

## Computer Sciences– I Facoltà di Ingegneria

Written exam of 1 March 2012

Write a C program to manage a budget sheet with the income and outgoings of a customer. The information is contained in two text files, **whose names are passed as the first and the second argument on the command line**, respectively. The files have the same format: each row corresponds to a specific income (or outgoing, respectively), and is composed by three fields:

**<date>   <amount>   <motivation>**

where:

- **<date>** indicates the date of the income (or outgoing) and it is represented by an integer number in the 1 to 366 range;
- **<amount>** indicates the amount of the income (or outgoing) in Euros with two decimal digits;
- **<motivation>** describes the type of expense, and it is a string without spaces containing at most 50 characters.

A file named **TODAY.TXT** reports all the information about the expenses of the current day. This file contains, on the first row, an integer number (between 1 and 366) that indicates the current date, and in the following rows it contains a list of movements (either income or outgoings) performed in that date. Each money movement is represented by an uppercase character (I for income and O for outgoing), by an amount and by a motivation (see example below).

The program should read the file **TODAY.TXT**, and consequently update the two files of income and outgoings, **adding the corresponding rows at the end** (see the example below). Finally, the program must also **calculate and visualize on the screen the total balance**, which is the difference between sum of all income and sum of all outgoings.

---

### Example of execution:

Suppose that the program `balance.exe` is executed as follows:

```
c:\ balance income.txt outgoings.txt
```

and the two corresponding text files contain the following data:

<b>income.txt</b>	<b>outgoings.txt</b>
130 10.00 tip	112 20.50 pizzeria
140 5.50 sold_notes	130 12.30 sweets

Suppose that the file **TODAY.TXT** contains the following data:

```
160
I 5.20 sold_book
O 3.70 lunch
```

The program will update the text files as follows:

<b>income.txt</b>	<b>outgoings.txt</b>
130 10.00 tip	112 20.50 pizzeria
140 5.50 sold_notes	130 12.30 sweets
160 5.20 sold_book	160 3.70 lunch

and will display on the screen:

```
Balance is equal to -15.80 Euros.
```

```
#include <stdio.h>
```

**FILE \*fopen(char \*filename, char \* mode)** – Opening a file (mode: “r” reading – “w” writing – “a” append)

**FILE \*freopen(char \*filename, char \* mode, FILE \*file\_pointer)** – Reassign a file pointer to a different file

**int fclose(FILE \*file\_pointer)** – Closing a file

**int feof(FILE \*file\_pointer)** – Checks if end-of-file has been reached.

**int fflush(FILE \*file\_pointer)** - Empties file’s buffer.

**int getchar(void)** – Reads a character from “stdin” (keyboard)

**int fgetc(FILE \*file\_pointer)** – Gets a character from a file

**char \*gets(char \*buffer)** - Reads a line from “stdin” (keyboard)

**char \*fgets(char \*string, int maxchar, FILE \*file\_pointer)** - Reads a line from a file

**int printf(char \*format\_string, ...)** – Writes formatted output on "stdout" (screen)

**int fprintf(FILE \*file\_pointer, char \*format\_string, ...)** – Writes formatted output on a file.

**int sprintf(char \*string, char \*format\_string, ...)** – Writes formatted output on a string.

**int fputc(int c, FILE \*file\_pointer)** – Writes a character on a file.

**int putchar(int c)** – Writes a character on “stdout” (screen).

**int puts(char \*string)** - Writes a string on “stdout” (screen).

**int fputs(char \*string, FILE \*file\_pointer)** - Writes a string on a file.

**int scanf(char \*format\_string, args)** – Reads formatted input from “stdin” (keyboard)

**int fscanf(FILE \*file\_pointer, char \*format\_string, args)** – Reads formatted input from a file.

**int sscanf(char \*buffer, char \*format\_string, args)** – Reads formatted input from a string.

**EOF** – end of file (constant with negative value)

**NULL** - null pointer (value 0)

```
#include <stdlib.h>
```

**double atof(char \*string)** – Converts a string into a floating point value.

**int atoi(char \*string)** – Converts a string into an integer value.

**int atol(char \*string)** – Converts a string into a long integer value.

**void exit(int val)** – Terminates the program returning the value ‘val’.

**EXIT\_FAILURE** – constant highlighting the unsuccessful termination of the program with exit(); non zero value.

**EXIT\_SUCCESS** - constant highlighting the successful termination of the program with exit(); zero value.

```
#include <string.h>
```

**char \*strcpy(char \*s1, char \*s2)** - Copies s2 in s1. Returns s1

**char \*strncpy(char \*s1, char \*s2, size\_t n)** – Copies the first "n" characters of s2 in s1. Returns s1.

**int strcmp(char \*s1, char \*s2)** - Compares s1 and s2 to determine the alphabetical order (<0, s1 precedes s2, 0 equal, >0 s1 follows s2)

**int strncmp(char \*s1, char \*s2, size\_t n)** – Compares the first "n" characters of two strings.

**int strlen(char \*string)** – Determines the length of a string.

**char \*strcat(char \*s1, char \*s2, size\_t n)** - Links s2 to s1. Returns s1

**char \*strncat(char \*s1, char \*s2, size\_t n)** - Links "n" characters of s2 to s1. Returns s1

**char \*strchr(char \*string, int c)** – Finds the first occurrence of the character ‘c’ in string;

returns a pointer to the first occurrence of c in s, NULL if not present.

**char \*strrchr(char \*string, int c)** – Finds the last occurrence of the character ‘c’ in string.

**char\* strstr(char\* s, char\* t)** – Returns a pointer to the first occurrence of t in s. returns NULL if not present.

**char\* strtok(char\* s, const char\* t)** – Decomposes s in tokens, the characters that limit the tokens are contained in t. returns the pointer to the token (NULL if any is found). At the first call the string to be decomposed is s and the characters delimiting the various tokens in t. To operate on the same string, at following calls NULL has to be passed instead of s.

```
#include <ctype.h>
```

**int isalnum(int c)** – True if ‘c’ is alphanumeric.

**int isalpha(int c)** – True if ‘c’ is an alphabetic character.

**int iscntrl(int c)** – True if ‘c’ is a control character.

**int isdigit(int c)** - True if ‘c’ is a decimal digit.

**int islower(int c)** - True if ‘c’ is lowercase.

**int isprint(int c)** - True if ‘c’ is a printable character.

**int ispunct (int c)** - True if ‘c’ is a punctuation character.

**int isspace(int c)** - True if ‘c’ is a space character.

**int isupper(int c)** - True if ‘c’ is uppercase.

**tolower(int c)** – Converts ‘c’ to lowercase.

**int toupper(int c)** – Convert ‘c’ to uppercase.

```
#include <math.h>
```

**int abs (int n)** – integer absolute value

**long labs(long n)** – long absolute value

**double fabs (double x )** – absolute value of x

**double acos(double x)** - arccosine

**double asin(double x)** - arcsin

**double atan(double x)** - arctangent

**double atan2(double y, double x)** – arctangent of y/x.

**double ceil(double x)** – round up value of x.

**double floor(double x)** – round down value of x.

**double cos(double x)** – cos (x in radians)

**double sin(double x)** – sin (x in radians)

**double tan(double x)** – tan (x in radians)

**double cosh(double x)** – hyperbolic cosine

**double sinh(double x)** – hyperbolic sin

**double tanh(double x)** – hyperbolic tangent

**double exp(double x)** - e<sup>x</sup>

**double log(double x)** - log(x).

**double log10 (double x )** – logarithm base 10

**double pow (double x, double y)** - x<sup>y</sup>

**int rand (void)** – random integer between 0 and RND\_MAX.

**int random(int max\_num)** – random value between 0 and max\_num.

**void srand(unsigned seed)** – initialize the sequence of random values

**double sqrt(double x)** – square root

```
#include <limits.h>
```

**INT\_MAX** – Maximum value that can be represented by int variable.

**INT\_MIN** – Minimum value that can be represented by int variable.

**LONG\_MAX** - Maximum value that can be represented by long variable.

**LONG\_MIN** - Minimum value that can be represented by long variable.

```
#include <float.h>
```

**FLT\_MAX, DBL\_MAX** - Maximum value that can be represented by float (or double) variable.

**FLT\_MIN, DBL\_MIN** - Minimum value that can be represented by float (or double) variable.