Hints for the LZW Project

This project is on a larger scale than the previous lab assignments, and you need to manage the time of your team wisely to be able to complete this project within the given time frame. It is unlikely that you will be able to complete the project if you start programming shortly before the deadline.

The extended deadline is: Friday, October 10

- Study the LZW compression program by Mark Nelson. Get familiar with the abstract algorithm in pseudocode, and then with the implementation details.
- For your implementation, you can assume that the uncompressed input text is stored in the data segment in the form:

```
        .data
inpstr: .ascii  "abracadabradiblabladidududidada"
        .ascii  "makethistextsmallmakethistexttiny"
        .ascii  "abracadabradiblabladidududidada"
        .asciiz "hokuspokussneezeandsqueeze"
```

- The encoding and decoding process of your assembly program is entirely performed within the memory because there are no file I/O operations available in `spim`.
- You need to allocate memory for the hash tables of the *encoder* and of the *decoder*. Each hash table consists of an array of integers containing the code symbols, and array of integers containing the code of the prefix, and an array of bytes containing the code of the character suffix.
- The encoded symbols are stored in an array of integers (representing the channel). We make the following simplification: You do not need to implement the buffering mechanism (Nelson's input_code and output_code procedures), since this belongs to the file I/O part. Store the code symbols that are transmitted by the encoder in the array representing the channel.
- Write a procedure that calculates the file length (in bytes) of the compressed file, based on the information given in the channel and the number of bits used to represent each symbol (BITS in Nelson's code). Should correspond to the file length generated by Nelson's lzw implementation.

- Include the following printf statement into the code generation routine of Nelson's compress procedure:

```
...
if (next_code <= MAX_CODE) {
   printf("code %d prefix %d suffixc %d\n",
              next_code,string_code,character);
   code_value[index]=next_code++;
   prefix_code[index]=string_code;
   append_character[index]=character;
...
```

  This printf statement illustrates the code symbol generation of the encoder. For example, if the input is `ABAAABBAA`, then the modified program will print

```
Compressing...
code 256 prefix 65 suffix 66
code 257 prefix 66 suffix 65
code 258 prefix 65 suffix 65
code 259 prefix 258 suffix 66
code 260 prefix 66 suffix 66
code 261 prefix 257 suffix 65
Expanding...
```

  You can test your encoder by comparing the symbol generation with Nelson's C program, even if you do not have completed the decoder. Whenever the encoder attempts to include some new string into the hash table, then it will output the code of the prefix, and add it to the channel.

- If the hash table gets full, then stop including new symbols, as in Nelsons C code (in practice, you might want to clear the table or use some LRU scheme).