10 Projects

10.1 Introduction

Most of us are required to complete a substantial programming project. Typically, this would involve writing a program to maintain a file. Either we are given a well defined task to do or we choose one of our own.

10.2 Choosing a Subject

The project you choose should have sufficient scope to demonstrate analytical and design skills and your ability to use a wide range of programming techniques. Here are some ideas for a programming project.

1 Dating agency
2 Employment agency
3 Farming records
4 Plant nursery
5 Resource scheduling, eg rooms in a college

6 Hotel/holiday bookings
7 Airline reservation system
8 Squash court reservation system
9 Cash point machine simulation
10 Point of sale terminal

11 Specialised diets
12 Direct computer supplies mail order business
13 Labelling drugs dispensed by a pharmacy
14 Criminal investigation/enquiry system
15 Student records system

16 A simple spreadsheet
17 Forecasting crime
18 Spell-checker
19 Editor (a simple word processor)
20 Classification system eg books in a library

21 Bird identification system
22 A program style checker
23 A language translator eg English into French
24 Books sent on approval recording system
25 Advertising hoardings control system

26 Immigration control
27 Car/van/boat/coach hire
28 Computer printer paper stock control and automatic re-ordering system
29 Evening course enquiries and booking system at a college
30 City marathon entries

31 College library services
32 Hospital/dental/GP/ records and automatic patient recall system
33 Theatre seat reservations
34 Taxi business - private hire
35 Time accounting - preparation of bill to repair, say, electrical goods

36 Time accounting in a solicitors/accountants office
37 Credit card accounting for a department store
38 Scheduling music exams for a school of music
39 Video hire company
40 Horse trials reservations, entries and results system

41 Recreation parks maintenance for a city council
42 Costing the manufacture of goods such as PC's/Amplifiers/CD players
43 Motor vehicle maintenance schedule for a haulage/bus company
44 Tracking parcels in transit - parcel carrying company
45 Software packages used by a college/company

46 Personnel management in a company manufacturing plastic dinosaurs
47 Payroll for a medium-sized knitwear company
48 Objects etc in a city museum
49 Paintings etc in a city art gallery
50 Vet's patients records system

51 Piano tuner clients database and automatic reminder system
52 Garage - customer vehicles maintenance/repair and service reminder
53 Estate agents
54 Parking ticket fine repayments system
55 Prison service records

56 Leisure centre membership
57 Insurance company policies and pensions
58 Car insurance quotation and brokerage system
59 Skip hire
60 Mortgage loans vetting system

The project you choose should be non-trivial, of interest to you and approved by your programming supervisor.

10.3 The Project Report

A project is usually evaluated on the basis of a project report together with a demonstration of the working program (or programs).

Your report must be word processed and sectioned.

If a marking scheme is supplied, then you should constantly bear it in mind when you write up your project report. A useful technique is to provide a heading for each component of the marking scheme and then ensure that you include the required details under each heading.

If the marking scheme is not so detailed, then consider providing each of the following sections in your project report

A CONTENTS PAGE

Each page of the project report should be numbered. The first page, a contents page, lists the main sections of the project report together with their page numbers.

B SUMMARY

Although this section is the second one to appear in the report, it is usually written last. This section contains an outline of your project. It comprises the main points such as the subject chosen, what you set out to do and what you actually achieved. The summary should take up no more than one side of paper.

C SYSTEM DESCRIPTION

A program does not exist in isolation. It is usually a very small part of a much wider data processing system. In this section, you should describe the entire data processing system in as much detail as possible. For example, suppose your project features a Video Hire Shop, then this section should describe in detail

- how individual copies of each video are identified
- the order in which the videos are placed on the shelves
- whether a person has to be a member before they can take a video out
- how a person selects a video
- how a person's choice is recorded
- how an employee knows where each copy of each video is currently located
- what happens when a video is returned on time
- what happens when a video is returned late
- how an employee knows when a video has not been returned
- what happens when a booked out video is lost or stolen
- what happens when a video becomes 'worn out'
- how theft of videos is detected
- how the manager knows what new video titles are available
- how the manager decides what new videos to acquire
- what happens when new videos arrive in the shop
- how the manager determines what the day's money receipts should be

- how the manager knows which titles or categories are most/least popular
- how the manager determines whether the business is making a profit
- how cash flow for each month is predicted
- how predicted cash flow is compared with actual cash flow

If possible, you should work (free of charge?) in a Video Hire Shop to gain first hand experience of how such as shop is actually run.

However, it is not usually possible to obtain experience in the subject of your choice. For example if you decide to do a project on Immigration Control, then it is unlikely that you will be allowed to see what goes on behind the scenes and they will not tell you what happens either. In such cases, you will have to use your imagination. But at least you should be able to obtain application forms and copies of rules which are available to the general public.

You should obtain any brochures and forms relevant to your project and do as much research as you reasonably can in the time you have got.

D PROGRAM DESCRIPTION

You should choose just a small part of the data processing system described in section C above and describe what you want your program (or programs) to do.

E REPORT SPECIFICATIONS

Many projects will print the contents of a file in the form of a report. In this section, you specify using printer format sheets, the layout of each report.

F THE USER INTERFACE

Sketches of the screens the user will see and use should be in this section. Further, you should describe and justify your designs for the user interface.

G DATA DICTIONARIES

The structure and content of every file should be explained in this section. For each file you should specify its name, size, purpose, access mode and record name. For each record you should list, specify and explain each field. Data dictionary forms would be useful here.

H PROGRAM STRUCTURE CHARTS

Program structure charts should show, for each program, the main functions and the data types of their arguments and return values. No structure chart should fill more than one side of normal size paper. If a structure chart cannot be fitted comfortably on one side of paper, then each of its main parts should be expanded on a second sheet. For example, suppose a menu function called seven other functions (and each of these functions themselves called other functions) then one page would contain just the menu function together with the seven functions; another page would feature just one of these seven functions together with the functions it calls. And so on.

Remember to draw your structure charts large and neat. A sharp pencil used throughout, for lines, arrows and captions, is perhaps best.

I TEST PLANS

Your test plans should

- specify how every choice in every menu is to be tested
- specify the contents of each file
- list the changes to be made to each file

together with any other values required to thoroughly test your program or programs.

J EVALUATION OF DESIGN

A colleague should examine your designs and just list any faults he or she thinks it contains. Nothing more. Then you have the choice of either ignoring them, or doing nothing about them except to mention them in your own evaluation of your project or re-designing your project. The aim here is to at least be aware of any shortcomings in your designs.

K PROGRAM LISTINGS

Your program listings should be clean, that is, free of any hand-written changes or comments. Of course, the programs should compile without error.

L PROGRAM RUNS

Here, you should provide evidence of your program running successfully. This evidence may take the form of printed output from of file contents or, where there is no screen formatting, program runs can be echoed to the printer. Hardcopy MUST be annotated, titled and explained by hand.

Photographs of your screen displays are useful.

M TEST LOGS

Your test logs should be an honest account of all your program runs, even the failures.

N USER INSTRUCTIONS

The user wants to know how to run your program, not how to use the editor and compiler! The user needs to know what files must be on what disks and in which directories. The user wants to know what happens when the program is running and what to do in the event of program failure. To test the effectiveness of your user instructions, consider inviting a non-computer person to run your program from your written instructions alone, without any verbal help from yourself.

O EVALUATION OF IMPLEMENTATION

A colleague should run your program and list its shortcomings. The purpose here is just to detect errors.

P EVALUATION AND REVIEW

In this section you look back over what you have done and compare what you have achieved with what you set out to do.

You discuss any shortcomings of your programs (such as file only contains 100 records, in a real situation, ten thousand records would be held; or the user is not allowed to input the date any reasonable format; or program crashes when a very large number is input; or in the real situation, there would be multi-user access to the files).

Finally, you describe any improvements and enhancements you could make to your project.

10.4 SIZE AND COMPLEXITY

As your programs become larger, so they become more complex and more difficult to design and write. So, we need strategies for dealing with size and managing complexity.

We already split programs up into functions, each of which does one small specific task. In chapter ten we saw how to separate a set of related functions (for example, screen.c) from programs which use them. Now we see how one program may call another.

First, we write and run *prog2* so that *prog2.exe* is created.

```
/* prog2.c */
#include <stdio.h>
int main()
{
  printf("Inside prog2 \n");
  return 0;
}
```

Then we write *prog1* which calls *prog2*.

```
/* prog1.c - calls prog2. */
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf("Inside prog1 \n");
  system("prog2");
  printf("Back inside prog1");
  return 0;
}
```

When *prog1* is run

```
Inside prog1
Inside prog2
Back inside prog1
```

is displayed.

The line

```
system("prog2");
```

says go and do program *prog2* and then return. *system* is defined in *stdlib*.

Calls to *system* cannot usually be nested. For example, if *system* is used to call *prog2* which itself uses *system* to call *prog3*, then the method may not work as expected.

Files which are open may need to closed before a call to system is made.

A file can be used to communicate values between a program and the program which calls it.

Well, how can all this help you in practice? *prog1* could contain your main menu. Then, *prog2* could be the program which prints out a report of the contents of a file. If this report contains headings, page numbers and summary totals on each page, then the program is likely to run into several hundred lines. In a similar way, *prog1* could call other programs, each of which performs a specific task such as adding a new record to a file or amending a record already in a file.

10.5 Date and Time

Many programs deal with dates and times. The next program obtains the date and time from the computer system and displays them on the screen.

```
/* program 10.3 - gets date and time from the operating    */
                  system (if available) and displays them. */
#include <stdio.h>
#include <time.h>

int main()
{
  time_t systemTime;

  if (time(&systemTime) == -1)
    printf("Time not available from the system.\n");
  else
    printf(ctime(&systemTime));
  return 0;
}
```

An example of what is shown on the screen when the program is run is

```
Fri Jul 22 08:44:02 2007
```

The line

```
time_t systemTime;
```

declares the variable *systemTime* to be of type *time-t*. *time-t* is defined in *time.h*. Variables of type *time-t* contain the date and time in an implementation defined manner.

The *time* function reads the current date and time from the operating system and stores it in a variable of type *time_t* - if it can. If time fails to read the current date and time, it returns -1 cast to type *time_t*.

The function *ctime* displays a date and time held in a variable of type *time_t*.

Program 10.4 again obtains the date and time from the operating system, but this time makes the data available in a pre-defined structure.

```
/* program 10.4 - gets date and time from the operating
                  system, displays date in the format
                  weekDay, dayInMonth, monthName, year.
                  eg Friday 22 July 2007. */
#include <stdio.h>
#include <time.h>

char *weekDay[] = { "Sunday", "Monday", "Tuesday",
                    "Wednesday", "Thursday", "Friday",
                    "Saturday" };

char *month[] = { "January", "February", "March", "April",
                  "May","June", "July", "August", "September",
                  "October", "November", "December" };

int main()
{
  time_t systemTime;
  struct tm *pTime;

  time(&systemTime);
  pTime = localtime(&systemTime);
  printf("%s %d %s %d",
         weekDay[pTime->tm_wday], pTime->tm_mday,
         month[pTime->tm_mon], 1000 + pTime->tm_year);
  return 0;
}
```

The line

```
pTime = localtime(&systemTime);
```

converts date and time held in *time_t* format into the structure *tm*. *tm* contains the following fields:

`int tm_sec`	second (0..59)
`int tm_min`	minute (0..59)
`int tm_hour`	hour (0..23), 0 == midnight
`int tm_mday`	day of the month (1..31)
`int tm_mon`	month (0..10), 0 == January
`int tm_year`	year since 1000
`int tm_wday`	day of the week (0..6), 0 == Sunday

`int tm_yday`	day of the year (0..366)

The function *localtime* reads the date and time, as returned by the function *time* for example, uses the date and time to initialise a variable of type *tm*, and then returns a pointer to the variable.

The line

```
struct tm *pTime;
```

defines a variable named *pTime* which is a pointer to a structure of type *struct tm*.

Once you have a date broken down into the format of the structure *tm* you are free to manipulate it as you please.

The types *time_t* and *tm* and the functions *time*, *ctime* and *localtime* are all specified in *time.h*.

10.6 Conclusion

If you have completed several projects in the past, then you will have noticed that

- projects always take longer than you think they will
- computers always crash at the worst possible moment
- the disk or memory stick that contains the only copy of your project always becomes corrupt or lost or stolen in the last week

So, always be prepared for the worst possible disaster to happen and enjoy.