This programming project is due on **Thursday, July 14, 2022** at 10:00 p.m. The best approach is to plan to have the solution submitted BEFORE the due date. Then, if you experience any last-minute difficulty, you will still meet the deadline.

Be sure that you read and understand this entire document before you begin writing your code. Pay close attention to the **Project Deliverables** and **Grading Criteria** sections of this document. If you have **questions**, please ask during class or send me an e-mail at my BHCC e-mail address (pmorgan@bhcc.edu).

Overview:

Your task is to write a program that performs two data processing tasks:

1. Read a text document (a file containing ASCII text) and perform the following:
   - Convert that text into a sequence of packed **unsigned integers.** (Each **unsigned int** contains the bits from 4 characters.)
   - Write the **unsigned int** values to another text file, as hexadecimal numbers, with one number on each line of the output file.

2. Read the **unsigned int** (hexadecimal) values and convert those numbers into a text file that matches the original text document.

Important Observation:

All of the concepts necessary to produce a solution for this assignment have been covered in class. If you need help to understand this assignment, ask the instructor.

Implementation Details:

The program must be a "command-loop" program (as discussed in class). The commands supported by this command-loop program must be:

p	Pack a text document into unsigned integers.
u	Unpack unsigned integers to text
h	Output "help" text
q	Exit the program.

(continued on the next page)

The "p" command:

The "p" command (pack) must perform the following steps:

- Issue prompts to the user, asking them to enter the name of the **input file**, and the **output file**.
- Read one complete line of text from the input text file, saving the text in a **string** object.
- Append a new-line character ("**\n**") to the end of the input text.
- Read characters from the string object, and use **bitwise operations** to pack four characters from the **string** object into an **unsigned int** variable, so that the bits of each character are positioned (in the **unsigned int** variable) as shown in the diagram below:

bits 24-31	bits 16-23	bits 8-15	bits 0-7
character 1	character 2	character 3	character 4
character 5	character 6	character 7	character 8
character 9	character 10	character 11	character 12
…	…	…	…

- Write the **unsigned int** variable to the output file, using an object of the **ofstream** class.
- Repeat the packing and output operations for each group of 4 characters from the original document.
- If there are any characters "left over" from the last group of four characters, then the final (partially full) **unsigned int** value must be written to the output file.

The "u" command:

The "u" command (unpack) must perform the following steps:

- Issue prompts to the user, asking them to enter the name of the **input file**, and the **output file**.
- Read one **unsigned int** value from the input file, extract four ASCII characters from the **unsigned int** value, and output the four ASCII characters to the output file.
- Repeat the unpacking operation for each **unsigned int** number: extract 4 characters and copy them to the output file.

This process of unpacking characters from the integer values, and then writing those characters to a text file accomplishes the **reverse** of what the "**p**" command did.

(Refer also to the **Sample Output** section of this document.)

Format of the Source Code:

The format of the source file must look something like the following example:

Format of the source code

```
//              CSC237 Project2:  Text Packing / Unpacking Operations
//   Student:    yourName
//   Due Date:   projectDueDate
//   Description:
//      This program reads a text document, "packs" the ASCII characters
//      from that document into unsigned int variables, and outputs those variables
//      to another text file, formatted as hexadecimal numbers.
//      This program also reverses the process, converting the unsigned int numbers
//      back into a copy of the original text document.
#include <iostream>
using namespace std;

int main()
{
    . . .
}
```

Sample Output:

Test your program with different input values. The samples that follow show correct output for several test cases. (In these examples, the text that the user types is shown in **BOLD** font. The actual input / output will all be displayed in the same font.)

Sample Input / Output: Example 1

```
Command: h
Supported commands:
     p    Build Packed Data File.
     u    Create unpacked (text) data from packed data.
     h    Print this help text.
     q    Quit (exit) the program.
     v    Set or Clear verboseMode (debug aid).

Command: p
Enter the input filename: alphabet.txt
Enter the output filename: alphabet_PACKED.txt
Input text (length=26): ABCDEFGHIJKLMNOPQRSTUVWXYZ

Command: u
Enter the input filename: alphabet_PACKED.txt
Enter the output filename: alphabet_UNPACKED.txt

Command: q
Are you sure that you want to exit the program? y
Exit the program.
```

Input File: alphabet.txt

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Output File: alphabet_PACKED.txt

```
41424344
45464748
494A4B4C
4D4E4F50
51525354
55565758
595A0A00
```

Output File: alphabet_UNPACKED.txt

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

In addition to testing your program with a small text file, you must also test your program with a larger (multi-line) text file:

Sample Input / Output: Example 2

```
Command: p
Enter the input filename: preamble.txt
Enter the output filename: preamble_PACKED.txt
Input text (length=75): We the People of the United States, in Order to form
a more perfect Union,
Input text (length=80): establish Justice, insure domestic Tranquility,
provide for the common defense,
Input text (length=78): promote the general Welfare, and secure the
Blessings of Liberty to ourselves
Input text (length=65): and our Posterity, do ordain and establish this
Constitution for
Input text (length=29): the United States of America.

Command: u
Enter the input filename: preamble_PACKED.txt
Enter the output filename: preamble_UNPACKED.txt

Command:
```

Input File: `Preamble.txt`

```
We the People of the United States, in Order to form a more perfect Union,
establish Justice, insure domestic Tranquility, provide for the common defense,
promote the general Welfare, and secure the Blessings of Liberty to ourselves
and our Posterity, do ordain and establish this Constitution for
the United States of America.
```

Output File: `Preamble_PACKED.txt`

```
57652074
68652050
656F706C
65206F66
20746865
20556E69
74656420
53746174
65732C20
696E204F
72646572
20746F20
666F726D
2061206D
6F726520
70657266
65637420
556E696F
6E2C200A
65737461
626C6973
68204A75
73746963
652C2069
6E737572
6520646F
6D657374
69632054
72616E71
75696C69
74792C20
70726F76
69646520
666F7220
74686520
636F6D6D
6F6E2064
6566656E
73652C20
0A000000
70726F6D
6F746520
74686520
67656E65
72616C20
```

Output File: `Preamble_PACKED.txt`

```
57656C66
6172652C
20616E64
20736563
75726520
74686520
426C6573
73696E67
73206F66
204C6962
65727479
20746F20
6F757273
656C7665
73200A00
616E6420
6F757220
506F7374
65726974
792C2064
6F206F72
6461696E
20616E64
20657374
61626C69
73682074
68697320
436F6E73
74697475
74696F6E
20666F72
200A0000
74686520
556E6974
65642053
74617465
73206F66
20416D65
72696361
2E0A0000
```

Output File: `Preamble_UNPACKED.txt`

```
We the People of the United States, in Order to form a more perfect Union,
establish Justice, insure domestic Tranquility, provide for the common defense,
promote the general Welfare, and secure the Blessings of Liberty to ourselves
and our Posterity, do ordain and establish this Constitution for
the United States of America.
```

Project Deliverables:

The project source file must be submitted to *Moodle*, using the *Moodle* assignment for this project.

Submit ***only*** your **source code** (**`*.cpp`**) file. I will need to compile your code on my home computer in order to grade it.

- Do ***not*** submit the entire project from your preferred IDE.
- Do ***not*** include any project folders, or any binary files.
- Do ***not*** place the source code file in a "ZIP" file, a "RAR" file, or any other file collection.

Grading Criteria:

The project will be graded according to the following grading criteria:

Feature	Portion of grade
1. The program functions correctly.	50%
2. The program **must** be organized as a "command-loop" program. (We discussed the "command-loop" design in class.)	10%

3. In the **main** function of the program, there is a loop that contains code to support the following input commands:
`p  Build Packed Data File.`
`u  Create unpacked (text) data from packed data.`
`h  Print help text.`
`q  Quit (exit) the program.`
4. The "command loop" in the **main** function must continue until the user enters a '**q**' command.
	10%

5. The **main** function must call other functions to implement the various commands.	5%
6. The program must NOT contain any global variables *except* the optional **verbose_mode** variable described in class. (Global constants are OK.)	3%
7. The program uses good, descriptive variable names.	5%
8. The program source code is clearly organized and **commented** so as to make it easy to read and understand: • The source file must have a heading comment, similar to the example shown in the project assignment document. • The comments within the code must describe each short section of the program. (Do **not** place a separate comment on every line of code.)	15%
9. The source code (.cpp) file must have a descriptive name such as "`project2.cpp`" or "`textPacker.cpp`". Do **NOT** use the default file name (for example "`Source.cpp`") provided by the IDE.	2%