SUPSI

C++: Introduction

Exercises #1

1. Streams

In the class we've seen how to use **cout** to write to the terminal. This object is declared in the **iostream** header, which also provides other types to manage input and output data streams. In this exercise we will use streams to read and write to the terminal and to access data inside a file. Consider this first example:

```
#include <string>
#include <iostream>

int main() {
    using namespace std;
    const int THISYEAR{2018};
    string yourName;
    int birthYear;

    cout << " What is your name? "  << flush;
    cin >> yourName;

    cout << "What year were you born? " ;
    cin >> birthYear;

    cout << "Your name is " << yourName
            << " and you are approximately "
            << (THISYEAR - birthYear)
            << " years old. " << endl;
}
```

⌨ Compile and execute the example.

The **cin** object represents the standard input. Whereas the **cout** uses the left-shift operator **<<** to put data into the stream, **in** uses the right-shift operator **>>** to read data from a stream. Try to insert your name and your birth year: the **>>** has been overloaded in order to correctly parse different types of input.

⌨ What happens if you insert your first and last name when prompted "What is your name?"

The >> operator extracts only one word from the standard input: if we write two or more words only the first one will be written to **yourName**; the second word remains in the input buffer and will be read by the next call to the right-shift operator on cin. We can overcome this issue by using another function to read a whole line (up to a newline character) from the input stream: **getline** [1]

⌨ Fix the program in order to use **getline**

Now that we fixed the first part of the program, let's focus on the second part:

1 http://www.cplusplus.com/reference/string/string/getline/

⌨ What happens if we insert a non-numerical value when the program asks “What year were you born?”

When the right-shift operator can’t parse the input into the desired type it fails, and the stream enters into a “failure” state: no further operation is possible while in this state. We can verify that the stream is in a fail state using the **fail()** method. To clear the failure we need to use the **clear()** method. Furthermore, we need to clear the buffer, otherwise the invalid input will trigger another failure on the next read. We flush the input using **ignore(N, char)**, where N is the number of characters to be ignored, and char is the expected character that will stop ignore from consuming the input (for example, the new line character ‘\n’).

⌨ Modify the program in order to check for failures. You will need to read the online documentation for the ignore method in order to determine the best value for the N parameter.

2. File input and output

Streams can also be used to read from and write to files. In the fstream header the relevant types are defined: **ifstream** (for input file stream) and **ofstream** (for output file stream). Let’s take a look at the following example:

```
#include <fstream>
int main() {
    using namespace std;
    int lucky{7};
    float pi{3.14};
    double e{2.71};
    ofstream outf;
    outf.open("mydata");
    outf << lucky << endl;
    outf << pi << endl;
    outf << e << endl;
    outf << "This is an example" << endl;
    outf.flush();
    outf.close();
}
```

Reading from a file is done using ifstream: we open the file and the use the right-shift operator to read the data.

⌨ Change the code of the program in order to read the values written to the file. Show the values on the screen.

⌨ What happens if we skip the end of line chars (**endl**) when writing to the file?

3. Reading characters

Another way to read from a stream (file o standard input) uses the get method, which returns N characters:

```
#include <iostream>
using namespace std;
// insert the sequence abcdefghijklmnopqrstuvwxyz
int main(void)
{
    char str[11];
    cin.get(str, 10);
    cout << "-" << str << "-" << endl;
    cin.ignore(3);
```

```
    char z;
    cin.get(z);
    cout << "-" << z << "-" << endl;
}
```

 Copy and compile the code, then modify it to read from a file.