Concepts of Programming - Project #1: Parser

Daniel Marin and Jennifer Vicentes

October 22^{nd}, 2024

Contents

1 Introduction

1.1 Context

For the 2^{nd} deliverable of this project we where expected to create a parser that does not compute the notes that make up the chords. For the elaboration of this section of the project we used the following grammar and sets to help us guide us when coding this parser:

1.1.1 Grammar Rules

```
      input:=  song EOF
       song:=  bar {bar} "|"
        bar:=  [meter] chords "|"
      meter:=  numerator "/" denominator
  numerator:=  "1" | "2" | "3" | ... | "15"
denominator:=  "1" | "2" | "4" | "8" | "16"
     chords:=  "NC" | "%" | chord {chord}
      chord:=  root [description] [bass]
       root:=  note
       note:=  letter [acc]
     letter:=  "A" | "B" | "C" | ... | "G" |
description:=  qual | qual qnum | qnum | qnum sus | sus
       qual:=  "-" | "+" | "o"
       qnum:=  ["^"] num
        num:=  "7" | "9" | "11" | "13"
        sus:=  "sus2" | "sus4"
       bass:=  "/" note
        acc:=  "b" | "#"
```

Figure 1: Chords Grammar. The input consists of the chords in a song separated by bar lines "|" plus optional meter information for the bars. (This grammar is a subset of the grammar used by Polynizer: `https://www.polynizer.com`)

All grammar related functions in 'Parser.c' are an interpretation of the grammar rules shown in figure 1. With the use of the concepts seen in class and in the book. Also, as mentioned before here we have the first and follow sets of this grammar from deliverable one of this project.

1.1.2 First and Follow Sets of Grammar

Nonterminal	First set	Follow set
input	{1, 2, 3, 4, 5, . . . , 15, %, NC, A, B, C, D, E, F, G}	{}
song	{1, 2, 3, 4, 5, . . . , 15, %, NC, A, B, C, D, E, F, G}	{EOF}
bar	{1, 2, 3, 4, 5, . . . , 15, %, NC, A, B, C, D, E, F, G}	{ |, 1, 2, 3, 4, 5, . . . , 15}
meter	{1, 2, 3, 4, 5, . . . , 15}	{%, NC, A, B, C, D, E, F, G}
numerator	{1, 2, 3, 4, 5, . . . , 15}	{ /}
denominator	{1, 2, 4, 8, 16}	{%, NC, A, B, C, D, E, F, G}
chords	{%, NC, A, B, C, D, E, F, G}	{ | }
chord	{A, B, C, D, E, F, G}	{ |, A, B, C, D, E, F, G}
root	{A, B, C, D, E, F, G}	{-, +, o, ^, 7, 9, 11, 13, sus2, sus4, /, |, A, B, C, D, E, F, G}
note	{A, B, C, D, E, F, G}	{-, +, o, ^, 7, 9, 11, 13, sus2, sus4, /, |, A, B, C, D, E, F, G}
letter	{A, B, C, D, E, F, G}	{#, b, -, +, o, ^, 7, 9, 11, 13, sus2, sus4, /, |, A, B, C, D, E, F, G}
description	{-, +, o, ^, 7, 9, 11, 13, sus2, sus4}	{/, |, A, B, C, D, E, F, G}
qual	{-, +, o}	{/, |, A, B, C, D, E, F, G,^, 7, 9, 11, 13}
qnum	{^, 7, 9, 11, 13}	{/, |, A, B, C, D, E, F, G, sus2, sus4}
num	{7, 9, 11, 13}	{/, |, A, B, C, D, E, F, G, sus2, sus4}
sus	{sus2, sus4}	{/, |, A, B, C, D, E, F, G}
bass	{/}	{ |, A, B, C, D, E, F, G}
acc	{#, b}	{-, +, o, ^, 7, 9, 11, 13, sus2, sus4, /, |, A, B, C, D, E, F, G}

Figure 2: First and Follow Sets of the Grammar in figure 1 developed in deliverable 1.

Figure 1 and 2 helped us develop the flow of the program and it's elaboration in a concise but proper manner. This report document focuses on explaining each function, the tests that where performed to prove functionality.

1.2 Usage

The usage of this program is fairly easy. To use this program as of the current state of this folder you must first add a song as a `.txt` file to the **Songs** folder. Run the program, when prompted "`Enter the path of the file to be parsed:`" copy the file path.

1.3 Platform

The development of this project was all done in C with the help of the grammar rules and Chapter 6.6 - Parsing Techniques and Tools from Programming Languages - Kenneth C. Louden - 3ed.

2 Functions

2.1 Main

The main function is fairly straightforward it is in charge of setting up the parser by following the next steps:

- Getting the filepath from the user, preferrebly one that uses the Songs folder.
- Determining if the type of file is supported and if it can be opened, if the file is a '.txt', proceed to parse, if not exit program.
- Parse the contents of the '.txt', if the contents are correct according to the grammar it should print them without the spaces.

To better comprehend this behavior, following is the code of the main function.

```
int main(void){
    char filepath[365];
    printf("Enter the path of the file to be parsed: ");
    scanf("%[^\n]s", filepath);
    if (strlen(filepath) < 4 ||
        strcmp(filepath + strlen(filepath) - 4, ".txt") != 0) {
        printf("Error: Only .txt files are supported\n");
        exit(1);
    }
    inputFile = fopen(filepath, "r");
    if (inputFile == NULL){
        printf("Error: cannot open file\n");
        exit(1);
    }
    printf("The following characters demonstrate the tokens being parsed.\n\n");
    getToken();
    input();
    printf("\n");
    printf("\nParsing completed successfully\n");
    fclose(inputFile);
    return 0;
}
```

Figure 3: Uncommented `main` function from Parser.c

2.2 Parsing Functions

This section contains the functions that aren't grammar dependant and can be used whenever needed for parsing a '.txt' file.

2.2.1 `void error(char *message)`

This function is at simple as it gets, whenever called it is supposed to print out a message for the user to see whenever there is an error, and exit the program. In this program it is primarily used to catch parsing errors and prevent them from propagating later on.

```
void error(char* message) {
    printf("\nParse error: %s\n", message);
    exit(1);
}
```

Figure 4: Uncommented `error` function from Parser.c

2.2.2 `void getToken()`

This function is in charge of retrieving the next token to be parsed whenever it is called, ignoring newlines, tabulars and spaces. Thus retrieving possibly parsable tokens from the file.

```
void getToken(){
    token = getc(inputFile);
    if (token == EOF) return;
    while(token == ' ' || token == '\n' || token == '\t'){
        token = getc(inputFile);
    }
}
```

Figure 5: Uncommented `getToken` function from Parser.c

2.2.3 `void match(char c, char* message)`

This function is simple, it is a controller that is in charge of deciding whether the current token is the desired character and we should prompt the `getToken()` function, or if there was an error parsing and it should prompt an `error(message)`.

```
void match(char c, char* message){
    if (token == c)
        getToken();
    else
        error(message);
}
```

Figure 6: Uncommented `match` function from Parser.c

2.3 Grammar Functions

This section contains all the grammar dependant functions used in the development of this parser. They are what trully parse the contents of the document with the help of the functions in the Parsing Functions Section.

- **Rule 1:** `void input()` - is in charge of initializing the parsing of the song and matching the EOF character. Basically, starting and ending the the parsing.
- **Rule 2:** `void song()` - is in charge of parsing the structure of the song, by parsing bars until it encounters a '—' after any bar, insinuating that the song has been completely parsed.
- **Rule 3:** `void bar()` - is in charge of parsing the structure of a bar, checking for the optional meter by asking if the token is a digit. The parsing the chords, and finally matching the '—' character.
- **Rule 4:** `void meter()` - is in charge of parsing the structure of a meter, first retrieving a numerator, matching the '/' and finally retrieving the denominator.
- **Rule 5:** `int numerator()` - is in charge of checking if the numerator is in the valid range (from 1 - 15), and then returning if valid. Else, it should prompt an error.
- **Rule 6:** `int denominator()` - is in charge of checking if the denominator is a valid one (denominator $\in$ [1,2,4,8,16]), and returning if so. Else, it should prompt an error.
- **Rule 7:** `void chords()` - is in charge of parsing the structure of a set of chords by either checking for the "`NC`" or the "`%`" token, or checking for repeated set of chord.
- **Rule 8:** `void chord()` - is in charge of parsing the structure of a chord by calling `root` and checking if the chord contains a description and a bass.

- **Rule 9:** `void root()` - is in charge of calling note and is placed here since it will help us represent the notes in deliverable 3, by separating a root from a bass.
- **Rule 10:** `void note()` - is in charge of parsing the structure of a note by calling `letter` function and then checking if it should call the `acc` function.
- **Rule 11:** `char letter()` - is in charge of checking if the letter is a valid one ([A-G]), and returning it for the note to parse.
- **Rule 12:** `char acc()` - is in charge of checking if the accent is a valid one (either # or b).
- **Rule 13:** `void description()` - is in charge of checking which of the optional description has by using a set of dedicated combinations seen in this rule. Anything else, should prompt an error.
- **Rule 14:** `char qual()` - is in charge of checking if the quality of the chord is the correct one (either -,+,o).
- **Rule 15:** `void qnum()` - is in charge of checking for valid combinations of a qnum and a num (options are a ^ symbol and a num or a num).
- **Rule 16:** `int num()` - is in charge of checking using the same method as the numerator for a valid number $\in$ [7,9,11,13]. This returns it also or prompts an error.
- **Rule 17:** `void sus()` - is in charge of checking if the suspension is either *sus*2 or *sus*4.
- **Rule 18:** `void bass()` - is in charge of parsing the structure of a bass by matching the '/' and prompting the `note` function.

Some of these functions have added functionality to print certain results for us as the user of the program to actually visualize the characters being parsed and the order of execution.

3 Test and Results

The tests for the program are all the '.txt' files posted by Dr. Arturo Camacho on blackboard, they are songs parsed using Polynizer app[1] which are just a set of terminals/tokens bundled in a neatly organized manner, that when played create a song. Those tests are encompassed by the following itemize, which are the files in the Songs folder and their respective results.

- **Test 1**: Bruno Mars, Anderson Paak, Silk Sonic - Skate.txt

```
Enter the path of the file to be parsed: C:\Users\Usuario\Desktop\Git Repositories\Concepts_Project\Songs\Bruno Mars, Anderson Paak, Silk Sonic - Skate.txt
The following characters demonstrate the tokens being parsed.

4/4A-7|%|Bb^7/C|%|A-7|%|Bb/C|C/D|G-7C|A-7D7|G-7C|A-7D|G-7C|A-7D7|A-7Bb^7C/D|C/D|Bb^7A-7G-7|G-7|A-7Bb^7|Bb^7|A-7G-9|Db/Eb|Ab^7|G-7A7sus4Csus4|G-7C|A-7D7|G-9C|A
-7D|G-7C|A-7D|A-7Bb^7C/D|C/D|A-7|Bb^7|Bb/C|C/D|G-7C|A-7D|G-7C|A-7D|G-7C7|A-7C/D|A-7Bb^7C/D|C/D|Bb^7A-7G-7|G-7|A-7Bb^7|Bb^7|A-7G-7|Db/Eb|Ab^9|G-11A7sus4Csus4|G
-9C|A-7D7|G-9C|A-7D|G-9C|A-7D7|A-7Bb^7C/D|C/D|Db^7/Eb|%|Ab^9|Bb-7Cb^7|Cb^7/Db|%|Gb^7|Bb^7/C|A-7|%|Bb^7/C|%|A-7|%|Bb^7/C|C/D|G-11C7|A-7D7|G-9C|A-7D7|G-9C7|A-7D
7|A-7Bb^7C/D|C/D|G-11C7|A-11D7|G-9C|A-7D7|G-9C|A-7D|A-7Bb^9C/D|C/D||

Parsing completed successfully
```

Figure 7: Image of outputs on the command line.

- **Test 2**: John Legend - All of me.txt

```
Enter the path of the file to be parsed: C:\Users\Usuario\Desktop\Git Repositories\Concepts_Project\Songs\John Legend - All of me.txt
The following characters demonstrate the tokens being parsed.

4/4F-|Db|Ab|Eb|F-|Db|Ab|Eb|F-|Db^7|Ab|Eb|F-|Db^7|Ab|Eb|F-|Db^7|Ab|Eb|F-|Db^9|Ab|Ebsus2|Bb-|%|Ab|Eb|Bb-|%|Ab|Eb|Ab|%|Fsus4|F-|Bb-/Db|%|Eb9sus4|Eb|Ab|%|F-|%|Bb-/Db|%|Eb
9sus4|Eb|F-|Db|Ab|Eb|F-|Db|Ab|Eb|F-|Db^7|Ab|Eb|F-|Db^9|Ab|Eb|F-9|Db^7|Ab|Eb|F-|Db^9|Ab|Eb|Bb-|%|Ab|Eb|Bb-|%|Ab|Ebsus2|Ab|%|Fsus4|F-|Bb-/Db|%|Eb9sus4|Eb|Ab|%|F-|%|Bb-/
Db|%|Ebsus4|Eb|F-|Db|Ab|Eb|F-|Db|Ab|Eb7sus2|Bb-|%|Ab|Ebsus2|Bb-|%|Ab|Eb|Ab|%|F-|%|Bb-/Db|%|Ebsus2|%|Ab|%|F-|%|Bb-/Db|%|Eb9sus4|Eb|F-9|Db|Ab|Eb|F-9|Db|Ab|Eb|F-|Db|Ab|E
b|F-|Db|Ab|Ebsus2||

Parsing completed successfully
```

Figure 8: Image of outputs on the command line.

[1] Polynizer app link: `https://www.polynizer.com`

- **Test 3**: Luis Miguel - El dia que me quieras.txt

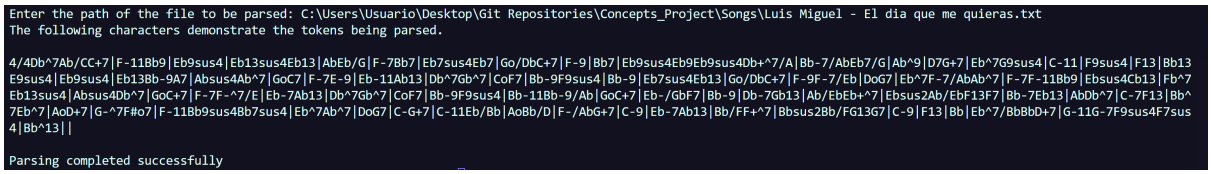

Figure 9: Image of outputs on the command line.

- **Test 4**: Paul McCartney - Uncle Albert / Admiral Halsey (medley).txt

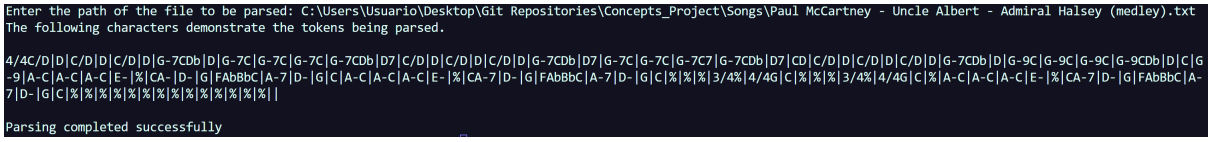

Figure 10: Image of outputs on the command line.

- **Test 5**: Queen - Don't stop me now.txt

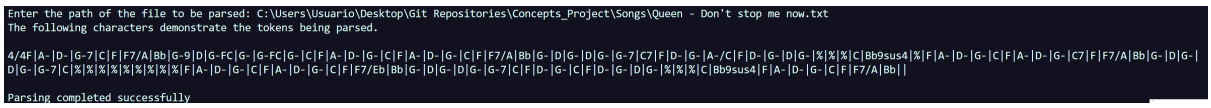

Figure 11: Image of outputs on the command line.

- **Test 6**: Rocio Durcal - La gata bajo la lluvia.txt

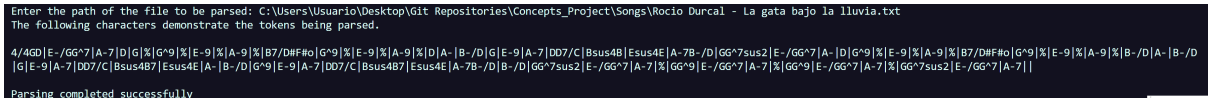

Figure 12: Image of outputs on the command line.

- **Test 7**: Soda Stereo - The musica ligera.txt

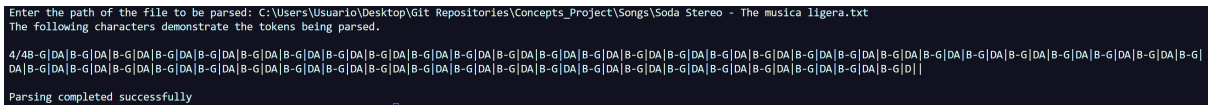

Figure 13: Image of outputs on the command line.