University of Technology Jamaica

Design of Programming Language

Group Project

DPLMusicCompileMagic

Team:

Julia Pottinger (1200639)

Tamoya Beckford(1203170)

Christopher Fogo(1202890)

Carlington Palmer(1201861)

Overview

***DPLMusicCompileMagic*** was created using JAVA.

The Java application is compatible with any system/platform so there is no doubt that users will have access to the application regardless of their device or OS. Alongside that, JAVA is a highly standardized language. It supports the three factors of standardization as it is: mature and it has been around for many years, constantly evolving to improve and update its constructs, consistency in versions eg. from Java 7 to now presently Java 8. It is also a good programming language as it easy to code due to its simple grammar and syntax, supports interoperability and is highly scalable.

JAVA supports great multi-threading technologies that would facilitate parallelism in this application to support the client-server architecture. Therefore, the compiler will be able handle more request at the same time to send to the application for processing.

Like CLANG, ***DPLMusicCompileMagic*** has its front-end and back-end. This is simulated using the client-server structure. The phases from Preprocessing to Semantic Analysis are operating on the client side of the application. The intermediate code and the code generation are running on the server side.

This compiler was formulated using various rules. It accepts a text file and this text file should consist of:

* [Chorus], [Verse] must be written in square brackets
* Verses should be clearly identified by [Verse]
* Chorus should be clearly identified by [Chorus]
* The category of the song must be in the first line of the text file. Accepted categories are Gospel and RnB.
* The title of the song is the second line of the text file and should be in double quotes.
* The word *repeat* should be in square brackets. This indicates that a line should be repeated once.
* [NUM x] or [x NUM] indicates that a line should be repeated NUM times.
* Whitespaces between words are allowed.
* New lines are accepted whitespaces.
* Ad-libs are bot accepted in songs.
* Start of each sentence must be capitalized.
* Each sentence should end with one of the following:.,!?
* Words that should be capitalized are: God, Faith, Lord, Saviour

**Preprocessing:**

* All lines that should be repeated are repeated
* Removes hyphen between text and join hyphenated text
* Checks the category of song
* If it is an accepted category is stores the song to the respected file i.e *Gospel* or *RnB*
* All other file are stored in an *Unknown* category

**Symbol Table:**

* Stores how many lines and words are in the song
* Stores the category of song
* Stores the title of song
* Stores the structure of song
* Captures the verses and chorus

**Lexical Analysis:**

* Tokens are: WORD, TITLE, CATEGORY, VERSE, SENTENCE, EXPRESSION

**Syntax Analysis:**

* Checks that each sentence start with a capital letter and ends with a delimiter.
* Checks that the category of the song is either Gospel or RnB.
* Checks if the title of the song is in double quotes.
* Check if keywords are capitalized i.e God, Lord, Faith, Saviour.
* Checks for unnecessary brackets.
* Checks if Chorus and Verse are enclosed by square brackets.
* Check if Chorus and Verse are enclosed by other brackets apart from square brackets.

**Semantic Analysis:**

* Conducts a spellcheck on the words.
* Numbers are not accepted
* Check if category is in line 1.
* Check if title is in line 2.

**Intermediated Code:**

* Converts the results from the semantic file to ASCII code

**Code Generation:**

* Converts the results from the semantic file to machine code

**Optimization:**

There are 2 levels of optimization

* Level 0: No optimization
* Level 1: Removes chorus and verses within song file, leaving only the words chorus, verses as it is stored in the symbol table where each verse and chorus can be retrieved.
* Level 2: Compresses the Ascii file.

Technical Contribution

|  |  |
| --- | --- |
| Team Member | Contribution |
| Julia Pottinger | Preprocessing  Lexical  Syntax  Semantic  Synthesized Beats  Interoperability |
| Tamoya Beckford | Symbol Table  GUI  Synthesized Beats  Interoperability |
| Christopher Fogo | Client-Server  Intermediate Code Generation  Code Generation  Code Optimization |
| Carlington Palmer | Client-Server  Code Generation  Code Optimization  English-Spanish Conversion  Synthesized Beats |