2020 Summer Computer Science I Section 2
Week 1 Program: Scrabble Scoring
Please Consult WebCourses for the due date/time

In the game of Scrabble, players get letter tiles and attempt to form words with them. Each tile has a score, based on the letter on the tile. Words are placed on the playing board, which is a two dimensional grid of squares, some of which have special scoring rules. Players place their tiles on contiguous grid squares either horizontally or vertically and their score is based on both the score of each tile as well as the grid squares with special rules on which the tiles were placed. There are two types of special rules for grid squares: one rule multiplies the score of the tile placed on it by 2 or 3, and the other rule multiplies the whole word score by 2 or 3. The latter rule is invoked after all instances of the former rule.

For this problem, we'll look at a fairly simplified version of Scrabble scoring. We'll assume that a player will form a word with their tiles only and that the scoring rules will be based on tile positions. For example, the third tile might multiply the tile score by 2 and the fifth tile may multiply the whole word score by 3. For the purposes of this problem, we'll assume that only one grid square will utilize a rule of the second type.

The tile scores by letter will be given in this constant array. Please copy and paste this into your program before main:

```
const int TILESCORES[] =
{1,3,3,2,1,4,2,4,1,8,5,1,3,1,1,3,10,1,1,1,1,4,4,8,4,10};
```

Namely, if an uppercase letter is stored in the variable let, then `TILESCORES[let-'A']` represents the score of a tile with the letter let on it. Recall that the Ascii values of uppercase letters are contiguous, so if you take an uppercase letter and subtract 'A' from it, you'll get an integer in between 0 and 25, inclusive. Namely, 'A' - 'A' = 0, 'B' - 'A' = 1, and so forth. The array above is arranged so that index 0 stores the score for 'A', index 1 stores the score for 'B', etc.

The problem you'll solve will be as follows. Given the following:

1. List of the letters on each of the tiles a player has
2. The positions of the special squares and the types of those squares
3. A full dictionary of words

you must write a program to determine the maximum score the player can achieve.

An Example Worked Out

Let the player have the following letter tiles: A, D, M, A, R, N, and G.

Let the dictionary of valid words contain these 10 words:

ANAGRAM, DAM, WRITE, PAPER, MOON, RAN, MAN, GRANDMA, GRAM and DRAG.

Let the second square position double the score of the whole word, the fourth square position triple the score of the tile on it and the sixth square position double the score of the tile on it.

Of all the words in the dictionary, the paper has the necessary tiles to form the following words:

DAM, RAN, MAN, GRANDMA, GRAM and DRAG

Here is the score of each of these words:

DAM = (2 + 1 + 3)*2 = 12 (only a double word score)
RAN = (1 + 1 + 1)*2 = 6 (only a double word score)
MAN = (3 + 1 + 1)*2 = 10 (only a double word score)
GRANDMA = (2 + 1 + 1 + 3*1 + 2 + 2*3 + 1)*2 = 32 (apply triple letter on N, double letter on M, then double the whole score at the end)
GRAM = (2 + 1 + 1 + 3*3)*2 = 26 (triple the M first, then double the whole score)
DRAG = (2 + 1 + 1 + 3*2)*2 = 20 (triple the G, then double the whole score)

Of these options, the highest scoring one is GRANDMA, worth 32 points.

Design wise, it's good to have a function that takes in two strings (one with a word, the other with the tiles) and determines if the word can be formed using the tiles.

It's also good to have another function that takes in a word and scoring information and returns the score of the word.

The Input (read from standard input)

The first line of input will contain a single positive integer, c ($c \leq 20$), representing the number of input cases. The first line of each input case will contain a single positive integer, n ($n \leq 100$), the number of words in the dictionary. The following n lines will each contain a single word in the dictionary. Each word will consist of upper case letters and be no more than 7 letters long. The next line of each input case will contain a string of seven uppercase letters, representing the tiles the player holds for the input case. The following line will contain seven space-separated integers, each 1, 2 or 3, representing the letter multiplier for that square position. (Thus, normal squares will be indicated with 1 and the special squares with 2 and 3. Note that by giving the input in this way, no square is actually special and you can treat each square the same. The last line will contain seven space-separated integers, each 1, 2 or 3 representing that square's word multiplier. It's guaranteed that at least 6 of the 7 values are 1 and that no square with a letter multiplier greater than 1 will have a word multiplier greater than 1.

The Output (to standard out)

For each input case, output a single integer, on a line by itself, representing the maximum possible score for the input case.

Sample Input

```
2
10
ANAGRAM
DAM
WRITE
PAPER
MOON
RAN
MAX
GRANDMA
GRAM
DRAG
ADMARNG
1 1 1 3 1 2 1
1 2 1 1 1 1 1
5
CAT
DOG
STAPLER
WATER
PAINT
TORDGAC
3 2 1 1 2 2 3
1 1 1 3 1 1 1
```

Sample Output

```
32
12
```

Implementation Details - Criteria for Full Credit

Although a solution may work, it may not be awarded full credit as programs are graded on other criteria than correctness. On most programs, my philosophy is not to explicitly state these criteria because I want to encourage independent thinking of my students and reward them for coming up with good design or using good implementation practices. Alternatively, I like having the flexibility of taking off credit for practices that are bug prone or not easily extendible. On most assignments I won't state items that I am looking for, but for this assignment I'll at least post an incomplete list:

1. Scoring options should NOT be hard-coded. Your code should work even if the letter multipliers aren't all 1, 2 or 3 and if the word multiplier is something other than 2 or 3.

2. Since this is testing COP 3223 knowledge only, no use of dynamically allocated memory is needed. The dictionary can be stored in a statically allocated array of size 100 by 8. (Recall that we need 1 extra space to store the NULL character.)

3. Good functional design is required. Programs written all in main will NOT get full credit. For this assignment, a couple suggestions were given for possible functions, but largely, students will be free to design their own functions. If their design shows a lack of understanding of the use of functions, credit may be taken off. (An example is when a student writes two functions that do the exact same thing with different formal parameter names which match their actual parameter names exactly.)

4. Nearly ALL variables must be defined INSIDE the main case loop. At least 10% will be taken off for students who store the input for ALL cases simultaneously. When processing multiple independent cases, it's generally better to process each case individually and to reuse these variables by simply defining them inside the case loop.

5. There must be good programming style. This means the following:

a) Header Comment
b) Good Use of White Space
c) Meaningful Variable Names
d) Reasonable Internal Comments
e) Comment for each Function indicating what it takes in and what it does/returns.

Deliverables

You must submit a two files over WebCourses:

1) A source file, *scrabble.c*. Please use stdin, stdout. There will be an automatic 10% deduction if you don't do so.

2) Your Process Document, which should be named LASTNAME_PROCESS.doc (or .docx or .pdf or .txt)