ECOTE lab tasks

Task 1. Aut1

Write a program able to input regular expressions, then constructing NFA using Thompson algorithm, and checking if input strings are generated by this expression (working on NFA).

Task 2. Aut2

Write a program able to input regular expressions, then constructing directly DFA using syntax tree for the expression. Show syntax tree and all functions. The program should check if input strings are generated by this expression.

Task 3. Aut3

Write a program able to input regular expressions, then constructing NFA using Thompson algorithm, converting NFA into DFA and checking if input strings are generated by this expression (working on DFA).

Task 4. Parser1

Write a Top-Down parser with backtracking. Output also the phases of the syntax tree development.

Task 5. Parser2

Write a Bottom-Up parser with backtracking. Output also the phases of the syntax tree development.

Task 6. CD1

Write a program reading a subset of C++, C# or Java code and constructing class inheritance tree (including also interface realizations, if appropriate). Output the tree. Specify appropriate code limitations (define syntax).

Task 7. CD2

Write a program reading a subset of C++, C# or Java code and constructing a graph of method's class dependency. Dependency between class is if a class calls method from other class. Output the dependencies. Specify appropriate code limitations (define syntax).

Task 8. CD3

Write a program reading a subset of C++, C# or Java code and constructing a graph of data member's dependency. Dependency between class is if a class uses data member from other class. Output the dependencies. Specify appropriate code limitations (define syntax).

Task 9. CD4

Write a program reading a subset of C++ code. Transform a code with the friendship dependencies between functions, methods and classes into the code without friendship. Specify appropriate code limitations (define syntax).

Task 10. SA1

Write a program reading a subset of C++, C# or Java code and constructing a graph of all possible nested blocks. Functions called by pointer can be omitted. Specify appropriate code limitations (define syntax).

Task 11. SA2

Write a program reading a subset of a selected language C, C++, C# , Java, Python code and showing all variables, which are not used. Specify appropriate code limitations (define syntax).

Task 12. SA3

Write a program reading a subset of a selected language C, C++, C# , Java, Python code and showing all functions/methods, constructors, which are not used. Specify appropriate code limitations (define syntax).

Task 13. SA4

Write a program reading a subset of a selected language C, C++, C# , Java, Python code and showing all classes and interfaces which are not used. Specify appropriate code limitations (define syntax.

Task 14. SA5

Write a program reading a subset of a selected language C, C++, C# , Java, Python code and indicating parts of code that which are not accessible (only according to the knowledge of the static analysis). Specify appropriate code limitations (define syntax).

Task 15. SA6

Substitute all strings of characters (“xxxxxx”) in a C program with constancies. Constancies should be defined with #define directive and stored in a common header file. Apply the same name for identical strings used in different locations. Take into account different comments in a program.

Task 16. SA7

Write a program that reads a subset of C-like code (in several files) and generates a cross-reference: locations of definitions, declarations and usage of all identifiers. The program verifies consistency of the files similarly, as a linker does, i.e. a function could be defined in one file but its body could be presented in another file. Specify appropriate code limitations (define syntax).

Task 17. Test1

Write a program reading a subset of C++, C# , Java, Python code and creating skeletons of unit tests for classes and their methods. Specify appropriate code limitations (define syntax).

Task 18. Test2

Write a program reading a subset of C++, C# or Java code and identifying all pairs <def,use> for all variables.
<def (X)i , use(X)j> for variable X, means that a following program run is possible:
value of variable X was assigned at ith line and used at jth line, and between the value was not changed. Specify appropriate code limitations (define syntax).

Task 19. Test3

Write a program reading a subset of C, C++, Java or C# code and identifying all pairs <def,p-use> for all variables. <def (X)i , use(X)j> for variable X, means that a following program run is possible:
value of variable X was assigned at ith line and used in a condition in jth line, and between the value was not changed. Specify appropriate code limitations (define syntax).

Task 20. Test4

Write a program that reads a subset of C code and generates a set of testing data. Using this data we should be able to cover all decisions in the code. Each decision (e.g. an instruction **if, switch, while)** should be executed at least as many times as the number of possible conclusions of the decision. For example, if (a>b) i1(); else i2();
One execution if for instruction i1(); and second execution for i2(). Define code syntax and take reasonable assumptions. Specify appropriate code limitations.

Task 21. HTML1

Write an analyzer of the HTML code, that deletes any "decorations" (e.g. colors, big pictures, complex graphics, frames, big fonts, etc). Define the syntax of deleted elements.

Task 22. HTML2

Write a program that transforms a HTML file into an equivalent file which is more suitable to be displayed in a mobile device.

Task 23. HTML3

Write a program that merges two similar HTML files. Make assumptions about resolving of conflicts.

Task 24. HTML4

Write a program that decides abut equivalence of two HTML files. Select a HTML scope to be taken into account . Specify equivalence rules.
In general equivalence is not identity. The files could have the same content, structure, etc, but there might be different font sizes, colors, …

Task 25. Refact1

For a subset of a selected programming language C++, C#, Java, Python write a program, that performs a simple refactoring transformations, e.g. move a method from one class to another, rename a class and all dependences. Specify appropriate code limitations (define syntax).

Task 26. Refact2

For a subset of a selected programming language C++, C#, Java, Python write a program, that performs a simple refactoring transformations, e.g move a field, a method in an inheritance hierarchy (pull up, push down). Specify appropriate code limitations (define syntax).

Task 27. Refact3

For a subset of a selected programming language C++, C#, Java, Python write a program, that performs a simple refactoring transformations, e.g. extract a method , rename a field/a method and all dependences. Specify appropriate code limitations (define syntax).

Task 28. MacroG1

Write a macrogenerator with “key” parameters:
#MDEF name
definition
#MEND
$ is a parameter dominator. $ZZ is for the substitution of parameter with key=ZZ.
Call:
#MCALL list of parameters
Eg.
#MCALL name ZZ=abc YY=xyz
In the body of macrodefinition a call of another macro and another macro definition can be included.

Task 29. MacroG2

Write a macrogenerator
.REPT expression
definition
.ENDM
This is also a call. The value of *Expression* defines how many times *definition* will be generated.
Propose the syntax for *Expression* and *definition*.
In the body of macrodefinition a call of another macro and another macro definition can be included.

Task 30. MacroG4

Write a macrogenerator with positional parameters.
#MDEF name
definition
#MEND
A parameter denominator is $. E.g. $1 - for the substitution of the first parameter.
Macrocall: #MCALL name list of current parameters
In the body of macrodefinition a call of another macro and another macro definition can be included.

Task 31. Opt1

Write a program that reads expressions and optimizes them. Minimize the number of usage of elements of arrays and calling of these functions (methods) that have no side effects. Define syntax of expressions.
E.g. A[i,j] * B + A[i,j] <==> A[i,j]*(B + 1) these expressions are equivalent, can be optimized

Fun(5) + Fun(5) / NewFun(x) <==> Fun(5) *(1 + 1/NewFun(x))
Above expressions are equivalent only if Fun() has no side effects, otherwise the expressions are not equivalent

Task 32. Opt2

Write a program that reads expressions and optimizes them. Minimize the number of access to array elements. Using a new (generated) variable, substitute usage of the same elements of an array with the same value. Define syntax of expressions.
Example, Input:
A = 5* tab[i,j]; B = 7.8 - tab[i,j]; i = t; a = tab[i,j];
Output:
__gen1= tab[i,j]; A=5*__gen1; B = 7.8 * __gen1; i = t; a = tab[i,j];
Where __gen1 is a name of a generated helper variable.

Task 33. Opt3

Write a program that reads a subset of C-like code and optimizes it. Remove unnecessary instructions from a loop body. Loops can be nested. Specify appropriate code limitations (define syntax).
Example, Input:

```
for (i=1; i<100; i++)
        { a=5; b[i] = c[i] *a; }
```

Output:

```
a=5;
for (i=1; i<100; i++)
        { b[i] = c[i] *a; }
```

Task 34. Opt4

Write a program that reads a subset of C-like code and optimizes it. Remove unnecessary calculations from the loop body. New (generated) variables can be used. Define syntax of the code. Loops can be nested, therefore, a variable could be moved before a loop or many loops.
Example, Input:

```
for (i=1; i<100; i++)
        { b[i] = c[i] *a * 135.8; }
```

Output:

```
float __gen1 = a*135.8;
for (i=1; i<100; i++)
        { b[i] = c[i] *___gen1; }
```

Where __gen1 is a name of a generated helper variable.

Task 35. Opt5

Decide about equivalence of expressions. Expressions can contain at least 4 basic operators (+,-,*,/), power operator, parenthesis, simple variables, integer and float numbers, function calls. An Expression used as a function call parameter does include another function call. All functions can have side effects. Integer is not equivalent to a float number. Define syntax of expressions.
Examples: a * (5 + b*2) is equivalent to 5 * a + b * a * b
fun(5) * (a + b) is not equivalent to a * fun(5) + b * fun(5)

Task 36. Bib1

Write a converter between at least three different forms of bibliography (e.g. bibtex, some IEEE https://www.ieee.org/conferences/publishing/templates.html,

some Springer https://www.springer.com/us/authors-editors/conference-proceedings/conference-proceedings-guidelines, Harvard notation,etc.). Define a syntax.

Task 37. Latex1

Write a translator of a LaTeX subset to a selected text format. Take into account the basic document structure (e.g. p. based on document or article, headers, sections, author, title, paragraphs, etc). Define a syntax of the subset.

Task 38. Latex2

Write a translator of a LaTeX subset to a selected text format. Take into account LaTeX tables. Define a syntax of the subset.

Task 39. Lambda1

Write a program that converts a code with lambda expressions into a code without lambda expressions. The input/output code is a subset of Java, C# or C++. Define a syntax of the subset.

Task 40. Lambda2

Write a program that converts a code without lambda expressions into a code with lambda expressions. The input/output code is a subset of Java, C# or C++. Define a syntax of the subset.

Task 41. Python1

Write a translator of a subset of the Python language to another selected programming language (C++,Java, C#). The translation is not based on Python annotations. Define a syntax of the subset.

Task 42. Python2

Write a translator of a subset of the Python language to another selected programming language (C++,Java, C#). Use Python annotations for type identification. Define a syntax of the subset.

Task 43. Python3

Write a translator of a subset of a selected programming language (C++,Java, C#) to the Python language. Define a syntax of the subset.