Data Coding and Compression 2012/2013: Project #3

Topic: GIF to BMP Image Format Conversion with LZW Decompression
Submission deadline: until 3rd May 2013 through web-based information system at FIT
Submission type: in electronic form - ZIP compressed archive
Number of points: max. 30
Remarks: ZIP archive with the project assignment contains also some templates in "*contrib*" directory and example image file in "*test*" directory for the evaluation of final application
Questions: simekv@fit.vutbr.cz

Assignment:
Prepare an implementation of library and front-end application (typically, a console-like) which will handle the conversion of GIF image file (Graphics Interchange Format) into the BMP (Microsoft Windows Bitmap) image file format.

Detailed instructions overview:
Interface of the library will contain prototype of a function that will be responsible for conversion of GIF89a into BMP. Besides that, it's necessary to define a type of structure which keeps the information about the size of GIF and BMP files in bytes. The library must implement such functionality that it's possible to process 8-bit static GIF format. The conversion of 1 up to 7-bit format doesn't have to be supported. The resulting BMP image must employ RLE compression.

An appropriate record data type should conform to the following specification::

```
typedef struct{
        int64_t bmpSize;
        int64_t gifSize;
} tGIF2BMP;
```

Conversion function prototype will take this form:

```
/* gif2bmp – coding process  record
inputFile – input file (GIF)
outputFile – output file (BMP)
function return value – 0 conversion process ended successfully, -1 an error occurred, or the format of input
                        GIF file isn't supported */

int gif2bmp(tGIF2BMP *gif2bmp, FILE *inputFile, FILE *outputFile);
```

The implementation of library must contain all the essentials compression and decompression functions, and set of relevant auxiliary functions (tree-based data structure, dictionary management routines, etc).

b) implementation of application:
The resulting application *gif2bmp* is based on the library for GIF-to-BMP conversion where these command line parameters will be supported:

-i <ifile> name of the input file <ifile>. If this parameter is missing, the application should consider standard input (*stdin*) as the source of data for processing.

-o <ofile> name of the output file <ofile>. If this parameter is missing, the application should consider standard output (*stdout*) as the destination for output data.

-l <logfile> name of the file with output report <logfile>. If this parameter is missing, the file with report won't be created.

-c selection coding operation (compression) applied on the input data.

-x selection of decoding operation (decompression) applied on the input data.

-h application help displayed on standard output (*stdout*), application will terminate afterwards.

Output report will contain login, size of the original file (in bytes), size of the converted file (in bytes) with regard to the example bellow. An example of output report received from student Pepa Zdepa is given:

```
login = xzdepa97
uncodedSize = 16384
codedSize = 14937
```

c) source code comments:
Source code should be commented at an appropriate level, so any additional defense won't be required. Closer attention should be given to parameters and behavior of functions, data types, local variables (if it can't be concluded directly from their names).

All the source code files must come with a heading with name, surname and login of the author. Furthermore, there must be date of creation, name and brief description of a given file.

d) compiler and development environment:
It must be possible to compile both library and application with compiler *GCC 4.2.X* (for example, available on server *merlin*). Its compatibility with *GNU MAKE* environment is also required. From technical point of view, the resulting application will be produce by invoking compilation process by running *make* command upon valid *Makefile*.

e) documentation:
Concise documentation of library and application must be delivered. An integral part of it is, of course, brief analysis of the algorithm implementing adaptive Huffman method. The content of the documentation shouldn't span more than 2 pages A4. Accepted format is PDF or DOC file.

f) submission requirements:
ZIP archive, where its name should follow the scheme *kko.proj3.”login”.zip* (ie. for student Pepa Zdepa *kko.proj3.xzdepa97.zip*), must come with the directory *kko.proj3.”login”* (for example, kko.proj3.xzdepa97) that contains:

- *gif2bmp.h* – library interface
- *gif2bmp.c* – library implementation
- *main.c* – application
- *Makefile* – definition of compilation process through the use of *make* command
- *gif2bmp.pdf* – documentation

If it turns out to be meaningful for the solution (better organization of source codes, modularity of the application), you can create even set of additional files (various auxiliary libraries, header files, etc), besides the required ones. In such case, don't forget to put these files in the ZIP archive to be submitted.

Hints and suggestions:

- Very important property is application portability. Various platforms may use different size of data types *int8_t*, *int32_t*, *u_int_8_t*, etc. Appropriate definitions are to be found in library *sys/types.h*.
- Memory based problems (segmentation faults, etc) can be traced and eliminated with tool *valgrind*. It can be used only with application, which is compiled with the following commands: *make debug* (or *gcc -ggdb3*). It's highly recommended to take this suggestion into account even if everything looks good on a first sight. The weak points behind the approach to using memory may be revealed (ie. memory failure) as late as with transition to different platform.
- There exists very useful tool *DDD* (DataDisplayDebugger) under GPL license, which is in fact the graphical extension of console-based *gdb* debugger. Again, it's recommended to compile the application with *make debug*.
- It's useful to adopt the function *getopt* (or *getopt_long*) for the processing of command line parameters. Its interface is define in unistd.h (resp. getopt.h).
- The implementation of selected data structures and their associated operations may be significantly facilitated when using STL library (C++ Standard Template Library).

References:

[1] Salomon, D.: "Data Compression, The Complete Reference," NY, USA, Springer, 2000, ISBN-0-387-95045-1

[2] Supporting materials for "Data Coding and Compression" course, Brno, FIT VUT, 2006

[3] Sayood, K.: "Losless Compression Handbook", San Diego, CA, USA, Elsevier Press 2003, ISBN 0-12-620861-1

[4] Sayood, K: "Introduction to Data Compression", San Francisco, CA, USA, Elsevier Press 2006, ISBN -0-12-620862-X

[5] "The Graphics File Formats Page". URL link: http://www.dcs.ed.ac.uk/home/mxr/gfx/2dhi.html