EE231002 Introduction to Programming

Lab13. Text Decoding

Due: Dec. 21, 2019

Traditional English characters are encoded using 7-bit ASCII format. Thus, some existing network communication protocols assume 7-bit per byte streams. Other bytes longer than 7 bits are used for network control. For non-ASCII characters, such Chinese, these protocols have troubles handling their transmission. This limitation was recognized in the early days, and some encoding schemes were developed. In essence, these encoding schemes convert non-ASCII characters into ASCII streams making them transmittable using the standard protocols. Then, decoders are invoked to translate those streams back to the original format. In this way, non-ASCII streams can be easily transmitted using the existing framework.

In this assignment we will implement a decoder for a special encoding scheme. A short encoded file is shown below:

```
---begin---
UHJvZ3JhbW1pbmcK
---end---
```

The first line is a header. The encoded file always starts with the `---begin---` marker. The encoded text starts from the next line. Depends on the original text, the number of encoded line varies and can be as large as needed. The tail of the encoded file is a single line, `---end---`.

The rest of encoded string is formed by taking 3 original characters, which have total of 24 bits, split them into 4 groups, each group then has 6 bits. Two 0's are added to the left to form 8-bit `char` again. Each byte is then converted to an ASCII character by the conversion table shown in Table 1.

An example of encoding the work 'dog' is shown below.

Original text:	d	o	g	
ASCII code:	0110,0100	0110,1111	0110,0111	
Splitting:	0001,1001	0000,0110	0011,1101	0010,0111
Encoded text:	Z	G	9	n

The encoding process continues by taking the next 3 characters as input and producing 4 characters output. This process is repeated until all the input characters are processed. In the case that we have less than 3 characters left to be encoded. `NULL` characters (`'\0'`) are added to form 3 characters such that the preceding encoding process can be carried out. And padding character `=` is printed at the end to make the number of encoded characters is divisible by 4. Two examples of adding padding characters are shown below.

Byte	ASCII	Byte	ASCII	Byte	ASCII	Byte	ASCII
0x00	A	0x10	Q	0x20	g	0x30	w
0x01	B	0x11	R	0x21	h	0x31	x
0x02	C	0x12	S	0x22	i	0x32	y
0x03	D	0x13	T	0x23	j	0x33	z
0x04	E	0x14	U	0x24	k	0x34	0
0x05	F	0x15	V	0x25	l	0x35	1
0x06	G	0x16	W	0x26	m	0x36	2
0x07	H	0x17	X	0x27	n	0x37	3
0x08	I	0x18	Y	0x28	o	0x38	4
0x09	J	0x19	Z	0x29	p	0x39	5
0x0A	K	0x1A	a	0x2A	q	0x3A	6
0x0B	L	0x1B	b	0x2B	r	0x3B	7
0x0C	M	0x1C	c	0x2C	s	0x3C	8
0x0D	N	0x1D	d	0x2D	t	0x3D	9
0x0E	O	0x1E	e	0x2E	u	0x3E	+
0x0F	P	0x1F	f	0x2F	v	0x3F	/

Table 1. Conversion table.

In the case that only two characters are left, a NULL character `\0` is added to the end. Then grouping of 6 bits and splitting into 4 bytes are performed as before. But the last byte of the encoded is changed to the padding character `=` to signify only two bytes are encoded.

Original text:	b	e		
ASCII code:	0110,0010	0110,0101	0000,0000	
Splitting:	0001,1000	0010,0110	0001,0100	0000,0000
Encoded text:	Y	m	U	=

In the case that only one character is left, then two NULL characters are added. The grouping and splitting process are performed. But the last two bytes are changed to padding character.

Original text:	a			
ASCII code:	0110,0001	0000,0000	0000,0000	
Splitting:	0001,1000	0001,0000	0000,0000	0000,0000
Encoded text:	Y	Q	=	=

It should be also noted that all control characters, such as `\t`, `\n`, `\0`, etc., are also encoded.

Your assignment is to write a `C` program to decode an encoded file. The program should read the header line and ignore it. Then convert the encoded string back to the original form by reversing the process as shown above. Of course, it needs to detect the tail section and stop the

conversion process. Two encoded files are provided for you to test your program: `etext.cd` and `ctext.cd`. You can use the standard Unix command line to redirect file as the standard input to the program as the following:

```
$ ./a.out < etext.cd
```

Of course, you can also save the decoded strings to a file using the following command as an example. By doing so, you can detect if any non-printable characters have been accidentally printed out.

```
$ ./a.out < etext.cd > etext
```

Notes.

1. Create a directory **lab13** and use it as the working directory.
2. Name your program source file as **lab13.c**.
3. The first few lines of your program should be comments as the following.

   ```
   // EE231002 Lab13. Text Decoding
   // ID, Name
   // Date:
   ```

4. After you finish verifying your program, you can submit your source code by

   ```
   $ ~ee2310/bin/submit lab13 lab13.c
   ```

   If you see a "submitted successfully" message, then you are done. In case you want to check which file and at what time you submitted your labs, you can type in the following command:

   ```
   $ ~ee2310/bin/subrec lab13
   ```

   It will show the submission records of lab13.

5. You should try to write the program as efficient as possible. The format of your program should be compact and easy to understand. These are part of the grading criteria.