |
| 23. | remove\_list | void | void | Deletes PRODUCT.dat & BILL.dat |
| 24. | remove\_bill | void | void | Deletes BILL.dat |

Introduction

Library functions used in the program:

|  |  |  |
| --- | --- | --- |
| **conio.h** | : | Stands for ‘console input & output’. Most of its functions process without buffer and without echo. |
|  |  |  |
| **ctype.h** | : | Its functions process on a single character at a time and check according to the given condition |
|  |  |  |
| **fstream.h** | : | Declares standard file input and output stream. |
|  |  |  |
| **process.h** | : | It contains function declarations and macros used in working with threads and processes. |
|  |  |  |
| **stdio.h** | : | It defines types and macros needed for the standard input-output package. |
|  |  |  |
| **stdlib,h** | : | It declares several commonly used routines, like conversion routines, search routines, sort routines, etc. |
|  |  |  |
| **string.h** | : | It declares several string manipulation and memory manipulation routines. |

The program operates on two \*.dat files:

|  |  |  |
| --- | --- | --- |
| **PRODUCT.dat** | : | Used to store and handle the details of the products kept in the store. |
|  |  |  |
| **BILL.dat** | : | Used to store the details of the products bought by a customer. |

Name of the class: **Product**.

|  |  |  |
| --- | --- | --- |
| Number of private data members | : | 6 |
| Number of public data members | : | 0 |
| Number of private member functions | : | 8 |
| Number of public member functions | : | 16 |

List of **data members** and their uses:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Variable | Data type | Usage |
| 1. | code | integer | Used to store product code for the identification of individual product. |
| 2. | name | character | Used to store the name of individual product. |
| 3. | rate | float | Used to store selling price and cost price of individual product as per product list or bill. |
| 4. | qty1 | integer | Used to store the quantity of a single product available in the store. |
| 5. | qty2 | integer | Used to store the quantity of a single product, bought from the store. |
| 6. | amount | long | Used to store the total value of a single product, if bought in multitude. |

Algorithm

|  |  |  |
| --- | --- | --- |
| 1. | Algorithm for **writing** data in the data file : | |
|  | 1. | Define input-output stream to open data file. |
|  | 2. | Open data file while including ios::out.  stream.open(“\*.dat”,ios::out); |
|  | 3. | Input values to data members with respect to an object. |
|  | 4. | Use write() function with the stream, object and sizeof(class).  stream.write((char\*)&object,sizeof(class)); |
|  | 5. | Close the stream.  stream.close(); |

|  |  |  |
| --- | --- | --- |
| 2. | Algorithm for **reading** data from the data file : | |
|  | 1. | Define input-output stream to open data file. |
|  | 2. | Open data file while including ios::in.  stream.open(“\*.dat”,ios::in); |
|  | 3. | Set pointer to the beginning of the file.  stream.seekg(0,ios::beg); |
|  | 4. | Until end of file eof(), repeat step 3. |
|  | 5. | Use read() function with the stream, object and sizeof(class).  stream.read((char\*)&object,sizeof(class)); |
|  | 6. | Close the stream.  stream.close(); |

|  |  |  |
| --- | --- | --- |
| 3. | Algorithm to **append** data in the data file : | |
|  | 1. | Define file stream to open data file. |
|  | 2. | Open data file while including ios::out & ios::app.  stream.open(“\*.dat”,ios::out|ios::app); |
|  | 3. | Input values to data members with respect to an object. |
|  | 4. | Use write() function with the stream, object and sizeof(class).  stream.write((char\*)&object,sizeof(class)); |
|  | 5. | Close the stream.  stream.close(); |

|  |  |  |
| --- | --- | --- |
| 4. | Algorithm to **search** data in the data file : | |
|  | 1. | Define file stream to open data file. |
|  | 2. | Open data file while including ios::in.  stream.open(“\*.dat”,ios::in); |
|  | 3. | Set pointer to the beginning of the file.  stream.seekg(0,ios::beg); |
|  | 4. | Input the serial number of the record to be searched.  cin>>s; |
|  | 5. | Until end of file eof(), repeat step 6. |
|  | 6. | while(!stream.eof())  stream.read((char\*)&object,sizeof(class)); |
|  | 7. | if(s==(serial code))  object.class::display(); |
|  | 8. | Close the stream.  stream.close(); |

|  |  |  |  |
| --- | --- | --- | --- |
| 5. | Algorithm to **insert** data in the data file : | | |
|  | 1. | Define file stream to open data file. | |
|  | 2. | Open data file using ios::out & ios::app.  stream.open(“\*.dat”,ios::out|ios::app); | |
|  | 3. | Input the details of the record to be inserted. | |
|  | 4. | Use write() function with the stream, object and sizeof(class).  stream.write((char\*)&object,sizeof(class)); | |
|  | 5. | Close the stream.  stream.close(); | |
|  | 6. | Sort the data file using Selection sort. | |
|  |  | 1. | for(int i=1 to i=n-1) |
|  |  | 2. | small=list\_of\_i;  position=I; |
|  |  | 3. | for(int j=1 to j=n)  repeat step 4. |
|  |  | 4. | if(list\_of\_j > small)  then small=list\_of\_j; |
|  |  | 5. | temp=list\_of\_i |
|  |  | 6. | list\_of\_i=list\_of\_j; |
|  |  | 7. | list\_position=temp; |

|  |  |  |
| --- | --- | --- |
| 6. | Algorithm to **delete** data from the data file : | |
|  | 1. | Define file stream to open data file. |
|  | 2. | Open data file using ios::in.  stream.open(“\*.dat”,ios::in); |
|  | 3. | Input the serial number of the record to be searched.  cin>>s; |
|  | 4. | Until end of file eof(), repeat step 5. |
|  | 5. | while(!stream.eof())  stream.read((char\*)&object,sizeof(class)); |
|  | 6. | Define another file stream temp to open TEMPORARY data file. |
|  | 7. | Open TEMPORARY data file using ios::out.  stream.open(“\*.dat”,ios::out); |
|  | 8. | Until serial code not equal to s, repeat step 9. |
|  | 9. | if(serial!=s)  temp.write((char\*)&object,sizeof(class)); |
|  | 10. | Close the file streams.  stream.close();  temp.close(); |
|  | 11. | Remove original data file.  Rename temporary data file to original data file.  remove(“\*.dat”);  rename(“TEMPORARY.dat”,”\*.dat”); |

|  |  |  |
| --- | --- | --- |
| 7. | Algorithm to **modify** data in data file : | |
|  | 1. | Define file stream to open data file. |
|  | 2. | Open data file using ios::in.  stream.open(“\*.dat”,ios::in); |
|  | 3. | Input the serial number of the record to be searched.  cin>>s; |
|  | 4. | Until end of file eof(), repeat step 5. |
|  | 5. | while(!stream.eof())  stream.read((char\*)&object,sizeof(class)); |
|  | 6. | if(serial==s)  Input new details. |
|  | 7. | Close stream(); |
|  | 8. | Open data file using ios::out & ios::ate.  stream.open(“\*.dat”,ios::out|ios::ate); |
|  | 9. | Move pointer to the specific serial  int pointer=(s-1)\*sieof(class); |
|  | 10. | seekp(pointer); |
|  | 11. | stream.write((char\*)&object,sizeof(class)); |
|  | 12. | Close the stream.  stream.close(); |

Screenshots

|  |
| --- |
| - Main menu and its options |
| - Product menu and its options |
| - Bill menu and its options |
| - Sales projection option |
| - Sample product list |
| - The purchase function |
| - **Write** and **Append** function for the product list |
| - **Insert** function for the product list |
| - **Search** function for the product list |
| - **Modify** function for the product list |
| - **Delete** function for the product list |
| - Option to delete product list |

Limitations

Bibliography

|  |  |  |
| --- | --- | --- |
| **System details** :- | | |
|  |  |  |
| Operating system used | : | Microsoft Windows 8 |
| Turbo C++ version | : | 3.0 |
| File prepared on | : | MS Word 2007 |
| Printer | : | HP LaserJet Professional M1136 MFP |
|  |  |  |
| **Resources** | : | Sumita Arora, class XI, 6th edition; Sumita Arora, class XII, 9th edition |