SIMPLIFICATION OF COMPILER DESIGN COURSE TEACHING USING CONCEPT MAPS

Venkatesan Subramanian[1] and Kalaivany Natarajan[2]

[1]Department of Information Technology, Indian Institute of Information Technology, Allahabad, India.
[2]Australia

ABSTRACT

The students of undergraduate expect simple, interactive and understandable method of teaching. But, in many universities, instructors simply follow text books and solve the examples given in it. In such case, students cannot learn anything other than the text book contents defined in the syllabus. This method of teaching will not motivate students to understand the subject in depth unless the industrial and practical applications of the subject explained. Once students do not show interest, instructor also loses the interest which leads to poor performance of students. Hence, it is mandatory to re-design the teaching method especially for the undergraduate students. This paper use concept maps to simplify teaching and learning of a compiler design subject with assignments and problems related to research and industry.

KEYWORDS

Compiler Design, Concept Map, Computer Science, Education

1. INTRODUCTION

The IT stalwarts are citing that most of the engineering graduates are not fit for hiring. The major reason for such issue is, while studying the undergraduate engineering, students does not understand subjects in depth. Students simply read the important topics of the subjects' from text books and pass the exam. Some do solve questions based on the same text book. We but can categorize the students in to four based on their expectation from the subject study as shown in figure 1.

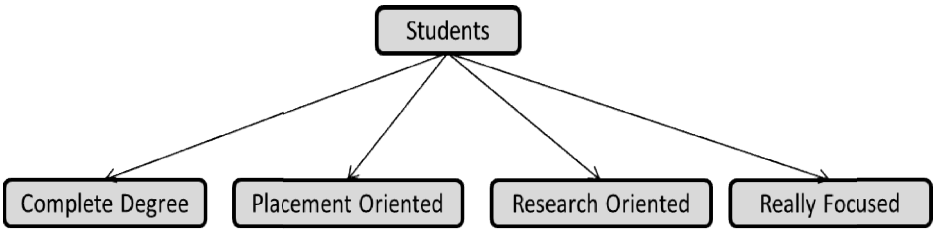

Figure 1. Categorization of Students

The first category of students studies the subject just to pass and get the degree. Second category of students concentrates only on the subjects those helps in industry placement and they consider other subjects are the one just to complete the degree. Third category of students interested to do research will concentrate on subjects, which they feel it is required for research. Fourth category of students really focuses on all subjects and study thoroughly. Usually, students of first and last

Dhinaharan Nagamalai et al. (Eds) : ACSIT, SIPM, ICITE, ITCA - 2019
pp. 91–102, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.90309

category will be very less and other two categories will have more students especially the second category that is placement oriented. All first three categories of students will realize the issue of not concentrating on all subjects while they do job or research because subjects are linked with each other and required while solving research or industry problems. However, it is very late to realize. Hence, students should concentrate on all subjects while they study. It can be achieved only when students are motivated towards the subject and the instructor change the traditional text book based teaching method. To understand the students' objectives towards each subject of the course, we conducted the survey with the following three questions.

a) What are the subjects you are interested in?
b) What are the subjects you feel required for placement?
c) What are the subjects you feel required for research?

There are 94 third year undergraduate students participated in the survey. The survey results are shown in table 1. The Interested column shows the percentage of students having interest towards the subject. The Placement and Research column shows the percentage of students feel that the subjects are required respectively to get industry placement and do research.

Table 1. Survey Results of students opinion on subjects (in percentage)

	Interested	Placement	Research
Introduction to Programming	52	45	2
Data Structure	95	81	17
Algorithms	95	72	36
Database Management System	86	65	10
Automata	11	29	33
Compiler Design	14	14	19
Software Engineering	21	14	1
Operating System	82	44	32
Computer Architecture	26	19	26
Programming Principles	16	12	11
Networks	52	37	9
Artificial Intelligence	20	48	83

We can understand from the survey results that students are not having enough interest on subjects such as Automata, Software Engineering, Computer Architecture and Compiler Design. But an experienced technologists and teacher knows the importance of these subjects. Hence, it is necessary to encourage and motivate students for these subjects by changing the teaching method. In this paper, we concentrate on Compiler Design course, which is considered by all categories of students as one of the complex and very less productive. Students keen to study Programming are not showing interest towards Compiler Design. They feel that Introduction to Programming helps in understanding programming language and writing programs but Compiler Design is not having any productive outcome. We need to beat this notion by highlighting important of compiler design course. Hence, it is essential to re-design the teaching style of Compiler Design course to motivate the students. It is possible when we have the proper lesson and concept plan in the beginning.

In this paper, we propose concept maps for Compiler Design course by the innovative ideas [2 and 3]. The concept map is discussed in overall and phase wise in the paper. We have listed core concepts of compiler in each phase and provided the problems to be discussed in the class room and assignments for students as homework.

The remaining paper is organized as follows: Section 2 discusses the existing works, Section 3 briefly discusses the overall concept map of Compiler Design course, Section 4 discusses about the phases of compiler with concept map. Section 5, has the discussion of teaching method. Finally, Section 6 concludes the paper with the directions of future work.

2. LITERATURE REVIEW

In the past, different approaches were proposed to motivate the students to concentrate on the compiler design course. Akim Demaille et al [12] introduced set of compiler construction tools for educational projects. This makes learning and evaluation easy. Li Xu and Fred G. Martin [13] proposed a Chirp system that provides a realistic and engaging environment to teach compiler course. However, concepts and its relationship may not reach the students. Tyson R. Henry [14] proposed a Game Programming Language (GPL) based teaching to motivate students towards the compiler project. Elizabeth White et al. [15] proposed an approach that enables students of compiler course to examine and experiment with a real compiler without becoming overwhelmed by complexity. S. R. Vegdahl [16] proposed visualization tool to teach compiler design to bring the interest of students on the subject. Marjan Mernik and Viljem Zumer [17] proposed a software tool called LISA for learning and conceptual understanding of compiler construction in an efficient, direct, and long lasting way. Henry D. Shapiro and M. Dennis Mickunas [18] replaced the term project on compiler design with several smaller, independent, programming assignments for better understanding and to motivate students. Martin Ruckert [19] argues that teaching compiler with unusual programming language is a good choice. Divya Kundra and Ashish Sureka [20] [21] discussed about case based teaching and learning of compiler design and proposed case studies for different concepts to make learning easier and interesting. Somya Sangal et al [22] proposed a Parsing Algorithms Visualization Tool (PAVT) tool to teach the process of parsing algorithms. Learners can visualise the intermediate steps of the parsing algorithm through the tool.

Even though game based, case based and tool based teaching makes learning easy and interesting, concepts and its relationship with some assignments related to current research and industry requirement should be discussed for motivation and long lasting remembrance. Also students should prepare the assignments for each concept based on their understanding. The concept map [2 and 3] based teaching is well tested with different subjects and proved that students can easily understand the concept. Hence, we propose concept map for teaching compiler design course. A concept map will simplify the teaching and make students understand the subject. In addition, we propose that for each concept; problems based on latest research and industry requirements to be discussed to motivate the students.

3. COMPILER CONCEPT MAP

Compiler is software that reads a program written in one language and translates it into another (target) or assembly, later to machine language without changing the semantics [1]. Interpreter is an alternate for compiler however it translates and executes directly instead of converting the complete code to the target language. Compiler does various operations to translate an input to output language. These operations are categorized into different phases. Figure 2 shows the overall concept map of the compiler with concepts and its relations.

It is important for an undergraduate students studying Compiler Design course to understand how the compiler is designed and program is getting translated. The concept map in figure 2 is designed with the motive to encourage the students to deeply focus on this subject. The concept map includes the input from the pre-processor and pre-requisite for this subject such as

instruction set, Context free and context sensitive grammar, regular expression, finite state machine and push down automata. The proposed concept map gives the complete overview of compiler design course and relationship among the concepts.

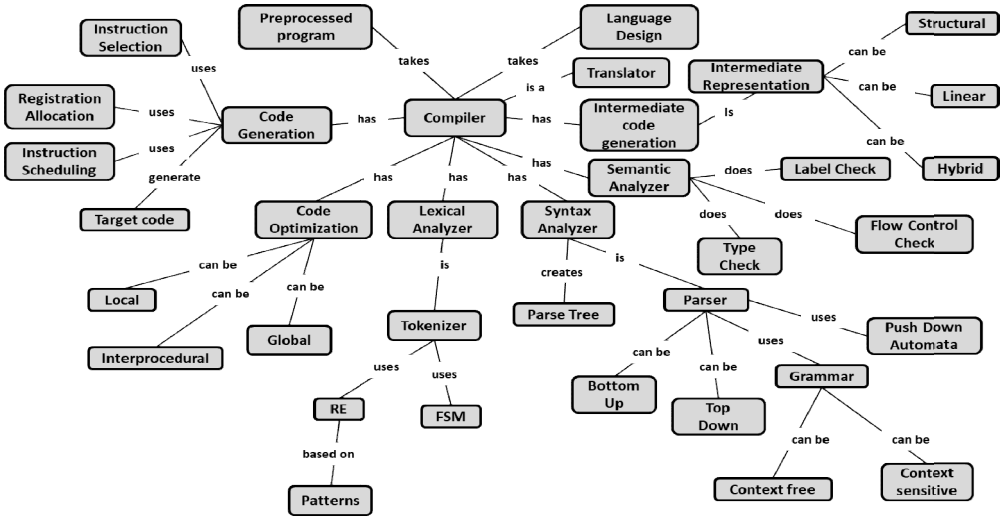

Figure 2. Basic concept map of Compiler

4. PHASES OF COMPILER

Compiler has six phases however there are references with seven phases dividing optimization into two parts that is machine dependent and independent optimization. In this paper, we have considered machine independent optimization in the code optimization phase and machine dependent optimization in code generation phase itself. In the beginning, it is required to explain to the students that what is the need for six phases instead of 1 or 10 phases? The major reason is that if we have all operations in one phase then it will increase the computational time and having more phases such as 10 or 15 will have the redundant process. Compiler cannot be efficient if we have 1 or 10 phases. Each phase of the compiler with its concept map is discussed in the following.

4.1. Lexical Analysis

Lexical analysis is the first phase of the compiler that takes the pre-processed program as input and produces tokens as output. The concept map for the lexical analysis phase is shown in figure 3. The important concepts to be discussed in lexical analysis phase are Tokenization, Buffering, Finite State Machine, Regular Expression and Symbol Table and obviously how these concepts are interlinked to generate the tokens. To tokenize the input programs, lexical analyser utilizes the pre-defined regular expression of the programming language and recognizes using the finite state machine. Lexical analyser reads the program characters from the buffer one by one to recognize the tokens. Instructor should also discuss the difference of single and pairs of buffer.

Even though, students studied regular expression and finite state machine in the pre-requisite subject Automata theory, the concepts need to be revised with an example of programming language patterns. In the classroom, instructor may solve the following problems for the better understanding of each concept.

- Create the regular expression for C Language variables.
- Construct the finite state machine for C variables using the regular expression generated in the previous problem.
- Take the hello world C program as input and recognize the tokens using single and double buffering. Assume buffer size as 10 bytes and show the advantage of double buffering over single or greater than two buffering.

In additional students should be given assignments and problems on each concept to solve.

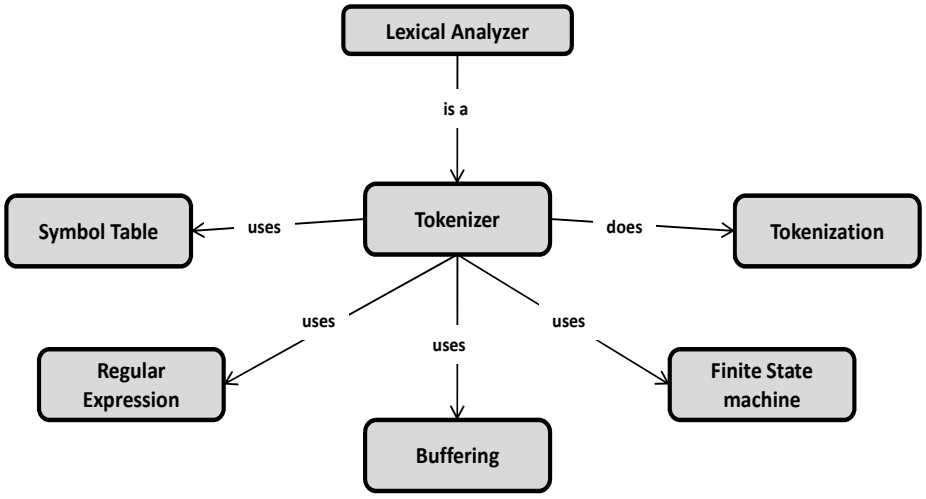

Figure 3. Concept map of Lexical Analyser

4.2. Syntax Analysis

Syntax Analysis takes the input as tokens from the lexical phase and produces the syntax tree, which will be used by the semantic phase. The concept map of syntax analyzer is shown in figure 4. Task of the syntax analyzer is to check whether the syntax present in the program is part of the programming language or not. To do this, Syntax Analyzer uses parser with Finite State Machine, Push down Automata and Context Free and Sensitive Grammar. Parser can be of universal, top down or bottom up approach and it will be chosen based on the developer requirement. The LL (Left-to-right, Leftmost derivation) and LR (Left-to-right, Rightmost derivation) parsers work only on the unambiguous grammar to parse in linear time. For LL parser, the unambiguous, deterministic and non-left recursive grammar will be taken as input and computes the first and follow. Using the first and follow, the parsing table will be constructed. In case of the LR parser, the finite state machine based item set for the unambiguous grammar will be generated and parsing table will be constructed from it.

The LL and LR parser takes the token from the lexical phase, parsing table and utilizes the push down automata to check the syntax and in parallel it produces the syntax. Instructor need to briefly discuss about the universal parsers like Earley's and CYK, which can take any type of grammar and parse it however the complexity of syntax validation will be more. Instructor need to solve one problem for each concept to make the student understand it. As a simple case, following problems can be solved in the class using the arithmetic operation grammar in figure 5.

- Eliminate Left recursion and compute the first & follow for the grammar.
- Compute the LR Automaton for the grammar.
- Compute LL and LR table using the first & follow and LR Automaton output respectively.
- Parse the input "a * b + c" through LL, LR and CYK parser using the parsing table computed in the previous assignment.

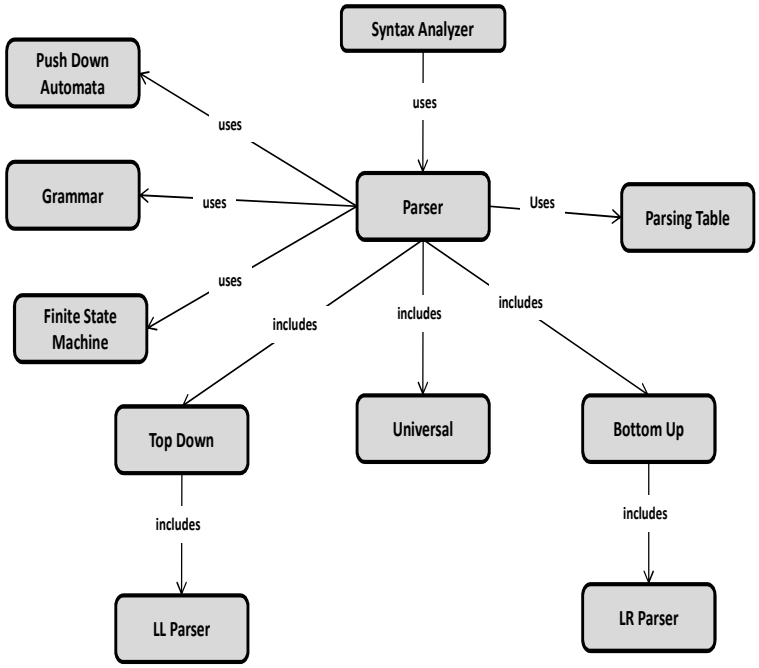

Figure 4. Concept map of Syntax Analyser

E- > E+T | E-T | T
T - > T*F | T/F | F
F - > Int

Figure 5. Sample Grammar for Arithmetic Operation

4.3 Semantic Analysis

Figure 6 shows the concept map of the semantic analyser. It takes the parse tree as input from the syntax analyzer and checks the semantics of the program such as type matching, parameter matching, label check, etc. This layer uses the attribute grammar or direct method to verify the semantics. The concepts in the semantic analyser are the Symbol Table, Attribute Grammars, Semantic check, Syntax Directed Translation and Definition. Attribute grammars can be represented using the Syntax Directed Translation (SDT) or Syntax Directed Definition (SDD) and use Dependency Graph at the time of evaluation to maintain the order. Semantic analyzer utilizes the symbol table to check the semantics of the program. Many high level programming languages do not have the attribute grammars for semantic check and it checks the semantics using the symbol table.

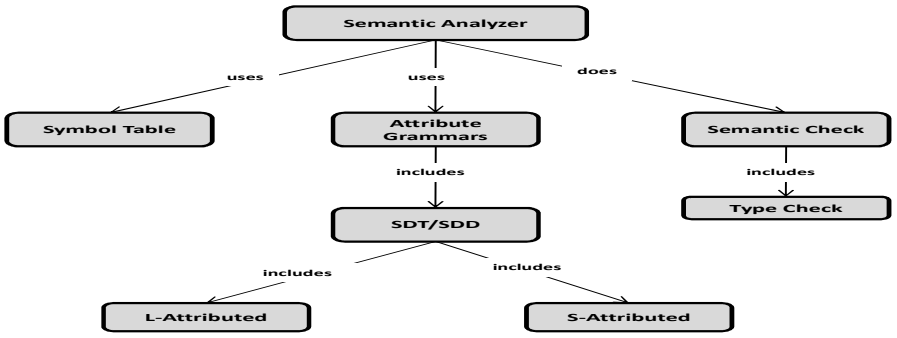

Figure 6. Concept map of Semantic Analyser

Instructor can verify the semantics of the snippet given in figure 7 using the C language constraints and show every step of the process in the class for better understanding

```
func(int a, float b):
int d, e
d = 0
e = a / round(b)
if (e > 5)  {
        printf("%d", d)
}
```

Figure 7. Sample snippet for semantic check

4.4. Intermediate Code Generation (ICG)

Figure 8 shows the concept map of the intermediate code generation phase of the compiler. ICG takes the input as syntax tree from the semantic analyzer and provides the Intermediate Representation. Intermediate Representation can be of structural, linear or hybrid. In most of the programming languages, linear representation is used and in that three address code is mostly preferred with any storage representations such as quadruple, triple and indirect triple. This phase may use the syntax directed translation to convert into three address code.

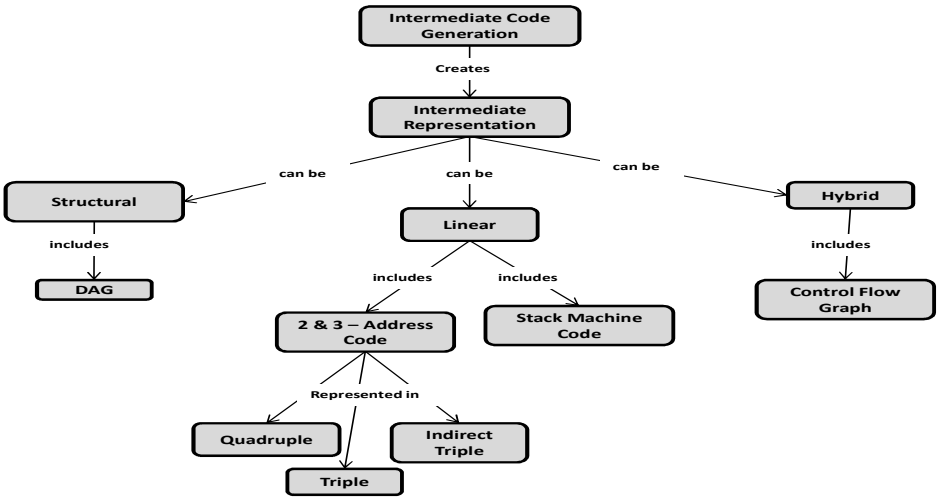

Figure 8. Concept map of Intermediate Code Generation

As an example, instructor can take the bubble sort program and generate the Direct Acyclic Graph (DAG) and three address code for it.

4.5. Code Optimization

Code Optimization is to make the program run efficiently after compilation. It helps in optimizing the time and space complexity of the program without changing the semantics. Code optimization can be done locally, globally or inter-procedural and it can be machine dependent or independent. Figure 9 shows the concept map of the code optimization phase. The different optimization concepts are Copy Propagation, Dead code elimination, Code Motion, Global Common Sub expressions, constant folding, loop optimization, unreachable code elimination, etc. Intermediate Representations from the previous phase will be grouped as blocks and then optimization will be applied locally and globally. In the classroom, instructor has to show one example for each optimization concept to make the concepts clear for students.

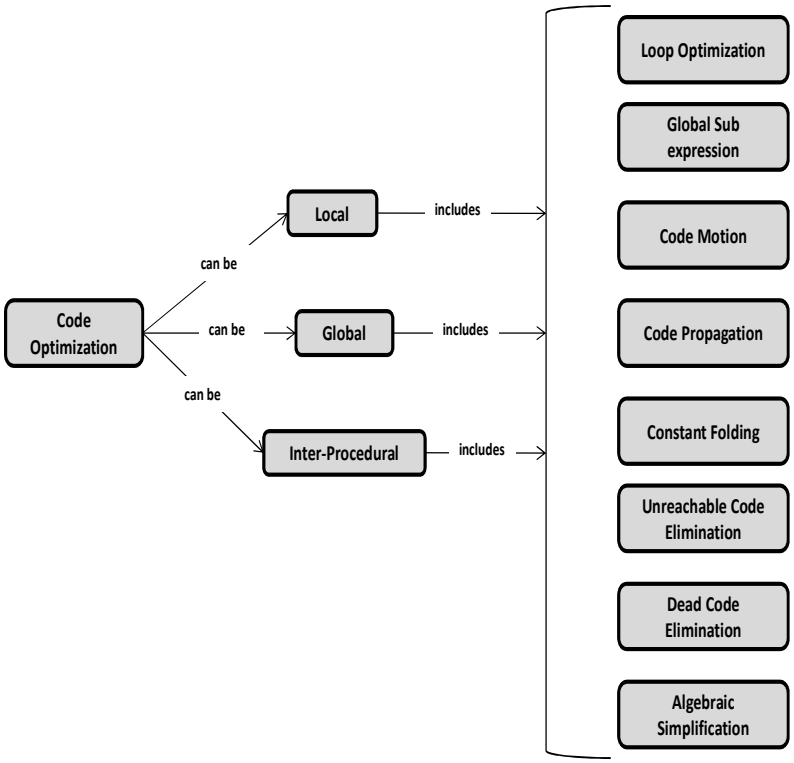

Figure 9. Concept map of Code Optimization

4.6. Code Generation

Code Generation is the final phase of the compiler that takes the optimized code and generates the target code. Target code may be another High level language code or Assembly level language code. To generate the assembly level, it is important to consider the Instruction Set, Register Allocation and Instruction Scheduling. The Next use information and Basic block concepts supports in register allocation for the target code. Figure 10 shows the concept map of the code generation.

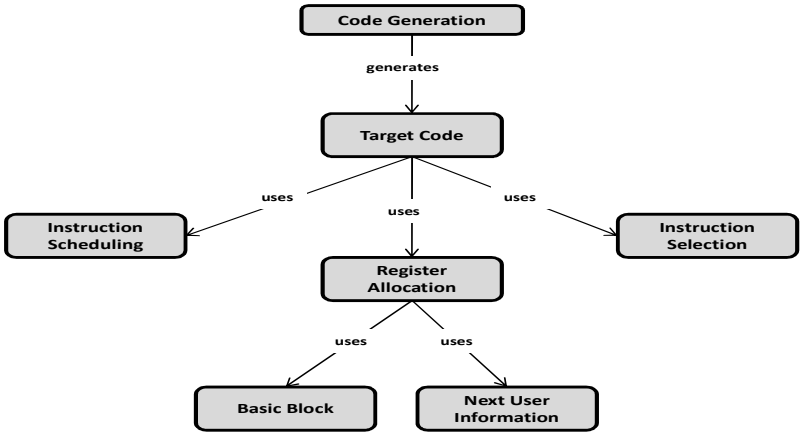

Figure 10. Concept map of Code Generation

Instruction set will be based on Reduced Instruction Set Computer (RISC) or Complex Instruction Set Computer (CISC) or Micro-op (mix of CISC and RISC). Since CISC architecture is used in popular Intel processors, CISC instruction set can be taken for solving the example in classroom. However, students should be asked to generate the target code for all architecture.

5. DISCUSSION

In the beginning of the Compiler Design course, instructor need to describe the course through the overall concept map, which includes the pre-requisites and new concepts students are going to learn. A concept map with the relationship among concepts can motivate the students to focus on the course. Every concept in the phase wise concept map should be discussed in the class with example problems that may be from the text book. In addition, the current research works and the industry products related to each concept should be briefly discussed to increase the interest of students. However, there should be assignments to the students on each concept from the recent research work and industry developments. Table 2 shows the sample assignments and problems related to each phase. Since students solve the research and industry oriented problems or assignments, they will be more interested on the course. Some assignments in table 2 especially research oriented assignments can be solved referring the literature cited.

In addition to the regular assignments, students should be asked to develop a compiler considering Classroom Object Oriented Language (COOL) [4] and Decaf [5]. This will help students to clearly understand the working process of each phase of the compiler. In case Compiler Design course contains the laboratory component then it should be used as Compiler design laboratory instead of programming laboratory.

Table 2. Assignments and Problems for each phase of the compiler

Concept	**General Problem**	**Industry oriented assignment**	**Research oriented assignment**
Lexical Analyser	How tokenization can be done for strings that are more than the length of two buffers.	Lexical Analysis supports in log analysis. Take a sample log file of an industry and apply lexical analysis to analyse the log file.	Apply Lexical analysis to prevent or detect SQL Injection Attack [8].

Syntax Analysis	How do we find the Shift/Shift or Shift/Reduce conflict in the grammar?	Syntax Analysis supports in log analysis. Take a sample log file of an industry and apply syntax analyser to analyse the log file.	Create the EMail syntax verification parser [7].
Semantic Analyser	Show SDT based semantic check is better than simple symbol table based check or otherwise.	Take a sample feedback file of an industry and verify the semantics.	Analyse the type checking [6].
Intermediate Code Generation	Convert any simple arithmetic expression parse tree to three address code and stack machine code. Identify the best with respect to time complexity.	Analyse the usage of Intermediate Representation in NetASM [7]	Analyse the graph-based higher-order intermediate representation [11]
Code Optimization	Remove the common sub-expression in the C code segment and find the time complexity. X+ = (4*j + 5*i) Y+ = (7 + 4*j)	Take the software or program from the Github and apply the code optimization. Analyse the time difference in execution.	Analyse the Optgen: A generator for local optimization [9]
Code Generation	Take any arithmetic expression (c = a + b) three address code and find how many move instructions are required to store the result? Assume there is only two registers.	Analyse how the Industry could generate code for the embedded systems.	Generate the Snowflake architecture instructions for calculator C program [10].

6. CONCLUSION AND FUTURE WORK

The Compiler Design course is chosen in this paper because from the survey results it is evident that many students are pre-decided that this subject is of less scope in the research and development. This paper introduced a concept map for the Compiler Design course with necessary class room problems and assignments based on recent research and industry development. With the concept map and assignments, students can understand the importance and relationship of the Compiler Design course with other courses. Instructors teaching Compiler Design course need to update the students' assignments based on the latest research and industry progress. Hence, students will get motivated and focus on this subject. The future work of this paper is to make the students to design the problems and assignments for each concept related to industry and research and measure the accuracy of the students' knowledge on the concept and show the relevance of each concepts and phases in recent Research and Development.

ACKNOWLEDGEMENTS

We would like to thank the authors of different books, web articles and research papers from which the assignment problems and class room examples are taken and discussed in this paper. Also would like to thank the students participated in the survey.

REFERENCES

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman, *Compilers: Principles, Techniques and Tools*, Second Edition Book, 2006.

[2] Joseph D. Novak and Alberto J. Ca´nas, The Theory Underlying Concept Maps and How to Construct Them,http://cmap.ihmc.us/Publications/ResearchPapers/TheoryCmaps/TheoryUnderlyingConceptMaps.bck-11-01-06.htm [accessed on 10 January 2019]

[3] Joseph D. Novak, A theory of education. NY: Cornell University, 1977.

[4] CoolAid: The Cool 2016 Reference Manual, URL: http://pabst.cs.uwm.edu/classes/cs654/handouts/cool-manual.pdf [accessed on 11 January 2019].

[5] Julie Zelenski, Jerry Cain and Keith Schwarz, Decaf Specification, URL:http://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/030%20Decaf%20Specification.pdf [accessed on 11 January 2019].

[6] Francisco Ortin, Daniel Zapico, and Juan Manuel Cueva, Design Patterns for Teaching Type Checking in a Compiler Construction Course, *IEEE Transactions on Education*, Vol. 50, No. 3, pp.273-283, 2007.

[7] Muhammad Shahbaz , Nick Feamster, The case for an intermediate representation for programmable data planes, *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015.

[8] L. Ntagwabira, S. L. Kang, Use of query tokenization to detect and prevent sql injection attacks, *3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT) 2010*, Vol. 2, pp. 438-440, 2010.

[9] S. Buchwald. Optgen: A generator for local optimizations, *Proceedings of the 24th International Conference on Compiler Construction (CC),* pp. 171-189, Apr. 2015.

[10] Andre Xian Ming Chang, Aliasger Zaidy and Eugenio Culurciello, Efficient compiler code generation for Deep Learning Snowflake coprocessor, *1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications*, pp. 24-28, 2018.

[11] R. Leißa, M. Köster, and S. Hack, A graph-based higher-order intermediate representation, *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pp.202-212, 2015.

[12] Akim Demaille, Roland Levillain and Benoît Perrot, A set of tools to teach compiler construction, in *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, (ITiCSE '08), pp. 68-72, 2008.

[13] Li Xu and Fred G. Martin, Chirp on crickets: Teaching compilers using an embedded robot controller, *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, Vol. 38, no. 1, pp. 82-86, 2006.

[14] Tyson R. Henry, Teaching compiler construction using a domain specific language, *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, Vol. 37, no. 1, pp. 7-11, 2005.

[15] Elizabeth White, Ranjan Sen, and Nina Stewart, “Hide and show: Using real compiler code for teaching, *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, Vol. 37, no. 1, pp. 12-16, 2005.

[16] S. R. Vegdahl, “Using visualization tools to teach compiler design,” in Proceedings of the Fourteenth Annual Consortium on Small Colleges South eastern Conference (CCSC ’00), pp. 72-83, 2000.

[17] Marjan Mernik and Viljem Zumer, An educational tool for teaching compiler construction, *IEEE Transactions on Education*, Vol. 46, no. 1, pp. 61-68, 2003.

[18] Henry D. Shapiro and M. Dennis Mickunas, "A new approach to teaching a first course in compiler construction," *Proceedings of the ACM SIGCSE-SIGCUE technical symposium on Computer science and education*, pp.158-166, 1976.

[19] Martin Ruckert, "Teaching compiler construction and language design: Making the case for unusual compiler projects with postscript as the target language, *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, Vol. 39, no. 1, pp. 435-439, 2007.

[20] Divya Kundra and Ashish Sureka, Application of Case-Based Teaching and Learning in Compiler Design Course, arXiv:1611.00271v1 [cs.PL] 1 Nov 2016.

[21] D. Kundra and A. Sureka, "An experience report on teaching compiler design concepts using case-based and project-based learning ap-proaches," in *International Conference on Technology for Education (T4E 2016)*, 2016.

[22] Somya Sangal, Shreya Kataria, Twishi Tyagi, Nidhi Gupta, Yukti Kirtani, Shivli Agrawal and Pinaki Chakraborty, PAVT: a tool to visualize and teach parsing algorithms, *Education and Information Technologies, An official Journal of the IFIP Technical Committee on Education*, Vol. 23, No.6, pp.2737-2764 pp.2018.