Programming Assignment

This is an assignment that should give you a chance to apply some of the techniques we've been discussing (bit manipulation, binary/text file I/O, etc.) and a to give you some practice in following an algorithm. The goal is to create a program that encodes a file using a *uuencode* algorithm and decodes it with a *uudecode* algorithm. A uuencode algorithm takes input, either binary or text, and converts it into plain text output. This is essentially how email programs can send binary files (e.g. zip files) across a text only network protocol (e.g. SMTP or email). The crux of the algorithm is that it converts 3 8-bit characters (24-bits) into 4 6-bit characters (also 24-bits.) The value 32 is added to each 6-bit character which guarantees that all new characters are plain ASCII printable characters. For example, given a file (named **test.txt**) containing the single 3-letter word **cat**, this is what the uuencoded text (placed in a file called **test.uue)** looks like:

```
begin 644 test.txt
#8V%T
`
end
```

NOTE:
on Linux it's difficult to create a file containing "cat" only, most editors add "0a" (new line) automatically and you cannot delete it. On Windows "notepad" will do that easily. Before comparing outputs make sure you know exactly what is in the file – just opening it doesn't help – check file size, if it's 3 bytes, then there is no "newline", if it's 4, then there is a "newline". Uuencoded text above is for 3-byte file. Even better you may use "od" (see below).

The actual encoding of the word **cat** is in the second line. Dividing the 3 8-bit characters into 4 6-bit characters and adding 32 can be shown graphically:

	c								a								t							
	0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1	1	0	1	0	0
+32	8						V						%						T					

011000=24, 110110=54, 000101=5, 110100=52
and
ASCII(8)=56, ASCII(V)=86, ASCII(%)=37, ASCII(T)=84.

A longer passage (preamble.txt):

```
When, in the course of human events, it becomes necessary for a people to advance from
that subordination in which they have hitherto remained, and to assume among the
powers of the earth, the equal and independent station to which the laws of nature and
of nature's god entitle them, a decent respect to the opinions of mankind requires
that they should declare the causes which impel them to the change.
```

The passage above will be encoded as such **(preamble.uue)**:

```
begin 644 mypreamble.txt
M5VAE;BP@:6X@=&AE(&-O=7)S92!O9B!H=6UA;B!E=F5N=',L(&ET(&)E8V]M
M97,@;F5C97-S87)Y(&9O<B!A('!E;W!L92!T;R!A9'9A;F-E(&9R;VT@=&AA
M="`-"G-U8F]R9&EN871I;VX@:6X@=VAI8V@@=&AE>2!H879E(&AI=&AE<G1O
M(')E;6%I;F5D+"!A;F0@=&\@87-S=6UE(&%M;VYG('1H92!P;W=E<G,@;V8@
M=&AE(`T*96%R=&@L('1H92!E<75A;"!A;F0@:6YD97!E;F1E;G0@<W1A=&EO
M;B!T;R!W:&EC:"!T:&4@;&%W<R!O9B!N871U<F4@86YD(&]F(&YA='5R92=S
M(&=O9"`-"F5N=&ET;&4@=&AE;2P@82!D96-E;G0@<F5S<&5C="!T;R!T:&4@
M;W!I;FEO;G,@;V8@;6%N:VEN9"!R97%U:7)E<R!T:&%T('1H97D@<VAO=6QD
M(&1E8VQA<F4@=&AE(`T*8V%U<V5S('=H:6-H(&EM<&5L('1H96T@=&\@=&AE
((&-H86YG92X`
`
end
```

Implementation details:
The converted file consists of 4 components:
1. First, there is a line that consists of the word **begin**, the mode, and the name of the file given to the decoded file. The name of the file is needed so when the encoded file is decoded, it will be given the appropriate name, which may differ from the original name. The mode (permissions) will always be 644.
2. Then, the actual converted characters follow, containing at most 45 converted characters per line.
3. After all of the characters have been encoded, a line with a single back-quote is emitted.
4. Finally, the word **end** on a line by itself terminates the encoded text.

Note that the encoding is in a text format, so every line must end with a newline, **including the last line**. The converted characters make up the bulk of the encoded text and are subject to additional constraints.

Encoding details:

- The first character on the line is the number of converted characters that this line represents . Since the count is also encoded, you have to subtract 32 (the ASCII value for a space) from the ASCII value of the character. Since all lines but the last line will be full, meaning they contain 60 characters, they will all start with the letter **M**. If the **last** line is not full, it will start with a letter other than **M**. (**M** is ASCII 77 and 77 – 32 = 45)
- Characters that are converted to spaces (ASCII 32) are changed to a backquote (ASCII 96).
- **If the input does not consist of a number of characters divisible by 3**, you have to pad the output with extra characters. The extra characters are the result of extending the input with 0 bits to account for the shortage of bits. The length of the encoded lines will *always* be a multiple of 4. (Not including the first character in the line, which is the count.) You have to account for this during encoding, otherwise you'll see extra characters in the output file that may look something like "^@" (NULL)
- Encoding: read in files as binary and write (text) to stdout; Decoding: read in files as text and write files as binary.

Testing

Take a look at "driver-args.cpp", use it as follows:
1) to **encode** to stadard output (terminal)
```
./gcc-args.exe 1 text.txt remote_name.txt
```
you should see
```
begin 644  remote_name.txt
.....
```
2) to **encode** to a file, so that you can use this later with decoder
```
./gcc-args.exe 1 text.txt remote_name.txt > text.uue
```
now "text.uue" contains the output from 1)
3) to **decode**
```
./gcc-args.exe 2 text.uue
```
this line should create a new file "remote_name.txt" in the current directory.

Testing

– The Cygwin utilities come with programs called **uuencode.exe** and **uudecode.exe** (shareutils package) that you can use to help test your programs. If you don't have that program, you can download it for free and add it to your other Cygwin utils.
Example of a file named **cats.txt** containing the 4-letter word "**cats"**:
type "uuencode cats.txt cats2.txt"
output:

```
begin 644 cats2.txt
$8V%T<P``
`
end
```

type "uuencode cats.txt cats2.txt > cats.uue" to save encoded text into file, rather than displaying it in the terminal.

The input above (4 bytes) is not a multiple of 3, it's short by two bytes (16 bits), so you'll add zeros appropriately.

Testing

– CygWin (again) has "od" -- octal dump (coreutils package), which may help you to examine the produced file. Usage
```
od -x filename
```
output
```
0000000 6923 6e66 6564 2066 4946 454c 414e 454d
0000020 485f 0a0d
0000024
```

where the left column contains addresses, all other columns are ASCII codes in hexadecimal format corresponding to characters in the file, note that “6923” represents 2 characters – 23 and 69 (in this order) which are '#' and 'i'. Also note that the file was in Windows text format, since we can clearly see “0a0d ” in the end, which is Windows CRLF (carriage return “0d ”, line feed “0a”) line terminator.

Testing

CygWin (again) has “diff”. For this assignment you may want to

- Test encoding
  - take a file (binary or text) “filename.ext”
  - uuencode “filename.ext” using CygWin or provided Windows executable “UUCoder.exe” into “filename.uue” with a different remote name, say “filename2.ext”
  - uuencode “filename.ext” using your code into “filename-my.uue” with a remote name “filename2.ext” (same as above)
  - `diff filename.uue filename-my.uue –strip-trailing-cr`

    “--strip-trailing-cr” flag makes “diff” to NOT pay attention to the newline style.
    if there is a single line of output from “diff” - there is(are) differences, and you should figure out were exactly they are. Use
    diff filename.uue filename-my.uue –strip-trailing-cr -y
    “-y” flag makes “diff” to display the output in two columns, marking lines that differ with a bar '|'.
    Hint: 2 columns will not fit into a terminal, so you should either resize the terminal (click icon “C:\”, then Properties->Layout->Window Size), or redirect output into a file (add “ > diiference.txt” in the end of diff line, and then open file “difference.txt” in any editor)

- To test decoding use this
  - take a file (binary or text) “filename.ext”
  - uuencode (assuming this is already tested, or use CygWin for encoding) it into “filename.uue” with a different remote name, say “filename2.ext”
  - decode using your code just created “filename.uue” (this step should create a new file “filename2.ext”)
  - compare original file “filename.ext” and “filename2.ext”

    ```
    diff filename.uue filename-my.uue
    ```

    (**DO NOT USE** “--strip-trailing-cr” - the 2 files should match exactly)
    if there is a single line of output from “diff” - there is(are) differences, and you should figure out were exactly they are. Here is how I would do that (especially for a binary file, text files may be compared without “od”):

    ```
    od -x  filename.ext >  filename.ext.dump
    od -x  filename2.ext >  filename2.ext.dump
    diff -y --strip-trailing-cr filename.ext.dump filename2.ext.dump
    ```

Function Signatures:

The prototype for the functions are as follows:

```
int uuencode(const char *InputFilename, const char *RemoteFilename);
int uudecode(const char *InputFilename);
```

Parameter Description
InputFilename Name of the file to encode/decode.
RemoteFilename Name of the decoded file . (This is simply placed on the **begin** line of the encoded file.)
Returns 0 if the conversion was successful, -1 if unsuccessful.

To get credit on this assignment, you must NOT use any subscripting in your program. All array access is to be done using pointer arithmetic. The only place you should use the square brackets (array operator) is in a declaration.
Examples usage:

```
int result = uuencode("E:\\Data\\Courses\\program3.exe", "myprog3.exe");
```

would produce output to **stdout** (myprog3.exe is a name used by decoder) like this: (the encoded lines would actually be included)

```
begin 644 myprog3.exe
**** encoded lines here ****
`
end
```

Given a file (named program3.uue) containing the above text, this function call:

```
int result = uudecode("program3.uue");
```

will result in a file named myprog3.exe that contains the decoded data from program3.uue. Note that **uuencode** sends its output (text) to the screen (stdout) and **uudecode** sends its output (text or binary) to a file.

What to submit
You must submit a zip-file containing implementation/header files (**uucode.cpp** and **uucode.h**) through the online submission page.

uucode.h	The header file
uucode.cpp	The implementation file. All implementation for the functions goes here. You must document the file and functions using **Doxygen** tags as specified.