Programming Languages HW 1

Jiraphon Yenphraphai (jy3694)

June 10, 2022

1 Language Standards

1.1 1

According to the introduction in (7) and (7.1) from Java standard, Java packages are stored in hierarchical structure because it is convenient to manage packages so that it prevents name conflicts. However, Java language doesn't require the class to be defined hierarchies or a separate header file for a class.

According to (7.2) from Java standard (passage 3), packages may be stored in a local file system or a distributed file system or a database.

1.2 2

According to (4.3.4) from Java standard (passage 1), two reference types (class types, interface types, type variables and array types) are the same compile-time when they both have the same binary name (compile name) or argument or are in the same modules.

1.3 3

According to (4.12.4), explicitly final is assigned to a variable when we desire to keep its value the same. However, there are some cases where variables are implicitly declared as final:

1. A local variable where it doesn't have final keyword, has been initialized, and its value hasn't been changed through operations: an assignment expression, prefix or postfix increment or decrement.

2. A local variable where it doesn't have final keyword, has not been initialized, and is unassigned before it is assigned. It is never used as an incrementor or decrementor.

3. A method, constructor, lambda and exception statement have a local variable whose declaratory has an initializer.

1.4 4

To ensure that the precision across different platforms be the same, with the **strictfp** keyword, floating-point calculation will be the same followed FP-strict regardless of the platform, according to (8.1.1.3) and (8.4.3.5) and (9.1.1.2). Here is the example usage of **strictfp**:

```
import java.lang.Math;
class test {
    public strictfp double log (double var)
    {
        double out;
        out = Math.log(a);
        return out;
    }
}
```

1.5 5

According to (4.3.3), a string object has a constant (unchanging) value. Thus, changing the content of string is not allowed in Java. According to (10.9), we can use an array of char to do so as the elements are mutable. Or we can use StringBuffer which is the class that allows the arrays of characters to change.

1.6 6

According to (8) and (8.3), a instance variable in the class can hide all accessible variables with the same name from its superclass and superinterface even though it does not have the same type. To access the variables from superclass, we use the keyword super.

A method of class C ($m^{'}$) which inherits this method m from its superclass can be hidden by m if this method($m^{'}$) is accessible to C and it is declared as static, according to (8.4.8.2).

According to (6.4), a local variable in the inner scope would shadow or hide the variable with the same name from the outer scope.

1.7 7

According to (6.3.10) from C++ standard paragraph 2, if a class or enumeration is declared with the same name as that of a variable, function, or enumerator in the same scope, it will be hidden.

In this case, if there exists the same name in the same scope, the class and enumeration name are hidden. For example,

```
#include <stdio.h>
int main() {
   int a=10;
   class a;
   // it will use variable a in this case, not from class
}
```

1.8 8

1.8.1 C++

According to (6.3.10) from C++ standard paragraph 5, if an identifier is in scope and is not hidden, it is visible.

1.8.2 C

According to (6.2.1) from C standard (paragraph 2), an identity which designates for each entity is visible within its scope. Different entities in that scope should be declared with different identifier. Within the inner scope, the identifier which designates the entity declared in the inner scope hides the outer scope.

1.9 9

According to (14.2) from C++ standard paragraph 1, if a class inherits from a base class using the protected access specifier, the public and protected members will be accessible as the protected members of the derived class.

1.10 10

```
foreach-statement:
    foreach (variable_type variable in expression) {...}
```

According to (13.9.5), foreach statement iterates over the elements of a collection. Two ways in which foreach is different from for loop:

1. An iteration variable is read-only local variable; thus, it can't be modified. However, in the case of for loop, there is no restriction on modifying the iteration variable.
2. foreach enumerates over the elements of a collection until there is no elements left. However, the for statement evaluates a condition every iteration until the condition is false. (13.9.4)

2 Grammars and Parse Trees

Assuming that, the lower-case letter is terminal symbol.

2.1 1

2.1.1 a

S $\rightarrow$ S S - | S S + | a | b
The examples of the language described by this grammar are: a, b, aa+, aa-, ab+, ab-, aba+, abb-, aaa++, ab-a-, ...

In the case of language where it has 1 terminal symbols, it can only be: a or b.

Other than that, the language described by the grammar is strings with two a or b followed by zero or more combination of zero or more a or b or + or - , where the sum of number of a and b equals the sum of number of + and -, followed by + or -.

Non-terminal Symbols: S
Non-terminal S is the start symbol
Terminal Symbols: +, -, a, b

2.1.2 b

S $\rightarrow$ a S a a | B
B $\rightarrow$ b B | ϵ
In the first case, the examples of the language described by this grammar are: ϵ, b, bb, bbb, ...
The language contains only zero or more b. The regular expression is as follows:

[b]*

In the second case, the examples of the language described by this grammar are also: aaa, abaa, abbaa, ... The language described by the grammar is also strings with a followed by zero or more b followed by aa. The regular expression is as follows:

[a]+[b]*[aa]+

In conclusion, the strings of this language include both the first and second cases.

Non-terminal Symbols: S
Non-terminal S is the start symbol
Terminal Symbols: a, b, ϵ

2.1.3 c

S $\rightarrow$ A a | M S | S M A
A $\rightarrow$ A a | ϵ
M $\rightarrow$ ϵ | M M | b M a | a M b
The examples of the language described by this grammar are: ϵ, a, baa, aba, aabbaa, babaaa, abababaa, bbaaaa, ...
We can notice that the uniqueness of this language by the number of a and b. The language described by the grammar is strings that have the number of a more than b.

Non-terminal Symbols: S, A, M
Non-terminal S is the start symbol
Terminal Symbols: a, b, ϵ

2.1.4 d

S $\rightarrow$ a S a | b S b | a | b | ϵ
The examples of the language described by this grammar are: ϵ, a, b, aaa, aba, baab, aabbaa, abababa, ...

We can notice that the pattern for this language is palindrome words (the character reads the same forward as backward). For odd-length strings (n), we can pick any combination of a and b to compose of ($(n-1)/2$)-length strings followed by a or b followed by the mirror of the previous combination. For example, we pick abb then pick b, then its mirror bba, and concatenate these strings. For even-length strings (n), e can pick any combination of a and b to compose of ($n/2$)-length strings followed by the mirror of the previous combination.

Non-terminal Symbols: S
Non-terminal S is the start symbol
Terminal Symbols: a, b, ϵ

2.1.5 e

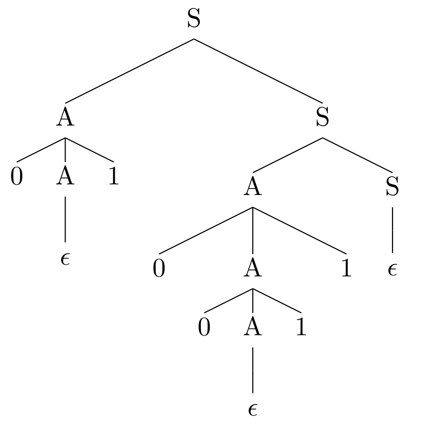
S $\rightarrow \epsilon$ | S S | [S]
The examples of the language described by this grammar are: ϵ, [], [[]], [][], [[[[]]]][], [[]][][][], [[][][]]...
The language described by the grammar is also strings with zero or more []. Inside each [], it can contain zero of more of [] recursively or concatenate zero or more of [] .

Non-terminal Symbols: S
Non-terminal S is the start symbol
Terminal Symbols: [,], ϵ

2.2 2

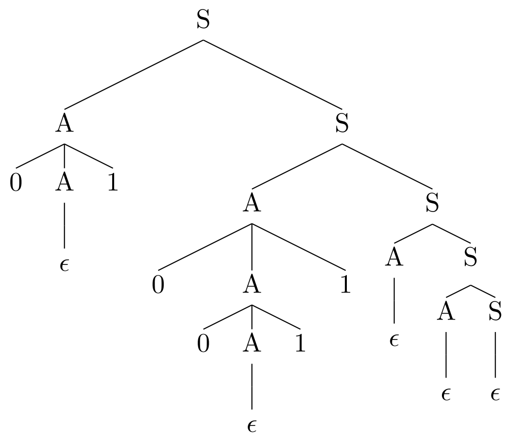
Let the example string be 010011. The first parse tree

$$\begin{aligned} \text{S} &\Longrightarrow \underline{\text{A}}\ \text{S} \\ &\Longrightarrow 0\ \text{A}\ 1\ \underline{\text{S}} \\ &\Longrightarrow 0\ \text{A}\ 1\ \underline{\text{A}}\ \text{S} \\ &\Longrightarrow 0\ \text{A}\ 1\ 0\ \underline{\text{A}}\ 1\ \text{S} \\ &\Longrightarrow 0\ \underline{\text{A}}\ 1\ 0\ 0\ \underline{\text{A}}\ 1\ 1\ \underline{\text{S}} \\ &\Longrightarrow 0\ \epsilon\ 1\ 0\ 0\ \epsilon\ 1\ 1\ \epsilon \end{aligned}$$



We can rewrite S infinitely as it can be ϵ. The second parse tree

$$\begin{aligned} \text{S} &\Longrightarrow \text{A}\ \underline{\text{S}} \\ &\Longrightarrow \underline{\text{A}}\ \text{A}\ \text{S} \\ &\Longrightarrow 0\ \text{A}\ 1\ \underline{\text{A}}\ \text{S} \\ &\Longrightarrow 0\ \text{A}\ 1\ 0\ \underline{\text{A}}\ 1\ \text{S} \\ &\Longrightarrow 0\ \text{A}\ 1\ 0\ 0\ \text{A}\ 1\ 1\ \underline{\text{S}} \\ &\Longrightarrow 0\ \text{A}\ 1\ 0\ 0\ \text{A}\ 1\ 1\ \text{A}\ \underline{\text{S}} \\ &\Longrightarrow 0\ \text{A}\ 1\ 0\ 0\ \text{A}\ 1\ 1\ \text{A}\ \text{A}\ \text{S} \\ &\Longrightarrow 0\ \epsilon\ 1\ 0\ 0\ \epsilon\ 1\ 1\ \epsilon\ \epsilon\ \epsilon \end{aligned}$$



We can see that these two parse trees are different, thus, the grammar is ambiguous.

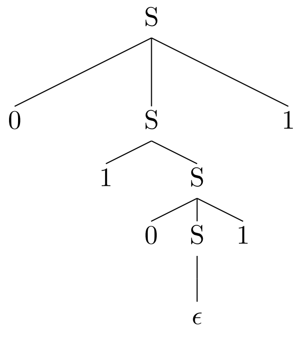
2.2.1 b

The unambiguous grammar:
S → 1 S | 0 S 1 | ϵ
Non-terminal Symbols: S
Non-terminal S is the start symbol
Terminal Symbols: 0, 1

2.2.2 c

Write the new parse tree for 010011.

$$\begin{aligned} \mathrm{S} &\Longrightarrow 0\ \underline{\mathrm{S}}\ 1 \\ &\Longrightarrow 0\ 1\ \underline{\mathrm{S}}\ 1 \\ &\Longrightarrow 0\ 1\ 0\ \underline{\mathrm{S}}\ 1\ 1 \\ &\Longrightarrow 0\ 1\ 0\ \epsilon\ 1\ 1 \end{aligned}$$



2.3 3

2.3.1 a

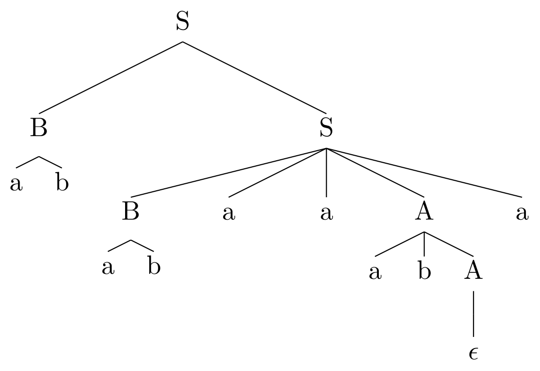
The string from this language always have one or more of ab in the front. Thus, our grammar needs to always have ab in the front which expresses in the first and third grammar rules below. Following one or more of ab, we need to have aa all the time which expresses in the first rule. Following aa, there is zero or more ab which is determined by the second rule. Finally, every string ends with a which expresses in the first rule.

S → B S | B a a A a
A → a b A | ϵ
B → a b

2.3.2 b

A parse tree for ababaaaba:

$$\begin{aligned} \mathrm{S} &\Longrightarrow \underline{\mathrm{B}}\ \mathrm{S} \\ &\Longrightarrow a\ b\ \underline{\mathrm{S}} \\ &\Longrightarrow a\ b\ \underline{\mathrm{B}}\ \mathrm{S} \\ &\Longrightarrow a\ b\ a\ b\ \underline{\mathrm{S}} \\ &\Longrightarrow a\ b\ a\ b\ a\ a\ \underline{\mathrm{A}}\ a \\ &\Longrightarrow a\ b\ a\ b\ a\ a\ a\ b\ \underline{\mathrm{A}}\ a \\ &\Longrightarrow a\ b\ a\ b\ a\ a\ a\ b\ \epsilon\ a \end{aligned}$$



2.4 4

2.4.1 a

We need at least three 1 in the string. Thus, I put three 1 in the string where each one get separated by either zero or more 0 or 1 as it doesn't necessarily consecutive.
Regular expression: [0|1]*[1][0|1]*[1][0|1]*[1][0|1]*

Context-free grammar:
S → A 1 A 1 A 1 A
A → 0 A | 1 A | ϵ
Non-terminal Symbols: S, A
Non-terminal S is the start symbol
Terminal Symbols: 0, 1, ϵ

2.4.2 b

We need 0 in the center so I choose the non-terminal symbol to stay in the center all the time shown below. The length has to be odd, so I put 0 and 1 before and after the non-terminal symbol.

Context-free grammar:
S → 0 | 0 S 0 | 0 S 1 | 1 S 0 | 1 S 1
Non-terminal Symbols: S
Non-terminal S is the start symbol
Terminal Symbols: 0, 1

2.4.3 c

In the case of $i = j$, the number of a equals to b, which is independent to that of c. We use a and b to stay side-by-side with non-terminal A to constraint on the number of a's and b's. Likewise, we use a and c to stay side-by-side with non-terminal C to constraint the number of a's and c's.
Context-free grammar:
S → A B | C
A → a A b | ϵ
B → c B | ϵ
C → a C c | D | ϵ
D → b D | ϵ
Non-terminal Symbols: S, A, B, C, D
Non-terminal S is the start symbol
Terminal Symbols: a, b, c, ϵ

2.4.4 d

We need the sum of the number of a and b to be equal to that of c, so every time we has a or b, we need to has c. To get the consecutive a, b, or c, we need to make that terminal symbols stay before and after the non-terminal symbol.
Context-free grammar:
S → a S c | A
A → b A c | ϵ
Non-terminal Symbols: S, A

Non-terminal S is the start symbol
Terminal Symbols: a, b, c, ϵ

2.5 5

2.5.1 a

The grammar from the question is as follows:
S $\rightarrow$ S + X | X
X $\rightarrow$ X * Y | Y
Y $\rightarrow$ a
We can see that there are 2 non-terminal symbols S and X in the first positions of the right-hand side. These cause the issue. Assume we have the string a+a*a and LL(1) parser. The parser looks at a, and it can't decide whether it would choose S + X or X as it sees the non-terminal symbol again. It needs to look ahead to solve the issue, but our LL parser is 1 look ahead. So, we need to rewrite the grammar to get rid of the non-terminal symbols which are in the first positions of the right-hand side.

2.5.2 b

We can rewrite the rules as follows:
Context-free grammar:
S $\rightarrow$ X $S^{'}$
$S^{'}$ $\rightarrow$ + X $S^{'}$ | ϵ
X $\rightarrow$ Y $X^{'}$
$X^{'}$ $\rightarrow$ * Y $X^{'}$ | ϵ
Y $\rightarrow$ a
Non-terminal Symbols: S, $S^{'}$, X, $X^{'}$, Y
Non-terminal S is the start symbol
Terminal Symbols: a, ϵ

3 Regular expressions

3.1 1

3.1.1 a

As we want only 1 a in any order, repeat, be any length of any number of b's and c's , we specify only 1 a in the regular expression where a stays side-by-side with zero or more b's or c's.

Regular expression: [b|c]*a[b|c]*

3.1.2 b

We want a to be in the groups of three; thus, we group it together. This group, b, and c can be zero or more occurrence.

Regular expression: [(aaa)|b|c]*

3.1.3 c

To ensure that the alphabet length is a multiple of five, we put it inside the zero of more repetition where each repetition has length five.

Regular expression: ([a|b|c][a|b|c][a|b|c][a|b|c][a|b|c])*

3.1.4 d

To ensure that it is even integers, we need to end with 0, 2, 4, 6, 8.

Regular expression: [0-9]*[0|2|4|6|8]

3.1.5 e

To prevent 101, we need to have at least two 0 before and after 1 or we can have consecutive 1.

Regular expression: 0*(1*0*00)*1*0*

3.2 2

The statement is true.
Let's first consider ((R∗ R) | R)∗.
The first case where we pick R from or statement

$$\begin{aligned}((\mathrm{R}*\ \mathrm{R})|\ \mathrm{R})* &= (\mathrm{R})* \\ &= \mathrm{R}*\end{aligned}$$

The second case where we pick (R∗ R) from or statement

$$\begin{aligned}((\mathrm{R}*\ \mathrm{R})|\ \mathrm{R})* &= (\mathrm{R}*\ \mathrm{R})* \\ &= \mathrm{R}**\ \mathrm{R}* \\ &= \mathrm{R}*\ \mathrm{R}* \\ &= \mathrm{R}*\end{aligned}$$

Thus, from both cases, we can see that this statement is always true.

4 Extended calculator

5 Associativity and Precedence

5.1 1

$$\begin{aligned}5*2-6+7/7 &= 5*2+(-6+7)/7 \\ &= 5*2+1/7 \\ &= 5*(2+1)/7 \\ &= 5*3/7 \\ &= 5*(3/7) \\ &= \frac{15}{7}\end{aligned}$$

5.2 2

$$\begin{aligned}8*8/4+4/2*2 &= 8*8/(4+4)/2*2 \\ &= 8*8/8/(2*2) \\ &= 8*8/(8/4) \\ &= 8*(8/2) \\ &= 8*4 \\ &= 32\end{aligned}$$

5.3 3

$$\begin{aligned}5-3*5\%3+2 &= 5-3*5\%(3+2) \\ &= (5-3)*5\%5 \\ &= 2*(5\%5) \\ &= 2*0 \\ &= 0\end{aligned}$$

5.4 4

$$\begin{aligned}10-2+6*10-2/6+4 &= 10-2+6*10-2/(6+4)\\ &= 10-2+6*(10-2)/10\\ &= 10-2+6*8/10\\ &= 10+(-2+6)*8/10\\ &= 10+4*8/10\\ &= (10+4)*8/10\\ &= 14*(8/10)\\ &= \frac{56}{5}\end{aligned}$$

5.5 5

$$\begin{aligned}2+5/2*5-4\%2 &= 2+5/2*(5-4)\%2\\ &= 2+5/2*1\%2\\ &= (2+5)/2*1\%2\\ &= 7/2*(1\%2)\\ &= 7/(2*1)\\ &= \frac{7}{3}\end{aligned}$$

5.6 6

$$\begin{aligned}12-4*9-4/10/6 &= 12-4*(9-4)/10/6\\ &= (12-4)*5/10/6\\ &= 8*5/(10/6)\\ &= 8*(\frac{5}{5/3})\\ &= 8*3\\ &= 24\end{aligned}$$

6 Short-Circuit Evaluation

6.1 1

```
if ( f() && h() && i() || g() || f() && i())
{
    cout << "What lovely weather!" << endl;
}
bool f()
{
    cout << "Hello";
    return true;
}
bool g()
{
    cout << "World!";
    return false;
}
bool h()
{
    cout << "There ";
    return false;
}
```

```
bool i()
{
    cout << "Darling! " << endl;
    return false;
}
```

6.2 2

Yes, C++ compilers requires to implement short-circuit evaluation. According to (8.14), if there are more than one logical AND, the operand after the first one are not executed if the first operand returns false. Also, for the case of logical OR, the operand after the first one are not executed if the first operand returns true, according to (8.15).

6.3 3

```
int main(){
    int a=1, b=0;
    if (b != 0 && a/b > 100) {
    // It will not executed inside
    }
}
```

In this example, it checks whether the denominator is not zero preventing the execution of a/b to become NaN.

7 Bindings and Nested Subprograms

Unit	Var	Where Declared
main	a	main
	b	main
sub1	a	sub1
	b	main
sub2	a	sub2
	b	main
	c	sub2
sub3	a	sub2
	b	sub3
	c	sub2
	d	sub3
	e	Undefined