Jonas Carlsson
Joca1604@student.miun.se

Labb1 Programspråksteori

Programmet som skapats I detta projekt är en Regular expression parser. Programmet funkar genom att Tokenizea det expression som skickats in genom en Tokenizer. Utrycket omvandlas från text form till en vector av tokens. Varje token består av mText som är karaktären samt mType vilket beskriver Token typ. Efter att utrycket körts igenom tokenizer så byggs parseträdet med hjälp av tokens.

Parseträdet är byggt efter metoden recursive decent parsing och följer grammatik byggt I formen Backus Naur Form (BNF). Parseträdet uttnyjar structar som noder där varje nod innehåller en vector som lagrar childnodes. För att enkelt kunna se hur trädet ser ut byggdes även en print funktion som uttnyttjar ID() funktioner inom structarna som returnerar namnen på nodernas typ.

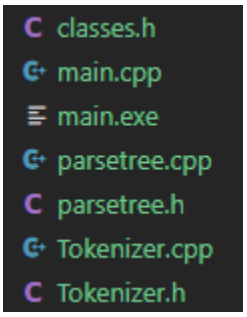
Till sist så byggdes även evalueringsmetoder för alla structar som jämför parseträdet med text som skickas in med exekveringsfunktionen.

BNF

```
<Regular-expr> ::= <Output> | <Or-expr> | <simple-expr>

<Output> ::= <Regular-expr> "\O" "{" -digit- "}" | <Regular-expr> "\O"

<OR-expr> ::= <simple-expr> "+" <Regular-expr>

<simple-expr> ::= <concatenation> | <basic-expr>

<concatenation> ::= <basic-expr> <simple-expr>

<basic-expr> ::= <lowercase> | <group> | <repetition> | <counter> | <any> |
<character>

<lowercase> ::= <Regular-expr> "\I"

<repetition> ::= <element> "*"

<counter> ::= <element> "{" -digit- "}"

<group> ::= "(" <Regular-expr> ")"

<any> ::= "."

<character> ::= -any non metacharacter-
```

Programmet består av följande filer:

Jonas Carlsson DT096G 2022-03-03
Joca1604@student.miun.se



Dessa filer kan delas upp I 3 huvudsakliga delar. Tokenizer.cpp och Tokenizer.h består av klassen för att omvandla reg Exp till tokens. Parsetree.cpp och parsetree.h består av klassen för att bygga parseträdet. Classes.h består av noderna I form av structs som används för att bygga parseträdet. Dessa noder innehåller evalueringsfunktionen vilket sedan används till att evaluera uttrycket mot inputtexten.

TokenTypes

```
enum TokenType {
    NONTOKEN,
    POTENTIAL_OPERATOR,
    CHARACTER,
    OR_SYMBOL,
    REPETITION_SYMBOL,
    REPETITION_VALUE,
    ANY_SYMBOL,
    LEFT_PARANTHESES,
    RIGHT_PARANTHESES,
    AMOUNT_VALUE,
    LEFT_BRACKET,
    DIGIT,
    RIGHT_BRACKET,
    IGNORE_SENSITIVITY,
    OUTPUT
};
```

Tokenizer

```
//Checking for different Operations
switch(currCh) {
    case '+':
        endToken(currentToken, tokens);
        currentToken.mType = OR_SYMBOL;
        currentToken.mText.append(1, currCh);
        break;
    case '*':
        endToken(currentToken, tokens);
        currentToken.mType = REPETITION_SYMBOL;
        currentToken.mText.append(1, currCh);
        break;
```

Building ParseTree (Character)

Returns expr with appended information to _id if successful and returns nullptr if not successful.

```
op* ParseTreeClass::character_expr(std::vector<simpleparser::Token>::iterator& first, s

    //Get token
    simpleparser::Token thisToken = *first;

    //If token is <character>
    if(simpleparser::sTokenTypeStrings[thisToken.mType] == "CHARACTER"){
        character* expr = new character;
        expr->_id.append(thisToken.mText);
        first++;
        std::cout << "Found: " << thisToken.mText << std::endl;
        return expr;
    }

    //If token is not <character>
    return nullptr;
};
```

Struct of the Character Node

Evaluation function returns nullptr if non match and updated object if match.

Object is a pointer to the text positions.

Joca1604@student.miun.se

```
struct character : op {

    std::string _id; //Identifier

    object* eval(object* obj) override {
        //Check if we reached end of string
        if (obj->rhs == obj->end) {
            return nullptr;
        }

        //If not match
        if (*obj->rhs != *_id.begin()) {
            return nullptr;
        }

        //Match, return character
        obj->rhs++;
        return obj;

    }

    std::string id() override {
        return "<character> (" + _id + ")";
    }
};
```