ХИМИКОТЕХНОЛОГИЧЕН И МЕТАЛУРГИЧЕН УНИВЕРСИТЕТ – СОФИЯ

ИНФОРМАТИКА

част втора

Основи на програмирането на C++

лектор: гл. ас. д-р Стефан М. Панов

Катедра “Информатика”

Лекция 5 (13)

Обработка на файлове в C++

(C файлова система)

В езика C++ като супермножество на C са налични две входно-изходни системи. Те са известни като C++ входно-изходна (В/И) система и C-ориентирана В/И система. Първата се поддържа посредством заглавния файл **<iostream>**, а втората от C заглавния файл **<stdio.h>**, известен в C++ под името **<cstdio>**. Веднага ще добавим, че поне за dev-C++ среда **<iostream>** съдържа и декларациите, намиращи се в **<cstdio>** - оттук и изводът, че **<cstdio>** може и да не се включва в нашите програми, доколкото **<iostream>** винаги присъства.

Потоци в C++

И двете В/И системи опират до понятието **поток (stream)**. Потокът е **общ** логически интерфейс с различните устройства, съставящи компютъра. Потокът се свързва с което и да е физическо устройство с помощта на В/И система. При това поведението на всички потоци е еднакво, независимо от различията между устройствата. Еднаквото поведение означава, че

практически към всички типове устройства можем да приложим едни и същи функции и В/И оператори. Например, методите, използвани за запис на данните на екран, могат да ползват и за извеждането им на принтер или за запис на дисков файл. Затова още по-обобщено може да кажем, че **поток е логически интерфейс с файл**. В C++ по определение термина "файл" се отнася към дисков файл, екран, клавиатура, порт, файл на флашка, SD карта, магнитна лента и т.н.

Потокът се свързва с файла при изпълнение на операцията отваряне на файл и се разделя с него с помощта на операцията затваряне на файл.

Съществуват два типа потоци: текстов и двоичен (binary). Текстовият поток се използва за въвеждане и извеждане на символи. При това могат да настъпят и някои преобразувания на символите. Например, при извеждане на символ за нов ред той може да бъде преобразуван в последователност от символи: връщане на курсора и преминаване към нов ред (в ОС MS

Windows). Затова не можем да кажем, че е налице взаимно-еднозначно съответствие между това, което се изпраща в потока, и това, което в действителност се записва във файла. Двоичният поток може да се ползва с данни от произволен тип, при това не се извършва никакво преобразуване на символи - между това, което се изпраща в потока и това, което реално се съдържа във файла, съществува взаимно-еднозначно съответствие.

Говорейки за потоци сме длъжни да уточним, какво се влага в понятието "**текуща позиция**". Това е мястото (адресът) във файла, от което ще се изпълнява следващата операция за четене/запис от/във файла.

В **C++** В/И система се съдържат 4 вградени потока (**cin, cout, cerr и clog**), които автоматично се отварят когато програмата започне да се изпълнява. Вече познатият cin е стандартният входен, а cout — стандартният изходен поток. Потоците cerr и clog са предназначени за извеждане на информация за грешки и също са свързани със стандартното извеждане на данни. По

подразбиране стандартните C++ потоци са свързани с конзолата, но програмно могат да се пренасочат към други устройства или файлове. Но защо има две В/И системи? Отговорът е – С ориентираната В/И система не обезпечава никаква поддръжка за обектите, дефинирани от потребителя. Т.е. при истинското ООП (обектно-ориентирано програмиране) само C++ В/И система предоставя възможности операторите >> и << да работят и с класове (с обектите от дадените класове).

С ориентирана В/И система

Също се базира на понятието поток. Има 3 предварително определени текстови потока **stdin, stdout и stderr**. Наричат ги съответно стандартен входен поток, стандартен изходен поток и стандартен поток на грешките. Тия потоци представляват С версиите съответно на cin, cout и cerr. По подразбиране **stdin** е свързан с клавиатурата, а другите два с екрана. Функциите в С В/И система, които са свързани с въвеждането (основна scanf)

и извеждането (основна функция printf) на данни съответно от клавиатурата и на екран няма да се разглеждат.

С-система за обработка на файлове

Състои се от няколко взаимносвързани функции. При работа с файлове се използва специална структура, дефинирана в stdio.h, за управление на файлове с тип **FILE**. (В някои учебници се нарича файлов дескриптор, в други – **блок за управление на файл**.) В структурата има информация за файла, която включва неговото име, статус и текуща позиция (заедно с други неща). Винаги се работи с указател към тая структура. По същество, файловият указател идентифицира конкретен (дисков) файл и съобщава на всички входно-изходни С-функции, къде те трябва да изпълняват операциите. Преди да се пристъпи към изучаване на конкретните функции за работа с файлове ще се разгледат аргументите на функцията main().

Аргументи на функцията main () – argc и argv

Понякога възниква необходимост да се предаде информация към програмата при нейното пускане. Това става с подаване на аргументи към функцията

main(). Когато програмата се пуска за изпълнение в работната среда, обикновено зависи от конкретната среда как точно става подаването на аргументите. За dev-C++ се избира от основното меню:

Execute-Parameters-Parameters to pass to your program

и там се попълват фактическите параметри **преди** пускане на програмата. Когато командата се изпълнява от командния ред аргументите (фактическите параметри) се задават след името на програмата. Примерно, командата:

prog66 "Nezakonna sech.txt" "Nova sech.txt"

ще стартира програмата prog66.exe с два аргумента, зададени като символни низове. Разширението на програмата ".exe" може да се пропусне, но и prog66.exe също е позволено. Самата команда се изпълнява от операционната система.

В C++ за функция **main()** са дефинирани два вградени, но незадължителни параметъра, **argc** и **argv**, които получават своите стойности от аргументите на командния ред.

Заб. Формално за имена на параметрите могат да се изберат произволни други идентификатори, но имената argc и argv се употребяват по общо съгласие от доста години. Затова няма смисъл да се прибягва до други идентификатори - нека всеки програмист, който ще трябва да вниква в нашите програми, да може бързо да идентифицира argc и argv като параметри на командния ред.

Параметърът **argc има целочислен тип и е предназначен для съхранение на броя на аргументите на командния ред**. Стойността му винаги е най-малко 1 защото името на програмата също е аргумент, при това първият. **Параметърът argv е указател към масив от символни указатели.** Всеки указател в масива argv сочи към низ, явяващ се аргумент на командния ред. Елементът argv[0] сочи към името на програмата, **като в името се включва и пътят към програмата, когато тя е извикана в dev-C++**; елементът argv[1] — към първия аргумент, argv[2] — към втория и т.н. Всички аргументи на командния ред се предават на програмата **като низове, затова числовите аргументи трябва да се преобразуват в програмата в съответния вътрешен формат** с помощта на някоя от вече разгледаните функции atoi, atol и atof.

Обикновено argv се обявява така: **char *argv[ ];**

Достъпът в програмата към отделните аргументи на командния ред се получава чрез индексиране, както при всеки друг масив. **Въпреки, че зависи**

от средата за изпълнение обикновено аргументите се отделят един от друг с интервал или с табулация. Например реда: edno, dwe i tri

се състои от 4 низови аргумента, докато реда: edno,два, три
само от два, доколкото запетаята не е допустим разделител.

Когато е необходимо да се подаде като един аргумент на командния ред низ, съдържащ интервали той се поставя в двойни кавички. (Както беше направено с аргументите на prog66 в началото на настоящия раздел).

Да се има предвид, че дадените тука примери макар и валидни за много среди не са в сила за всички. Например в нашата dev-C++ среда (поне във версията, която е използвана за тестване на написаните програми) има проблем с аргументи, които съдържат интервали, но същият проблем липсва, когато програмата се дебъгва!

За да получим достъп към отделен символ в аргумент на командния ред се добавя втори индекс в **argv.** Така е направено в следващата програма, която посимволно извежда всички аргументи, с които тя е била извикана. Всеки аргумент се извежда на отделен ред.

```
#include <iostream> // nashata 67-a programa
using namespace std; // programata izvejda na ekran posimvolno
int main(int argc, char *argv[ ]) { // argumentite na komandnia red
      int i; // pozicia v tekushtia argument
      for(int t=0; t<argc; ++t) { // t.e. za vseki argument
            i = 0;
            while(argv[t][i]) { // dokato ne e dostignat kraia na niza
                  cout << argv[t][i];
                  ++i;
            }
            cout << '\n';
      }
      return 0;
}
```

Както вече се спомена, когато има аргументи, съдържащи интервал, програмата работи по различен начин, ако се пуска от средата и от командния ред заради особености на dev-C++ средата. Има разлика и за първият аргумент с индекс 0, дали присъства пътят към програмата или не.

Функция fopen () - изпълнява три задачи:

1. Отваря поток.

2. Свързва файл с потока.

3. Връща указател от типа **FILE** към тоя поток.

Може да се каже, че обектът от тип **FILE** се явява поток! Това е общият поток за всички C функции за обработка на файлове.

Прототипът на функцията е:

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

Параметърът-указател filename сочи към името на файла, който предстои да бъде отворен, а параметърът mode— към низа, съдържащ нужния статус

mode	Значение
"r"	Отваря текстов файл за четене
"w"	Създава текстов файл за запис
"a"	Отваря текстов файл за запис в края на файла
"rb"	Отваря двоичен файл за четене
"wb"	Създава двоичен файл за запис
"ab"	Отваря двоичен файл за запис в края на файла
"r+"	Отваря текстов файл за четене и запис
"w+"	Създава текстов файл за четене и запис
"a+"	Отваря текстов файл за четене и запис в края на файла
"r+b"	Отваря двоичен файл за четене и запис
"w+b"	Създава двоичен файл за четене и запис
"a+b"	Отваря двоичен файл за четене и запис в края на файла

Таблица 13.1 – разрешените стойности за параметъра mode

(режим) на отваряне на файла. Възможните стойности на mode са показани в приведената табл. 13.1. Парам. filename може да включва и спецификацията на пътя към файла. Пътят може да съдържа и име на устойство, пр. "C:\... ".
Да не се забравя, че в низа самият символ "\"се указва с "\\".

Както се вижда от таблицата, файлът се отваря или в текстови, или в двоичен режим. Ако функцията fopen отвори успешно зададения файл, тя връща указател от тип FILE. Той идентифицира файла и се използва от почти всички други файлови системни функции. Ако файлът не може да се отвори се връща NULL. Най-често отварянето на файл става с код подобен на следния:

```
If ((fp = fopen("test", "w"))==NULL) {cout("Failyt ne moje da se otvori!"); exit(1);}
```

Тоест, първо се проверява дали възниква грешка при отварянето на файла (примерно, ако дискът е пълен или е защитен срещу записване) и само ако всичко е наред се правят следващи операции, примерно запис във файла.

Когато в режима присъства символът "w" това означава, че ако съществува файл със същото име той ще бъде унищожен и вместо него ще се създаде нов празен файл. Ако файлът не съществува отново се създава нов файл. Когато е указан символът "r" файлът трябва да съществува, иначе ще възникне грешка. Когато е указан символът "a" файлът ще се създаде, ако не съществува.

Функция fputc() – записва един символ в потока, предварително отворен с fopen. Прототипът на функцията има вида:

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

където fp е файлов указател, върнат от функцията fopen(), а ch —символът, който ще се записва. Независимо от това, че параметърът ch има тип int, в него се използва само младшият байт. При успешен запис fputc () връща като резултат записания символ, в противен случай се връща EOF. Стойността на EOF е определена в заглавния файл <iostream>, най-често е -1.

Записът се осъществява в текущата позиция на файла. Самата позиция се указва от т. нар. „вътрешен индикатор на позицията във файла“ (ИПФ). Разбира се, текущата позиция на ИПФ нараства съответно при успешен запис (както стойността на указателите при операцията ++).

Функция fgetc() – чете един символ от потока, предварително отворен с fopen. Прототипът на функцията има вида:

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

Отново fp е файлов указател, върнат от функцията fopen(). Независимо, че върнатата стойност е от типа int, нейния старши байт е равен на нула. Четенето се осъществява от текущата позиция на файла.

При възникване на грешка или достигане на края на файла функцията fgetc() връща стойност EOF. Следва код, с който можем да прочетем съдържанието на текстови файл до края: **ch = fgetc(fp); while(ch!=EOF) { ch = fgetc(fp); }**

Функция feof() – проверява дали е достигнат края на файла при четене от файл. Прототипът на функцията има вида `int feof(FILE *fp);`

Ако краят на файла е достигнат се връща ненулева стойност, иначе се връща нула. Ако анализираме примера, даден с предишната функция лесно можем да видим, че за двоичен файл цикълът може да спре и преди достигане истинския край на файла - когато прочетената двоична стойност е равна на **EOF**. Именно за избягване на подобна ситуация служи feof. Със следващата инструкция може да се прочете съдържанието както на текстов, така и на двоичен файл: `while (!feof(fp)) ch = fgetc(fp);`

Функция fclose() – затваря потока, който е бил отворен с fopen(). Нейният прототип има вида: `int fclose(FILE *fp)`

При успешно затваряне функцията връща 0, иначе EOF. Опит да затворим вече затворен файл се оценява като грешка. Грешка се генерира и ако се

опитаме да затворим файл когато носителят на данните (примерно флашка) е премахнат преди това. При извикване на fclose се освобождава блока за управление на файла, сочен от fp. Следователно, същият блок ще може да се използва за отваряне на друг файл. Тук е мястото да споменем, че във всяка ОС съществува ограничение на количеството файлове, които могат да бъдат отворени едновременно. Оттук и препоръката да се затварят файловете, когато те вече не се необходими.

На следващата страница е даден пример (prog66) за копиране на съдържанието на един файл в друг, в който са използвани всички разгледани дотук функции. Обърнете внимание, че файловете се отварят в двоичен режим. Така можем да копираме и двоични, и текстови файлове.

Функции ferror () и rewind()

Функцията ferror() се използва за проверка дали е възникнала грешка при изпълнение на файлова операция. Нейният прототип има вида:

int ferror(FILE *fp); // ferror често се нарича „**индикатор за грешка**“

Функцията връща стойност true, ако при изпълнение на **последната** файлова операция е възникнала грешка; в противен случай — стойност false. Доколкото възникването на грешка е възможно при изпълнение на която и

```
#include <iostream> // nashata 66-ta programa
using namespace std; // kopirane na sydyrjanieto na edin fail v drug
int main(int argc, char *argv[ ]){ // da se pusne i ot command prompt
      FILE *in, *out;  char ch;
      if (argc!=3) {
            cout << "Vyvedete i imenata na failovete kato parametri.\n";
            return 1;
      }
      if ( ( in=fopen(argv[1], "rb")) == NULL) {
            cout << "Vhodniat fail ne moje da se otvori"; return 1;
      }
      if( (out = fopen(argv[2], "wb")) == NULL) {
            cout << "failyt priemnik ne moje da se otvori.\n"; return 1;
      }
```

```
while ( !feof (in)) {
      ch = fgetc (in);
      if ( !feof ( in)) fputc(ch, out);
}
fclose(in); fclose (out);return 0; }
```

да е операция с файл, е необходимо ferror() да се извиква веднага след всяка функция за обработка на файлове; в противен случай информацията за грешката може просто да се изгуби.

Функцията rewind() премества ИПФ в началото на файла, зададен като аргумент. Прототипът ѝ изглежда така: void rewind(FILE *fp);

Функция fgets() – чете низ от потока. Прототипът на функцията е:

char * fgets (char * str, int num, FILE * fp);

str – указател към масив от символи, където се копира прочетеният низ.

num – максимален брой символи, които трябва да бъдат копирани в str (включително нулевият символ '\0' за край на низа).

fp - указател към потока (блока за управление на файла).

fgets чете символи от потока и ги запомня като низ в str докато се прочетат (num-1) символа. Четенето се прекратява по-рано, ако е достигнат или символът за нов ред или краят на файла. Символът за нов ред се добавя към низа. При всички случаи накрая в str се добавя нулевият символ '\0' за край на низ.

При успех, функцията връща str. Ако възникне грешка, се връща NULL, а ferror се подготвя да върне true. В зависимост от това, кога възниква грешката съдържанието на масива, сочен от str може да бъде променено.

Функция fputs() – записва низ в потока. Прототипът на функцията е:

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

str – указател към низ, който трябва да се запише в потока.
fp - указател към потока (блока за управление на файла).

fputs копира символите от низа в потока докато се достигне нулевият символ '\0' за край на низ. Самият нулев символ не се записва в потока.

При успех, функцията връща неотрицателна стойност. Ако възникне грешка, се връща EOF, а ferror се подготвя да върне true.

Следва условието на задачата, където ще бъде показана употребата на fgets и fputs:

Да се състави програма, която приема като входен параметър текстов файл и връща орязана версия на същия файл, като запазва само последните му **n** реда, където **n** също е входен параметър. Новата версия на файла да има същото име както началния файл плюс добавяне на разширението ".txt ".

Избрано е решение, където към main() да се подават 4 аргумента, като пред името на файла се намира аргумента "-f", а пред новия брой на редовете аргумента "-n". В случай, че ползващият програмата не е задал 4 аргумента

(т.е. общият брой аргументи не е 5) или аргументите "-f" и "-n"не са зададени правилно програмата ще спира работа като ще изведе помощно съобщение как трябва да се извиква програмата. Същото съобщение ще се вижда и ако не е зададен правилен брой на редовете, като под правилен брой ще разбираме всяко цяло положително число.

Алгоритъмът на програмата трябва да намери броя на редовете на входния файл. Това намиране в самата програма става с функцията fgets. Когато тоя брой е по-голям от новия брой редове се прескачат нужния брой редове на началния файл, а останалите се копират с fputs в орязаната версия на файла.

Преди да се чете нов ред в буфера е добре той да се изчисти, а това най-лесно става като запише нулевият символ в самото му начало.

```
#include <iostream> // programata chete tekstow fail i zapiswa w now fajl
#include <string.h> // poslednite n reda ot pyrvia fail. Noviat fail ima syshtoto
#include <stdlib.h> // ime kato nachalnia + razshirenieto ".txt"
using namespace std;
int cutme (char* filename, long int newNumOfLines){
	FILE *f1, *f2;  long int i;
	long int ulNumOfLines = 0;
	char buffer[1024];  char newfilename[512];
	if ((f1 = fopen(filename,"r")) == 0) {
cout << "file "<< filename << " could not be opened!  File is not truncated.\n";
		return -1;
	}
// niama nujda ot sledvashtiat red - pri otvariane se sochi nachaloto na faila
//	rewind(f1);
	while (!feof(f1)) { // calculate number of lines of a text file
		buffer[0] = '\0'; //  strcpy(buffer, "");
		fgets(buffer, sizeof(buffer),f1); ulNumOfLines++;
	}
	cout << "num of lines = "<<ulNumOfLines<<"\n";
	if (newNumOfLines >= ulNumOfLines){
```

```
            fclose(f1); return 0;// no need to resize, stop
        }
  rewind(f1); //fseek( f1, 0L, SEEK_SET);// go to the first line of truncated file
        for (i = 0; i< ulNumOfLines - newNumOfLines;i++){
            buffer[0] = '\0'; fgets(buffer, sizeof(buffer),f1);
        }
        strcpy(newfilename, filename); strcat(newfilename, ".txt");
        cout << "newfilename="<<newfilename<<endl;
        if ((f2 = fopen(newfilename,"w+")) == 0) { // open new file
cout << "new file could not be created!\nFile "<<filename<<" is not truncated\n";
            fclose(f1); return -1;
        }
        for (i = 0; i< newNumOfLines;i++){
            buffer[0] = '\0'; // sledva kopiraneto
            fgets(buffer, sizeof(buffer),f1);
            fputs(buffer,f2);
        }
        fclose(f1); fclose(f2);
        return 0;
} // cutme
```

```
void printUsage(){
	cout<<"Usage:\n\ncutme -f <filename> -n <num_of_lines>\n\n"<<
		"where <num_of_lines> is number of last lines that will be kept\n"<<
		"in the new truncated version of the text file <filename>\n";
	return;
}
int main(int argc,  char* argv[]) {
	if (argc < 5 || (strcmp(argv[1],"-f") || strcmp(argv[3],"-n"))) {
		printUsage(); return 0;
	}
	long int numOfLines = atol(argv[4]);
	if (numOfLines <= 0L){
		printUsage(); return 0;
	}
	cutme (argv[2], numOfLines);
	return 0;
}
```

Функции fread() и fwrite() – за четене/запис на блокове от данни.

Прототипите им са:

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

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

При извикване на функцията fread() параметърът buffer представлява указател към памет, предназначена за съхранение на данните, прочетени от файла. Функцията трябва да прочете count обекти, всеки с дължина numbytes от потока, указан с fp. Функция fread() връща броя прочетени обекти, който може да се окаже по-малък от зададената стойност на count, ако по време на изпълнение на функцията възникне грешка или е бил достигнат края на файла.

При извикване на функцията fwrite() параметърът buffer представлява указател към данните, които подлежат на запис във файла. Функцията

записва count обекти, всеки с дължина numbytes в потока, указан с fp. Връщаната от функцията стойност е действителният брой записани успешно обекти, който брой е равен на стойността на count, ако по време на изпълнение на функцията не е възникнала грешка. Когато файлът, с който се работи е бил отворен в двоичен режим fread() и fwrite() могат да четат или записват данни от произволен тип.

Функции fseek() и remove()

Функцията **remove** изтрива зададения файл и има следния прототип:
int remove(const char *filename); Тя връща 0 при успешно изтриване на файла и друга стойност в противен случай.

Функцията fseek() установява индикатора на позицията във файла (ИПФ) и има следния прототип:

int fseek(FILE *fp, long numbytes, int origin);

Както и за другите функции fp е указател към блока за управление на файла, а numbytes е броя байтове относно позицията, зададена с параметъра origin. Стойностите, които може да приеме параметърът origin са само 3:

SEEK_SET – търсене от началото на файла
SEEK_CUR – търсене от текущата позиция на файла
SEEK_END – търсене от края на файла

Например, следната инструкция ще премести ИПФ на 174-я байт във файла:

```
fseek(fp, 174L, SEEK_SET);
```

докато `fseek(fp, -90L, SEEK_END);` ще го премести 90 байта преди края му.

При успешно изпълнение на функцията тя връща 0 и друга стойност в противен случай. Препоръчва се функцията да се ползва за файлове, отворени в двоичен режим.

Примери с използването на fread(), fwrite() и fseek()

Ще добавим нова функционалност към голямата програма от лекция 12, включваща файлови операции. В този раздел ще се покаже само новият и промененият код. Трите нови функции водят до промяна и на printMenu(), която в новия си вид връща цяло число вместо символ.

```
unsigned short printMenu(){
        char buffer[32];
        short k;
        do {
             system("cls");
             cout <<"1. Izvejdane na ekran\n";
             cout <<"2. Izchisliavane na sreden uspeh\n";
             cout <<"3. Podrejdane po familia\n";
             cout <<"4. Podrejdane po sreden uspeh\n";
             cout <<"5. Tyrsene po familia\n";
             cout <<"6. Tyrsene po sreden uspeh\n";
```

```
        cout <<"7. Dobaviane na nov student\n";
        cout <<"8. Obnovi inf. za student\n";
        cout <<"9. Premahni inf. za student\n";
        cout <<"10. Zapis na masiva vyv fail\n";
        cout <<"11. Chetene na dannite ot fail\n"; // novata funkcionalnost
        cout <<"12. Chetene na edin zapis ot faila\n"; // v cherverno
        cout <<"50. Krai na programata\n\n";
        cout <<"Vashiat izbor: ";
        gets(buffer);
        k = atoi(buffer);
        if (k==50) { // kakvo shte byde sled while bez toia if ?
              break;
        }
    }while (k<=0 || k > 12);
    cout << endl;
    return k;
} // printMenu
```

Функцията main() също ще претърпи изменения: Типът на променливата ch сега е unsigned short, както върнатата стойност в printMenu. Добавят се 3 нови

```
case 10: writeInFile(p2, numOfElements) ; break;
case 11: numOfElements = readFromFile(p2) ;break;
case 12: readOneRecordFromFile(p2) ;break;
```

случая в switch за новите функции, а цикълът завършва при while (ch !=50);

Последната промяна е в условието на инструкцията if, новият му вид е:

```
if (!numOfElements && ch != 7 && ch != 50  && ch != 11  && ch != 12){
```

защото при празен масив можем да четем записи от файла с новите функции.

В показаната по-надолу функция writeInFile са дадени два варианта на работа с fwrite().

```
void writeInFile(struct  student *p, int numOfElements) {
	FILE *f1; char filename[256];
//	strcpy(filename, "Grupa_35");
//	strcpy(filename, "D:\\Stefan\\HTMU\\Study\\Lekcii_srs\\Grupa_34");
	cout <<"Vyvedete ime na faila, v koito shte pishem.\n"
	  << "Mojete da ukavete i pytia pred imeto!\nfilename=";
	cin.sync();   gets(filename);
	if ((f1 = fopen(filename,"w+b")) == 0) {
		cout << "file "<< filename << "  ne moja da byde otvoren!Stop.\n";
		cin.sync(); getchar(); return;
	}
	cout<<"ot writeInFile: Shte zapishem vyv faila "<<numOfElements<< "
zapisa"<<endl;
//	for (int k=0;k<numOfElements;++k){
//		fwrite((p+k), sizeof(student_t), 1, f1);// !! samo edin zapis
		fwrite(p, sizeof(student_t), numOfElements, f1);// wsichki zapisi nakup
//	}
	fclose(f1); cin.sync(); getchar();
	return;
 } // writeInFile
```

Подобно, в readFromFile() са дадени два варианта на работа с fread().

```
unsigned int  readFromFile(student *p) {
	FILE *f1;
	student_t el; // buffer, v koito chetem edin zapis
	char filename[256]; // goliamo, zashtoto moje da vkliuchi i pytia kym faila
	int numOfElements=0; // za broiat na zapisite
	short checkIt;
//strcpy(filename, "Grupa_35"); // vmesto vyvejdane ot klaviatura, tekushta dir
//	strcpy(filename, "D:\\Stefan\\HTMU\\Study\\Lekcii_srs\\Grupa_36");
	cout <<"Vyvedete ime na faila, ot koito shte chetem.\n"
	  << "Mojete da ukavete i pytia pred imeto!\nfilename=";
	cin.sync(); gets(filename);
//  \Stefan\HTMU\Study\Lekcii_srs\Grupa_36
//  \\Stefan\\HTMU\\Study\\Lekcii_srs\\Grupa_36
// C:\\Siemens\\Grupa_33 - moje i da ne raboti s edno \, zatowa \\
	if ((f1 = fopen(filename,"r+b")) == 0) {
		cout << "file "<< filename << "  ne moja da byde otvoren!Stop.\n";
		cin.sync();
		getchar(); return 0;
```

```
        }
        init_list(p, 0);// Inicializirame masiwa predi da go popylvame!
// pyrvi variant, navednyj, shte prochete kolkoto ima
//      numOfElements = fread(p, sizeof(student_t), N, f1);
// vtori variant s cikyl
        while ( !feof (f1)) {
// tuk ima osobenost, che sled kato sa procheteni vsichki zapisi ne e dosignat
// kria na fajla i opitva da chete oshte edin zapis za koito shte wyrne checkIt=0
                checkIt = fread(&el, sizeof(student_t), 1, f1);
                if (checkIt==1){
                        *(p+numOfElements++)=el;
                }
        }
        fclose(f1);
        cout <<"ot readFromFile: Procheteni sa "<<numOfElements<< "
                zapisa"<<endl;
        getchar(); return numOfElements;
 } // readFromFile
```

Последната функция чете един запис от файла, като от клавиатура се въвежда номерът на записа, който искаме да прочетем. Тъй като функцията atoi връща 0 при грешка е решено, ако искаме да прочетем първия запис да въвеждаме единица, за втория 2 и т.н. Позиционирането на нужния запис става с fseek, като отчитаме, че първият запис има номер 1 . Само ако записът е успешно прочетен се извиква printOneStudent(), за да се покаже съдържанието му на екрана.

```
void readOneRecordFromFile(student *p) {
    FILE *f1;
    student_t el; // tuka shte zapishem prochetenia zapis
    int rez; // za proverka
    unsigned int numOfRec; // nomera na zapisa, koito shte prochetem
    char buffer[32];
    char filename[256];
    cout <<"Vyvedete ime na faila, ot koito shte chetem.\n"
      << "Mojete da ukavete i pytia pred imeto!\nfilename=";
    cin.sync();
```

```
        gets(filename);
//      strcpy(filename, "Grupa_35"); // ako e v tekushtata na programata papka
//  \\Stefan\\HTMU\\Study\\Lekcii_srs\\Grupa_33
// C:\\Siemens\\Grupa_33 \\ tuk pytiat e na drugo ustroistvo
        if ((f1 = fopen(filename,"r+b")) == 0) {
                cout << "file "<< filename << " ne moja da byde otvoren!Stop.\n";
                cin.sync();
                getchar();
                return;
        }
        do {
                cout <<   "Vyvedete nomer na zapisa, koito iskate da prochetete.\n";
               cout <<   "Pyrviat zapis ima nomer 1 (a ne 0).\n\nNomer na zapisa=";
                gets(buffer); numOfRec = atoi(buffer);
                if (numOfRec < 1){
                        system("cls");
                }
        }while (numOfRec < 1) ;
        rez = fseek( f1, (numOfRec-1)*sizeof(student_t), SEEK_SET);
        if (rez == 0){
```

```
        rez = fread(&el, sizeof(student_t), 1, f1);
        if (rez == 1){
          cout << "Procheten e uspeshno zapis nomer "<<numOfRec<<"!\n\n";
          printOneStudent (&el);
        }else{
              cout << "Vyveli ste nevaliden nomer na zapis!\n";
        }
    }
    else{
        cout << "Vyveli ste nevaliden nomer na zapis!\n";
    }
    fclose(f1);
    cin.sync(); getchar();
    return;
} // readOneRecordFromFile
```