Concepts of Programming - Project #1: Chord Calculator

Daniel Marin and Jennifer Vicentes

October 22^{nd}, 2024

Contents

1 Introduction **2**
1.1 Context . . . 2
1.1.1 Grammar Rules . . . 2
1.1.2 First and Follow Sets of Grammar . . . 3
1.2 Usage . . . 3
1.3 Platform . . . 4

2 Data Structures **4**

3 Functions **4**
3.1 Main . . . 4
3.2 Token Validation . . . 5
3.2.1 `void error(char *message)` . . . 5
3.2.2 `void getToken()` . . . 5
3.2.3 `void match(char c, char* message)` . . . 6
3.3 Grammar Functions . . . 6
3.3.1 `void input()` . . . 6
3.3.2 `void song(ChordNode** head)` . . . 6
3.3.3 `void bar(ChordNode** head)` . . . 6
3.3.4 `void meter()` . . . 6
3.3.5 `int numerator()` . . . 6
3.3.6 `int denominator()` . . . 7
3.3.7 `void chords(ChordNode** head)` . . . 7
3.3.8 `void chord(Chord* chord)` . . . 7
3.3.9 `void root(char* root)` . . . 7
3.3.10 `void note(char* res)` . . . 7
3.3.11 `char letter()` . . . 7
3.3.12 `char acc()` . . . 7
3.3.13 `void description(Chord* chord)` . . . 7
3.3.14 `char qual()` . . . 7
3.3.15 `void qnum(Chord* chord)` . . . 7
3.3.16 `int num()` . . . 7
3.3.17 `void sus(Chord* chord)` . . . 8
3.3.18 `void bass(char* bass)` . . . 8
3.4 Chord Calculator Functions . . . 8
3.4.1 `ChordNode* createNode(Chord chord)` . . . 8
3.4.2 `ChordNode* appendChord(ChordNode* head, Chord chord)` . . . 8
3.4.3 `void printChords(ChordNode* head)` . . . 8
3.4.4 `void createChordArr(Chord* chord, int* arr)` . . . 8
3.4.5 `int noteToPitchClass(const char* root)` . . . 8
3.4.6 `void printChord(Chord* chord)` . . . 8
3.4.7 `void freeChords(ChordNode* head)` . . . 8

4 Test and Results **8**

5 Discussion **11**

1 Introduction

The purpose of this project was to put in practice de concepts seen in class, by creating a fully fledged chord calculator based on the subject of parsing primarily and the concepts of programming as a whole.

1.1 Context

For the 3^{rd} deliverable of this project we where expected to create a chord calculator which is basically a pitch class description of each chord in a song. For the elaboration of this deliverable we used the code from the 2^{nd} deliverable and modified it so as to be able to calculate the pitch class of each chord. As usual we used the grammar rules and the follow sets as the base for the creation of this project.

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 'ChordCalc.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 program developed for this deliverable is named 'ChordCalc.c', to use this program you must first have the '.txt' file in a folder (preferrebly the songs folder). Run the program, when prompeted "`Enter the path of the file to be parsed:`" copy the file path. The outputs would appear as follows:

```
4/4 Db^7 Ab/C C+7 | F-11 Bb9 | Eb9sus4 | Eb13sus4 Eb13 ||
```

Figure 3: Sample input '.txt' contents consisting of the introductory chords of *El dia que me quieras* by Carlos Gardel and Alfredo Le Pera (version by Luis Miguel)

```
    0  1  2  3  4  5  6  7  8  9  A  B
    -  -  -  -  -  -  -  -  -  -  -  -
1.  *  *           *        *              - Db^7
2.  *        *              *              - Ab/C
3.  *           *           *     *        - C+7
4.  *        *     *        *     *        - F-11
5.  *     *        *        *     *        - Bb9
6.     *     *     *        *     *        - Eb9sus4
7.  *  *     *              *     *        - Eb13sus4
8.  *  *     *           *        *        - Eb13
    -  -  -  -  -  -  -  -  -  -  -  -
    7  4  1  5  1  4  0  1  7  0  6  0
```

Figure 4: Format for the output. Showing the pitch class number as the header of a table and marking with stars the notes that make up the chords, where A = 10 and B = 11. At the end of document we have the totals.

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 Data Structures

Two data stuctures where used to develop this program, the `Chord` which repersents the values of individual Chords being parsed. With these two data structures we managed to create the logic of the Chord Calculator accordingly. The chord data structure is of the format:

```
typedef struct {
    char root[3];       // D, Db or D#
    char qual;          // -, +, o, none
    int sus;            // sus2 or sus4
    char ext_type;      // ^ or none
    int extension;      // 7, 9, 11, 13
    char bass[3];       // D, Cb or C
} Chord;
```

Also, we utilized the linked list data structure to dynamically hold a list of chords that gets created as we parse the current .txt file, which is of form:

```
typedef struct ChordNode {
    Chord chord;
    struct ChordNode* next;
} ChordNode;
```

3 Functions

3.1 Main

The main function is in charge of the logic behind coordinating the functions to parsing and printing, and the file retrival. The code behind this function is:

```
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 5: Uncommented `main` function in charge of coordinating the parsing, printing and file retrieving for this program.

As demonstrated in the section of code from above, this function follows a set of steps to properly coordinate the behavior of the Chord Calculator:

- Getting the filepath from the user.
- Checking if the file is supported and if it can be opened.
- Parse the contents of the program.

3.2 Token Validation

The functions in this section are in charge of retrieving tokens, checking if the token is what is expected, and prompting error messages when needed.

3.2.1 `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 parsing, 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 6: Uncommented `error` function from Parser.c

3.2.2 `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. This function is based on what was seen in class.

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

Figure 7: Uncommented `getToken` function from Parser.c

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

This function is a controller that is in charge of deciding whether the current token is the desired character and we should prompt the `getToken()` function to continue parsing, 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 8: Uncommented `match` function from Parser.c

3.3 Grammar Functions

The functions in this section are based on the grammar rules seen in figure 1, which define the rules in which the parsing should take place. They are fairly similar to the functions from deliverable 2, but with the added functionality that when parsing they create a list of chords.

3.3.1 `void input()`

This function is in charge of initializing the parsing and the list of chords (linked list), calling the match, finalize the parsing by matching the `EOF` character at the end of the file. Once the parsing takes place, this function prints out the Chords in the format stated by figure 4 before freeing the list of chords.

3.3.2 `void song(ChordNode** head)`

This function 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 ended, with the added functionality of passing the pointer to bar as to record the chords being played.

3.3.3 `void bar(ChordNode** head)`

This function is in charge of parsing the structure of a bar, checking for the optional meter by checking if the current token is a digit, calling the chords method, and finally matching the end of a bar with the `|` char.

3.3.4 `void meter()`

This function is in charge of parsing the structure of a meter, first retrieving a numerator, matching the '/' and finally retrieving the denominator.

3.3.5 `int numerator()`

This function 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.

3.3.6 int denominator()

This function 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.

3.3.7 void chords(ChordNode** head)

This function is in charge of parsing the structure of a set of chords by either matching the characters `NC` or `%`, or checking for a repeated set of chords. It has the added functionality of passing the head of the list to the chord function for it to register the current chord it is parsing.

3.3.8 void chord(Chord* chord)

This function is in charge of parsing the structure of a chord by retrieving the root, checking if the chord has a description and a bass and parsing the chord accordingly. It also, appends each chord to the list of chords whenever parsing of the chord has been completed.

3.3.9 void root(char* root)

This function is charge of calling note and is used to separate the logic of the root from the bass which represent different parts of the same chord.

3.3.10 void note(char* res)

This function is in charge of parsing the structure of a note by calling `letter` function and then checking if it should call the `acc` function, it then places those results in the correct position of the chord data structure.

3.3.11 char letter()

This function is in charge of checking if the letter is in the range from A to G, and returing it for the note to store it if it is in the range.

3.3.12 char acc()

This function is in charge of checking if the accent is a valid one (either # or b), and returning it to the note to store.

3.3.13 void description(Chord* chord)

This function is in charge of checking which of the optional description rules is being used by the chord, by using a dedicated set of valid combinations seen in this rule, and then storing the information of the chord.

3.3.14 char qual()

This function is in charge of checking if the quality of the chord is a valid one, and returning which quality it is to the description.

3.3.15 void qnum(Chord* chord)

This function is in charge of checking for valid combinations of a qnum and a num (options are a ^ symbol and a num or a num), and saving this information accordingly.

3.3.16 int num()

This function 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.

3.3.17 `void sus(Chord* chord)`

This function is in charge of is in charge of checking if the suspension is either *sus*2 or *sus*4, and saving it in the chord.

3.3.18 `void bass(char* bass)`

This function is in charge is in charge of parsing the structure of a bass by matching the '/' and prompting the `note` function, and storing the results of the note into the bass section of the chord.

3.4 Chord Calculator Functions

3.4.1 `ChordNode* createNode(Chord chord)`

This function allocates memory for a new `ChordNode` containing the provided chord data. It initializes the node with the chord data and sets its `next` pointer to `NULL`. If memory allocation fails, an error message is displayed, and the program exits.

3.4.2 `ChordNode* appendChord(ChordNode* head, Chord chord)`

This function appends a new chord to the end of the linked list of chords. If the list is empty, the new node becomes the head. Otherwise, it traverses to the end of the list and links the new node there to maintain the order of printing.

3.4.3 `void printChords(ChordNode* head)`

This function traverses the linked list, printing each chord's pitch class and description. It also maintains a tally of pitch class occurrences for each chord and displays a summary of counts after listing all chords.

3.4.4 `void createChordArr(Chord* chord, int* arr)`

This function creates a pitch class array for a chord by marking notes based on the chord's root, quality, extension, and bass. The array is updated based on the chord's characteristics to represent pitch classes on a 12-tone scale.

3.4.5 `int noteToPitchClass(const char* root)`

This function converts a root note into its corresponding pitch class (0-11). It accounts for accidentals (`b` and `#`) and returns -1 if the note is invalid.

3.4.6 `void printChord(Chord* chord)`

This function prints the chord's root, quality, extension, suspension, and bass. It formats the chord's attributes into a readable representation, including any extensions, suspensions, or bass notes.

3.4.7 `void freeChords(ChordNode* head)`

This function deallocates memory for each node in the chord linked list, preventing memory leaks by iterating through the list and freeing each node individually.

4 Test and Results

The test for the program are all the '.txt' files posted by Dr. Arturo Camacho on blackboard and the sample input .txt, 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. The format of these results are as follows, only the last few chords and the total times each pitch class is played will be in this document.

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

```
      0       1       2       3       4       5       6       7       8       9       A       B
      -       -       -       -       -       -       -       -       -       -       -       -
1.    *       *                               *                       *                               - Db^7
2.    *                               *                               *                               - Ab/C
3.    *                                       *                       *               *               - C+7
4.    *                               *               *               *               *               - F-11
5.    *               *                               *               *               *               - Bb9
6.            *               *               *                       *               *               - Eb9sus4
7.    *       *               *                                       *               *               - Eb13sus4
8.    *       *               *                               *                       *               - Eb13
      -       -       -       -       -       -       -       -       -       -       -       -
       7       4       1       5       1       4       0       1       7       0       6       0
```

Figure 9: Test and Results Format

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

```
135.    *                               *                       *                                       - C
136.    *                               *                       *               *                       - A-7
137.    *               *                               *       *               *                       - D7
138.            *               *               *               *               *       *               - G-9
139.    *                               *                       *                                       - C
140.    *                               *                       *               *                       - A-7
141.            *                               *               *               *                       - D
142.    *                               *                       *               *                       - A-7
143.    *               *                               *                       *       *               - Bb^9
144.    *               *               *                       *                                       - C/D
145.    *               *               *                       *                                       - C/D
        -       -       -       -       -       -       -       -       -       -       -       -
       95       6      81       8      71      52      20     102       7      80      55       2
```

Figure 10: Image of outputs on the command line.

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

```
111.            *                               *                       *                               - Db
112.    *                       *                                       *                               - Ab
113.                            *                               *                       *               - Eb
114.    *                                       *                       *                               - F-
115.            *                               *                       *                               - Db
116.    *                       *                                       *                               - Ab
117.                            *                               *                       *               - Eb
118.    *                                       *                       *                               - F-
119.            *                               *                       *                               - Db
120.    *                       *                                       *                               - Ab
121.                            *               *                                       *               - Ebsus2
        -       -       -       -       -       -       -       -       -       -       -       -
       64      35       0      71       0      66       0      27      77       0      49       0
```

Figure 11: Image of outputs on the command line.

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

```
137.  *           *     *                       *                 *           - C-9
138.  *           *     *           *                       *                 - F13
139.              *           *                                   *           - Bb
140.              *     *                       *                 *           - Eb^7/Bb
141.              *           *                                   *           - Bb
142.  *           *                       *                       *           - D+7
143.  *           *           *           *                       *           - G-11
144.              *           *           *                       *           - G-7
145.  *                 *     *           *                       *           - F9sus4
146.  *                 *     *                                   *           - F7sus4
147.              *           *           *           *           *           - Bb^13
      -     -     -     -     -     -     -     -     -     -     -     -
      82    46    39    86    15    82    20    52    68    20    85    13
```

Figure 12: Image of outputs on the command line.

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

```
115.  *                       *                 *           *                 - A-7
116.              *                 *                       *                 - D-
117.              *                             *                       *     - G
118.  *                             *                       *                 - F
119.  *                 *                             *                       - Ab
120.              *                 *                             *           - Bb
121.  *                       *                 *                             - C
122.  *                       *                 *           *                 - A-7
123.              *                 *                       *                 - D-
124.              *                             *                       *     - G
125.  *                       *                 *                             - C
      -     -     -     -     -     -     -     -     -     -     -     -
      70    6     58    3     64    34    16    78    9     45    20    11
```

Figure 13: Image of outputs on the command line.

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

```
 99.              *                       *                 *           - G-
100.  *                       *           *                             - C
101.  *                 *           *                 *           *     - Bb9sus4
102.  *                             *                       *           - F
103.  *                       *                             *           - A-
104.              *                 *                       *           - D-
105.              *                             *                 *     - G-
106.  *                       *                 *                       - C
107.  *                             *                       *           - F
108.  *                 *           *                       *           - F7/A
109.              *                 *                             *     - Bb
      -     -     -     -     -     -     -     -     -     -     -     -
      52    0     57    7     26    48    9     48    2     55    40    0
```

Figure 14: Image of outputs on the command line.

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

```
79.   *                       *                   *               *                       - A-7
80.                   *                           *                               *       - G
81.                   *                   *       *               *               *       - G^9
82.                           *                   *                               *       - E-/G
83.                   *                   *       *                               *       - G^7
84.   *                       *                   *               *                       - A-7
85.                   *                           *                               *       - G
86.                   *                   *       *               *                       - G^7sus2
87.                           *                   *                               *       - E-/G
88.                   *                   *       *                               *       - G^7
89.   *                       *                   *               *                       - A-7
      -       -       -       -       -       -       -       -       -       -       -       -
      23      0       49      5       40      0       50      52      3       47      0       60
```

Figure 15: Image of outputs on the command line.

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

```
185.                  *                               *                                       *       - B-
186.                  *                                       *                               *       - G
187.                  *                               *                       *                       - D
188.          *                       *                                       *                       - A
189.                  *                               *                                       *       - B-
190.                  *                                       *                               *       - G
191.                  *                               *                       *                       - D
192.          *                       *                                       *                       - A
193.                  *                               *                                       *       - B-
194.                  *                                       *                               *       - G
195.                  *                               *                       *                       - D
      -       -       -       -       -       -       -       -       -       -       -       -
      0       48      147     0       48      0       98      49      0       97      0       98
```

Figure 16: Image of outputs on the command line.

5 Discussion

This deliverable manages to encompass the desire behavior of the chord calculator. It manages to do so with the use of the functions seen in section 3, with proven results in Section 4 with the use of the examples sent by the professor. The examples are in the folder **Songs** appended with this document. For a more detailed analysis on the functions used to develop the overall functionality please revise the **ChordCalc.c** file found in the same folder as this document.