# **Работа с вход-изход. Текстови и двоични файлове**

# **1. Входно-изходни потоци. Файлове. Работа с файлове.**

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

# **2. Какво е поток?**

Входно-изходната система на С 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 С, терминът файл може да се отнася за дисков, екранен, лентов, клавиатурен файл или такъв в паметта, но без значение от източника, форматът 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 символи, докато двоичният поток може да се използва със всякакъв тип данни. Не се извършват никакви символни преобразувания и предаването на информация е едно към едно между изпратеното от потока, и това, което се съдържа във файла.

# **3. Работа с файлове**

**Асоцииране на файл:**

За да отворим файл, използваме функцията fopen().

Прототип:

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.**

**Режими на работа:**

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

r+ – отваря текстов файл за четене/запис

w+ – създава текстов файл за четене/запис

а+ – 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+

а+b – добавя към или създава двоичен файл за четене/запис. Можете да използвате и аb+

**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.**

# **4. Затваряне на файл – fclose()**

Прототип:

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(), секторът се записва автоматично от буфера. Функцията fclose() връща нула, ако операцията е успешна, а ако се появи грешка, връща EOF.

## **5. Текущо местоположение на файла**

Ако файлът 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.

**Функции за произволен достъп във файл – fseek() и ftell().**

Използвайки fseek(), може да достъпим определено място във файла.

Прототип:

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

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

SEEK\_SET – търси началото на файла

SEEK\_CUR – търси текущата позиция на курсора

SEEK\_END – търси 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.

Използвайки ftell(), можем да върнем текущата позиция на курсора във файла.

Прототип:

long int ftell(FILE \*stream)

Ако се появи греша връща -1.

# **6. Преименуване на файл**

Може да преименувате файл с функцията – rename().

Прототип:

int rename(const char \*old\_filename, const char \*new\_filename)

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

# **7. Изтриване на файл**

Можете да изтриете файл чрез remove().

Прототип:

int remove(char\* име\_на\_файл);

Позициониране в началото на файл чрез използването на функция rewind().

Прототип:

int rewind(FILE \* fp);

**8. Функции за четене на файлове**

|  |  |  |
| --- | --- | --- |
| **Функция** | **Текстов файл** | **Двоичен файл** |
| fgetc() | Прочита един символ.  При край на файла, връща EOF. | Прочита един байт.  При край на файла, връща -1. |
| fputc() | Записва един символ. | Записва един байт. |
| fgets() | Прочита ред до определен знак. | Не се поддържа. |
| fputs() | Записва низ. |
| fprintf() | Записва форматиран изход. |
| fscanf() | Прочита форматиран вход. |
| fread() | Прочита байтов блок в буфер. | |
| fwrite() | Записва байтов блок от буфер. | |

**Легенда:**

Оцветени клетки – могат да се използват за двата вида файлове, но е препоръчително да се използват с файловете, в чийто цвят са оцветени.

**9. Работа с текстови файлове**

**Работа с fgetc() и fputc().**

Прототипи:

int fgetc(FILE \*fp); int fputc(int ch, FILE \*fp);

Първата функция връща EOF, ако не успее да прочете следващия символ, докато втората връща EOF, ако се появи грешка.

Пример:

#include<stdio.h>

#include<stdlib.h>

int main()

{

    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 = fgetc(fp);

        if(i == EOF) break;

        putchar(i);

    }

    fclose(fp);

    return 0;

}

**Работа с fgets() и fputs()**

Прототип на fgets():

char \*fgets(char \*str, int n, FILE \*stream);

Функцията изисква пойнтер към стринга, където ще записваме текста, дължината на текста и файла, от който ще чете. Ако не успее да прочете, връща NULL пойнтер.

Пример:

#include <stdio.h>

int main () {

   FILE \*fp;

   char str[60];

   /\* opening file for reading \*/

fp = fopen("file.txt" , "r");

   if(fp == NULL) {

      perror("Error opening file");

      return(-1);

   }

   if( fgets (str, 60, fp)!=NULL ) {

      /\* writing content to stdout \*/

      puts(str);

   }

   fclose(fp);

   return(0);

}

Прототип на fputs():

int fputs(const char \*str, FILE \*stream)

Функцията изисква пойнтер към стринга, който ще записваме, и пойнтер към файла, в който ще записваме. Ако не успее запише, връща EOF, а ако успее, положително цяло число.

Пример:

#include <stdio.h>

int main () {

   FILE \*fp;

   fp = fopen("file.txt", "w+");

   fputs("This is c programming.", fp);

   fputs("This is a system programming language.", fp);

   fclose(fp);

   return(0);

}

**Работа с fscanf() и fprintf()**

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

int fprintf(FILE \*stream, const char \*format, ...)

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

Имат вече познатите свойства от scanf() и printf().

fprintf() - връща броя на успешно изписаните символи или -1, ако се появи грешка.

fscanf() – връща броя на успешно сканираните символи или 0, ако се появи грeшка.

**Бонус: 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;

}

**10. Работа с бинарни файлове**

**Функциите fread() и fwrite()**

Функциите fread и fwrite могат да четат всякакви типове данни, като използват техните двоични представяния. Прототипите им са:

size\_t fread(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream)

size\_t fwrite(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream)

Тъй като функциите са по-сложни, ще опишем и параметрите им.

fread()/fwrite():

* ptr – мястото, където ще записваме елементите/взимаме информацията
* size – големината на един елемент
* nmemb – брой елементи
* stream – потокът, от който четем/записваме

fread() връща броя на прочетените елементи, докато fwrite() връща броя на записаните елементи.

Пример:

#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;

}

**Предимството на работата с текстови файлове е четимостта за сметка на повечето памет, която заемат, и по-бавното им прочитане поради двойното им преобразуване. На по-ниско ниво разлика между текстови и бинарни файлове няма.**

**11. По-сложни примери**

В следващата програма ще демонстрираме запис и четене от двоичен файл на различни типове данни. Основната трудност при работата с двоични файлове идва от това, че да прочетем даден файл, е необходимо да знаем как този файл е записан. В предната задача записахме едно целочислено число, което е или 2, или 4 байта. Ако не знаем, че сме записали целочислено число, бихме прочели тези 2 или 4 байта последователно по един байт като отделни символи, което ще ни върне напълно безсмислен текст.

Целта на дадената програма е да запише във файл изреченията:

„My name is Ivan. I am 5 years old.” , като два отделни низа и целочислена стойност между тях.

#include<stdio.h>

#include<stdlib.h>

main()

{

    int a=5,b,i; FILE \*fp;

    char mas[]="My name is Ivan.I am "; char mas2[]=" years old!";

    char mas1[sizeof mas]; /\* масив с големината на масив mas \*/

    char mas3[sizeof mas2]; /\* масив с големината на масив mas2 \*/

    if((fp=fopen("myfile","wb"))==NULL) /\* отваря файл за запис\*/

    {

        printf("cant open the file"); exit(1);

    }

    if(fwrite(mas,1,strlen(mas),fp)!=strlen(mas)) /\*записва масива mas във файла\*/

    {

        printf("write error occurred"); exit(2);

    }

    if(fwrite(&a,sizeof(int),1,fp)!=1) /\* записва променливата а във фаила\*/

    {

        printf("write error occurred"); exit(3);

    }

    if(fwrite(mas2,1,strlen(mas2),fp)!=strlen(mas2)) /\*записва масива mas във файла\*/

    {

        printf("write error occurred"); exit(4);

    }

    fclose(fp);

    if((fp=fopen("myfile","rb"))==NULL) /\* отваря файл за четене\*/

    {

        printf("cant open the file"); exit(5);

    }

    if(fread(mas1,1,(sizeof mas1)-1,fp)!=((sizeof mas1)-1)) /\* чете от файла и

    записва в масива mas1 \*/

    {

        printf("read error occurred"); exit(6);

    }

    if(fread(&b,sizeof(int),1,fp)!=1) /\*чете от файла годините и записва в b\*/

    {

        printf("read error occurred"); exit(7);

    }

    if(fread(mas3,1,(sizeof mas3)-1,fp)!=(sizeof mas3)-1) /\* чете от файла и

    записва в масива mas3 \*/

    {

        printf("read error occurred"); exit(8);

    }

    fclose(fp);

    for(i=0; i<(sizeof mas1)-1; i++)  /\* отпечатва информацията последователно на екрана\*/

    {

        printf("%c",mas1[i]);

    }

    printf("%d",b);

    for(i=0; i<(sizeof mas3)-1; i++)

    {

        printf("%c",mas3[i]);

    }

    return 0;

}

Какъв е основният недостатък на тази програма? Когато четем от файла, ние четем дадените низове, като използваме това, че знаем тяхната дължина предварително. Основателен би бил въпросът възможно ли е да се напише програма, в която програмистът, който чете от файла, не знае предварително дължината на всеки низ, а само подредбата на данните във файла.

Пример:

Даден файл има следния формат: **int | string | double**

В този случай ние не знаем дължината на дадения низ. Съществуват два стандартни начина за решение на дадения проблем. Първият начин е когато записваме файла, да добавим едно поле от целочислен тип, което показва дължината на дадения низ.

Например: **int | length of string | string | double**

В този случай четем и записваме целочислена стойност, след това четем байтове на брой дължината на низа и накрая четем числена стойност с двойна точност.

При втория начин избираме един специален символ, който ще ни служи за край на низа, например „;“ . Дължината на низа се определя от избрания разделител. Този метод обаче е свързан с изрична забрана на дадения символ за използването му в низа. Недостатък на този метод е необходимостта от проверката на всички символи. Нека покажем един пример как би изглеждала една такава задача.

Да се реализира програма за четене и запис във файл чрез меню за избор. Да се реализират полета както следва: идентификационен номер на дадена стока, име на стока с технически характеристики , цена на дадената стока. Четенето от файл да се реализира, като по зададен идентификационен номер на екрана да се отпечатват всички полета на дадената стока.

#include<stdio.h>

void writening();

void reading();

main()

{

    char choise;

    do

    {

        printf("Enter 1 for writening in file and 2 for reading!\n"); choise=getchar();

        fflush(stdin);

    }

    while(choise!='1' && choise!='2');

  switch(choise)

    {

        case '1': writening(); break;

        case '2': reading(); break;

    }

    return 0;

}

void writening()

{

    int i,number,j; double price; char article[50]; FILE \*fp;

    printf("enter the string less than 50\n");

    gets(article);

    j=((sizeof(article)));

    printf("enter the number");

    scanf("%d",&number);

    printf("enter the price");

scanf("%lf",&price);

    if((fp=fopen("withIn","ab"))==NULL)

    {

        printf("the file cant open");

    }

    if(fwrite(&number,sizeof(int),1,fp)!=1)

    {

        printf("writening error");

    }

    if(fwrite(&j,sizeof(int),1,fp)!=1)

    {

        printf("writening error");

    }

    if(fwrite(article,sizeof (article),1,fp)!=1)

    {

        printf("writening error");

    }

    if(fwrite(&price,sizeof(double),1,fp)!=1)

    {

        printf("writening error");

    }

    fclose(fp);

}

void reading()

{

    int i,number,number1; char article[50]; double price;

    FILE \*fp;

    printf("enter the number");

    scanf("%d",&number1);

if((fp=fopen("withIn","rb"))==NULL)

    {

    printf("the file cant open");

    }

    for(;;)

    {

        if(fread(&number,sizeof(int),1,fp)!=1)

        {

            printf("reading error");

            break;

        }

        if(fread(&i,sizeof(int),1,fp)!=1)

        {

            printf("readung error");

            exit(1);

        }

        if(fread(article,1,i,fp)!=i)

        {

            printf("reading error");

            exit(7);

        }

        if(fread(&price,sizeof(double),1,fp)!=1)

        {

            printf("reading error"); exit(5);

        }

        if(number==number1)

        {

            printf(" %d ",number); printf(article);

            printf(" %g",price); break;

        }

    }

    fclose(fp);

}

**12. Задачи**

1.В двoичен файл имате цяло число N и след него N цели числа.

а)Да се напише програма, която създава файла.

б)Да се напише програма, която изкарва на екрана броя на четните и броя на нечетните цифри от така дадените N числа.

в)Да се напише програма, която прочита така подадения масив от N елемента и изкарва в текстови файл сортирания по възходящ ред масив.

2. В двоичен файл имате цели числа (не е указано колко на брой).

а)Да се напише програма, която създава файла.

б)Да се напише програма, която изкарва на екрана броя на четните и броя на нечетните цифри от така дадените N числа.

в)Да се напише програма, която прочита така подадения масив от N елемента и изкарва в текстови файл сортирания по възходящ ред масив.

3. Трябва да опишете база данни която работи с автокъща. Данните ви трябва да имат следния вид - марка на автомобил, модел на автомобила - 5 знака, обем на двигателя, цена, има ли регистрация или не (символ Y/N). Създайте програма, която записва така дадената структура на базата данни във двоичен и паралелно в текстови файл. Напишете възможност за добавяне на нов запис. Създайте програма, която прочита двата създадени файла и изкарва един след друг данните от бинарния и текстовия файл, за да направите сравнение за правилната работа на програмата ви.