CSI 2131 File Management | Winter 2002
Computer Science | University of Ottawa

Homework Assignment #2 (100 points, weight 10%)
Due: Tuesday, February 26, 2:00 p.m., at the box CSI2131A or CSI2131B (MCD, 3rd floor)

---

1 Problem 1: Programming Assignment (85 marks)

You are going to program a **Lempel-Ziv decoder**, that is, your program will uncompress files that have been compressed using the Lempel-Ziv algorithm discussed in class.
As seen in class, the compressed file corresponding to the index sequence:

```
Index : 1  2  3  4  5  6  ...

        0a|1a|0b|1b|3a|5c|...            (sample 1)
```

is actually encoded as the following sequence of bits:
`01100001101100001000110001001011000100110110000110101100011...`
noting that the ASCII code for `a` is `01100001`, for `b` is `01100010` and for `c` is `01100011`.
This sequence of bits is then packed into 8-bit bytes and written to the file.

Recall that the number present within an index i, can be any previous index, namely $0, 1, \ldots, i-1$. For this reason, the number of bits reserved to store the number within index i is the number of bits required to encode $i-1$ (number of bits in the binary representation of $i-1$). So the number within index 2 will be represented using 1 bit, the numbers within indexes 3 and 4 using 2 bits, within indexes 5,6,7 and 8 using 3 bits, and so on. Since the first index contains always the number zero, this number is not included in the encoded file at all.
Your program will:

- receive an input file that has been encoded as described above;
- decode/uncompress the input file, producing an output file;
- print the number of bytes in both files and the compression ratio:
  (size of compressed file)/(size of uncompressed file)

To facilitate your task of reading bytes from files and converting them into bits for your decoding, we have provided you with the class: `IBitStream` (contained in files `ibitstream.cpp`, `ibitstream.h`).
You will have to gather the bits[1] that you read into the appropriate units for your program manipulation. In the example above, the first 8 bits must be combined into a character so

[1] Use bit shifting operator `<<` for `char` and `int` types. Example `samplebit.cpp` provided in the web page.

that you detect that `"a"` is the letter within index 1. Then the next bit is understood to be the number in index 2 and the next 8 bits must be combined into a character that represents the letter `"a"` which is the letter in index 2. Then, the next 2 bits must be combined into an integer to represent the number within index 3, then the next 8 bits must be combined into a character that represents the letter `b`, and so on.

After the process described in the previous paragraph is complete, you will have the data in a form equivalent to `(sample 1)`, more precisely, you will know for each index `i`, its pair: `(number,letter)`. You may find it convenient to place this pair into an array indexed by `i` (either an array of `struct` with fields `number` and `letter` or two arrays `number[i]` and `letter[i]`). The decoding of each index can be easily done by going backwards in this array, concatenating the appropriate characters. In the example above, we could use the array:

	number	letter
1	0	a
2	1	a
3	0	b
4	1	b
5	3	a
6	5	c
⋮	⋮	⋮

For instance, after decoding indexes 1 to 5, as the sequence
`aaababba`
we would decode index `6` as follows:

- `(5,c)` indicates that the last character is `c`, and we must go back to index `5`;
- `(3,a)` indicates that the previous character is `a` (the last two characters are now known to be `ac`), and we must go back to index `3`;
- `(0,b)` indicates that the previous character is `b` (last three characters are now known to be `bac`), and we are done since we got number 0.
- So, index `6` was decoded as `bac`.

For more details about Lempel-Ziv encoding/decoding, please refer to the lecture notes.

Note: Since the encoded file could have a number of bits that is not a multiple of 8, we have applied the following convention into our encoding: the last index is composed of the pair (0,etx), where "etx" is the character with ASCII code equal to 3.

Standards for your program and what to hand in

Marks will be deducted if you don't follow these standards. Input should be standard. Use the partial code provided by us in the course web page: `a2.cpp`. In this code, the input of the file names is already done for you.

Your program will be compiled and tested with Microsoft Visual C++ compiler. Failure to compile or run under this compiler will be consider as a failure in your code. As before, the project type must be `W32 Console Application`.

Your project should be called `"a2"` and contain only the following files:

- `a2.cpp`: main program and any other auxiliary procedures
- `LZdecoder.h, LZdecoder.cpp`: definition and implementation for a class that implements the Lempel-Ziv decoder.
  Details on how to implement this class are up to the programmer, but we give some suggestions. LZdecoder should provide a method "decode" or "uncompress" which will decode the compressed input file into the uncompressed output file.
- `ibitstream.cpp, ibitstream.h` (provided by us)

You should hand in, on a labeled large envelope:

- stapled sheets of paper of your handwritten or typed answer for Problem 2;
- a printout of every file in your program `(*.h, *.cpp)`;
- a printout of the output of your program for standard inputs (encoded files provided by us) available from the course web page;
- a 3.5-inch DOS/Windows diskette containing only: the project workspace with all the relevant files in the project `(*.h, *.cpp, *.exe)`.

The envelope as well as all pages handed in should contain: student name, student number, course code (CSI2131), section (A or B), instructor's name, lab group, lab TA name.
All programs handed in the diskette should also contain all the information mentioned above (student name, student number, etc); the diskette itself must be labeled with the same information. Don't forget to always keep a copy of the programs and written questions of the assignment that you hand in.

2 Problem 2: Written Assignment (15 marks)

2.1 Compression using Huffman code (5 marks)

1. Construct a Huffman tree for the following letter/frequency pairs:
   (A,6), (C, 10), (E, 40), (K, 12), (L, 9), (U, 23).
   Label the edges of the tree so that each left edge receives the value "0" and each right edge receives the value "1". Use the convention that for any node in the tree, its left child is labeled with a smaller frequency than its right child.
2. Using the Huffman tree you built in Part 1, decode the following message: 1110110011110
3. Using the Huffman tree you built in Part 1, encode the message "luck".

4. Calculate the average number of bits per character used in the encoding of a message containing the letters (with the corresponding frequencies) given in Part 1. Show your work.

2.2 Compression using Lempel-Ziv code (5 marks)

1. Construct a Lempel-Ziv Encoding of the following string:
   "CAN A CAT PUT ON A CAP?". Show the tree built from your encoding.

2. How many bits are necessary to encode the above message? How much space, then, gets saved by your encoding (in number of bits or bytes)?

3. Decode the following encoded message: `"|0C|0A|0N|0 |1A|3 |5N|0!|"`. Note that this has nothing to do with Part 1.

2.3 Reclaiming spaces on files (5 marks)

Consider the following file containing variable-length records and fields. The record-length is indicated by a length indicator at the beginning of each record. Fields are separated by delimiter: '|'. The file contains 4 records: R1 of size 24, R2 of size 36, R3 of size 19 and R4 of size 29. (Note: in the actual file, no carriage-return was inserted after the second record. It was inserted here, only for the sake of readability)

```
24Smith|James|Dec|17|1973|36Dupont-Sur-Seine|Michel|Feb|22|1970|

19Elk|Ann|Mar|6|1976|29Shelling|Anthony|29|May|1969|
```

Draw the contents of the AVAIL LIST and those of the data file after all of the following steps have been completed: record R3 has been deleted, record R5 of size 25 has been added, record R4 has been deleted and record R6 of size 17 has been added (all in this order), if

1. a FIRST-FIT placement strategy has been used and AVAIL LIST has been implemented as a stack.

2. a WORST-FIT placement strategy has been used.

3. a BEST-FIT placement strategy has been used.