

AI, Vibe Coding, and Literate Programming

Writing software in a conversation

May 2025

By Dwayne Phillips

Summary

Vibe coding is the practice of using a chatbot to produce computer programs. Literate programming is an old concept of interweaving descriptions of a program with a programming language's code. Recent chatbots allow a person to produce a computer program via a conversation. The conversation resembles the natural interaction between a user and a computer programmer. The example given below illustrates these ideas and lends credibility to further experiments.

Introduction

Vibe coding is the practice of using a chatbot to produce computer programs. That is a rough definition as rough definitions are about all that is available at this point. The term was created by Andrej Karpathy and has been described in various places including (Siu, 2025).

Vibe coding involves prompting a chatbot with something like, "Write a FORTRAN (pick your language) program that does something for me that I want but I don't want to type in all the syntax of that language."

Out comes the expression of the desire in the specified language. What more could a person want when they know something about computer programming, but don't want to actually write a computer program?

Now try to recall the topic of Literate Programming (Knuth, 1984). Donald Knuth created this concept 40 years ago. A literate program interweaved a description of a computer program with statements in a program's syntax. For various reasons, the concept didn't gain widespread adoption.

Literate programming, however, did lead to things like the Jupyter Notebook. The Jupyter Notebook, and many related programming notebooks, borrowed from the idea of a literate program in that a description was mixed with a program. Interpreted languages such as Python work quite well using the notebook concept.

What follows is an example of using a chatbot. Instead of a single, long prompt describing a computer program, a series of prompts instruct the chatbot to write pieces of a computer program. The conversation is much like literate programming and the result looks like something a Jupyter Notebook would contain.

Cautions

Think! Let's repeat that caution: think!

Read the code produced by the chatbot. Test the code produced by the chatbot. Test, test, and test again. Think before deploying.

The Example

The following is an interaction with Google's Gemini to produce and then modify a simple program. This creates a CRUD (Create, Retrieve, Update, Delete) application for student records. The C programming language is used. This example is not an endorsement of Google's Gemini as there are many chatbots that could do just as well. This is not an endorsement of the C programming language as many other languages would do just as well.

The example consumes several pages of text and is tedious to read. Peruse as necessary to see what is happening. The reader may choose to skip to the Observations section of this essay.

The prompts to the chatbot are in this font.

The code produced by the chatbot is in this font.

This is going to be a simple, console-based program written in the C programming language. List the include statements for the standard library and the other .h files needed for console input and output for strings and characters.

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

Let's define three constants:

1. the maximum number of students is 10
2. The maximum number of characters in a student name is 80

3. The maximum number of grades for each student is 3

```
#define MAX_STUDENTS 10
#define MAX_NAME_LENGTH 80
#define MAX_GRADES 3
```

Let's create a structure that will hold the data for each student. This will include (a) The student's name, (b) a student id number (int), (c) the student's grades.

```
// Define the Student structure
struct Student {
    char name[MAX_NAME_LENGTH + 1]; // +1 for the null terminator
    int id;
    int grades[MAX_GRADES];
};
```

Let's store this information into an array that holds all the information for all the students.

```
int main() {
    // Declare an array of Student structures
    struct Student students[MAX_STUDENTS];

    // You would also typically have a variable to keep track of the actual
    // number of students currently stored, as it might be less than
    MAX_STUDENTS.
    int numStudents = 0;

    return 0;
}
```

The program will display a menu that the user will use to input the information for students and display it and quit. That menu will look like this:

1. Add New Student
2. Display All Students
3. Search Student by ID
4. Exit

Enter your choice:

```
do {
    // Display the menu
    printf("\n--- Student Management System ---\n");
    printf("1. Add New Student\n");
    printf("2. Display All Students\n");
    printf("3. Search Student by ID\n");
    printf("4. Exit\n");
```

```

    printf("Enter your choice: ");

    // Read user's choice
    // Using `scanf` with a space before %d can help consume any leftover
newline character
    if (scanf("%d", &choice) != 1) {
        printf("Invalid input. Please enter a number.\n");
        // Clear the input buffer in case of non-integer input
        while (getchar() != '\n');
        choice = 0; // Set choice to an invalid value to re-show menu
        continue; // Skip to the next iteration of the loop
    }

    // Clear the input buffer after reading the integer
    // This is important because `scanf("%d", ...)` leaves the newline
character
    // in the buffer, which can cause issues with subsequent `fgets` or
`getchar` calls.
    while (getchar() != '\n');

    // Process the choice using a switch statement
    switch (choice) {
        case 1:
            addNewStudent(students, &numStudents);
            break;
        case 2:
            displayAllStudents(students, numStudents);
            break;
        case 3:
            searchStudentByID(students, numStudents);
            break;
        case 4:
            printf("Exiting program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice. Please enter a number between 1 and
4.\n");
            break;
    }
    } while (choice != 4); // Loop continues until user chooses to exit

    return 0;
}

```

To add a new student, the program will allow the user to enter (a) the student's name, (b) the student's ID number, and (c) all the student's grades.

```

// --- Implementation of addNewStudent function ---
void addNewStudent(struct Student students[], int *numStudents) {
    if (*numStudents >= MAX_STUDENTS) {
        printf("\nError: Maximum number of students (%d) reached. Cannot add
more.\n", MAX_STUDENTS);
        return; // Exit the function if no space
    }
}

```

```

// Get a pointer to the current student slot in the array
struct Student *currentStudent = &students[*numStudents];

printf("\n--- Add New Student ---\n");

// 1. Enter Student Name
printf("Enter student name (max %d characters): ", MAX_NAME_LENGTH);
// Use fgets for safer string input (reads up to n-1 chars or newline)
// stdin is the standard input stream
if (fgets(currentStudent->name, MAX_NAME_LENGTH + 1, stdin) == NULL) {
    perror("Error reading student name"); // Print system error message
    return;
}

// Remove the trailing newline character that fgets might include
// Check if a newline exists and replace it with null terminator
currentStudent->name[strcspn(currentStudent->name, "\n")] = '\0';

// 2. Enter Student ID
int temp_id;
printf("Enter student ID (integer): ");
if (scanf("%d", &temp_id) != 1) {
    printf("Invalid input for ID. Please enter an integer.\n");
    while (getchar() != '\n'); // Clear input buffer
    return;
}
currentStudent->id = temp_id;
while (getchar() != '\n'); // Clear input buffer after reading ID

// 3. Enter Student Grades
printf("Enter %d grades for the student:\n", MAX_GRADES);
for (int i = 0; i < MAX_GRADES; i++) {
    int temp_grade;
    printf("Grade %d: ", i + 1);
    if (scanf("%d", &temp_grade) != 1) {
        printf("Invalid input for Grade %d. Please enter an integer.\n",
i + 1);
        while (getchar() != '\n'); // Clear input buffer
        // Optionally, you might want to break the loop or assign a
default grade
        // For now, we'll just return, but in a real app, you might re-
prompt or skip this grade.
        return;
    }
    currentStudent->grades[i] = temp_grade;
    while (getchar() != '\n'); // Clear input buffer after reading each
grade
}

// Increment the counter for the number of students
(*numStudents)++;
printf("Student '%s' added successfully!\n", currentStudent->name);
}

```

To display the students, show each student starting with their name, then their ID, and then their grades.

```
// --- Implementation of displayAllStudents function ---
void displayAllStudents(const struct Student students[], int numStudents) {
    printf("\n--- Display All Students ---\n");

    if (numStudents == 0) {
        printf("No students to display yet. Please add some students
first.\n");
        return;
    }

    // Loop through each student currently stored
    for (int i = 0; i < numStudents; i++) {
        printf("\nStudent %d:\n", i + 1); // Display student number (1-based)
        printf("  Name: %s\n", students[i].name);
        printf("  ID: %d\n", students[i].id);
        printf("  Grades: ");
        // Loop through each grade for the current student
        for (int j = 0; j < MAX_GRADES; j++) {
            printf("%d", students[i].grades[j]);
            if (j < MAX_GRADES - 1) {
                printf(", "); // Add a comma after each grade except the last
one
            }
        }
        printf("\n"); // Newline after grades
    }
    printf("--- End of Student List ---\n");
}
```

Finally, the program will find a student by the student ID and then display the information for that student.

```
// --- Implementation of searchStudentByID function ---
void searchStudentByID(const struct Student students[], int numStudents) {
    printf("\n--- Search Student by ID ---\n");

    if (numStudents == 0) {
        printf("No students to search yet. Please add some students
first.\n");
        return;
    }

    int searchID;
    printf("Enter the Student ID to search for: ");
    if (scanf("%d", &searchID) != 1) {
        printf("Invalid input. Please enter an integer ID.\n");
        while (getchar() != '\n'); // Clear input buffer
        return;
    }
    while (getchar() != '\n'); // Clear input buffer after reading ID
```

```

int found = 0; // Flag to indicate if the student was found
for (int i = 0; i < numStudents; i++) {
    if (students[i].id == searchID) {
        printf("\n--- Student Found! ---\n");
        printf("  Name: %s\n", students[i].name);
        printf("  ID: %d\n", students[i].id);
        printf("  Grades: ");
        for (int j = 0; j < MAX_GRADES; j++) {
            printf("%d", students[i].grades[j]);
            if (j < MAX_GRADES - 1) {
                printf(", ");
            }
        }
        printf("\n");
        found = 1; // Set flag to true
        break;    // Exit the loop once the student is found
    }
}

if (!found) { // If the loop finished and 'found' is still 0
    printf("Student with ID %d not found.\n", searchID);
}
}

```

Modify the Program

Another twist to the example is having the chatbot modify the “finished” program. This illustrates how a person can change their mind, describe the changes, and have the chatbot change the program.

Modification: Add an average grade to the student information. This is the average of the grades entered for that student.

The chatbot replied:

Let's add the average grade calculation to the Student structure and update the relevant functions to include this new piece of information.

Here's how we'll modify the code:

Modify struct Student: Add a float averageGrade; member.

Modify addNewStudent: Calculate the average grade after all individual grades are entered and store it in the averageGrade member.

Modify displayAllStudents: Display the averageGrade for each student.

Modify searchStudentByID: Display the averageGrade for the found student.

All the modifications were made correctly.

Observations

1. It is possible to “converse” with a chatbot to build a computer program piece by piece.
2. This piecemeal construction is much easier and more natural than specifying a program in one prompt.
3. The pieces of the program don’t have to be specified in order.
4. The program can later be modified with further conversation.
5. This method of specifying and producing a program is much more like the conversation a user would have with a programmer.

The above observations come after only a few hours of experimentation. Further experiments and refinements seem to justify the resource required.

References

Knuth, D. E. (1984, May 29). Donald E. Knuth - Literate Programming. Literate Programming. <http://www.literateprogramming.com/knuthweb.pdf>

Siu, E. (2025, May 7), I took a 2-day ‘vibe coding’ class and successfully built a product. Here are my biggest takeaways. CNBC. <https://www.cnbc.com/2025/05/08/i-took-a-2-day-vibe-coding-class-and-successfully-built-a-product.html>