Micro Scanner

Wygenerowano przez Doxygen 1.9.3

Chapter 1

Indeks struktur danych

1.1 Struktury danych

Tutaj znajdują się struktury danych wraz z ich krótkimi opisami:

Chapter 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Chapter 3

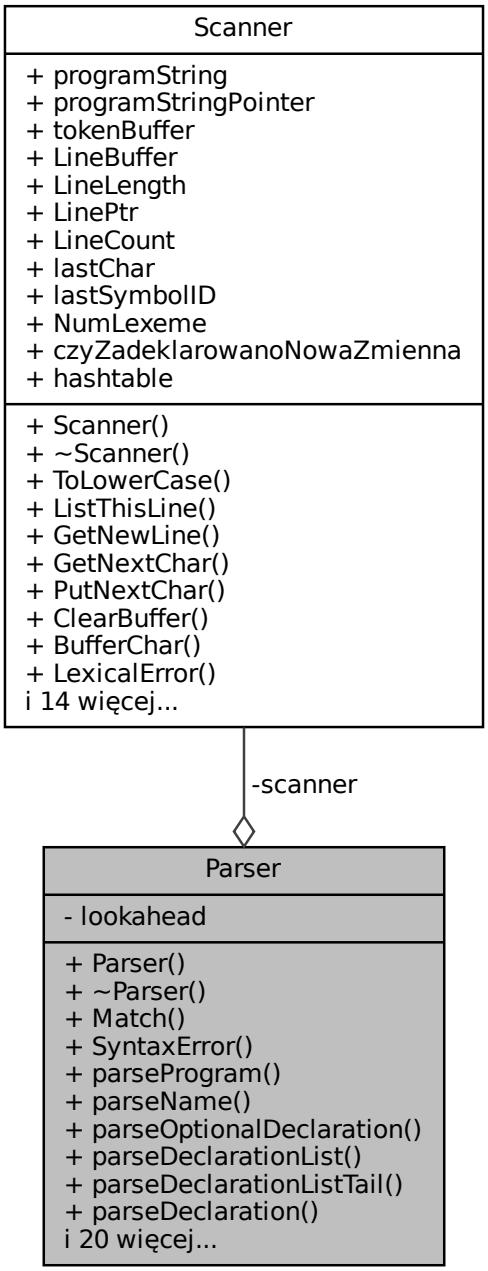
Dokumentacja struktur danych

3.1 Dokumentacja klasy Parser

Klasa realizująca funkcje parsera kodu Micro.

`#include <Parser.hh>`

Diagram współpracy dla Parser:



Metody publiczne

- Parser (std::string program)
- ~Parser ()
- void Match (Token)

  Sprawdza czy kolejny Token w kodzie jest tym który otrzymuje jako argument. Następnie prosi scanner o znalezienie kolejnego Tokena.

- void SyntaxError (Token T)
- void parseProgram ()

  parse <program> -> PROGRAM <name> <optional declaration> BEGIN <optional statement> END.
- void parseName ()

  parse <name> -> <ident>;
- void parseOptionalDeclaration ()

  parse : <optional declaration> -> <declaration list> | NULL
- void parseDeclarationList ()

  parse : <declaration list> -> <declaration> <declaration list tail>
- void parseDeclarationListTail ()

  parse : <declaration list tail> -> NULL | <declaration> <declaration list>
- void parseDeclaration ()

  parse : <declaration> -> VAR <id list> : <type> ;
- void parseIdList (bool sprawdzZadeklarowanie=false, bool niePowinnaBycZadeklarowana=false)

  parse : <id list> -> <ident> <id list tail>
- void parseIdListTail (bool sprawdzZadeklarowanie=false, bool niePowinnaBycZadeklarowana=false)

  parse : <id list tail> -> NULL | ,<ident> <id list tail>
- void parseType ()

  parse : <type> -> INTEGER | REAL
- void parseOptionalStatement ()

  parse : <optional statement> -> <statement list> <statement> | NULL
- void parseStatementList ()

  parse : <statement list> -> <statement> <statement list tail>
- void parseStatementListTail ()

  parse : <statement list tail> -> NULL | <statement> <statement list tail>
- void parseStatement ()

  parse : <statement> -> BEGIN <statement list> END;
  | <ident> := <expression>; | READ(<id list>); | WRITE(<expression list>); | IF(<bool>) THEN <statement> | WHILE(<bool>) DO <statement>
- void parseExpressionList ()

  parse : <expression list> -> <expression> <expression list tail>
- void parseExpressionListTail ()

  parse : <expression list tail> -> NULL | , <expression> <expression list tail>
- void parseExpression ()

  parse : <expression> -> <expression tail>
- void parseExpressionTail ()

  parse : <expression tail> -> NULL | + <expression tail> | - <expression tail>
- void parseTerm ()

  parse : -> <factor>
- void parseTermTail ()

  parse : -> NULL | ∗ <factor> | / <factor>
- void parseFactor ()

  parse : <factor> -> (<expression>) | <ident> | INTNUM | FLOATNUM
- void parseIdent ()

  parse : <ident> -> ID
- void parseBool ()

  parse : <bool> -> <expression> <relation op> <expression>
- void parseRelationOp ()

  parse : <relation op> -> < | = | > | <= | <> | >=
- bool isBeginOfStatemnt (Token token)

  sprawdza czy token jest jednym z tokenów rozpoczynających <statemnt>

- void sprawdzCzyZadeklarowanaWczesniej ()

  Sprawdza czy następna napotkana zmienna była już zadeklarowana wcześniej. Jeśli tak to wypisuje error, i kończy parser.
- void sprawdzCzyPonownaDeklaracja ()

  Sprawdza czy następna napotkana zmienna była już zadeklarowana i następuje jej ponowna deklaracja. Jeśli tak, to wypisuje error i kończy parser.

Atrybuty prywatne

- Scanner ∗ scanner

  wskaźnik na obiekt realizujący funkcję skanera
- Token lookahead

  Kolejny znaleziony token.

3.1.1 Opis szczegółowy

Klasa realizująca funkcje parsera kodu Micro.

3.1.2 Dokumentacja konstruktora i destruktora

3.1.2.1 Parser()

```
Parser::Parser (
            std::string program )
```

3.1.2.2 ∼Parser()

```
Parser::~Parser ( )
```

3.1.3 Dokumentacja funkcji składowych

3.1.3.1 isBeginOfStatemnt()

```
bool Parser::isBeginOfStatemnt (
            Token token )
```

sprawdza czy token jest jednym z tokenów rozpoczynających <statemnt>

Parametry

token	token do sprawdzenia

Zwraca

true jest jednym z tokenów rozpoczynających <statemnt>

false nie jest jednym z tokenów rozpoczynających <statemnt>

3.1.3.2 Match()

```
void Parser::Match (
            Token T )
```

Sprawdza czy kolejny Token w kodzie jest tym który otrzymuje jako argument. Następnie prosi scanner o znalezienie kolejnego Tokena.

Parametry

T	Token do którego będzie porównywany następny token w kodzie

3.1.3.3 parseBool()

```
void Parser::parseBool ( )
```

parse : <bool> -> <expression> <relation op> <expression>

3.1.3.4 parseDeclaration()

```
void Parser::parseDeclaration ( )
```

parse : <declaration> -> VAR <id list> : <type> ;

3.1.3.5 parseDeclarationList()

```
void Parser::parseDeclarationList ( )
```

parse : <declaration list> -> <declaration> <declaration list tail>

3.1.3.6 parseDeclarationListTail()

```
void Parser::parseDeclarationListTail ( )
```

parse : <declaration list tail> -> NULL | <declaration> <declaration list>

3.1.3.7 parseExpression()

```
void Parser::parseExpression ( )
```

parse : <expression> -> <expression tail>

3.1.3.8 parseExpressionList()

```
void Parser::parseExpressionList ( )
```

parse : <expression list> -> <expression> <expression list tail>

3.1.3.9 parseExpressionListTail()

```
void Parser::parseExpressionListTail ( )
```

parse : <expression list tail> -> NULL | , <expression> <expression list tail>

3.1.3.10 parseExpressionTail()

```
void Parser::parseExpressionTail ( )
```

parse : <expression tail> -> NULL | + <expression tail> | - <expression tail>

3.1.3.11 parseFactor()

```
void Parser::parseFactor ( )
```

parse : <factor> -> (<expression>) | <ident> | INTNUM | FLOATNUM

3.1.3.12 parseIdent()

```
void Parser::parseIdent ( )
```

parse : <ident> -> ID

3.1.3.13 parseIdList()

```
void Parser::parseIdList (
            bool sprawdzZadeklarowanie = false,
            bool niePowinnaBycZadeklarowana = false )
```

parse : <id list> -> <ident> <id list tail>

3.1.3.14 parseIdListTail()

```
void Parser::parseIdListTail (
            bool sprawdzZadeklarowanie = false,
            bool niePowinnaBycZadeklarowana = false )
```

parse : <id list tail> -> NULL | ,<ident> <id list tail>

3.1.3.15 parseName()

```
void Parser::parseName ( )
```

parse <name> -> <ident>;

3.1.3.16 parseOptionalDeclaration()

```
void Parser::parseOptionalDeclaration ( )
```

parse : <optional declaration> -> <declaration list> | NULL

3.1.3.17 parseOptionalStatement()

```
void Parser::parseOptionalStatement ( )
```

parse : <optional statement> -> <statement list> <statement> | NULL

3.1.3.18 parseProgram()

```
void Parser::parseProgram ( )
```

parse <program> -> PROGRAM <name> <optional declaration> BEGIN <optional statement> END.

3.1.3.19 parseRelationOp()

```
void Parser::parseRelationOp ( )
```

parse : <relation op> -> < | = | > | <= | <> | >=

3.1.3.20 parseStatement()

```
void Parser::parseStatement ( )
```

parse : <statement> -> BEGIN <statement list> END;
| <ident> := <expression>; | READ(<id list>); | WRITE(<expression list>); | IF(<bool>) THEN <statement>
| WHILE(<bool>) DO <statement>

3.1.3.21 parseStatementList()

```
void Parser::parseStatementList ( )
```

parse : <statement list> -> <statement> <statement list tail>

3.1.3.22 parseStatementListTail()

```
void Parser::parseStatementListTail ( )
```

parse : <statement list tail> -> NULL | <statement> <statement list tail>

3.1.3.23 parseTerm()

```
void Parser::parseTerm ( )
```

parse : -> <factor>

3.1.3.24 parseTermTail()

```
void Parser::parseTermTail ( )
```

parse : -> NULL | ∗ <factor> | / <factor>

3.1.3.25 parseType()

```
void Parser::parseType ( )
```

parse : <type> -> INTEGER | REAL

3.1.3.26 sprawdzCzyPonownaDeklaracja()

```
void Parser::sprawdzCzyPonownaDeklaracja ( )
```

Sprawdza czy następna napotkana zmienna była już zadeklarowana i następuje jej ponowna deklaracja. Jeśli tak, to wypisuje error i kończy parser.

3.1.3.27 sprawdzCzyZadeklarowanaWczesniej()

```
void Parser::sprawdzCzyZadeklarowanaWczesniej ( )
```

Sprawdza czy następna napotkana zmienna była już zadeklarowana wcześniej. Jeśli tak to wypisuje error, i kończy parser.

3.1.3.28 SyntaxError()

```
void Parser::SyntaxError (
            Token T )
```

3.1.4 Dokumentacja pól

3.1.4.1 lookahead

```
Token Parser::lookahead  [private]
```

Kolejny znaleziony token.

3.1.4.2 scanner

```
Scanner* Parser::scanner  [private]
```

wskaźnik na obiekt realizujący funkcję skanera

Dokumentacja dla tej klasy została wygenerowana z plików:

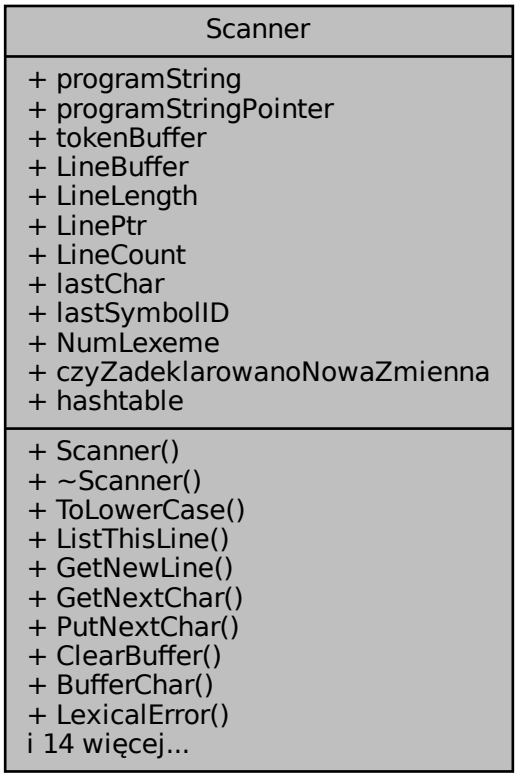
- Parser.hh
- Parser.cpp

3.2 Dokumentacja klasy Scanner

Klasa realizująca funkcje skanera kodu.

```
#include <Scanner.h>
```

Diagram współpracy dla Scanner:



Metody publiczne

- Scanner ()
- ~Scanner ()
- void ToLowerCase ()

  Convert characters to lower case characters.
- void ListThisLine ()

  produce a listing of the source program
- void GetNewLine ()

  get a new line from cached program
- void GetNextChar (char *c)

  Pobiera kolejny znak z zbuforowanej linii, jeśli brak takich to buforuje kolejną linie i podaje jej pierwszy znak.
- void PutNextChar ()

  Zmniejsza znacznik pozycji zbuforowanego znaku.
- void ClearBuffer ()

  Czyści bufor przeznaczony do bufforowania Tokenów.
- void BufferChar (char c)

  Dodaje znak z parametru do bufora tokenBuffer.
- void LexicalError (char c)

  Wypisuje informacje o błędzie lexykalnym na podanym znaku.
- void BufferName (char c)

  Bufforuje wyraz, (wyraz składa się ze znaków alfanumerycznych lub '_' lub '.')
- int BufferNumLiterals (char c)

  Bufforuje liczby, (liczba składa się z cyfry lub '.')
- Token CheckReserved ()

  Check is the symbol into already reserved Symbols.
- Token GetNextToken ()

  Znajduje kolejny Token w tekscie programu.
- void scan (std::string program)

  Przeprowadza procedurę skanowania podanego programu.
- int Enter (std::string S, Token Code)

  Put S unconditionally into the symbol table and returns index of the entry for S.
- void LoadKeywords ()

  Dodaje Słowa kluczowe do hashListy.
- void ListSymbolTable ()

  Wypisuje wszystkie Symbole przechowywane w HashTable.
- int LookUp (std::string S)

  Returns id in hashTable of the entry for S, or 0 if it is not found.
- void printOutProgramCodeWithLines ()

  Wypisuje kod programu z ponumerowanymi liniamii.
- void printTokenName (Token token)
- std::string getTokenName (Token token)

  Zwraca podany token jako string Służy do zmiany tokena na string zawierający jego nazwę.
- Symbol getSymbolFromHashTableById (int id)
- void addProgram (std::string program)

Pola danych

- std::string programString = ""

  zacachowany kod programu.
- int programStringPointer = 0

  Przechowuje index kolejnego znaku do wczytania do buforuLini.
- std::string tokenBuffer

  Buffor uzywany do znajdywania kolejnych tokenów.
- char LineBuffer [MAX_LINE_LENGTH+1]

  używany do przechowywania kolejnych linii programu
- int LineLength

  Przechowuje długość aktualnie zbuforowanej linii.
- int LinePtr
- int LineCount

  Numer szeregowy aktualnej linii.
- char lastChar

  Ostatni zbuforowany znak.
- int lastSymbolID = 0

  Zmienna przechowująca id(w hashTable) ostatnio znaleziony symbol.
- std::string NumLexeme = ""

  Do tymczasowego przechowawania wartości zmiennej numerycznej (w postaci stringa)
- bool czyZadeklarowanoNowaZmienna = false

  Zmienna przechowująca czy ostatnio znalezione id zostało dodane do listy(true), czy była już wcześniej dodana (false)
- std::unordered_map< int, Symbol > * hashtable

  hash tablica

3.2.1 Opis szczegółowy

Klasa realizująca funkcje skanera kodu.

Autor

Marek Pałdyna

3.2.2 Dokumentacja konstruktora i destruktora

3.2.2.1 Scanner()

```
Scanner::Scanner ( )
```

3.2.2.2 ~Scanner()

```
Scanner::~Scanner ( )
```

3.2.3 Dokumentacja funkcji składowych

3.2.3.1 addProgram()

```
void Scanner::addProgram (
            std::string program )
```

3.2.3.2 BufferChar()

```
void Scanner::BufferChar (
            char c )
```

Dodaje znak z parametru do bufora tokenBuffer.

Zobacz również

tokenBuffer

Parametry

c	znak do dodania do bufora

3.2.3.3 BufferName()

```
void Scanner::BufferName (
            char c )
```

Bufforuje wyraz, (wyraz składa się ze znaków alfanumerycznych lub '_' lub '.')

Parametry

c	znak od którego zaczy się buforowane słowo

3.2.3.4 BufferNumLiterals()

```
int Scanner::BufferNumLiterals (
            char c )
```

Bufforuje liczby, (liczba składa się z cyfry lub '.')

Parametry

c	- pierwsza cyfra/znak

Zwraca

int - 1 jeśli liczba jest typu REAL, 0 jeśli liczba jest typu INTEGER

3.2.3.5 CheckReserved()

```
Token Scanner::CheckReserved ( )
```

Check is the symbol into already reserved Symbols.

Zwraca

Token::ID jeśli nie znaleziono wśród słów już zarezerowanych w przeciwnym razie zwraca Typ zarezerwowanego słowa

3.2.3.6 ClearBuffer()

```
void Scanner::ClearBuffer ( )
```

Czyści bufor przeznaczony do bufforowania Tokenów.

3.2.3.7 Enter()

```
int Scanner::Enter (
            std::string S,
            Token Code )
```

Put S unconditionally into the symbol table and returns index of the entry for S.

Parametry

S	Name
Code	Token

Zwraca

int id wpisu w hashLiście

3.2.3.8 GetNewLine()

```
void Scanner::GetNewLine ( )
```

get a new line from cached program

3.2.3.9 GetNextChar()

```
void Scanner::GetNextChar (
            char * c )
```

Pobiera kolejny znak z zbuforowanej linii, jeśli brak takich to buforuje kolejną linie i podaje jej pierwszy znak.

Parametry

c	zmienna do której zostaje przypisany kolejny znak

3.2.3.10 GetNextToken()

```
Token Scanner::GetNextToken ( )
```

Znajduje kolejny Token w tekscie programu.

Zwraca

Token

3.2.3.11 getSymbolFromHashTableById()

```
Symbol Scanner::getSymbolFromHashTableById (
            int id )
```

3.2.3.12 getTokenName()

```
std::string Scanner::getTokenName (
            Token token )
```

Zwraca podany token jako string Służy do zmiany tokena na string zawierający jego nazwę.

Parametry

token	

Zwraca

std::string

"BEGIN"

"END."

"END"

"READ"

"WRITE"

"VAR"

"IF"

"THEN"

"WHILE"

"DO"

"INTEGER"

"REAL"

"+"

"-"

"*"

"/"

"<"

"<="

">"

">="

"="

"<>"

"("

")"

":="

";"

","

3.2.3.13 LexicalError()

```
void Scanner::LexicalError (
            char c )
```

Wypisuje informacje o błędzie lexykalnym na podanym znaku.

Parametry

c	

3.2.3.14 ListSymbolTable()

```
void Scanner::ListSymbolTable ( )
```

Wypisuje wszystkie Symbole przechowywane w HashTable.

3.2.3.15 ListThisLine()

```
void Scanner::ListThisLine ( )
```

produce a listing of the source program

3.2.3.16 LoadKeywords()

```
void Scanner::LoadKeywords ( )
```

Dodaje Słowa kluczowe do hashListy.

3.2.3.17 LookUp()

```
int Scanner::LookUp (
            std::string S )
```

Returns id in hashTable of the entry for S, or 0 if it is not found.

Parametry

S	Entry

Zwraca

int

3.2.3.18 printOutProgramCodeWithLines()

```
void Scanner::printOutProgramCodeWithLines ( )
```

Wypisuje kod programu z ponumerowanymi liniamii.

3.2.3.19 printTokenName()

```
void Scanner::printTokenName (
            Token token )
```

3.2.3.20 PutNextChar()

```
void Scanner::PutNextChar ( )
```

Zmniejsza znacznik pozycji zbuforowanego znaku.

3.2.3.21 scan()

```
void Scanner::scan (
            std::string program )
```

Przeprowadza procedurę skanowania podanego programu.

Ostrzeżenie

program musi kończyć się znakiem EOF!

Parametry

program	zmienna typu string z wczytanym kodem programu

3.2.3.22 ToLowerCase()

```
void Scanner::ToLowerCase ( )
```

Convert characters to lower case characters.

3.2.4 Dokumentacja pól

3.2.4.1 czyZadeklarowanoNowaZmienna

```
bool Scanner::czyZadeklarowanoNowaZmienna = false
```

Zmienna przechowująca czy ostatnio znalezione id zostało dodane do listy(true), czy była już wcześniej dodana (false)

3.2.4.2 hashtable

```
std::unordered_map<int, Symbol>* Scanner::hashtable
```

hash tablica

3.2.4.3 lastChar

```
char Scanner::lastChar
```

Ostatni zbuforowany znak.

3.2.4.4 lastSymbolID

```
int Scanner::lastSymbolID = 0
```

Zmienna przechowująca id(w hashTable) ostatnio znaleziony symbol.

3.2.4.5 LineBuffer

```
char Scanner::LineBuffer[MAX_LINE_LENGTH+1]
```

używany do przechowywania kolejnych linii programu

3.2.4.6 LineCount

```
int Scanner::LineCount
```

Numer szeregowy aktualnej linii.

3.2.4.7 LineLength

```
int Scanner::LineLength
```

Przechowuje długość aktualnie zbuforowanej linii.

3.2.4.8 LinePtr

```
int Scanner::LinePtr
```

3.2.4.9 NumLexeme

```
std::string Scanner::NumLexeme = ""
```

Do tymczasowego przechowawania wartości zmiennej numerycznej (w postaci stringa)

3.2.4.10 programString

```
std::string Scanner::programString = ""
```

zacachowany kod programu.

3.2.4.11 programStringPointer

```
int Scanner::programStringPointer = 0
```

Przechowuje index kolejnego znaku do wczytania do buforuLini.

3.2.4.12 tokenBuffer

```
std::string Scanner::tokenBuffer
```

Buffor uzywany do znajdywania kolejnych tokenów.

Dokumentacja dla tej klasy została wygenerowana z plików:

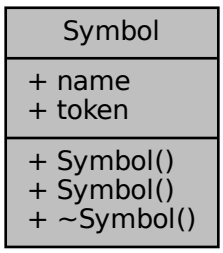
- Scanner.h
- Scanner.cpp

3.3 Dokumentacja klasy Symbol

Klasa Przechowująca dane (nazwę i token) - elemnt hashtable.

```
#include <Symbol.h>
```

Diagram współpracy dla Symbol:



Metody publiczne

- Symbol ()
- Symbol (std::string name, Token token)
- ~Symbol ()

Pola danych

- std::string name
- Token token

3.3.1 Opis szczegółowy

Klasa Przechowująca dane (nazwę i token) - elemnt hashtable.

Autor

Marek Pałdyna

3.3.2 Dokumentacja konstruktora i destruktora

3.3.2.1 Symbol() [1/2]

```
Symbol::Symbol ( )
```

3.3.2.2 Symbol() [2/2]

```
Symbol::Symbol (
            std::string name,
            Token token )  [inline]
```

3.3.2.3 ~Symbol()

```
Symbol::~Symbol ( )
```

3.3.3 Dokumentacja pól

3.3.3.1 name

```
std::string Symbol::name
```

3.3.3.2 token

```
Token Symbol::token
```

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symbol.h
- Symbol.cpp

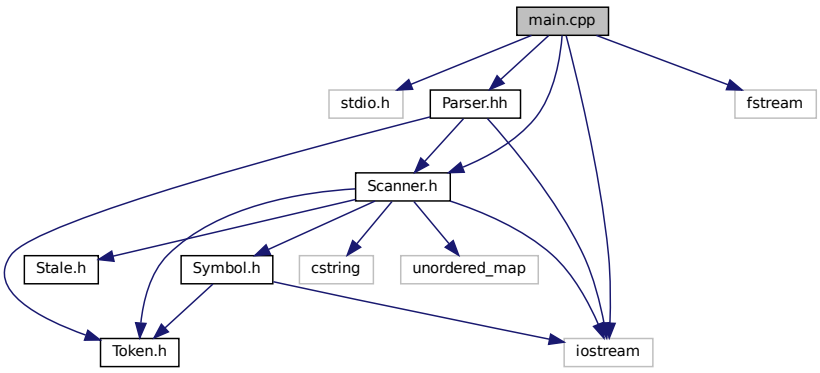
Chapter 4

Dokumentacja plików

4.1 Dokumentacja pliku main.cpp

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include "Scanner.h"
#include "Parser.hh"
```

Wykres zależności załączania dla main.cpp:



Funkcje

- string readFile (string path)
- int main (int argc, char const ∗argv[ ])

4.1.1 Dokumentacja funkcji

4.1.1.1 main()

```
int main (
             int argc,
             char const * argv[] )
```

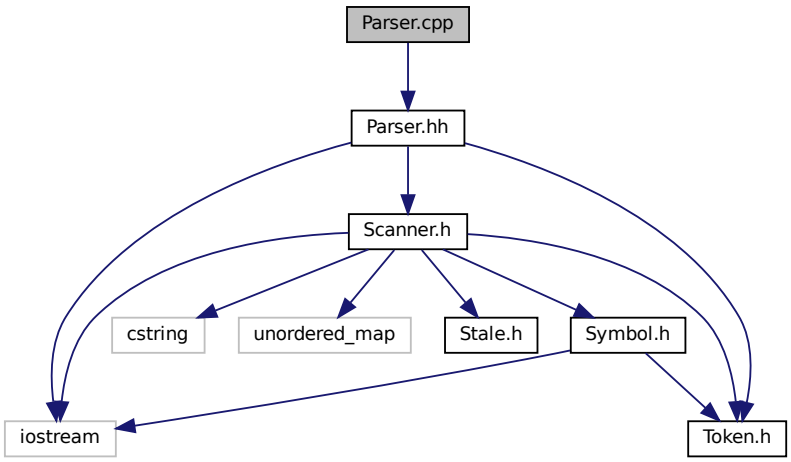
4.1.1.2 readFile()

```
string readFile (
             string path )
```

4.2 Dokumentacja pliku Parser.cpp

```
#include "Parser.hh"
```

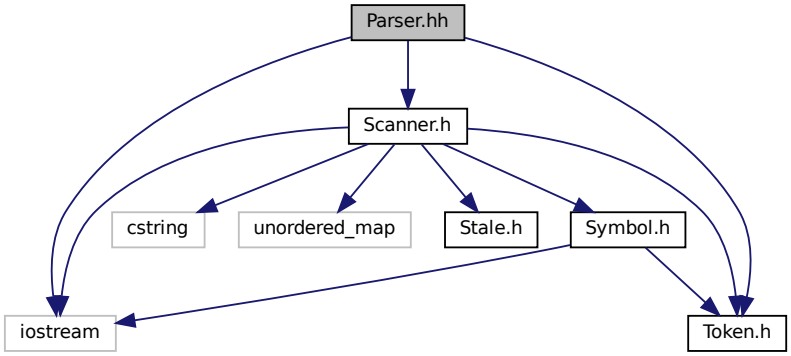
Wykres zależności załączania dla Parser.cpp:



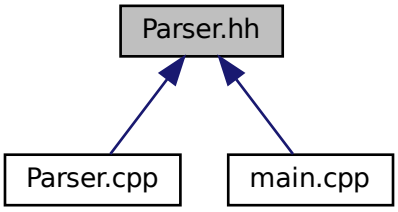
4.3 Dokumentacja pliku Parser.hh

```
#include <iostream>
#include "Token.h"
```

`#include "Scanner.h"`
Wykres zależności załączania dla Parser.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Struktury danych

- class Parser

  Klasa realizująca funkcje parsera kodu Micro.

4.4 Parser.hh

Idź do dokumentacji tego pliku.

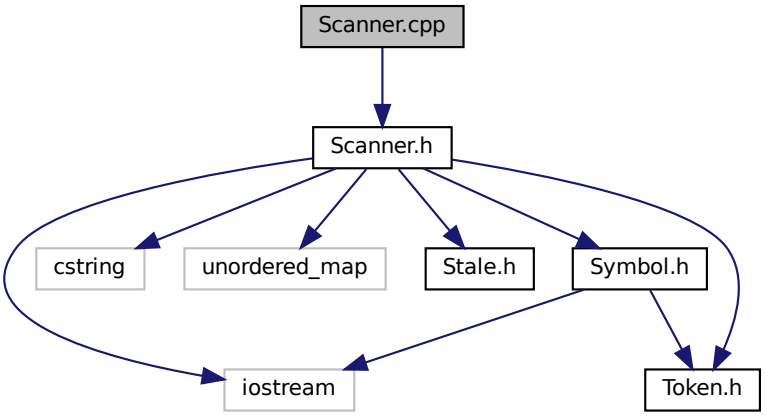
```
1 #ifndef PARSER_H
2 #define PARSER_H
3
4 #include <iostream>
5 #include "Token.h"
6 #include "Scanner.h"
7
8
9
10
11
```

```
12 using namespace std;
13
14
19 class Parser
20 {
21     private:
23     Scanner* scanner;
24
26     Token lookahead;
27
28     public :
29         Parser(std::string program);
30         ~Parser();
35         void Match(Token);
36
37         void SyntaxError(Token T);
42         void parseProgram();
43
48         void parseName();
53         void parseOptionalDeclaration();
54
59         void parseDeclarationList();
60
65         void parseDeclarationListTail();
70         void parseDeclaration();
75         void parseIdList(bool sprawdzZadeklarowanie = false, bool niePowinnaBycZadeklarowana = false);
80         void parseIdListTail(bool sprawdzZadeklarowanie = false, bool niePowinnaBycZadeklarowana =
     false);
85         void parseType();
90         void parseOptionalStatement();
95         void parseStatementList();
100         void parseStatementListTail();
110         void parseStatement();
115         void parseExpressionList();
120         void parseExpressionListTail();
125         void parseExpression();
130         void parseExpressionTail();
135         void parseTerm();
140         void parseTermTail();
145         void parseFactor();
150         void parseIdent();
155         void parseBool();
160         void parseRelationOp();
161
169         bool isBeginOfStatemnt(Token token);
170
175         void sprawdzCzyZadeklarowanaWczesniej();
176
177
182         void sprawdzCzyPonownaDeklaracja();
183 };
184 #endif
```

4.5 Dokumentacja pliku Scanner.cpp

```
#include "Scanner.h"
```

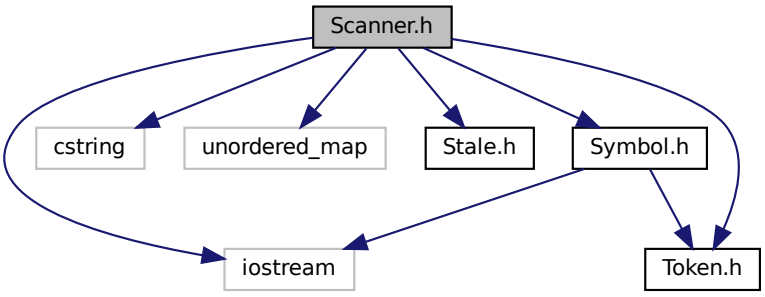
Wykres zależności załączania dla Scanner.cpp:



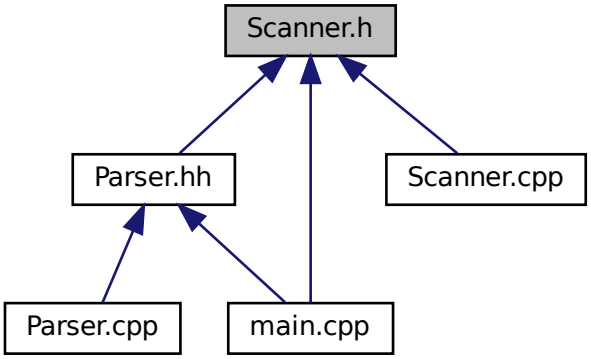
4.6 Dokumentacja pliku Scanner.h

```
#include <iostream>
#include <cstring>
#include <unordered_map>
#include "Stale.h"
#include "Token.h"
#include "Symbol.h"
```

Wykres zależności załączania dla Scanner.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Struktury danych

- class Scanner

  Klasa realizująca funkcje skanera kodu.

4.7 Scanner.h

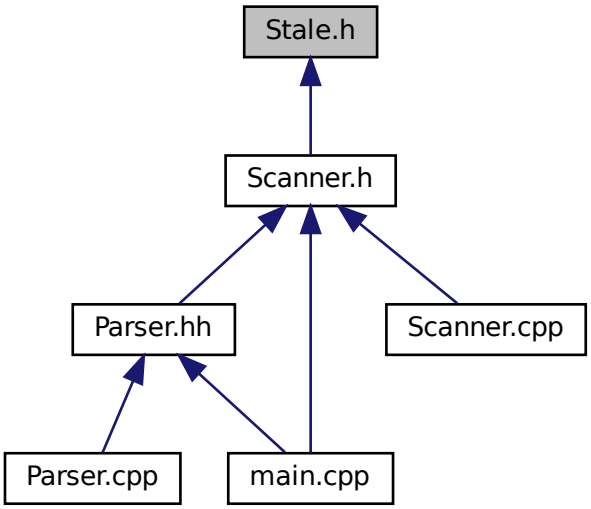
Idź do dokumentacji tego pliku.

```
1 #ifndef SCANNER_H
2 #define SCANNER_H
3 #include <iostream>
4 #include <cstring>
5 #include <unordered_map>
6 #include "Stale.h"
7 #include "Token.h"
8 #include "Symbol.h"
9
15 class Scanner
16 {
17     public:
18
20         std::string programString = "";
22         int programStringPointer = 0;
23
25         std::string tokenBuffer;
26
28         char LineBuffer[MAX_LINE_LENGTH + 1];
29
31         int LineLength;
32
33         int LinePtr;
35         int LineCount;
37         char lastChar;
38
40         int lastSymbolID = 0;
42         std::string NumLexeme = "";
43
45         bool czyZadeklarowanoNowaZmienna = false;
47         //TODO czy to na pewno o to chodziło???
48         std::unordered_map<int, Symbol> *hashtable;
49
50         Scanner();
51         ~Scanner();
52
57         void ToLowerCase();
```

```
58
62          void ListThisLine();
63
68          void GetNewLine();
69
75          void GetNextChar(char *c);
76
81          void PutNextChar();
82
87          void ClearBuffer();
88
95          void BufferChar(char c);
101          void LexicalError(char c);
102
108          void BufferName(char c);
109
116          int BufferNumLiterals(char c);
117
123          Token CheckReserved();
124
130          Token GetNextToken();
131
137          void scan(std::string program);
138
146          int Enter(std::string S, Token Code);
147
152          void LoadKeywords();
153
158          void ListSymbolTable();
159
166          int LookUp(std::string S);
171          void printOutProgramCodeWithLines();
172
173          void printTokenName(Token token);
180          std::string getTokenName(Token token);
181
182          Symbol getSymbolFromHashTableById(int id);
183
184          void addProgram(std::string program);
185 };
186 #endif
```

4.8 Dokumentacja pliku Stale.h

Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Definicje

- #define FALSE 0
- #define TRUE 1
- #define EOL '\n'
- #define EOS '\0'
- #define ID_STRING_LENGTH 32 /* max length of a line */
- #define MAX_LINE_LENGTH 132 /* max length of a line */
- #define TOKEN_SIZE 32 /* max length of a token */
- #define MAX_SYMBOL 100 /* size of symbol table */
- #define NONE -1 /* default token attribute */
- #define DEBUG
- #define D_LN(x) std::cout << x << endl
- #define D(x) std::cout << x

4.8.1 Dokumentacja definicji

4.8.1.1 D

```
#define D(
              x ) std::cout << x
```

4.8.1.2 D_LN

```
#define D_LN(
              x ) std::cout << x << endl
```

4.8.1.3 DEBUG

```
#define DEBUG
```

4.8.1.4 EOL

```
#define EOL '\n'
```

4.8.1.5 EOS

```
#define EOS '\0'
```

4.8.1.6 FALSE

```
#define FALSE 0
```

4.8.1.7 ID_STRING_LENGTH

```
#define ID_STRING_LENGTH 32 /* max length of a line */
```

4.8.1.8 MAX_LINE_LENGTH

```
#define MAX_LINE_LENGTH 132 /* max length of a line */
```

4.8.1.9 MAX_SYMBOL

```
#define MAX_SYMBOL 100 /* size of symbol table */
```

4.8.1.10 NONE

```
#define NONE -1 /* default token attribute */
```

4.8.1.11 TOKEN_SIZE

```
#define TOKEN_SIZE 32 /* max length of a token */
```

4.8.1.12 TRUE

```
#define TRUE 1
```

4.9 Stale.h

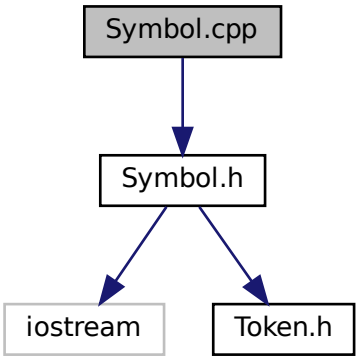
Idź do dokumentacji tego pliku.

```
1 #ifndef __STALE_H__
2 #define __STALE_H__
3
4 #define FALSE 0
5 #define TRUE 1
6 #define EOL '\n'
7 #define EOS '\0'
8
9 /* size definitions */
10
11 #define ID_STRING_LENGTH 32 /* max length of a line */
12 #define MAX_LINE_LENGTH 132 /* max length of a line */
13 #define TOKEN_SIZE 32       /* max length of a token */
14 #define MAX_SYMBOL 100      /* size of symbol table */
15 #define NONE -1             /* default token attribute */
16
17 #define DEBUG
18
19
20
21 #ifdef DEBUG
22 #define D_LN(x) std::cout « x « endl
23 #define D(x) std::cout « x
24
25 #endif // DEBUG
26 #ifndef DEBUG
27 #define D_LN(x)
28 #define D(x)
29 #endif // !DEBUG
30
31 #endif // __STALE_H__
```

4.10 Dokumentacja pliku Symbol.cpp

```
#include "Symbol.h"
```

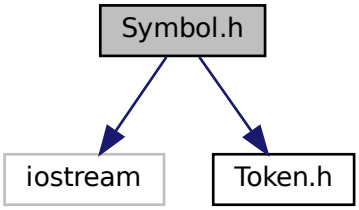
Wykres zależności załączania dla Symbol.cpp:



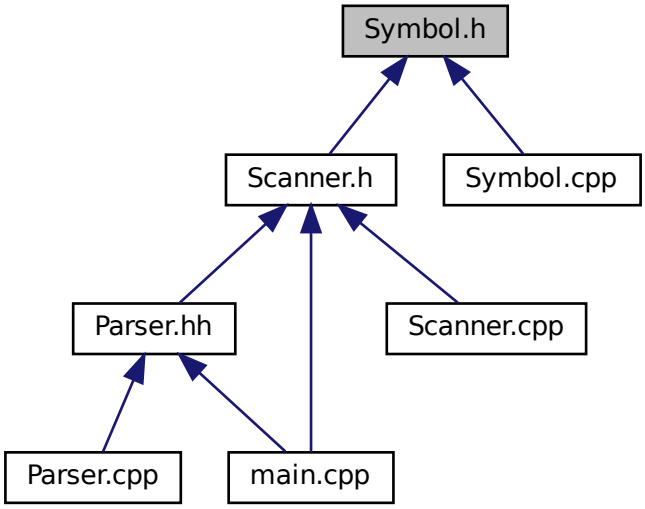
4.11 Dokumentacja pliku Symbol.h

```
#include <iostream>
#include "Token.h"
```

Wykres zależności załączania dla Symbol.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Struktury danych

- class Symbol

  Klasa Przechowująca dane (nazwę i token) - elemnt hashtable.

4.12 Symbol.h

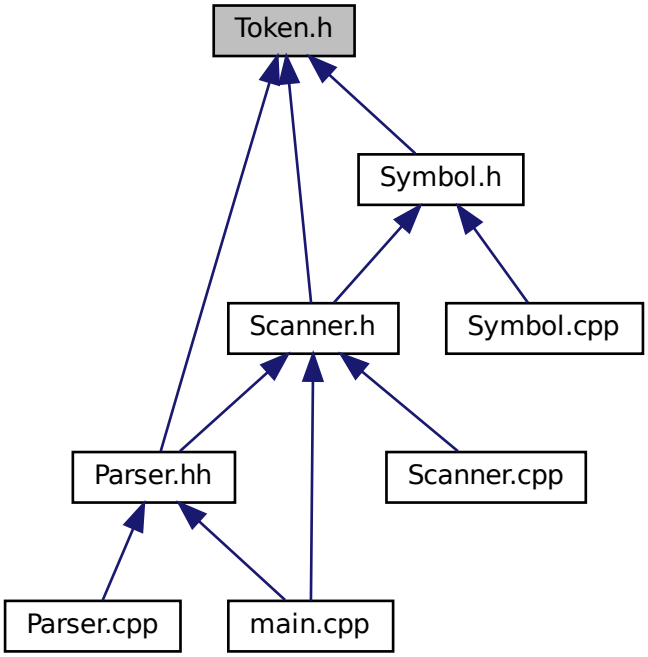
Idź do dokumentacji tego pliku.

```
1 #ifndef SYMENTRY_H
2 #define SYMENTRY_H
3 #include <iostream>
```

```
4 #include "Token.h"
5
12 class Symbol
13 {
14     public:
15         std::string name;
16         Token token;
17         Symbol();
18         Symbol(std::string name, Token token):name(name),token(token) {};
19         ~Symbol();
20
21 };
22 #endif
```

4.13 Dokumentacja pliku Token.h

Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Wyliczenia

- enum Token {
  ProgramSym , BeginSym , EndProgramSym , EndSym ,
  ReadSym , WriteSym , VarSym , IfSym ,
  ThenSym , WhileSym , DoSym , Id ,
  IntSym , IntLiteral , RealSym , RealLiteral ,
  Plus , Minus , Multiply , Devide ,
  Less , LessEq , Greater , GreaterEq ,
  Equal , Diffrent , LParen , RParen ,
  Assign , SemiColon , Comma , EofSym ,
  ErrorSym }

  Tokeny.

4.13.1 Dokumentacja typów wyliczanych

4.13.1.1 Token

```
enum Token
```

Tokeny.

Wartości wyliczeń

ProgramSym	"PROGRAM"
BeginSym	"BEGIN"
EndProgramSym	"END."
EndSym	"END"
ReadSym	"READ"
WriteSym	"WRITE"
VarSym	"VAR"
IfSym	"IF"
ThenSym	"THEN"
WhileSym	"WHILE"
DoSym	"DO"
Id	
IntSym	"INTEGER"
IntLiteral	
RealSym	"REAL"
RealLiteral	
Plus	"+"
Minus	"-"
Multiply	"∗"
Devide	"/"
Less	"<"
LessEq	"<="
Greater	">"
GreaterEq	">="
Equal	"="
Diffrent	"<>"
LParen	"("
RParen	")"
Assign	":="
SemiColon	";"
Comma	","
EofSym	
ErrorSym	

4.14 Token.h

Idź do dokumentacji tego pliku.

```
1 #ifndef TOKEN_H
2 #define TOKEN_H
7 enum Token
8 {
10     ProgramSym,
12     BeginSym,
14     EndProgramSym,
16     EndSym,
18     ReadSym,
20     WriteSym,
22     VarSym,
24     IfSym,
26     ThenSym,
28     WhileSym,
30     DoSym,
31     Id,
33     IntSym,
34     IntLiteral,
36     RealSym,
37     RealLiteral,
39     Plus,
41     Minus,
43     Multiply,
45     Devide,
47     Less,
49     LessEq,
51     Greater,
53     GreaterEq,
55     Equal,
57     Diffrent,
59     LParen,
61     RParen,
63     Assign,
65     SemiColon,
67     Comma,
68     EofSym,
69     ErrorSym
70 };
71
72 #endif // !TOKEN_H
```