Session

20 Advanced Data Types & Sorting (Lab)

Objectives

At the end of this session, you will be able to:

- *Use structures and structure arrays*
- *Pass structures to functions*
- *Sort arrays*

The steps given in this session are detailed comprehensive and carefully thought through. This has been done so that the learning objectives are met and the understanding of the tool is complete. Please follow the steps carefully.

Part I – For the first 1 Hour and 30 Minutes:

20.1 Structures

A structure is a grouping of a number of data items, which may be different data types. Every structure has to be defined before it is declared. A structure definition can contain another structure. Initialization of structures is similar to that of arrays.

20.1.1 Structure arrays and Sorting

In C, it is possible to create a structure array. As with arrays, data in structure arrays can be sorted using Selection sort or Bubble Sort. Let us write a C program to implement a basic library management system. The system maintains a book catalog and a register for books issue and receipt transactions. Using the system, it is possible to add book details, record issue/receipt transactions and sort the issue/receipt register. The steps for creating the system are listed below:

1. Define a structure to store the book details.

The code will be,

```
struct book_st {
  int book_cd;
  char book_nm[30];
  char author[30];
  int copies;
};
```

2. Define a structure to store the issue/receipt register. Note that the issue/receipt date will also be a structure, and will have to be defined too.

The code will be,

```
struct date_st {
  int month;
  int day;
  int year;
};
struct tran_st {
  int book_code;
  char tran_type;
  struct date_st tran_dt;
};
```

3. Declare variables of the two structure types. For practical purposes, let us assume that details of 5 books and 10 transactions will be recorded.

The code will be,

```
struct book_st books[5];
struct tran_st trans[10];
```

4. Set up a loop to display a menu for the possible operations.

The code for the same will be,

```
while(choice!=4)
{
  clrscr();
  printf("\nSelect from Menu\n1. Add book names\n2. Record Issue/Return\
n3. Sort Transactions\n4. Exit\n\nEnter choice: ");
  scanf("%d", &choice);
  :
  :
}
```

5. If the selected operation is adding book details, accept the details within a loop.

The code will be,

```
for(i=0; i<5 && addflag=='y'; i++)
{
    books[i].book_cd = i+1;
    printf("\n\nBook code: %d\n\nBook name: ", i+1);
    scanf("%s",books[i].book_nm);
    printf("\nAuthor: ");
    scanf("%s",books[i].author);
    printf("\nNumber of copies: ");
    scanf("%d", &books[i].copies);
    printf("\n\nContinue? (y/n): ");
    scanf("%c", &addflag);
}
```

6. If the selected operation is adding transactions, set up a loop to accept the details.

The code will be,

```
for(i=0; i<10 && addflag=='y'; i++)
{
   printf("\n\nBook code:");
   scanf("%d", &trans[i].book_code);
   printf("\nIssue or Return?(I/R):");
   scanf("%c", &trans[i].tran_type);
   printf("\nDate:");
   scanf("%d %d %d", &trans[i].tran_dt.month, &trans[i].tran_dt.day,
&trans[i].tran_dt.year);
   printf("\n\nContinue? (y/n):");
   scanf("%c", &addflag);
}
```

7. If the selected operation is sorting transactions, pass the structure array to a function. The function should sort the array on book codes using selection sort method.

The code will be,

```
for(i=0; i<10; i++)
   for(j=i+1;j<10;j++)
   {
      if(tran[i].book_code > tran[j].book_code)
      {
         temptran=tran[i];
         tran[i]=tran[j];
         tran[j]=temptran;
      }
   }
```

8. Display the number of transactions for every book in the sort function.

The code will be,

```
for(i=0, j=0; i<10; j=0)
{
    tempcode=tran[i].book_code;
    while(tran[i].book_code==tempcode && i<10)
    {
       j++;
       i++;
    }
    printf("\nBook code %d had %d transactions", tempcode, j);
}
```

Let us look at the complete program.

1. Invoke the editor in which you can type the C program.

2. Create a new file.

3. Type the following code:

```
#include<stdio.h>
struct book_st
{
    int book_cd;
    char book_nm[30];
    char author[30];
    int copies;
};
struct date_st
{
    int month;
    int day;
    int year;
};
struct tran_st
{
    int book_code;
    char tran_type;
    struct date_st tran_dt;
};
void main()
{
    int choice=1, i;
    char addflag;
    struct book_st books[5];
    struct tran_st trans[10];
    while(choice!=4)
    {
        clrscr();
        printf("\nSelect from Menu\n 1. Add book names\n 2. Record
Issue/Return\n 3. Sort Transactions\n 4. Exit\n\n Enter choice: ");
        scanf("%d", &choice);
        if(choice==1)
        {
            addflag='y';
            clrscr();
            for(i=0; i<5 && addflag=='y'; i++)
            {
                books[i].book_cd=i+1;
                printf(”\n\nBook code: %d\n\nBook name: ”, i+1);
                scanf(”%s”,books[i].book_nm);
                printf(”\nAuthor: ”);
                scanf(”%s”,books[i].author);
                printf(”\nNumber of copies: ”);
                scanf(”%d”, &books[i].copies);
                printf(”\n\nContinue? (y/n): ”);
                scanf(“%c”, &addflag);
        }
    }
    else if(choice==2)
    {
        addflag=’y’;
        clrscr();
        for(i = 0; i<10 && addflag == ‘y’; i++)
        {
            printf(“\n\nBook code:”);
            scanf(“%d”, &trans[i].book_code);
            printf(“\nIssue or Return?(I/R):”);
            scanf(“%c”, &trans[i].tran_type);
            printf(“\nDate:”);
            scanf(“%d %d %d”, &trans[i].tran_dt.month,&trans[i].tran_
dt.day, &trans[i].tran_dt.year);
            printf(“\n\nContinue? (y/n):”);
            scanf(“%c”, &addflag);
        }
    }
    else if(choice==3)
    {
        sorttran(trans);}
    }
}
sorttran(struct tran_st tran[10])
{
    int i, j, tempcode;
    struct tran_st temptran;
    clrscr();
    for(i=0;i<10;i++)
        for(j = i+1; j < 10; j++)
    {
        if(tran[i].book_code>tran[j].book_code)
        {
            temptran=tran[i];
            tran[i]=tran[j];
            tran[j]=temptran;
        }
    }
    for(i=0, j=0;i<10;j=0)
    {
        tempcode=tran[i].book_code;
        while(tran[i].book_code==tempcode && i<10)
        {
            j++;
            i++;
        }
        printf("\nBook code %d had %d transactions", tempcode, j);
    }
    getch();
}
```

To see the output, follow these steps:

4. **Save the file with the name `structI.C`.**

5. **Compile the file, `structI.C`.**

6. **Execute the program, `structI.C`.**

7. **Return to the editor.**

The sample output of the program is given below.

```
Select from Menu
1. Add book names
2. Record Issue/Return
3. Sort Transactions
4. Exit
Enter choice:
```

If 1 is entered, the sample output of the program will be as shown below.

```
Book code: 1
Book name: Detective
Author: Hailey
Number of copies: 3
Continue? (y/n): y
```

If 2 is entered, the sample output of the program will be as shown below.

```
Book code: 1
Issue or Return? (I/R): I
Date: 2 22 03
Continue? (y/n): y
```

If 3 is entered, the sample output of the program will be as shown below.

```
Book code 1 had 3 transactions
Book code 2 had 1 transactions
Book code 3 had 2 transactions
Book code 4 had 0 transactions
Book code 5 had 4 transactions
```

Part II – For the next 30 Minutes:

1. Write a C Program to store student data in a structure. The data should include student ID, name, course registered for, and year of joining. Write a function to display the details of students enrolled in a specified academic year. Write another function to locate and display the details of a student based on a specified student ID.

   To do this,

   a. Define the structure to store the student details.

   b. Declare and initialize the structure with details of 10 students.

   c. Set a loop to display a menu for the operations to be performed.

   d. Accept the menu choice and invoke appropriate functions with the structure array as parameter.

   e. In the function to display students for a year, accept the year. Set a loop to check each student's enrollment year, and display if it matches. At the end, allow the user to specify another year.

   f. In the function to locate student details, accept the student ID. Set a loop to check each student's ID, and display if it matches. At the end, allow the user to specify another student ID.

Try It Yourself

1. Write a C program to store 5 lengths in a structure array. The lengths should be in the form of yards, feet and inches. Sort and display the lengths.

2. Write a C program to store employee details in a structure array. The data should include employee ID, name, salary, and date of joining. The date of joining should be stored in a structure. The program should perform the following operations based on a menu selection:

   a. Increase the salaries according to the following rules:

Salary Range	Percentage increase
<= 2000	15 %
> 2000 and <= 5000	10 %
>5000	No increase

   b. Display the details of employees who complete 10 years with the company.

Session

21 File Handling

Objectives

At the end of this session, you will be able to:

- *Explain streams and files*
- *Discuss text streams and binary streams*
- *Explain the various file functions*
- *Explain file pointer*
- *Discuss current active pointer*
- *Explain command-line arguments*

Introduction

Most programs need to read and write data to disk-based storage systems. Word processors need to store text files, spreadsheets need to store the contents of cells, and databases need to store records. This session explores the facilities in C for input and output (I/O) to a disk system.

The C language does not contain any explicit I/O statements. All I/O operations are carried out using functions from the standard C library. This approach makes the C file system very powerful and flexible. I/O in C is unique because data may be transferred in its internal binary representation or in a human-readable text format. This makes it easy to create files to fit any need.

It is important to understand the difference between files and streams. The C I/O system provides an interface to the user, which is independent of the actual device being accessed. This interface is not actually a file but an abstract representation of the device. This abstract interface is known as a stream and the actual device is called the file.

21.1 File Streams

The C file system works with a wide variety of devices including printers, disk drives, tape drives and terminals. Even though all these devices are very different from each other, the buffered file system transforms each device into a logical device called a stream. Since all streams act similarly, it is easy to handle the different devices. There are two types of streams-the text and binary streams.

21.1.1 Text Streams

A text stream is a sequence of characters. The text streams can be organized into lines terminated by a new line character. However, the new line character is optional on the last line and is determined by the implementation. Most C compilers do not terminate text streams with new line characters. In a text stream, certain character translations may occur as required by the environment. For example, a new line may be converted to a carriage return/linefeed pair. Therefore, there may not be a one-to-one relationship between the characters that are written (or read) and those in the external device. Also, because of possible translations, the number of characters written (or read) may not be the same as those in the external device.

21.1.2 Binary Streams

A binary stream is a sequence of bytes with a one-to-one correspondence to those in the external device, that is, there are no character translations. Also, the number of bytes written (or read) is the same as the number on the external device. Binary streams are a flat sequence of bytes, which do not have any flags to indicate the end of file or end of record. The end of file is determined by the size of the file.

21.2 File functions and FILE Structure

A file can refer to anything from a disk file to a terminal or a printer. However, all files do not have the same capabilities. For example, a disk file can support random access while a keyboard cannot. A file is associated with a stream by performing an open operation. Similarly, it is disassociated from a stream by a close operation. When a program terminates normally, all files are automatically closed. However, when a program crashes, the files remain open.

21.2.1 Basic File Functions

The ANSI file system is composed of several interrelated functions. The most common ones are listed in Table 21.1.

Name	Function
`fopen()`	Opens a file
`fclose()`	Closes a file
`fputc()`	Writes a character to a file
`fgetc()`	Reads a character from a file
`fread()`	Reads from a file to a buffer
`fwrite()`	Writes from a buffer to a file
`fseek()`	Seeks a specific location in the file
`fprintf()`	Operates like printf(), but on a file
`fscanf()`	Operates like scanf(), but on a file
`feof()`	Returns true if end-of-file is reached

Name	Function
`ferror()`	Returns true if an error has occurred
`rewind()`	Resets the file position locator to the beginning of the file
`remove()`	Erases a file
`fflush()`	Writes data from internal buffers to a specified file

Table 21.1: Basic File Functions

The above functions are contained in the header file stdio.h. This header file must be included in a program that makes use of the functions. Most of the functions are similar to the console I/O functions. The stdio.h header file also defines several macros useful for file processing. For example, the EOF macro defined as -1, contains the value returned when a function tries to read past the end of the file.

21.2.2 File Pointer

A file pointer is essential for reading or writing files. It is a pointer to a structure that contains information about the file. The information includes the file name, current position of the file, whether the file is being read or written, and whether any errors or the end of the file have occurred. The user does not need to know the details, because the definitions obtained from stdio.h include a structure declaration called `FILE`. The only declaration needed for a file pointer is symbolized by:

```
FILE *fp;
```

This denotes that fp is a pointer to a `FILE`.

21.3 Text Files

There are various functions to handle text files. They are discussed below:

21.3.1 Opening a Text File

The `fopen()` function opens a stream for use and links a file with that stream. The file pointer associated with the file is returned by the `fopen()` function. In most cases, the file being opened is a disk file. The prototype for the `fopen()` function is:

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

where `filename` is a pointer to a string of characters that make up a valid file name and can also include a path specification.

The string pointed to by mode determines how the file is to be opened. Table 21.2 shows the valid modes in which a file may be opened.

Mode	Meaning
r	Open a text file for reading
w	Create a text file for writing
a	Append to a text file
r+	Open a text file for read/write
w +	Create a text file for read/write
a+f	Append or create a text file for read/write

Table 21.2: File Opening Modes for Text Files

As can be seen from Table 21.2, the files can be opened in the text or the binary mode. A null pointer is returned if an error occurs when the `fopen()` function is opening a file. Note that strings like “`a+f`” can also be represented as “`af+`”.

If a file xyz were to be opened for writing, the code for it would be:

```
FILE *fp;
fp = fopen (“xyz”, “w”);
```

However, a file is generally opened using the set of statements similar to the following:

```
FILE *fp;
if ((fp = fopen (“xyz”, “w”)) == NULL)
{
    printf(“Cannot open file”);
    exit (1);
}
```

The macro NULL is defined in stdio.h as ‘`\0`’. If a file is opened using the above method, `fopen()` detects any error in opening a file, such as a write-protected or a full disk, before attempting to write to it. A null is used to indicate failure because no file pointer will ever have that value.

If a file is opened for writing, any file with the same name and is already open will be overwritten. This is because when a file is opened in a write mode, a new file is created. If records have to be added to an existing file, it should be opened with the mode “`a`”. If a file is opened in the read mode and it does not exist, an error is returned. If a file is opened for read/ write operations, it will not be erased if it exists. However, if it does not exist, it will be created.

According to the ANSI standard, eight files can be opened at any one time. However, most C compilers and environments allow more than eight files to be opened.

21.3.2 Closing a Text File

Since there is a limit on the number of files that can be opened at one time, it is important to close a file once it has been used. This frees system resources and reduces the risk of overshooting the set limit. Closing a stream also flushes out any associated buffer (an important operation that prevents loss of data) when writing to a disk. The `fclose()` function closes a stream that was opened by a call to `fopen()`. It writes any data still remaining in the disk buffer to the file. The prototype for `fclose()` is:

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

where `fp` is the file pointer.

The function `fclose()` returns an integer value 0 for successful closure. Any other return value would indicate an error. The `fclose()` function will fail if a disk has been prematurely removed from the drive or there is no more space on the disk.

The other function used for closing streams is the `fcloseall()` function. This function is useful when many open streams have to be closed at the same time. It closes all open streams and returns the number of streams closed or EOF if any error is detected. It can be used in the following manner:

```
fcl = fcloseall();
if (fcl == EOF)
    printf("Error closing files");
else
    printf("%d file(s) closed ", fcl);
```

21.3.3 Writing a Character

Streams can be written to a file either character by character or as strings. Let us first discuss writing characters to a file. The `fputc()` function is used for writing characters to a file previously opened by `fopen()`. The prototype for this is:

```
int fputc(int ch, FILE *fp);
```

where `fp` is the file pointer returned by `fopen()` and `ch` is the character to be written.

Even though `ch` is declared as an `int`, it is converted by `fputc()` into an unsigned char. The `fputc()` function writes a character to a specified stream at the current file position and then advances the file position indicator. If `fputc()` is successful it returns the character written, otherwise it returns EOF.

21.3.4 Reading a Character

The `fgetc()` function is used for reading characters from a file opened in read mode, using `fopen()`. The prototype for `fgetc()` is:

```
int fgetc (FILE *fp);
```

where `fp` is a file pointer of type `FILE`, returned by `fopen()`.

The `fgetc()` function returns the next character from the current position in the input stream, and increments the file position indicator. The character read is an unsigned char and is converted to an integer. If the end of file is reached, `fgetc()` returns `EOF`.

To read a text file until the end of file is encountered, the code will be:

```
do
{
    ch = fgetc(fp);
} while (ch != EOF);
```

The following program uses the functions discussed till now. It takes in characters from the keyboard and writes them to a file till the character '`@`' is entered by the user. After the user has entered the information, the program displays the file contents on the screen.

Example 1:

```
#include <stdio.h>
main()
{
    FILE *fp;
    char ch= ' ';
    /* Writing to file JAK */
    if ((fp=fopen("jak", "w"))==NULL)
    {
        printf("Cannot open file \n\n");
        exit(1);
    }
    clrscr();
    printf("Enter characters (type @ to terminate): \n");
    ch = getche();
    while (ch !='@')
    {
        fputc(ch, fp) ;
        ch = getche();
    }
    fclose(fp);
    /* Reading from file JAK */
    printf("\n\nDisplaying contents of file JAK\n\n");
    if((fp=fopen("jak", "r"))==NULL)
    {
        printf("Cannot open file\n\n");
        exit(1);
    }
    do
    {
        ch = fgetc (fp);
        putchar(ch) ;
    } while (ch!=EOF);
    getch();
    fclose(fp);
}
```

A sample run for the above will be:

```
Enter Characters (type @ to terminate):
This is the first input to the File JAK@
Displaying Contents of File JAK
This is the first input to the File JAK
```

21.3.5 String I/O

In addition to `fgetc()` and `fputc()`, C supports the related functions `fputs()` and `fgets()`. These functions write and read character strings to and from a disk file.

The prototypes for these functions are as follows:

```
int fputs(const char *str, FILE *fp);
char *fgets( char *str, int length, FILE *fp);
```

The `fputs()` function works like `fputc()`, except that it writes the entire string to the specified stream.

It returns EOF if an error occurs.

The `fgets()` function reads a string from the specified stream until either a new line character is read or length-1 characters have been read. If a new line is read, it is considered as a part of the string (unlike `gets()`). The resultant string will be terminated by the null character. The function returns a pointer to the string if successful and a null pointer if an error occurs.

21.4 Binary Files

The functions used to handle binary files are same as the ones used to handle text files. However the opening modes of the `fopen()` functions are different in the case of binary files.

21.4.1 Opening a binary file

The following table lists the various modes for the `fopen()` function in case of binary files.

Mode	Meaning
`rb`	Open a binary file for reading
`w b`	Create a binary file for writing
`ab`	Append to a binary file
`r+b`	Open a binary file for read/write
`w + b`	Create a binary file for read/write
`a+b`	Append a binary file for read/write

Table 21.3: File Opening Modes for Binary Files

If a file **`xyz`** were to be opened for writing, the code for it would be:

```
FILE *fp;
fp = fopen (“xyz”, “wb”);
```

21.4.2 Closing a binary file

The `fclose()` function can also be used to close a binary files in addition to text files. An example of `fclose()` is:

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

where `fp` is a file pointer that points to an open file.

21.4.3 Writing a binary file

Some applications involve the use of data files to store blocks of data, where each block consists of contiguous bytes. Each block will generally represent a complex data structure or an array.

For example, a data file may consist of multiple structures having the same composition, or it may contain multiple arrays of the same type and size. For such applications it may be desirable to read the entire block from the data file or write the entire block to the data file rather than reading or writing the individual components (i.e. structure members or array elements) within each block separately.

The `fwrite()` function is used to write data to data under such circumstances. This function can be used to write any type of data. The prototype for the `fwrite()` function is:

```
size_t fwrite(const void *buffer, size_t num_bytes, size_t count, FILE *fp);
```

The data type `size_t` is an ANSI C addition to improve portability. It is predefined as an integer type large enough to hold `sizeof()` results. For most systems it can be taken as an unsigned integer.

The buffer is a pointer to the information that will be written to the file. The number of bytes to be read or written is specified by **`num_bytes`**. The argument count determines how many items (each **`num_bytes`** in length) are read or written. Finally, `fp` is a file pointer to a previously opened stream. Files opened for these operations should be in binary mode.

This function returns the number of objects written to the file if the write operation was successful. If this value is less than **`num`** then an error has occurred. The `ferror()` function (that will be discussed shortly) can be used to determine the problem.

21.4.4 Reading a binary file

The `fread()` function can be used to read any type of data. The prototype for the same is:

```
size_t fread(void *buffer, size_t num_bytes, size_t count FILE *fp);
```

The **`buffer`** is a pointer to the region of memory that will receive the data from the file. The number of bytes to be read or written is specified by **`num_bytes`**. The argument count determines how many items (each `num_bytes` in length) are read or written. Finally, `fp` is a file pointer to a previously opened stream. Files opened for these operations should be in binary mode.

This function will return the number of objects read if the read operation is successful. It returns 0 if either end of file is reached or an error has occurred. The `feof()` and the `ferror()` functions (that will be discussed shortly) can be used respectively to determine the problem.

The `fread()` and `fwrite()` functions are often referred to as unformatted read or write functions.

As long as the file has been opened for binary operations, `fread()` and `fwrite()` can read and write any type of information. For example, the following program writes and then reads back a double, an `int` and a long value to and from a disk file. Notice how it uses the `sizeof()` function to determine the length of each data type.

Example 2:

```
#include <stdio.h>
main ()
{
    FILE *fp;
    double d = 23.31 ;
    int i = 13;
    long li = 1234567L;
    clrscr();
    if ( ( fp= fopen (“jak”, “wb+”)) == NULL )
    {
        printf(“cannot open file “);
        exit(1);
    }
    fwrite (&d, sizeof(double), 12, fp);
    fwrite (&i, sizeof(int), 1, fp);
    fwrite (&li, sizeof(long), 1,fp);
    fclose (fp);
    if ((fp= fopen (“jak”, “rb+”)) == NULL )
    {
        printf(“cannot open file “);
        exit(1);
    }
    fread (&d, sizeof(double), 1, fp);
    fread(&i, sizeof(int), 1, fp);
    fread (&li, sizeof(long), 1, fp);
    printf (“%f %d %ld”, d, i, li);
    fclose (fp);
}
```

As this program illustrates, the buffer can be read and often is simply the memory used to hold a variable. In this simple program, the return value of `fread()` and `fwrite()` are ignored. These values should however, be checked for errors for efficient programming.

One of the most useful applications of `fread()` and `fwrite()` involves reading and writing user-defined data types, especially structures. For example given the structure:

```
struct struct_type
{
    float balance;
    char name[80];
} cust;
```

The following statement writes the contents of **`cust`** to the file pointed to by **`fp`**.

```
fwrite(&cust, sizeof(struct struct_type), 1, fp);
```

21.5 File Handling Functions

The other file handling functions are discussed in this section.

21.5.1 Using ‘feof()’

When a file is opened for binary input, an integer value equal to the EOF may be read. The input routine will indicate an end of file in such a case, even though the physical end of the file has not reached. A function `feof()` can be used in such cases. The prototype of this function is:

```
int feof(FILE *fp );
```

It returns true if the end of the file has been reached, otherwise it returns false (`0`). This function is used while reading binary data.

The following code segment reads a binary file till the end of file is encountered.

```
.
.
while (!feof(fp))
    ch = fgetc(fp);
.
.
```

21.5.2 The ‘rewind()’ function

The `rewind()` function resets the file position indicator to the beginning of the file. It takes the file pointer as its argument.

The syntax for `rewind()` is:

```
rewind(fp);
```

The following program opens a file in write/read mode, takes strings as input using `fgets()`, rewinds the file and then displays the same strings using `fputs()`.

Example 3:

```
#include <stdio.h>
main()
{
    FILE *fp;
    char str [80];
    /* Writing to File JAK */
    if ( ( fp=fopen ("jak", "w+") ) == NULL)
    {
        printf ( "Cannot open file \n\n");
        exit(1);
    }
    clrscr ();
    do
    {
        printf ("Enter a string (CR to quit): \n");
        gets (str);
        if(*str != '\n')
        {
            strcat (str, "\n"); /* add a new line */
            fputs (str, fp);
        }
    } while (*str != '\n');
    /*Reading from File JAK */
    printf ("\n\n Displaying Contents of File JAK\n\n");
    rewind (fp);
    while (!feof(fp))
    {
        fgets (str, 81, fp);
        printf ("\n%s", str);
    }
    fclose(fp);
}
```

A sample run for the above program is:

```
Enter a string (CR to quit):
This is input line 1
Enter a string (CR to quit) :
This is input line 2
Enter a string (CR to quit):
This is input line 3
Enter a string (CR to quit):
Displaying Contents of File JAK
This is input line 1
This is input line 2
This is input line 3
```

21.5.3 The ‘ferror()’ function

The `ferror()` function determines whether a file operation has produced an error. Its prototype is:

```
int ferror(FILE * fp) ;
```

where `fp` is a valid file pointer. It returns true if an error has occurred during the last file operation; otherwise, it returns `false`.

As each operation sets the error condition, `ferror()` should be called immediately after each operation; otherwise, an error may be lost. The earlier program can be modified to check and warn about any errors in writing as given below.

```
...
do
{
    printf(“ Enter a string (CR to quit): \n”);
    gets(str);
    if(*str != ‘\n’)
    {
        strcat (str, “\n”); /* add a new line */
        fputs (str, fp);
    }
    if(ferror(fp))
        printf(“\nERROR in writing\n”);
    } while(*str!=’\n’);
...
```

21.5.4 Erasing Files

The `remove()` function erases a specified file. Its prototype is:

```
int remove (char *filename);
```

It returns `0` if successful else it returns a nonzero value.

As an example, consider the following code segment:

```
.
.
printf ("\nErase file %s (Y/N) ? ", file1);
ans = getchar ();
.
.
if(remove(file1))
{
    printf ("\nFile cannot be erased");
    exit(1);
}
```

21.5.5 Flushing streams

Often, the standard output file is buffered. This means that the output to the file is collected in memory but not actually displayed until the buffer is full. If the program crashes, some characters may still be in the buffer. The result is that the program appears to have terminated earlier than it actually did. The `fflush()` function resolves this problem. As the name suggests, it flushes out the buffer. The action of flushing depends upon the file type. A file opened for read will have its input buffer cleared, while a file opened for write will have its output buffer written to the files.

The prototype for this function is:

```
int fflush(FILE * fp);
```

The `fflush()` function will write the contents of any buffered data to the file associated with `fp`. The `fflush()` function, with a null, flushes all files opened for output. It returns `0` if successful, otherwise, it returns EOF.

21.5.6 The Standard Streams

Whenever a C program starts execution under DOS, five special streams are opened automatically by the operating system. These five streams are:

- The standard input (`stdin`)
- The standard output (`stdout`)
- The standard error (`stderr`)
- The standard printer (`stdprn`)
- The standard auxiliary (`stdaux`)

The `stdin`, `stdout` and `stderr` are assigned by default to the system's console where as the `stdprn` is assigned to the first parallel printer port and `stdaux` is assigned to the first serial port. They are defined as fixed pointers of type `FILE`, so they can be used wherever the use of `FILE` pointer is legal. They can also effectively be transferred to other streams or device files whenever redirection is involved.

The following program prints the contents of the file onto the printer.

Example 4:

```
#include <stdio.h>
main()
{
    FILE *in;
    char buff[81], fname[13];
    clrscr();
    printf("Enter the Source File Name:");
    gets(fname);
    if((in=fopen(fname, "r"))==NULL)
    {
        fputs("\nFile not found", stderr);
        /* display error message on standard error rather than standard output */
        exit(1);
    }
    while(!feof(in))
    {
        if(fgets(buff, 81, in))
        {
            fputs(buff, stdprn);
            /* Send line to printer */
        }
    }
    fclose(in);
}
```

Note the use of the stderr stream with the fputs() function in the above program. It is used instead of the printf() function because the destination of the printf() function is the stdout, which can be redirected. If the output of a program was redirected and an error occurred during execution, then any error message given to the stdout stream would also be redirected. To avoid this, the stderr stream is used to display the error message on the screen because the destination of the stderr is also the console, but the stderr stream cannot be redirected. It always displays the message on the screen.

21.5.7 Current Active Pointer

In order to keep track of the position where I/O operations take place, a pointer is maintained in the FILE structure. Whenever a character is read from or written to the stream, the current active pointer (known as curp) is advanced. Most of the I/O functions refer to curp, and update it after the input or output procedures on the stream. The current location of the current active pointer can be found with the help of the ftell() function. The ftell() function returns a long int value that gives the position of curp from the beginning of the file in the specified stream. The prototype of the ftell() function is:

```
long int ftell(FILE *fp);
```

The following extract of a program displays the location of the current pointer on the stream fp.

```
printf("The current location of the file pointer is : %1d ", ftell(fp));
```

Setting Current Position

Immediately after opening the stream, the current active pointer position is set to zero and points to the first byte of the stream. As seen earlier, whenever a character is read from or written to the stream, the current active pointer is advanced. The pointer may be set to any position other than the current pointer at any point of time in a program. The rewind() function, sets the pointer position to the start of the program. Another function, which can be used to set the pointer position is fseek().

The fseek() function repositions the curp by the specified number bytes from the start, the current position or the end of the stream depending upon the position specified in the fseek() function. The prototype of the fseek() function is

```
int fseek(FILE *fp, long int offset, int origin);
```

where `offset` is the number of bytes beyond the file location given by origin.

The origin indicates the starting position of the search and must have the value of either 0, 1 or 2, which represent three symbolic constants (defined in stdio.h) as shown in Table 21.4:

Origin	File location
SEEK_SET or 0	Beginning of file
SEEK_CUR or 1	Current file pointer position
SEEK_END or 2	End of file

Table 21.4: Symbolic Constants

A return value of zero means `fseek()` has been successful and a non-zero value means `fseek()` has failed.

The following code segment seeks 6th record in the file:

```
struct addr
{
    char name[40];
    char street[40];
    char city[40];
    char state[3];
    char pin[7];
}
FILE *fp;
.
.
.
fseek(fp,5L*sizeof(struct addr),SEEK_SET);
```

The `sizeof()` function is used to find the length of each record in terms of bytes. The return value is used to determine the number of bytes, to skip the first 5 records.

21.5.8 ‘fprintf()’ and ‘fscanf()’

In addition to the discussed I/O functions, the buffered I/O system also includes `fprintf()` and `fscanf()`. These functions are similar to `printf()` and `scanf()` except that they operate with files.

The prototypes of fprintf() and fscanf() are:

```
int fprintf(FILE * fp, const char *control_string,..);
int fscanf(FILE *fp, const char *control_string,...);
```

where, fp is the file pointer returned by a call to fopen().

The fprintf() and fscanf() functions direct their I/O operations to the file pointed to by fp. The following program code reads a string and an integer from the keyboard, writes them to a disk file, and then reads the information and displays it on the screen.

```
.
.
printf("Enter a string and a number: ");
fscanf(stdin, "%s %d", str, &no);
/* read from the keyboard */
fprintf(fp, "%s %d", str, no);
/* write to the file */
fclose (fp);
.
.
fscanf(fp, "%s %d", str, &no)
/* read from file */
fprintf(stdout, "%s %d", str, no)
/* print on screen */
.
.
```

Remember that, although fprintf() and fscanf() often are the easiest way to write and read assorted data to and from disk files, they are not always the most efficient. The reason being that extra overhead is incurred with each call, since the data is written in formatted ASCII data (as it would appear on the screen) instead of being written in binary format. So, if speed or file size is a concern, fread() and fwrite() are a better choice.

Summary

- The C language does not contain any explicit I/O statements. All I/O operations are carried out using functions from the standard C library.
- There are two types of streams - the text and binary streams.
- A text stream is a sequence of characters.
- A binary stream is a sequence of bytes.
- A file may be anything from a disk file to a terminal or a printer.
- A file pointer is a pointer to a structure, which contains information about the file, including its name, current position of the file, whether the file is being read or written, and whether errors or end of the file have occurred.
- The fopen() function opens a stream for use and links a file with that stream.
- The fclose() function closes a stream that was opened by a call to fopen().
- The function fcloseall() can be used when many open streams have to be closed at the same time.
- The function fputc() is used to write characters, and the function fgetc() is used to read characters from an open file.
- The functions fgets() and fputs() do the same operations as that of fputc() and fgetc(), except that they work with strings.
- The feof() function is used to indicate end-of-file when the file is opened for binary operations.
- The rewind() function resets the file position indicator to the beginning of the file.
- The function ferror() determines whether a file operation has produced an error.
- The remove() function erases the specified file.

Summary

- The fflush() function flushes out the buffer. If a file is opened for read the input buffer will be cleared, while a file opened for write will have its output buffer written to the files.
- The fseek() function can be used to set the file pointer position.
- The library functions, fread() and fwrite() are used to read and write entire blocks of data onto the file.
- The buffered I/O system also includes two functions fprintf() and fscanf(), which are similar to the functions printf() and scanf(), except that they operate on files.

Check Your Progress

1. The two types of streams are the __________ and __________ streams.

2. Open files are closed when a program crashes. **(True/False)**

3. The __________ function opens a stream for use and links a file with that stream.

4. The function used for writing characters to a file is __________.

5. The `fgets()` function considers a new line character as a part of the string. **(True/False)**

6. The __________ function resets the file position indicator to the beginning of the file.

7. Whenever a character is read from or written to the stream, the __________ is incremented.

8. Files on which `fread()` and `fwrite()` operate must be opened in __________ mode.

9. The current location of the current active pointer can be found with the help of the ______ function.

Try It Yourself

1. Write a program that accepts data into a file and prints it in reverse order.

2. Write a program that transfers data from one file to another, excluding all the vowels (a, e, i, o, u). Exclude vowels in both upper and lower case. Display the contents of the new file.

Session
22

File Handling (Lab)

Objectives

At the end of this session, you will be able to:

- *Perform operations on text and binary files*
- *Open and close files*
- *Read from and write to files*
- *Use the file pointer*

The steps given in this session are detailed, comprehensive and carefully thought through. This has been done so that the learning objectives are met and the understanding of the tool is complete. Follow the steps carefully.

Part I – For the first 1 Hour and 30 Minutes:

22.1 File Handling in C

C provides a uniform interface for handling input and output (I/O). The access methods for files are same as those for handling devices. The key to this uniformity is that there are no file types in C. C treats all the files as streams.

22.1.1 Reading, Writing and Accessing data in files

There are several file handling functions in the stdio.h header file. Let us write a C program that makes use of these functions. The program creates a simple banking system. Customer details are accepted and stored in a file called customer. Details of transactions such as deposit and withdrawal are validated against the customer file. Valid transactions are recorded in the trans file. A report on low customer balances is also printed. The steps are listed below:

1. Define structures for customer and transaction data.

The code will be,

```
struct cust_st
{
    int acc_no;
    char cust_nm[30];
    float bal;
};
struct tran_st
{
    int acc_no;
    char trantype;
    float amt;
};
```

2. Display a menu to perform the various operations based on user input.

The code will be,

```
while(choice!=4)
{
    clrscr();
    printf("\nSelect choice from menu\n\n1. Accept customer details\
n 2. Record Withdrawal/Deposit transaction\n 3. Print Low Balance
Report\n 4. Exit\n\n Enter choice: ");
    scanf(" %d", &choice);
    .
    .
}
```

3. Invoke the appropriate function based on user choice.

The code will be,

```
if(choice==1)
    addcust();
else if(choice==2)
    rectran();
else if(choice==3)
    prnlowbal();
```

4. In the function to add customer details, define the file pointer to be associated with the customer file. Declare a structure variable for accepting the customer data.

The code for the same will be,

```
FILE *fp;
struct cust_st custdata;
```

5. Open the customer file in append mode so as to be able to add customer records. Confirm that the file open operation takes place.

The code for the same will be,

```
if((fp=fopen("customer", "a+"))==NULL)
{
    printf("\nERROR opening customer file");
    getch();
    return;
}
```

6. Accept the customer data into the structure variable and write the data to the customer file.

The code for the same will be,

```
fwrite(&custdata, sizeof(struct cust_st), 1, fp);
```

7. Close the customer file at the end of data entry.

The code for the same will be,

```
fclose(fp);
```

8. In the function to record transactions, define variables for the file pointers to the customer and trans files. Also, define structure variables to accept transaction data and read customer data.

The code for the same will be,

```
FILE *fp1, *fp2;
struct cust_st custdata;
struct tran_st trandata;
```

9. Open the two files in appropriate modes. The customer file should be opened for reading and updation, whereas the trans file should allow adding of new records.

The code for the same will be,

```
if((fp1=fopen("customer", "r+w"))==NULL)
{
    printf("\nERROR opening customer file");
    getch();
    return;
}
if((fp2=fopen("trans", "a+"))==NULL)
{
    printf("\nERROR opening transaction file");
    getch();
    return;
}
```

10. Accept the account number for the transaction and ensure that it exists in the customer file.

The code for the same will be,

```
while((fread(&custdata, size, 1, fp1))==1 && found=='n')
{
    if(custdata.acc_no==trandata.acc_no)
    {
        found='y';
        break;
    }
}
```

11. Ensure that a valid transaction type is entered.

The code for the same will be,

```
if(trandata.trantype!='D' && trandata.trantype!='d' && trandata.
trantype!='W' && trandata.trantype!='w')
    printf("\t\tInvalid transaction type, please reenter");
```

12. For withdrawal transactions, ensure that the withdrawal amount is available in the customer account. If available, update the account balance. Update the account balance for deposit transactions too.

The code for the same will be,

```
if(trandata.trantype=='W' || trandata.trantype=='w')
{
    if(trandata.amt>custdata.bal)
        printf("\nAccount balance is %.2f. Please reenter withdrawal
amount.", custdata.bal);
    else
    {
        custdata.bal-=trandata.amt;
        .
        .
    }
    else
    {
        custdata.bal+=trandata.amt;
        .
        .
    }
}
```

13. Write the new transaction record to the trans file and the updated customer record to the customer file.

The code for the same will be,

```
fwrite(&trandata, sizeof(struct tran_st), 1, fp2);
fseek(fp1, (long)(-size), 1);
fwrite(&custdata, size, 1, fp1);
```

Note that during the check for the customer account number, the last record read was for the customer whose transaction is being processed. So, the file pointer to the **`customer`** file would be at the end of the record that needs to be updated. The file pointer is repositioned to the beginning of the record using the `fseek()` function. Here **`size`** is the integer variable that stores the size of the structure for customer data.

14. Close the two files after the transactions data entry.

The code for the same will be,

```
fclose(fp1);
fclose(fp2);
```

15. In the function to display low balances, define the file pointer to be associated with the customer file. Declare a structure variable for reading the customer data.

The code for the same will be,

```
FILE *fp;
struct cust_st custdata;
```

16. After opening the file in the read mode, read each customer record and check the balance. If it is less than 250, print the record.

The code for the same will be,

```
while((fread(&custdata, sizeof(struct cust_st), 1, fp))==1)
{
    if(custdata.bal<250)
    {
        .
        .
        printf("\n%d\t%s\t%.2f", custdata.acc_no, custdata.cust_nm,
custdata.bal);
    }
}
```

17. Close the customer file.

The code for the same will be,

```
fclose(fp);
```

Let us look at the complete program.

1. Invoke the editor in which you can type the C program.

2. Create a new file.

3. Type the following code:

```
#include<stdio.h>
struct cust_st
{
    int acc_no;
    char cust_nm[30];
    float bal;
};
struct tran_st
{
    int acc_no;
    char trantype;
    float amt;
};

void main()
{
    int choice=1;
    while(choice!=4)
    {
        clrscr();
        printf("\nSelect choice from menu\n\n 1. Accept customer
details\n 2.Record Withdrawal/Deposit transaction\n 3. Print Low
Balance Report\n 4. Exit\n\n Enter choice: ");
        scanf(" %d", &choice);
        if(choice==1)
            addcust();
        else if(choice==2)
            rectran();
        else if(choice==3)
            prnlowbal();
    }
}
```

```
addcust()
{
    FILE *fp;
    char flag='y';
    struct cust_st custdata;
    clrscr();
    if((fp=fopen("customer", "a+"))==NULL)
    {
        printf("\nERROR opening customer file");
        getch();
        return;
    }
    while(flag=='y')
    {
        printf("\n\nEnter Account number: ");
        scanf(" %d", &custdata.acc_no);
        printf("\nEnter Customer Name: ");
        scanf("%s", custdata.cust_nm);
        printf("\nEnter Account Balance: ");
        scanf(" %f", &custdata.bal);
        fwrite(&custdata, sizeof(struct cust_st), 1, fp);
        printf("\n\nAdd another? (y/n): ");
        scanf(" %c", &flag);
    }
    fclose(fp);
}

rectran()
{
    FILE *fp1, *fp2;
    char flag='y', found, val_flag;
    struct cust_st custdata;
    struct tran_st trandata;
    int size=sizeof(struct cust_st);
    clrscr();
    if((fp1=fopen("customer", "r+w"))==NULL)
    {
        printf("\nERROR opening customer file");
        getch();
        return;
    }
    if((fp2=fopen("trans", "a+"))==NULL)
    {
        printf("\nERROR opening transaction file");
        getch();
        return;
    }
    while(flag=='y')
    {
        printf("\n\nEnter Account number: ");
        scanf(" %d", &trandata.acc_no);
        found='n';
        val_flag='n';
        rewind(fp1);
        if(found=='y')
        {
            while(val_flag=='n')
            {
                printf("\nEnter Transaction type (D/W): ");
                scanf(" %c", &trandata.trantype);
                if(trandata.trantype!='D' && trandata.trantype!='d' &&
trandata.trantype!='W' && trandata.trantype!='w')
                    printf("\t\tInvalid transaction type, please
reenter");
                else
                    val_flag='y';
                }
            val_flag='n';
            while(val_flag=='n')
            {
               printf("\nEnter amount: ");
               scanf(" %f", &trandata.amt);
               if(trandata.trantype=='W' ||trandata.trantype=='w')
                {
                    if(trandata.amt>custdata.bal)
                        printf("\nAccount balance is %.2f. Please
reenter withdrawal amount.",custdata.bal);
```

```
                    else
                    {
                         custdata.bal-=trandata.amt;
                         val_flag='y';
                    }
                }
                else
                {
                   custdata.bal+=trandata.amt;
                   val_flag='y';
                }
            }
            fwrite(&trandata, sizeof(struct tran_st), 1, fp2);
            fseek(fp1, (long)(-size), 1);
            fwrite(&custdata, size, 1, fp1);
        }
        else
            printf("\nThis account number does not exist");
        printf("\nRecord another transaction? (y/n): ");
        scanf(" %c", &flag);
    }
    fclose(fp1);
    fclose(fp2);
}

prnlowbal()
{
    FILE *fp;
    struct cust_st custdata;
    char flag='n';
    clrscr();
    if((fp=fopen("customer", "r"))==NULL)
    {
        printf("\nERROR opening customer file");
        getch();
        return;
    }
    printf("\nReport on account balances below 250\n\n");
    while((fread(&custdata, sizeof(struct cust_st), 1, fp))==1)
    {
        if(custdata.bal<250)
        {
            flag='y';
            printf("\n%d\t%s\t%.2f", custdata.acc_no, custdata.cust_nm,
custdata.bal);
        }
    }
    if(flag=='n')
        printf("\nNo account balances found below 250");
    getch();
    fclose(fp);
}
```

To see the output, follow these steps:

4. **Save the file with the name `filesI.C`.**

5. **Compile the file, `filesI.C`.**

6. **Execute the program, `filesI.C`.**

7. **Return to the editor.**

The output of the program is shown below:

```
Select choice from menu

1. Accept customer details

2. Record Withdrawal/Deposit transaction

3. Print Low Balance Report

4. Exit

Enter choice:
```

A sample output of the function to add customer details is shown below:

```
Enter Account number: 123

Enter Customer Name: E.Wilson

Enter Account Balance: 2000

Add another? (y/n):
```

A sample output of the function to add transaction details is shown below:

```
Enter Account number: 123

Enter Transaction type (D/W): W

Enter amount: 1000

Record another transaction? (y/n):
```

A sample output of the function to display the low balance report is shown below:

```
Report on account balances below 250

104    Jones    200
113    Sharon   150
120    Paula    200
```

Session 22

Part II – For the next 30 Minutes:

1. Write a C Program to display the differences between two files accepted as command line arguments. For each difference, display the position at which the difference is located and the characters in the two files at that position. Also, ensure that the user enters a valid number of command-line arguments. Lastly, display the total number of differences found.

   To do this,

   a. Declare the variables `argv` and `argc` to receive the command line arguments.

   b. Declare file pointers for the two files.

   c. Validate `argc` to ensure the correct number of command-line arguments is entered.

   d. Open the two files in the read mode.

   e. Set a loop to read a character from both the files till the end of file is encountered for either.

   f. If the characters are different, display them along with their position. Increment the counter for differences.

   g. If the end of file is encountered for a file, print the remaining characters of the other file as differences.

   h. Check the differences counter to display appropriate messages.

   i. Close the two files.

Session 22

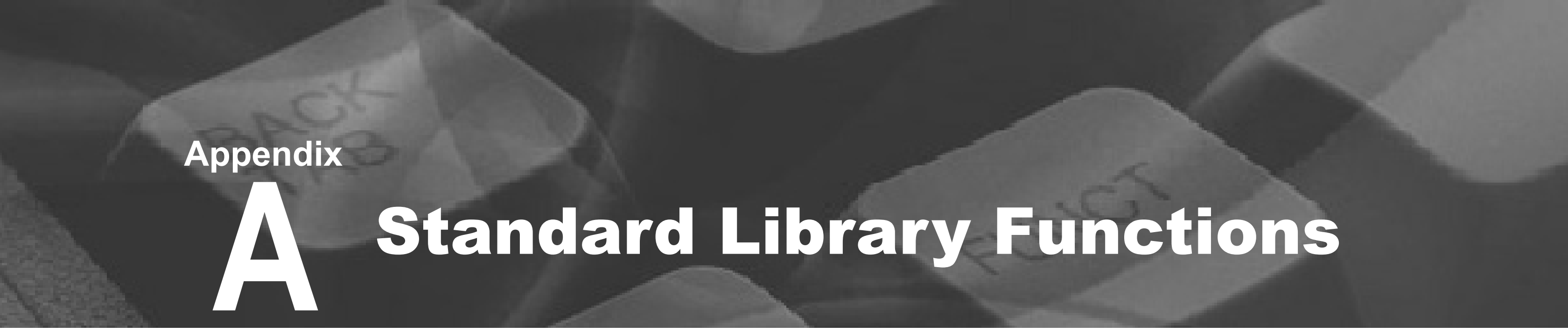
Try It Yourself

1. Write a C program to copy the contents of one file onto another excluding the words **`a`**, **`an`** and **`the`**.

2. Write a C program to accept two series of numbers. Store each series in a separate file. Sort the series in each file. Merge the two series into one, sort, and store the resultant series into a new file. Display the contents of the new file.

APPENDIX

“The real voyage of discovery consists
not in seeking new lands,
but in seeing with new eyes”



Appendix A Standard Library Functions

Input and Output: <stdio.h>

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

FILE *freopen(const char *filename, const char *mode, FILE *stream)

int fflush(FILE *stream)

int fclose(FILE *stream)

int remove(const char *filename)

int rename(const char *oldname, const char *newname)

FILE *tmpfile(void)

char *tmpnam(char s[L_tmpnam])

int setvbuf(FILE *stream, char *buf, int mode, size_t size)

void setbuf(FILE *stream, char *buf)

int fprint(FILE *stream, const char *format, ...)

int sprintf(char *s, const char *format, ...)

vprintf(const char *format, va_list arg)

vfprintf(FILE *stream, const char *format, va_list arg)

vsprintf(char *s, const char *format, va_list arg)

int fscanf(FILE *stream, const char *format, ...)

int scanf(const char *format, ...)

int sscanf(char *s, const char *format, ...)

int fgetc(FILE *stream)

char *fgets(char *s, int n, FILE *stream)

int fputc(int c, FILE *stream)

int fputs(const char *s, FILE *stream)

int getc(FILE *stream)

int getchar(void)

char *gets(char *s)

int putc(int c, FILE *stream)

int putchar(int c)

int ungetc(int c, FILE *stream)

size_t fread(void *ptr, size_t size, size_t nobj, FILE *stream)

size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *stream)

int fseek(FILE *stream, long offset, int orogin)

long ftell(FILE *stream)

void rewind(FILE *stream)

int fgetpos(FILE *stream, fpos_t *ptr)

int fsetpos(FILE *stream, const fpos_t *ptr)

void clearerr(FILE *stream)

int feof(FILE *stream)

int ferror(FILE *stream)

void perror(const char *s)

Character Class Tests: <ctype.h>

isalnum(c)

isalpha(c)

iscntrl(c)

isdigit(c)

isgraph(c)

islower(c)

isprint(c)

ispunct(c)

isspace(c)

isupper(c)

isxdigit(c)

String Functions: <string.h>

char *strcpy(s , ct)

char *strncpy(s , ct , n)

char *strcat(s , ct)

char *strncat(s , ct , n)

int strcmp(cs , ct)

int strncmp(cs , ct ,n)

char *strchr(cs , c)

char *strrchr(cs , c)

size_t strspn(cs , ct)

size_t strcspn(cs , ct)

char *strstr(cs , ct)

size_t strlen(cs)

char *strerror(n)

char *strtok(s , ct)

Mathematical Functions: <math.h>

sin(x)

cos(x)

tan(x)

asin(x)

acos(x)

atan(x)

atan2(x)

sinh(x)

cosh(x)

tanh(x)

exp(x)

log(x)

log10(x)

pow(x,y)

sqrt(x)

ceil(x)

floor(x)

fabs(x)

ldexp(x)

frexp(x,double *ip)

modf(x,double *ip)

fmod(x,y)

Utility Functions: <stdlib.h>

double atof(const char *s)

int atoi(const char *s

long atol(const char *s)

double strrod(const char *s, char **endp)

long strtol(const char *s, char **endp, int base)

unsigned long strtoul(const char *s, char **endp, int base)

int rand(void)

void srand(unsigned int seed)

void *calloc(size_t nobj, size_t size)

void *malloc(size_t size)

void *realloc(void *p, size_t size)

void free(void *p)

Standard Library Functions

void abort(void)

void exit(int status)

int atexit(void (*fcn)(void))

int system(const char *s)

char *getenv(const char *name)

void *bsearch(const void *key, const void *base, size_t n, size_t size, int (*cmp)(const void

*keyval, const void *datum))

void qsort(void *base, size_t n, size_t size, int (*cmp)(const void *, const void *))

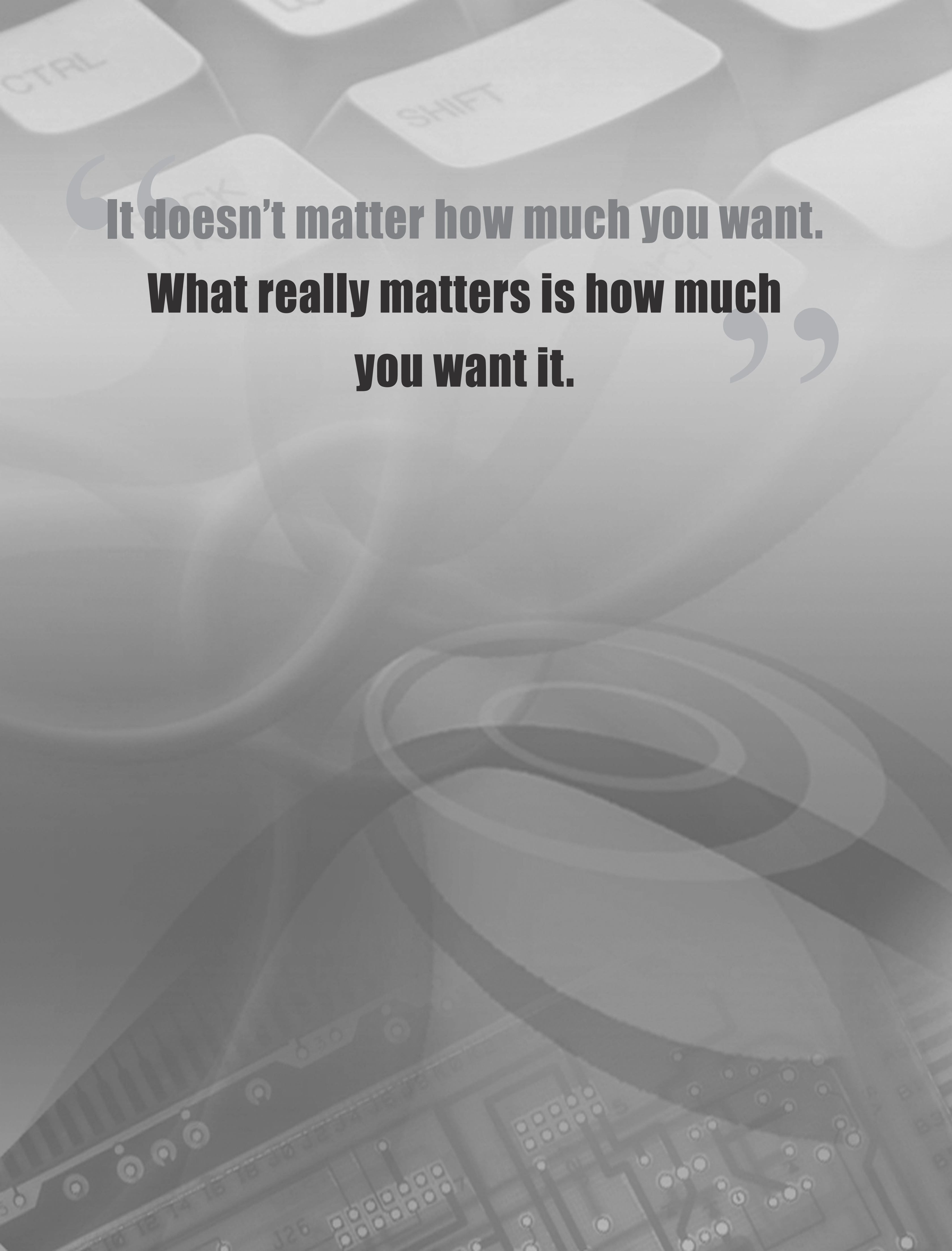
int abs(int n)

long labs(long n)

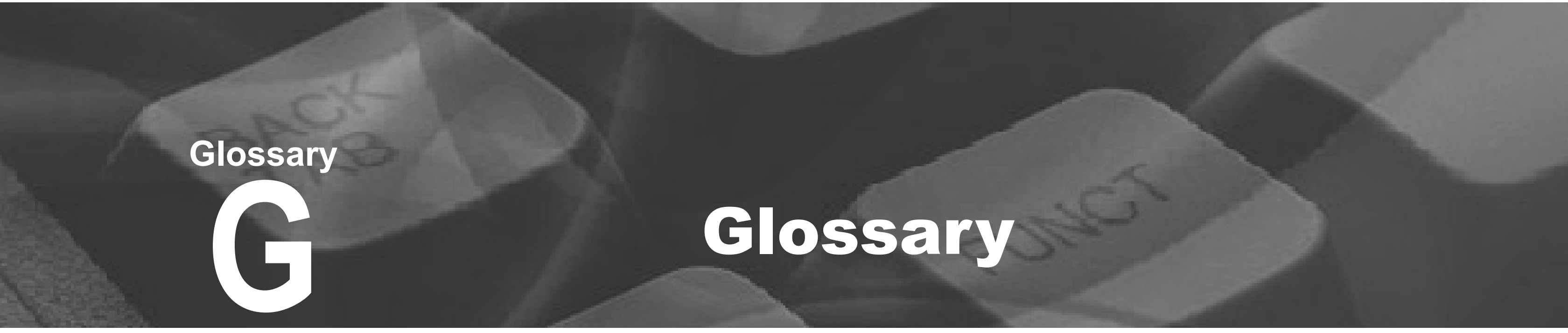
div_t div(int num, int denom)

ldiv_t ldiv(long num , long denom)

GLOSSARY



“It doesn’t matter how much you want.
What really matters is how much you want it.”



Glossary

G

Glossary

A

Algorithm

Logical and concise list of steps required in solving a problem.

Array

A group of variables, which are of the same data type, and can be accessed using a common name.

B

Binary operator

An operator which requires two operands.

Binary Stream

A sequence of bytes with a one-to-one correspondence to those in the external device, that is, there are no character translations.

Boolean data type

It consists of either of the two values, True or False. Some languages use 0 for False and some non-zero value for True.

Bubble sort

A type of sort algorithm. In this the values of the elements are compared with the value in the adjacent element. If it is smaller, swapping takes place. In this manner, the smaller elements bubble up, and at the end, the array is sorted.

Buffer

A buffer is a temporary storage area, either in the memory or on the controller card for the device.

C

Char

Char type occupies one byte long and is capable of holding one character.

Glossary

Code

Collection of program statements; A Program.

Code snippet

Few lines from a program. A part of a program, performing an individual function.

Constant

A constant is a value whose worth never changes.

Construct

A set of instructions or steps in a program/pseudocode.

Counter variable

A type of variable used to keep track of the number of times a particular operation has been performed in a loop.

D

Data type

It is used to specify the type of data to be stored in a variable. Indirectly, therefore, it decides the amount of memory to be allocated to a variable to store that particular type of data.

E

Expression

Combination of an operator and its operand.

F

Flowchart

A graphical representation of an algorithm. It charts the flow of instructions or activities in a process. Each such activity is depicted using symbols.

Function

A set of statements, which perform a specific task. Functions may or may not return a value.

Function library

A collection of functions. Generally, a function library contains functions that deal with a specific task.

I

Index

It indicates the array element which is to be accessed. The array index generally starts at 0.

Initialization

It is the process of assigning some initial value to a variable.

Insertion Sort

In insertion sort, each element in the array is examined, and inserted into its proper place among the elements that have already been sorted.

Integer

A number without any decimal portion. Integers can be positive or negative.

Iteration

See looping construct

K

Keyword

All languages reserve certain words for their internal use. These words hold a special meaning within the context of the particular language, and are referred to as ‘keywords’. While naming variables we need to ensure that we do not use one of these keywords as a variable name. Refer to Appendix B for the list of keywords in C.

L

Looping Construct

Often it is necessary to repeat certain steps a specific number of times or till some specified condition is met. The constructs which achieve these are known as iterative or looping constructs.

O

Operand

The value upon which the operator acts.

Operator

Symbols that perform some sort of operation upon data.

Glossary

P

Parameter

A value that is passed to a function.

Pass by value

A way of passing values to a function. The address of the original variables, and not the values, are passed in the pass-by-reference method to the called function. Therefore, modifications made to the contents of this variable affects the values of the original variables.

Pass by reference

A way of passing values to a function. When variables are passed by value, the values of the variables within the functions are not reflected back in the main program, since a copy of the original variables is made.

Pointer

A Pointer is a variable, which contains the address of a memory location of another variable.

Procedure

These are subprograms that essentially break up a program into modules. Procedures cannot return a value.

Program

We need to provide the computer with a set of instructions to solve any problem at hand. This set of instructions is called a program.

Pseudocode

Pseudocode is not actual code (pseudo=false), but a method of algorithm-writing which uses a certain standard set of words which makes it resemble code. However, unlike code, pseudocode cannot be compiled or run.

S

Standard function

These functions are generally built into a programming language, and are used for performing often-required tasks.

Statement

A single line of a pseudocode or a program.

Structure

A collection of variables of different data types that can be accessed as one unit using a common name.

Sub-program

Most programming languages provide us a way of breaking a long, continuous program into a series of small-programs, each of which perform a specific task. These small-programs are known as Sub-programs.

Syntax

Refers to the grammar of a programming language.

U

Unary operator

An operator requiring a single operand.

User defined function

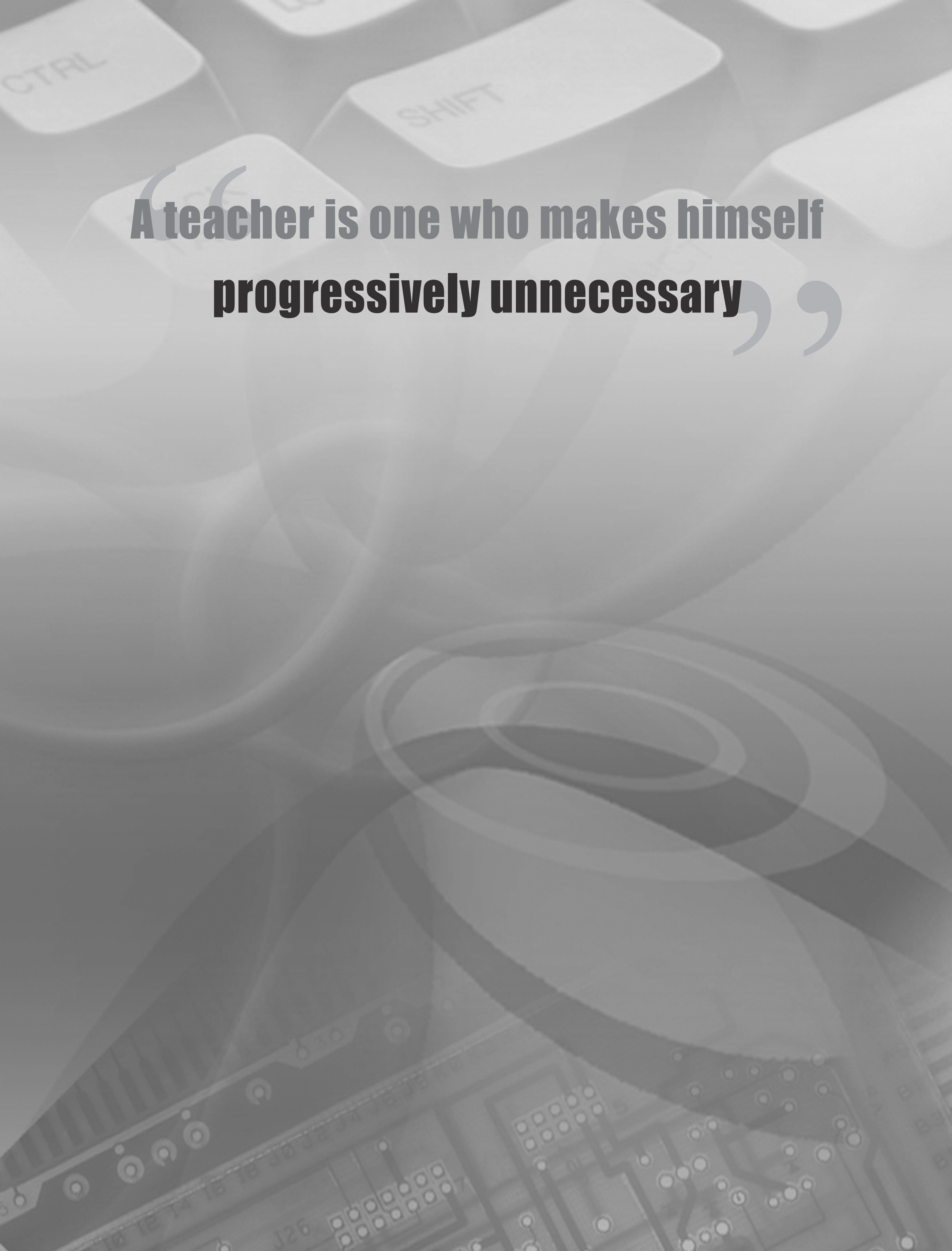
These are functions written by the programmers.

V

Variables

Named locations in memory. Programmers use variables to refer to the memory location where a particular value is to be stored.

“A teacher is one who makes himself progressively unnecessary”



Reader's Response

Name Of Book : ____

Batch :______________________________ **Date :** ________________

The members of the design team at Aptech Worldwide are always striving to enhance the quality of the books produced by them. As a reader, your suggestions and feedback are very important to us. They are of tremendous help to us in continually improving the quality of this book. Please rate this book in terms of the following aspects.

Aspects	**Rating**			
	Excellent	**Very Good**	**Good**	**Poor**
Presentation style	☐	☐	☐	☐
Simplicity of language	☐	☐	☐	☐
Topics chosen	☐	☐	☐	☐
Topic coverage	☐	☐	☐	☐

Presentation style

Suggestion :

____

____

Simplicity of language

Suggestion :

____

____

Topics chosen

Suggestion :

____

____

Topic coverage

Suggestion :

____

____

Aspects	**Rating**			
	Excellent	**Very Good**	**Good**	**Poor**
Explanation provided	☐	☐	☐	☐
Quality of pictures / diagrams	☐	☐	☐	☐

Suggestion :

Suggestion :

Overall suggestions :

Please fill up this response card and send it to :

The Design Centre,
Aptech Limited.
Aptech House,
A-65, MIDC, Marol,
Andheri (East),
Mumbai - 400 093.
INDIA

Your efforts in this direction will be most appreciated