Random Access File Example.

Mr. Smith has just been elected secretary of the Hecklethorpe Astrological Society, and he wants to hold the membership details on his computer. He needs each member's name, address and telephone number, but also wishes to store their star sign. He is not sure whether he can legally do this without registering under the Data Protection Act, since this is extra information not normally included under the club membership exemption rule. However, he decides (rightly or wrongly) that he can get round this problem by including the star sign coded as part of the member's membership number.

The society limits itself always to no more than exactly 100 members. These could be numbered 0 to 99, i.e. minimally a 2-digit membership id. Mr. Smith decides to use randomly generated 2-digit ids, prefixed by a 2-digit star sign code, between 01 and 12 for the twelve star signs, making a four-digit number. Thus the range of ids lies between 0100 and 1299.

Mr. Smith, being a tidy person, decides that as members join and leave, he does not want his database file to change size, perhaps resulting in fragmentation and slowing the system down. "Aha!" he says, "I will generate a random access file which will stay the same size!" He happens to be interested in mathematics in addition to conjunctions of Jupiter and Venus, so it takes him but a few seconds to figure that his id method gives him a possible 1200 ids. So if he allows for all these, his file would have to accommodate 1200 records. "Aha!" he repeats, musingly, "My idea of using a Random file is brilliant – I have only a maximum of 100 members, so I will use a hash formula to condense the id range to a physical file size of only 100 records. Actually, since the society is paying, and I'm not the treasurer, and memory is reasonably cheap anyway, I will settle for a file size of 200 records. This will reduce the rate of collisions, and speed up searching, and still avoids wasting 1000 record spaces".

So he works on a suitable hash formula. Having a recent quite fast processor on his computer, he decides to use each digit of the membership id in the hash formula. He had heard somewhere that multiplying by prime numbers might be good for a laugh, so he decided to use the following:

> If the membership id is represented by abcd, where ab is the star sign and cd is a two digit number generated to make the final id unique, then
> Total = (a * 43) + (b*47) + (c*71) + (d*97)
> This would give a number between 47 and 2021, so to reduce this to a number between 0 and 199, the total should be divided by 200, taking the remainder, i.e.
> Record number = total mod 200.

To store a member's record, then, he would generate an i.d. for the member, then check that it didn't already exist. If it did, he would generate another, until a unique id had been found.
The id would be hashed to give the location (record number on the file). If that location was available, then the record would be stored there. In the event of a collision, the record would be stored in the first free slot following the calculated position (using wraparound from the end of the file to the start if necessary).

He then wrote a Pascal program to generate the (empty) data file. The record structure would be:

tag	Boolean	true if details stored, false if free
id	word	membership id as described, interpreted as 4-digit id but stored as binary word
name	string[30]	
address1	string[20]	
address2	string[20]	
address3	string[20]	
postcode	string[10]	
tel	string[15]	

Total record size = 1 + 2 + 31 + 21 + 21 + 21 +11 + 16 = 124 bytes
Expected file size = 124 * 200 = 24800 bytes = 24.22Kb

The Pascal program was:

```
program createHASfile;
 uses crt;

 type Tmember = record
      tag : boolean;
      id  : word;
      name :  string[30];
      address1 : string[20];
      address2 : string[20];
      address3 : string[20];
      postcode : string[10];
      tel  : string[15];
 end;

 var rec : Tmember;
     f : file of Tmember;
     i: integer;
     fn : string[50];

 begin
    clrscr;
    writeln('Type pathname of file to create');
    readln(fn);
    assign(f,fn);
    rewrite(f);
    rec.tag:=false;
    rec.id:=0;
    rec.name:='';
    rec.address1:='';
    rec.address2:='';
    rec.address3:='';
    rec.postcode:='';
    rec.tel:='';
    for i:=1 to 200 do
       write(f,rec);
    close(f);
    writeln('OK, file ',fn,' created.  Press <enter>');
    readln;
 end.
```

Mr. Smith then decided to develop the rest of the project using Delphi. He would subsequently use the Pascal program only to create new files, but he would use Delphi to do all the manipulation.

The required manipulations would be:

- add new member
- display all members
- select members (and display) by
    - star sign
    - name
    - membership id
- Possibly produce mailing labels from such a selection.

Mr. Smith came up with some pseudocode for these functions, which would be chosen from a menu:

<u>Add new member.</u>

Enter rec.name, rec.address etc.
rec.tag:=true;
Enter dateofbirth;
calculate starsign from dateofbirth;

repeat
produce rec.id from starsign and random 2-digit number;
read record at location hash(rec.id);
until record.tag = false;
write rec at this location.

Display all members

This could be done either by reading the database sequentially, and printing all records for which .tag is true. Alternatively, it could be done by hashing each possible id in turn, and printing the corresponding records. Mr. Smith opted for a simple sequential read of the 200 records in the file.

while not eof(membershipfile) do
begin
read rec;
if rec.tag then print details;
end;

Being in a hurry, and keen to start button-pushing, Mr. Smith (realising that Delphi is a Rapid Application Development tool) at this point decided to implement these two functions. Remembering what his Computing teacher had taught him, he created a new directory for his Delphi project, then having loaded Delphi-5, he immediately saved his new project into the new directory. He set suitable form captions, and created a menu bar. He also cannily copied the record type definition from his Pascal file create program, thus ensuring that the definitions would be identical. He then added code for the open file option, having added an opendialog component to the form. He also added the close command to the exit option. At this stage, the code looked like:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    Fileopen1: TMenuItem;
    Addmember1: TMenuItem;
    Display1: TMenuItem;
    Exit1: TMenuItem;
    OpenDialog1: TOpenDialog;
    procedure Fileopen1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure Addmember1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

type Tmember = record
      tag : boolean;
      id  : word;
      name :  string[30];
      address1 : string[20];
      address2 : string[20];
      address3 : string[20];
```

```
      postcode : string[10];
      tel  : string[15];
end;

var
  Form1: TForm1;
  rec, rec1 : Tmember;
  f : file of Tmember;

implementation

uses Unit2;

{$R *.DFM}

procedure TForm1.Fileopen1Click(Sender: TObject);
begin
    if opendialog1.Execute then
        begin
            assignfile(f,opendialog1.filename);
            reset(f);
        end;
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
   close;
end;

end.
```

He then decided to implement the add member option using a second form, Form2, with code Unit2. (He could have renamed the forms and units, giving them meaningful names, but being somewhat lazy, he didn't get round to this). He added edit boxes to allow entry of name, address, tel, and to display the membership id, and a maskedit box for the DOB (using the default short date mask). He then added a button which would add the record based on the details entered. He added code to the addmember menu option on Form1 in Unit1:

```
procedure TForm1.Addmember1Click(Sender: TObject);
begin
  randomize;
  Form2.show;
end;
```

(Randomize was added to seed the random number generator prior to producing hash values).

To facilitate the coding of the addmember button, Mr. Smith copied his pseudocode into Delphi's buttonclick procedure skeleton. He carefully enclosed each line of pseudocode in curly brackets, cleverly creating comments to self-document the code, simultaneously making his coding job easier by prompting him as he went along. He then added lines of code, including a function to calculate the hash values, producing:

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Mask;

type
```

```
  TForm2 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    Edit6: TEdit;
    Edit7: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Button1: TButton;
    MaskEdit1: TMaskEdit;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

uses Unit1;

{$R *.DFM}
function hash(id:integer):integer;
var t,i : integer;
begin
      {Total = (a * 43) + (b*47) + (c*71) + (d*97)}
      t:=0;
      i:=id;
      t:=(i mod 10)*97;
      i:=i div 10;
      t:=t + (i mod 10)*71;
      i:=i div 10;
      t:=t + (i mod 10)*47;
      i:=i div 10;
      t:=t + (i mod 10)*43;
      t:=t mod 200;
      hash:=t;
end;

procedure TForm2.Button1Click(Sender: TObject);
var d, m, y, ss, loc : integer;
    ok : boolean;
begin
{Enter rec.name, rec.address etc.}
   rec.name:=edit1.text;
   rec.address1:=edit2.text;
   rec.address2:=edit3.text;
   rec.address3:=edit4.text;
   rec.postcode:=edit5.text;
   rec.tel:=edit6.text;
{rec.tag:=true;}
   rec.tag:=true;
{Enter dateofbirth;}
```

```
   d:=strtoint(copy(maskedit1.text,1,2));
   m:=strtoint(copy(maskedit1.text,4,2));
   y:=strtoint(copy(maskedit1.text,7,2));
   ok:=true;
   if d<1 then ok:=false;
   case m of
     9,4,6,11 : if d>30 then ok:=false;
     1,3,5,7,8,10,12 : if d>31 then ok:=false;
     2 : if (d>29) or ((d=29)and((y mod 4)<>0)) then ok:=false;
    else ok:=false;
    end;
   if not ok then showmessage('Date invalid - please check')
       else
  begin

{calculate starsign ss from dateofbirth;  aries=1 to pisces=12}
   case m of
      1 : if d<21 then ss:=10 else ss:=11;
      2 : if d<19 then ss:=11 else ss:=12;
      3 : if d<21 then ss:=12 else ss:=1;
      4 : if d<21 then ss:=1 else ss:=2;
      5 : if d<22 then ss:=2 else ss:=3;
      6 : if d<22 then ss:=3 else ss:=4;
      7 : if d<23 then ss:=4 else ss:=5;
      8 : if d<24 then ss:=5 else ss:=6;
      9 : if d<23 then ss:=6 else ss:=7;
      10: if d<24 then ss:=7 else ss:=8;
      11: if d<23 then ss:=8 else ss:=9;
      12: if d<22 then ss:=9 else ss:=10;
   end;
{repeat}
   repeat
{produce rec.id from starsign and random 2-digit number;}
        rec.id:=ss*100+random(99);
{read record at location hash(rec.id);}
        loc:=hash(rec.id);
        seek(f,loc);
        read(f, rec1);
{until record.tag = false;   }
   until not rec1.tag;
{write rec at this location.}
   seek(f,loc);
   write(f,rec);
   edit7.text:=inttostr(rec.id);
   if Messagedlg('Record written',mtconfirmation,[mbOK],0)
      = mrOK then form2.hide;
  end;
end;

end.
```

To implement the display routine, Mr. Smith added a memo box to Form1, with scrollbars on, then coded the display menu option to read sequentially through the file, printing details for those records for which the tag field is set.

```
procedure TForm1.Display1Click(Sender: TObject);
var i:integer;
begin
   reset(f);
   memo1.clear;
   i:=0;
   while (not eof(f)) and (i<199) do
    begin
```

```
      i:=i+1;
      read(f,rec);
      if rec.tag then
        begin
          memo1.lines.add(inttostr(i));
          memo1.Lines.add('Membership id: '+inttostr(rec.id));
          memo1.Lines.add('Address: '+rec.address1);
          memo1.lines.add('         '+rec.address2);
          memo1.lines.add('         '+rec.address3);
          memo1.lines.add('Postcode:'+rec.postcode);
          memo1.lines.add('Tel: '+rec.tel);
          memo1.lines.add('');
        end;
    end;
end;
```

A working application was thus created. Mr. Smith gave himself a star form.

(Note added later: Mr. Smith had overlooked one problem with mixing Pascal and Delphi. If you can find it, you can give yourself a star form too!)