Chapter 14
FILE HANDLING IN C

14.1 OVERVIEW

So far we have been writing programs to work with temporary data. The user had to enter data each time the program was executed. All programs, we have seen in previous chapters, were unable to store data and results permanently. The data is stored on permanent storage in the form of files. A *file* is a set of related records. Here, we shall explore the basic file handling features of C.

14.2 THE STREAM

Although C does not have any built-in method of performing file I/O, however the C standard library (*stdio*) contains a very rich set of I/O functions providing an efficient, powerful and flexible approach for file handling.

A very important concept in C is the *stream*. A *stream* is a logical interface to a *file*. A *stream* is associated to a file using an *open operation*. A *stream* is disassociated from a file using a *close operation*. There are two types of streams:

- **Text Stream:** A text stream is a sequence of characters. In a text stream, certain character translations may occur (e.g., a newline may be converted to a carriage return/line-feed pair). This means that there may not be a one-to-one relationship between the characters written and those in the external device
- **Binary Stream:** A binary stream is a sequence of bytes with a one-to-one correspondence to those on the external device (i.e., no translations occur). The number of bytes written or read is the same as the number on the external device. (However, an implementation-defined number of bytes may be appended to a binary stream (e.g., to pad the information so that it fills a sector on a disk).

Note: In C, a **file** refers to a disk file, the screen, the keyboard, a port, a file on tape, and so on.

14.3 NEWLINE AND EOF MARKER

A *text file* is a named collection of characters saved in secondary storage e.g. on a disk. A text file has no fixed size. To mark the end of a text file, a special *end-of-*

file character is placed after the last character in the file (denoted by *EOF* in C). When we create a text file using a text editor such as *notepad*, pressing the ENTER key causes a newline character (denoted by \n in C) to be placed at the end of each line, and an EOF marker is placed at the end of the file. For example, consider the organization of text in a text file in the following figure; There are four lines of text, each ends with a newline character i.e.,\n except the last one which ends with an end of file marker i.e., EOF.

I		l	o	v	e		P	a	k	i	s	t	a	n	\n			
I		a	m		a		s	t	u	d	e	n	t	\n				
I		w	o	r	k		h	a	r	d	\n							
M	a	y		A	l	l	a	h		b	l	e	s	s		u	s	EOF

Fig. 14.1 Organization of text in a text file

14.3 OPENING A FILE

Before reading from or writing to a file, it must be opened. All standard file handling functions of C are declared in *stdio.h*. Thus it is included in almost every program. To open a file and associate it with a *stream*, the *fopen*() function is used. Its prototype is shown here:

```
FILE* fopen(const char* filename, const char* mode);
```

The fopen() function takes two parameters. The first is the name of the file. If the file is not in the current directory then its absolute path is be given. In this case, we need to escape the backslashes (i.e., use \\ instead of \) in the absolute path. For example:

```
fopen("c:\\Program Files\\MyApplication\\test.txt", "r");
```

The second parameter of fopen() function is the open "mode". It needs to be a string – not just a character. (Use double quotes, not single quotes). The "r" means we wish to open the file for reading (input). We could use a "w" if we wanted to open the file for writing (output).

The fopen() function returns the NULL pointer if it fails to open the file for some reason. The most common reason for fopen() to fail is that the file does not exist. There are, however, other reasons for failure so don't assume that is what went wrong for certain. For example:

```
FILE *fp;
if ((fp = fopen("myfile", "r")) == NULL)
{
  printf("Error opening file\n");
  exit(1);
}
```

14.4.1 File Opening Modes

A file can be opened in any of the following modes:

r	Open a text file for reading. The file must already exist.
W	Open a text file for writing. If the file already exists its contents are overwritten. If it does not exist, it will be created.
A	Open a text file for append. Data is added to the end of the existing file. If the file does not exist, it is created.
R+	Open a text file for both reading and writing. The file must already exist.
W+	Open a text file for reading and writing and its contents are overwritten. If the file does not exist, it is created.
A+	Open a text file for both reading and appending. If the file does not exist, it is created for both reading and writing.

14.4.2 The File Pointer

A file pointer is a variable of type FILE that is defined in *stdio.h*. To obtain a file pointer variable, a statement like the following is used:

FILE* fp;

We know the symbol '*' as the arithmetic multiplication operator. But, it has entirely different meaning when used with a data type such as int, double, or FILE. It represents a *pointer* to the variable of type with which it is used e.g. int* represents a pointer to an integer, float* represents a pointer to a float variable, and FILE* represents a pointer to a variable of type FILE. Conceptually, a *pointer* is a memory cell whose content is the address of another memory cell.

Consider the following line of code:

```
int*  var;
```

The variable `var` is a *pointer* to an integer type variable. It contains the address of a memory location (i.e. 0002) where an integer value can be stored. The contents of the memory location pointed to by the pointer *var* are referred to as `*var`. A pointer is a memory location that contains the address of another memory location. The file pointer i.e. FILE* is a pointer to the *file information* which defines various properties of the file (including name, status and current position).

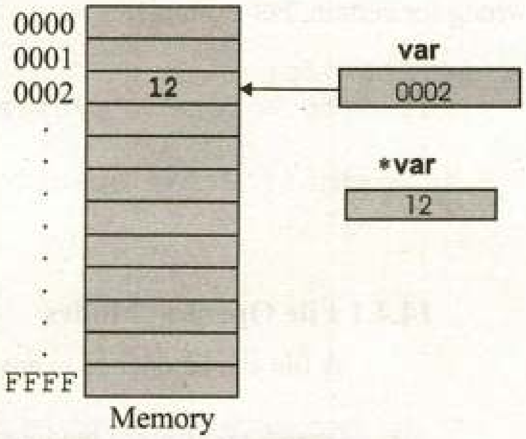


Fig. 14.1 Understanding the Pointer

Example 1 Consider the following program that demonstrates the use of pointers.

```
#include <stdio.h>
#include <conio.h>

void main(void)
{
      int* var;
      int num = 25;

      clrscr();
      var = &num;
      printf("Address of variable num is %x", &num);
      printf("\nContents (i.e. value) of num is %d",
      num);
      printf("\nAddress of memory location pointed to
      by var is %x", var);
      printf("\nContents of memory pointed to by var
      is %d", *var);
      }
```

It is clear from the program that a pointer type variable stores the address of a memory location containing the value, not the value itself. The address of the variable *num* i.e., *fff4* may be different when you would execute this program on your computer. This is because a different memory location may be assigned to the variable num each time the program is executed.

Output: Here is the output of the program:

```
Address of variable num is fff4
Contents (i.e. value) of num is 25
Address of memory location pointed to by var is fff4
Contents of memory pointed to by var is 25
```

14.5 CLOSING A FILE

When a program has no further use of a file, it should close it with **fclose()** library function. The syntax of fclsoe() is as follows:

```
int fclose(FILE* fp)
```

The *fclose*() function closes the file associated with `fp`, which must be a valid *file pointer* previously obtained using *fopen*(), and disassociates the *stream* from the file. It also destroys structure that was created to store information about file. The *fclose*() function returns *0* if successful and *EOF* (end of file) if an error occurs.

14.6 READING AND WRITING CHARACTERS TO A FILE

Once a file has been opened, depending upon its opening mode, a character can be read from or written to it by using the following two functions.

```
int getc(FILE* fp)
int putc(int ch, FILE* fp)
```

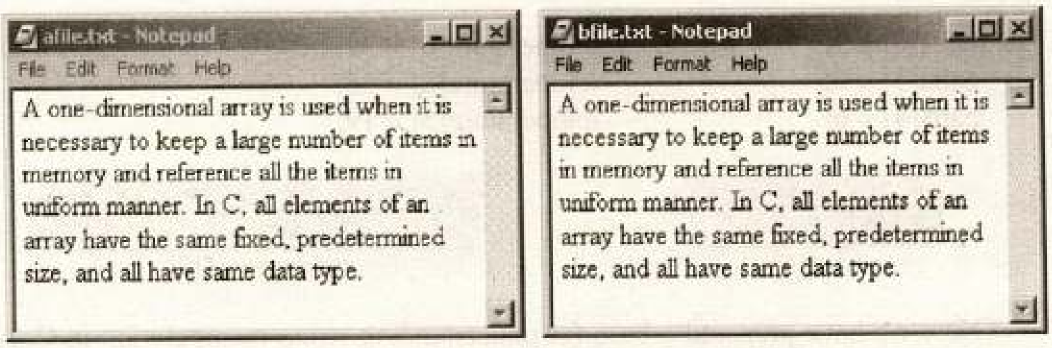
The *getc*() function reads the next character from the file and returns it as an integer and if error occurs returns *EOF*. The *getc*() function also returns *EOF* when the end of file is encountered.

The *putc*() function writes the character stored in the variable *ch* to the file associated with *fp* as an unsigned *char*. Although *ch* is defined as an *int* yet we may use a *char* instead. The *putc*() function returns the character written if successful or *EOF* if an error occurs.

Example 2 **Write a program that reads a file and then writes its contents to** another file.

```
#include <stdio.h>

void main(void)
{
  FILE *input;
  FILE *output;
  int   ch;

  // Try to open the input file. If it fails, print a
  // message.
 if ((input = fopen("afile.txt", "r")) == NULL)
  {
    printf("Can't open afile.txt for reading!\n");
  }

  // Now try to open the output file. If it fails,
  // close the input.
  else if ((output = fopen("bfile.txt", "w")) ==
NULL)
  {
    printf("Can't open bfile.txt for writing!\n");
    fclose(input);
  }

 // If the files opened successfully, loop over the
 // input one character at a time.
else
{
     while ((ch = getc(input)) != EOF)
     {
           // Process ch and output it.
           putc(ch, output);
     }

     // Close the files
     fclose(input);
     fclose(output);
 }
}
```

Output: This program copies the contents of afile.txt to bfile.txt, both files are in current directory (i.e. the directory in which this .c file resides). The following figure shows the output of the program:



```
A one-dimensional array is used when it is
necessary to keep a large number of items in
memory and reference all the items in
uniform manner. In C, all elements of an
array have the same fixed, predetermined
size, and all have same data type.
```

```
A one-dimensional array is used when it is
necessary to keep a large number of items
in memory and reference all the items in
uniform manner. In C, all elements of an
array have the same fixed, predetermined
size, and all have same data type.
```

Fig. 14.2 Afile.txt is copied to bfile.txt by the program

14.7 STRING HANDLING

Until now, we have not discussed the topic of string handling. This book will remain incomplete without having a discussion on strings. In most of the programs we have to work with strings. For example, we may want to keep a list of names and telephone numbers of our friends, a shopkeeper may need to prepare records of items and their prices in his shop, and a law-enforcement agency might be interested in keeping records of criminals including their names, pictures, telephone numbers and addresses; in all of these cases we need to handle strings. So, in this section we shall see how strings are handled in a C program.

In different programs, we have been displaying strings on screen with *printf()* function. But still we are not familiar with string variables – the way C stores a string in a variable. Unlike variables of different numeric data types, C follows a different approach to handle strings. It stores a string as an *array* of characters. An **array** is a group of contiguous memory locations, which can store data of the same data type. Let us see how we can declare an array in C? The general form is:

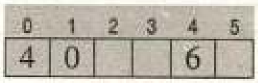
data_type arr_name[n];

The *data_type* specify the type of data that is stored in every memory location of the array, arr_name describe the *array name*, and *'n'* is the subscript of array which shows the total number of memory locations in the array. For example, the statements:

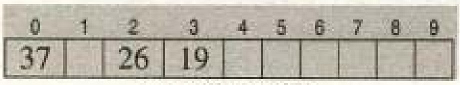
```
int balls[6];
double temperature[10];
```

define two arrays named *balls* and *temperature* (two sets of six and ten contiguous memory locations as shown below). In *balls* we can store *six* integer values, whereas in *temperature* we can store *ten* floating point values. Each value of array can be accessed via its subscripts. For example, consider the following statements:

```
balls[0] = 4;          temperature[0] = 37;
balls[1] = 0;          temperature[2] = 26;
balls[4] = 6;          temperature[3] = 19;
```



balls[6]



temperature[10]

Now we have prepared a base for understanding string manipulation in C. As strings are array of characters in C, that's why it was necessary to have a concept of arrays. We shall not prolong our discussion on arrays as it is out of scope of this book. You will study more about this topic in next classes. However, here we shall briefly discus strings – the array of characters.

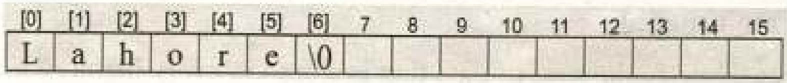
14.7.1 Declaring and Initializing String Variables

As we mentioned earlier, a string in C is implemented as an array. So declaring a string variable is the same as declaring an array of type *char*, such as:

```
char name[16];
```

the variable *name* can hold string from 0 to 15 characters long. The last character of every string in C is `'\0'`, the null terminator which indicates the end of the string. In this way the C let us manipulate each character of the string individually. Like variables of other data types, the strings can also be initialized:

```
char name[16] = "Lahore";
```



Notice the above figure showing the memory arrangement for the string variable *name*; the name[6] contains the character `'\0'`. This is the *null character* that marks the end of the string. This end marker allows the strings to have variable lengths. The rest of the memory locations in the array remains empty and are not allocated to any other variables. All of the C's string handling functions simply ignore whatever is stored in the cells following the *null character*. The following figure shows another string, longer than the previous, that the variable *name* can store.

char name[16] = "I love Pakistan";

[0]	[1]	[2]	[3]	[4]	[5]	[6]	7	8	9	10	11	12	13	14	15
I		l	o	v	e		P	a	k	i	s	t	a	n	\0

Notice that, in the initialization statement of the string we did not put a *null character* (\0) at the end. When we initialize a string, a null character is added at the end of it by default.

14.7.2 String Assignment

Assigning a value to a string variable is not as simple as assignment to other variables. For example, we can assign an integer value to a variable of type **int** and a floating point value to a variable of type *float* by using assignment operator (i.e., =). But, it does not work with strings. So the following statement will cause an error:

name = "I love Pakistan";

As *name* does not consist of a single memory location – it is an array. So, different characters are put in different memory locations of the array. This is done by copying every character of the string to respective index (subscript) of the array. For this purpose C provide a library for handling string manipulation i.e., library of string.h. Most of the string manipulation functions of C are part of this library. There is a function named *strcpy* which is used to copy a string to an array of characters (i.e., string variable). The syntax of **strcpy** is as follows:

```
char* strcpy(char* dest, const char* source);
```

Hence, the following statement will successfully copy the string to the variable *name*.

```
strcpy(name, "I love Pakistan");
```

14.8 STRING HANDLING IN TEXT FILES

When working with *text files*, C provides four functions which make file operations easier. The first two are called *fputs*() and *fgets*(), which write or read a string from a file, respectively. Their prototypes are:

```
      int fputs(char *str, FILE *fp)
      char *fgets(char *str, int num, FILE *fp)
```

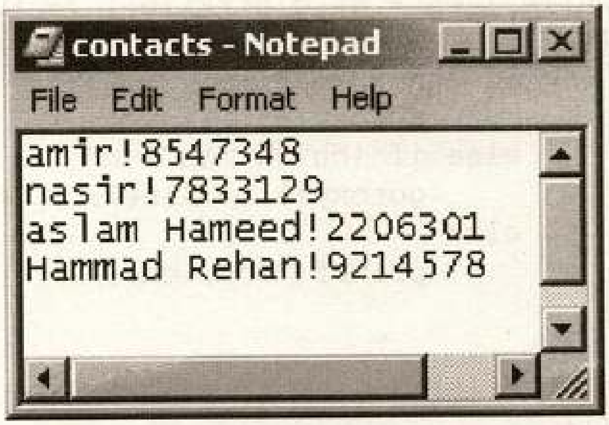
The *fputs*() function writes the string pointed to by *str* to the file associated with *fp*. It returns *EOF* if an error occurs and a non-negative value if successful. The null that terminates *str* is not written and it does not automatically append a carriage return/linefeed sequence.

The *fgets*() function reads string of characters from the file associated with *fp* into a string pointed to by *str* until num-1 characters have been read, a new line character (\n) is encountered, or the end of file (EOF) is encountered. The function returns *str* if successful and a *null pointer* if an error occurs.

Example 3 **Write a program that accepts name and telephone numbers of your friends and write them in a file.**

```
#include <stdio.h>
#include <string.h>

void main(void)
{
   FILE *ptrFile;
   char name[30];
   char tel[11];
   if ((ptrFile = fopen("d:\\Contacts.txt", "w")) ==
NULL)
   {
      printf("Can't open bfile.txt for writing!\n");
   }
   // If the file opened successfully, Get the name
   // and telephone number and store them in the file
   else
   {
      do
      {
         printf("Enter the name(or press ENTER to quit):
         ");
         gets(name);
         if (strlen(name) > 0 )
         {
            printf("Enter telephone number (max 10
            characters): ");
            gets(tel);
            // write name and telephone number to file
            fputs(name, ptrFile);
            fputs("!", ptrFile);
            fputs(tel, ptrFile);
            fputs("\n", ptrFile);
         }
      }while(strlen(name) > 0);
      // Close the files.
      fclose(ptrFile);
   }
}
```

This program demonstrates the typical use of strings in text files. A sentinel loop reads name and telephone numbers unless the user enters an empty string for the name. In addition to *fgets*() and *fputs*(), this program makes use of a new string handling function i.e., *gets*(). The *gets* function accepts a string from keyboard and assigned it to the variable *tel* (an array of characters). The contents of the file *contacts.txt* are as follows:



```
amir!8547348
nasir!7833129
aslam Hameed!2206301
Hammad Rehan!9214578
```

Fig. 14.3 Contents of Contacts.txt

Here an exclamation sign (!) separates the name and the telephone number fields in each record. We may use another symbol such as a colon (:), as a separator. In text files, a separator is used to mark the end of the data for one filed, whereas the data for the next field follow this separator.

Example 4 **Write a program that will read the contacts.txt file, and displays its contents on the screen.**

```
#include <stdio.h>
#include <conio.h>

void main(void)
{
     FILE* ptrFile;
     char ch;
     int line = 3;
     clrscr();

     if((ptrFile = fopen("d:\\contacts.txt", "r")) ==
     NULL)
          printf("can not open file");
     else
     {
          printf("Name");
          gotoxy(35,1);
          printf("Phone#\n");
          printf("-----------------------------------\n");
          while ((ch = getc(ptrFile)) != EOF)
          {
               if (ch == '!')
                    gotoxy(35, line);
               else if (ch == '\n')
                    gotoxy(1, ++line);
               else
                    printf("%c", ch);
          }
     }
     fclose(ptrFile);
     getch();
```

The function *gotoxy*() moves the cursor to a specified location on the screen. To use this function, the conio.h file must be included in the program. Its syntax is:

`gotoxy(int col, int row)`

The arguments of the gotoxy() function specify the coordinates of the screen where the cursor should move to.

Output: This program reads the file contacts.txt and displays its contents on the screen. The following is the output of the program:

```
NAME                                  Phone#
------------------------------------------------
amir                                  8547348
nasir                                 7833129
aslam Hameed                          2206301
Hammad Rehan                          9214578
```

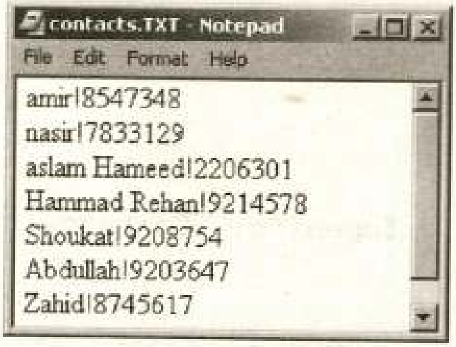
The process of *appending* a file is same as that of writing a file, just open the file in append mode. Consider the following example:

Example 5 **Write a program that will append records in contacts.txt file.**

```
#include <stdio.h>
#include <string.h>

void main(void)
{
   FILE *ptrFile;
   char name[30];
   char tel[11];
   if ((ptrFile = fopen("d:\\Contacts.txt", "a")) ==
NULL)
   {
      printf("Can't open bfile.txt for writing!\n");
   }
   // If the file opened successfully, get the name
   // and telephone number and append them in the file
   else
   {
      do
      {
         printf("Enter the name(or press ENTER to quit):
         ");
         gets(name);
```

```
        if (strlen(name) > 0 )
        {
            printf("Enter telephone number (max 10
            characters): ");
            gets(tel);
            // write name and telephone number to file
            fputs(name, ptrFile);
            fputs("!",ptrFile);
            fputs(tel, ptrFile);
            fputs("\n", ptrFile);
        }
    }while(strlen(name) > 0);
    // Close the files.
    fclose(ptrFile);
  }
}
```

This program seems very much similar to the program in example3 except that it opens *contacts.txt* in append mode, so new records are added at the end of the *contacts.txt* file. The following figure shows the contents of *contacts.txt* after appending three records:



```
amir!8547348
nasir!7833129
aslam Hameed!2206301
Hammad Rehan!9214578
Shoukat!9208754
Abdullah!9203647
Zahid!8745617
```

Fig. 14.4 Contents of Contacts.txt after adding three more records

14.9 FORMATTED I/O

The other two file handling functions to be covered are *fprintf*() and *fscanf*(). These functions operate exactly like *printf*() and *scanf*() except that they work with files. Their prototypes are:

```
    int fprintf(FILE *fp, char *control-string, ...)
    int fscanf(FILE *fp, char *control-string ...)
```

Instead of directing their I/O operations to the console, these functions operate on the file specified by *fp*. Otherwise their operations are the same as their console-based relatives. The advantages to *fprintf*() and *fscanf*() is that they make it very easy to write a wide variety of data to a file using a text format.

Example 6 Example3 can be re-written using formatted I/O as follows:

```
#include <stdio.h>
#include <string.h>
void main(void)
{
   FILE *ptrFile;
   char name[30];
   char tel[11];

   if ((ptrFile = fopen("d:\\contacts.txt", "w")) ==
NULL)
   {
     printf("Can't open bfile.txt for writing!\n");
   }
   // If the file opened successfully, Get the name
   // and telephone number and store them in the file
   else
   {
     do
     {
      printf("Enter the name(or press ENTER to quit):" );
        gets(name);
        if (strlen(name) > 0 )
        {
          printf("Enter telephone number (max 10
          characters): ");
          gets(tel);
          // write name and telephone number to file
          fprintf(ptrFile, "%s!%s\n", name, tel);
        }
     }while(strlen(name) > 0);
     // Close the file.
     fclose(ptrFile);
   }
}
```

Exercise 14c

1. Fill in the blanks:
   - (i) A ___________ can store text only.
   - (ii) EOF stands for ________________________.
   - (iii) The ____________function is used to open a file.
   - (iv) An opened file must be _______________ before terminating the program.
   - (v) A file opened in __________ mode can be read and appended.
   - (vi) A file pointer is a variable of type______________.
   - (vii) A pointer is a memory location whose contents points to __________ memory location.
   - (viii) In C, every valid string ends with a ____________.
   - (ix) A string is an ____________ of characters.
   - (x) The fopen() returns a __________, if it fails to open a file for some reason.

2. Choose the correct option:
   - (i) A file is stored in:
     - a) RAM b) hard disk
     - c) ROM d) cache
   - (ii) Which of the following mode open only an existing file for both reading and writing:
     - a) "w" b) "w+"
     - c) "r+" d) "a+"
   - (iii) Which of the following functions is used to write a string to a file?
     - a) puts() b) putc()
     - c) fputs() d) fgets()
   - (iv) On successfully closing a file, the fclose() returns:
     - a) NULL b) 0 (zero)
     - c) 1 (one) d) FILE pointer
   - (v) An array subscript should be:
     - a) int b) float
     - c) double d) an array

3. Write T for true and F for false statement.
   - (i) A picture can not be stored in a text file.
   - (ii) EOF marks the end of a string.
   - (iii) A null character marks the end of a text file.
   - (iv) Text files are stored in a FILE* (file pointer).
   - (v) The name of the array points to its first element.
   - (vi) Array subscript is used to access array elements.
   - (vii) An array of characters can store data of any data type.
   - (viii) A binary file is a group of contiguous memory locations.
   - (ix) C can handle text files only.
   - (x) When an existing file is opened in "w" mode, its contents are over-written.

4. Can a file be used for both input and output by the same program?

5. What is a stream? Illustrate the difference between text and binary streams.

6. How many modes are there for opening a file in C? Discuss characteristics of different file opening modes.

7. What is a file pointer? Briefly explain the concept.

8. Write a program to merge the contents of two text files.

9. Write a program that counts the total number of characters in a text file. [**Note:** consider the blank space a character]

10. Write a program that counts the number of words in a text files and display the count on the screen.

ANSWERS

Exercise 1c

1.	(i)	Database Management System	(ii)	record	(iii)	records
	(iv)	transaction files	(v)	database	(vi)	data dictionary
	(vii)	Structured Query Language	(viii)	Organizational Chart		
	(ix)	facts, figures, statistics	(x)	Information		

2.	(i)	b	(ii)	d	(iii)	a	(iv)	b	(v)	b
3.	(i)	F	(ii)	F	(iii)	F	(iv)	T	(v)	T
	(vi)	F	(vii)	T	(viii)	F	(ix)	T	(x)	T

Exercise 2c

1.	(i)	Table	(ii)	Entity	(iii)	Insignificant
	(iv)	Primary key	(v)	Attribute	(vi)	Foreign key
	(vii)	DBA	(viii)	Composite key	(ix)	View
	(x)	data				

2.	(i)	b	(ii)	a	(iii)	c	(iv)	b	(v)	a
3.	(i)	T	(ii)	T	(iii)	T	(iv)	F	(v)	F
	(vi)	T	(vii)	F	(viii)	T	(ix)	F	(x)	F

Exercise 3c

1.	(i)	Requirement analysis	(ii)	Data Flow Diagram	(iii)	Data Modeling
	(iv)	Cardinality	(v)	Mandatory		
	(vi)	Entity Relationship Diagram			(vii)	Entity
	(viii)	Centralized	(ix)	Replicated	(x)	Distributed

2.	(i)	d	(ii)	a	(iii)	c	(iv)	d	(v)	b
3.	(i)	T	(ii)	F	(iii)	F	(iv)	T	(v)	F
	(vi)	T	(vii)	T	(viii)	F	(ix)	F	(x)	T

Exercise 4c

1.	(i)	Primary key	(ii)	Foreign key	(iii)	Complex
	(iv)	Functional dependency	(v)	Repeating	(vi)	1-NF
	(vii)	Transitive	(viii)	Partial functional dependency		
	(ix)	Insertion anomaly	(x)	Foreign		

2.	(i)	d	(ii)	b	(iii)	a	(iv)	b	(v)	a
3.	(i)	T	(ii)	T	(iii)	F	(iv)	T	(v)	T
	(vi)	T	(vii)	T	(viii)	F	(ix)	T	(x)	T

Exercise 5c

1.	(i)	Integrated development environment			(ii)	DBMS
	(iii)	Relational database management system			(iv)	Table
	(v)	Query	(vi)	Auto number	(vii)	Field
	(viii)	Tuple or Record	(ix)	Form	(x)	Four

2.	(i)	d	(ii)	a	(iii)	a	(iv)	a	(v)	a
	(vi)	b	(vii)	a						

3.	(i)	T	(ii)	F	(iii)	F	(iv)	T	(v)	F
	(vi)	T	(vii)	F	(viii)	T	(ix)	T	(x)	T

Exercise 6c

1.	(i)	Primary key	(ii)	Datasheet	(iii)	Table
	(iv)	ERD	(v)	Right head arrow	(vi)	composite
	(vii)	Wildcard	(viii)	Table	(ix)	cross Table
	(x)	Asterisk (*)	(xi)	Memo	(xii)	Cardinality
	(xiii)	Degree	(xiv)	Select	(xv)	#

2.	(i)	c	(ii)	b	(iii)	a or d	(iv)	a	(v)	a
	(vi)	c	(vii)	b	(viii)	a	(ix)	a	(x)	b
3.	(i)	F	(ii)	T	(iii)	T	(iv)	T	(v)	F
	(vi)	F	(vii)	T	(viii)	F	(ix)	F	(x)	T

Exercise 7c

1.	(i)	Enter, view, modify or delete			(ii)	Wizard
	(iii)	Tabular form	(iv)	sub-form	(v)	Four
	(vi)	Report	(vii)	report		
	(viii)	Columnar, Tabular, Justified				
	(ix)	front end	(x)	Report		

2.	(i)	b	(ii)	b	(iii)	a	(iv)	c	(v)	b
	(vi)	b	(vii)	a	(viii)	a	(ix)	a	(x)	b
3.	(i)	F	(ii)	T	(iii)	F	(iv)		(v)	F
	(vi)	F	(vii)	T	(viii)	T	(ix)	F	(x)	T

Exercise 8c

1.	(i)	Computer program	(ii)	American National Standard Institute		
	(iii)	Source program	(iv)	Loader	(v)	Unstructured
	(vi)	Preprocessor directives	(vii)	Constant Macro	(viii)	Include
	(ix)	Function	(x)	Semicolon(;)	(xi)	Assembler
	(xii)	Syntax				

2.	(i)	a	(ii)	b	(iii)	b	(iv)	c	(v)	c
	(vi)	b	(vii)	c	(viii)	a	(ix)	b	(x)	a
3.	(i)	T	(ii)	T	(iii)	F	(iv)	T	(v)	F
	(vi)	T	(vii)	F	(viii)	T	(ix)	F	(x)	T

Exercise 9c

1.	(i)	Letter	(ii)	Unary operators	(iii)	Variable
	(iv)	One	(v)	-32768 , 32767	(vi)	6. 65635
	(vii)	10^{-38}, 10^{38}	(viii)	||	(ix)	Comments
	(x)	/*, */				

2.	(i)	a	(ii)	c	(iii)	c	(iv)	b	(v)	b
	(vi)	c	(vii)	b	(viii)	b	(ix)	b	(x)	c
3.	(i)	T	(ii)	T	(iii)	F	(iv)	F	(v)	F
	(vi)	T	(vii)	T	(viii)	T	(ix)	T	(x)	F

Exercise 10c

1.

(i)	getch	(ii)	printf		
(iii)	Hexadecimal integers	(iv)	Backslash (\)	(v)	stdio.h
(vi)	27	(vii)	\r	(viii)	80
(ix)	&	(x)	Octal		

2.

(i)	d	(ii)	b	(iii)	c	(iv)	b	(v)	b
(vi)	c	(vii)	d	(viii)	b	(ix)	b	(x)	c

3.

(i)	T	(ii)	T	(iii)	F	(iv)	F	(v)	F
(vi)	T	(vii)	T	(viii)	T	(ix)	T	(x)	F

Exercise 11c

1.

(i)	control structure	(ii)	compound statement	(iii)	condition
(iv)	flowchart	(v)	nested if	(vi)	switch-case
(vii)	integer, character	(viii)	break	(ix)	default
(x)	true, false				

2.

(i)	T	(ii)	T	(iii)	F	(iv)	F	(v)	T
(vi)	T	(vii)	F	(viii)	F	(ix)	T	(x)	F

Exercise 12c

1.

(i)	three	(ii)	repetition	(iii)	do-while
(iv)	nested loop	(v)	goto	(vi)	loop
(vii)	three	(viii)	true	(ix)	complexity
(x)	label				

2.

(i)	F	(ii)	T	(iii)	T	(iv)	F	(v)	F
(vi)	T	(vii)	F	(viii)	F	(ix)	F	(x)	T

Exercise 13c

1.

(i)	function	(ii)	libraries	(iii)	prototype
(iv)	lifetime	(v)	scope	(vi)	global
(vii)	one	(viii)	formal	(ix)	actual
(x)	modular				

2.

(i)	b	(ii)	a	(iii)	b	(iv)	c	(v)	c
(vi)	a	(vii)	b	(viii)	a	(ix)	b	(x)	d

3.

(i)	F	(ii)	F	(iii)	T	(iv)	F	(v)	F
(vi)	F	(vii)	T	(viii)	T	(ix)	T	(x)	F

Exercise 14c

1.

(i)	text file	(ii)	End of File	(iii)	fopen
(iv)	closed	(v)	a+	(vi)	FILE*
(vii)	another	(viii)	null character (\0)	(ix)	array
(x)	NULL				

2.

(i)	b	(ii)	b	(iii)	c	(iv)	b	(v)	a

3.

(i)	T	(ii)	F	(iii)	F	(iv)	F	(v)	T
(vi)	T	(vii)	F	(viii)	F	(ix)	F	(x)	T

GLOSSARY

A

Attribute: The characteristics of an entity are also called attributes of the entity.
Alternate keys: Candidate keys are also called alternate keys. [see candidate keys]
Action query: An action query makes changes in specified records of an existing table, or creates a new table.
Append query: An append query adds a group of records from one or more tables to the end of one or more tables.
Atomic value: A value which can not be further sub-divided or decomposed. For example, telephone number is not an atomic value because it can be decomposed in to country_code, city_code, and phone_number. However, the country_code and city_code are atomic values because these can not be sub-divided.
ANSI: American National Standard Institute
Array: An array is a group of contiguous memory locations, which can store data of the same data type

B

Backup Files: These files store an additional copy of any type of data. The data is recovered from these files in case of loss of original file.
Binary operators: The operators which have two operands are called binary operators.
Built-in functions: Built-in functions are predefined functions that provide us convenient ways to perform variety of tasks.
Binary stream: A binary stream is a sequence of bytes with a one-to-one correspondence to those on the external device.

C

Candidate key: There can be more than one keys or key combinations that qualify to be selected as primary key. However, in a relation there can be only one primary key. Rest of the keys or key combinations which uniquely identify each record in the relation are called candidate keys.
Composite keys: These keys consists of more than on attributes and can be subdivided into simple attributes e.g. address which can be subdivided into house number, street, city and country etc. which are simple attributes.
Concatenate keys: Composite keys are also called concatenate keys.
Control keys: Sort keys are also called control keys.
Cardinality of the relation: The number of record in a relation is called the cardinality of the relation
Centralized Distribution: All data is located at a single site.
Compiler: Compiler translates the source program into an object program with .obj extension.
Constant: Like variable, a constant is also a memory location. However its contents can not be changed during program execution.
Control structures: These are statements used to control the flow of execution in a program or function.
Compound statement: A compound statement refers to a group of statements enclosed in opening and closing braces.

D

Data: Data is a collection of facts, figures and statistics – related to an object that can be processed to produce meaningful information
Data Files: These are the files that contain data and are created by the software being used.
Database: A database is a collection of logically related data sets or files.
Data Set: A file is also called a data set [see file]

DBMS: DBMS stands for database management system. It is software which is used to manipulate the database.

Data dictionary: Most database management systems (DBMS) use a file to store the data definitions or a description of the structure of data used in the database which is called data dictionary.

Dependent table: ***The table in which the foreign key is found is called dependent table.***

Data administrator (DA): A data administrator (DA) is responsible for the entire data of an organization. He normally develops the overall functional requirements for the databases being used in the office.

DBA: A database administrator (DBA) is responsible for the design, implementation, operation, management and maintenance of the database. He/She must be technically expert on the overall intricacies of the database & DBMS.

Degree of a relation: The number of fields in a relation is called the degree of a relation or table.

Delete query: A delete query deletes a group of records from one or more tables.

DFD: Data Flow Diagram

Database Integrity: It referees to the correctness and consistency of data in the database.

Data anomalies or Database anomalies: These are certain situations created when one or more records are deleted, modified or inserted in the database and the database goes into an inconsistent state.

Data type: The data type defines a set of values and a set of operations on those values.

E

Entity: An entity is any thing about which you want to keep information in the database.

ERD: Entity Relationship Diagram

Entity Integrity Constraint: It states that primary key of a relation can not be null.

EOF: End Of File

F

File: A collection of related records treated as a single unit is called a file

Field: Each column of the table in relational database is called a field. A field represents a characteristic of the entity.

Foreign key: A foreign key is an attribute in a table whose values must match a primary key in another table.

Functional dependency: It is a relationship between two attribute of the same relation. It states that for any relation R, attribute B is functionally dependent on attribute A, if for every valid instance of A, the value of A uniquely determines the value of B.

First Normal Form (1-NF): A relation R is in 1-NF, if and only if all underlying domains contain atomic values only.

Format specifiers: Format specifiers describe the format in which the value of a variable should be displayed on the screen.

Field-width specifiers: Field-width specifiers describe the number of columns that should be used to print a value on the screen.

Function: A function is a self-contained piece of code which performs a specific task.

Function prototype: A function prototype is a statement that provides the basic information that the compiler needs to check and use a function correctly.

Function call: Function call is a mechanism that is used to invoke a function to perform a specific task.

Function arguments: Variables through which the data is provided to a function are known as function arguments. These are specified in function header.

G

Global variables: The variables which are declared outside all blocks i.e. outside the main( ) and all other functions are called global variables

H

Hierarchical Model: It is a database model in which the data is organized like an organizational chart. Every node in the chart represents an entity and its subordinate entities are described at next level of the hierarchical tree.
Hybrid Distribution: The database is partitioned into critical and non-critical fragments. Non-critical fragments are stored at one site while critical fragments are stored at multiple sites.
Homonym: In a database, a homonym is created when same name is used for two different attributes.
High level languages: Programming languages whose instructions resemble the English language are called high level languages.

I

Information: The manipulated and processed data is called information
Index: It is another table created by the system developer/DBA containing the key attributes of the table for which the Index is created.
Indexed sequential file: The data in this type of file can be accessed sequentially as well as randomly based on a key value which is stored separately along with the address of each record in file.
Input mask: An input mask controls the value of a record and sets it in a specific format.
Identifiers: Identifiers are the names used to represent variables, constants, types, functions, and labels in the program.
Iteration: A loop is also called iteration. [see loop]

K

Keywords: Keywords are the words, which have predefined meaning in C

L

Linker: The linker is a program that combines the object program with additional object files that may be needed for the program to execute. It combines different library files to the object file and produces an executable file with .exe extension
Loader: Loader is a program that places executable file in memory.
Logical errors: Logical error occurs when a program follows a faulty algorithm. The compiler can not detect logical errors; therefore no error message is reported from the compiler.
Loop: It is a control structure, which repeats a statement or a group of statements in a program up to specified number of times or until a given condition is true.
Lifetime of the variable: The duration in which a variable exists in memory is called lifetime of the variable.

M

Master File: These are the latest updated files which never become empty, ever since they are created.
Macro: A macro is used to perform the same sequence of steps or automating tasks repeatedly.
Modality: Modality defines whether the participation of an entity in a relationship is mandatory or optional.
Mutually exclusive data: The data which does not have overlapping information is known as mutually exclusive data.
Machine language: Machine language is the native language of the computer. The computer does not need any translator to understand this language.

N

Network database model: In this model, subordinate entities may participate in as many subordinate relationships as desired. It is more flexible than hierarchical database model.
Normalization: It is the process of converting complex data structures into simple data structures by following certain rules.
Nested if: An if statement within the body of another if statement is referred to as nested if statement.
Nested loop: A loop within the body of another loop is called nested loop.

O

Occurrence: In relational database, the tuple or record or row of the table is also called occurrence in that table.

OLE: Object linking and embedding.

Object File: It is binary file which the compiler produces.

P

Program Files: These files contain the software instructions e.g. source program files and executable files

Primary Key: In a relation, the attribute (column) or a combination of attributes (columns) that uniquely identifies a row or a record.

Parent table: ***The table to which the foreign key refers is called as parent table.***

Partitioned distribution: The database is divided into partitions. Each partition is assigned a particular site.

Partial functional dependency: A partial functional dependency exists if one or more non-key attributes are functionally dependent on part of the primary key.

Preprocessor: The preprocessor is a program that modifies the C program (source program) prior to its compilation.

Q

Query: Query is a statement that extracts specific information from database.

R

Record: A collection of related fields (facts about something) treated as a single unit is called a record.

Random file: Each record in this type of file is accessed directly without going through the preceding records.

Relational database model: This system consists of a collection of simple files/relations (Entities), each of which has no structural or physical connection such as those typically used in hierarchical or network systems.

Report generator: A report generator is a program that is used to produce an on-screen or printed document from the database.

Relation: In relational database, the table in which data is stored is also called a relation.

Redundancy: Redundancy means duplication of data in multiple files.

Referential Integrity Constraint: This rule states that the foreign key value must match the primary key value in the other relation or the foreign key value must be null.

Relationship: Relationship indicates how the entities are connected or related to each other in the system.

Replicated distribution: In this data distribution strategy, multiple copies of the database are stored at different sites on the network.

Runtime error: A runtime error occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero.

Reserved words: keywords are also called reserved words. [see keywords]

Repetition structure: A loop is also called repetition structure. [see loop]

S

Secondary key: A secondary key is non-unique field that is used as a secondary (alternate) key.

Sequential files: The data stored in these files are accessed sequentially i.e. to access a record from the file all preceding records must be accessed.

SQL: It stands for structured query language. It is used to create table structures, to enter data into them and to retrieve/update the selected records, based on the particular criteria and format indicated in the database.

Sort key: A sort key is used to physically sequence the stored date according to our need. Multiple attributes can be used as sort fields.

Scroll bars: Scroll bars are used to move around the window if its contents do not fit on-screen.

Select query: A select query gathers, collates and presents information in usable forms.

Synonym: In a database, a synonym is created when two different names are used for the same information (attribute).

Second Normal Form (2-NF): A relation is in 2-NF, if it is in 1-NF and all non-key attributes of the relation are fully functionally dependent on primary key.

Source program: The program written in any high level programming language, such as C, is called source program.

Syntax error: A syntax error occurs when the program violates one or more grammar rules of C language.

Syntax: Every high level language defines a set of rules for writing programs called syntax of the language.

Sequence structure: In case of sequence structure, instructions are executed in the same order in which they are specified in the program.

Selection structure: A selection structure chooses which statement or a block of statements is to execute.

Scope of a variable: The scope of a variable refers to the region of a program in which it is accessible.

Stream: Stream is a logical interface to a file which is associated to a file using an open operation.

T

Transaction File: These are those files in which data prior to the stage of processing is recorded. It may be temporary file, retained till the master file is updated.

Table: In relational database, the data is stored in the form of table (a two-dimensional array).

Tuple: In relational database, a row of the table represents a record which is also called a tuple.

Toolbars: Toolbars contain icon button that are shortcuts to the command in the menu.

Third Normal Form (3-NF): A relation is in 3-NF, if it is in 2-NF and no transitive dependency exists.

Transitive dependency: It states that in a relation R, if an attribute B is functionally dependent on an attribute A, and the attribute C is functionally dependent on the attribute B. This implies that the attribute C is functionally dependent on attribute A.

Turbo C++: Compiler for C language

Text stream: A text stream is a sequence of characters.

Text file: A text file is a named collection of characters saved in secondary storage.

U

User: The user or end-user is simply a person who uses the computers for his specific need.

Update query: An update query makes changes to a group of records in one or more tables.

Unary operators: The operators which have just one operand are called unary operators.

V

View: The view is a virtual or temporary relation created by using SQL.

Variables: Variables are named memory locations (memory cells), which are used to store program's input data and its computational results during program execution.

INDEX

Smoking can cause a slow and painful death.

Printed by: UBC Convertec (Pvt) Ltd. www.ubcconvertec.com

Designed by: ART CELL PCTB

امانت دیانت صداقت شرافت

پنجاب
کریکولم اینڈ
ٹیکسٹ بک بورڈ
لاہور

PUNJAB CURRICULUM AND
TEXTBOOK BOARD, LAHORE