CS441 Fall 04 Internet Section

Solutions for hw #1

Chapter 1

Q4. What arguments can you make against the idea of a single language for all programming domains? (2)

Some arguments against having a single language for all programming domains are: The language would necessarily be huge and complex; compilers would be expensive and costly to maintain; the language would probably not be very good for any programming domain, either in compiler efficiency or in the efficiency of the code it generated.

Q5. Name and explain another criterion by which languages can be judged (in addition to those discussed in this chapter). (2)

One possibility is wordiness. In some languages, a great deal of text is required for even simple complete programs. For example, COBOL is a very wordy language. In Ada, programs require a lot of duplication of declarations. Wordiness is usually considered a disadvantage, because it slows program creation, takes more file space for the source programs, and can cause programs to be more difficult to read.

Q8. Many languages distinguish between uppercase and lowercase in user-defined names. What are the pros and cons of this design decision? (2)

The reasons why a language would distinguish between uppercase and lowercase in its identifiers are:

(1) So that variable identifiers may look different than identifiers that are names for constants, such as the convention of using uppercase for constant names and using lowercase for variable names in C, and

(2) So that catenated words as names can have their first letter distinguished, as in `Total Words`. (I think it is better to include a connector, such as underscore.) The primary reason why a language would not distinguish between uppercase and lowercase in identifiers is it makes programs less readable, because words that look very similar are actually completely different, such as `SUM` and `Sum`.

Q.10 What are the arguments for writing efficient programs even though hardware is relatively inexpensive?(2)

One of the main arguments is that regardless of the cost of hardware, it is not free. Why write a program that executes slower than is necessary. Furthermore, the difference between a well-written efficient program and one that is poorly written can be a factor of two or three. In many other fields of endeavor, the difference between a good job and a poor job may be 10 or 20 percent. In programming, the difference is much greater.

Q.15 How do type declaration statements for simple variables affect the readability of a language, considering that some languages do not require them? (2)

The use of type declaration statements for simple scalar variables may have very little effect on the readability of programs. If a language has no type declarations at all, it may be an aid to readability, because regardless of where a variable is seen in the program text, its type can be determined without looking elsewhere. Unfortunately, most languages that allow implicitly declared variables also include explicit declarations. In a program in such a language, the declaration of a variable must be found before the reader can determine the type of that variable when it is used in the program.

Q.18 Many contemporary languages allow two kinds of comments, one in which delimiters are used on both ends (for multiple-line comments), and one in which a delimiter marks only the beginning of the comment (for one-line comments). Discuss the advantages and disadvantages of each of these with respect to our criteria. (2)

The main disadvantage of using paired delimiters for comments is that it results in diminished reliability. It is easy to inadvertently leave off the final delimiter, which extends the comment to the end of the next comment, effectively removing code from the program. The advantage of paired delimiters is that you *can* comment out areas of a program. The disadvantage of using only beginning delimiters is that they must be repeated on every line of a block of comments. This can be tedious and therefore error-prone. The advantage is that you cannot make the mistake of forgetting the closing delimiter.

Chapter 2

Q.10 Outline the major motivation of IBM in developing PL/I. (2)

The main motivation for the development of PL/I was to provide a single tool for computer centers that must support both scientific and commercial applications. IBM believed that the needs of the two classes of applications were merging, at least to some degree. They felt that the simplest solution for a provider of systems, both hardware and software, was to furnish a single hardware system running a single programming language that served both scientific and commercial applications

Q.11 Was IBM's major motivation for developing PL/I correct, given that history of computers and language developments since 1964? (2)

IBM was, for the most part, incorrect in its view of the future of the uses of computers, at least as far as languages are concerned. Commercial applications are nearly all done in languages that are specifically designed for them. Likewise for scientific applications. On the other hand, the IBM design of the 360 lines of computers was a great success--it still dominates the area of computers between supercomputers and minicomputers. Furthermore, 360 series computers and their descendants have been widely used for both scientific and commercial applications. These applications have been done, in large part, in FORTRAN and COBOL.

Q.14 What are the arguments both for and against the idea of a type less language?(2)

The argument for type less languages is their great flexibility for the programmer. Literally any storage location can be used to store any type value. This is useful for very low-level languages used for systems programming. The drawback is that type checking is impossible, so that it is entirely the programmer's responsibility to insure that expressions and assignments are correct.

Q.18 Languages continually evolve. What sort of restrictions do you think are appropriate for changes in programming languages? Compare your answer with the evolution of Fortran.(2)

A good deal of restraint must be used in revising programming languages. The greatest danger is that the revision process will continually add new features, so that the language grows more and more complex. Compounding the problem is the reluctance, because of existing software, to remove obsolete features.

Chapter 3

Q.2 Write EBNF descriptions for the following:
a. A Java class definition header statement (2 each)

2a. <class_head> → {<modifier>} **`class`** <id> [**`extends`** class_name]

[**`implements`** <interface_name> {, <interface_name>}]

<modifier> → **`public`** | **`abstract`** | **`final`**

2c. <switch_stmt> → **`switch`** (<expr>) {**`case`** <literal> **`:`** <stmt_list>
{**`case`** <literal> : <stmt_list> } [**`default :`** <stmt_list>] }

*Q.3 Rewrite the BNF of Example 3.4 to give + precedence over * and force + to be right associative. (2 each)*

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <expr> * <term>

| <term>

<term> → <factor> + <term>

| <factor>

<factor> → (<expr>)

| <id>

Q.6 Using the grammar in Example 3.2, show a parse tree derivation for each of the following statements: (2 each)
*a. A = A * (B + (C * A))*

(a) <assign> => <id> = <expr>

=> `A` = <expr>

=> `A` = <id> `*` <expr>

=> `A = A *` <expr>

=> A = A * (<expr>)

=> A = A * (<id> + <expr>)

=> A = A * (B + <expr>)

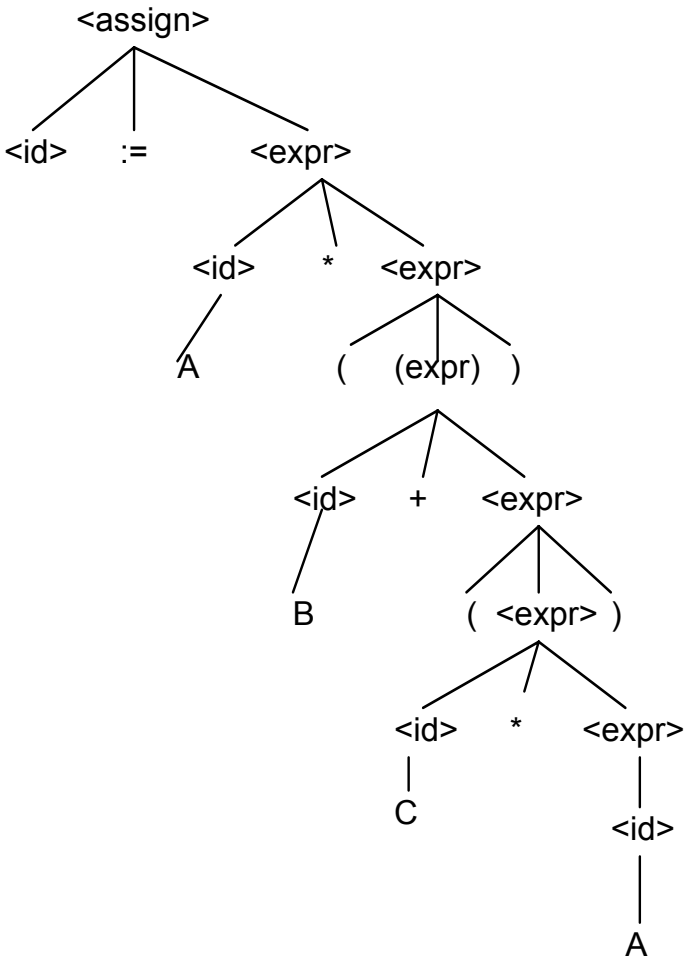
=> A = A * (B + (<expr>))

=> A = A * (B + (<id> * <expr>))

=> A = A * (B + (C * <expr>))

=> A = A * (B + (C * <id>))

=> A = A * (B + (C * A))



Q.7 Using the grammar in Example 3.4, show a parse tree derivation for each of the following statements: (2 each)

*a. A = (A + B) * C*

(a) <assign> => <id> = <expr>

=> A = <expr>

=> A = <term>

=> A = <factor> * <term>

=> A = (<expr>) * <term>

=> A = (<expr> + <term>) * <term>

=> A = (<term> + <term>) * <term>

=> A = (<factor> + <term>) * <term>

=> A = (<id> + <term>) * <term>

=> A = (A + <term>) * <term>

=> A = (A + <factor>) * <term>

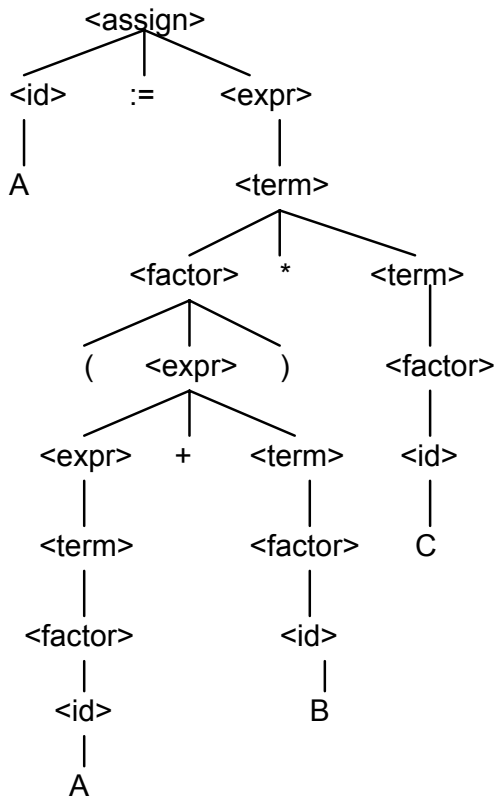
=> A = (A + <id>) * <term>

=> A = (A + B) * <term>

=> A = (A + B) * <factor>

=> A = (A + B) * <id>

=> A = (A + B) * C



Q.8 Prove that the following grammar is ambiguous:

<S>→ <A>

<A>→<A>+<A> | <id>

<id>→a | b | c

(2 each)

The following two distinct parse trees for the same string, which proves that the grammar is ambiguous.