2012 International Conference on Education Technology and Computer (ICETC2012)
IPCSIT vol.43 (2012) © (2012) IACSIT Press, Singapore

Teaching Practice And Exploration Of Compiler Principle Based On The Capability Cultivation For Application-Oriented Talents

Zhang Yajuan[+], Liu Hanbing, Feng Lingxia, Wang Xuechun
Information Engineering School, Huanghe Science and Technology College,
Zhengzhou, Henan,China 450006

Abstract. demand for application-oriented talents is increasing as the application of computer is becoming more and more popular. The course of Compiler Principle is an effective carrier to develop application-oriented talent's capability in virtue of a wide range of highly integrated knowledge. Based on present teaching status, this paper proposes that visualization method should be used to explain the theoretical knowledge, cultivate abstract thinking ability, develop innovative research and teaching, integrate theory and practice, set up experiment carefully, train students' systematic capability and team collaboration, aiming to improve teaching effectiveness while enhance employability after graduation.

Keywords: Compiler Principle; application-oriented talents; teaching reform; capability cultivation;

According to the strategic research report issued by the Steering Committee on Higher Education of Computer Science and Technology, Ministry of Education, computer science and technology should cultivate scientific, engineering and application-oriented talents[1]. Students in Chinese private college generally have poor foundation, so we must cultivate their capability from multiple angles guided by the goals of training application-oriented talents. The course of Compiler Principle is an effective carrier to develop application-oriented talent's capability in virtue of a wide range of highly integrated knowledge, [2] also it has an important potential to promote the abstract thinking ability, creative ability, practical ability, and systematic capability in students.

1. Status analysis

Compiler Principle is a very theoretical, abstract and logical course which integrates knowledge of many other ones, including Introduction to Computer Science, Program Design, Discrete Mathematics, Data Structures, Assembly Language and Software Engineering. However, students in private college have relatively poor foundation, they feel difficult and not practical when learning it, they can't recognize the great potential value in cultivating scientific methods and critical thinking. Driven by the profit-oriented thinking, they have low interest in learning, lack of enthusiasm and initiatives. Accordingly, it is impossible to achieve the goal of capability cultivation by learning the course.

[+] Corresponding author. *E-mail address*: wwwokboy@163.com.

To fundamentally change such situation, making the Compiler Principle become a course that can really improve the capability of students in Department of Computer Science, it is necessary to reform the existing teaching mode.

2. Capability Cultivation

2.1 Explain the theoretical knowledge and develop abstract thinking ability with visualization methods

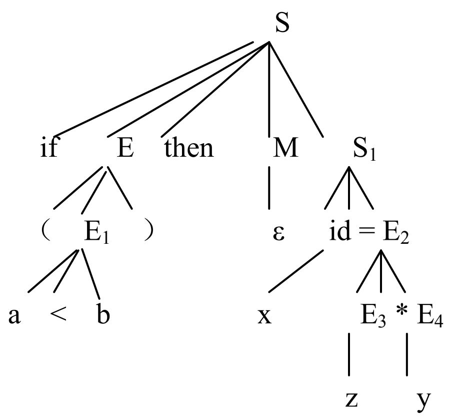
Chapters of semantic analysis and intermediate code generation are quite abstract, textbook introduces a lot of attribute translation grammars and quad statement generation, where students find difficult. When dealing with specific issues, simple arithmetic expressions and translation of assignment statements simulate the quad statement generation in the form of stack, which becomes unsuitable in the conditional expression and translation of control flow statements, so syntax tree[3] is used. For example, if (a <b) then x = z * y, the translation process of quad sequence is illustrated by grammar guided translation.

Following attribute translation grammars are required:

Grammatical rules	Semantic rules
（1）S->id=E	{ p=lookup(id.name); if(p!=null) GEN(=, E.place, , p); else error(); }
（2）E->E_1*E_2	{ E.place=NewTemp(); GEN(*,E_1 .place,E_2.place, E.place); }
（3）E->（E_1）	{ E.true= E_1.true; E.false= E_1.false; }
（4）E->id_1 rop id_2	{ E.true=nextquad; E.false=nextquad+1; GEN(‘j’rop,id_1,id_2, 0); GEN (j,_,_,0); }
（5）S->if E then M S_1	{ backpatch(E.true, M.Quad); S.nextlist=merge(E.false,S_1.nextlist); }
（6）M->ε	{ M.quad=nextquad; }

Problem-solving steps are as follows:

(1) Draw the syntax tree corresponding to the statement according to the grammatical rules.



(2) According to the semantic rules, add attribute value to the parent nodes of all leaf nodes from left to right, while generating the corresponding quad sequence.

E_1.true={100 }

E_1.false={101}

100(j<, a, b,0) 102

101(j,_,_,0)

M.quad={102}
102(*,y,z,T_1)
103(=,T_1,_,x)
104

(3) Starting from the bottom to gradually complete the zipper and backfill.

E.true={100}
E.false={101}
S.nextlist=101
Finally, the resulting quad sequence is as below:
100(j<,b,a,102)
101(j,_,_,104)
102(*,y,z,T_1)
103(=,T_1,_,x)
104

2.2 Stimulating and inspiring students' interest in learning gradually to cultivate innovation and creativity

Top-down parsing tries to derive a given symbol string step by step from start symbol. When it is a LL(1) grammar, top-down parsing can be conducted. Here LL(1) grammar decision is taken as an example, stimulate and inspire students' interest in learning gradually to cultivate innovation and creativity

(1) Consider the grammar G[A]:

A->Ax
A->y

The first production of G[A] is left recursion, when deriving symbol string yx^3, the parsing program cannot pass through finite look-ahead symbols, so choose a production that can terminate recursion after the recursion times are known.

(2) Will it be able to successfully derive if a grammar has no recursive production? Consider the grammar G[S]:

S->aAb
A->bAc|bBc
B->aB|a

G[S] has no recursive production, but if it begins from the start symbol S of the grammar when deriving the symbol string abacb, to match the first character a, aAb should be selected because S has only one production, and then a is successfully matched. In order to match the second character b, A needs to be replaced, but A has two productions both beginning from b, there is a question of selection, it is necessary to do back tracking to determine whether the grammar can generate this symbol string. Thus, a grammar with left factor but without left recursion cannot be successfully derived.

(3) Will it be able to successfully derive if a grammar has neither left recursion nor left factors? Consider the grammar G[P]:

P->Bc|ad
B->aA|b

P has two candidates "Bc" and "ad" during the derivation of input string abc. As to "Bc", aA and b two candidates of B should be checked. Therefore, two candidates "Bc" and "ad" of P are both begin from a. Thus it is unable to derive successfully when there is an intersection in the First Sets of candidates for the same nonterminal symbol.

(4) Will it be able to successfully derive if a grammar has no left recursion, left-factor, or intersection in the First Set of candidate? Consider the grammar G[Q]:

Q->Aa|Bb

A->a|cA|ε

B->b|dB

During the derivation of the input string cca, the first character is c, as for two candidates of Q, c beginning is selected, but two candidates of Q begin from A and B, consider cA in A and B begins from c, so select Aa, the first character is matched; the second character is c, select cA; the third should match a, A has a candidate defining a, if it is selected there is one more symbol a, so there is an only choice ε. The reason is that First Set in candidate and Follow set of A have an intersection a.

After above excavation, we can summarize the conditions of LL (1) grammar decision, for each nonterminal symbol A in G, candidate of A should meet:

(1) No left recursion;

(2) No left factor;

(3) If there is no empty production, First set of candidate has no intersection;

(4) If there is an empty production, First Set of candidate and Follow set of A have no intersection.

2.3 Integrate theory and practice to develop practical abilities

Compiler Principle is an algorithm problem in essence, but the implementation of turning algorithm into program is not easy, so sufficient details should be provided to students in teaching: when explaining some knowledge, first to analyze the overall goal of knowledge, and then subdivide it into many small targets, which refines the content of each learning module into little "tasks" easy to grasp, so that the overall learning goal is achieved through all of these specific "tasks". To prevent students from needlessly spending too much time on the details which affect the main purpose of practice, it is necessary to pay attention to inspire students to summarize the contents, guiding them to master the process of analyzing and solving problems.

Taking LR parser for an example, if a given grammar G [A] is:

A -> B a|Db

B->c

D->c

Parsing table can only be constructed when the category of LR grammar is known, thus showing the process of constructing the parser. The category of LR grammar cannot be identified from the senses, but from following process:

(1) Expand grammar, add a new production, sort and code it;

(2) Construct and identify DFA of viable prefix;

(3) Check each item set, if there is no conflict, it is LR(0) grammar; if any conflict, then turn to step (4);

(4) Consider the reason of conflict, as to shift/reduce conflict, if the Follow sets of the shift symbols and nonterminal symbols to be reduced have no intersection, it is SLR(1) grammar; if any intersection, turn to step (5); as to reduce/reduce conflict, if the Follow sets of both nonterminal symbols to be reduced have no intersection, it is SLR(1) grammar, if any intersection, turn to step (5);

(5) Consider look-ahead symbols, if they are not the same, it is LR(1) grammar, or else it isn’t.

(6) If there is no conflict after concentric set is merged for an LR (1) grammar, it is LALR(1) grammar, or else it isn’t.

Only every step of above process have been achieved, students can truly understand how to design the LR parser. Hence good theory learning can contribute to cultivate practical ability.

3. Conclusion

As a private college, it is necessary for teachers to improve teaching methods continuously during teaching process based on students’ characteristics under the guidance of training application-oriented talents, so as to stimulate the enthusiasm in learning, while improving the quality of teaching and promoting student ability.

4. Reference:

[1] Jiang Zongli, Jiang Shouxu. Compiler Principle [M]. Beijing: Higher Education Press, 2010.

[2] Fan Lili, Wang Zhongqun. Professionalism Exploration and Capability Cultivation in Teaching "Compiler

Principle" [J]. Modern computer, 2010, (7) :68-71.

[3] Chen Huowang, Liu Chunlin, Tan Qingping et al. Compiler Principle of Programming Language [M].Beijing: National Defense Industry Press, 2002.