Упражнение 5 – Файлове

Входно изходни потоци. Файлове. Работа с файлове.

Въпреки, че C няма дефинирани ключови думи за извършване на файлов вход/изход, стандартната библиотека на C съдържа богата колекция от входно/изходни функции, даващи доста гъвкави и ефикасни решения за работа с потоци от данни.

Какво е поток?

Входно/изходната система на C предоставя постоянен интерфейс на програмиста, който не зависи от използваното устройство. Това отделяне на хардуера от софтуера се нарича поток, а действително устройството което извършва вход/изход се нарича файл. Потокът се явява логически интерфейс към файла. Така дефиниран в C, терминът файл може да се отнася за дисков, екранен, лентов, клавиатурен файл или такъв в паметта, но без значение от източника, формата и възможностите на файла потоците са еднакви и автоматично се преодоляват различията, което значи, че за програмиста хардуерните устройства са еднотипни.

Потокът се свързва с файл чрез операция за отваряне, а освобождаването става чрез операция за затваряне.

Има два типа потоци: текстов и двоичен. Текстовият поток съдържа ASCII знакове, докато двоичният поток може да се използва със всякакъв тип данни. Не се извършват никакви знакови преобразование и предаването на информация е едно към едно между изпратеното от потока, и това което се съдържа във файла.

Текущо местоположение на файла

ако файла е голям 500 байта, и вече са прочетени 150 байта, следващият оператор за четене ще започне от 151 байт, тоест от мястото до последната извършена с файла операция.

Работа с фаилове

Асоцииране на файл ->

За да отворите файл и да го асоциирате с поток, използвайте fopen().

FILE *fopen(char *име_на_файл, char *режим);

Функцията fopen() се намира в библиотеката STDIO.H . Името на файла и точният път до него седят в име_на_файл. Това име трябва да е валидно файлово име, както е определено от операционната система.

Режим

r – Отваря текстов файл за четене
w – Създава текстов файл за запис
a – добавя към текстов файл, като ако не съществува го създава
rb/wb/ab прави горните операции но за двоичен файл (бинарен)
r+ -отваря текстов файл за четене/запис

хон. ас. инж. Валентин Бахтев, хон. ас. Момчил Петков,
София, 2013

w+ – Създава текстов файл за четене/запис
a+ -добавя към или създава текстов файл за четене/запис
r+b – отваря двоичен файл за четене/запис. Можете да използвате и rb+
w+b – създава двоичен файл за четене/запис. Можете да използвате и wb+
a+b – Добавя към или създава двоичен файл за четене/запис. Можете да използвате и ab+

Ако операцията fopen() е валидна се връща валиден файлов указател. Никога не бива да променяте този указател, или обекта, сочен от него.

Ако fopen() не успее да отвори файл се връща нулев указател.

ПРИМЕР:

```
FILE *fp;
If((fp = fopen(“myfile” , ”r”)) ==NULL){
prinf(“Error opening file.\n”);
exit(1); // или заменяте с ваш код за грешки
}
```

Особеност на режимите за работа е, че ако имате файл и го отворите с w то файла ще бъде изтрит и ще започне записването отначало в този файл. Ако използвате r+ няма да създаде файл, ако той не съществува, отварянето на файл с w+ отново би изтрило старото съдържание на файла.

Затваряне на файл – fclose()

Int fclose(FILE *fp);

Използвайки този оператор премахвате асоциацията между потока и файла. Информацията в системите се съхранява на сектори, и до запълване на нужното количество данни за запис на сектор информацията се складира в буфер. Чрез използване на fclose() автоматично записване от буфера на сектора. Функцията fclose() връща нула ако операцията е успешна, а ако се появи грешка връща EOF.

Функции за четене/запис на байтове (знаци)

Int fgetc(FILE *fp); -> ако не успее да прочете следващия бит връща EOF

Int fputc(int ch, FILE *fp); -> връща EOF ако се появи грешка.

Пример:

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    char str[80] = "This is a file system test. \n";
    FILE *fp;
    char *p;
```

```
    int i;

    /* отваря myfile за изход */
    if((fp = fopen("myfile", "w"))==NULL)
    {
        printf("Cannot open file. \n");
        exit(1);
    }

    /* записва str на диска */
    p = str;
    while(*p)
    {
        if(fputc(*p, fp)==EOF)
        {
            printf("Error writing file. \n");
            exit(1);
        }
        p++;
    }
    fclose(fp);

    /* отваря myfile за вход */
    if((fp = fopen("myfile", "r"))==NULL)
    {
        printf("Cannot open file. \n");
        exit(1);
    }

    /* чете отново файла */
    for(;;)
    {
        i = getc(fp);
        if(i == EOF) break;
        putchar(i);
    }
    fclose(fp);
    return 0;
}
```

Функции от високо ниво

fprintf() и fscanf() действат като scanf() и printf(), с изключение на това, че работят с файлове. Прототипите им са:

int fprintf(FILE *fp, char * форматиращ низ);

int fscanf(FILE *fp, char * форматиращ низ);

имат същите, вече познати свойства като printf() и scanf().

Функцията feof() дава индикатор дали сме стигнали до края на файла, който използваме.

Прототип:

Int feof(FILE *fp);

ПРИМЕР:

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

int main(int argc, char *argv[])
{
    FILE *fp;
    double ld;
    int d;
    char str[80];

    /* проверка за аргумент от командния ред */
    if(argc!=2)
    {
        printf("Specify file name. \n");
        exit(1);
    }

    /* отваряне на файл за изход */
    if((fp = fopen(argv[1], "w"))==NULL)
    {
        printf("Cannot open file. \n");
        exit(1);
    }

    fprintf(fp, "%f %d %s", 12345.342, 1908, "hello");
    fclose(fp);

    /* отваряне на файл за вход */
    if((fp = fopen(argv[1], "r"))==NULL)
    {
        printf("Cannot open file. \n");
        exit(1);
    }

    fscanf(fp, "%lf%d%s", &ld, &d, str);
    printf("%f %d %s", ld, d, str);
    fclose(fp);

    return 0;
}
```

Четене и записване на двоични данни

Колкото и да са полезни fprintf() и fscanf(), те не са най-ефективният начин за четене и запис на цифрови данни, поради факта че се преминава от двоичен формат в ASCII текст. Функциите fread() и fwrite() могат да четат всякакви типове данни, като използват техните двоични представяния. Прототипите им са:

size_t fread(void *buffer, size_t size,size_t num,FILE *fp);

size_t fwrite(void *buffer, size_t size,size_t num,FILE *fp);

fread() чете от файл *fp num броя обекти, всеки от тях с размер size байта, в буфер, сочен от buffer. Тя връща броя на прочетените обекти.

fwrite() е противоположна на fread() и записва вместо да чете, аналогично на fread();

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

int main(void)
{
    FILE *fp;
    int i;

    /* отваряне на файл за изход */
    if((fp = fopen("myfile", "wb"))==NULL)
    {
        printf("Cannot open file. \n");
        exit(1);
    }

    i = 100;

    if(fwrite(&i, 2, 1, fp) != 1)
    {
        printf("Write error occured. \n");
        exit(1);
    }
    fclose(fp);

    /* отваряне на файл за вход */
    if((fp = fopen("myfile", "rb"))==NULL)
    {
        printf("Cannot open file. \n");
        exit(1);
    }

    if(fread(&i, 2, 1, fp) != 1)
    {
        printf("Read error occured. \n");
        exit(1);
```

```
    }
    printf("i is %d", i);
    fclose(fp);

    return 0;
}
```

Произволен достъп

До момента записвахме или четяхме файл последователно от началото до края му (ако достигнем такъв). Може да имаме достъп до всяка позиция във файла чрез fseek()

Int fseek(FILE *fp, long отместване, int начало);

Стойността на отместване определя броя на байтовете след начало, които ще зададат новата текуща позиция. Начало може да бъде:

SEEK_SET - търси от началото на файла

SEEK_CUR – търси от текущата позиция

SEEK_END – търси от края на файла

Например ако искате да зададете текущото положение на 10 байта от началото на даден файл, тогава начало ще бъде SEEK_SET, а отместването – 10.

ftell() – определя текущата позиция на файла.

Int ftell(FILE *fp);

Ако се появи греша връща -1.

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

double d[10] = {10.23, 19.87, 1002.23, 12.9, 0.897, 11.45, 75.34,
0.0, 1.01, 875.875};

int main(void)
{
    long loc;
    double value;
    FILE *fp;

    if((fp = fopen("myfile", "wb"))==NULL)
    {
        printf("Cannot open file. \n");
        exit(1);
    }

    /* записване на целия масив в една стъпка */
    if(fwrite(d, sizeof d, 1, fp) != 1)
    {
        printf("Write error. \n");
```

```
        exit(1);
    }
    fclose(fp);

    if((fp = fopen("myfile", "rb"))==NULL)
    {
        printf("Cannot open file. \n");
        exit(1);
    }
    printf("Which element? ");
    scanf("%ld", &loc);
    if(fseek(fp, loc*sizeof(double), SEEK_SET))
    {
        printf("Seek error. \n");
        exit(1);
    }

    fread(&value, sizeof(double), 1, fp);
    printf("Element %ld is %f", loc, value);

    fclose(fp);

    return 0;
}
```

Преименуване на файл

Може да преименувате файл, като използвате rename() с прототип:

Int rename(char *старо име, char *ново име);

Функцията връща нула, ако е завършила и ненулев резултат, ако има грешка.

Изтриване на файл

Можете да изтриете файл чрез remove(), с прототип:

Int remove(char * име_на_файл);

Позициониране в нчалото на файл чрез използването на функция rewind() с прототип:

Int rewind(FILE *fp);