This programming project is due on **Wednesday, October 16** at 11:00 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.

Reminder: Do your own work on this project. Do not obtain any code from another student, or from the Internet. Do not show your code to anyone except the instructor, or an official BHCC Tutor. Refer also to the last page of the course *Syllabus*, for details about the BHCC policy regarding academic dishonesty. You may NOT use any code-generating software, such as **ChatGPT** or similar software. Also, you may NOT assist any other student to cheat in any way.

Practical Tip: When you work on **Lab06a**, that is good preparation for working on this project. It is suggested that you complete **Lab06a** before beginning this project.

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**, ask the instructor during class or contact the instructor by BHCC e-mail: pmorgan@bhcc.edu.

Overview:

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

1. **Pack** the contents of a text document into integer values:
   - Read a text document (a file containing ASCII text) **one line at a time**.
   - Append a newline ("**\n**") character to each line of text after reading it.
   - Pack 4 characters at a time into an **unsigned int** variable.
   - Save each **unsigned int** value to a text file.
2. **Unpack** the contents of the file produced by the **Pack** command into a new text file:
   - Read one **unsigned integer** value at a time and convert each unsigned integer to 4 characters.
   - Append those 4 characters to the (output) text file.

After the user has executed the **Pack** and **Unpack** commands, the final output file has the same contents as 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.

The "p" command:

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

1. Issue prompts to the user, asking them to input the name of the **input file**, and the **output file**. Open an **ifstream** object and an **ofstream** object.
2. For **each** line of text in the input file:

    - Read one complete line of text from the input text document, saving the text in a **string** object.
    - Append a new-line character ("**\n**") to the end of the **string**.
    - Process the **string** contents **1 character at a time**, keeping track of the **position** (in the string) of each character:
        a. Use the **position** value to assist in deciding how to merge the individual characters into the correct position of an **`unsigned int`** variable:

bits 24-31	bits 16-23	bits 8-15	bits 0-7
character from position 0	character from postion 1	character from position 2	character from position 3

        b. After one complete group of four characters have been merged into the **unsigned int** variable, output that **`unsigned int`** to the output file, on a line by itself.
        c. Repeat this process until the end of the **string** object has been reached.
    - If there are any characters "left over" from the last group of four characters, then output the final (partially filled) **unsigned int** value to the output file.
    - Output a **blank line** to the output file. (This helps make the final output file easier for a person to read.)

3. After **all** lines from the input file have been processed, close both files.

The "u" command:

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

1. Issue prompts to the user, asking them to input the name of the **input file**, and the **output file**. Open an **ifstream** object and an **ofstream** object.
2. Process the input data **one unsigned int at a time**:

    - Read one **unsigned int** value from the input file, extract four ASCII characters from the **unsigned int** value.
    - Output each ASCII character to the output file (unless its value is **hex `00`**).

3. After all of the **unsigned int** values have been processed, close the **ifstream** and **ofstream** objects.

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.)

SPECIAL NOTE for users of the GraderThan environment:

Because **Windows** and **Linux** have slightly different text file formats, you may notice some small differences with your results. We will discuss this in class.

Format of the Source Code:

The beginning of the source code file **must** look something like the following example:

Format of the source code

```
//              CSC237 Project1:  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 as integers.
//       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()
{
    . . .
}
```

However, your program must NOT have all of the code in the "**main**" function.

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.

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
```

Sample Input / Output: Example 1

```
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

```
1094861636
1162233672
1229605708
1296977744
1364349780
1431721816
1499073024
```

Output File: alphabet_UNPACKED.txt

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Sample Input / Output: Example 2

```
Command: p
Enter the input filename: fruit.txt
Enter the output filename: fruit_PACKED.txt
Input text (length=13): apple apricot
Input text (length=6): banana
Input text (length=17): cantaloupe cherry
Input text (length=5): grape
Input text (length=10): peach plum

Command: u
Enter the input filename: fruit_PACKED.txt
Enter the output filename: fruit_UNPACKED.txt
```

Input File: fruit.txt

```
apple apricot
banana
cantaloupe cherry
grape
peach plum
```

Output File: `fruit_PACKED.txt`

```
1634758764
1696620912
1919509359
1946812416

1650552417
1851853312

1667329652
1634496373
1885675619
1751478898
2030698496

1735549296
1695154176

1885692259
1746956396
1970080256
```

Output File: `fruit_UNPACKED.txt`

```
apple apricot
banana
cantaloupe cherry
grape
peach plum
```

Sample Input / Output: Example 3

```
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`

```
1466245236
1751457872
1701802092
1696624486
544499813
542469737
1952801824
1400136052
1702046752
1768824911
1919182194
544501536
1718579821
543236205
1869767968
1885696614
1701016608
1433299311
1848385546

1702065249
1651272051
1746946677
1937008995
1697390697
1853060466
1696621679
1835365236
1768104020
1918987889
1969843305
1954098208
1886547830
1768187168
1718579744
1952998688
1668246893
1869488228
1701209454
1936010272
167772160

1886547821
```

Output File: `preamble_PACKED.txt`

```
1869899040
1952998688
1734700645
1918987296
1466264678
1634886956
543256164
544433507
1970431264
1952998688
1114400115
1936289383
1931505510
541878626
1701999737
544501536
1869967987
1701607013
1931479552

1634624544
1869967904
1349481332
1701996916
2032935012
1864396658
1684105582
543256164
543519604
1633840233
1936203892
1751741216
1131376243
1953068149
1953066862
543584114
537526272

1952998688
1433299316
1701060691
1952543845
1931505510
541158757
1919509345
772407296
```

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 Activity:

CSC237_Project1

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 *Visual Studio* project.
- Do ***not*** include the *Visual Studio* 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 <u>other functions</u> to implement the various “Command Loop” commands.	10%
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.)	10%

Feature	Portion of grade
9. The source code (.cpp) file must have a descriptive name such as "`project1.cpp`" or "`textPacker.cpp`". Do **NOT** use the default file name (for example "`Source.cpp`") provided by the IDE.	2%