

EVENT MANAGEMENT SYSTEM

CHAPTER 1

INTRODUCTION

1.1 Vision

Event management is the application of project management to the creation and development of large-scale events such as festivals, conferences, ceremonies, weddings, formal parties, concerts, or conventions. Event management involves studying the intricacies of the brand, identifying the target audience, devising the event concept, planning the logistics and coordinating the technical aspects before actually launching the event. The purpose of the management system is to help the agency keep track of all the clients, sponsors, employees their details and manage the events. This software covers all the functionality related to events, their vendors, advertisers etc. This system will take the operations of event management to an upper level by providing faster access to data and allowing addition, modification and deletion of data in a very systematic and reliable manner.

1.2 Scope

The scope of the project contains following features:

- It is an event management system aimed at computerizing and automating the work of event managers by maintaining a record of all the details in order.
- It provides a platform for comprehensive data maintenance and information access.
- It is simple and user friendly and easy to use .
- It is an event management system based on files, hence allows access to large amount of information.
- Allows search operations to cater to the requirement of prospective clients.

1.3 Aim and Objectives

The project has data divided into different files corresponding to the entry of type of records. The use of offset hashing is used in this project to help easy access/retrieval of data records in the system.

The aim and objectives of the project are:

- Storage of address on a secondary storage device.
- Addition, deletion and modification of address.
- Quick search of the required address using the concept of offset indexing
- Varied entity entries to help the event management user.
- Commit and Rollback options to help user to manage records efficiently.

CHAPTER 2

LITERATURE SURVEY

2.1 File Structures

File Structures is the Organization of Data in Secondary Storage Device in such a way that minimizes the access time and the storage space. A File Structure is a combination of representations for data in files and of operations for accessing the data. A File Structure allows applications to read, write and modify data. A file contains data entries in the form of records.

A record is a collection of pieces of information pertaining to a single object. The object may be physical or conceptual. For example a student record may comprise items of information such as student's name, roll number, registration number, age, grade in semester 1, grade in semester 2, etc.

The file system developed helps in storing, retrieving and manipulating all the data related to the event management system on secondary storage using concept of file structures.

2.2 Opening the file:

A file can be opened in C++ by creating an object of the file stream class in read mode, append mode, write mode, truncate mode. The syntax for opening the file is,

```
obj.open (filename, mode);
```

The different modes include,

- `ios::in`, for opening the file in read mode.
- `ios::out`, for opening the file in write mode. If the file is not present it is created and if the file exists, its contents are erased before use.
- `ios::app`, for opening the file in append mode. The new contents are appended to the file at the end of the file.
- `ios::trunc`, for opening the file in truncate mode. The contents of the file are erased completely.

2.3 Reading from/Writing to the file:

Contents can be written to/read from the file by using the object of the file stream class. The same procedure is followed to write to the file as in to write to the console. The classes used to do so are:

- **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.

These classes are derived directly or indirectly from the classes istream and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout, with the only difference that we have to associate these streams with physical files.

2.3 Closing the file:

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function close. This member function takes flushes the associated buffers and closes the file:

```
myfile.close();
```

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes.

2.4 File Structure Concept:

File Concepts used:

For the event_list file, we use a very simple file which takes the event data and automatically adds a sequentially incremented index to it and stores it in the list, in order.

For the remaining files we needed to adapt indexing and hashing to meet our needs as follows:

An **indexed file** is a computer file with an index that allows easy random access to any record given its file key.

The key must be such that it uniquely identifies a record. If more than one index is present the other ones are called *alternate indexes*. The indexes are created with the file and maintained by the system.

In our project we have used multiple files for representing various entities involved in the Event Management System. So creating and managing multiple index files for each entity wouldn't be feasible, as in most of the files such as *client_records* or *attendee_list* we retrieve the list with respect to one event and do not need direct access to individual clients or attendees. Hence, it would be inefficient to use the pure form of indexing as a method for storage and retrieval.

We can only adapt a part of the concept of indexing, where we store in a separate location the key- index pair.

Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value. It is also used in many encryption algorithms.

Hashing is also used for fast recovery using mathematical computations. It generates the index, based on the key value. One of the problems that arise with hashing is that when the hashing function generates the same index for multiple keys, collision takes place, to deal with this one method is creating a linked list and appending all new keys, with the same index, to it.

As explained above, we do not require fast recovery for retrieving individual clients, participants etc. Instead we could use the concept of collision resolution by adding linked

lists. Since we are adding and retrieving entities using the event_id as an identifier, it acts as a key, and there are bound to be collisions as there will be multiple entities associated with a single event.

Hence we combine parts from both concepts to create a modified version, which mainly works on offsets.

In this method, we add an index line to the beginning of each file.

The format of the index line is as follows:

e1:o1|e2:o2|e3:o3|

{e1, e2, e3,...} - Event ids

{o1, o2, o3,...} - Corresponding offsets

The set of event ids is not the entire list of event ids in the system, but just the ones that are mentioned in the file.

When a new record is to be added to the file, and its event id is e1 then the record is added to line at offset o1 from the beginning.

All offsets in the file with $o \geq o1$ are then incremented by one.

Hence at all times the offset points to the location where the next record of the corresponding index is to be stored.

Locating the record position is very easy as all the files have been created using fixed length records and the seek offset of the file is $o * \text{record_size}$.

When a new record with an event id not mentioned in the index line is to be entered, the event id offset pair is appended to the index line.

This method is very useful for our application as:

1. Saves space by using same file for indexing
2. Direct access to records.
3. Last offset in index line can be used to find the number of records in the file, without the need for counting.
4. $\text{Current_offset} - \text{previous_offset} = \text{number of records under corresponding event id}$
5. Entries with same event id are grouped together as sequential access is very easy.

Each of these benefits greatly help in reducing the processing time for the system.

As the last offset can be used to find the number of records in the file we use it to generate unique indices for participant ids.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Software Requirements

Software requirements deal with defining software requirements and pre-requisites that need to be installed on a computer to provide optimal functioning of a software.

These requirements or pre-requisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Following are the various aspects of software requirements

- **Operating System:** Windows 7/8/8.1/10 can be used as the operating systems.
- **IDE:** An integrated development environment for C++ such as Visual Studio Code.

3.2 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources.

Following are the various aspects of hardware requirements

- **Processor:** Any Intel or AMD processor that supports multi-core capabilities is required.
- **RAM:** More than 2GB RAM is sufficient for seamless working of the program.
- **Storage:** Approximately 100 MB of storage is required to store the records of the users.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

The project is designed so as to cater to all the needs of an event organizing firm. It provides quick response and retrieval of information from the file system enabling good accessibility to the user. The system navigates through the whole event management process with also an option to commit the changes or to roll back to the old data.

4.1 Add Records

The user can enter any new records by entering the corresponding details of each entity used in the event management system. The event list is stored as a simple file with auto generated index to each of its corresponding record. The other file records are sorted according to the **event id** of each of it respectively.

The system is designed in such a way that the entries of the new users are appended along with the entries of the previous users.

```
Code: int add(string fname, string e_id, int rec)
{
    ifstream f;
    f.open(fname);
    string temp;
    vector<string> v, e;
    int i, loc, n, sum=0;
    if (fname.compare(emfile) == 0)
    {
        while (!f.eof())
        {
            getline(f, temp, '\n');
            cout << temp;
            if (temp.length() > 10)
            {
                v = split(temp, "|");
                if (e_id.compare(trim(v[5])) == 0)
                    sum += stoi(v[4]);
            }
        }
        return sum;
    }
}
```

```
getline(f, temp, '\n');
if (temp.length() < 4)
    return 0;

v = split(temp, "|");

for (i = 0; i < v.size(); i++)
{
    if (v[i].length() > 3);
    {
        e = split(v[i], ":");

        if (e[0] == e_id)
            break;
    }
}

if (i == v.size())
{
    return 0;
}
n = stoi(e[1]);

if (i == 0)
    loc = 1;
else
{
    e = split(v[i - 1], ":");
    loc = stoi(e[1]);
}

n = n - loc;
if (fname.compare(atfile) == 0)
    return n;

for (i = 1; i < loc; i++)
    getline(f, temp, '\n');

for (i = 1; i <= n; i++)
{
    getline(f, temp, '\n');
    cout << temp;
    sum += stoi(getFeild(temp, rec));
}
f.close();
return sum;
}
```

4.2 Display Records

The details entered by the users which are stored in the text file are retrieved by opening the Files in read mode and displaying them to the users. Each file displays all the details entered by the user about a particular entity.

```
Code: void display(string fname, string e_id)
{
    ifstream f;
    f.open(fname);
    string temp;
    int i, loc, n;
    getline(f, temp, '\n');
    if (temp.length() < 4)
        return;
    vector<string> v, e;
    v = split(temp, "|");

    for (i = 0; i < v.size(); i++)
    {
        if (v[i].length() > 3);
        {
            e = split(v[i], ":");

            if (e[0] == e_id)
                break;
        }
    }

    if (e_id.compare("-1") == 0)
    {
        e = split(v[i - 2], ":");
        n = stoi(e[1]);

        loc = 1;
        goto print;
    }
    if (i == v.size())
    {
        cout << "No sponsor information found for specified
event\n";
        return;
    }
    n = stoi(e[1]);

    if (i == 0)
        loc = 1;
    else
    {
        e = split(v[i - 1], ":");
        loc = stoi(e[1]);
    }

    n = n - loc;
```

```
        //cout << "Test - loc " << loc << "\t no. of lines " << n << endl;
        //cout << f.tellg() << endl;
print://f.seekg((loc-1) * recsize), ios::cur);
        //cout << f.tellg() << endl;

        for(i=1;i<loc;i++)
            getline(f, temp, '\n');
        for (i = 1; i <= n; i++)
        {
            getline(f, temp, '\n');
            //cout << temp << endl;
            v = split(temp, "|");
            for (int j = 1; j < v.size(); j++)
            {
                cout << trim(v[j]) << "\t\t ";
            }
            cout << endl;
        }
        f.close();
    }
```

4.3 Modify Records

The employee records stored in the text file can be modified for any changes to be added or updated. The correct id of the record that has to be modified is given as input and then the desired options are selected. The system matches the given id and identifies the record to be modified. After modifying, the system reflects the same changes in the file.

Code: void modify()// does not modify changes to file

```
{
    int ch;
    string key_id;
    cout << "enter the Employee id to be modified\n";
    cin >> key_id;
    //Search the record store the index
    //return if not found
    cout << "1.Change id\n2.Update details\n";
    cin >> ch;
    fstream f1;
    string buff;
    char strid[5];
    f1.open(emfile, ios::in || ios::out);
    switch (ch)
    {
        case 1:
        {
            //buff = add new_id+genspace+record(5:)
            int ind = search_rec(key_id, emfile);
            if (ind != -1)
            {
                int new_id;
```

```
        cout << "Enter the new Employee id\n";
        cin >> new_id;

        fl.seekg(ind);
        getline(fl, buff, '\n');
        _itoa_s(new_id, strid, 10);
        //fl << strid << genspace(5 -
strlen(strid)) << buff << "\n";

        cout << "id modified";
        buff = strid + genspace(5 -
strlen(strid)) + buff + "\n";

        //read record
        //write buff
        /*cout << buff.length();
        cout << "test\n";
        cout << buff;
        cout << "test\n";
        fl.seekp(ios::end);
        fl << buff;*/
        // write buff to file
        //close file
        writeto(emfile, buff);
        fl.close();
    }
    else
        cout << "Invalid id \n";
    //fl.close();

}

break;
case 2:
{
    // open file
    // browse for id

    cout << "Enter new details \n\nName\tContact
no.\tDesignation\tSalary\tEvent id\n";
    cin >> name >> mobile >> desg >> salary >>
e_id;

    string buff;
    buff = key_id + genspace(5 - strlen(id)) +
    "|" + name + genspace(15 - strlen(name)) + "|" + mobile + "|" + desg +
    genspace(10 - strlen(desg)) + "|" + salary + genspace(10 - strlen(salary))
    + "|" + e_id + genspace(5 - strlen(e_id)) ;
    //cout << buff.length();
    //cout << "test\n";
    cout << buff;
    //cout << "test\n";
    fl << buff;
    // write buff to file
    //close file
    fl.close();
}

break;
```

```
        }  
    }  
}
```

4.4 Search Records

Once the user gives an id for a record, the system searches the record corresponding to that id and fetches it. Since the records are of fixed length, it uses the offset of the records as reference in searching.

```
Code: int search_rec(string id, string fname)  
{  
    //cout << "|" << id << "|" << endl<<id.length()<<endl;  
    fstream f;  
    int ind;  
    string temp, trunc;  
    f.open(fname, ios::in);  
    while (!f.eof())  
    {  
  
        getline(f, temp, '|');  
        ind = f.tellg();  
        temp = trim(temp);  
  
        getline(f, trunc, '\n');  
        if (temp.compare(id) == 0)  
        {  
            f.close();  
            return ind;  
        }  
    }  
    return -1;  
}
```

4.5 Source Code (main())

```
int main()  
{  
    clrscr();  
  
    int chl;  
  
    while (true)  
    {  
  
        home: clrscr();
```

```
type("1.Enter Data\n2.View Event Information\n");
cin >> ch1;
clrscr();
switch (ch1)
{
    enter:case 1:
    {
        type("1.New Event\n2.New Employee\n3.New
Client\n4.New Vendor\n5.New Sponsor\n6.New Advertising Agency\n7.Add
Participants/Attendees\n8.Commit Changes\n9.RollBack Changes\n");
        int ch;
        cin >> ch;
        switch (ch)
        {
            case 1:
            {
                event e;
                clrscr();
                e.addnew();
                //modify: details (except id)
                //no delete
            }
            break;

            case 2:
            {
                clrscr();
                //1.add: similar add func
                //2.modify: other details
                //3.modify: id - change id
                in program, sort, replace old e_id with new e_id in all files
                //4.delete : remove employee
                cout << "1.Create an entry
for a new employee\n2.modify existing employee\n";
                int ch2;
                cin >> ch2;employee em;
                switch (ch2)
                {
                    case 1:
                    {
                        em.addnew();
                    }
                    break;
                    case 2:
                    {
                        em.modify();
                    }
                    break;
                    default: cout << "invalid
choice\n";
                }
            }
            break;

            case 3:
            {
                clrscr();
                client c;
                c.addnew();
                //add no modify no delete
```



```
        }
        break;
    case 4:
    {
        clrscr();
        vendor v;
        v.addnew();
        //similar to client
    }
    break;
    case 5:
    {
        clrscr();sponsor s;
        s.addnew();
        //similar to client
    }
    break;
    case 6:
    {
        clrscr();advert a;
        a.addnew();
        //similar to client
    }
    break;
    case 7:
    {
        clrscr();attendee at;
        at.addnew();
        //similar to client
    }
    break;
    case 8:
    {
        commit();
    }
    break;
    case 9:
    {
        rollback();
    }
    break;

    default: cout << "Invalid choice";
            goto home;
    }

    //Sleep(3000);    //test
    goto enter;
}

break;

case 2:
{
    type("1.Display Event List \n2.View
Sponsorship Information\n3.View Clients\n4.View Vendors\n5.List of
```

```

int ch;
cin >> ch;
switch (ch)
{
    case 1: // display event list
    {
        fstream fl;
        string temp;
        fl.open(evfile, ios::in);
        clrscr();
        while (!fl.eof())
        {
            getline(fl, temp);
            cout << temp << endl;
        }
        fl.close();
    }
    Sleep(3000);
    break;

    case 2:// View Sponsorship
    {
        cout << "Enter event id -";
        string e_id;cin >> e_id;

        cout << "Name\t\t Mobile\t\t
        Email\t\t
        Amount\n_____ \n";

        display(spfile, e_id);
        cin >> e_id; //test

        // match event id with
        sponsors file
    }
    break;

    case 3:// View Clients
    {
        clrscr();
        cout << "1.Display all
        Clients\n2.Specific to an event\n";

        int ch;string e_id;
        cin >> ch;

        if (ch == 1)
        {
            cout << "Name\t\t
            Mobile\t\t Amount\n_____ \n";
            display(clfile, "1");
            cin >> e_id; //test
        }
        else

```

```
id -";

{
    cout << "Enter event

    cin >> e_id;

    cout << "Name\t\t
Mobile\t\t Email\t\t
Amount\n_____ \n";
    display(clfile,
e_id);
    cin >> e_id; //test
}

// match event id with
sponsors file
}
// match event id with client file
break;

case 4:// View Vendors
{
    cout << "Enter event id -";
    string e_id;cin >> e_id;
    vendor v;
    cout << "Name\t\t Mobile\t\t
Description\t\t
Amount\n_____ \n";
    display(vfile, e_id);
    cin >> e_id;
}
// match event id with client file
break;

case 5: // View Attendees
{
    cout << "Enter event id -";
    string e_id;
    cin >> e_id;
    cout << "id\t\tName\t\t
Mobile\t\t e-
mail\n_____ \n";
    display(atfile, e_id);
    cin >> e_id;
}
break;

case 6:// income expense
{
    cout << "Enter event id -";
    string e_id;
    cin >> e_id;
    // income = sum of amount
from (client+sponsor) + num(attendees)*event_id.entryfee
    long sum = 0;
    fstream f;
    f.open(evfile, ios::in);
```

```
evfile));

split(temp, "|");
stoi(trim(v[3]));
stoi(getFeild(temp, 5));

add(emfile, e_id);
add(spfile, e_id) + (add(atfile, e_id) * entryfee);
+ add(adfile, e_id) + add(vefile, e_id);

of amount from (employee+advert+vendor)
gain/loss%

} break;

case 7:// advert analysis
// enter event no
// total amount spent:
expected attendees: actual attendees: cost/attendee:
break;

case 0: goto home;
default: cout << "Invalid choice";

}
}
default: cout << "Invalid choice";

}
}
```

CHAPTER 5

SNAPSHOTS

Home Screen:

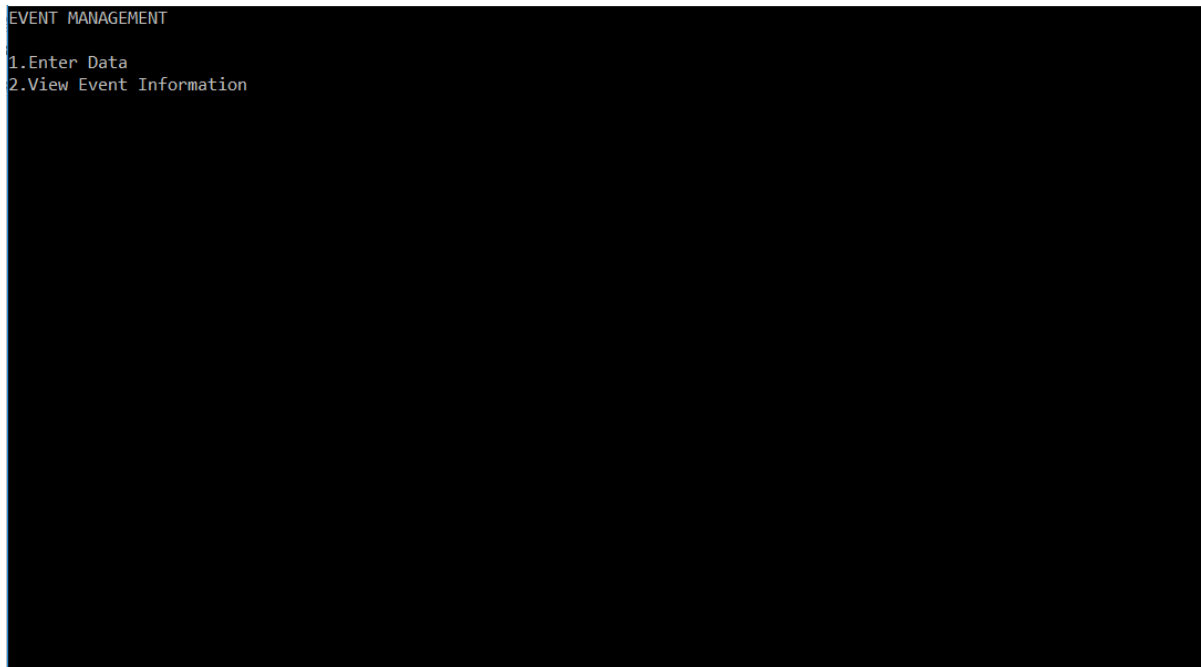


Fig. 1, Home Screen

At the initialization of the software, the user is displayed with the above screen. The name of the software is “Event Management” and it immediately gives the user two options to choose from:

1. Enter Data

This option can be chosen by the user to enter new records under various entities in the corresponding files.

2. View Event Information

This option is chosen by the user to view existing information on all entities in the system and also to analyze and compare few details.

Upon giving input as **1** the user is displayed with the menu given below.

5.1 Enter Data:

```
EVENT MANAGEMENT
1.New Event
2.New Employee
3.New Client
4.New Vendor
5.New Sponsor
6.New Advertising Agency
7.Add Participants/Attendees
8.Commit Changes
9.Rollback Changes
```

Fig. 2, Enter data

This page is displayed after the above action to enter data. It consists of 9 options of new event, new employee, new client, new vendor, new ad agency, new attendees, commit changes and rollback changes. The desired option is chosen by the user by giving the corresponding number as input to the system. This is implemented using the switch statement in the program.

Now, if the user inputs **1** the following page appears.

5.1.1 New Event

```
EVENT MANAGEMENT

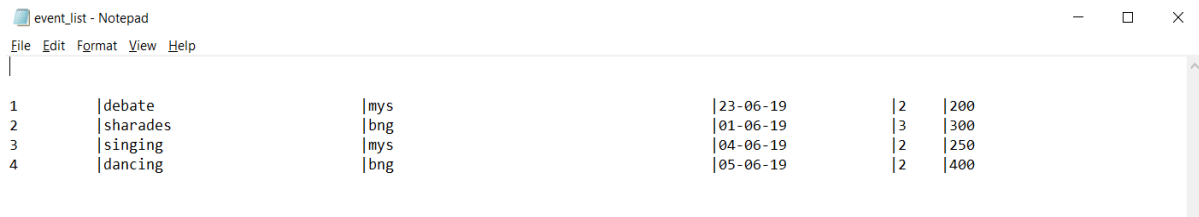
Number of records to enter :2
Enter the details
Name      Location      Date(dd-mm-yyyy)      Duration(number of days)      Entry Fee
singing
mys
04-06-19
2
250
dancing
bng
05-06-19
2
400
1.New Event
2.New Employee
3.New Client
4.New Vendor
5.New Sponsor
6.New Advertising Agency
7.Add Participants/Attendees
8.Commit Changes
9.Rollback Changes
```

Fig. 3, New Event

The fields for name, location and rest is given as input. We could simultaneously enter many records at once continuously by entering the number at first.

An **event id** is auto generated by the system for every event entry which is the first column of the event file.

The following figure shows the file where the event list is stored. It is a simple file with indexes given to records linearly.



1	debate	mys	23-06-19	2	200
2	sharades	bng	01-06-19	3	300
3	singing	mys	04-06-19	2	250
4	dancing	bng	05-06-19	2	400

The user then can select the option for entry of new employees.

5.1.2 New Employee

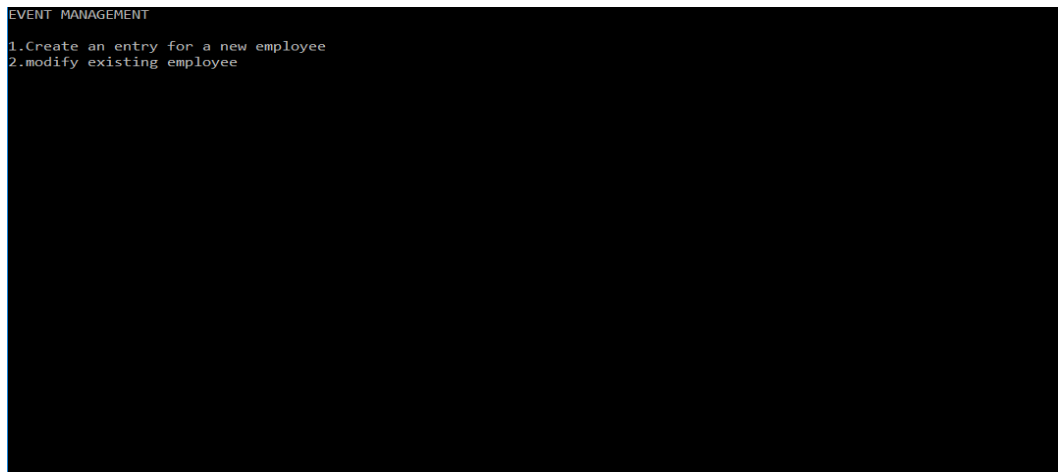


Fig. 4, New Employee

The first option is chosen to create an entry for a new employee. The system asks the user for the number of employee records it wants to enter simultaneously at once. The details are taken as input in order.

Each employee is supposed to be given an event id to represent the event to which he/she belongs.

The snapshot below shows the process.

```
1.Create an entry for a new employee
2.modify existing employee
1
Number of records to enter :2
Enter the details
Name      Contact no.      Designation      Salary      Event id
zoya
7348903756
volunteer
2300
2
1      |zoya      |7348903756volunteer |volunteer      |2300      |2
suraj
8890123876
volunteer
3000
2
1      |suraj      |8890123876volunteer |volunteer      |3000      |1
1.New Event
2.New Employee
3.New Client
4.New Vendor
5.New Sponsor
6.New Advertising Agency
7.Add Participants/Attendees
8.Commit Changes
9.RollBack Changes
```

Fig. 5, Employee Entry

The corresponding file for employee records is shown below with an index for the number of employees and also the event id for each entry.

```
employee_records - Notepad
File Edit Format View Help
|
1      |zoya      |7348903756volunteer |volunteer      |2300      |2
2      |suraj      |8890123876volunteer |volunteer      |3000      |1
3      |diya      |9980053016volunteer |volunteer      |2000      |3
4      |payo      |9670383979volunteer |volunteer      |3000      |4
```

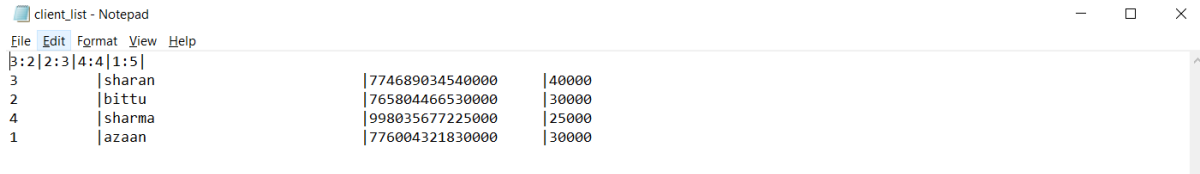
5.1.3 New Client

```
EVENT MANAGEMENT
Enter details
Name      Contact no.      Amount      Event id
azaan
7760043218
30000
1
1.New Event
2.New Employee
3.New Client
4.New Vendor
5.New Sponsor
6.New Advertising Agency
7.Add Participants/Attendees
8.Commit Changes
9.RollBack Changes
```

Fig. 6, New Client

The user can enter the record for a new client by pressing 3. Then the details to be entered are asked by the system. Each client is given event id representing the event it belongs to.

The file that stores the client information is shown below. This file has a special offset line at the start that indicates the **event_id:offset**. Here, this pairs (separated by delimiters) represent the event id of the record along with the next entry address for a record with the same event id. Also, this line also gives the length of the file, as the last offset represents the end of file always. This method is used for easy retrieval.



```
client_list - Notepad
File Edit Format View Help
3:2|2:3|4:4|1:5|
3      |sharan      |774689034540000    |40000
2      |bittu       |765804466530000    |30000
4      |sharma      |998035677225000    |25000
1      |azaan       |776004321830000    |30000
```

5.1.4 New Vendor

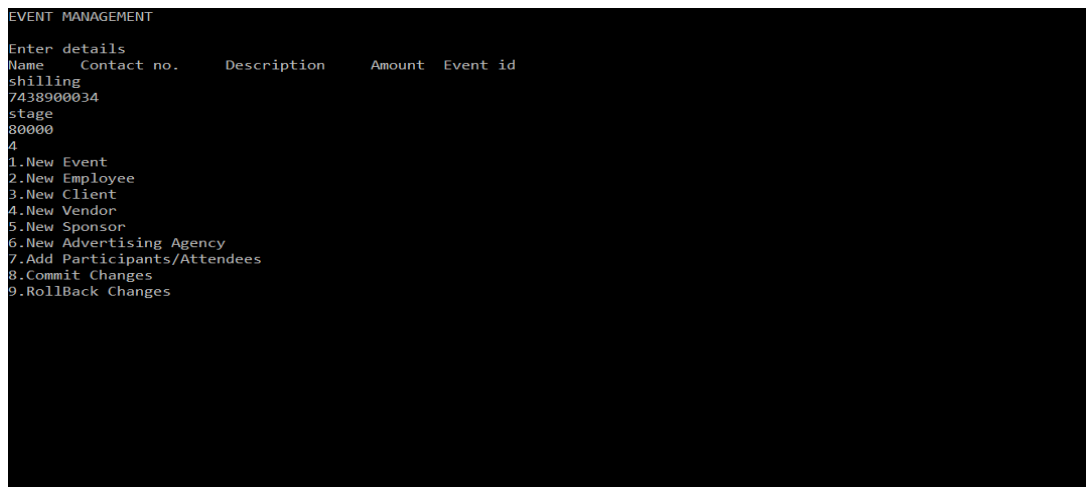
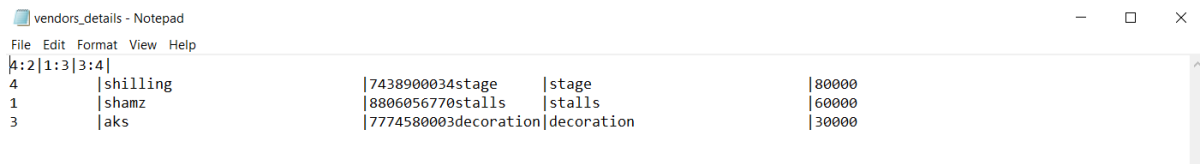


Fig. 7, New Vendor

This option takes the details of any vendors in the event management process. Their details are all stored in a file along with the corresponding event id for each vendor. The file that stores vendor records is shown below.



```
vendors_details - Notepad
File Edit Format View Help
4:2|1:3|3:4|
4      |shilling    |7438900034stage    |stage      |80000
1      |shamz      |8806056770stalls   |stalls     |60000
3      |aks        |7774580003decoration|decoration  |30000
```

The file for vendor details also has the offset line for easy retrieval of data. The vendor records are stored according to entry.

5.1.5 New Sponsor

```
EVENT MANAGEMENT
Enter details
Name      Contact no.      Email      Amount      Event id
```

The sponsors' details are entered by the user which include the amount sponsored and for which event by including the event id. The sponsor records file has all the entries stored along with their event id. It is similar to the client records file.

5.1.6 New Ad Agency

```
EVENT MANAGEMENT
Enter details
Company Name      Email      Location      Expected Reach      Amount      Event id
```

The advertising details for the events are entered in this entity. The name of the company advertising, location, the expected marketing and the amount spent on it is entered. Event id is also entered to indicate the event which the ad agency is advertising.

5.1.7 Participants details

This entry is for the all participant details for all the events. Each participant is given an auto generated participant id for uniqueness. Each participant is also assigned an event id to indicate the event in which he/she is participating.

5.1.8 Commit

Commit is an option given to save the changes/new entries made in the file system. It saves the entries made to prevent loss of data/records after a system crash/failure.

Code: void commit()

```
{
    movef(oevfile, evfile);
    movef(oatfile, atfile);
    movef(ospfile, spfile);
    movef(ovefile, vefile);
    movef(oadfile, adfile);
    movef(oemfile, emfile);
    movef(oclfiler, clfile);
}
```

5.1.9 Rollback

Rollback is an option given to go back to the previous state of the system, i.e. before the new entries/modifications. This option enables the user to go back to the old data/records in the system. However, Rollback doesn't work once Commit is done for the new entries/modifications.

Code: `void rollback()`

```
{
    movef(evfile, oevfile);
    movef(atfile, oatfile);
    movef(spfile, ospfile);
    movef(vefile, ovefile);
    movef(adfile, oadfile);
    movef(emfile, oemfile);
    movef(clfile, oclfile);
}
```

5.2 View Event Information

This option is displayed on the home screen as soon as the program executes. This option is used to view the already entered data, i.e. the event list, sponsorship details, clients, vendors and attendees. This option also has options to compare expenditure and income on a particular event and also analyze the advertising for an event.

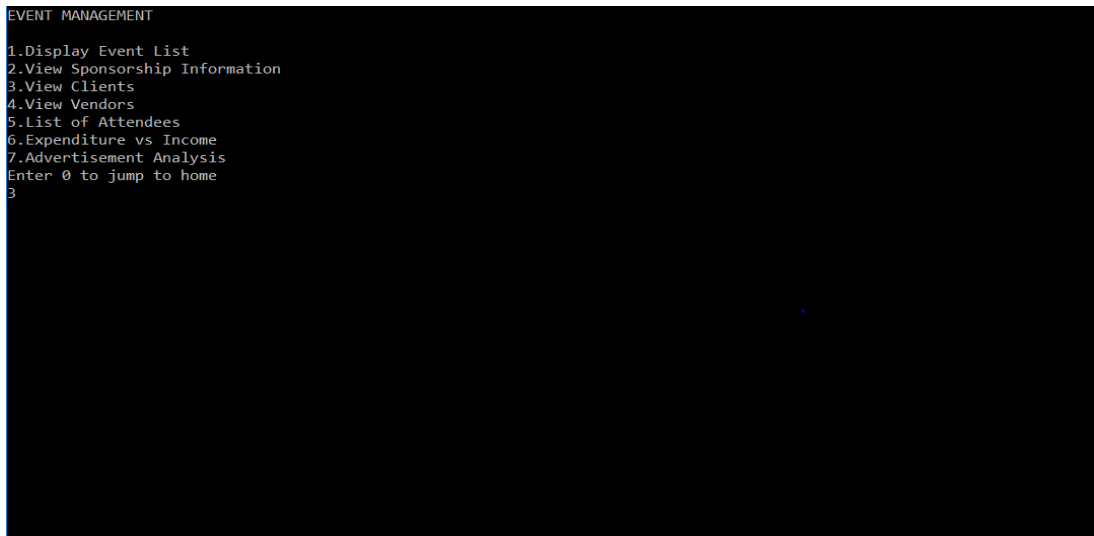
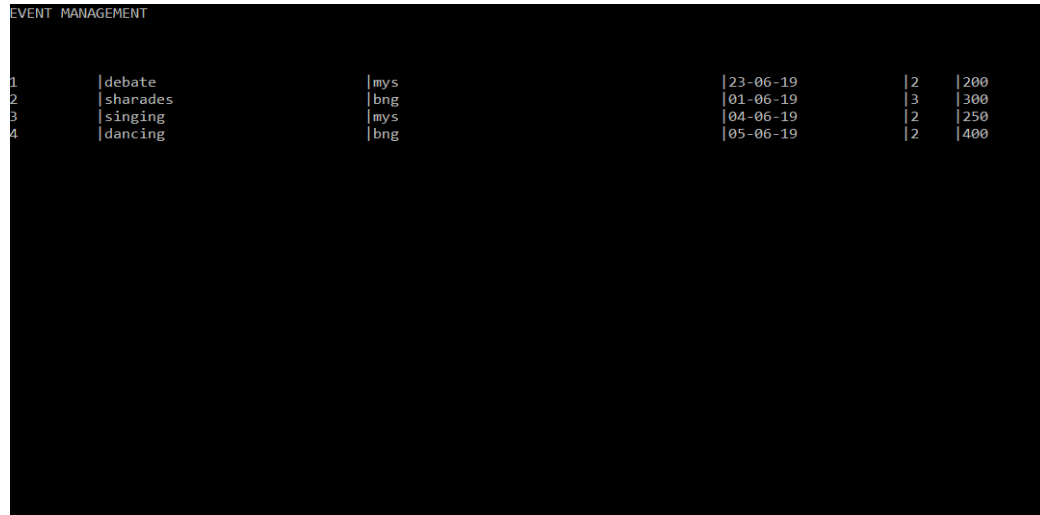


Fig. 8, View event Information

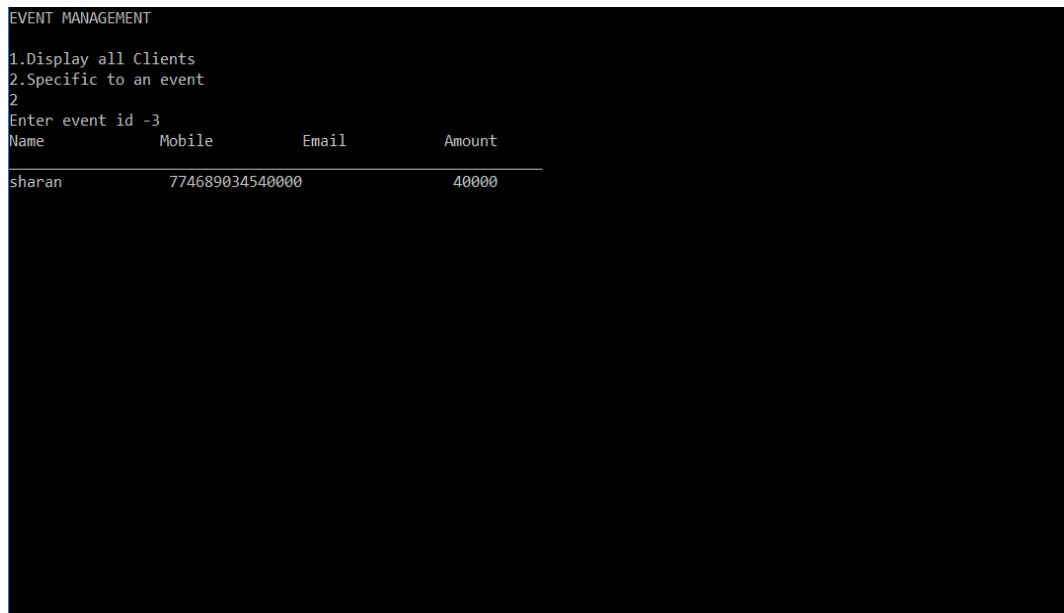


The screenshot shows a terminal window titled "EVENT MANAGEMENT". It displays a list of four events, each with an ID, name, category, date, and amount. The events are: 1. debate (mys, 23-06-19, 2, 200), 2. sharades (bng, 01-06-19, 3, 300), 3. singing (mys, 04-06-19, 2, 250), and 4. dancing (bng, 05-06-19, 2, 400).

ID	Name	Category	Date	Amount
1	debate	mys	23-06-19	200
2	sharades	bng	01-06-19	300
3	singing	mys	04-06-19	250
4	dancing	bng	05-06-19	400

Fig. 9, Event List

The client details are displayed according to options, either all the details or the ones specific to a single event.



The screenshot shows a terminal window titled "EVENT MANAGEMENT". It displays a menu with two options: "1.Display all Clients" and "2.Specific to an event". Option 2 is selected, and the user is prompted to "Enter event id -3". The system then displays a table of client details for event ID 3, including Name, Mobile, Email, and Amount. The client details shown are: sharan, 774689034540000, 40000.

Name	Mobile	Email	Amount
sharan	774689034540000		40000

Fig. 10, Client Details

```
EVENT MANAGEMENT
1.Display Event List
2.View Sponsorship Information
3.View Clients
4.View Vendors
5.List of Attendees
6.Expenditure vs Income
7.Advertisement Analysis
Enter 0 to jump to home
2
Enter event id -1
Name          Mobile          Email          Amount
-----
vidit         7689055669vidits@ymail.com    vidits@ymail.com    35000
```

Fig. 11, Sponsorship details

To go back to the main menu, the user hits 0. The home screen is displayed for further working of the user.

CONCLUSION

The package was designed in such a way that future modifications could be done easily.

The primary outcome of this software is to present an efficient retrieval-based Event Management System for the end users such that they can efficiently organize events and have faster access to details at their end.

The following conclusions can be made from the development of the above project:

- The Event Management software improves the efficiency at which a person can handle the events being organized with detailed entries for everything needed.
- By using the concepts of offsets, the access time of the records is decreased.
- The system has adequate scope for modification in the future.

REFERENCES

- <https://www.geeksforgeeks.org/dbms-gq/file-structures-sequential-files-indexing-b-and-b-trees-gq/>
- <http://www.cplusplus.com/>
- www.quora.com/
- <https://stackoverflow.com/>
- <https://www.sciencedirect.com/topics/computer-science/byte-offset>