CS110: Computer Programming Lab
Department of CSE
IIT, Guwahati

Module 04 Stage 01 Exercise 06

Exercise

A common puzzle problem set has problems where letters are assigned digits to match the equations.

For example, S E N D + M O R E = M O N E Y.

We, however, have a different problem included in the code listed in the Appendix. In this assessment exercise, the problem is specified through 4 strings (words). You are required to first convert these strings into arrays of letters. The letters at the least significant positions should be at index `0` in the arrays. This can be easily done by adding less than 10 lines of code in recursive function `getLettes()` already declared in the outline code given in the appendix.

Global array `inuse[]` is there to track which digits (there are only 10 different digits) are already allocated to the letters in the puzzle. Second global array `digit[]` will be indexed by the letters in our puzzle. Global variables, like global declarations, are visible and usable in all functions in the program.

Work on the problem as follows. First copy the code given code into your c file. Add code to complete printing related functions so that you get the following output from the program.

```
        T W O
    T H R E E
    S E V E N
-------------
  T W E L V E
```

After you have done this you can begin to write recursive function `assign()` as per the specifications below and comments in the given code. You should notice that array `sum` and 3 arrays of letters are listed as formal parameters in many functions. In all functions, parameter `idx` is the index of the location which is to be solved next; `carry` is the carry coming from index position `idx-1`. As we all know numbers are added beginning with the units position and progressively moving to the positions of tens, hundreds, and thousands,….

Your recursive function `assign()` must follow this practice. It will first assign digits to the letters in the index for units position (`idx = 0`). Corresponding letters in 4-arrays will be assigned digit values; auxiliary functions `assignX()`, `assignY()`, `assignZ()` support his process. The digits assigned to a letter are no more available for the other letters. This is recorded by setting an element in array `inuse[]` to 1. Setting `inuse[digit] = 1` indicates `digit` is assigned to a letter. Read given code to understand how a letter is paired with a digit.

A recursive call may succeed or report failure to assign digit values to the remaining unpaired letters. If it reports a failure you need to clear `inuse[digit]` status for the digit and pair a different digit to the letter.

Once all letters at index `idx` have digits paired to them, you must check that their values validate correctly. That is, `digit[sum[idx]] == (digit[x[idx]]+digit[y[idx]]+digit[z[idx]]+ carry)%10`. If the pairing does not validate, the association needs to be changed and a new pairing tried again.

Once the letters at index `idx` are set, make a recursive call to function `assign()` to arrange things at index positions `idx+1` and above. As shown in the appendix functions `assign()`, `assignX()`, `assignY()` and `assignZ()` each work on one letter at a time. The programming pattern used, spreads the work over these functions neatly. Each function is concerned with one array. And they all have similar code.

Final solution when program is done will print result as follows.

```
          1 0 6
      1 9 7 2 2
      8 2 5 2 4
    ------------
    1 0 2 3 5 2
```

Please also read a guideline provided as a separate document in this drill set.

Appendix

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

int digit[200], /* Digit value for char with code of index */
inuse[11]; /*Digits already in use. Actually only 10 needed*/

char * getLetters(char *from, char *to)
{
/* ******************************************************
 * Recursive function to move chars in string from
 * into array to. Reverses order to help align letters
 * in puzzle words on unit position (index 0 in to)
 * ****************************************************/
}

void printOneNum(char n[])
{
    /* Code provided for training purposes */
    int i;
    for (i = 10; i>=0; i--)
        if (n[i] == ' ' || digit[n[i]] > 9 ||
                digit[n[i]] <0 || n[i] =='0')
            printf(" %c", n[i]);
        else printf("%2d", digit[n[i]]);
    printf("\n");
}

void printPuzzle(char sum[], char x[], char y[], char z[])
{
    int i;
    printOneNum(x);
    printOneNum(y);
    printOneNum(z);
    for (i = 10; i>=0; i--)
        if (sum[i] == ' ')
            printf("  ");
        else printf("--");
    printf("\n");
    printOneNum(sum);
    printf("\n");
}

int genCarry (char sum[], char x[], char y[], char z[],
              int idx, int carry)
{
    /* ************************************************
     * Checks that values for letters at inddex idx
     * in arrays x, y and z add to letter value in sum.
     * Carry is carry in. Computes carry out (carry
     * into idx+1) that is returned as function value.
     * Carry out of -1 denotes letter values are wrong.
     * ***********************************************/
}

int assignZ(char sum[], char x[], char y[], char z[],
            int idx, int carry)
{
    /* ***********************************************
     * Assigns a trial digit value to letter z[idx],
     * if needed. Will return 0 if the previously set
     * letter-value pairs do not give any puzzle solution.
     *
     * Uses genCarry() to check if all is good upto
     * idx. Makes recursive call to assign() to let
     * that function set letter-value pairs for letters at
     * locations > idx.
     *
     * Return status is based on the success or
     * failure to assign digit value to z[idx] that
     * delivers a puzzle solution.
     * ***********************************************/
}

int assignY(char sum[], char x[], char y[], char z[],
            int idx, int carry)
{
    /* **********************************************
     * Assign digit value to y[idx]. Calls assignZ().
     * Called by assignX().
     * ***********************************************/
}

int assignX(char sum[], char x[], char y[], char z[],
            int idx, int carry)
{
     /* **********************************************
     * Assign digit value to x[idx]. Calls assignY().
     * Called by assign().
     * ***********************************************/
}

int assign(char sum[], char x[], char y[], char z[],
           int idx, int carry)
{
    /* ************************************************
     * Recursive function to assign values to letters in
     * the puzzle. It sets digit value for sum[idx] and then
     * calls assignX() to set digit values for other letters
     *
     * The function gets recursive call from assignZ() to
     * (recursively) continue work at index idx+1.
     *
     * Will return 1 if a conbination is found that solves
     * the puzzle.
     * ***************************************************/
}

int main()
{
    /* ***************************************************
     * Code provided for training and learning
     *
     * About 100 lines of code have been renomved from
     * this program.
     * ***************************************************/
    char two[] = "TWO", three[] = "THREE";
    char seven[] = "SEVEN", twelve[] = "TWELVE";
    /* This can be used to solve another problem
    char two[] = "", three[] = "SEND";
    char seven[] = "MORE", twelve[] = "MONEY";
    */

    char adder[3][11];
    char sum[11];
    int i;

    for (i=0; i<200; i++) digit[i] = i;
    digit[' '] = 0;
    digit['0'] = 0;

    for (i=0; i<11; i++)
    {
        adder[0][i] = ' ';
        adder[1][i] = ' ';
        adder[2][i] = ' ';
        sum[i] = ' ';
        inuse[i] = 0;
    }

    getLetters(two, &adder[0][0]);
    getLetters(three, &adder[1][0]);
    getLetters(seven, &adder[2][0]);
    getLetters(twelve, sum);

    printPuzzle(sum, adder[0], adder[1], adder[2]);
    printf("\n");

    if (assign(sum, adder[0], adder[1], adder[2], 0, 0))
    {
        printf ("Solution found:\n");
        printPuzzle(sum, adder[0], adder[1], adder[2]);
    }
    else printf("No solution exists\n");
}
```

Error Reporting and Suggestions for Improvements

My sincere apologies if the document has errors or mistakes. Please report errors in this document to vmm@iitg.ernet.in. Also, I welcome suggestions and advice to improve the quality of the document for the students of CS110.