U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MASTER IN INFORMATICS AND COMPUTING ENGINEERING | 1ST YEAR
EIC0012 | PROGRAMMING | 2017/2018

PRACTICAL WORK Nº 1

Word Games - Data Preparation & First Developments

WORK OVERVIEW

The objective of this practical work is to develop two text processing programs and to create a data file that can be used as the basis for the development of several word games, such as scrabble, hidden words, crosswords, or others. The development of the programs must be made in a modular way, so that some of the functions can be reused in other programs to be developed in the future.

The purpose of the first program is to "isolate" the entries (headwords) in a dictionary, in order to produce a list of valid words. The dictionary is stored in a text file, as illustrated in appendix A. The list of resulting words must also be stored in a text file, as illustrated in appendix B.

The purpose of the second program is to implement and test some functions, that can be used in the future development of some word games, for doing common operations like: verify if a given word exists in a word list; verify if a given word can be made using a given set of letters or show all the words that can be made using a given set of letters.

LEARNING OBJETIVES

The development of these programs will give students the opportunity to practice their skills of C/C++ programming, namely:

- development of simple program interfaces;
- management of invalid keyboard inputs;
- formatting outputs;
- reading from and writing to text files;
- use of program control structures (selection and repetition);
- use of several types of data structures (*strings*, *arrays/vectors, structs, files*);
- use of functions.

PROGRAM SPECIFICATION

Program 1 - Extraction of a word list from a dictionary

This program must do the following:

- Read the contents of the dictionary, stored in a text file, isolating all the headlines and storing them in an internal data structure. In the given dictionary, an headline has the following characteristics: it only contains uppercase letters ('A'-'Z') or the characters ' ' (space), ';' (semicolon), '-' (hyphen), or ''' (apostrophe). If an headline contains more than one word/expression, decompose it into several strings, keeping only the strings that represent simple words (words that only have letters 'A'-'Z').
- Sort the resulting words, stored in the internal data structure.
- Remove duplicate words.
- Store the resulting words in a text file.

Some extracts of the dictionary file and of the resulting words file are shown in annexes A and B, respectively.

The interface of the program, named **procdic**, must be similar to the following:

```
EXTRACTION OF WORD LIST FROM DICTIONARY
=======================================

Dictionary file ? 29765-8.txt
Word list file  ? 29765-8_w.txt

Extracting simple words from file 29765-8.txt,
beginning with letter ...

A
...............................................................................
B
...........................................................
C
....................................................................................................
.....
D
...................................................................
E
..................................................
F
............................................
G
....................................
H
.........................................
I
.....................................................
J
........
K
..........
L
....................................
M
..............................................................
N
.....................
O
................................
P
....................................................................................................
.
Q
.......
R
.....................................................
S
....................................................................................................
................................
T
............................................................
U
..........................
V
...................
W
...........................
X
..
Y
...
Z
....

Number of simple words = ??????

Sorting words ...

Removing duplicate words ...

Number of non-duplicate simple words = ??????

Saving words into file 29765-8_w.txt ...

End of processing.
```

The beginning of the processing of the words that start with 'A', with 'B', and so on, must be signaled by writing the corresponding letter on the screen. For each set of 100 headlines detected in the dictionary, a dot must be written on the screen.

Program 2 - Playing with words

This program, named **playwords**, must do the following:

- Read the contents of the file containg the word list, produced by program **procdic**, and store the words in an internal data structure.
- Show a menu that allows the user to choose one of the following actions:
    - Check if a user-entered word belongs to the word list.
    - Randomly choose a word from the word list, scramble the letters of the word, and show them to the user. Then ask the user to guess the word. The user must be given more than one chance (at least 3) to guess the word, because it may happen that different words can be made with the same set of letters.
    - Ask the user a set of N letters and show all the words present in the dictionary that can be built using the set of ~~the~~ given letters ~~or any subset of them~~.

- Randomly choose a set of N letters (it may contain repeated letters) and ask the user to build a valid word, then verify if the word belongs to the word list or not. This set of letters must be extracted from a larger set that may contain multiple instances of each letter. The number of instances of each letter must be proportional to the frequency of ocurrence of each letter in the list of all words; the letter with the lowest frequency must have two instances in the larger set.
- Read a string containing one or more wildcard characters ('*' or '?') and show all the words in the dictionary that match the given string (see explanation below).

The matching of two strings one of which contains wildcards consists of comparing the two strings, using the following rules: '*' matches with zero or more instances of any characters; '?' matches with one character. For example:

- "F?LL" matches with "FALL", "FELL" , "FILL" and "FULL";
- " FU*LL" matches with "FULFILL", "FULL" and "FUMERELL";
- "FU?*LL" matches with "FULFILL" and "FUMERELL";
- "FU?????" matches with "FUBBERY" and "FUCATED".

A function that implements a matching algorithm for two C-strings, one of which may have wildcards, is given in annex C.

This program must be case-insensitive.

You are free to choose the interface of the program.

PROGRAM DEVELOPMENT

Code writing

When writing the program code you should take into account the suggestions given in class, specially the ones concerning the following issues:

- Adequate choice of identifiers of types, variables and functions.
- Code commenting.
- Adequate choice of the data structures to represent the data manipulated by the program.
- Modular structure of the code.
- Separation, as much as possible, of data processing from program input/output.
- Code robustness. Precautions should be taken in order to prevent the programs to stop working due to incorrect input by the user, specially values outside the allowed ranges, non existent files, and so on.
- Code efficiency. Try to write efficient code (for example, when searching for a word in a sorted word list).

Code testing

Start with a short dictionary, for example, an extract of the original dictionary, covering all types of headlines.

Code improvement

For the implementation of the programs, start by using a **vector<string>** to store the words. When other STL containers and STL algorithms are introduced in the lectures, modify your code in order to take as much profit as possible from them; for example, use a **set** to store the words, or other data structures and algorithms that you find useful. Notice the differences between the two implementations.

WORK SUBMISSION

- Important note: Although the first practical work will not be scored, it is compulsory to do it in order to get approval in the practical part of the course. It is necessary to develop the program, to deliver and present it (if/when requested). ~~The first practical work is an single student's work.~~
- Create a folder named **TxGyy**, in which **x** represents the class number (Portuguese "turma") and **yy** represents the group number (with 2 digits), for example, **T5G07**, for the group 7 from class ("turma") 5. Create two subfolders, one named **P1** and another named **P2**. Copy to subfolder **P1** the source code (only the files with extension **.cpp** and **.h**, if existing) of Program 1 (**procdic**) and to subfolder **P2** the source code of Program 2 (**playwords**). If you have developed the improved versions, taking as much profit as possible from the STL library, create also subfolders **P1_STL** and **P2_STL** for these versions. Include also a file, **ReadMe.txt** (in simple text format), indicating the development state of the programs, that is, if all the objectives were accomplished or, otherwise, which ones were not achieved, and also what improvements were made, if any. Do not send the data files (the dictionary and the word list produced by Program 1).
- Compress the content of the folder **TxGyy** into a file named **TxGyy.zip** or **TxGyy.rar**, depending on the compression tool that you use, and upload that file in the FEUP Moodle's page of the course. Alternative ways of delivering the work will not be accepted.
- Deadline for submitting the work: **13th/April/2018 (at 23:55h)**.

ANNEX A

Sample of the "Webster's Unabridged Dictionary by Various"

Some samples from the "Webster's Unabridged Dictionary by Various", available from "Project Guttenberg" (http://www.gutenberg.org/), at http://www.gutenberg.org/files/29765/ . The ellipsis ("...") indicate ommited text.

```
The Project Gutenberg EBook of Webster's Unabridged Dictionary, by Various

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.net

Title: Webster's Unabridged Dictionary

Author: Various

Release Date: August 22, 2009 [EBook #29765]
[Last updated: May 18, 2014]

Language: English

Character set encoding: ISO-8859-1

*** START OF THIS PROJECT GUTENBERG EBOOK WEBSTER'S UNABRIDGED DICTIONARY ***


Produced by Graham Lawrence


A
A (named a in the English, and most commonly ä in other languages).

Defn: The first letter of the English and of many other alphabets.
The capital A of the alphabets of Middle and Western Europe, as also
the small letter (a), besides the forms in Italic, black letter,
etc., are all descended from the old Latin A, which was borrowed from
the Greek Alpha, of the same form; and this was made from the first
letter (Aleph, and itself from the Egyptian origin. The Aleph was a
consonant letter, with a guttural breath sound that was not an
element of Greek articulation; and the Greeks took it to represent
their vowel Alpha with the ä sound, the Phoenician alphabet having no
vowel symbols. This letter, in English, is used for several different
vowel sounds. See Guide to pronunciation, §§ 43-74. The regular long
a, as in fate, etc., is a comparatively modern sound, and has taken
the place of what, till about the early part of the 17th century, was
a sound of the quality of ä (as in far).

2. (Mus.)

Defn: The name of the sixth tone in the model major scale (that in
C), or the first tone of the minor scale, which is named after it the
scale in A minor. The second string of the violin is tuned to the A
in the treble staff.
 -- A sharp (A#) is the name of a musical tone intermediate between A
and B.
 -- A flat (A) is the name of a tone intermediate between A and G.

A per se Etym: (L. per se by itself), one preëminent; a nonesuch.
[Obs.]
O fair Creseide, the flower and A per se Of Troy and Greece. Chaucer.

...

AAM
Aam, n. Etym: [D. aam, fr. LL. ama; cf. L. hama a water bucket, Gr.

Defn: A Dutch and German measure of liquids, varying in different
cities, being at Amsterdam about 41 wine gallons, at Antwerp 36½, at
Hamburg 38¼. [Written also Aum and Awm.]

AARD-VARK
Aard"-vark`, n. Etym: [D., earth-pig.] (Zoöl.)

Defn: An edentate mammal, of the genus Orycteropus, somewhat
resembling a pig, common in some parts of Southern Africa. It burrows
in the ground, and feeds entirely on ants, which it catches with its
long, slimy tongue.

AARD-WOLF
Aard"-wolf`, n. Etym: [D, earth-wolf] (Zoöl.)

Defn: A carnivorous quadruped (Proteles Lalandii), of South Africa,
resembling the fox and hyena. See Proteles.

AARONIC; AARONICAL
Aa*ron"ic, Aa*ron"ic*al, a.

Defn: Pertaining to Aaron, the first high priest of the Jews.

AARON'S ROD
Aar"on's rod`. Etym: [See Exodus vii. 9 and Numbers xvii. 8]

1. (Arch.)

Defn: A rod with one serpent twined around it, thus differing from
the caduceus of Mercury, which has two.

2. (Bot.)

Defn: A plant with a tall flowering stem; esp. the great mullein, or
hag-taper, and the golden-rod.

AB-
Ab-. Etym: [Latin prep., etymologically the same as E. of, off. See
Of.]

Defn: A prefix in many words of Latin origin. It signifies from, away
, separating, or departure, as in abduct, abstract, abscond. See A-
(6).

...
```

ABELIAN; ABELITE; ABELONIAN
A*bel"i*an, A"bel*ite, A`bel*o"ni*an, n. (Eccl. Hist.)

Defn: One of a sect in Africa (4th century), mentioned by St. Augustine, who states that they married, but lived in continence, after the manner, as they pretended, of Abel.

...

ABOVE-MENTIONED; ABOVE-NAMED
A*bove"-men`tioned, A*bove"-named`(#), a.

Defn: Mentioned or named before; aforesaid.

...

ANIDIOMATIC; ANIDIOMATICAL; UNIDIOMATIC; UNIDIOMATICAL
An*id`io*mat"ic*al, a. Etym: [Gr. idiomatical.]

Defn: Not idiomatic. [R.] Landor.

...

ABRAUM; ABRAUM SALTS
A*braum" or A*braum" salts, n. Etym: [Ger., fr. abräumen to remove.]

Defn: A red ocher used to darken mahogany and for making chloride of potassium.

...

ATELETS SAUCE; SAUCE AUX HATELETS
A`te*lets" sauce or  Sauce` aux ha`te*lets". [F. hâtelet skewer.]

Defn: A sauce (such as egg and bread crumbs) used for covering bits of meat, small birds, or fish, strung on skewers for frying.

...

BIRD'S NEST; BIRD'S-NEST
Bird's" nest`, or Bird's-nest, n.

1. The nest in which a bird lays eggs and hatches her young.

2. (Cookery)

Defn: The nest of a small swallow (Collocalia nidifica and several allied species), of China and the neighboring countries, which is mixed with soups.

Note: The nests are found in caverns and fissures of cliffs on rocky coasts, and are composed in part of algæ. They are of the size of a goose egg, and in substance resemble isinglass. See Illust. under Edible.

3. (Bot.)

Defn: An orchideous plant with matted roots, of the genus Neottia (N. nidus-avis.) Bird's-nest pudding, a pudding containing apples whose cores have been replaces by sugar.
 -- Yellow bird's nest, a plant, the Monotropa hypopitys.

...

BOWER-BARFF PROCESS
Bow"er-Barff" proc`ess . (Metal.)

Defn: A certain process for producing upon articles of iron or steel an adherent coating of the magnetic oxide of iron (which is not liable to corrosion by air, moisture, or ordinary acids). This is accomplished by producing, by oxidation at about 1600° F. in a closed space, a coating containing more or less of the ferric oxide (Fe_2O_3) and the subsequent change of this in a reduced atmosphere to the magnetic oxide (Fe_2O_4).

...

BRAHMANIC; BRAHMANICAL; BRAHMAN-ICAL; BRAHMINIC; BRAHMINICAL; BRAHMIN-ICAL
Brah*man"ic, -ic*al , Brah*min"ic (, *ic*al (, a.

Defn: Of or pertaining to the Brahmans or to their doctrines and worship.
...

FUGUIST
Fu"guist, n. (Mus.)

Defn: A musician who composes or performs fugues. Busby.

-FUL
-ful. Etym: [See Full, a.]

Defn: A suffix signifying full of, abounding with; as, boastful, harmful, woeful.

...

ZYTHEPSARY
Zy*thep"sa*ry, n. Etym: [Gr.

Defn: A brewery. [R.]

ZYTHUM
Zy"thum. Etym: [L.fr. Gr.

Defn: A kind of ancient malt beverage; a liquor made from malt and wheat. [Written also zythem.]

...

ANNEX B

Word list produced by Program 1

The dictionary used to produce this word list was the "Webster's Unabridged Dictionary by Various" (see Annex A). The words in red are those that are present in the sample shown in annex A. The ellipsis ("...") indicate ommited words.

```
A
AAM
AARONIC
AARONICAL
AB
ABACA
ABACINATE
ABACINATION
ABACISCUS
ABACIST
ABACK
ABACTINAL
ABACTION
ABACTOR
ABACULUS
ABACUS
ABADA
...
ABECEDARY
ABED
ABEGGE
ABELE
ABELIAN
ABELITE
ABELMOSK
ABELONIAN
ABERR
...
ANI
ANICUT
ANIDIOMATIC
ANIDIOMATICAL
...
BIRD
BIRDBOLT
BIRDCAGE
BIRDCALL
BIRDCATCHER
BIRDCATCHING
BIRDER
BIRDIE
...
BRAHMA
BRAHMAN
BRAHMANESS
BRAHMANI
BRAHMANIC
BRAHMANICAL
BRAHMANISM
BRAHMANIST
BRAHMIN
BRAHMINIC
BRAHMINICAL
BRAHMINISM
BRAHMINIST
...
UNIDEAL
UNIDIMENSIONAL
UNIDIOMATIC
UNIDIOMATICAL
UNIFACIAL
UNIFIC
...
ZYTHEPSARY
ZYTHUM
```

ANNEX C

A simple wildcard matching function

```
//////////////////////////////////////////////////////////////////////////
// WildcardMatch
//   str      - Input string to match
//   strWild - Match mask that may contain wildcards like ? and *
//
//   A ? sign matches any character, except an empty string.
//   A * sign matches any string inclusive an empty string.
//   Characters are compared caseless.
//
// ADAPTED FROM:
// https://www.codeproject.com/Articles/188256/A-Simple-Wildcard-Matching-Function

bool wildcardMatch(const char *str, const char *strWild)
{
  // We have a special case where string is empty ("") and the mask is "*".
  // We need to handle this too. So we can't test on !*str here.
  // The loop breaks when the match string is exhausted.
  while (*strWild)
  {
    // Single wildcard character
    if (*strWild== '?')
    {
      // Matches any character except empty string
      if (!*str)
        return false;

      // OK next
      ++str;
      ++strWild;
    }
    else if (*strWild== '*')
    {
      // Need to do some tricks.

      // 1. The wildcard * is ignored.
      //    So just an empty string matches. This is done by recursion.
      //    Because we eat one character from the match string,
      //    the recursion will stop.
      if (wildcardMatch(str,strWild+1))
        // we have a match and the * replaces no other character
        return true;

      // 2. Chance we eat the next character and try it again,
      //    with a wildcard * match. This is done by recursion.
      //    Because we eat one character from the string,
      //    the recursion will stop.
      if (*str && wildcardMatch(str+1,strWild))
        return true;

      // Nothing worked with this wildcard.
      return false;
    }
    else
    {
      // Standard compare of 2 chars. Note that *str might be 0 here,
      // but then we never get a match on *strWild
      // that has always a value while inside this loop.
      if (toupper(*str++)!=toupper(*strWild++))
        return false;
    }
  }

  // Have a match? Only if both are at the end...
  return !*str && !*strWild;
}
```