ZPR - szkielet

Specyfika języka i zachowania programu

Specyfika języka i zachowania programu opisana jest w pliku TKOM-projekt-wstępny. Była ona aktualizowana po implementacji parsera.

Wymagania projektu

- cmake - wersja minimalna ustawiona na 3.22
- c++ 20 (osobiście korzystam z clang 19)
- wykorzystywane biblioteki:
- spdlog
- fmt
- boost (korzystam z wersji 1.88) wraz z elementami:
  - unit_test_framework
  - program_options

Wykorzystywane narzędzia

- budowanie projektu i zarządzanie zależnościami: cmake
- autoformatowanie: clang-format z opcjami opisanymi w pliku `.clang-format`
  - bazowy styl: Google
  - indentacja:
    - spacje zamiast tabów
    - wcięcia na 4 spacje
  - długość linii: 120
  - wskaźniki ustawione z lewej strony
  - funkcje jednolinijkow: tylko puste
  - klamry

Uruchomienie programu

Wywołanie jest możliwe z następującymi flagami:

- `--help` or `-h` - wyświetlenie informacji pomocniczej (wołane domyślnie)
- `--verbose` or `-v` - uruchomienie programu z większą informacją pokazywaną użytkownikowi
- `--stdin` or `-s` - program wczytywany z wejścia standardowego - w przypadku nie zapewnienia tej flagi program oczekuje podania pliku
  Przed uruchomieniem budujemy projekt:

```
# z katalogu głównego projektu
cmake --build build
```

Przykład uruchomienia programu z pliku:

```
/build/TKM_Compiler -v ./example_programs/nth.tkm
```

Przykład uruchomienia programu z wejścia standardowego

```
./build/TKM_Compiler -v -s <<EOF
> def main() -> int {
>     print("Hello World!");
>     return 0;
> }
> EOF
```

Output:

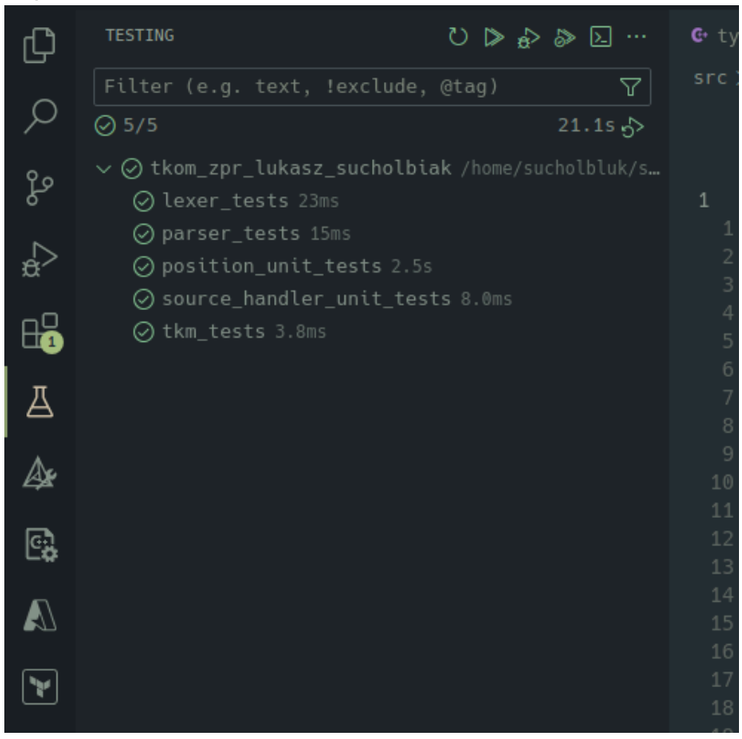
```
[info] [LEXER] Created Token: Token(TokenType::T_DEF,Position(1,1))
[info] [LEXER] Created Token: Token(TokenType::T_IDENTIFIER,Position(1,5),"main")
```

```
[info] [LEXER] Created Token: Token(TokenType::T_L_PAREN,Position(1,9))
[info] [LEXER] Created Token: Token(TokenType::T_R_PAREN,Position(1,10))
[info] [LEXER] Created Token: Token(TokenType::T_ARROW,Position(1,12))
[info] [LEXER] Created Token: Token(TokenType::T_INT,Position(1,15))
[info] [LEXER] Created Token: Token(TokenType::T_L_BRACE,Position(1,19))
[info] [LEXER] Created Token: Token(TokenType::T_IDENTIFIER,Position(2,5),"print")
[info] [LEXER] Created Token: Token(TokenType::T_L_PAREN,Position(2,10))
[info] [LEXER] Created Token: Token(TokenType::T_LITERAL_STRING,Position(2,11),"Hello World!")
[info] [LEXER] Created Token: Token(TokenType::T_R_PAREN,Position(2,25))
[info] [LEXER] Created Token: Token(TokenType::T_SEMICOLON,Position(2,26))
[info] [LEXER] Created Token: Token(TokenType::T_RETURN,Position(3,5))
[info] [LEXER] Created Token: Token(TokenType::T_LITERAL_INT,Position(3,12),0)
[info] [LEXER] Created Token: Token(TokenType::T_SEMICOLON,Position(3,13))
[info] [LEXER] Created Token: Token(TokenType::T_R_BRACE,Position(4,1))
[info] [LEXER] Created Token: Token(TokenType::T_EOF,Position(5,1))
[Program] <0x60332685f590> at:[1:1]
  |_[FunctionDefinition] <0x60332685f560> at:[1:1]
    |_[FunctionSignature] <0x60332685e400> at:[1:1] identifier: main, type: function<none:int>
    |_[CodeBlock] <0x60332685f7b0> at:[1:19]
      |_[ExpressionStatement] <0x60332685f780> at:[2:5]
        |_[FunctionCall] <0x603326860140> at:[2:5]
          |_Calle:
            |_[Identifier] <0x60332685f930> at:[2:5]name:print
          |_Arguments:
            |_[LiteralString] <0x60332685f730> at:[2:11]type:string, value:"Hello World!"
      |_[ReturnStatement] <0x60332685faf0> at:[3:5]
```

W folderze example_programs znajdują sie przykładowe programy, które mogą być podane na wejście

Testy

W projekcie dostępne są testy jednostkowe. Aby je uruchomić należy:

1. Zbudować projekt `cmake --build build`
2. Przejść do katalogu z testami: `cd ./build/tests/`
3. Uruchomić testy: `ctest`
   Alternatywnie, przy korzystaniu z Visual Studio Code – testy można uruchomić z edytora:
   

Kompilacja w Visual Studio pod windowsem

Przy projekcie pracowałem na linuxie, przy próbie kompilacji pod windowsem pojawia się konflikt nazw TokenType – windows sdk definiuje własny TokenType. Do czasu dokumentacji finalnej zostanie to rozwiązane.

Generowanie dokumentacji z kodu

- narzędzie: `doxygen`

1. Wygenerowanie dokumentacji: `doxygen Doxyfile`
2. Przykład otworzenia dokumentacji: `xdg-open ./doxydoc/html/index.html`