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

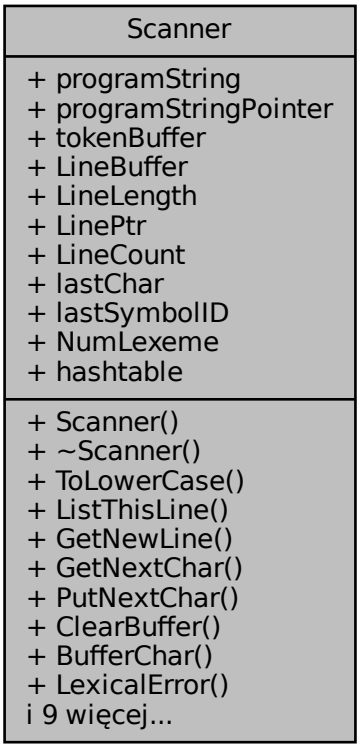
Dokumentacja struktur danych

3.1 Dokumentacja klasy 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.

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)
- std::unordered_map< int, Symbol > * hashtable

  hash tablica

3.1.1 Opis szczegółowy

Klasa realizująca funkcje skanera kodu.

Autor

Marek Pałdyna

3.1.2 Dokumentacja konstruktora i destruktora

3.1.2.1 Scanner()

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

3.1.2.2 ~Scanner()

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

3.1.3 Dokumentacja funkcji składowych

3.1.3.1 BufferChar()

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

Dodaje znak z parametru do bufora tokenBuffer.

Zobacz również

tokenBuffer

Parametry

c	znak do dodania do bufora

3.1.3.2 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

3.1.3.3 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

3.1.3.4 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

3.1.3.5 ClearBuffer()

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

Czyści bufor przeznaczony do bufforowania Tokenów.

3.1.3.6 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

3.1.3.7 GetNewLine()

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

get a new line from cached program

3.1.3.8 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

3.1.3.9 GetNextToken()

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

Znajduje kolejny Token w tekscie programu.

Zwraca

Token

3.1.3.10 LexicalError()

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

Wypisuje informacje o błędzie lexykalnym na podanym znaku.

Parametry

c	

3.1.3.11 ListSymbolTable()

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

Wypisuje wszystkie Symbole przechowywane w HashTable.

3.1.3.12 ListThisLine()

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

produce a listing of the source program

3.1.3.13 LoadKeywords()

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

Dodaje Słowa kluczowe do hashListy.

3.1.3.14 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

3.1.3.15 PutNextChar()

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

Zmniejsza znacznik pozycji zbuforowanego znaku.

3.1.3.16 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

3.1.3.17 ToLowerCase()

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

Convert characters to lower case characters.

3.1.4 Dokumentacja pól

3.1.4.1 hashtable

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

hash tablica

3.1.4.2 lastChar

```
char Scanner::lastChar
```

Ostatni zbuforowany znak.

3.1.4.3 lastSymbolID

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

Zmienna przechowująca id(w hashTable) ostatnio znaleziony symbol.

3.1.4.4 LineBuffer

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

używany do przechowywania kolejnych linii programu

3.1.4.5 LineCount

```
int Scanner::LineCount
```

Numer szeregowy aktualnej linii.

3.1.4.6 LineLength

```
int Scanner::LineLength
```

Przechowuje długość aktualnie zbuforowanej linii.

3.1.4.7 LinePtr

```
int Scanner::LinePtr
```

3.1.4.8 NumLexeme

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

Do tymczasowego przechowawania wartości zmiennej numerycznej (w postaci stringa)

3.1.4.9 programString

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

zacachowany kod programu.

3.1.4.10 programStringPointer

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

Przechowuje index kolejnego znaku do wczytania do buforuLini.

3.1.4.11 tokenBuffer

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

Buffor uzywany do znajdywania kolejnych tokenów.

Dokumentacja dla tej klasy została wygenerowana z plików:

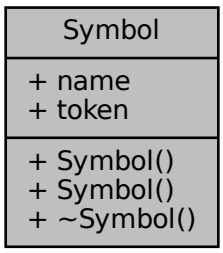
- Scanner.h
- Scanner.cpp

3.2 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

3.2.1 Opis szczegółowy

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

Autor

Marek Pałdyna

3.2.2 Dokumentacja konstruktora i destruktora

3.2.2.1 Symbol() [1/2]

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

3.2.2.2 Symbol() [2/2]

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

3.2.2.3 ∼Symbol()

```
Symbol::∼Symbol ( )
```

3.2.3 Dokumentacja pól

3.2.3.1 name

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

3.2.3.2 token

```
Token Symbol::token
```

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symbol.h
- Symbol.cpp

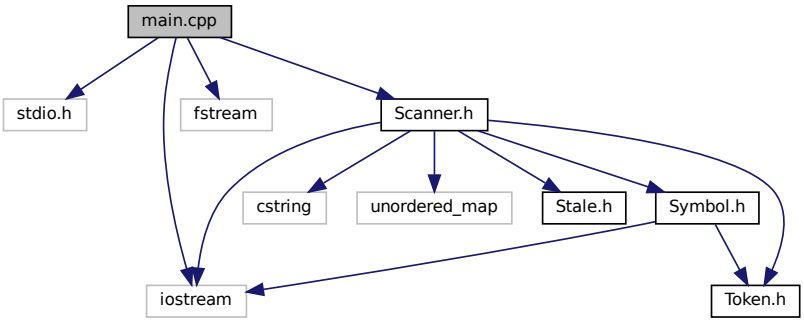
Chapter 4

Dokumentacja plików

4.1 Dokumentacja pliku main.cpp

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

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[] )
```

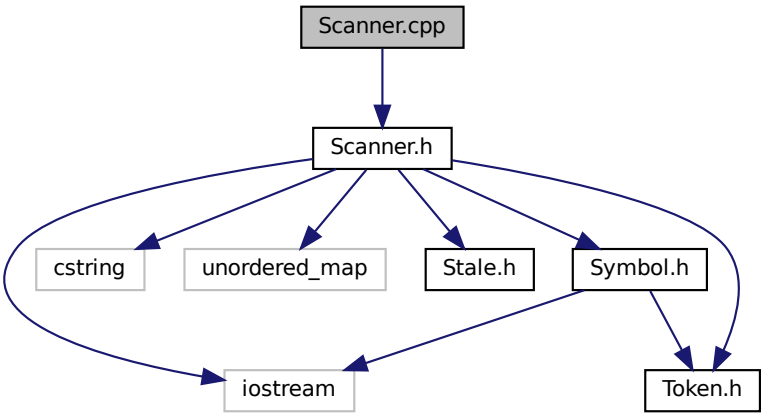
4.1.1.2 readFile()

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

4.2 Dokumentacja pliku Scanner.cpp

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

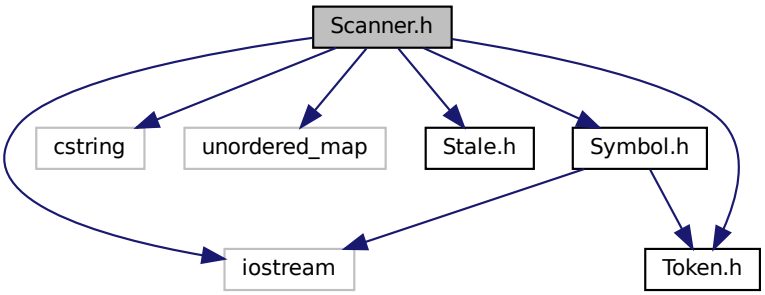
Wykres zależności załączania dla Scanner.cpp:



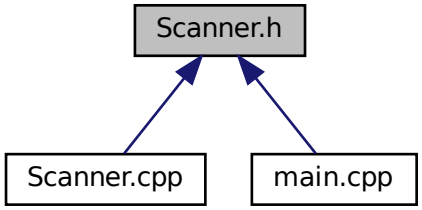
4.3 Dokumentacja pliku Scanner.h

```
#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.

4.4 Scanner.h

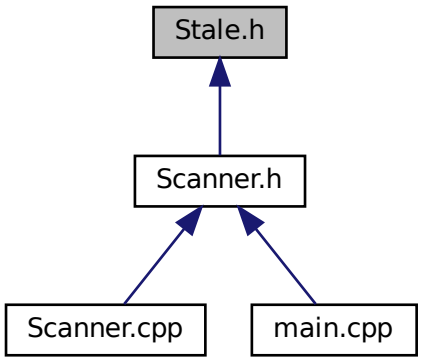
Idź do dokumentacji tego pliku.

```
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          //TODO czy to na pewno o to chodziło???
46          std::unordered_map<int, Symbol> *hashtable;
47
48          Scanner();
49          ~Scanner();
50
55          void ToLowerCase();
56
60          void ListThisLine();
61
66          void GetNewLine();
67
73          void GetNextChar(char *c);
74
79          void PutNextChar();
80
85          void ClearBuffer();
86
93          void BufferChar(char c);
99          void LexicalError(char c);
100
106          void BufferName(char c);
107
114          int BufferNumLiterals(char c);
115
121          Token CheckReserved();
122
128          Token GetNextToken();
129
135          void scan(std::string program);
136
144          int Enter(std::string S, Token Code);
145
150          void LoadKeywords();
151
156          void ListSymbolTable();
157
164          int LookUp(std::string S);
165 };
166 #endif
```

4.5 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 ∗/

4.5.1 Dokumentacja definicji

4.5.1.1 EOL

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

4.5.1.2 EOS

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

4.5.1.3 FALSE

```
#define FALSE 0
```

4.5.1.4 ID_STRING_LENGTH

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

4.5.1.5 MAX_LINE_LENGTH

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

4.5.1.6 MAX_SYMBOL

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

4.5.1.7 NONE

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

4.5.1.8 TOKEN_SIZE

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

4.5.1.9 TRUE

```
#define TRUE 1
```

4.6 Stale.h

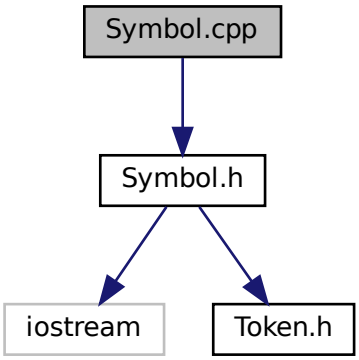
Idź do dokumentacji tego pliku.

```
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 #endif // __STALE_H__
```

4.7 Dokumentacja pliku Symbol.cpp

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

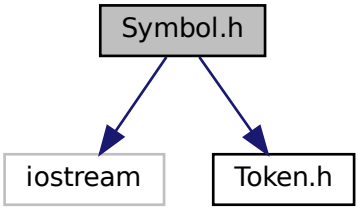
Wykres zależności załączania dla Symbol.cpp:



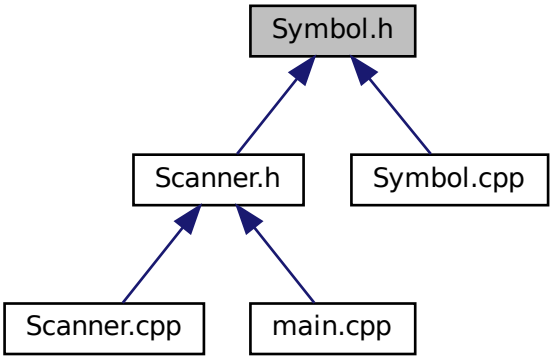
4.8 Dokumentacja pliku Symbol.h

```
#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.

4.9 Symbol.h

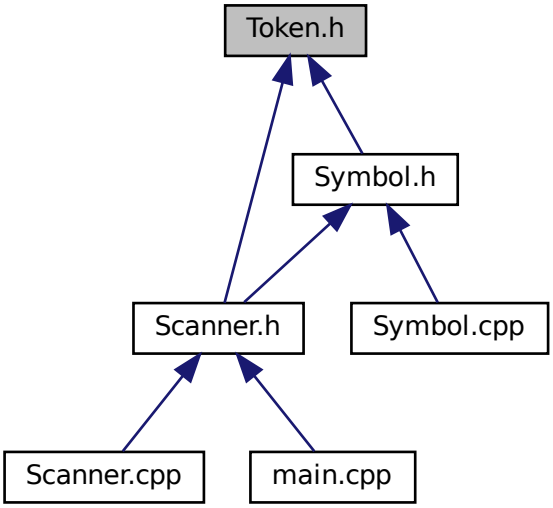
Idź do dokumentacji tego pliku.

```
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
```

4.10 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.

4.10.1 Dokumentacja typów wyliczanych

4.10.1.1 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	

4.11 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
```