Обектно-ориентирано програмиране
(Информатика, Информационни системи)
2021/ 22 ФМИ, СУ

Файлове (продължението)
До сега за файловете като механизъм за *устойчиво* съхраняване и извличане на информация…

Файл. Представяне.
Файлът е именувана редица от данни (байтове), които са подредени последователно. Може да се счита, че номерацията им започва от 0 (подобно на масивите) и завършва с маркер за край на файл.

Типове файлове.
Файловете могат да се класифицират по няколко признака.

Според **операциите**, които могат да бъдат извършвани върху файловете, различаваме:

- файлове само за *писане*;
- файлове само за *четене;*
- файлове *и за четене, и за писане*.

Според **начина на достъп** до данните различаваме:

- файлове с последователен достъп;
- файлове с пряк достъп.

Според **типа на данните** различаваме:

- текстови файлове;
- двоични файлове.

Входно-изходни операции от гледна точка на нашата програма.
Когато данни се съхраняват програмно във файл, те се пренасят от оперативната памет към файла. Това се нарича още *изходна операция* за една програма.

Когато данни от файл се четат програмно, те се пренасят от файла към оперативната памет. Това се нарича още *входна операция* за една програма.

В C++ съществува понятието *поток*. Потокът е абстракция, с която описваме връзката между програмата и даден файл или устройство. За потока може да се мисли като за канал за комуникация, канал, по който данните протичат в определена посока (от паметта към файла или обратно).

Как да се направи връзката между поток и файл? От какъв тип трябва да бъде потокът?

Файлови потоци
В езика C++ има три вида файлови потоци, които могат да се използват след включването на заглавния файл `<fstream>` (`съкратено от file stream`):

- `ifstream` (`съкратено от input file stream`) предназначен за връзка с файлове, от които се четат (извличат) данни;
- `ofstream` (`съкратено от output file stream`) предназначен за връзка с файлове, в които се записват данни;
- `fstream` (`съкратено от file stream`) предназначен за връзка с файлове, с които ще се извършват както входни, така и изходни операции, т.е. файлове за четене и писане.

Как се отваря файл за записване?

При записване на информация във файл, потокът на данните е от оперативната памет към файла, т.е. операцията е *изходна* за програмата.

Необходим е *изходен файлов поток*.

```
std::ofstream outputFile;
outputFile.open("hello.txt");
// или директно след името на потока
std::ofstream outputFile("hello.txt");
```

Файлът може да се укаже с *относителен* или *абсолютен* път.

Изходният поток, който се създава и свързва с файла, може да носи произволно, валидно за езика име, подобно на всяка една променлива, която се създава.

Как се отваря файл за четене?

При извличане (четене) на информация от файл, потокът на данните е от файла към оперативната памет, т.е. операцията е *входна* за програмата.

Необходим е *входен файлов поток*.

```
std::ifstream inputFile;
inputFile.open(“hello.txt”);
```

Променливата за входния поток, който се създава и свързва с файла, може да носи произволно, валидно за езика име.

Проверка дали файл е отворен успешно

```
if (!inputFile.is_open())
{ … }
```

Какво може да се обърка?

Някои често срещани проблеми са следните. Възможно е файлът или пътят към него да не съществува, т.е. да не може да бъде намерен на указаното място. Възможно е за нашата програма да не е разрешен достъпът за четене или писане в този файл

Режими за достъп до файл.

Функцията `open` има още един аргумент, с който можем да се укаже режимът, в който се отваря даденият файл. Допустими са следните режими за достъп до файл:

Режим	Значение
`ios::in`	За извличане. Подразбира се за `ifstream`.
`ios::out`	За вмъкване. Може да се пише на произволни места във файла. Ако файлът съществува, съдържанието му се изтрива. Подразбира се за `ofstream`.
`ios::app`	За вмъкване, като се пише само в края на файла.
`ios::ate`	Този режим запазва всички останали, като допълнително премества маркера в края на файла.
`ios::binary`	Файлът се отваря в двоичен режим, не в текстов.
`ios::trunc`	Ако файлът съществува, съдържанието му се изтрива. Това поведение се подразбира за режима `ios::out`.

Режимите могат да бъдат комбинирани с побитовата операция or (|).

Затваряне на файл

Файлът е *външен* ресурс. Това предполага, че достъпът до него трябва да се контролира правилно. След като ресурсът е заявен с функцията `open()`, след приключване на работа с него, ресурсът трябва да бъде освободен. Използва се функцията `close()`.

```
outputFile.close();

inputFile.close();
```

Операции по извличане и запис в текстов файл

Четене (необходим е **`ifstream`**)	**Писане** (необходим е **`ofstream`**)

`operator >>`
`get`
`getline`
	`operator <<`
	`put`

Състояния на потока

В зависимост от това дали последната операция над потока е била успешна, поток може да бъде в 4 основни състояния, които се описват с отделни битове.

good

Потокът е в добро състояние, ако последната операция е била успешна. Това е най-общият флаг. За да е вдигнат, никой от останалите флагове не трябва да е вдигнат. Състоянието може да се провери с булевата функцията `good()`, приложена върху потока.

```
fileStream.good();
```

bad

Потокът е в състояние `bad`, ако последната операция не е успешна. Възникнала е критична грешка по време на работата с потока, от която потокът не може да се възстанови. Например при хардуерен проблем. Състоянието може да се провери с булевата функцията `bad()`, приложена върху потока. Когато потокът е в състояние `bad`, той не е в състояние `good`, но състоянието `bad` не е противоположно на състоянието `good`.

```
fileStream.bad();
```

fail

Потокът е в състояние `fail`, ако последната операция не е успешна. Когато потокът е в състояние `fail`, той не е в състояние `good`. Пример за това е грешен формат на данните. Например очакваме да се прочете число, но от текущата позиция не може да се разпознае такова. С определени действия, състоянието на потока може да бъде възстановено и работата с него да продължи. Състоянието може да се провери с булевата функцията `fail()`, приложена върху потока. Обикновено този флаг се вдига, когато е вдигнат флага `bad`.

```
fileStream.fail();
```

eof

Флагът за край на файл `eof` (end of file) се вдига, ако е направен опит за четене *след последния* символ, а не при прочитането на последния символ. Заедно с него се вдига и флага `fail`, защото се търсят някакви данни, но такива не се откриват и операцията пропада. Състоянието може да се провери с булевата функцията `eof()`, приложена върху потока.

```
fileStream.eof();
```

Важно!

- Ако състоянието на потока не е добро, почти всички операции, които се прилагат върху него ще бъдат неуспешни. Програмата ще премине през указаните операции, но няма да произведе никакъв резултат.
- Ако състоянието на потока не е добро, то остава такова, докато не бъде възстановено. За да се възстанови обратно в състояние `good`, може да се използва функцията `clear()`.

  ```
  fileStream.clear();
  ```

  Функцията сваля всички флагове за грешка и възстановява състояние `good`.
- Добре е състоянието на потока да се проверява след определени операции (например отваряне на файл, четене на данни от определен тип и др.).
- При обхождане на даден файл, най-добре е да не се използва проверката за достигане на край на файла като условие в цикъл. Така няма да се провери за състояния `bad` или `fail, получено поради друга причина`. Най-добре е в условието на цикъла да се проверява дали състоянието на потока е `good`.
- При получаване на състояние `fail`, ако за програмата има значение по каква причина то е възникнало, трябва да се изследват останалите състояния.
- Затварянето на файл *не изчиства* състоянието на потока. Ако файлът е изчетен докрай и трябва да се изчете повторно със същия поток, преди това трябва да се изчисти неговото състояние.

Други начини да се провери състоянието на потока

Потокът може да бъде преобразуван неявно до булева стойност. Стойност `true` е съответствие на добро състояние на потока.

```
if (inputFile) ...
```

Проверява се дали потокът е в добро състояние. Еквивалентно на

```
if (inputFile.good()) ...
```

```
if (!inputFile) ... еквивалентно на if (!inputFile.good())
```

```
while (inputFile >> number)
```

Четенето от потока продължава, докато потокът е в добро състояние. Ако при приключване на цикъла е необходимо да се потвърди, че всички данни са изчетени, то трябва да се провери дали е вдигнат флагът за край на файл.

Работа с текстови файлове в примери

Пример

Даден е текстов файл numbers.txt, който съдържа последователност от числа, разделени с интервал:

1 23 456a 7890

Да се напише програма, която прочита записаните във файла числа и ги извежда на стандартния изход.

Бележки

Добре е след отваряне на файла да се провери дали операцията е успешна.

Потоците не могат да бъдат предавани по стойност, не могат да бъдат копирани!

Състоянието им е много динамично, променя се след всяка операция, извършена върху потока.

Потоците се предават по референция и така текущото им състояние може да се проследи след приключването на функцията.

```
void rewind(std::ifstream& inputFile);
void showNumbers(std::ifstream& inputFile);

int main()
{
	std::ifstream inputFile("numbers.txt", std::ios::in);
	if (!inputFile.is_open())
	{
		std::cout << "The file cannot be opened for reading!\n";
		return 1;
	}
	showNumbers(inputFile);
	inputFile.close();
}

void rewind(std::ifstream& inputFile)
{
	inputFile.clear();
	inputFile.seekg(0, std::ios::beg);
}

void showNumbers(std::ifstream& inputFile)
{
	rewind(inputFile);

	int number = 0;
	while (inputFile >> number)
		std::cout << number << ' ';
}
```

Функцията `rewind` изчиства състоянието на потока, подаден като аргумент, и позиционира указателя му get в началото на файла. По този начин файлът може да се изчете отначало.

Функцията `showNumbers` получава като аргумент входния поток, свързан с файла. Четенето продължава, докато потокът е в добро състояние.

Четенето се осъществява с оператора за вход. Разпознава се 1, прескача се интервала, разпознава се 23, прескача се и следващият разделител. Разпознава се 456 и четенето спира върху символа ‘a’, който не може да е част от цяло число, след което потокът изпада в състояние `fail` и проверката на условието в цикъла пропада.

След приключване на функцията `showNumbers` може да се провери, че не е достигнат краят на файла.

```
	if (!inputFile.eof())
	{
		std::cout << "The end of file has not been reached!\n";
	}
```

Този прост вариант на четене от файла е оправдан, когато (1) е сигурно, че данните са валидни и (2) няма да възникнат сериозни входно-изходни грешки. При тези условия потокът изпада в състояние `fail` само при достигане до края на файла.

Как да се прочете цяло число от текстовия файл, като се изследват всички възможни състояния на потока, а при възникване на грешка, да се прескочат всички невалидни символи?

```
void showNumbers(std::ifstream& inputFile)
{
	rewind(inputFile);

	int number = 0;
	while (inputFile)
	{
		if(numberWasFound(inputFile, number))
			std::cout << number << ' ';
	}
}

bool numberWasFound(std::ifstream& inputFile, int& number)
{
	inputFile >> number;

	if (inputFile.good())
		return true;

	if (inputFile.eof())
	{
		if (!inputFile.fail())
		{
			// краят на файла е веднага след последното число,
			// няма нов ред или друг символ след него
			// с прочитането на последното число eof се вдига,
			// но потокът не изпада в състояние fail
			return true;
		}

		std::cout << "The end of file has been reached!\n";
		return false;
	}

	if (inputFile.fail())
	{
		// опит да се възстанови състоянието на потока
		inputFile.clear();

		// и да се изчистят символите до следващото бяло поле
		char c;
		while (inputFile.get(c) && !isspace(c));
```

```
            return false;
        }

        // потокът е в състояние bad
        return false;
}
```

Файлов указател

Файловете могат да се разглеждат като последователност от байтове, номерацията на които започва от 0 и завършва с край на файл.

Всеки поток съдържа файлов указател. Ако файлът е отворен за четене, файловият указател показва текущата позицията за четене. Ако файлът е отворен за запис, файловият указател показва текущата позиция, на която ще се извърши записването.

По подразбиране, при отваряне на файл за четене или запис, указателят се установява в началото на файла. Той се премества с всяка операция по четене / запис във файла, с толкова байта, колкото са били съответно прочетени/ записани във файла.

Ако файлът е отворен в append режим (`ios::app`), файловият указател се премества в края на файла, като по този начин новите данни не презаписват старите.

Как се мести файловият указател?

Файловият указател може да се премества програмно, като се използват функциите `seekg()` и `seekp()`. g идва от get, т.е. `seekg()` премества позицията за четене. p идва от put, т.е. `seekp()` премества позицията за запис.

Ако файлът е отворен едновременно и за четене, и за писане (поток `fstream`), трябва да се внимава дали операциите се извършват на правилните позиции. Позицията за четене се променя както след всяко четене, така и след всяко писане във файла. Аналогично, позицията за писане се променя както след всяко писане, така и след всяко четене.

```
istream& seekg( pos_type pos );
istream& seekg( off_type off, std::ios_base::seekdir dir);
```

Функцията премества файловия указател на нова позиция за четене.

Първият вид на функцията отмества указателя на указана *абсолютна* позиция от началото на файла.

```
    inputFile.seekg(32);
```

Отместването е в брой байтове.

Вторият вид на функцията отмества указателя на указана *относителна* позиция, определена като `off` байта спрямо позицията `dir`. Относителната позиция `off` може да бъде както положително, така и отрицателно число. Позицията `dir` е една от следните константи:

Флаг	**Значение**
`ios::beg`	`Начало на файловия поток.`
`ios::cur`	`Текущата позиция на файловия указател`
`ios::end`	`Края на файловия поток.`

Примери:

```
inputFile.seekg(-1, std::ios::cur); // 1 байт вляво (назад) от текущата позиция
inputFile.seekg(5, std::ios::cur); // 5 байта вдясно (напред) от текущата позиция
inputFile.seekg(-4, std::ios::end); // 4 байта преди края
```

Преместването в началото, съответно в края на файла се осъществява по следния начин:

```
inputFile.seekg(0, std::ios::beg);
inputFile.seekg(0, std::ios::end);
```

Аналогично, за преместване на файловия указател на нова позиция за писане може да се използва функцията:

```
ostream& seekp( pos_type pos );
ostream& seekp( off_type off, std::ios_base::seekdir dir );
```

Първият вид на функцията отмества указателя на указана *абсолютна* позиция от началото на файла.

Вторият вид на функцията отмества указателя на указана *относителна* позиция, определена като `off` байта спрямо позицията `dir`. Относителната позиция `off` може да бъде както положително, така и отрицателно число. Позицията `dir` е `ios::beg`, `ios::cur` или `ios::end`.

Текуща позиция на файловия указател

За да се установи текущата позиция за четене във входен файлов поток, се използва функцията `tellg()`.

```
streampos currentReadingPosition = inputFile.tellg();
```

Ако файлът е текстов, текущата позиция за четене е позицията на следващия символ, който ще бъде прочетен. Ако файлът е двоичен, това е позицията на следващия байт, който ще бъде прочетен.

Ако потокът не е в добро състояние или възникне друг проблем при извикването на функцията, резултатът е -1.

`streampos` е специален тип данни, който се използва за позиция във файла. Може да бъде преобразуван до интегрални типове като `int`, `long`, `long long`. Стойности от тип `streampos` могат да се събират или изваждат.

За да се установи текущата позиция за писане в изходен поток, се използва функцията `tellp()`.

```
streampos currentWritingPosition = outputFile.tellp();
```

Ако потокът не е в добро състояние при извикването ѝ, то функцията връща -1.

Функциите `seekg` и `tellg` (съответно `seekp` и `tellp`) трябва да се използват с особено внимание, когато се работи с текстови файлове. В текстовите файлове говорим за символи. Seek операциите работят с байтове. Препоръчително е употребата им в текстови файлове да се сведе само до позициониране в началото и края на файла, тъй като няма пълно съответствие между физическата позиция в байтове и логическата позиция в символи.

Файлове с последователен достъп. Обобщение.

Как се добавя нов запис на дадена позиция?

Данните се записват и четат последователно. Следователно, първо трябва да се прочетат всички записи преди търсената позиция. С файла трябва да е свързан поток от тип `fstream`, за да са възможни операциите четене и запис.

Може ли да се добави нов запис без да се презапише вече съществуваща информация във файла? Директно, не.

Един начин е всички прочетени до търсената позиция записи да се запишат в нов файл, след което в този нов файл да се добави новият запис, а след него да се пренесат и останалите записи от оригиналния файл. Оригиналният файл се премахва, а новият се преименува.

Друг начин е всички записи след исканата позиция да се преместят надолу, подобно на добавяне на нов елемент в масив.

Как се променя запис?

Данните се записват и четат последователно. Следователно, първо трябва да се прочетат всички записи преди търсения. Ако новият и старият запис са с еднаква дължина, то записът може да се презапише спокойно. В противен случай, има риск да бъдат презаписани и други записи или да останат частични данни от записа, който трябва да бъде променен.

Едно възможно решение е да се използва нов файл, в който да се пренесат записите от оригиналния файл, като исканият запис се замени с новия. Оригиналният файл се премахва, а новият се преименува.

Как се изтрива запис от файла?

Данните се записват и четат последователно. Следователно, първо трябва да се прочетат всички записи преди търсения.

Може да се прочете следващият запис и да се запише на мястото на записа, който трябва да бъде изтрит. След това операцията да се повтори за следващия запис. Подобно на изтриване на елемент на масив, като се изместят всички елементи след него с една позиция наляво. Няма стандартен начин за намаляване на размера на файл. Това значи, че при подобно изтриване в края на файла ще остане ненужна, но неизтрита информация.

Може да се използва нов временен файл, в който да се пренесат всички записи без този, който трябва да бъде изтрит. Оригиналният файл се премахва, а новият се преименува.

Извод

Файловете с последователен достъп не са подходящи, когато:

- *е необходим бърз, директен достъп до даден запис;*
- *записите трябва да се редактират на място. Операциите изискват обработване на цялото съдържание на файла.*

Файлове с пряк достъп

C++ не налага някаква определена структура на файловете.

Най-лесният начин да се създаде файл с пряк достъп е, ако записите във файла са с еднакъв фиксиран размер. Това дава възможност бързо да се определи относителната позиция на даден запис спрямо началото на файла, а следователно улеснява директния достъп до записите, дори и в големи файлове.

Как се създава файл с пряк достъп?

Необходим е изходен файлов поток.

Нека файлът да съдържа последователност от цели числа.

Ако за записване на тези цели числа се използва оператора за изход << (форматиран изход), то всяко от числата ще бъде записано с различен брой символи, от 1 до 11 символа. В оперативната памет, всяко от целите числа се записва във фиксиран брой байтове.

Как числата да бъдат записани във файла по начин, подобен на начина, по който се случва това в паметта?

За да може цялото число да се запише като последователност от байтове:

1. Файлът трябва да бъде отворен не в текстов, а в *двоичен* режим.
2. Данните да се запишат с помощта на функцията `write()`, а не с оператор за форматиран изход.

Прототип на функцията `write()`:

```
ostream& write (const char* s, streamsize n)
```

Функцията записва във файловия поток n наброй байта, които се извличат от указания адрес s.

```
std::ofstream outputFile("numbers.bin",
                         std::ios::out | std::ios::binary);
if (!outputFile.is_open())
{ … }

int number = 123;
outputFile.write(reinterpret_cast<const char*>(&number), sizeof(number));
```

&number е адресът на променливата, чиято стойност трябва да бъде съхранена. Променливата е от тип `int`, следователно адресът ѝ е стойност от тип `int*`. Функцията обаче очаква аргумент от тип `const char*`, иначе казано, последователност от байтове, достъпни само за четене. (Променлива от тип `char` се съхранява в 1B). Затова се налага явно преобразуване до необходимия тип.

Операторът `reinterpret_cast` служи за преобразуване между несъвместими иначе типове. Не осъществява никаква промяна на стойността, която е аргумент на оператора. Компилаторът просто интерпретира байтовете по различен начин – като данни от типа, указан в ъгловите скоби <>.

Как да се прочетат данните, записани по този начин?

Необходим е входен файлов поток, отворен за работа в *двоичен* режим.

Данните във файла са сурови, записани са като последователност от байтове.

За да бъдат извлечени от даден входен поток определен брой байтове, се използва функцията `read()`. Тя има следния прототип:

```
istream& read (char* s, streamsize n)
```

Функцията извлича n наброй байтове и ги съхранява в паметта, сочена от s.

```
std::ifstream inputFile("numbers.bin",
                        std::ios::in | std::ios::binary);
if (!inputFile.is_open())
{ … }

int number;
inFile.read(reinterpret_cast<char*>(&number), sizeof(number));
```

От файла се прочитат толкова байта, колкото е размерът на една целочислена променлива. Данните, които се извличат трябва да бъдат съхранени в паметта, заделена за променливата number. Поради това, функцията получава като първи аргумент адреса на тази променлива. const липсва, защото стойността на number се променя.

Може ли да се запише, съответно да се прочете, по-голям блок наведнъж?

Масив от цели числа

```
int array[10] = {...};
outputFile.write(reinterpret_cast<const char*>(array), sizeof(array));
```

Винаги ли, когато става въпрос за масив, `sizeof(array)` ще определи правилно капацитета на масива?

Структурна променлива

```
struct Student
{
	int id;
	char name[24];
	double grade;
};

Student me = { 10, "Me", 5.78 };
outputFile.write(reinterpret_cast<const char*>(&me), sizeof(me));
```

В кои случаи структурната променлива не може (не трябва) да бъде записана наведнъж, като едно цяло? (Обяснение по-долу)

Работа с файлове с пряк достъп.

Функциите `seekg()` и `seekp()`, в зависимост от операцията, която ще се изпълни върху записа, задава нова позицията за четене, съответно позиция за запис, във файла.

Функциите `tellg()` и `tellp()` могат да се използват, за да се установи текущата позиция на четене, съответно запис.

Функциите се използват за прескачане до произволна позиция във файл с пряк достъп.

Пример

Нека във файл с пряк достъп са записани данните за студенти от ФМИ, които трябва да положат определен изпит. Информацията, която се съхранява за всеки студент е факултетен номер, име и оценка. Факултетните номера са последователни и започват от 1. Оценката се нанася след провеждането на изпита.

1. Да се запише информация за студент с факултетен номер 3.
   Ако факултетните номера са последователни и започват от единица, това означава, че за да се достигне до третия запис, трябва да се прескочат първите два. За да се може да се запише информацията, позицията за запис трябва да бъде променена.

```
outputFile.seekp(2 * sizeof(Student), std::ios::beg);
outputFile.write(reinterpret_cast<const char*>(&student),
                 sizeof(Student));
```

2. Изтриването на запис може да означава и презаписване с празен, фиктивен запис.
   Ако във файла има фиктивни записи, то обработването на записите трябва бъде съпроводено от проверка.

   Space-time Trade-off
   Ако обемът на данните е ясен и го позволява, файлът може предварително да се попълни с толкова фиктивни записи, колкото се очакват в действителност. Това означава, че през голяма част от времето, файлът ще бъде разреден, т.е. не всички записи в нещо ще бъдат валидни. Има преразход на памет, но достъпът до отделните записи е бърз, както и операциите по добавяне, редактиране и изтриване на отделни записи. В допълнение, данните много лесно могат да се поддържат сортирани.

Особености. Файлове и указатели.

Нека структурата Студент е дефинирана по следния начин:

```
struct Student
{
	int id;
	char* name;
	double grade;
};
```

Името на студента може да бъде съхранено в динамичната памет. В този случай в структурната променлива се съхранява адресът, на който са записани данните.

Какво ще се случи, ако записването на студент във файла се извърши така?

```
outputFile.write(reinterpret_cast<const char*>(&me), sizeof(me));
```

Какво се записва във файла?

В структурната променлива се съхранява адресът, на който е записано името, т.е. във файла също ще бъде записан адрес.

Внимание!

Не записвайте адреси във файл! Данните ще останат ли на този адрес? Нищо не го гарантира!

Как се записват данните, а не стойността на указателя?

1. Може да се запише самият низ, заедно с терминиращата 0.
   До какво води това? Четенето се усложнява.
   Начинът, по който са записани данните, определя начина, по който трябва да бъдат прочетени.
   Прочита се всичко до срещането на терминиращата 0. След като се определи броят на символите, може да се задели необходимото количество памет. Низът трябва да бъде прочетен отново, за да се съхрани в паметта. За целта, позицията за четене трябва да се върне отново там, където започва низът.

2. Преди самия символен низ може да се запише броят на символите, след което самите данни.
   До какво води това?
   Записва се допълнително и размерът. Това ограничава обема на данните, които могат да се запишат във файла. Низът не могат да надвишава като дължина максималната стойност на

типа, който се използва за съхраняването на размера. Много лесно обаче данните могат да бъдат извлечени от файла.

Записите с еднакъв размер ли са? Файлът е двоичен, но запазва ли се прекият достъп до елементите?

Пример

Как да се определи размерът на даден файл в байтове?

```
ifstream inputFile(“file.txt”, std::ios::in | std::ios::binary);
…
inputFile.seekg(0, std::ios::end);
std::streampos positionAtEnd = inputFile.tellg();
std::cout
      << "The size of the input file in bytes: "
      << positionAtEnd << std::endl;
```

Полезни връзки

http://www.cplusplus.com/doc/tutorial/files/

http://www.cplusplus.com/reference/istream/istream/

https://www.youtube.com/watch?v=_OqTD6ax3l4

https://www.cprogramming.com/tutorial/c++-iostreams.html

http://www.cplusplus.com/reference/iolibrary/