SUEC – AN EDITOR AND INTERPRETER FOR PSEUDOCODE

Student: Mihai PÎȚU

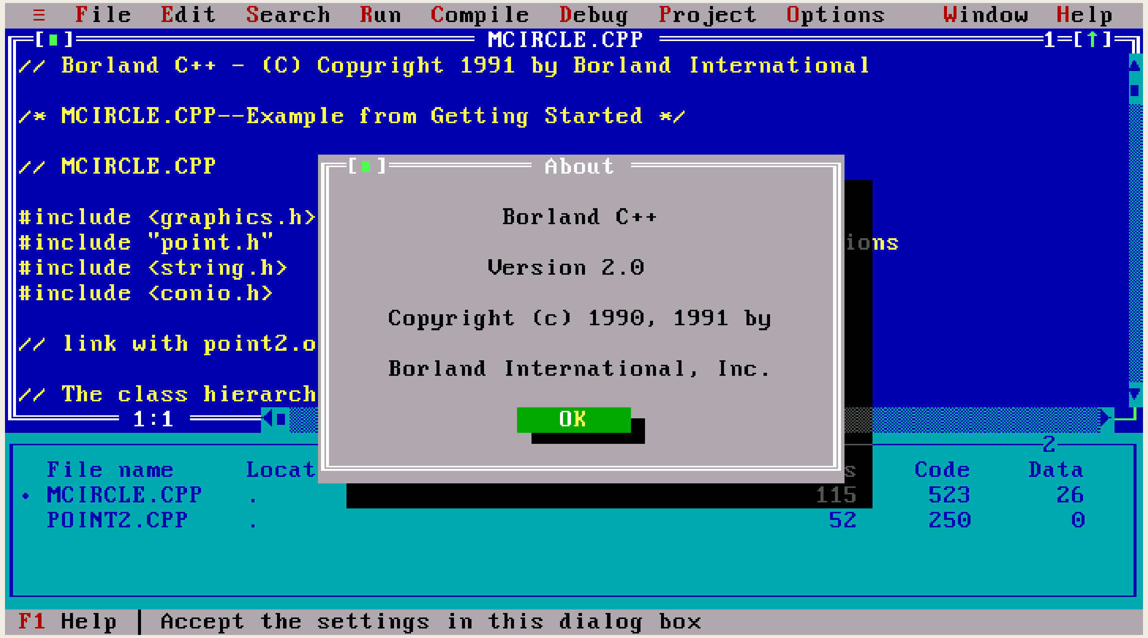
Supervisor: Assoc. Prof. Dr. Eng. Emil Ștefan Chifu

Content

- Introduction
- Motivation
- Theoretical Foundation
- Implementation and Design
- Personal Contribution
- Conclusion
- Bibliography

Introduction

- Computer science is taught in schools in Romania ~30 years, starting with 5th/9th grade.
- Teachers use pseudocode for explaining algorithms and basic concepts – close to natural language.
- Working on laboratories, C/C++/Pascal was used for writing code based on the pseudocode.


≡ File Edit Search Run Compile Debug Project Options Window Help
MCIRCLE.CPP
1:1
About
OK

```
// Borland C++ - (C) Copyright 1991 by Borland International

/* MCIRCLE.CPP--Example from Getting Started */

// MCIRCLE.CPP

#include <graphics.h>
#include "point.h"
#include <string.h>
#include <conio.h>

// link with point2.o

// The class hierarch
```

Borland C++

Version 2.0

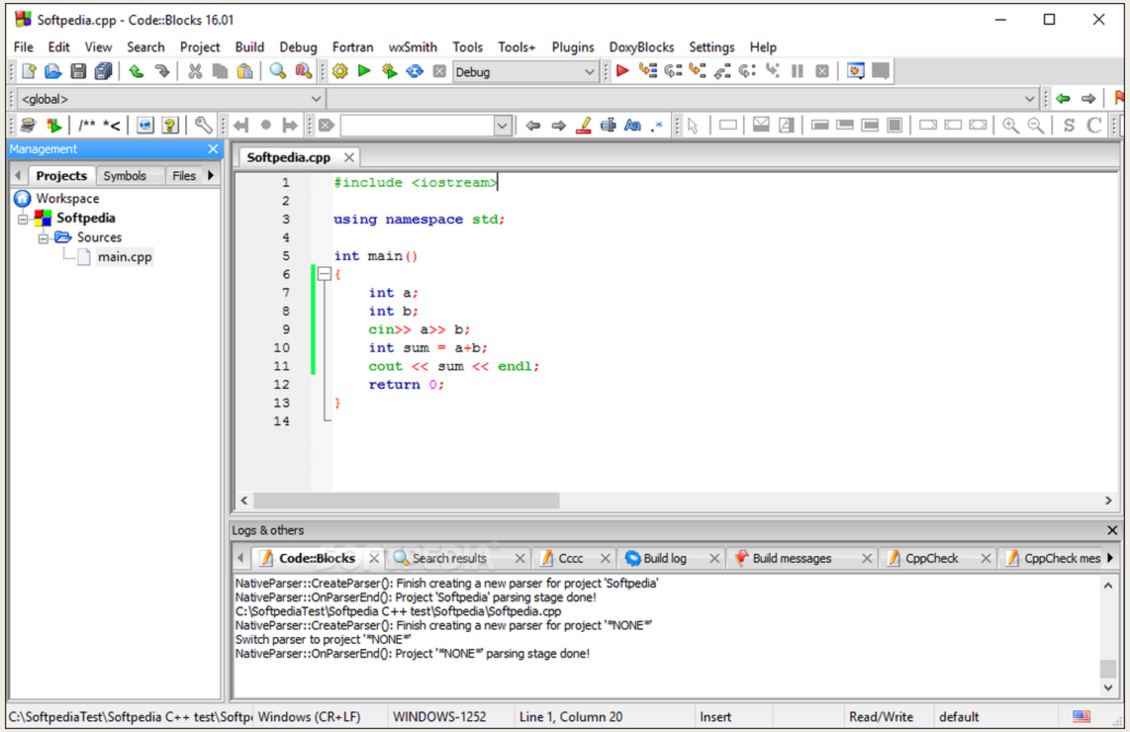
Copyright (c) 1990, 1991 by

Borland International, Inc.

```
  File name      Locat                   s     Code      Data
• MCIRCLE.CPP    .                     115      523        26
  POINT2.CPP     .                      52      250         0
```

F1 Help | Accept the settings in this dialog box

Borland C++ [1]



```
#include <iostream>

using namespace std;

int main()
{
    int a;
    int b;
    cin>> a>> b;
    int sum = a+b;
    cout << sum << endl;
    return 0;
}

```

```
NativeParser::CreateParser(): Finish creating a new parser for project 'Softpedia'
NativeParser::OnParserEnd(): Project 'Softpedia' parsing stage done!
C:\SoftpediaTest\Softpedia C++ test\Softpedia\Softpedia.cpp
NativeParser::CreateParser(): Finish creating a new parser for project '*NONE*'
Switch parser to project '*NONE*'
NativeParser::OnParserEnd(): Project '*NONE*' parsing stage done!
```

C:\SoftpediaTest\Softpedia C++ test\Softp | Windows (CR+LF) | WINDOWS-1252 | Line 1, Column 20 | Insert | Read/Write | default

Code :: Blocks [2]

Motivation

- Helping students to understand programming concepts with a user friendly editor and easy to learn programming language.
- Dabbling with tutorials and educational apps – since recent events.

Thesis Project

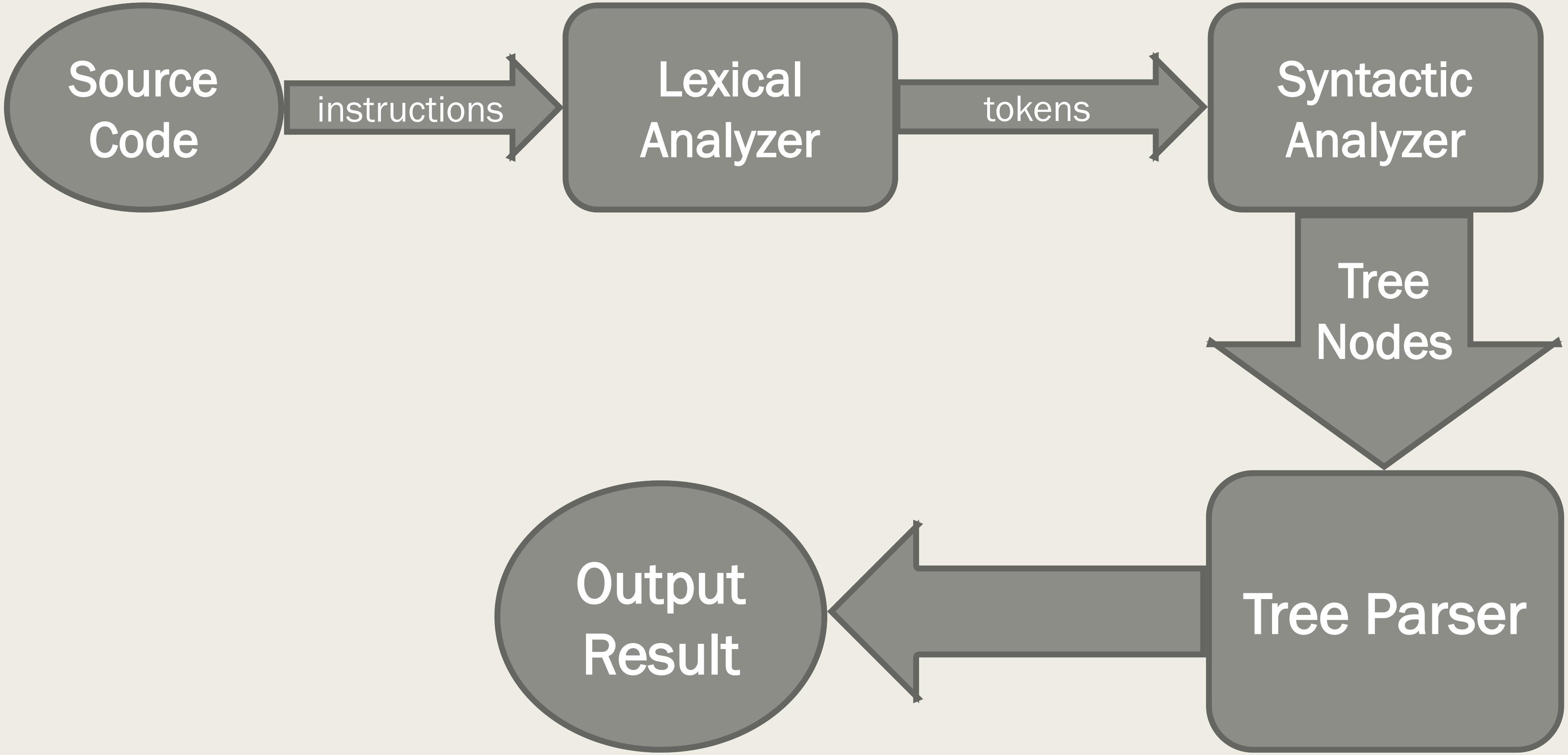
- Purpose
  - *Develop an application used by anyone new to programming to edit and compile pseudocode.*
- Objectives
  - ✓ *Develop a user-friendly editor.*
  - ✓ *Create a programming language similar to pseudocode.*

Theoretical Foundation

- Interpreter
  - *Lexical Analyzer*
  - *Syntactic Analyzer*
  - *Tree Parser*

Interpreter

- A computer program that directly executes instructions written in a programming language – no need for a machine code to compile.
- Historically, Lisp had the first interpreter, but there are interpreters written and run alongside compilers (e.g. for Fortran, Cobol, C).
- Strategies for program execution:
  - *Parsing the code and perform its behavior directly*
  - *Translating the code into an effiicient intermediate state and execute that state.*


Source Code
instructions
Lexical Analyzer
tokens
Syntactic Analyzer
Tree Nodes
Tree Parser
Output Result

Lexical Analyzer [5]

- A computer program that is designed to parse the source code into manageable tokens.
- Contains a series of rules defined as regular expressions
  - *A specific word/sentence (e.g. “int”, “do while” etc.)*
  - *A set of symbols (e.g. “[0-9]+”)*
- All the rules return tokens (optionally, adding raw data) – sent to the syntactic analyzer

Syntactic Analyzer [5]

- Computer program that defines the “grammar” of a programming language.
- Contains rules defined with tokens:
  - *Internal tokens(denoted with lowercase letters) – non-terminals*
  - *External tokens(denoted with uppercase letters) – terminals – from lexical analyzer*
- These rules can return
  - *The result directly*
  - *A tree node with the operation*

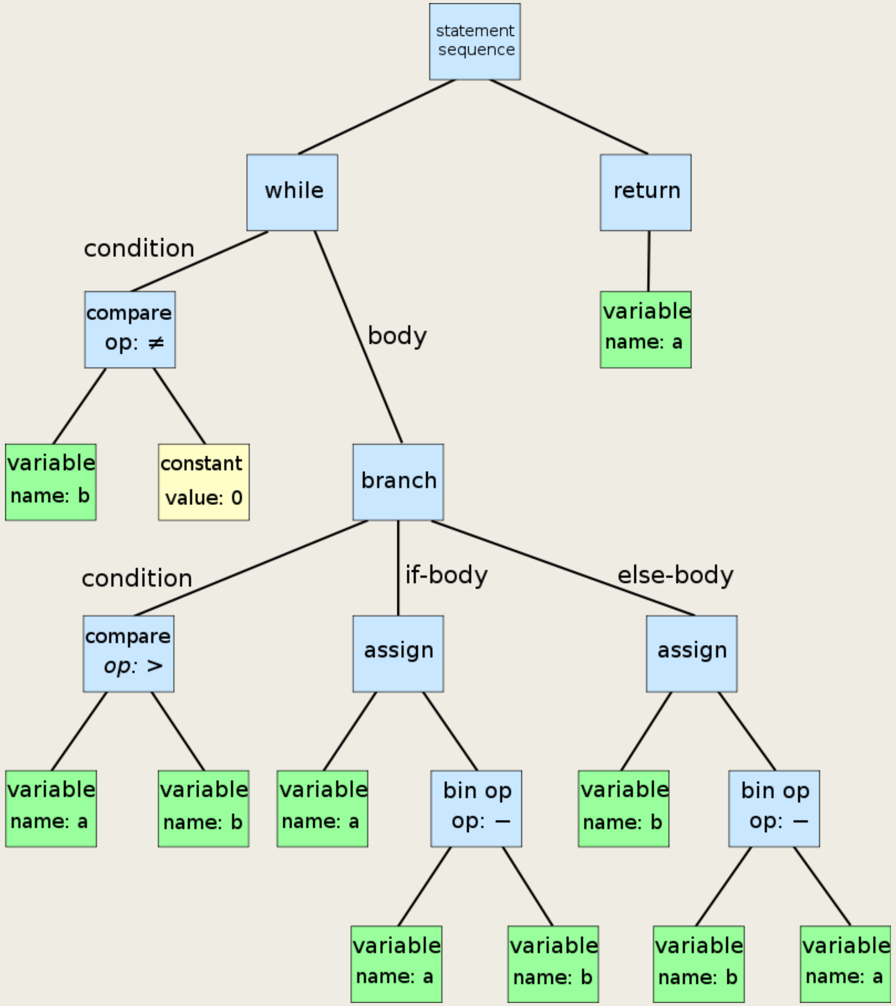
Tree Parser

- A program that parses a syntax tree generated from the syntactic analyzer.
- Syntax tree = representation of the statements as a tree
- A node is defined as having a type value, operation and (sometimes) the raw data associated
- Two ways of parsing the tree[3]
  - ***Top-down parsing*** *– "primordial soup" approach; searching the parse tree top-down(from the highest level), rewriting the rules of the formal grammar (a prefix search)*
  - ***Bottom-up parsing*** *– starting from the lowest level (leaves of the tree) and build the result, finishing with root node (postfix search)*

Syntax tree - example

- Euclidean algorithm representation for the code:

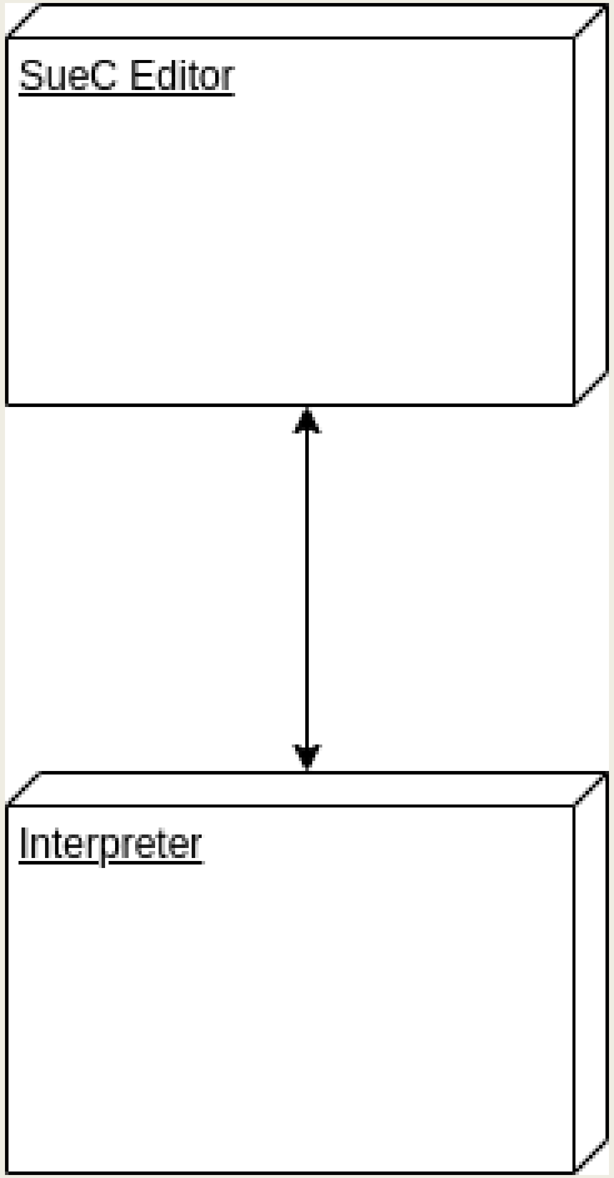
```
while b!=0
        if a>b
                a=a-b
        else
                b=b-a
 return a
```



Abstract Syntax Tree [4]

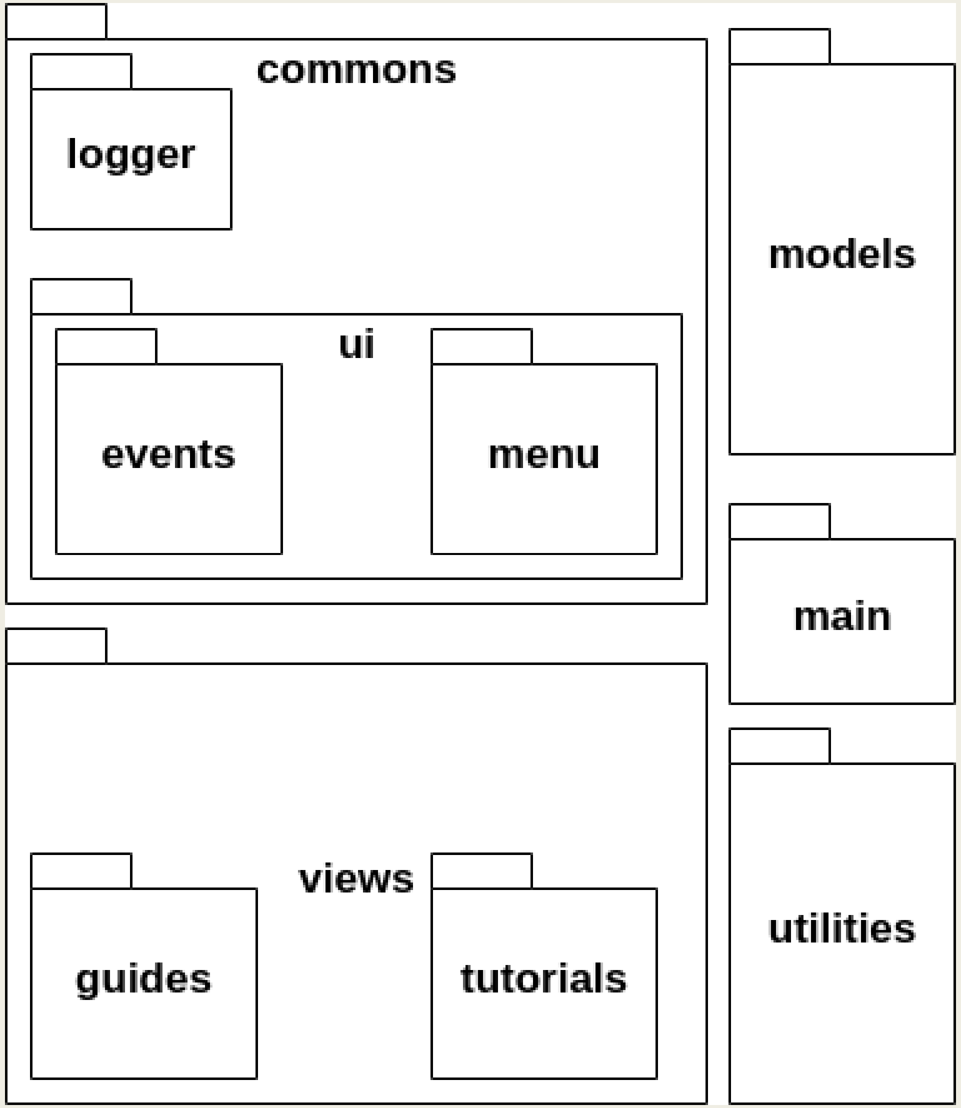
Implementation and Design

- Project is split in two main components
  - *SueC Editor Application* – Java Desktop Application
  - *Interpreter* – A C Executable program accessed by the editor

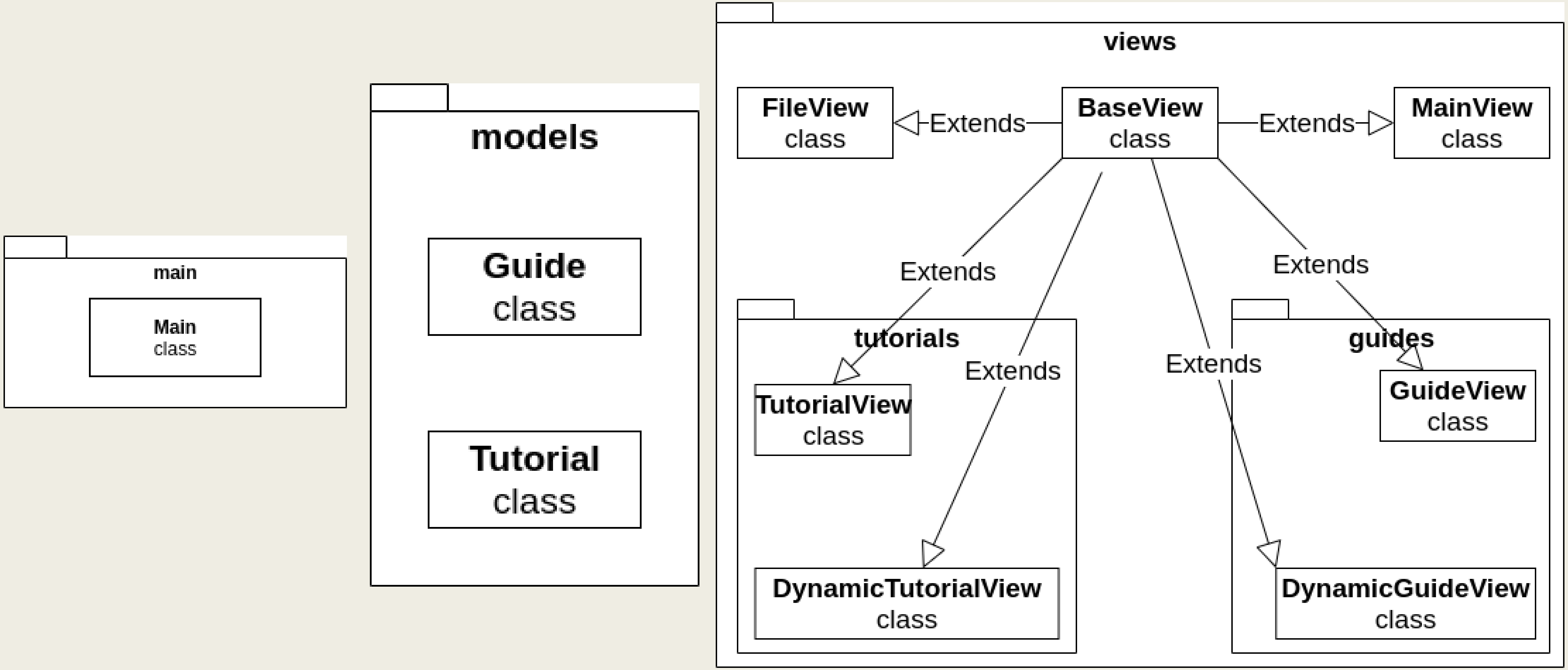


Editor Application

- Java Swing Desktop Application
- Adapted structure of Model-View-Controller (MVC) Pattern
  - *models & views* defined directly
  - *controller* defined internally in the views – Swing structure
- Loose coupling – few methods used specific classes for grouping the operations
- High cohesion – classes are defined for specific tasks that are not performed in other classes



Editor Application – *main, models, views* class diagram



main

Main class

models

Guide class

Tutorial class

views

FileView class

BaseView class

MainView class

Extends

Extends

Extends

Extends

Extends

Extends

tutorials

TutorialView class

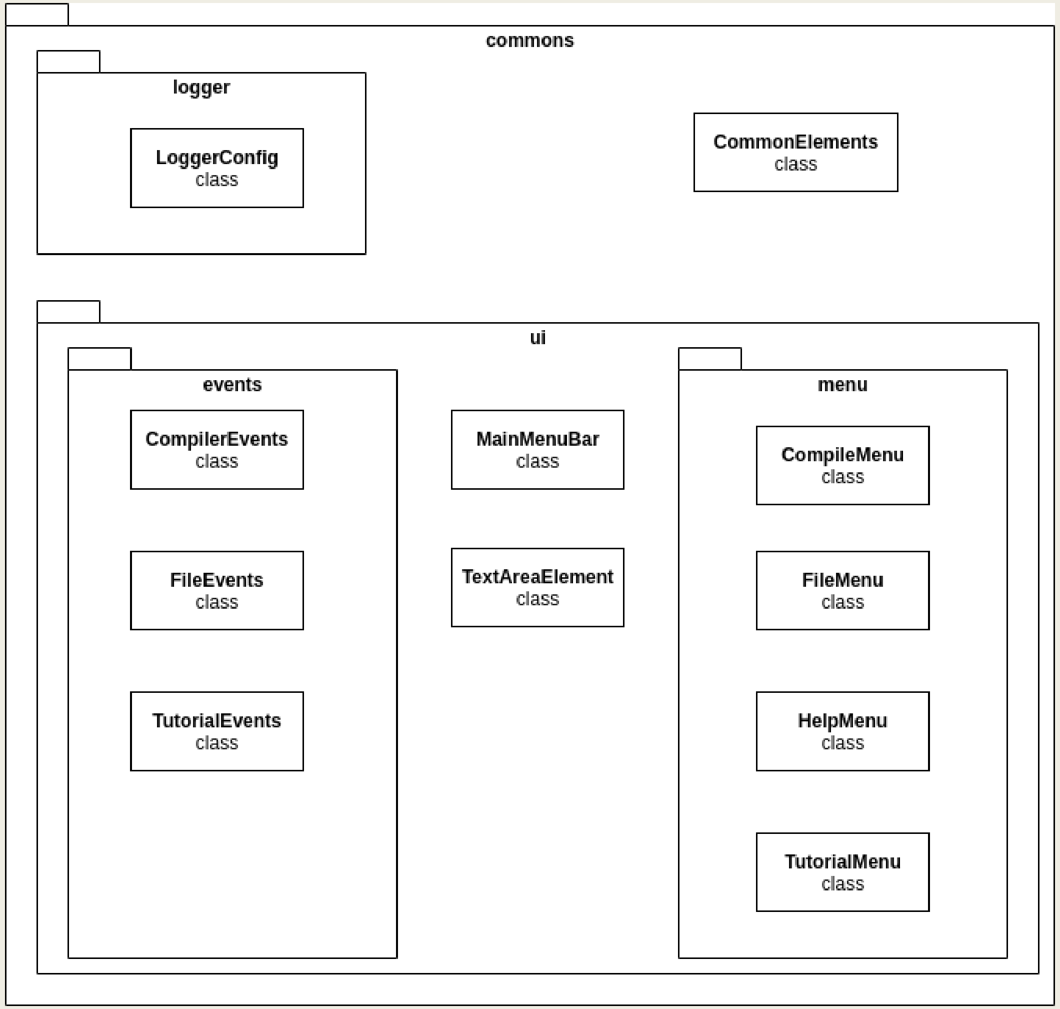
DynamicTutorialView class

guides

GuideView class

DynamicGuideView class

Editor Application – *commons, utilities* class diagram



commons

logger

LoggerConfig
class

CommonElements
class

ui

events

CompilerEvents
class

FileEvents
class

TutorialEvents
class

MainMenuBar
class

TextAreaElement
class

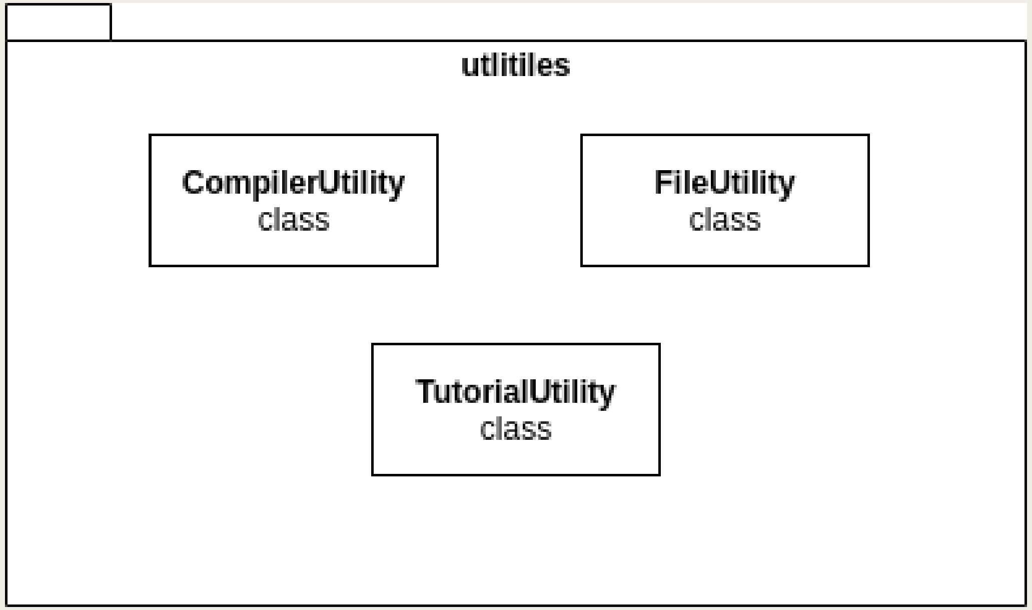
menu

CompileMenu
class

FileMenu
class

HelpMenu
class

TutorialMenu
class

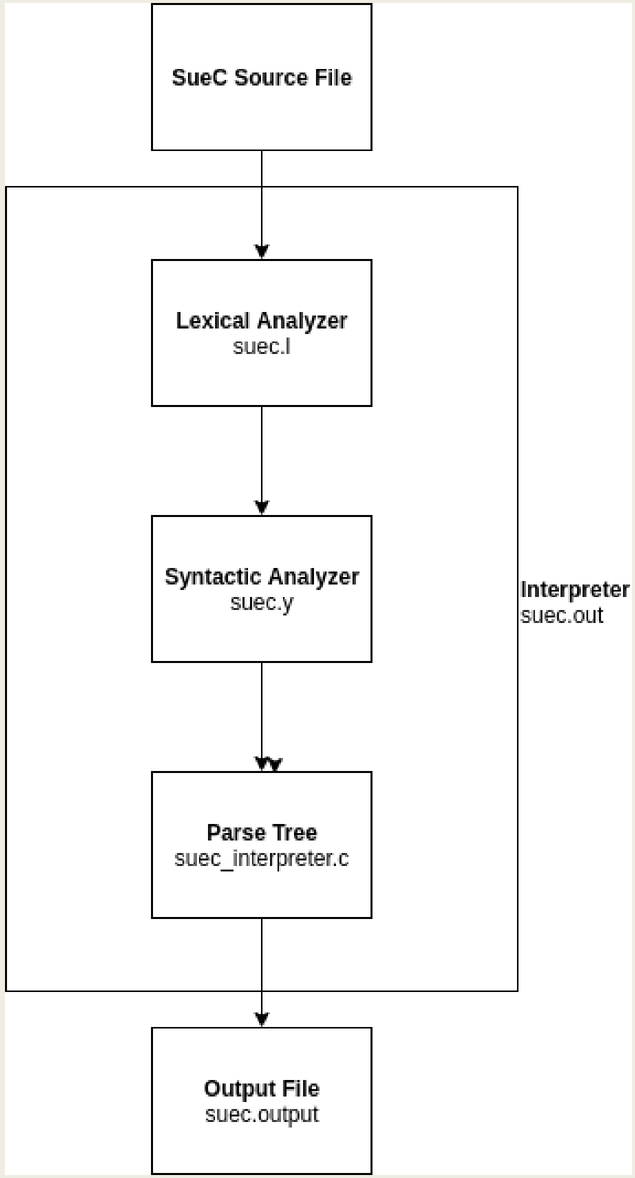


utlitiles

CompilerUtility
class

FileUtility
class

TutorialUtility
class



Interpreter

- Executable C program that contains the definition of the rules of the programming language
- Lexical analyzer – Lex source file
- Syntactic analyzer – Yacc source file
- Tree Parser – contains the parsing method and definition of the nodes
  - *constNodeType* for constants
  - *idNodeType* for identifiers
  - *operNodeType* for operators

Programming Language – pseudocode(SueC)

- Similar structures as in C/C++ and Python
- Close to the natural language and pseudocode
- Example:

  -
    ```
    write “Hello”;
    ```

  -
    ```
    int a;
    a = 5;
     write a*2 ;
    ```

Personal Contribution

- Designing the programming language’s interpreter based on a template ([6]) and laboratory work done at FLT and TD laboratories
- Designing and implementing the editor application to access the interpreter
- Implementing tutorial and guide menus and operations for learning to implement the programming language

Conclusion

- Upon following all the required stages for this project: choosing the theme, documenting, implementing, designing, testing and bug-fixing, I have created a functional application that:
  - *Contains an interpreter designed for pseudocode*
  - *Creates, edits and saves SueC source code files than can be compiled in the editor*
  - *Contains tutorials and guides for learning the SueC programming language*

Further developments

- Editor Application
  - *Support for cross-platform (currently working only on Linux systems)*
  - *Adding more tutorials & guides*
- Programming language
  - *Support for arrays and matrices*
  - *Support for floating point data types*

Bibliography

- [1] Borland C++, [Online]. Available: https://winworldpc.com/product/borland-c/30
- [2] Code::Blocks, [Online]. Available: https://en.wikipedia.org/wiki/Code::Blocks
- [3] Parsing – Types of parsers, [Online]. Available: https://en.wikipedia.org/wiki/Parsing#Types_of_parsers
- [4] Abstract Syntax Tree, [Online]. Available: https://en.wikipedia.org/wiki/Abstract_syntax_tree
- [5] Lex – A Lexical Anaylzer Generator, M.E. Lesk and E. Schmidt, [Online]. Available: https://wolfram.schneider.org/bsd/7thEdManVol2/lex/lex.pdf
- [6] Lex & Yac Tutorial, Tom Niemann, [Online]. Available: https://cse.iitkgp.ac.in/~bivasm/notes/LexAndYaccTutorial.pdf

Thank you for your attention!