

# 7 Files

## 7.1 Introduction

You have probably noticed when running our programs during the course so far, each run has started from the same origin as previous run. The data entered to the program is gone when running the program the next time. This is of course unacceptable. We must have the possibility to save data entered or calculated during one run, so we can continue where we stopped last time. The solution to this is to save the data on disk in files.

In this chapter we will go through the basic concepts about file management and how to read from and write to files in C++.

In professional programming relational databases of some kind are mostly used for data storage. A lot of special code is however required in C++ to do that, which is outside the scope of this course. Here, we will only store data in the simplest format, namely text files which can be read and updated with a simple text editor like the Notepad program.

To be able to handle files in C++ we need some knowledge about streams, which we will first go through in this chapter. We will then show how to declare a file, open it, save data in it, read from it and close it.

You should normally open the file as late as possible in the program and close it as early as possible, since you want to minimize the risk for loss of data by keeping the files open as short time as possible. If the system would break down while a file is open, the result might be a corrupt file not possible to read from. You will then have to return to the last backup of the file.



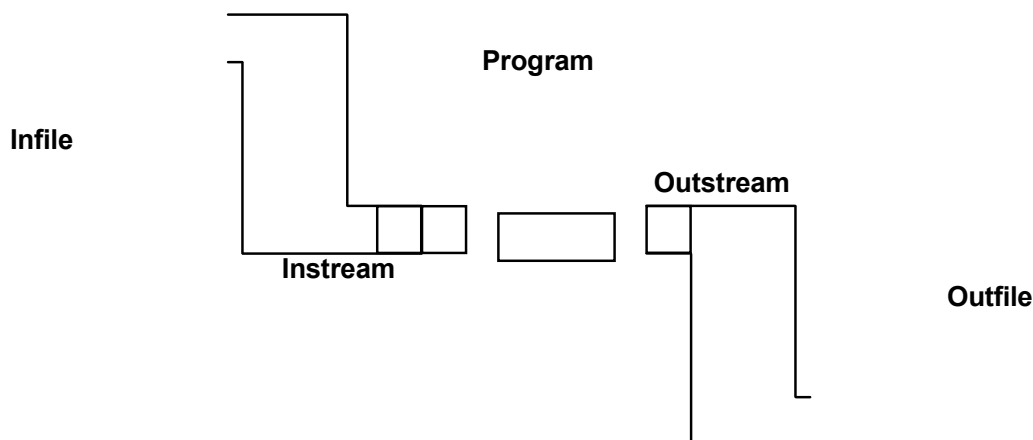
"I studied English for 16 years but...  
...I finally learned to speak it in just six lessons"  
Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download

## 7.2 Streams

When files are processed in C++ the communication goes between hard disk file and program via a stream:



When data is to be read from a disk file (infile) to a program, it goes via an intermediary store (instream) which works as a buffer between the hard disk and the program, where data is queued to be read to the program.

Similarly, when data is to be written from the program to a disk file (outfile), it is first stored in an intermediary store (outstream) before it is finally written to the file.

As programmer you only have to bother about reading from the instream and writing to the outstream. The operating system takes care of the physical reading and writing on the disk file.

## 7.3 Reading from a Stream

So far, you have read data from the keyboard with statements like:

```
cin >> cName;
```

We have said that cin stands for 'console in', i.e. reading from the keyboard. But actually, the data has been transferred via an instream called cin.

The same applies to reading from an instream. Suppose that we have an instream called is and that it is connected to a particular disk file. Then we read data from the instream with statements like:

```
is >> cName;
is >> dAmount;
```

Reading data from an instream with the >> operator is called '**formatted input**', because the data from the instream is automatically accommodated to the data type of the receiving variable.

In some situations it is not possible to accommodate the data to a specific data type, for instance if you try to read letters to an integer variable. A run-time error will then occur.

You can also use '**unformatted input**', which means that characters are read from the file exactly as they are stored, without any accommodation. Here is an example:

```
char cName[30];  
is.getline(cName, 29);
```

The last statement reads up to 29 characters from the instream, and the null character is put after the last read character. The read operation continues until the end line character is reached. Suppose the data in the file is stored linewise (for instance if data has been entered using Notepad and Enter has been pressed after each line). Then one line at a time is read.

If there happens to be fewer characters than 29 at the current line in the file, for instance 17 characters, the null character is stored in the 18th position.

If there are more than 29 characters at the line in question in the file, the input is interrupted after 29 characters.

Thus, the programmer must carefully check how data is stored in the file, so as not to lose important information.

## 7.4 Writing to a Stream

Writing data to an outstream can also be done in two ways:

**Formatted output** is done with statements like:

```
os << cName;
```

Here we presume that an outstream named `os` has been created and been connected to a specific disk file. The statement implies that the characters in the variable `cName` are written to the outstream.

Formatted output also implies that you have the opportunity to control the layout of the data, for instance with the function `width()`. Compare the 'Variables' chapter, where we discussed formatted output.

**Unformatted output** means that the characters in the variable are written to the outstream exactly in the format they are stored in the variable, for instance:

```
os.put(c);  
os.write(cName, 30);
```

The `put()` function prints a character, namely the character represented by the variable `c`, to the outstream `os`. The `write()` function writes 30 characters from the variable `cName` to the outstream `os`.

## 7.5 Attaching a File to a Stream

Before being able to use an instream or outstream, it must be declared and attached to a disk file. The statement:

```
ifstream infile("address.txt");
```

declares the instream `infile` and attaches it to the disk file named `address.txt`.

`ifstream` is the short for 'input file stream'. You can regard `ifstream` as a data type similarly to integer or double. But actually, `ifstream` is a class from which we derive an object of the `ifstream` type with the object name `infile`.

When this statement has been executed the stream is ready for read operations.

Below we declare an outstream:

```
ofstream outfile("newadr.txt");
```

The stream is called `outfile` and is connected to a disk file named `newadr.txt`. `ofstream` is short for 'output file stream'. `ofstream` is also a class from which we create the object `outfile`.

At completion of this statement the outstream is ready for write operations.

If the file `newadr.txt` exists, it will be deleted and a new file with the same name is created. Many times you want to add data to the end of an existing file without destroying existing information. The outstream is then declared as follows:

```
ofstream outfile("newadr.txt", ios::app);
```

`app` is short for 'append'.

To make the stream declarations above work you must *include* the file `fstream.h`:

```
#include <fstream.h>
```

To allow for reading from and writing to streams, you must as usual include the file `iostream.h`:

```
#include <iostream.h>
```

At completion of reading from or writing to the streams, you *close the streams*:

```
infile.close();
outfile.close();
```

Excellent Economics and Business programmes at:



**university of  
 groningen**





**“The perfect start  
 of a successful,  
 international career.”**

[www.rug.nl/feb/education](http://www.rug.nl/feb/education)

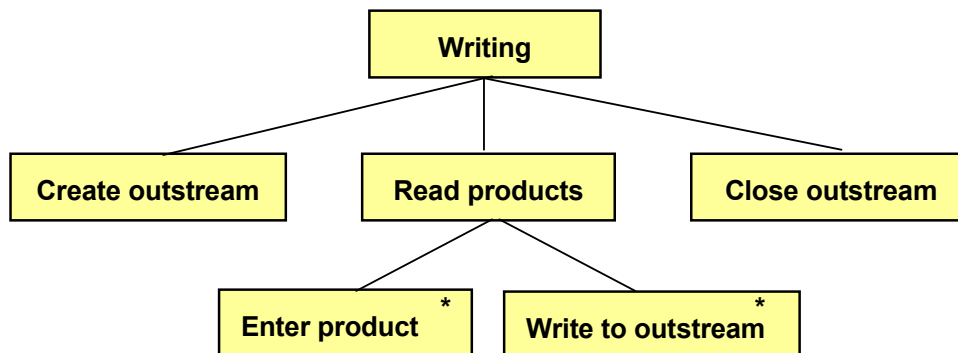
**CLICK HERE**  
 to discover why both socially  
 and academically the University  
 of Groningen is one of the best  
 places for a student to be



## 7.6 A Complete Write Program

To summarize our experiences we will now create a simple program for writing of data to file. The program will read product names from the user (keyboard) and store them in a file named `prodfile.txt`

We begin with a JSP graph:



First we declare an ostream and attach it to the file `prodfile.txt`. Entry of product names from the keyboard is made in a loop. As soon as a product name has been entered by the user, it is written to the ostream. At entry completion, we close the ostream.

Here is the code:

---

```

#include<iostream.h>
#include<fstream.h>
void main()
{
    char cProd[30] = "";
    ofstream outfile("prodfile.txt");
    cout << endl << "Enter product, (only Enter to exit): ";
    cin.getline(cProd,29);
    while(cProd[0]!='\0')
    {
        outfile << cProd << endl;
        cout << endl << "Enter product: ";
        cin.getline(cProd,29);
    }
    outfile.close();
}

```

---

First we include the two header files `iostream.h` (to allow for input and output) and `fstream.h` (to allow for stream management).

Download free eBooks at [bookboon.com](http://bookboon.com)

In main() we declare the string variable cProd used for storage of product names in the program. Then we declare the outstream outfile and attach it to the disk file prodfile.txt.

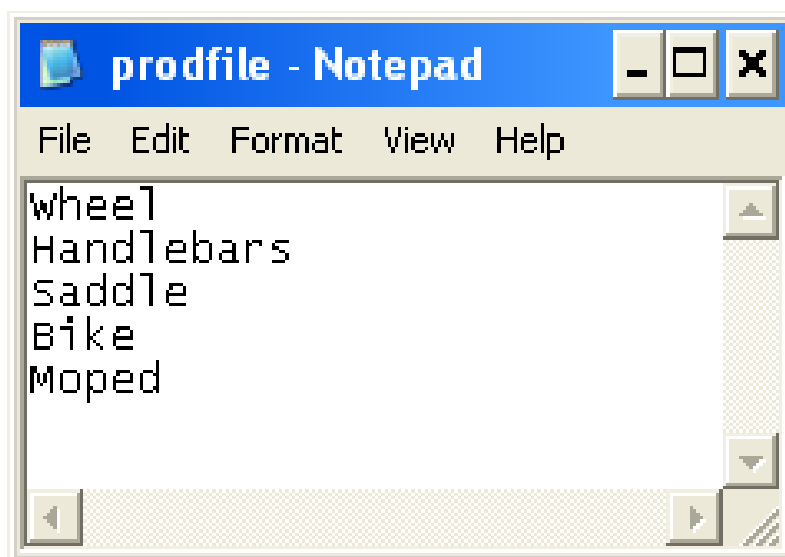
Entry and output is made by first reading the first product from the keyboard with the cin.getline() function before the while loop starts. Since the while condition checks the variable cProd to actually hold a string, the string variable cProd must contain a string.

The while condition checks that the first character of the string variable cProd (cProd[0]) is not the null character. If it were, the user would have pressed Enter without having entered any product name, and the loop is then terminated.

The first statement in the loop prints the product name to the outstream outfile. The two subsequent statements read a new product name from the user.

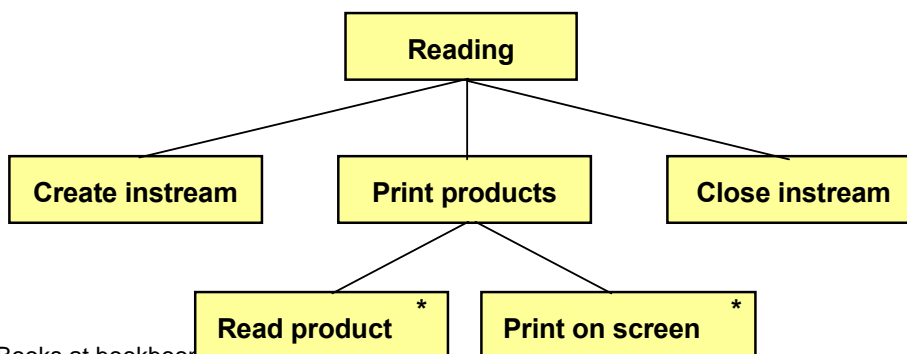
The loop is terminated when the user presses Enter without entering any product name. The outstream is then closed and the file operation is complete. The file prodfile.txt now contains a number of product names.

When having run the program you would probably like to examine the result. Start 'Explore' and find the file prodfile.txt, which is in the project folder where the cpp file is saved, or maybe in the 'Debug' subfolder, depending on your Visual Studio settings. Double-click the file to make the Notepad program be started and the file content be shown:



### 7.7 A Complete Reading Program

We will now create a new program that reads data from prodfile.txt and prints the information on the screen. We start with a JSP graph:



First we declare the instream and attach it to the prodfile.txt file. Reading and printing on the screen is made in a loop where we read one product at a time from the instream and print it on the screen. At completion, we close the instream.

Here is the program:

```
#include<iostream.h>
#include<fstream.h>
void main()
{
    char cProd[30] = "";
    ifstream infile("prodfile.txt");
    while(infile.getline(cProd,29))
        cout << cProd << endl;
    infile.close();
}
```

We include the same header files as in the previous program.

In main() we declare the string variable cProd used for holding product names in the program. Then we declare the instream infile and attach it to the disk file prodfile.txt.

The while loop has the condition of a successful reading from the instream. If so, the read product is printed on the screen. When there is no more data in the file, the read operation is unsuccessful and the loop is terminated. The instream is closed.



**LIGS University**  
based in Hawaii, USA

is currently enrolling in the  
Interactive Online **BBA, MBA, MSc,**  
**DBA and PhD** programs:

- ▶ enroll **by October 31st, 2014** and
- ▶ **save up to 11%** on the tuition!
- ▶ pay in 10 installments / 2 years
- ▶ Interactive **Online education**
- ▶ visit [www.ligsuniversity.com](http://www.ligsuniversity.com) to find out more!

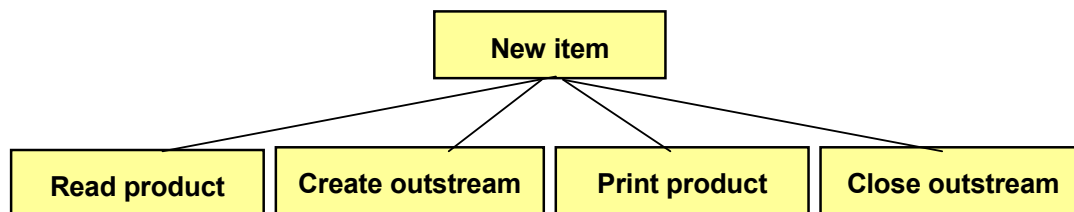
**Note:** LIGS University is not accredited by any nationally recognized accrediting agency listed by the US Secretary of Education. More info [here](#).





## 7.8 New Item at the End of the File

We will now show how to add one more product at the end of the file `prodfile.txt`. The solution is given by the following JSP graph:



This program reads only one more product. But, by placing the input from the keyboard and printing to the ostream in a loop, you could make the program more flexible to allow for entry of any number of products.

First we read the product name from the user, then we create the ostream and attach it to the file `prodfile.txt`, print the product to the ostream, and close the ostream.

Here is the program code:

---

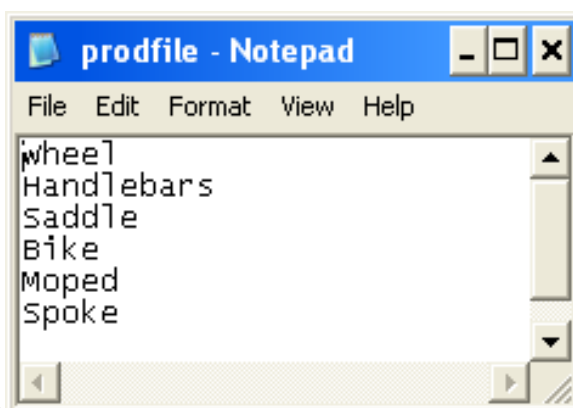
```

#include<iostream.h>
#include<fstream.h>
void main()
{
    char cProd[30] = "";
    cout << "Enter new product: ";
    cin.getline(cProd,29);
    ofstream outfile("prodfile.txt",ios::app);
    outfile << cProd << endl;
    outfile.close();
}
  
```

---

We use the same include files as previously. In `main()` we prompt the user for a new product. Then we declare the ostream `outfile` and attach it to the disk file `prodfile.txt`. Note that we use `ios::app` to make existing data be kept and new data be added at the end of the file. Then we print the entered product to the ostream and close the ostream.

If you check the file `prodfile.txt` in Notepad, you will see one more product having been added at the end:





## 7.9 Products and Prices

We will now recreate the prodfile.txt to store both product id:s and prices for a number of products. The structure of the file will be:

```
Product id
Price
Product id
Price
etc.
```

The price of each product comes after the product id.

The program will be similar to the one used for writing to file:

---

```
#include<iostream.h>
#include<fstream.h>
void main()
{
    int iProdId = 1;
    double dPrice;
    ofstream outfile("prodfile.txt");
    while(iProdId !=0)
    {
        cout << endl << "Enter product id: ";
        cin >> iProdId;
        cout << " ...and price: ";
        cin >> dPrice;
        if(iProdId > 0)
            outfile << iProdId << endl << dPrice << endl;
    }
    outfile.close();
}
```

---

We use the same include files as previously.

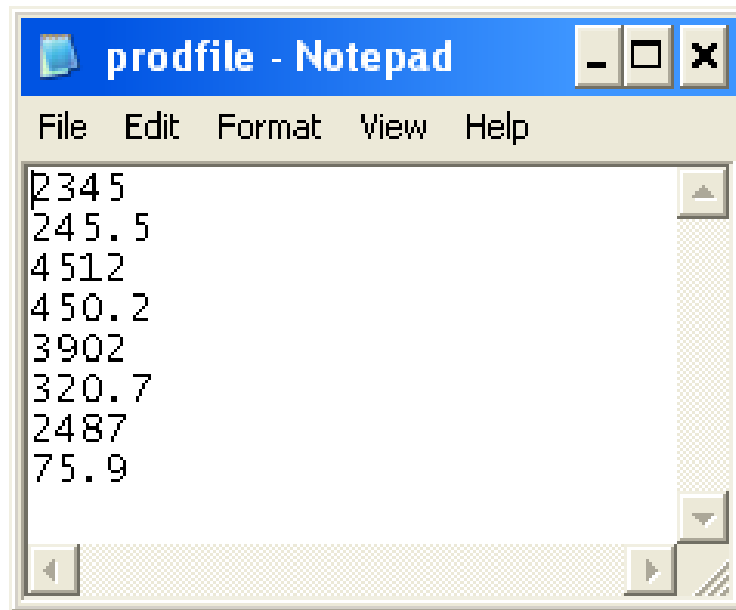
In main() we declare the variable iProdId used for storage of the product id:s in the program. It is initialized to 1 for the sake of making the while loop start with a valid value of iProdId. The variable dPrice will hold the product prices. We also declare the outstream outfile and attach it to the disk file prodfile.txt.

The while loop reads product id:s and prices from the user and prints them to the outstream. The while condition checks that there is a valid product id different from zero. If so, the user is prompted for a product id and a price. If the product id is greater than zero, the information is written to the outstream outfile. Note that we also print endl after each value, which makes each information item be printed on a separate line. This way of storing data in a file facilitates printing and reading from a programming point of view. Avoid several values per line!


Download free eBooks at [bookboon.com](http://bookboon.com)

At completion of the loop the ostream is closed.

By looking at the file in Notepad, you can figure out the structure:




First there is the product id 2345 and then the price of that product 245.5. Then comes the next product etc.

.....Alcatel-Lucent 

[www.alcatel-lucent.com/careers](http://www.alcatel-lucent.com/careers)

What if you could build your future and create the future?

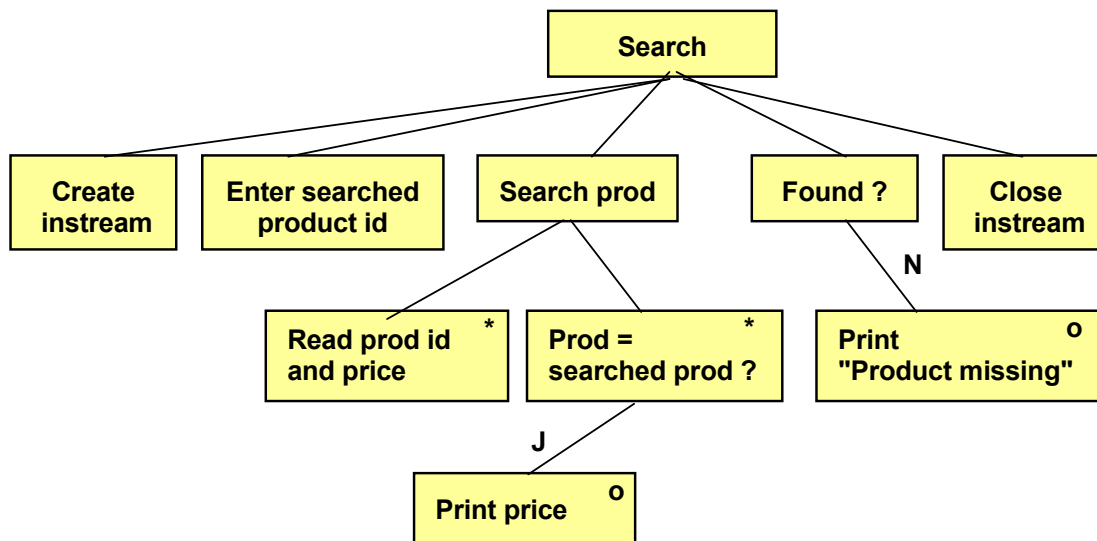
One generation's transformation is the next's status quo. In the near future, people may soon think it's strange that devices ever had to be "plugged in." To obtain that status, there needs to be "The Shift".



## 7.10 Search for a Product Price

We will now use the new `prodfile.txt` to find the price of a product specified by the user. The labourousome thing about this kind of files is that we always must start reading from the beginning of the file until we find the correct product. Then we also easily can find the corresponding price.

First we give a JSP graph:



First we create an instream, and then the user is prompted for the searched product id.

The loop 'Search prod' reads one product id and price at a time from the instream. If it is the searched product id, the price is printed.

After the loop we check if the correct product id was found. If not, an error text is printed. Finally we close the instream.

Here is the program code:

---

```

#include <iostream.h>
#include <fstream.h>
void main()
{
    int iProdId, iSrch, iFound=0;
    double dPrice;
    ifstream infile("prodfile.txt");
    cout << "Enter product id: ";
    cin >> iSrch;
    while(infile >> iProdId >> dPrice)
    {
        if (iProdId == iSrch)
        {
            cout << "The price is " << dPrice;

```

Download free eBooks at [bookboon.com](http://bookboon.com)

```

        iFound=1;
        break;
    }
}
if (!iFound)
    cout << "Product missing";
infile.close();
}

```

First in the program we declare the variables `iProdId` used for storage of the product id:s read from the instream, `iSrch` for the searched product id, `iFound` which is an indicator to remember whether or not the product id was found. The value 0 means that we have not found the correct product, and 1 means that we have found it. The variable `dPrice` is used for the price read from the instream.

Then the instream is created and attached to the disk file `prodfile.txt`.

The searched product id is read from the user and stored in the variable `iSrch`.

The while loop reads products and prices from the instream. The while condition reads one product id and the corresponding price from the instream. As long as there is data to be read, the loop continues.

When a product and a price has been read, the if statement checks if it equals the searched product id. If so, the price is printed, the variable `iFound` is set to 1 and the loop is terminated.

**Maastricht University** *Leading in Learning!*

**Join the best at the Maastricht University School of Business and Economics!**

**Top master's programmes**

- 33<sup>rd</sup> place Financial Times worldwide ranking: MSc International Business
- 1<sup>st</sup> place: MSc International Business
- 1<sup>st</sup> place: MSc Financial Economics
- 2<sup>nd</sup> place: MSc Management of Learning
- 2<sup>nd</sup> place: MSc Economics
- 2<sup>nd</sup> place: MSc Econometrics and Operations Research
- 2<sup>nd</sup> place: MSc Global Supply Chain Management and Change

Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012

**Maastricht University is the best specialist university in the Netherlands (Elsevier)**

**Visit us and find out why we are the best!**  
**Master's Open Day: 22 February 2014**

[www.mastersopenday.nl](http://www.mastersopenday.nl)

If the loop is allowed to complete, i.e. if all products have been read without finding the correct id, the variable `iFound` will still have the value 0.

After the loop the if statement checks if `iFound` still is 0. `iFound=1` means 'true', `iFound=0` means 'false', `!iFound=1` (not found) means 'true'. Thus, if 'not found' is true, the error message about missing product is printed.

Finally the instream is closed.

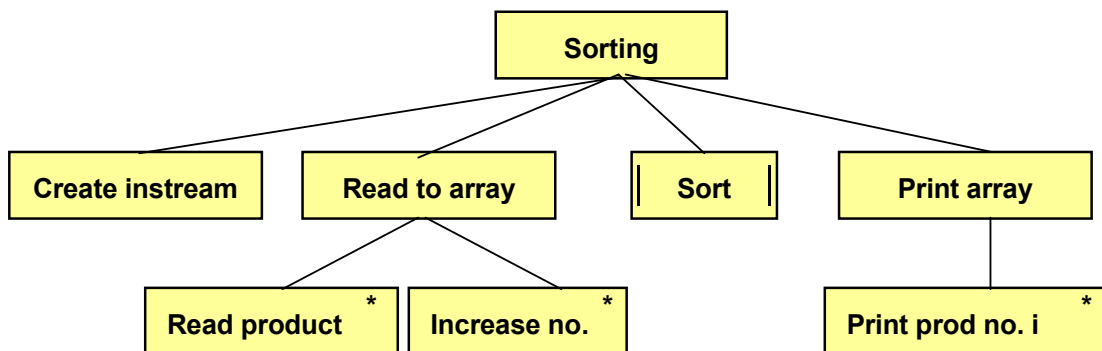
### 7.11 Sorting a File in Memory

We can't presume the products to be sorted in the file. But in a printout on the screen we want a sorted list of products. We will create a program which reads all products in the file to an array, sorts the array, and then prints the sorted array.

We now return to the first product file, namely the one only containing product names. You will probably with smaller amendments achieve the same result with the later file version.

The program will read all product names to an array (two-dimensional char array). The sorting is performed by a function. After the sorting by the function, the `main()` function will print the sorted list.

Here is first a JSP graph for the `main()` function:

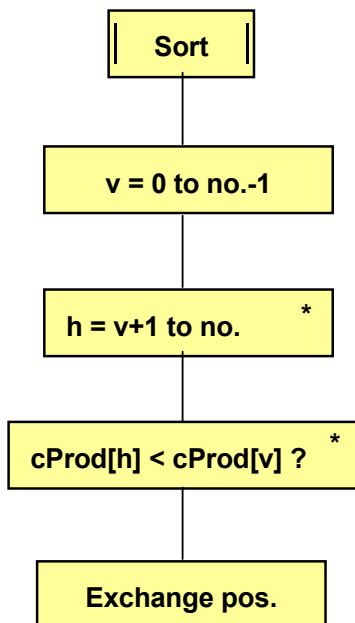


First we create the instream.

Reading of products to the array is made in a loop, where we increase the number of items for each single read. By doing so we keep track of the number of products read. This number is needed by the function `Sort` to be able to sort.

The printing of the sorted array is also made in a loop.

The JSP graph for the function Sort looks like this:



You probably recognize the sort algorithm from the Arrays chapter.

Here is the program code.

---

```

#include <iostream.h>
#include <fstream.h>
#include <string.h>
void sort(char cList[][30], int n);
void main()
{
    int i=0, j, iNo;
    char cProd[50][30];
    ifstream infile("prodfile.txt");
    while(infile.getline(cProd[i],29))
        i++;
    iNo=i;
    infile.close();
    sort(cProd,iNo);
    for(j=0; j<iNo; j++)
        cout << cProd[j] << endl;
}
void sort(char cList[][30], int n)
{
    int v,h;

```

```

char temp[30];
for(v=0; v<n-1; v++)
{
    for(h=v+1; h<n; h++)
        if(strcmp(cList[h],cList[v])<0)
        {
            strcpy(temp, cList[v]);
            strcpy(cList[v], cList[h]);
            strcpy(cList[h],temp);
        }
}
}

```

The include files are `iostream.h` for input and output, `fstream.h` for streams and `string.h` for the string functions. Furthermore, we also declare the function `sort()`.

In `main()` we declare the variable `i`, which is initialized to 0 and which will accumulate the number of products read, the variable `j` used as loop counter, and the variable `iNo` which finally stores the number of products. In addition, we declare the two-dimensional array `cProd`, which will be used for storage of the product names. We also create the instream `infile`, which is attached to the disk file `prodfile.txt`.



**> Apply now**

REDEFINE YOUR FUTURE  
**AXA GLOBAL GRADUATE  
PROGRAM 2015**

redefining / standards 

agence c&g - © Photomastop



The while loop manages reading of product names to the array. The while condition is true as long as there is data to read from the instream. In the first turn of the loop  $i = 0$  according to the initiation in the beginning of the program. Therefore, the first product is stored in the item `cProd[0]`. The loop increases the value of  $i$  to 1, and the next product is stored in `cProd[1]` etc.

After the loop we save the value of  $i$ , i.e. the number of products read, in the variable `iNo`, and then we close the instream.

Then we call the function `sort()` and send the array and `iNo`. After the sort operation we print the array in the last for-loop.

The function `sort()` takes the array and number of products as parameters. Note that  $n$  is the index of the last item of the array.

In the function we declare  $v$  and  $h$  to be used as indices in the array when two items are compared. We also declare the string array `temp`, which is used in the triangular exchange of array items.

The outer for-loop with  $v$  as loop counter goes from 0 to  $n-1$ , i.e. from the first to the next last position of the array. The inner for-loop goes from the position after  $v$  to the last position of the array.

Inside the inner for-loop we compare item  $h$  to item  $v$  by means of the function `strcmp()`, which gives a negative result if item  $h$  is less than item  $v$ . In that case the items will exchange positions, which is made in the triangular exchange by means of the string array `temp`.

At completion of the loop, the array has been sorted.

Remember that, when an array is sent as parameter to a function, it is always done as reference parameter, so the function operates on the same memory area as used by the array in `main()`. As a consequence, the array does not need to be returned from the function.

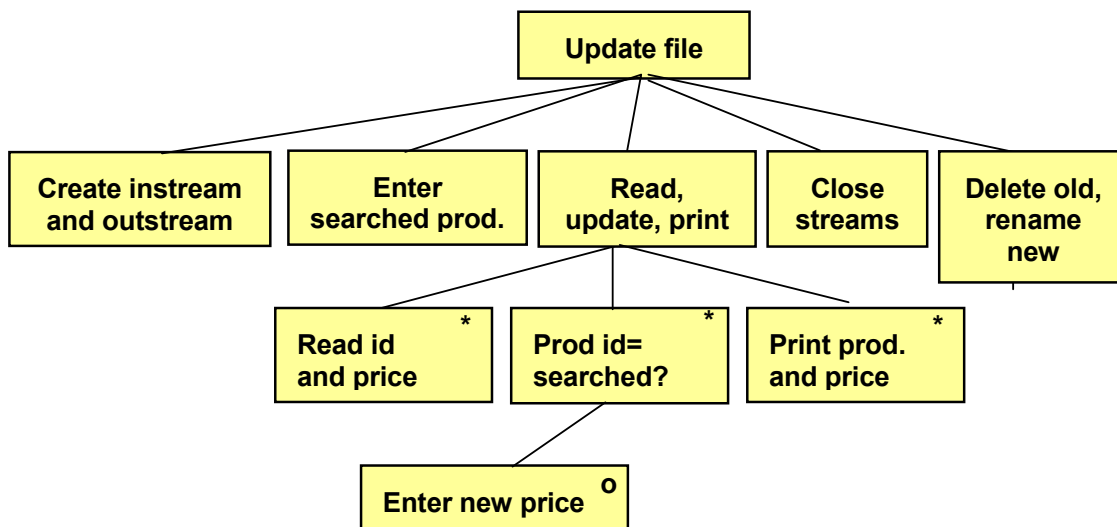
## 7.12 Updating File Content

Changing the content of a file of the type used in our programs is rather troublesome. The reason is that you can only read a file from start to end. You cannot jump into the requested position in the file and change information.

As a consequence you will have the original file as input file and a new file as output file. You read data from the infile and prints to the outfile. When arriving at the position in the file to be changed, after having read the input information, you change the value and print to the outfile. Then you will have to continue item by item from the infile and print to the outfile until all information has been transferred. Finally you delete the original file and change the name of the new file to equal the name of the original file. The information has then been updated.

We now presume that our product file contains product id:s and prices for each product. The user is prompted for a product id and a new price for that product.

We draw a JSP graph for this:



First we create the instream for the original file and the outstream for the new, which by now is empty. The user is then prompted for the product id to be updated.

Then we use a loop to read product id:s and prices from the original file. The product id is compared to the one entered by the user. If equal, the user is prompted for a new price, otherwise the old price will be used. The product id and price are then printed to the outstream.

The streams are closed and at the end of the program we delete the old file and rename the new file to the old name.

Here is the program code:

---

```

#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
void main()
{
    int iSrch, iProdId;
    double dPrice;
    ifstream infile("prodfile.txt");
    ofstream outfile("temp.txt");
    cout << "Specify product id: ";
    cin >> iSrch;
    while(infile >> iProdId >> dPrice)
    {
        if (iProdId == iSrch)
        {
            cout << "Specify the new price: ";
            cin >> dPrice;
        }
    }
}
  
```

Download free eBooks at [bookboon.com](http://bookboon.com)

```

    }
    outfile << iProdId << endl << dPrice << endl;
}
infile.close();
outfile.close();
remove("prodfile.txt");
rename("temp.txt", "prodfile.txt");
}

```

The include files are the usual ones, except that we also need `stdio.h` to be able to delete and rename files.

In `main()` we declare the variable `iSrch` to be used for the product id entered by the user, `iProdId` for product id:s read from the file, and `dPrice` for prices from the file.

Then we create the instream, which is attached to the original file `prodfile.txt`, and the outstream, which is attached to a new file, `temp.txt`. Then the user is prompted for the searched product id.

The while loop reads product id and price from the instream as long as there is data. Each product id is checked in the `if` statement against the product id specified by the user. If equal, the user is prompted for a new price, which is stored in the variable `dPrice`, i.e. the old price is replaced by the new one. Then the product id and price are written to the outstream. At completion of the while loop all products have been transferred to the new file and the requested price has been updated.

After the while loop the streams are closed, the old file is deleted by the `remove()` function and the new file `temp.txt` is renamed to `prodfile.txt` by the `rename()` function.



The image shows the BI Norwegian Business School logo, which is a blue square with the letters 'BI' in white. Surrounding the logo are various business programs represented by colorful, 3D bar-like shapes radiating outwards. The programs listed include: Business, Strategic Marketing Management, International Business, Leadership & Organisational Psychology, Shipping Management, and Financial Economics. Below the logo is the text 'BI NORWEGIAN BUSINESS SCHOOL' and the EFMD EQUIS ACCREDITED logo.

## Empowering People. Improving Business.

BI Norwegian Business School is one of Europe's largest business schools welcoming more than 20,000 students. Our programmes provide a stimulating and multi-cultural learning environment with an international outlook ultimately providing students with professional skills to meet the increasing needs of businesses.

BI offers four different two-year, full-time Master of Science (MSc) programmes that are taught entirely in English and have been designed to provide professional skills to meet the increasing need of businesses. The MSc programmes provide a stimulating and multi-cultural learning environment to give you the best platform to launch into your career.

- MSc in Business
- MSc in Financial Economics
- MSc in Strategic Marketing Management
- MSc in Leadership and Organisational Psychology

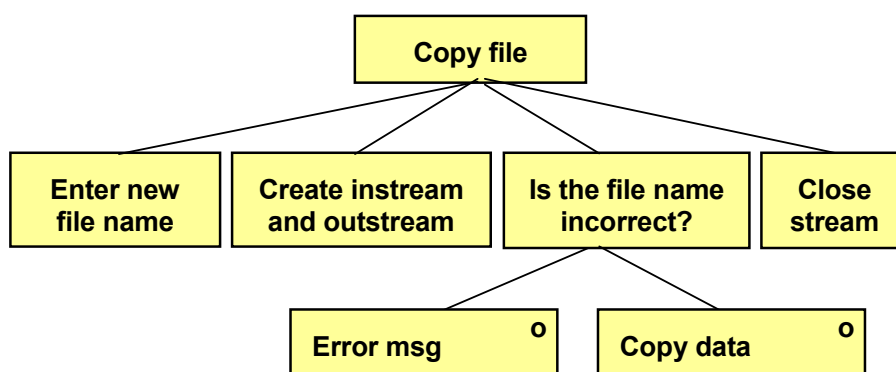
[www.bi.edu/master](http://www.bi.edu/master)

## 7.13 Copying Files

Copying a file could be done according to the same method as used by the previous program with the exception that no price is updated and that the original file is not deleted. We will however show a shortcut of copying a file with the file name specified by the user.

The copy of data is made by the function `rdbuf()`, which in one single operation reads all data from the original file without the need of picking item by item in a loop.

First we give a JSP graph:



First the user is prompted for the new file name. Then we create the instream for the original file and the ostream for the new file.

We then check if the ostream creation was successful. It might happen that the user enters characters not allowed in file names. If so, we would get a run time error. In case of an error, we print an error message. If, however, everything is OK, we copy all data. Finally we close the streams.

Here is the program code:

---

```

#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
void main()
{
    char cNewName[12];
    cout << "Specify new file name: ";
    cin >> cNewName;
    ifstream infile("prodfile.txt");
    ofstream outfile(cNewName);
    if (!outfile)
    {
        cout << "The file could not be created";
    }
}
  
```

```
else
{
    outfile << infile.rdbuf();
}
outfile.close();
infile.close();
}
```

---

The include files are `iostream.h` for input and output, `fstream.h` for stream management, and `stdio.h` to allow for using the function `rdbuf()`.

In `main()` we prompt the user for the new file name, which is stored in the variable `cNewName`. Then we create the `instream`, which is attached to the disk file `prodfile.txt`, and the `outstream`, which is attached to a disk file with the user supplied name. Note that `cNewName` is not enclosed in quotes, since it is a variable and not a specific string.

The `if` statement checks if the `outstream` creation succeeded. If so, the variable `outfile` contains an address to the `outfile` object. If it didn't succeed, the address is `= 0`. That means that `!outfile` is true if the address is 0. In that case we print an error message to the user. Otherwise, i.e. if the `outstream` could be created, we use the function `rdbuf()` to copy all data in one single operation from the `infile` to the `outfile`.

Finally the streams are closed.

Having run the program you can by means of 'Explore' check the new file.

## 7.14 Summary

In this chapter we have learnt the basics of file management. You have learnt how to use streams and attach them to physical disk files. You have also learnt that you communicate with the streams, and not directly with the disk files.

We have discussed the meaning of formatted and unformatted input and output. You are now able to write programs where the user can enter information to be stored in a file, and read information from a file and present it on the screen.

We have also studied examples of how to search for information in a file, read and sort file information before presentation on the screen, update file information and copy files.

## 7.15 Exercises

1. Start with the program in the section 'A Complete Write Program'. Expand the program so that it is also possible to specify warehouse location (for instance EH23) for each product. Check with the Notepad program that the file contains the expected information.
2. Start with the program in the section 'A Complete Reading Program' and modify it so it also will be capable of reading the warehouse locations entered in the previous exercise.
3. Start with the program in the section 'New Item at the End of the File' and modify it so that you also can enter the warehouse location of the new product. Use the same file as in the two previous exercises. Then run the program in exercise and check the existence of the new product in the output.

4. Start with the program in the section 'Products and Prices' and modify it so that you also can enter quantity in stock for each product.
5. Start with the program in the section 'Search for a Product Price' and modify it so that also quantity in stock is printed on the screen. Use the same file as created in the previous exercise.
6. Start with the program in the section 'Sorting a File in Memory' and accommodate it to also be able to manage the file with product names and warehouse locations created in the first exercise.
7. Start with the program in the section 'Updating File Content' and modify it so that the user will be able to update the quantity in stock. Use the file with product id, price and quantity in stock created in a previous exercise.
8. Create a program where you can enter
  - first name
  - surname
  - cityfor some of your course mates. These should be saved in a file. The program should be possible to run several times while keeping existing file information.
9. Create a program which reads the course mate information from the file created in the previous exercise and prints it on the screen.
10. Create a program which can update the city of a person. The first and surname must then be entered from the keyboard. The new city should also be possible to specify.
11. Create a program which can remove a person from the file. The first and surname of the person must then be entered.
12. Create a program which sorts the names of the file by surname. Then use the program from exercise 9 to check the result of the sorting.
13. Create a program which copies the file to a new file. The new file name should be entered by the user.
14. Write a menu program where you gather the tasks from the latest exercises. The menu could look like this:
  1. Enter information
  2. Print
  3. Update city
  4. Remove
  5. Sort
  6. Copy
  0. Exit

Select 0-6: