BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Programming and Computer Applications-1

File Processing

Instructor : PhD, Associate Professor Leyla Muradkhanli

FILE

- A file is a container in computer storage devices used for storing data.

File Operations

- Creating a new file
- Opening an existing file
- Closing a file
- Reading from and writing information to a file

Working with files

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and the program.

FILE *fptr;

Opening a file - for creation and edit

Opening a file is performed using the fopen() function defined in the stdio.h header file.

The syntax for opening a file in standard I/O is:

ptr = fopen("fileopen","mode");

For example,

fopen(“D:\\program\\program.txt","w");

fopen(“D:\\program\\program.txt","r");

Mode	Description
`r`	Open an existing file for reading.
`w`	Create a file for writing. If the file already exists, discard the current contents.
`a`	Append; open or create a file for writing at the end of the file.
`r+`	Open an existing file for update (reading and writing).
`w+`	Create a file for update. If the file already exists, discard the current contents.
`a+`	Append: open or create a file for update; writing is done at the end of the file.
`rb`	Open an existing file for reading in binary mode.
`wb`	Create a file for writing in binary mode. If the file already exists, discard the current contents.
`ab`	Append; open or create a file for writing at the end of the file in binary mode.
`rb+`	Open an existing file for update (reading and writing) in binary mode.
`wb+`	Create a file for update in binary mode. If the file already exists, discard the current contents.
`ab+`	Append: open or create a file for update in binary mode; writing is done at the end of the file.

Fig. 11.6 | File opening modes.

```c
# include <stdio.h>
# include <conio.h>
# include <string.h>
int main( )
{
    FILE *fptr ;
    char data[50] = "My program in C language";
    fptr = fopen("test.txt", "w");
    if ( fptr == NULL )
    {
        printf("test.txt file failed to open.");
    }
    else
    {
        printf("The file is now opened.\n");
        if (strlen(data) > 0 )
        {
            fputs(data, fptr);
}
        fclose(fptr) ;
        printf("Data successfully written in file test.txt\n");
        printf("The file is now closed.");
    }
getch();
    return 0;
}
```

```c
# include <stdio.h>
# include <conio.h>
# include <string.h>
int main()
{
    FILE *fptr ;
    char data[50];
    fptr = fopen("text.txt", "r");
    if (fptr == NULL)
    {
        printf("text.txt file failed to open.");
    }
    else
    {
        printf("The file is now opened.\n");
        while(fgets(data, 50, fptr) != NULL)
        {
           printf("%s", data);
         }
        fclose(fptr);
        printf("\nData successfully read from file text.txt\n");
        printf("The file is now closed.") ;
    }
getch();
    return 0;
}
```

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

int main()
{
   int num;
   FILE *fptr;
   fptr = fopen("program.txt","w");
   if(fptr == NULL)
   {
      printf("Error!");

   }

   printf("Enter num: ");
   scanf("%d",&num);
   fprintf(fptr,"%d",num);
   fclose(fptr);
   getch();
   return 0;
}
```

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

int main()
{
    int num;
    FILE *fptr;
    if ((fptr = fopen("test.txt","r")) == NULL){
        printf("Error! opening file");
}

    fscanf(fptr,"%d", &num);
    printf("Value of n=%d", num);
    fclose(fptr);
  getch();
    return 0;
}
```

Random-Access Files

- In C language, the data stored in the file can be accessed in two ways:
- Sequential Access
- Random Access
- If we want to read access record in the middle of the file and if the file size is too large sequential access is not preferable. In this case, random access to file can be used, which allows to access any record directly at any position in the file.

C supports three functions for random access file processing:

fseek()

ftell()

rewind()

Random-Access Files

fseek():

This function is used for seeking the pointer position in the file at the specified byte.

Syntax: fseek(file pointer, displacement, pointer position);

Where

file pointer - It is the pointer which points to the file.

displacement - It is positive or negative. This is the number of bytes which are skipped backward (if negative) or forward(if positive) from the current position. This is attached with L because this is a long integer.

If we want to read the n^{th} record we will skip n-1 records with fseek() function and place the position pointer at the beginning of the n^{th} record.

Random-Access Files

Pointer position:

This sets the pointer position in the file.

Value	Pointer position
0	Beginning of file.
1	Current position
2	End of file

Random-Access Files

Examples:

1) fseek(p,10L,0)

0 means pointer position is on beginning of the file, from this statement pointer position is skipped 10 bytes from the beginning of the file.

2)fseek(p,5L,1)

1 means current position of the pointer position. From this statement pointer position is skipped 5 bytes forward from the current position.

3)fseek(p,-5L,1)

From this statement pointer position is skipped 5 bytes backward from the current position.

Random-Access Files

ftell()

This function returns the value of the current pointer position in the file. The value is count from the beginning of the file.

Syntax: ftell(fptr);

Where fptr is a file pointer.

rewind()

This function is used to move the file pointer to the beginning of the given file.

Syntax: rewind(fptr);

Where fptr is a file pointer.