Guide to Assignment 3 Programming Tasks

CSE 2312
Computer Organization and Assembly Language Programming
Vassilis Athitsos
University of Texas at Arlington

Task 1

- Goal: convert a data file from one endian format to the other.
- Program flow:
  - Open input and output files.
  - While there is input data to be processed:
    - Read the next record from the input file.
    - Convert the record to the other endian format.
    - Save the record to the output file.
  - Close input and output files.

Task 1

- Goal: convert a data file from one endian format to the other.
- Program flow:
  - Open input and output files.
  - While there is input data to be processed:
    - Read the next record from the input file.
    - **Convert the record to the other endian format.**
    - Save the record to the output file.
  - Close input and output files.
- If you use task1.c, you just have to write the function that converts the record and saves it to the output file.

Converting the Record to the Other Endian Format.

- You need to reorder the bytes of each integer in the record.
- Pseudocode for reordering the bytes of the integer:
  - Convert the integer into an array of chars.
    - Use provided function **integer_to_characters.**
  - Reverse the order of the chars in that array.
  - Convert the array of chars back to an integer.
    - Use provided function **characters_to_integer.**
- Then, you need to write the converted record on the output file.
  - Use provided function **save_record.**

Task 1 Sample Output (1)

- Run on an Intel machine (little endian):

./a.out 0 test1_little.bin test2_big.bin

read: Record: age = 56, name = john smith, department = 6

read: Record: age = 46, name = mary jones, department = 12

read: Record: age = 36, name = tim davis, department = 5

read: Record: age = 26, name = pam clark, department = 10

Task 1 Sample Output (2)

- Run on an Intel machine (little endian):

./a.out 0 test2_big.bin out2_little.bin

read: Record: age = 939524096, name = john smith, department = 100663296

read: Record: age = 771751936, name = mary jones, department = 201326592

read: Record: age = 603979776, name = tim davis, department = 83886080

read: Record: age = 436207616, name = pam clark, department = 167772160

- Since the machine is little endian and the input data is big endian, the printout is nonsense.

The **diff** Command

- Suppose that you have run this command:

./a.out 0 test1_little.bin test2_big.bin

- How can you make sure that your output (test2_big.bin) is identical to test1_big.bin?
- Answer: use the **diff** command on omega.

diff test1_big.bin test2_big.bin

Task 2

- Goal: do parity-bit encoding/decoding of a file.
- Program flow:
  - Open input and output files.
  - While there is input data to be processed:
    - Read the next word W1 from the input file.
    - If (number == 0) convert W1 from original word to codeword W2.
    - If (number == 1):
      - convert W1 from codeword to original word W2.
      - print out a message if an error was detected.
    - Save W2 to the output file.
  - Close input and output files.

Task 2

- Goal: do parity-bit encoding/decoding of a file.
- Program flow:
  - Open input and output files.
  - While there is input data to be processed:
    - Read the next word W1 from the input file.
    - If (number == 0) convert W1 from original word to codeword W2.
    - If (number == 1):
      - convert W1 from codeword to original word W2.
      - print out a message if an error was detected.
    - Save W2 to the output file.
  - Close input and output files.
- If you use task2.c, you just have to write the functions that convert between original words and codewords.

Task 2 Files

- Task 2 works with bit patterns.
- In principle, the input and output files could be binary.
- Problem: difficult to view and edit (for debugging).
- Solution: use text files.
  - Bit 0 is represented as character '0'.
  - Bit 1 is represented as character '1'.

Task 2 Unencoded File (in1.txt)

1010100110100011001010100000110101111000011101110110
0111110000111100101101111111011110100000110100111100 1
1010000011000011101110010000011000011101110110100111
0110111000011101100010000011101001101000110000111101
0001000001101100110100111101101100101111001101000001
1010011101110010000010000011110101111001111101001110
0101100001110110011010011100010101110

- This binary pattern contains the 7-bit ASCII codes for: "The kangaroo is an animal that lives in Australia."

Task 2 Encoded File (coded1.txt)

1010100111010001110010100100000111010111110000111101
1101110011111100001111100100110111101101111001000001
1101001011100111010000011100001111011101010000011100
0011110111011101001011011011110000111101100001000001
1110100011010001110000111110100001000001110110001101
0010111011011100101011100111010000011101001011011101
0100000110000010111010111110011111101000111001001100
001111011000110100101100001101011100

- This binary pattern is the parity-bit encoding for: "The kangaroo is an animal that lives in Australia."

Task 2 - Sample Output (1)

- Encoding:

./a.out 0 in1.txt out1.txt

Start of translation:

The kangaroo is an animal that lives in Australia.

End of translation

Task 2 - Sample Output (2)

- Decoding (no errors found):

./a.out 1 parity1.txt out2.txt

Start of translation:

The kangaroo is an animal that lives in Australia.

End of translation

Task 2 - Sample Output (3)

- Decoding (errors found):

1 parity2.txt out2.txt
error detected at word 0
error detected at word 8
error detected at word 16
error detected at word 24
error detected at word 32
error detected at word 48

Start of translation:
he kangAroo is qn animad that lmves in Australi`.
End of translation

Practice Question 1

- Goal: do encoding/decoding of a file using an error correction code.
- It is specified as a text file, that the program reads.
- Example: code1.txt:
  - 3 is the number of bits in each original word.
  - 6 is the number of bits in each codeword.
  - 000 gets mapped to 000000.
  - 001 gets mapped to 001011.
  - and so on...

```
3 6
000 000000
001 001011
010 010101
011 011110
100 100110
101 101101
110 110011
111 111000
```

Practice Question 1

- Program flow:
  - Read code.
  - Open input and output files.
  - While there is input data to be processed:
    - Read the next word W1 from the input file.
    - If (number == 0) convert W1 from original word to codeword W2.
    - If (number == 1):
      - convert W1 from codeword to original word W2.
      - print out a message if an error was corrected or detected.
    - Save W2 to the output file.
  - Close input and output files.
- In general_codes.c, you just have to write the functions that convert between original words and codewords.

Practice Question 1: Code Struct

- This is the datatype that we use to store a code.

```
struct code_struct
{
  int m; // number of bits in original word
  int n; // number of bits in codeword columns.
  char ** original;   // original words
  char ** codebook;     // legal codewords
};
```

Practice Question 1: Encoding Logic

- Let W1 be the original word.
- Find the index K of W1 among the original words in the code book.
- Return the codeword stored at index K among the codewords.

Practice Question 1: Decoding Logic

- Let W1 be the codeword.
- Find the index K of the **legal codeword L most similar to W1**, among all legal codewords.
  - If L == W1, no errors.
  - If L != W1:
    - If unique L, error detected and corrected.
    - If multiple legal codewords were as similar to W1 as L was, error detected but not corrected.
- Return the original word stored at index K among the original words in the code book.