SMS Compress Documentation

Dokumentace úlohy SMS: SMS Compress v Perl do IPP 2010/2011

Jméno a příjmení: Dávid Molnár

Login: xmolna02

Parsing command line parameters

We are examining each item in `@ARGV` using a regular expression. If a correct switch (e.g. `-c`) has found, store it in a hash : `%sw{c}=1;`. If found an illegal switch, exit with error. Then test for illegal combinations and duplicates of switches.

Remove czech diacritics

At first we read the entire input file (or standard input) into a scalar variable (`$input`). We could use `tr//` to remove the diacritics, but I have decided to use this method:

1. Create a "dictionary" (hash) to store the relationships between national and ASCII characters, e.g. `%hash = (á => a);`
2. Split `$input` variable into pieces – convert it to an array. Every item in the array will correspond to one character in the input string.
3. Use `map` function to create a new array containing only ASCII characters: with `grep` find the actual examined character in the keys of the "dictionary" hash. If found, use the corresponding value from hash, if not, use the original value.
4. Join the final array into a scalar variable.

Camel notation

I have divided this task into 3 parts:

1. Find the first and the other letters of each word.
2. Capitalize the first letter.
3. Remove all white spaces.

I have used `s//` to find and capitalize the first letter of words with different regular expressions for each switch (`-c`, `-c` with `-a`, `-c` with `-b`).

Shortening and expansion

The main part of this task is the parsing of XML dictionary file. It is accomplished by `XMLin` method of the `XML::Simple` library. After successfully parsing loop each rule and use `s//` to replace.

If the rule is case-sensitive, don't use `/i` switch in `s//`.

Conclusion

In this project we can see the advantages of PERL in string processing. The regular expresions are very smart and easy to use.