Assignment description: Write a C program that allows creating and solving a crossword of size 12x12 cells. It should first ask the user to construct a crossword: ask new puzzle words and clues, check if the crossword is correct on each stage (words fit the board and they intersect each other in a proper way). Then the user is welcomed to solve the crossword.

Solution:

File crossword.c:

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

#define WORD_LENGTH 10
#define CLUE_LENGTH 25
#define MAX_NUM_OF_WORDS 10
#define BUFFER_SIZE 80
#define BOARD_SIZE 12

typedef struct {
    char word[WORD_LENGTH + 2];
    char row;
    int column;
    char direction;
    char clue[CLUE_LENGTH + 2];
    int isGuessed;
}
Clue;

/* Fills the board with dots */
void initBoard(char board[BOARD_SIZE][BOARD_SIZE], int size);
/* Displays the crossword */
void displayBoard(char board[BOARD_SIZE][BOARD_SIZE], int size);
/* Dispays the clues ACROSS and DOWN */
void displayClues(Clue * clues, int wordsCnt);
/* Parameters: clue pointer, string to parse.
 * Converts string to appropriate fields of a clue.
 * Returns -1 if the number of parsed parameters is less than 5.
 * Returns 1 if dot is entered.
 * Returns 0 otherwise. */
int parseClue(Clue * clue, char * str);
/* Returns -1 if the word has non-alphabetic characters or
 * is longer than WORD_LENGTH.
 * Returns 0 otherwise. */
int checkPuzzleWord(char * word);
/* Returns -1 if the row is not from 'a' - 'l' or column is not from 0 - 11.
 * Returns 0 otherwise. */
```

```
int checkStartingPositionFormat(char row, int column, int boardSize);
/* Returns 0 if direction is 'a' (across) or 'd' (down).
 * Returns -1 otherwise. */
int checkDirectionFormat(char direction);
/* Returns 0 if clue contains only characters 'a' - 'z' and white spaces and
 * the length is no more than 25 characters.
 * Returns -1 otherwise. */
int checkClue(char * clue);
/* If the puzzle word does not fit within the 12x12 grid,
 * returns -1. Otherwise returns 0. */
int fitWord(Clue clue, int boardSize);
/* Returns -1 if the word overlaps existing words.
 * Returns 0 otherwise. */
int checkWordOverlap(Clue clue, char board[BOARD_SIZE][BOARD_SIZE], Clue * clues,
int wordsCnt);
/* Writes the word in the corresponding place on the board */
void addWordToBoard(char board[BOARD_SIZE][BOARD_SIZE], Clue clue);
/* Replaces all the letters with white spaces. */
void clearBoard(char board[BOARD_SIZE][BOARD_SIZE], int size);
/* Parameters: clue pointer, string to parse.
 * Converts string to appropriate fields of a guess.
 * Returns -1 if the number of parsed parameters is less than 4. */
int parseGuess(Clue * guess, char * str);
/* Searchs the word starting at the position.
 * Returns -1 if the word is not found.
 * Returns 1 if dot is entered.
 * Returns 0 otherwise.
 */
int checkStartingPosition(char row, int column, Clue * clues, int wordsCnt);
/* Returns the index of the clue in the array.
 * Returns -1 if there is no clue at the position and with the same direction.
*/
int checkDirection(Clue guess, Clue * clues, int wordsCnt);
int stringToInt(char * str);

int main(int argc, char ** argv) {
    int wordsCnt = 0, guessedWords = 0, guessPosition, parseResult;
    char buffer[BUFFER_SIZE], board[BOARD_SIZE][BOARD_SIZE];
    Clue clues[MAX_NUM_OF_WORDS], guess;

    /* Stage 1 */
    initBoard(board, BOARD_SIZE);

    while (wordsCnt < MAX_NUM_OF_WORDS) {
        printf("Enter new clue (type . if finished):");
        gets(buffer);
        parseResult = parseClue(&clues[wordsCnt], buffer);
        if (parseResult == 1) /* dot was entered */
            break;
        if (parseResult == -1) {
            printf("Less than 5 fields provided, ");
            continue;
        }
        if (checkPuzzleWord(clues[wordsCnt].word) == -1) {
            printf("Puzzle word format is incorrect, ");
            continue;
        }
```

```
        if (checkStartingPositionFormat(clues[wordsCnt].row,
clues[wordsCnt].column, BOARD_SIZE) == -1) {
            printf("Starting position format is incorrect, ");
            continue;
        }
        if (checkDirectionFormat(clues[wordsCnt].direction) == -1) {
            printf("Direction format is incorrect, ");
            continue;
        }
        if (checkClue(clues[wordsCnt].clue) == -1) {
            printf("Clue format is incorrect, ");
            continue;
        }
        if (fitWord(clues[wordsCnt], BOARD_SIZE) == -1) {
            printf("Word cannot fit in the grid, ");
            continue;
        }
        if (checkWordOverlap(clues[wordsCnt], board, clues, wordsCnt) == -1) {
            printf("Word overlaps with existing word, ");
            continue;
        }
        /* Here the input is valid */
        addWordToBoard(board, clues[wordsCnt]);
        ++wordsCnt;
        displayBoard(board, BOARD_SIZE);
        displayClues(clues, wordsCnt);
    }
    printf("\nEnd of stage 1, proceeding to puzzle solving stage.\n");
    /* Stage 2 */
    /* clear board */
    clearBoard(board, BOARD_SIZE);
    displayBoard(board, BOARD_SIZE);
    displayClues(clues, wordsCnt);
    while (guessedWords < wordsCnt) {
        printf("Enter guess (type . if finished):");
        gets(buffer);
        parseResult = parseGuess(&guess, buffer);
        if (parseResult == 1)
            return 0;
        if (parseResult == -1) {
            printf("Less than 4 fields provided, ");
            continue;
        }
        if (checkStartingPositionFormat(guess.row, guess.column, BOARD_SIZE) ==
-1) {
            printf("Starting position format is incorrect, ");
            continue;
        }
        if (checkStartingPosition(guess.row, guess.column, clues, wordsCnt) ==
-1) {
            printf("There is no word starting at this position, ");
            continue;
        }
        if (checkDirectionFormat(guess.direction) == -1) {
            printf("Direction format is incorrect, ");
            continue;
        }
```

```
            guessPosition = checkDirection(guess, clues, wordsCnt);
            if (guessPosition == -1) {
                printf("Direction of the guessed word is incorrect, ");
                continue;
            }
            if (clues[guessPosition].isGuessed != 0) {
                printf("Word already guessed, ");
                continue;
            }
            /* Here the input is valid */
            if (strcmp(guess.word, clues[guessPosition].word) == 0) {
                printf("Your guess is correct.\n");
                clues[guessPosition].isGuessed = 1;
                addWordToBoard(board, clues[guessPosition]);
                ++guessedWords;
                displayBoard(board, BOARD_SIZE);
                displayClues(clues, wordsCnt);
            }
            else
                printf("Sorry, your guess is incorrect.\n");
        }

        printf("Congratulations, you have completed the crossword.\n");
        return 0;
    }

    void initBoard(char board[BOARD_SIZE][BOARD_SIZE], int size) {
        int i, j;

        for (i = 0; i < size; ++i)
            for (j = 0; j < size; ++j)
                board[i][j] = '.'; // fill board with dots */
    }

    void displayBoard(char board[BOARD_SIZE][BOARD_SIZE], int size) {
        char i;
        int j;

        printf("\n  ");
        for (j = 0; j < size; ++j) {
            printf("%i", j);
            if (j < 10)
                printf(" ");
        }
        printf("\n");
        for (i = 'a'; i < 'a' + size; ++i)
        {
            printf("%c ", i);
            for (j = 0; j < size; ++j)
                printf("%c ", board[i - (int)'a'][j]);
            printf("\n");
        }
        printf("\n");
    }

    void displayClues(Clue * clues, int wordsCnt) {
        int i;
```

```
    printf("ACROSS:\n");
    printf("Row\tCol\tClue\n");
    for (i = 0; i < wordsCnt; ++i)
        if (clues[i].direction == 'a')
            printf("%c\t%i\t%s\n", clues[i].row, clues[i].column,
clues[i].clue);
    printf("\nDOWN:\n");
    printf("Row\tCol\tClue\n");
    for (i = 0; i < wordsCnt; ++i)
        if (clues[i].direction == 'd')
            printf("%c\t%i\t%s\n", clues[i].row, clues[i].column,
clues[i].clue);
    printf("\n");
}

int parseClue(Clue * clue, char * str, int numOfParameters) {
    int i = 0, pos;

    /* clear clue */
    clue->isGuessed = 0;
    clue->direction = 0;
    clue->row = 0;
    clue->column = -1;
    strcpy(clue->word, "");
    strcpy(clue->clue, "");

    if (str[0] == '.') /* dot was entered */
        return 1;

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;
    /* copy puzzle word */
    while (str[i] != ' ' && str[i] != '\t' && str[i] != '\0') {
        if (i <= WORD_LENGTH) /* do not overflow the buffer */
            clue->word[i] = str[i];
        ++i;
    }
    if (i <= WORD_LENGTH)
        clue->word[i] = '\0'; /* end of the string */
    else
        clue->word[WORD_LENGTH + 1] = '\0'; /* end of the string, the word is longer
that WORD_LENGTH */

    if (str[i] == '\0') /* lack of input parameters */
        return -1;

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;

    pos = i;

    /* get row letter */
    while (str[i] != ' ' && str[i] != '\t' && str[i] != '\0') {
        clue->row = str[i];
        ++i;
    }
```

```
    if (str[i] == '\0') /* invalid number of parameters */
        return -1;

    if (i - pos > 1) /* more than 1 letter for row, wrong row, check it later */
        clue->row = 'a' - 1; /* assign wrong row */

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;

    /* get column number */
    clue->column = stringToInt(&str[i]);
    while (str[i] != ' ' && str[i] != '\t' && str[i] != '\0') /* skip characters
*/
        ++i;

    if (str[i] == '\0') /* invalid number of parameters */
        return -1;

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;

    pos = i;
    /* get direction */
    while (str[i] != ' ' && str[i] != '\t' && str[i] != '\0') {
        clue->direction = str[i];
        ++i;
    }

    if (str[i] == '\0') /* invalid number of parameters */
        return -1;

    if (i - pos > 1) /* more than 1 letter for direction, wrong direction, check
it later */
        clue->direction = 'a' - 1; /* assign wrong direction */

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;

    if (str[i] == '\0') /* invalid number of parameters */
        return -1;

    pos = i;
    /* copy clue */
    while (str[i] != '\0') {
        if (i - pos <= CLUE_LENGTH) /* do not overflow the buffer */
            clue->clue[i - pos] = str[i];
        ++i;
    }
    if (i - pos <= CLUE_LENGTH)
        clue->clue[i - pos] = '\0'; /* end of the string */
    else
        clue->clue[CLUE_LENGTH + 1] = '\0'; /* end of the string, the word is longer
that WORD_LENGTH */

    return 0;
}
```

```
int checkPuzzleWord(char * word) {
    int i;

    if (strlen(word) > WORD_LENGTH) /* wrong length */
        return -1;

    for (i = 0; i < strlen(word); ++i)
        if (word[i] < 'a' || word[i] > 'z') /* non-alphabetic character */
            return -1;

    return 0;
}

int checkStartingPositionFormat(char row, int column, int boardSize) {
    if (row < 'a' || row >= 'a' + boardSize || column < 0 || column >= boardSize)
        return -1; /* if BOARD_SIZE == 12 then 'a' + 12 = 'm' > 'l' */
    else
        return 0;
}

int checkDirectionFormat(char direction) {
    if (direction == 'a' || direction == 'd')
        return 0;
    else
        return -1;
}

int checkClue(char * clue) {
    int i;

    if (strlen(clue) > CLUE_LENGTH)
        return -1;

    for (i = 0; i < strlen(clue); ++i)
        if ((clue[i] < 'a' || clue [i] > 'z') && clue[i] != ' ')
            return -1;

    return 0;
}

int fitWord(Clue clue, int boardSize) {
    if (clue.direction == 'a') /* across */
        if (clue.column + strlen(clue.word) <= boardSize) /* position of the last
character */
            return 0;
        else
            return -1;
    if (clue.direction == 'd') /* down */
        if (clue.row - 'a' + strlen(clue.word) <= boardSize) /* position of the
last character */
            return 0;
        else
            return -1;
    return -1;
}

int checkWordOverlap(Clue clue, char board[BOARD_SIZE][BOARD_SIZE], Clue * clues,
```

```
    int wordsCnt) {
    int i;

    /* search for the same clue */
    for (i = 0; i < wordsCnt; ++i)
        if (clues[i].column == clue.column && clues[i].row == clue.row &&
clues[i].direction == clue.direction)
            return -1;

    /* check letters on the board */
    if (clue.direction == 'a') {
        for (i = 0; i < strlen(clue.word); ++i)
            if (board[clue.row - (int)'a'][clue.column + i] != '.' &&
                board[clue.row - (int)'a'][clue.column + i] != clue.word[i]) /*
letters don't match */
                return -1;
        return 0;
    }
    if (clue.direction == 'd') {
        for (i = 0; i < strlen(clue.word); ++i)
            if (board[clue.row - (int)'a' + i][clue.column] != '.' &&
                board[clue.row - (int)'a' + i][clue.column] != clue.word[i]) /*
letters don't match */
                return -1;
        return 0;
    }
    return -1;
}

void addWordToBoard(char board[BOARD_SIZE][BOARD_SIZE], Clue clue) {
    int i;

    if (clue.direction == 'a')
        for (i = 0; i < strlen(clue.word); ++i)
            board[clue.row - (int)'a'][clue.column + i] = clue.word[i];
    else /* down */
        for (i = 0; i < strlen(clue.word); ++i)
            board[clue.row - (int)'a' + i][clue.column] = clue.word[i];
}

void clearBoard(char board[BOARD_SIZE][BOARD_SIZE], int size) {
    int i, j;

    for (i = 0; i < size; ++i)
        for (j = 0; j < size; ++j)
            if (board[i][j] != '.')
                board[i][j] = ' ';
}

int parseGuess(Clue * guess, char * str)
{
    int i = 0, pos;

    /* clear guess */
    guess->direction = 0;
    guess->row = 0;
    guess->column = -1;
```

```
    strcpy(guess->word, "");

    if (str[0] == '.')
        return 1;

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;

    pos = i;
    /* get row letter */
    while (str[i] != ' ' && str[i] != '\t' && str[i] != '\0') {
        guess->row = str[i];
        ++i;
    }

    if (str[i] == '\0') /* invalid number of parameters */
        return -1;

    if (i - pos > 1) /* more than 1 letter for row, wrong row, check it later */
        guess->row = 'a' - 1; /* assign wrong row */

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;

    /* get column number */
    guess->column = stringToInt(&str[i]);
    while (str[i] != ' ' && str[i] != '\t' && str[i] != '\0') /* skip characters
*/
        ++i;

    if (str[i] == '\0') /* invalid number of parameters */
        return -1;

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;

    pos = i;
    /* get direction */
    while (str[i] != ' ' && str[i] != '\t' && str[i] != '\0') {
        guess->direction = str[i];
        ++i;
    }

    if (str[i] == '\0') /* invalid number of parameters */
        return -1;

    if (i - pos > 1) /* more than 1 letter for direction, wrong direction, check
it later */
        guess->row = 'a' - 1; /* assign wrong direction */

    while (str[i] == ' ' || str[i] == '\t') /* ignore blank characters */
        ++i;

    if (str[i] == '\0') /* invalid number of parameters */
        return -1;

    pos = i;
```

```
    /* copy guess */
    while (str[i] != ' ' && str[i] != '\t' && str[i] != '\0') {
        if (i - pos <= WORD_LENGTH) /* do not overflow the buffer */
            guess->word[i - pos] = str[i];
        ++i;
    }
    if (i - pos <= WORD_LENGTH)
        guess->word[i - pos] = '\0'; /* end of the string */
    else
        guess->word[WORD_LENGTH + 1] = '\0'; /* end of the string, the word is
longer that WORD_LENGTH */
    return 0;
}

int checkStartingPosition(char row, int column, Clue * clues, int wordsCnt) {
    int i;

    for (i = 0; i < wordsCnt; ++i)
        if (clues[i].row == row && clues[i].column == column) /* the word at the
position */
            return 0;

    /* the word was not found */
    return -1;
}

int checkDirection(Clue guess, Clue * clues, int wordsCnt) {
    int i;

    for (i = 0; i < wordsCnt; ++i)
        if (clues[i].row == guess.row && clues[i].column == guess.column &&
            clues[i].direction == guess.direction) /* the word at the position
and with the same direction */
            return i;

    return -1;
}

int stringToInt(char * str) {
    int a = 0, i = 0;

    while(str[i] >= '0' && str[i] <= '9') {
        a *= 10;
        a += str[i] - '0';
        ++i;
    }

    if (i == 0 || !(str[i] == ' ' || str[i] == '\t' || str[i] == '\0')) /* input
is not an integer */
        return -1;

    return a;
}
```