```
/**
 * Project 1
 * Assembler code fragment for LC-2K
 */

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

#define MAXLINELENGTH 1000

struct Table {
    int off;
    char type;
    char ins[7];
    char lab[7];
    char tab;
};

int readAndParse(FILE *, char *, char *, char *, char *, char *);
int isNumber(char *);
char instrType (char* s);
int opcodeBin (char* s);

int
main(int argc, char *argv[])
{
    char *inFileString, *outFileString;
    FILE *inFilePtr, *outFilePtr;
    char label[MAXLINELENGTH], opcode[MAXLINELENGTH], arg0[MAXLINELENGTH],
            arg1[MAXLINELENGTH], arg2[MAXLINELENGTH];

    if (argc != 3) {
        printf("error: usage: %s <assembly-code-file> <machine-code-file>\n",
            argv[0]);
        exit(1);
    }

    inFileString = argv[1];
    outFileString = argv[2];

    inFilePtr = fopen(inFileString, "r");
    if (inFilePtr == NULL) {
        printf("error in opening %s\n", inFileString);
        exit(1);
    }
    outFilePtr = fopen(outFileString, "w");
    if (outFilePtr == NULL) {
        printf("error in opening %s\n", outFileString);
        exit(1);
    }

    char labelList[256][7];
    char globalList[256][7];
    struct Table table[256];
    int size = 0;
    int count = 0;
    int dCount = 0;
```

```
    int text = 0;
    int data = 0;
    int sym = 0;
    int rec = 0;
    while (readAndParse(inFilePtr, label, opcode, arg0, arg1, arg2)) {
        if (label[0] >= 65 && label[0] <= 90) {
            for(int i = 0; i < count; ++i) {
                if(!strcmp(globalList[i], label) && strlen(globalList[i]) != 0) {
                    exit(1);
                }
            }
            strcpy(globalList[count], label);
        }
        else {
            for(int i = 0; i < count; ++i) {
                if(!strcmp(labelList[i], label) && strlen(labelList[i]) != 0) {
                    exit(1);
                }
            }
            strcpy(labelList[count], label);
        }
        if (!strcmp(opcode,".fill")) {
            data++;
            if (label[0] >= 65 && label[0] <= 90) {
                table[size].type = 'D';
                table[size].off = dCount;
                strcpy(table[size].lab,label);
                table[size].tab = 'S';
                size++;
                sym++;
            }
            dCount++;
        }
        else {
            if(label[0] >= 65 && label[0] <= 90) {
            table[size].type = 'T';
            table[size].off = count;
            strcpy(table[size].lab,label);
            table[size].tab = 'S';
            size++;
            sym++;
            }
            text++;
        }
        count++;
    }

    /* this is how to rewind the file ptr so that you start reading from the
        beginning of the file */
    rewind(inFilePtr);
    int t = 0;
    int d = 0;
    while (readAndParse(inFilePtr, label, opcode, arg0, arg1, arg2)) {
        if (!strcmp(opcode, "lw") || !strcmp(opcode, "sw") || !strcmp(opcode,
".fill")) {
            int j = -1;
            if (!strcmp(opcode, ".fill")) {
                if (!isNumber(arg0)) {
                    for(int i = 0; i < count; ++i) {
```

```
                    if(!strcmp(globalList[i], arg0)) {
                        j = i;
                    }
                }
                if (j == -1 && (arg0[0] >= 65 && arg0[0] <= 90)) {
                    for(int i = 0; i < size; ++i) {
                        if (!strcmp(table[i].lab, arg0)) {
                            j = i;
                        }
                    }
                    if (j == -1) {
                        table[size].type = 'U';
                        table[size].off = 0;
                        strcpy(table[size].lab,arg0);
                        table[size].tab = 'S';
                        size++;
                        sym++;
                    }
                }
                table[size].off = d;
                strcpy(table[size].lab,arg0);
                strcpy(table[size].ins,opcode);
                table[size].tab = 'D';
                size++;
                rec++;
            }
            d++;
        }
        else {
            if (!isNumber(arg2)) {
                for(int i = 0; i < count; ++i) {
                    if(!strcmp(globalList[i], arg2)) {
                        j = i;
                    }
                }
                if (j == -1 && (arg2[0] >= 65 && arg2[0] <= 90)) {
                    for(int i = 0; i < size; ++i) {
                        if (!strcmp(table[i].lab, arg2)) {
                            j = i;
                        }
                    }
                    if (j == -1) {
                        table[size].type = 'U';
                        table[size].off = 0;
                        strcpy(table[size].lab,arg2);
                        table[size].tab = 'S';
                        size++;
                        sym++;
                    }
                }
                table[size].off = t;
                strcpy(table[size].lab,arg2);
                strcpy(table[size].ins,opcode);
                table[size].tab = 'D';
                size++;
                rec++;
            }
        }
    }
```

```
    t++;
}
/* after doing a readAndParse, you may want to do the following to test the
    opcode */
fprintf(outFilePtr,"%d ", text);
fprintf(outFilePtr,"%d ", data);
fprintf(outFilePtr,"%d ", sym);
fprintf(outFilePtr,"%d\n", rec);
rewind(inFilePtr);
int c = 0;

while (readAndParse(inFilePtr, label, opcode, arg0, arg1, arg2) ) {
    int beg = 0;
    if (instrType(opcode) == 'R') {
        beg <<= 10;
        beg |= opcodeBin(opcode);
        int regA = atoi(arg0);
        beg <<= 3;
        beg |= regA;
        int regB = atoi(arg1);
        beg <<= 3;
        beg |= regB;
        int unused = 0;
        beg <<= 13;
        beg |= unused;
        int dest = atoi(arg2);
        beg <<= 3;
        beg |= dest;
    }
    else if (instrType(opcode) == 'J') {
        beg <<= 3;
        beg |= opcodeBin(opcode);
        int regA = atoi(arg0);
        beg <<= 3;
        beg |= regA;
        int regB = atoi(arg1);
        beg <<= 3;
        beg |= regB;
        //int unused = 0;
        beg <<= 16;
        //beg |= unused;
    }
    else if (instrType(opcode) == 'O') {
        beg <<= 10;
        beg |= opcodeBin(opcode);
        int unused = 0;
        beg <<= 22;
        beg |= unused;
    }
    else if (instrType(opcode) == 'I') {
        int off = -1;
        int lab = 0;
        int globUndef = 0;
        if (isNumber(arg2) == 1) {
            off = atoi(arg2);
            if (off < -32768 || off > 32767) {
                exit(1);
            }
        }
```

```
else {
    if(arg2[0] >= 65 && arg2[0] <= 90) {
        for(int i = 0; i < count; ++i) {
            if (!strcmp(arg2,globalList[i])) {
                off = i;
                lab = 1;
            }
            if (off == -1) {
                off = 0;
                globUndef = 1;
            }
        }
    }
    else {
        for(int i = 0; i < count; ++i) {
            if (!strcmp(arg2,labelList[i])) {
                off = i;
                lab = 1;
            }
        }
        if (off == -1) {
            exit(1);
        }
    }

}
if (strcmp(opcode,"beq") == 0) {
    if(globUndef == 1 && off == 0) {
        exit(1);
    }
    beg <<= 10;
    beg |= opcodeBin(opcode);
    int regA = atoi(arg0);
    beg <<= 3;
    beg |= regA;
    int regB = atoi(arg1);
    beg <<= 3;
    beg |= regB;
    beg <<= 16;
    if (lab == 1) {
    off = off - c - 1;
    }
    off &= 0xffff;
    beg |= off;

}
else {
    beg <<= 10;
    beg |= opcodeBin(opcode);
    int regA = atoi(arg0);
    beg <<= 3;
    beg |= regA;
    int regB = atoi(arg1);
    beg <<= 3;
    beg |= regB;
    beg <<= 16;
    off &= 0xffff;
    beg |= off;
}
```

```
        }
        else if (!strcmp(opcode, ".fill")){
            if (isNumber(arg0) == 1) {
                beg = atoi(arg0);
            }
            else {
                if(arg0[0] >= 65 && arg0[0] <= 90) {
                    for(int i = 0; i < count; ++i) {
                        if (!strcmp(arg0,globalList[i])) {
                            beg = i;
                        }
                    }
                }
                else {
                    for(int i = 0; i < count; ++i) {
                        if (!strcmp(arg0,labelList[i])) {
                            beg = i;
                        }
                    }
                }
            }
        }
        else
        {
            exit(1);
        }
        fprintf(outFilePtr,"%x\n", beg);
        c++;
    }
    for(int i = 0; i < size; ++i) {
        if (table[i].tab == 'S') {
        fprintf(outFilePtr,"%s ", table[i].lab);
        fprintf(outFilePtr,"%C ", table[i].type);
        fprintf(outFilePtr,"%d\n", table[i].off);
        }
    }
    for(int i = 0; i < size; ++i) {
        if (table[i].tab == 'D') {
        fprintf(outFilePtr,"%d ", table[i].off);
        fprintf(outFilePtr,"%s ", table[i].ins);
        fprintf(outFilePtr,"%s\n", table[i].lab);
        }
    }

    return(0);
}

char instrType (char* s){
    if (strcmp(s, "add") == 0 || strcmp(s, "nor") == 0) {
        return 'R';
    }
    else if (strcmp(s,"lw") == 0 || strcmp(s,"sw") == 0 || strcmp(s,"beq") == 0) {
        return 'I';
    }
    else if (strcmp(s, "jalr") == 0) {
        return 'J';
    }
    else if (strcmp(s, "halt") == 0 || strcmp(s, "noop") == 0) {
        return 'O';
```

```
    }
    return 'W';
}

int opcodeBin (char* s) {
    if (strcmp(s, "add") == 0) {
        return 0;
    }
    else if (strcmp(s, "nor") == 0) {
        return 1;
    }
    else if (strcmp(s, "lw") == 0) {
        return 2;
    }
    else if (strcmp(s, "sw") == 0) {
        return 3;
    }
    else if (strcmp(s, "beq") == 0) {
        return 4;
    }
    else if (strcmp(s, "jalr") == 0) {
        return 5;
    }
    else if (strcmp(s, "halt") == 0) {
        return 6;
    }
    else if (strcmp(s, "noop") == 0) {
        return 7;
    }
    return -1;
}

/*
 * Read and parse a line of the assembly-language file.  Fields are returned
 * in label, opcode, arg0, arg1, arg2 (these strings must have memory already
 * allocated to them).
 *
 * Return values:
 *     0 if reached end of file
 *     1 if all went well
 *
 * exit(1) if line is too long.
 */
int
readAndParse(FILE *inFilePtr, char *label, char *opcode, char *arg0,
    char *arg1, char *arg2)
{

    char line[MAXLINELENGTH];
    char *ptr = line;

    /* delete prior values */
    label[0] = opcode[0] = arg0[0] = arg1[0] = arg2[0] = '\0';

    /* read the line from the assembly-language file */
    if (fgets(line, MAXLINELENGTH, inFilePtr) == NULL) {
      /* reached end of file */
        return(0);
    }
```

```
    /* check for line too long */
    if (strlen(line) == MAXLINELENGTH-1) {
      printf("error: line too long\n");
      exit(1);
    }

    /* is there a label? */
    ptr = line;
    if (sscanf(ptr, "%[^\t\n ]", label)) {
      /* successfully read label; advance pointer over the label */
        ptr += strlen(label);
    }

    /*
     * Parse the rest of the line.  Would be nice to have real regular
     * expressions, but scanf will suffice.
     */
    sscanf(ptr, "%*[\t\n\r ]%[^\t\n\r ]%*[\t\n\r ]%[^\t\n\r ]%*[\t\n\r ]%[^\t\n\r ]
%*[\t\n\r ]%[^\t\n\r ]",
        opcode, arg0, arg1, arg2);
    return(1);
}

int
isNumber(char *string)
{
    /* return 1 if string is a number */
    int i;
    return( (sscanf(string, "%d", &i)) == 1);
}
```