USA COMPUTING OLYMPIAD

Name That Number

Russ Cox & Rob Kolstad

There are two ways to do this problem. One is, given the number, to generate all possible strings that encode to that number and look them up in the dictionary. Since there are 3 letters for each number and 12 digits in the string, that's $3^{12} = 531441$ lookups into a dictionary of size 5000, which although manageable would be a little on the long side (binary search can help this).

Or, we can examine each word in the dictionary to see if it maps to the digits of the number in question. This has the the advantage of a shorter program that probably will work right first time.

Here is Argentina competitor's Michel Mizrah's solution using the first method with a binary search. While it is blazingly fast, it does have the disadvantage of some fairly tricky coding in the binary search routine. A single off-by-one error would doom a program in a contest.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char num[12],sol[12];
char dict[5000][13];
int nsolutions = 0;
int nwords;
int maxlen;
FILE *out;

void calc (int charloc, int low, int high) {
    if (charloc == maxlen) {
        sol[charloc] = '\0';
        for (int x = low; x < high; x++) {
            if (strcmp (sol, dict[x]) == 0) {
                fprintf (out, "%s\n", sol);
                nsolutions++;
            }
        }
        return;
   }
   if (charloc > 0) {
        for (int j=low; j <= high; j++){
            if (sol[charloc-1] == dict[j][charloc-1]) {
                low=j;
                while (sol[charloc-1] == dict[j][charloc-1])
                    j++;
                high=j;
                break;
            }
            if (j == high) return;
        }
    }
    if (low > high) return;
    switch(num[charloc]){
      case '2':sol[charloc] = 'A'; calc(charloc+1,low,high);
               sol[charloc] = 'B'; calc(charloc+1,low,high);
               sol[charloc] = 'C'; calc(charloc+1,low,high);
               break;
      case '3':sol[charloc] = 'D'; calc(charloc+1,low,high);
```

```
                sol[charloc] = 'E'; calc(charloc+1,low,high);
                sol[charloc] = 'F'; calc(charloc+1,low,high);
                break;
      case '4':sol[charloc] = 'G'; calc(charloc+1,low,high);
                sol[charloc] = 'H'; calc(charloc+1,low,high);
                sol[charloc] = 'I'; calc(charloc+1,low,high);
                break;
      case '5':sol[charloc] = 'J'; calc(charloc+1,low,high);
                sol[charloc] = 'K'; calc(charloc+1,low,high);
                sol[charloc] = 'L'; calc(charloc+1,low,high);
                break;
      case '6':sol[charloc] = 'M'; calc(charloc+1,low,high);
                sol[charloc] = 'N'; calc(charloc+1,low,high);
                sol[charloc] = 'O'; calc(charloc+1,low,high);
                break;
      case '7':sol[charloc] = 'P'; calc(charloc+1,low,high);
                sol[charloc] = 'R'; calc(charloc+1,low,high);
                sol[charloc] = 'S'; calc(charloc+1,low,high);
                break;
      case '8':sol[charloc] = 'T'; calc(charloc+1,low,high);
                sol[charloc] = 'U'; calc(charloc+1,low,high);
                sol[charloc] = 'V'; calc(charloc+1,low,high);
                break;
      case '9':sol[charloc] = 'W'; calc(charloc+1,low,high);
                sol[charloc] = 'X'; calc(charloc+1,low,high);
                sol[charloc] = 'Y'; calc(charloc+1,low,high);
                break;
   }
}

int main(){
    FILE *in=fopen ("namenum.in", "r");
    FILE *in2=fopen ("dict.txt", "r");
    int j;
    out=fopen ("namenum.out","w");
    for (nwords = 0; fscanf (in2, "%s", &dict[nwords++]) != EOF; )
        ;
    fscanf (in, "%s",&num);
    maxlen = strlen(num);
    calc (0, 0, nwords);
    if (nsolutions == 0) fprintf(out,"NONE\n");
    return 0;
}
```

The solution below might be considered to be a bit more straightforward: no tricky offsets, no +1 or -1, no knowledge about character values. The lines of actual code in this solution are minimal.

This is the sort of program that might work reliably the first time and every time. The only tricky part is knowing that scanf will yield string *without* a newline on the end:

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

int main() {
    FILE *in = fopen ("namenum.in", "r");
    FILE *in2 = fopen ("dict.txt", "r");
    FILE *out = fopen ("namenum.out","w");
    int nsolutions = 0;
    int numlen;
    char word[80], num[80], *p, *q, map[256];
    int i, j;
    map['A'] = map['B'] = map['C'] = '2';
    map['D'] = map['E'] = map['F'] = '3';
    map['G'] = map['H'] = map['I'] = '4';
    map['J'] = map['K'] = map['L'] = '5';
    map['M'] = map['N'] = map['O'] = '6';
    map['P'] = map['R'] = map['S'] = '7';
```

```
    map['T'] = map['U'] = map['V'] = '8';
    map['W'] = map['X'] = map['Y'] = '9';
    fscanf (in, "%s",num);
    numlen = strlen(num);
    while (fscanf (in2, "%s", word) != EOF) {
        for (p=word, q=num; *p && *q; p++, q++) {
            if (map[*p] != *q)
                break;
        }
        if (*p == '\0' && *q == '\0') {
            fprintf (out, "%s\n", word);
            nsolutions++;
        }
    }
    if (nsolutions == 0) fprintf(out,"NONE\n");
    return 0;
}
```