Week 15 - Introductory Programming Lab

Exercise 12: Strings are Arrays of Characters

Background

You may recall that in Lab 1, we defined strings variables as:

```
char animal[100] = "cat";
```

As you may guess now, we were actually creating an array of characters.

You may have noticed that we can print out a string without needing to specify the total number of characters:

```
char animal[100] = "ferocious kitten";
printf("The %s sleeps!", animal);
```

The above code will print out `"The ferocious kitten sleeps"`, even though the char array `animal` contains 100 chars. This is because the final character is a special ascii-null character: `'\0'` (added automatically by the compiler in this case)

Technical details

Dealing with strings:

```
#include <stdio.h>
#include <string.h> // extra include!

int main() {
    char animal[100] = "ferocious kitten";

    // length of the string
    int length = strlen(animal);

    int i;
    for (i=0; i<length; i++) {
        printf("%c", animal[i]);
    }
    printf("\n");

    getchar();
    return 0;
}
```

We can also copy strings.

```
#include <stdio.h>
#include <string.h>

int main () {
    char orig[100]= "Cats are awesome";
    char next[100];

    // 99 is the maximum number of chars
    strncpy(next,orig,99);

    printf("%s\n", next);

    getchar();
    return 0;
}
```

Your task...

The Coleman-Liau Index test[1] is an easily-computable estimation of the difficulty of reading some text. It gives an approximate "grade level" (in US school system: grade 1 is for students 5-6 years old, and the numbers increase linearly until grade 12).

$$CLI = 5.89\left(\frac{characters}{words}\right) - 29.5\left(\frac{sentences}{words}\right) - 15.8$$

In the above formula, "character" means "letter" -- for the purposes of that formula, spaces and periods do not count as characters.

> [1] Coleman, M.; and Liau, T. L. (1975); *A computer readability formula designed for machine scoring,* Journal of Applied Psychology, Vol. 60, pp. 283-284.
> Original journal paper (campus access only)

Write a program that calculates the Coleman-Liau Index of some text.

- Perform the actual calculation in a function which accepts char `text[]` as input and returns the Coleman-Liau grade level.
- If the calculated grade is less than 1, set it to 1 instead of returning a lower number.
- You may assume that the text:
    - begins with a word
    - ends with a period
    - contains no numbers or non-letters
    - there is one space between words, and one space after a period (other than the final period).
    - each sentence ends with a period; no exclaimation marks or question marks are used.

(hint: what is the relationship between the number of periods, number of spaces, and the words+sentences in the text?)
(another hint: how can you count the number of periods and spaces in the text?)

- Test it with the following three exerpts. For these definitions, you *may ignore* the code style requirement that lines should be less than 80 characters long. When you cut&paste this into your .c file, you might need to ensure that it all goes onto one line.

1. `char text[100] = "I like cats. Cats like me. Miao miao miao. Dogs are bad. Bad dogs bad.";`

   (words of wisdom)
   (hint: the above fragment has 70 characters, 51 letters, 15 words, 14 spaces, 5 periods, 5 sentences, and a calculated CLI of -5.6, which we change to be grade 1.)

2. `char text[500] = "Tomorrow, and tomorrow, and tomorrow, Creeps in this petty pace from day to day, To the last syllable of recorded time; And all our yesterdays have lighted fools The way to dusty death. Out, out, brief candle. Life's but a walking shadow, a poor player That struts and frets his hour upon the stage And then is heard no more. It is a tale Told by an idiot, full of sound and fury Signifying nothing."`

   (Macbeth: Act 5, Scene 5, lines 19-28)

3. `char text[1000] = "Existing computer programs that measure readability are based largely upon subroutines which estimate number of syllables, usually by counting vowels. The shortcoming in estimating syllables is that it necessitates keypunching the prose into the computer. There is no need to estimate syllables since word length in letters is a better predictor of readability than word length in syllables. Therefore, a new readability formula was computed that has for its predictors letters per hundred words and sentences per hundred words. Both predictors can be counted by an optical scanning device, and thus the formula makes it economically feasible for an organization such as the US Office of Education to calibrate the readability of all textbooks for the public school system."`

   (abstract of Coleman-Liau journal paper.)

(optional: research and implement the Automated Readibility Index (ARI). Compare those results with the Coleman-Liau results.)

Show your work to a demonstrator/GTA.

Exercise 13: Combining variables with `struct`

Background

You may have noticed that functions can only return one variable. But what if want to return more information than we can hold in one variable? The answer is to pack multiple variables together into a single *structure*.

This creates a *new type of variable* which contains a number of *fields*. In this section's example, we create a new type of variable called `struct SquareData`. We can declare this just like other variables -- instead of writing `int` or `float`, we write `struct SquareData`.

After we have declared a variable of this type with:

```
struct SquareData mySquare;
```

we can access its fields with `mySquare.side`, `mySquare.area` or any other field name after the period.

Returning a `struct` is a modern addition to the language. It was not supported in the original C standard in 1972, but was added in the ANSI C standard (also sometimes known as "C89" or "C90"). Some very old compilers might not support ANSI C. We will see another way to return multiple values in a few weeks.

Technical details

Structure:

```
#include <stdio.h>

struct SquareData {
  float area;
  float perimeter;
  int side;
};

struct SquareData calcSquare(int x) {
    //define variable of data type structure
    struct SquareData mySquare;

    mySquare.side = x;
    mySquare.perimeter = 4*x;
    mySquare.area = x*x;

    // we need to return this to main()
    return mySquare;
}

void printStuff(struct SquareData mySquare) {
    // this is not useful here, but it works
    float areaBAD = mySquare.area;
```

```
    printf("The square's sides are ");
    printf("%i units long, ", mySquare.side);
    printf("and thus has \nan area of ");
    printf("%f units, and ", areaBAD);
    printf("a perimeter of ");
    printf("%f units.\n", mySquare.perimeter);
}


int main() {
    struct SquareData square;
    square = calcSquare(2);
    printStuff(square);

    getchar();
    return 0;
}
```

The syntax of a `struct` is a bit confusing. Make sure that you follow the example carefully when you use them. Be particularly careful about the final semicolon "`;`"after the `struct` definition's final "`}`", and about adding `struct` in front of the structure's name.

Here's another example of `struct`:

```
#include <stdio.h>

struct TimeData {
  int total_minutes;
  int hours;
  int minutes;
};

struct TimeData calcTime(int day_minutes) {
    struct TimeData time;

    time.total_minutes = day_minutes;
    time.hours = day_minutes / 60;
    time.minutes = day_minutes % 60;

    return time;
}

void printTime(struct TimeData myTime) {
    printf("It has been ");
    printf("%i", myTime.total_minutes);
    printf(" minutes since midnight.\n");

    printf("The time is therefore ");
    printf("%02i:", myTime.hours);
    printf("%02i ", myTime.minutes);
    printf("o'clock.\n");
}
```

```
int main() {
    struct TimeData time;
    time = calcTime(123);
    printTime(time);
    time = calcTime(1234);
    printTime(time);

    getchar();
    return 0;
}
```

Your task...

Vending machines use simple programs to calculate the amount of change to return. For a machine, the coins are dispensed mechanically. In our program, we will output the numbers to the screen, looking something like this:

```
Customer gave 10 pence, item(s) cost 7 pence.
Give customer:
£2      £1      50      20      10      5       2       1
0       0       0       0       0       0       1       1

Customer gave 70 pence, item(s) cost 56 pence.
Give customer:
£2      £1      50      20      10      5       2       1
0       0       0       0       1       0       2       0

Customer gave 200 pence, item(s) cost 124 pence.
Give customer:
£2      £1      50      20      10      5       2       1
0       0       1       1       0       1       0       1

Customer gave 2000 pence, item(s) cost 1232 pence.
Give customer:
£2      £1      50      20      10      5       2       1
3       1       1       0       1       1       1       1
```

Write a program that calculates the amount of coins to return.

Before you begin, figure out the math *on paper*. Show your formula(s) to a lab demonstrator before you start programming.

- Your `int main()` loop must be this:

```
int main() {
    struct Change coins;
    coins = getChange(7, 10);
    printChange(coins);
    coins = getChange(56, 70);
    printChange(coins);
    coins = getChange(124, 200);
    printChange(coins);
```

```
    coins = getChange(1232, 2000);
    printChange(coins);

    getchar();
    return 0;
}
```

Remember that C is case-sensitive. As far as the computer is concerned, UPPER-CASE letters are *completely* different from lower-case letters!

- You must write the `getChange(...)` and `printChange(...)` functions, and the definition of `struct Change`. The first function must do all the math; the second function must do all the `printf(...)`. Use the `struct Change` to pass information between these functions.
- Britain has the following coins: 1 pence, 2 pence, 5 pence, 10 pence, 20 pence, 50 pence, 1 pound (100 pence), and 2 pounds (200 pence). Your machine should not dispense any bills.
- You may assume that your vending machine has an infinite amount of coins, but you should always dispense the largest coins first.
- You do not need to print the pound £ symbol. You may output 200 instead of £2.
- Your program should use `int` for all variables, and keep track of the number of pence instead of pounds.

  You should never keep track of money with float.

  Can you guess why?

(optional: try using float (storing pounds, not pence) to program your vending machine. Can you make it give accurate results?)

Show your work to a demonstrator/GTA.