NL2UNIX

Andrei Mateescu

June 2023

1 Introduction

The Natural Language to Unix Time Converter is a program that converts natural language expressions representing dates and times into Unix timestamps. This documentation provides an overview of the project.

2 Input and output format

The Natural Language to Unix Time Converter accepts natural language expressions that represent dates and times. The supported input formats include: Expressions like "in the morning," "in the afternoon," "in the evening," "at night," "tonight,", etc. Specific times like "noon" and "midnight." Relative times like "now," "tomorrow," "yesterday," "from," "last," "next," "ago," "before," "after," and "in." Date formats like "month/day/year," "day/month/year," and "year/month/day." Numeric values for day, month, and year.

Output format

It provides Unix timestamps as output. A Unix timestamp represents the number of seconds elapsed since January 1, 1970, at 00:00:00 UTC. The output format is an integer value representing the Unix timestamp corresponding to the input expression.

3 Lex File

The Lex file is responsible for tokenizing the input and generating appropriate tokens based on the specified regular expressions. It works in conjunction with the Yacc (Bison) file to build a complete date/time conversion system. The Lex file defines the following tokens:

3.1 DateTime Terms

Tokens that represent units of time such as seconds, minutes, hours, days, weeks, months, and years.

3.2 Regular Time

Tokens that represent specific time values in regular clock format, such as "12:30 pm" or "8:45am".

3.3 Specific Times

Tokens that represent specific times of the day, such as "noon" or "midnight".

3.4 General Time

Tokens that represent general time periods, such as "morning", "afternoon", "evening", "night", "tonight", or "tonite".

3.5 Months

Tokens that represent the names of months, such as "January", "February", etc.

3.6 Days

Tokens that represent the names of days of the week, such as "Sunday", "Monday", etc.

3.7 Generic Numbers

Tokens that represent generic numerical values from 1 to 12.

3.8 Day Dates

Tokens that represent dates in the format of "day/month" or "month/day".

3.9 Year Dates

Tokens that represent dates in the format of "year/month/day" or "year-month-day".

3.10 Set Modifier

Tokens that represent the "on" modifier used to set a specific day.

3.11 Other Tokens

Tokens for various other elements such as "now", "tomorrow", "yesterday", "from", "last", "next", "ago", "before", "after", "in", "this", "at", "the", ",", "+", and errors.

4 Token Handling

The Lex file contains rules to match the input text and generate the appropriate tokens. Here are some examples of the token handling rules:

```
{dateTimeTerms} {
    /* Handling code for DateTime Terms tokens */
    return TYPENAMES;
}

{regularTime} {
    /* Handling code for Regular Time tokens */
    return WHENTIME;
}

{specificTimes} {
    /* Handling code for Specific Times tokens */
    return WHENTIME;
}

{generalTime} {
    /* Handling code for General Time tokens */
    return GENERALTIME;
}

{months} {
    /* Handling code for Months tokens */
    return MONTHNUM;
}

{days}(\.|days?)? {
    /* Handling code for Days tokens */
    return DAYOFWEEK;
}

{daydate}"/"{genericNum}"/"{yeardate} {
    /* Handling code for Day Dates tokens */
    return SETDATE;
}

...
```

5 Error Handling

The Lex file also includes error handling rules to handle unrecognized input. These rules typically concatenate the unrecognized text and switch to an error state. Here's an example:

```
. {
    BEGIN(error);
    concatUnusedString(strdup(yytext));
}

<error>. {
    concatUnusedString(strdup(yytext));
}
```

6 To sum up the lex file

The Lex file is an essential component of the program that converts natural language date/time to Unix timestamp. It defines the tokens and handles the matching and processing of input text based on the specified regular expressions. Overall, the Lex file works in conjunction with the Yacc (Bison) file to create a complete date/time conversion system. The Lex file tokenizes the input text, while the Yacc file defines the grammar rules and performs semantic actions based on the token stream generated by the Lex file.

7 Yacc File

The Yacc file is responsible for parsing user input and generating appropriate actions based on the specified grammar rules. It works in conjunction with the Lex file to build a complete time parsing system.

8 Grammar Rules

The following grammar rules are implemented in the Yacc file:

8.1 Rule 1: Expression

This rule represents the main expression in the time parsing grammar.

```
expr    : SETDATE DATE
        {
            /* Actions to set the specified date */
        }
        |
        WHENTIME TIMESTRING
```

```
{
    /* Actions to handle the "whentime" modifier */
}
|
expr BEFORE expr
{
    /* Actions to handle the "before" modifier */
}
|
expr AFTER expr
{
    /* Actions to handle the "after" modifier */
}
|
expr LAST TIMEOFDAY
{
    /* Actions to handle the "last" modifier */
}
|
expr NEXT TIMEOFDAY
{
    /* Actions to handle the "next" modifier */
}
|
expr IN TIMEOFDAY
{
    /* Actions to handle the "in" modifier */
}
|
expr INTE
```

Continued on the next page...

```
|
expr INTHE TYPENAMES
{
    /* Actions to handle the "in the" modifier with type names */
}
|
INTHE
{
    /* Actions to handle the "in the" modifier without type names */
}
;
```

8.2 Rule 2: Error Handling

This rule is used to handle syntax errors in the input.

```
void yyerror(char *s)
{
    fprintf(stderr, "%s", s);
}
```

8.3 Rule 3: Main Function

This rule represents the main function of the Yacc file.

```
int main(void)
{
    yyparse();
    return 0;
}
```

9 To sum up the yacc file

The Yacc file provides the grammar rules and actions necessary for parsing time-related input in the time parsing application. It works in conjunction with the Lex file to create a robust time parsing system. The provided documentation gives an overview of the implemented grammar rules and their associated actions.

10 C Function Definitions

10.1 `void setCurrentTime()`

[language=C] void setCurrentTime();

This function sets the current time by obtaining the current system time and populating the `str_time` structure with the corresponding time values. It also initializes various variables used for time manipulation.

10.2 `void setTime(int sec, int min, int hr)`

[language=C] void setTime(int sec, int min, int hr);

This function sets the time values (seconds, minutes, and hours) in the `str_time` structure.

10.3 `void setDate(int day, int month, int year, int wday)`

[language=C] void setDate(int day, int month, int year, int wday);

This function sets the date values (day, month, year, and weekday) in the `str_time` structure.

10.4 `void setBoth(int sec, int min, int hr, int day, int month, int year, int wday)`

[language=C] void setBoth(int sec, int min, int hr, int day, int month, int year, int wday);

This function sets both the time and date values in the `str_time` structure.

10.5 `void setFinalTime(struct tm *temp, int amount[], int *change)`

[language=C] void setFinalTime(struct tm *temp, int amount[], int *change);

This function sets the final time based on the provided time structure (`temp`), amount values, and change values. It performs various calculations and updates the `str_time` structure accordingly.

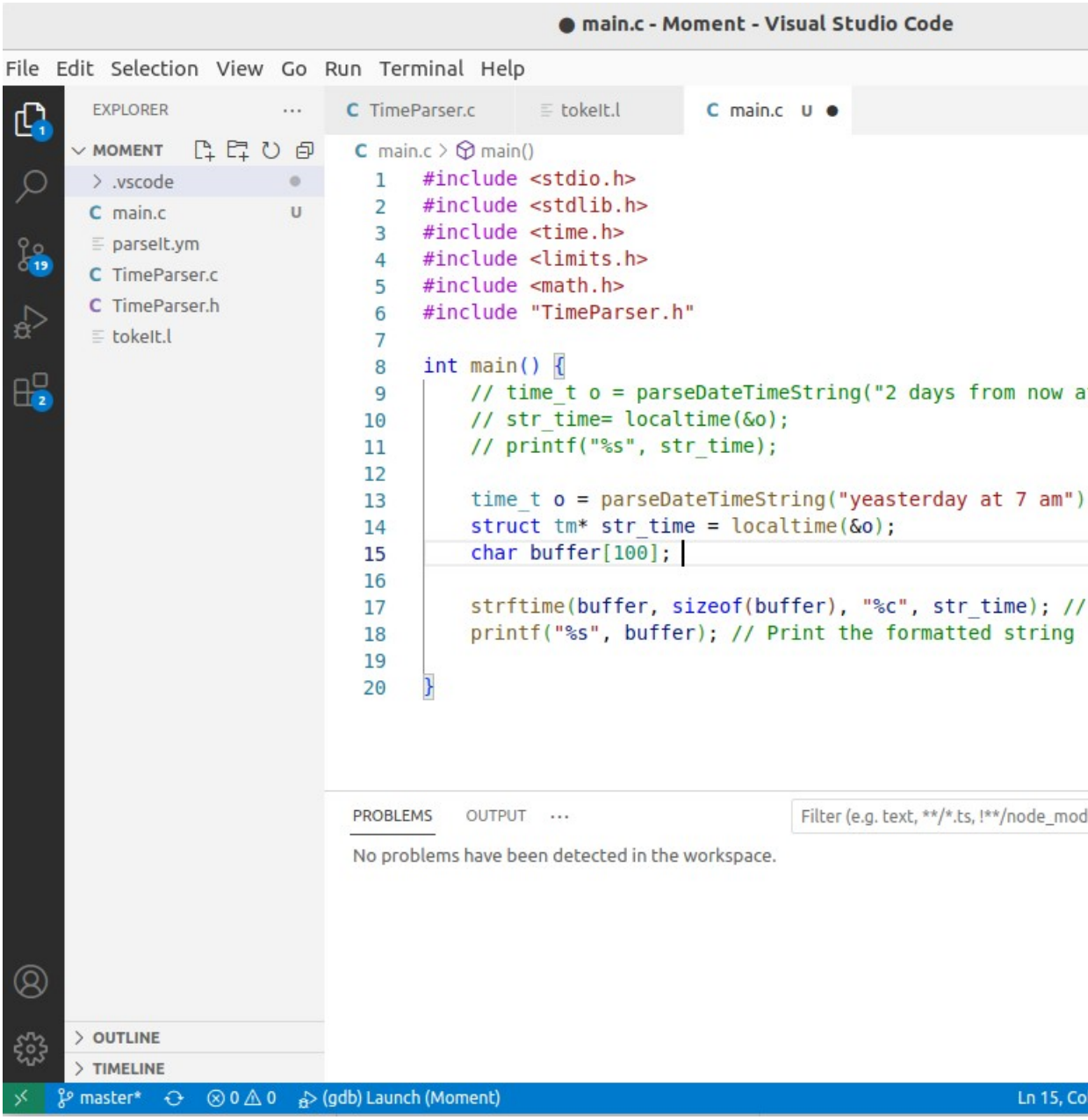
10.6 `void setDayOfWeek(int weekday, int *amount)`

[language=C] void setDayOfWeek(int weekday, int *amount);

This function sets the day of the week in the `str_time` structure based on the provided weekday value. It also takes an array of amount values for further calculations.

11 Example

Since we cannot understand UNIX time, we have to convert it to string using strftime().



Call the parseDateTimeString() funtion and convert the output to string.

12 Github

https://github.com/Andrei-2147483647/MateescuAndreiLFT