File I/O

Exercise 1: Morse Code Converter

For this exercise, you will read in an **input.txt** file, which has English text, and you will write out a **output.txt** file, which will contain the same message from the input, but in morse code.

You will read each word from the file, and create a “converted” string by reading through each letter of the word and writing the morse code equivalent to the converted string. Then, the conversion will be written back out. All words will be capital letters. Separate each letter in morse code by a space, and each word in morse code by a slash /.

Morse Code

A	B	C	D	E	F
.-	-...	-.-.	-..	.	..-.
G	H	I	J	K	L
--.	....	..	.---	-.-	.-..
M	N	O	P	Q	R
--	-.	---	.--.	--.-	.-.
S	T	U	V	W	X
...	-	..-	...-	.--	-..-
Y	Z	0	1	2	3
-.--	--..	-----	.----	..---	...--
4	5	6	7	8	9
....-	.....	-....	--...	---..	----.

Message File

Copy this text into a text editor (like notepad) and save it into the same path as your source files.

input.txt

```
TOP GAMES OF 1998 ON METACRITIC
1 THE LEGEND OF ZELDA OCARINA OF TIME
2 TEKKEN 3
3 HALF LIFE
4 GRAN TURISMO
5 METAL GEAR SOLID
6 GRIM FANDANGO
7 BANJO KAZOOIE
8 THIEF THE DARK PROJECT
9 BALDURS GATE
```

The output should look like this:

output.txt

```
- --- .--. / --. .- -- . ... / --- ..-. / .---- ----. ----. ---.. / --- -. / -- . - .- -.-. .-. .. - .. -.-. / .---- / - .... . / .-.. . --. . -. -.. / --- ..-. / --.. . .-.. -.. .- / --- -.-. .- .-. .. -. .- / --- ..-. / - .. -- . / ..--- / - . -.- -.- .
-. / ...-- / ...-- / .... .- .-.. ..-. / .-.. .. ..-. . / ....- / --. .-. .- -. / - ..- .-. .. ... -- --- / ..... / -- . - .- .-.. / --. . . .- .-. / ... --- .-.. .. -.. / -.... / --. .-. .. -- / ..-. .- -. -.. .- -. --. --- / --... / -... .- -. .--- --- / -.- .- --.. --- --- .. . / ---.. / - .... .. . ..-. / - .... . / -.. .- .-. -.- / .--. .-. --- .--- . -.-. - / ----. / -... .- .-.. -.. ..- .-. ... / --. .- - .
```

Step-by-step

Saving the input file

1. Open up Notepad, or another text editor
2. Copy-paste the “TOP GAMES OF 1998 ON METACRITIC” content into Notepad.
3. Save this .txt file in the path of your C++ project, the same folder where your .cpp file goes.

Iteration 1: Reading the file

Utilize the **ifstream** object to open the text file for reading.

Write the program so that it only reads in the **input.txt** file, one word at a time, and displays each word to the screen with the **cout** statement. You can read in one word at a time with the following:

```
ifstream input( "input.txt" );

string word;
while ( input >> word )
{
    // logic for each word
}

input.close();
```

Iteration 2: Breaking down the words

Add on to the program that reads each word. Now instead of using **cout** on each word, you will use **cout** to display each letter separately. Remember that you can treat a string as an array, and use a for loop to iterate through every letter:

```
for ( int i = 0; i < word.size(); i++ )
{
    cout << word[i] << endl;
}
```

Iteration 3: Building a new string

For each word that is read in, you will want to build a new string. The string will start empty, and every letter that is read in, it will convert into morse code. You might want to write a function to convert a single letter into a morse code string.

```
ifstream input( "input.txt" );

string word;
while ( input >> word )
{
    string converted = "";
    for ( int i = 0; i < word.size(); i++ )
    {
        converted += ConvertToMorseCode( word[i] );
    }
}

input.close();
```

Write the function that will convert a single letter (a character) into a morse code string. You can use switch statements or if statements if you'd like. If you're really clever, you will use arrays, and realize that:

```
int('A') = 65       int('Z') = 90
```

But it not necessary to do this.

Iteration 4: Output the result

After a word is converted to the morse code version, make sure to output the results to a file called **output.txt**. Instead of using cout, it will look more like:

```
ofstream output( "output.txt" );

// after conversion:
output << conversion;
```

File I/O and Structs

Exercise 2: Inventory and Receipts

Write a program that will load in a list of items that a store sells from an input file, **inventory.txt**.

Create a struct **Product**:

```
struct Product
{
    string itemCode;
    string section;
    float price;
    int quantityInStock;
};
```

Loading in the products

Create an array of Product items that symbolizes all the items available in this store.

First thing in your program, you will load in all the items from **inventory.txt**. There is a sample of this file below. Every line has 4 pieces of information: Item code, Section, Price, and Quantity.

Load in each item from the inventory list into the Product array. Don't use `getline`, just use the >> operator to get one word at a time from the text file.

Hints

- Since you can assume the file always is in the same format – four items per row – you can use >> multiple times to get all 4 pieces of information for the Product item.
- Make sure to skip the header of the text file – the first 4 pieces of information in the document should be skipped. (To skip, just use >> four times but store it in a temporary string variable.)

Shop

You will create a second Product array to store the items in the user's cart, so you will have something like:

```
Product inventory[50];
Product cart[50];
```

The sizes are set to an arbitrarily large number because we don't know how many items could be in the **inventory.txt** file, or how many items the user may want to buy.

Create a loop that will continue running until the user is done and ready to check out:

First, it will display all the SECTIONS of the store – MOVIES, GAMES, and SNACKS, as well as an option to Check Out. Have the user select where they want a product from, then you will display a sub-menu based on this.

In the sub-menu, you will display all inventory items that have this section. You don't know which items these will be, since **inventory.txt** can change, so *loop through the entire inventory list*, and only display the option if the `section` variable of the Product item is the same as the section the user is currently looking at.

After an item is selected, it is added to an array of items in the cart . After the user selects an item, decrease its QUANTITY in the inventory by 1. If the user tries to buy an item that has a quantity of 0, do not allow them to. The user can buy the same item multiple times if there is enough quantity.

Once the user is done selecting items to purchase (has selected the “Check Out” option), tally the total, calculate the price with tax, and then output the purchase information (each item name and price, quantity of each, price per item, total per item, total of all items, and total after tax) to a file called **receipt.txt**. See the next page for sample files.

Input

inventory.txt

```
ITEM_CODE           SECTION        PRICE          QUANTITY
The_Godfather       MOVIES         13.99          10
Seven_Samurai       MOVIES         25.99          2
The_Producers       MOVIES         12.99          5
Super_Mario_Galaxy  GAMES          19.99          10
Half_Life_2         GAMES          5.99           2
Resident_Evil_4     GAMES          12.75          3
Popcorn             SNACKS         3.99           21
Chocolate_Bar       SNACKS         1.19           15
Soda                SNACKS         1.59           20
```

Sample Output

receipt.txt

```
ITEM                    COUNT     PPI            TOTAL
Popcorn                 1         $3.99          $3.99
Soda                    5         $1.59          $7.95
Super_Mario_Galaxy      1         $19.99         $19.99
The_Godfather           1         $13.99         $13.99
The_Producers           1         $12.99         $12.99

Total Cost: $58.91
Tax: 8.85%
Price Due: $64.12
```

It is OK if your price ends up having more items to the right of the decimal place than this.