Subjects for Laboratory Test in Week 14

DRAFT - Not final!!

Lab 2

1. Write a C program to compute the entropy of a file.

- The program shall receive the name of the file as a command-line argument: `entropy.exe myfile.txt`
- The program should follow the following steps:
  - Open the file for reading (in binary format)
  - Count the number of apparitions of every byte value:
    * hold an array of 256 counters, one for every possibly byte value
    * repeatedly read one byte from the file
    * increment the counter corresponding to the byte read
    * also store and increment a counter for the total number of bytes
  - Compute the probability of every byte value: divide each byte counter to the global counter
  - Compute the entropy, based on the probabilities
  - Show the result

Lab 5

1. Write a C program to perform a linear block encoding of every byte from a given data file. The program shall be called as follows:

   `Encode.exe code.dat input.txt output.txt`

- The arguments are:
  - `code.dat`: a file containing the code to be used (known as the "codebook" file)
  - `input.txt`: the file to encode
  - `output.txt`: the output (encoded) file

The codebook file contains a vector of 256 elements of the following structure type:

```
typedef struct
{
    int len;                /* length of code, in bits */
    unsigned long code;     /* the first "len" bits are the codeword */
} CODE32BIT;
```

- The program will follow the following steps:
  - Read the full vector from the codebook file;
  - Allocate an array named `out` of `unsigned char` of max size 10MB (i.e. 10000000 bytes);
  - The, open the input file and read every byte in a loop. For each byte do the following:
  - Write the code for the byte, bit by bit, in the `out` vector. You need to keep track of the number of bits written, in order to continue writing from where the previous code stopped.
  - Write the output data to the output file, as follows:
    - Open the second file for writing
    - Write first the total number of bits
    - Write afterwards the vector `out`, but not more than the number of bytes actually used for coding
    - *Note: when decoding the file, we will read back the data in the same order.*

Lab 8

1. Write a C program that computes and appends the parity bit for every byte in a data file. The program shall be called as follows:

   ```
   Parity.exe input.dat output.dat
   ```

   - The arguments are:
     - `input.dat`: the input file
     - `output.dat`: the output file produced by the program (12.5% larger than the input)
   - The program should consist of the following steps:
     - declare two large vectors of `unsigned char`, for input and output bits
     - open the input file and read everything into the input vector
     - for every group of 8 bits from the input vector:
       - copy them to the output vector
       - compute the parity bit
       - append it to the output vector
     - write the output vector to the output data file

Lab 9

1. Write a C program that performs encoding of a data file with the Hamming (8,4) SECDED code.

   The program shall be called as follows:

   `HammingEncode.exe original.dat encoded.dat`

   - The arguments are:
     - `original.dat`: the original input file
     - `encoded.dat`: the encoded output file
   - The program should consist of the following steps:
     - declare two large vectors of `unsigned char`, for input and output bits
     - open the input file and read everything into the input vector
     - for every group of 4 bits from the input vector:
       * compute the control bits c_0, c_1, c_2, c_4
       * write all the 8 bits in the correct order in the output vector
       * advance by 4 and repeat
     - write the output vector to the output data file

   Hamming (8,4) SECDED encoding procedure operates on a block of 4 information bits (denoted as i_3, i_5, i_6, i_7) and produces a block of 8 output bits:

   $$\mathbf{c} = c_0 c_1 c_2 i_3 c_4 i_5 i_6 i_7$$

   The bits denoted with c are parity bits (or *control* bits), and are computed as follows:

   $$\begin{cases} c_0 = i_3 \oplus i_5 \oplus i_6 = c_1 \oplus c_2 \oplus i_3 \oplus c_4 \oplus i_5 \oplus i_6 \oplus i_7 \\ c_1 = i_3 \oplus i_5 \oplus i_7 \\ c_2 = i_3 \oplus i_6 \oplus i_7 \\ c_4 = i_5 \oplus i_6 \oplus i_7 \end{cases}$$

Lab 11

1. Write a C program that computes the CRC-16 value of a data file and appends it to the file.

   The program shall be called in two possible ways:

   a. with two arguments. In this case the program takes the first as input and produces the encoded file as output.

   `CRC16.exe original.dat encoded.dat`

   b. with one argument. In this case the program takes the encoded file as input and checks if the CRC is OK or not.

`CRC16.exe encoded.dat`

- The arguments are:
  - `original.dat`: the input file (original / encoded)
  - `encoded.dat` [optional]: the encoded file, with the CRC value appended
- The program should consist of the following steps:
  - define an array g with the values 10100000000000011
  - declare one large vector of `unsigned char` for input bits
  - open the input file and read everything into the input vector
  - for every bit in the input vector
    - if the bit is 1:
      - do XOR of the next 17 bits with the bits in g
  - there will be 16 bits remaining at the end of the original input vector (the CRC-16 value)
  - then:
    a. If the program is called with two arguments:
       - write the vector to the output data file, including the CRC-16 value at the end

    b. If the program is called with one argument:
       - if the CRC-16 value is 0, display "File OK\n", otherwise display "Data corrupted\n"