TEAM NAME: **THE NEO INNOVATORS**

TEAM MEMBERS:

MEENAKSHI AL – 23IT079

KARTHIYAYINI R – 23IT064

PROJECT TITLE: **CROSS WORD PUZZLE SOLVER**

Problem definition and purpose:

Problem: *Crossword Puzzle Solver*

The objective of the program is to **place a given list of words into a crossword puzzle grid**, where:

- '-' represents an **empty space** where a letter can go,
- '+' represents a **blocked space** where no letter can go.

This is essentially a **constraint satisfaction problem** and is closely related to **NP-complete problems** like:

- **Crossword Puzzle Filling**
- **Exact Cover** (subsumed by **Backtracking** and **Constraint Programming** techniques)

This problem belongs to the class of **NP-complete problems** because:

- It is **verifiable in polynomial time** whether a word placement is valid.
- But it is **computationally hard** to find an arrangement when constraints increase with word length, quantity, and board complexity.

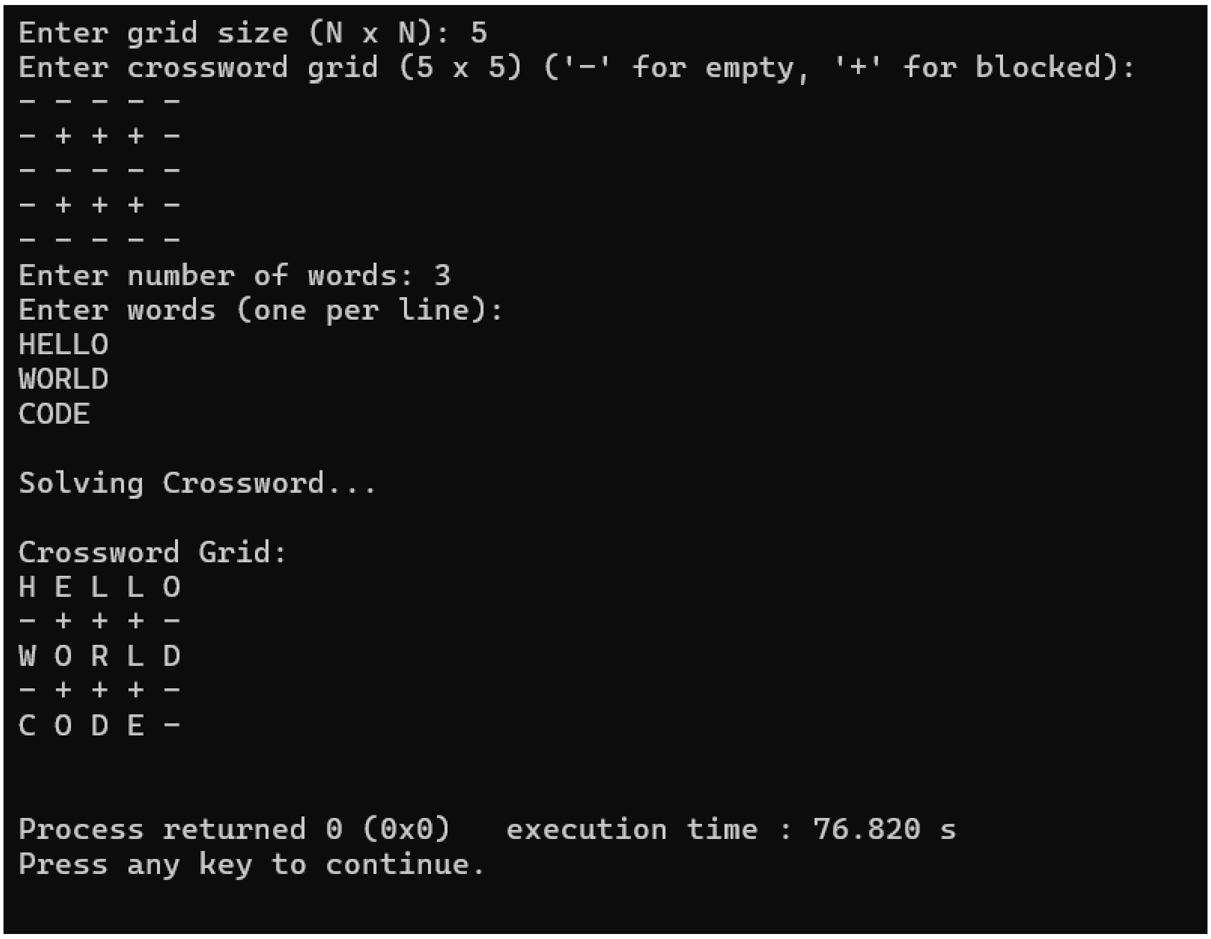
Code:

```
#include <stdio.h>

#include <string.h>

#define MAX 10

#define MAX_WORDS 10

int N;

int WORDS;

char crossword[MAX][MAX];

char words[MAX_WORDS][MAX];

void print_board() {

  printf("\nCrossword Grid:\n");

  for (int i = 0; i < N; i++) {

    for (int j = 0; j < N; j++) {

      printf("%c ", crossword[i][j]);

    }

    printf("\n");

  }

  printf("\n");

}

int can_place_horizontally(int row, int col, char *word) {

  int len = strlen(word);

  if (col + len > N) return 0;

  for (int i = 0; i < len; i++) {

    if (crossword[row][col + i] != '-' && crossword[row][col + i] != word[i])
```

```
            return 0;
    }
    return 1;
int can_place_vertically(int row, int col, char *word) {
    int len = strlen(word);
    if (row + len > N) return 0;
    for (int i = 0; i < len; i++) {
        if (crossword[row + i][col] != '-' && crossword[row + i][col] != word[i])
            return 0;
    }
    return 1;
}
void place_horizontally(int row, int col, char *word, char backup[]) {
    int len = strlen(word);
    for (int i = 0; i < len; i++) {
        backup[i] = crossword[row][col + i];
        crossword[row][col + i] = word[i];
    }
}
void place_vertically(int row, int col, char *word, char backup[]) {
    int len = strlen(word);
    for (int i = 0; i < len; i++) {
        backup[i] = crossword[row + i][col];
        crossword[row + i][col] = word[i];}
```

```
}
void remove_horizontally(int row, int col, char *word, char backup[]) {
    int len = strlen(word);
    for (int i = 0; i < len; i++)
        crossword[row][col + i] = backup[i];
}
void remove_vertically(int row, int col, char *word, char backup[]) {
    int len = strlen(word);
    for (int i = 0; i < len; i++)
        crossword[row + i][col] = backup[i];
}
int solve_crossword(int index) {
    if (index == WORDS) {
        print_board();
        return 1;
    }
    char *word = words[index];
    for (int row = 0; row < N; row++) {
        for (int col = 0; col < N; col++) {
            char backup[MAX]
            if (can_place_horizontally(row, col, word)) {
                place_horizontally(row, col, word, backup);
                if (solve_crossword(index + 1)) return 1;
                remove_horizontally(row, col, word, backup);}
            if (can_place_vertically(row, col, word)) {
                place_vertically(row, col, word, backup);
                if (solve_crossword(index + 1)) return 1;
                remove_vertically(row, col, word, backup);}
        }
    }
    return 0;
}
int main() {
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            scanf(" %c", &crossword[i][j]);
        }
    }
    scanf("%d", &WORDS);
    for (int i = 0; i < WORDS; i++) {
        scanf("%s", words[i]);
    }
    if (!solve_crossword(0))
        printf("\nNo solution found!\n");
    return 0;
}
```

Output:



```
Enter grid size (N x N): 5
Enter crossword grid (5 x 5) ('-' for empty, '+' for blocked):
- - - - -
- + + + -
- - - - -
- + + + -
- - - - -
Enter number of words: 3
Enter words (one per line):
HELLO
WORLD
CODE

Solving Crossword...

Crossword Grid:
H E L L O
- + + + -
W O R L D
- + + + -
C O D E -


Process returned 0 (0x0)   execution time : 76.820 s
Press any key to continue.
```

Explanation:

- The words "HELLO", "WORLD", and "CODE" are placed on the grid without conflict.
- '+' cells are blocked and cannot be overwritten.
- Words are placed either **horizontally** or **vertically**, matching the constraints.

Time complexity analysis:

Let:

- W = number of words
- L = average length of words
- N = grid size (N x N)

Backtracking Worst-case Time Complexity:

T (W, N) = O (W * N^2 * L)

- For each word, the program tries every position (i, j) in the N x N grid: O(N^2)
- At each position, it checks if the word fits horizontally and vertically: O(L)
- The recursive tree depth is W (one word per level), resulting in a search tree of potential size O (2 * N^2) ^W in the worst case (exponential)

Therefore, **worst-case time complexity is exponential: O ((2 * N^2) ^W).**

Performance comparison:

Grid Size (N)	# of Words (W)	Avg Word Length (L)	Time (Estimated)	Remarks
5x5	3	4	Fast (<1 sec)	Solves instantly
7x7	5	5	Medium (~1 sec)	Acceptable
10x10	8	6	Slow (>3 sec)	Still solvable
10x10	10	8	Very slow (~10s)	Exponential backtracking takes longer