

UNIT V FILE PROCESSING

Files – Types of file processing: Sequential access, Random access – Sequential access file - Example Program: Finding average of numbers stored in sequential access file - Random access file -Example Program: Transaction processing using random access files – Command line arguments

5.1 Introduction

A file is a semi-permanent, named collection of data. A File is usually stored on magnetic media, such as a hard disk or magnetic tape. Semi-permanent means that data saved in files stays safe until it is deleted or modified.

Named means that a particular collection of data on a disk has a name, like mydata.dat and access to the collection is done by using its name.

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a readymade structure.

5.1.1 Types of Files

1. Text files
2. Binary files

1. Text Files

A text file consists of consecutive characters, which are interpreted by the library functions used to access them and by format specifiers used in functions.

Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.

They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

2. Binary files

A binary file consists of bytes of data arranged in continuous block. A separate set of library functions is there to process such data files.

Binary files are mostly the .bin files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold higher amount of data, are not readable easily and provides a better security than text files.

5.1.2 File Operations

In C, you can perform four major operations on the file, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing information to a file

Function description

fopen() - create a new file or open an existing file

fclose() - closes a file

getc() - reads a character from a file

putc() - writes a character to a file

fscanf() - reads a set of data from a file

fprintf() - writes a set of data to a file

getw() - reads an integer from a file

putw() - writes an integer to a file

fseek() - set the position to desired point

ftell() - gives current position in the file

rewind() - sets the position to the beginning point

1. Opening a File or Creating a File

Opening a file means creating a new file with specified file name and with accessing mode.

The fopen() function is used to create a new file or to open an existing file.

Syntax:

****fp = FILE *fopen(const char *filename, const char *mode);***

Here, *fp is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file.

filename is the name of the file to be opened and mode specifies the purpose of opening the file.

Mode can be of following types:

Mode Description

Mode	Purpose
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode
rb	opens a binary file in reading mode
wb	opens or create a binary file in writing mode
ab	opens a binary file in append mode
rb+	opens a binary file in both reading and writing mode
wb+	opens a binary file in both reading and writing mode
ab+	opens a binary file in both reading and writing mode

2. Closing a File

A file must be closed after all the operation of the file have been completed. The `fclose()` function is used to close an already opened file.

Syntax:

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

Here `fclose()` function closes the file and returns zero on success, or EOF if there is an error in closing the file. This EOF is a constant defined in the header file `stdio.h`.

3. Reading and writing to a text file***i. Read a character from a file: fgetc function***

The 'fgetc' function is used to read a character from a file which is opened in read mode.

Syntax:

c=fgetc(p1);

where p1 is the file pointer.

ii. Read a data from a file: fscanf function

The fscanf function is used to read data from a file. It is similar to the scanf function except that fscanf() is used to read data from the disk.

Syntax:

fscanf(fb "format string", &v1, &v2...&vn);

where fb refers to the file pointer. v1, v2, ... vn refers variables whose values are read from the disk "format string" refers the control string which represents the conversion specification.

iii. Write a character to a file:fputc function

The function 'fputc' is used to write a character variable x to the file opened in write mode.

Syntax:

fputc(x,fp1);

where fp1 is the file pointer.

iv. Writing data to a file : fprintf()

fprintf() function is used to write data to a file. It is similar to the printf() function except that fprintf() is used to write data to the disk.

Syntax:

fprintf(fp, "format string", v1,v2... vn);

where fp refers to the file pointer.

5.2 Types of file processing

There are two main ways a file can be organized:

1. **Sequential Access** — In this type of file, the data are kept sequentially. To read last record of the file, it is expected to read all the records before that particular record. It takes more time for accessing the records.

2. **Random Access** — In this type of file, the data can be read and modified randomly. If it is desired to read the last record of a file, directly the same record can be read. Due to random access of data, it takes less access time as compared to the sequential file.

Sequential Access File

A Sequential file is characterized by the fact that individual data items are arranged serially in a sequence, one after another. They can only be processed in serial order from the beginning. In other words, the records can be accessed in the same manner in which they have been stored. It is not possible to start reading or writing a sequential file from anywhere except at the beginning.

Random Access File

The second and better method of arranging records of a file is called direct access or random access. In this arrangement one can have access to any record which is situated at the middle of the file without reading or passing through other records in the file.

5.3 Reading Sequential Access file

Data is stored in files so that the data can be retrieved for processing when needed

Example:

clients.dat file contents

100	Jones	9023.00
200	Frank	234.00
300	Mano	29023.00
400	Bala	2344.00

Program:

```
// Reading and printing a sequential file
```

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

```
#include <stdlib.h>

int main(void) {

    unsigned int account; // account number

    char name[30];        // account name

    double balance;       // account balance

    FILE *cfPtr; // cfPtr = clients.dat file pointer

    // fopen opens file; exits program if file cannot be opened

    if ((cfPtr = fopen("clients.dat", "r")) == NULL) {

        puts("File could not be opened");

        exit(0);

    }

    printf("%-10s%-13s%s\n", "Account", "Name", "Balance");

    fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);

    // while not end of file

    while (!feof(cfPtr)) {

        printf("%-10d%-13s%7.2f\n", account, name, balance);

        fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);

    } // end while

    fclose(cfPtr); // fclose closes the file

} // end main
```

Output:

Account	Name	Balance
100	Jones	9023.00
200	Frank	234.00
300	Mano	29023.00

400 Bala 2344.00

5.4 Read numbers from file and calculate Average

/ Program to read from the num.dat file and find the average of the numbers */*

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

```
#include <stdlib.h>
```

```
#define DATAFILE "prog15.dat"
```

```
int main() {
```

```
    FILE* fp;
```

```
    int n[50], i = 0;
```

```
    float sum = 0;
```

```
    if ((fp = fopen(DATAFILE, "r")) == NULL) {
```

```
        printf("Unable to open %s...\n", DATAFILE);
```

```
        exit(0);
```

```
    }
```

```
    puts("Reading numbers from num.dat");
```

```
    while (!feof(fp)) {
```

```
        fscanf(fp, "%d ", &n[i]);
```

```
        printf("%d %d\n", i, n[i]);
```

```
        sum += n[i];
```

```
        i++;
```

```
    }
```

```
    fclose(fp);
```

```
    // if no data is available in the file
```

```
    if (i == 0)
```

```
        printf("No data available in %s", DATAFILE);
```

```

float average = sum / i;

printf("The average is %.3f for %d numbers\n", average, i);

return 0;

}

```

Output:

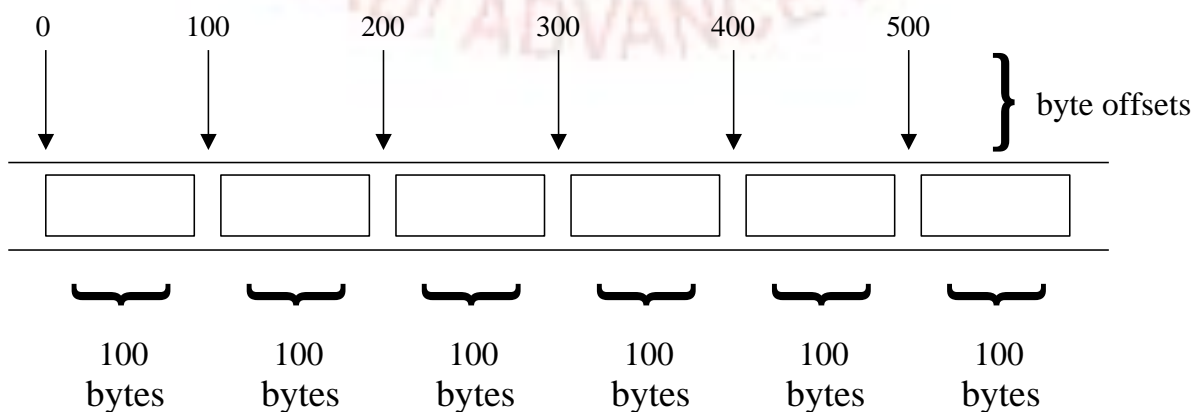
```

✧ prog15
Reading numbers from prog15.dat
0 90
1 10
2 20
3 30
4 40
5 50
The average is 40.000 for 6 numbers
✧

```

5.5 Random access file

- Access individual records without searching through other records
- Instant access to records in a file
- Data can be inserted without destroying other data
- Data previously stored can be updated or deleted without overwriting.
- Implemented using fixed length records
 - Sequential files do not have fixed length records



5.5.1 Functions For Selecting A Record Randomly

The functions used to randomly access a record stored in a file are `fseek()`, `ftell()`, `rewind()`, `fgetpos()`, and `fsetpos()`.

1. `fseek()`

- `fseek()` is used to reposition a binary stream. The prototype of `fseek()` can be given as,
- `int fseek(FILE *stream, long offset, int origin);`
- `fseek()` is used to set the file position pointer for the given stream. Offset is an integer value that gives the number of bytes to move forward or backward in the file. Offset may be positive or negative, provided it makes sense. For example, you cannot specify a negative offset if you are starting at the beginning of the file. The *origin* value should have one of the following values (defined in `stdio.h`):
- `SEEK_SET`: to perform input or output on offset bytes from start of the file
- `SEEK_CUR`: to perform input or output on offset bytes from the current position in the file
- `SEEK_END`: to perform input or output on offset bytes from the end of the file
- `SEEK_SET`, `SEEK_CUR` and `SEEK_END` are defined constants with value 0, 1 and 2 respectively.
- On successful operation, `fseek()` returns zero and in case of failure, it returns a non-zero value. For example, if you try to perform a seek operation on a file that is not opened in binary mode then a non-zero value will be returned.
- `fseek()` can be used to move the file pointer beyond a file, but not before the beginning.

Example: Write a program to print the records in reverse order. The file must be opened in binary mode. Use `fseek()`

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

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

```
main()
```

```
{    typedef struct employee
```

```
{    int emp_code;

    char name[20];

    int hra;

    int da;

    int ta;

};

FILE *fp;

struct employee e;

int result, i;

fp = fopen("employee.txt", "rb");

if(fp==NULL)

{    printf("\n Error opening file");

    exit(1);

}

for(i=5;i>=0;i--)

{    fseek(fp, i*sizeof(e), SEEK_SET);

    fread(&e, sizeof(e), 1, fp);

    printf("\n EMPLOYEE CODE : %d", e.emp_code);

    printf("\n Name : %s", e.name);

    printf("\n HRA, TA and DA : %d %d %d", e.hra, e.ta, e.da);

}

fclose(fp);

getch();

return 0;

}
```

2. rewind()

- `rewind()` is used to adjust the position of file pointer so that the next I/O operation will take place at the beginning of the file. It's prototype can be given as
- **`void rewind(FILE *f);`**
- `rewind()` is equivalent to calling `fseek()` with following parameters: `fseek(f,0L,SEEK_SET);`

3. `fgetpos()`

- The `fgetpos()` is used to determine the current position of the stream. It's prototype can be given as
`int fgetpos(FILE *stream, fpos_t *pos);`
- Here, `stream` is the file whose current file pointer position has to be determined. `pos` is used to point to the location where `fgetpos()` can store the position information. The `pos` variable is of type `fpos_t` which is defined in `stdio.h` and is basically an object that can hold every possible position in a `FILE`.
- On success, `fgetpos()` returns zero and in case of error a non-zero value is returned. Note that the value of `pos` obtained through `fgetpos()` can be used by the `fsetpos()` to return to this same position.

4. `fsetpos()`

- The `fsetpos()` is used to move the file position indicator of a stream to the location indicated by the information obtained in "`pos`" by making a call to the `fgetpos()`. Its prototype is
- **`int fsetpos(FILE *stream, const fpos_t pos);`**
- Here, `stream` points to the file whose file pointer indicator has to be re-positioned. `pos` points to positioning information as returned by "`fgetpos`".
- On success, `fsetpos()` returns a zero and clears the end-of-file indicator. In case of failure it returns a non-zero value

The program opens a file and reads bytes at several different locations.

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

```
main()
```

```
{
```

```
FILE *fp;

fpos_t pos;

char feedback[20];

fp = fopen("comments.txt", "rb");

if(fp == NULL)
{
    printf("\n Error opening file");
    exit(1);
}

// Read some data and then check the position.
fread( feedback, sizeof(char), 20, fp);
if( fgetpos(fp, &pos) != 0 )
{
    printf("\n Error in fgetpos()");
    exit(1);
}

fread(feedback, sizeof(char), 20, fp);

printf("\n 20 bytes at byte %ld: %s", pos, feedback);

// Set a new position and read more data
pos = 90;
if( fsetpos(fp, &pos ) != 0 )
{
    printf("\n Error in fsetpos()");
    exit(1);
}

fread( feedback, sizeof(char), 20, fp);
```



```
printf( "\n 20 bytes at byte %ld: %s", pos, feedback);

fclose(fp);

}
```

5. ftell()

The ftell function is used to know the current position of file pointer. It is at this position at which the next I/O will be performed. The syntax of the ftell() defined in stdio.h can be given as:

long ftell (FILE *stream);

On successful, ftell() function returns the current file position (in bytes) for stream. However, in case of error, ftell() returns -1.

When using ftell(), error can occur either because of two reasons:

First, using ftell() with a device that cannot store data (for example, keyboard)

Second, when the position is larger than that can be represented in a long integer. This will usually happen when dealing with very large files

```
FILE *fp;
char c;
int n;
fp=fopen("abc","w");
if(fp==NULL)
{
    printf("\n Error Opening The File");
    exit(1);
}
while((c=getchar())!=EOF)
    putc(c,fp);
n = ftell(fp);
fclose(fp);
fp=fopen("abc","r");
```

```
if(fp==NULL)
{
    printf("\n Error Opening The File");
    exit(1);
}

while(ftell(fp)<n)
{
    c= fgetc(fp);
    printf('%c', c);
}

fclose(fp);
```

5.6 Example Program: Transaction processing using random access files

The program maintains a bank's account information—updating existing accounts, adding new accounts, deleting accounts and storing a listing of all the current accounts in a text file for printing.

The program has five options.

Option 1

calls function `textFile` to store a formatted list of all the accounts (typically called a report) in a text file called `accounts.txt` that may be printed later. The function uses `fread` and the sequential file access techniques used in the program of Section below.

After **option 1** is chosen, the file **accounts.txt** contains:

<i>Acct Last</i>	<i>Name First Name</i>	<i>Balance</i>
<i>29 Brown</i>	<i>Nancy</i>	<i>-24.54</i>
<i>33 Dunn</i>	<i>Stacey</i>	<i>314.33</i>
<i>37 Barker</i>	<i>Doug</i>	<i>0.00</i>
<i>88 Smith</i>	<i>Dave</i>	<i>258.34</i>
<i>96 Stone</i>	<i>Sam</i>	<i>34.98</i>

Option 2

calls the function **updateRecord** to update an account. The function will update only a record that already exists, so the function first checks whether the record specified by the user is empty. The record is read into structure client with fread, then member acctNum is compared to 0. If it's 0, the record contains no information, and a message is printed stating that the record is empty. Then the menu choices are displayed. If the record contains information, function updateRecord inputs the transaction amount, calculates the new balance and rewrites the record to the file.

Enter account to update (1 - 100): 37

37 Barker Doug 0.00

Enter charge (+) or payment (-): +87.99

37 Barker Doug 87.99

Option 3

calls the function **newRecord** to add a new account to the file. If the user enters an account number for an existing account, newRecord displays an error message indicating that the record already contains information, and the menu choices are printed again

Enter new account number (1 - 100): 22

Enter lastname, firstname, balance

? Johnston Sarah 247.45

Option 4

calls function deleteRecord to delete a record from the file. Deletion is accomplished by asking the user for the account number and re-initialising the record. If the account contains no information, deleteRecord displays an error message indicating that the account does not exist.

Option 5

terminates program execution.

Example Program:

// Bank-account program reads a random-access file sequentially, updates data already written to the file, creates new data to be placed in the file, and deletes data previously in the file. //

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

```
#include <stdlib.h>
```

```
// clientData structure definition
```

```
struct clientData {
```

```
    unsigned int acctNum; // account number
```

```
    char lastName[15]; // account last name
```

```
    char firstName[10]; // account first name
```

```
    double balance; // account balance
```

```
}; // end structure clientData
```

```
// prototypes
```

```
unsigned int enterChoice(void);
```

```
void textFile(FILE *readPtr);
```

```
void updateRecord(FILE *fPtr);
```

```
void newRecord(FILE *fPtr);
```

```
void deleteRecord(FILE *fPtr);
```

```
int main(int argc, char *argv[]) {
```

```
    FILE *cfPtr; // credit.dat file pointer
```

```
    unsigned int choice; // user's choice
```

```
    // fopen opens the file; exits if file cannot be opened
```

```
    // Do not change the mode "rb+" - it will not work!
```

```
    if ((cfPtr = fopen("credit.dat", "rb+")) == NULL) {
```

```
        printf("%s: File could not be opened.\n", argv[0]);
```

```
        exit(-1);
```



```
}

// enable user to specify action

while ((choice = enterChoice()) != 5) {

    switch (choice) {

        // create text file from record file

        case 1:

            textFile(cfPtr); break;

        // update record

        case 2:

            updateRecord(cfPtr); break;

        // create record

        case 3:

            newRecord(cfPtr); break;

        // delete existing record

        case 4:

            deleteRecord(cfPtr); break;

        // display if user does not select valid choice

        default:

            puts("Incorrect choice"); break;

    } // end switch

} // end while

fclose(cfPtr); // fclose closes the file

} // end main

// create formatted text file for printing

void textFile(FILE *readPtr) {

    FILE *writePtr; // accounts.txt file pointer
```

```
int result;    // used to test whether fread read any bytes

// create clientData with default information

struct clientData client = {0, "", "", 0.0};

// fopen opens the file; exits if file cannot be opened

if ((writePtr = fopen("accounts.txt", "w")) == NULL) {

    puts("File could not be opened.");

} // end if

else {

    rewind(readPtr); // sets pointer to beginning of file

    fprintf(writePtr, "%-6s%-16s%-11s%10s\n", "Acct", "Last Name", "First Name",
"Balance");

    // copy all records from random-access file into text file
    while (!feof(readPtr)) {

        result = fread(&client, sizeof(struct clientData), 1, readPtr);

        // write single record to text file

        if (result != 0 && client.acctNum != 0) {

            fprintf(writePtr, "%-6d%-16s%-11s%10.2f\n", client.acctNum,
                client.lastName, client.firstName, client.balance);

        } // end if

    } // end while

    fclose(writePtr); // fclose closes the file

} // end else

} // end function textFile

// update balance in record

void updateRecord(FILE *fPtr) {
```

```
unsigned int account; // account number

double transaction; // transaction amount

// create clientData with no information
struct clientData client = {0, "", "", 0.0};

// obtain number of account to update
printf("%s", "Enter account to update ( 1 - 100 ): ");
scanf("%d", &account);

// move file pointer to correct record in file
fseek(fPtr, (account - 1) * sizeof(struct clientData), SEEK_SET);

// read record from file
fread(&client, sizeof(struct clientData), 1, fPtr);

// display error if account does not exist
if (client.acctNum == 0) {
    printf("Account #%d has no information.\n", account);
} else { // update record
    printf("%-6d%-16s%-11s%10.2f\n\n", client.acctNum, client.lastName,
        client.firstName, client.balance);

    // request transaction amount from user
    printf("%s", "Enter charge ( + ) or payment ( - ): ");
    scanf("%lf", &transaction);

    client.balance += transaction; // update record balance

    printf("%-6d%-16s%-11s%10.2f\n", client.acctNum, client.lastName,
        client.firstName, client.balance);

    // move file pointer to correct record in file

    // move back by 1 record length
    fseek(fPtr, -sizeof(struct clientData), SEEK_CUR);
```

```
// write updated record over old record in file

    fwrite(&client, sizeof(struct clientData), 1, fPtr);

} // end else

} // end function updateRecord
```

// delete an existing record

```
void deleteRecord(FILE *fPtr) {

    struct clientData client; // stores record read from file

    struct clientData blankClient = {0, "", "", 0}; // blank client

    unsigned int accountNum; // account number

    // obtain number of account to delete

    printf("%s", "Enter account number to delete ( 1 - 100 ): ");

    scanf("%d", &accountNum);

    // move file pointer to correct record in file

    fseek(fPtr, (accountNum - 1) * sizeof(struct clientData), SEEK_SET);

    // read record from file

    fread(&client, sizeof(struct clientData), 1, fPtr);

    // display error if record does not exist

    if (client.acctNum == 0) {

        printf("Account %d does not exist.\n", accountNum);

    } // end if

    else { // delete record

        // move file pointer to correct record in file

        fseek(fPtr, (accountNum - 1) * sizeof(struct clientData), SEEK_SET);
```



```
// replace existing record with blank record

fwrite(&blankClient, sizeof(struct clientData), 1, fPtr);

} // end else

} // end function deleteRecord

// create and insert record

void newRecord(FILE *fPtr) {

    // create clientData with default information

    struct clientData client = {0, "", "", 0.0};

    unsigned int accountNum; // account number

    // obtain number of account to create

    printf("%s", "Enter new account number ( 1 - 100 ): ");

    scanf("%d", &accountNum);

    // move file pointer to correct record in file

    fseek(fPtr, (accountNum - 1) * sizeof(struct clientData), SEEK_SET);

    // read record from file

    fread(&client, sizeof(struct clientData), 1, fPtr);

    // display error if account already exists

    if (client.acctNum != 0) {

        printf("Account #%d already contains information.\n", client.acctNum);

    } // end if

    else { // create record user enters last name, first name and balance

        printf("%s", "Enter lastname, firstname, balance\n? ");

        scanf("%14s%9s%lf", client.lastName, client.firstName, &client.balance);

        client.acctNum = accountNum;
```

```
// move file pointer to correct record in file

fseek(fPtr, (client.acctNum - 1) * sizeof(struct clientData), SEEK_SET);

// insert record in file

fwrite(&client, sizeof(struct clientData), 1, fPtr);

} // end else

} // end function newRecord


// enable user to input menu choice

unsigned int enterChoice(void) {
    unsigned int menuChoice; // variable to store user's choice
    // display available options
    printf("%s", "\nEnter your choice\n")
        "1 - store a formatted text file of accounts called\n"
        "  \"accounts.txt\" for printing\n"
        "2 - update an account\n"
        "3 - add a new account\n"
        "4 - delete an account\n"
        "5 - end program\n? ");

    scanf("%u", &menuChoice); // receive choice from user
    return menuChoice;
} // end function enterChoice
```

Output

```

> prog22
Acct  Last Name      First Name      Balance
1     Ashok          B               2378.00
2     Bala           M               234558.00
3     Mala           Seven           22244.00
55    Andres         John            410.00
99    Zebra          Jock            2234.00
> trans

Enter your choice
1 - store a formatted text file of accounts called
    "accounts.txt" for printing
2 - update an account
3 - add a new account
4 - delete an account
5 - end program
? 2
Enter account to update ( 1 - 100 ): 55
55    Andres         John            410.00

Enter charge ( + ) or payment ( - ): +1000
55    Andres         John            1410.00

Enter your choice
1 - store a formatted text file of accounts called
    "accounts.txt" for printing
2 - update an account
3 - add a new account
4 - delete an account
5 - end program
? 5
> prog22
Acct  Last Name      First Name      Balance
1     Ashok          B               2378.00
2     Bala           M               234558.00
3     Mala           Seven           22244.00
55    Andres         John            1410.00
99    Zebra          Jock            2234.00

```

5.7 Command line arguments

Command line argument is a parameter supplied to the program when it is invoked. Command line argument is an important concept in C programming. It is mostly used when you need to control your program from outside. Command line arguments are passed to the `main()` method.

Syntax:

int main(int argc, char *argv[])

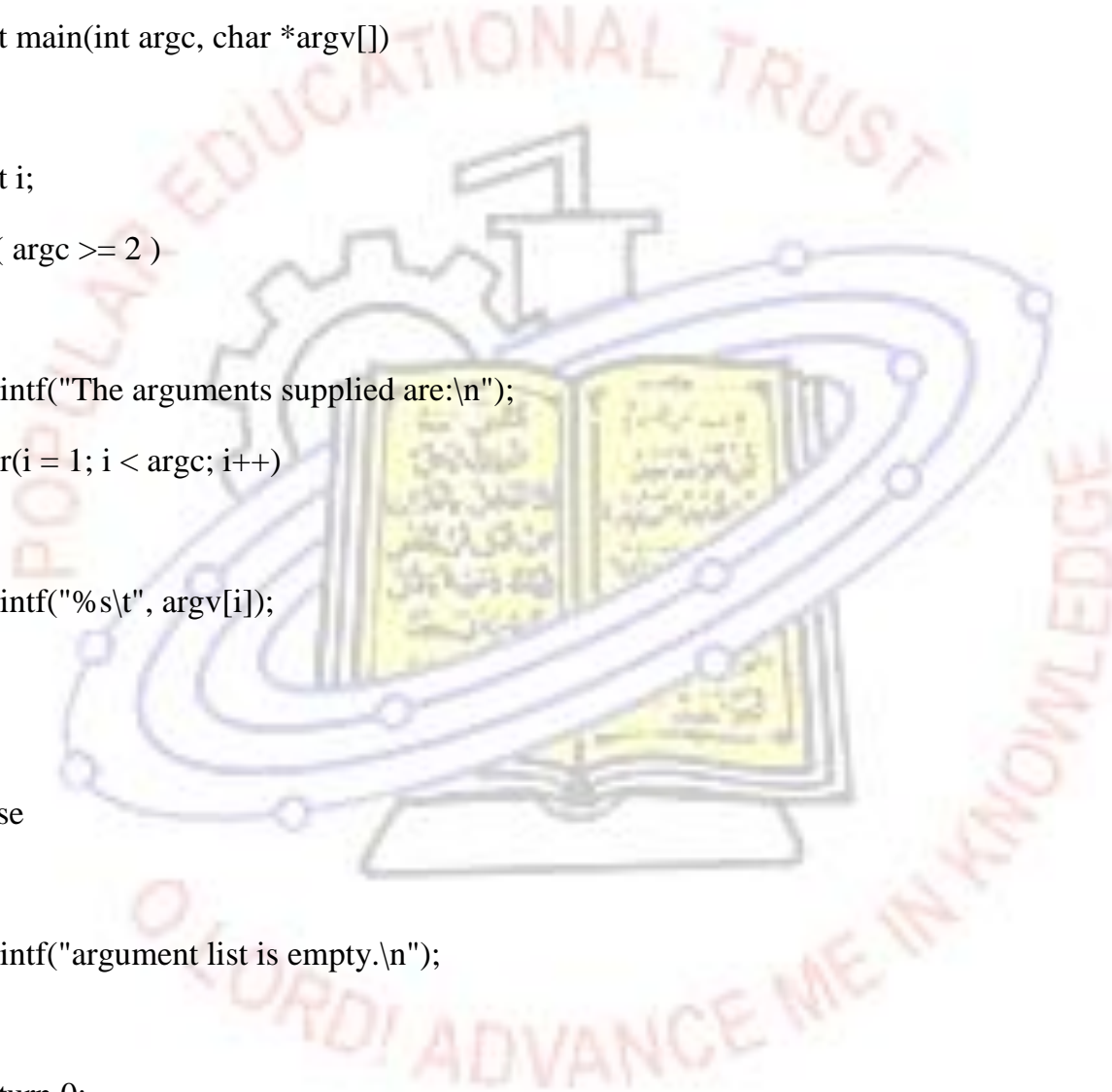
Here argc counts the number of arguments on the command line and argv[] is a pointer array which holds pointers of type char which points to the arguments passed to the program

Example:

```
#include <stdio.h>

#include <conio.h>

int main(int argc, char *argv[])
{
    int i;
    if( argc >= 2 )
    {
        printf("The arguments supplied are:\n");
        for(i = 1; i < argc; i++)
        {
            printf("%s\t", argv[i]);
        }
    }
    else
    {
        printf("argument list is empty.\n");
    }
    return 0;
}
```



Remember that argv[0] holds the name of the program and argv[1] points to the first command line argument and argv[n] gives the last argument. If no argument is supplied, argc will be 1.

Multiple Choice Questions

1. _____ is a collection of data.

- A. Buffer
- B. Stream
- C. File

Answer: File

2. If the mode includes b after the initial letter, what does it indicates?

- a) text file
- b) big text file
- c) binary file

Answer: binary file

3. What is the function of the mode 'w+'?

- a) create text file for writing, discard previous contents if any
- b) create text file for update, discard previous contents if any
- c) create text file for writing, do not discard previous contents if any
- d) create text file for update, do not discard previous contents if any

Answer: create text file for update, discard previous contents if any

4. fflush(NULL) flushes all _____

- a) input streams
- b) output streams
- c) previous contents
- d) appended text

Answer: output streams

5. What is the keyword used to declare a C file pointer.?

- A) file

- B) FILE
- C) FILEFP
- D) filefp

Answer: FILE

6. What is a C FILE data type.?

- A) FILE is like a Structure only
- B) FILE is like a Union only
- C) FILE is like a user define int data type
- D) None of the above

Answer: FILE is like a Structure only

7. Where is a file temporarily stored before read or write operation in C language.?

- A) Notepad
- B) RAM
- C) Hard disk
- D) Buffer

Answer: Buffer

8. Which function gives the current position of the file.

- A. fseek()
- B. fsetpos()
- C. ftell()
- D. Rewind()

Answer: ftell()

9. Which function is used to perform block output in binary files?

- A. fwrite()
- B. fprintf()

C. fputc()

D. fputs()

Answer: fwrite()

10. Select the standard stream in C

A. stdin

B. stdout

C. stderr

D. all of these

Answer: all of these

11. From which standard stream does a C program read data?

A. Stdin

B. stdout

C. stderr

D. all of these

Answer: stderr

12. Which acts as an interface between stream and hardware?

A. file pointer

B. buffer

C. stdout

D. stdin

Answer: buffer

13. Which function is used to associate a file with a stream?

A. fread()

B. fopen()

C. floes()

D. fflush()

Answer: fopen()

14. Which function returns the next character from stream, EOF if the end of file is reached, or if there is an error?

A. fgetc()

B. fgets()

C. fputs()

D. fwrite()

Answer: fgetc()

