Programming Languages Assignment 1 - Narasimman Sairam

C++ Standard

1. There are five kinds of tokens: identifiers, keywords, literals, 16 operators, and other separators. Blanks, horizontal and vertical tabs, newlines, form feeds, and comments (collectively, “white space”), as described below, are ignored except as they serve to separate tokens.
2. There are 87 keywords (including alternate representations)

3. BNF of floating literals

<floating-literal> ::= <fractional-constant> {<exponent-part>} {< floating-suffix>}
| <digit-sequence> <exponent-part> {<floating-suffix>}
<fractional-constant> ::= {<digit-sequence>} ‘.’ <digit-sequence>
| <digit-sequence> ‘.’
<exponent-part> ::= ‘e’ { <sign> } <digit-sequence>
‘E’{ <sign> } <digit-sequence>
<sign> ::= + | -
<digit-sequence> ::= <digit> | <digit-sequence> <digit>
<floating-suffix> ::= f | l | F | L

4. C++ grammar is ambiguous, context-dependent and potentially requires infinite lookahead to resolve some ambiguities. LR parsers can't handle ambiguous grammar rules, by design. (Made the theory easier back in the 1970s when the ideas were being worked out).

C and C++ both allow the following statement:

```
x * y ;
```

It has two different parses:

It can be the declaration of y, as pointer to type x
It can be a multiply of x and y, throwing away the answer.

Identifiers, Lifetimes, and Binding

1. Is it possible to have an identifier associated with more than one address?
    - No. An identifier is can be associated with only one address at any point of time during the execution of the program.

2. Is it possible to have an address associated with more than one identifier?
    - Yes. There can be multiple pointer variables associated to the same address in the memory.

```
Int a = 0
```

Int *b = &a;

Here, a and b points to the same memory location.

3. Is it possible to have an identifier whose lifetime is greater than the object(s) in memory it binds to? Why or why not?

- Yes. This can happen in case of Dangling Pointers.
- Dangling pointers arise during object destruction, when an object that has an incoming reference is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory. As the system may reallocate the previously freed memory to another process, if the original program then dereferences the (now) dangling pointer,*unpredictable behavior may result*, as the memory may now contain completely different data.

4. Is it possible to have an object(s) in memory whose lifetime is greater than the identifier it binds to? Why or why not?

- Yes. This can happen when a memory leak occurs. A memory leak is memory which hasn't been freed, there is no way to access (or free it) now, as there are no ways to get to it anymore. (E.g. a pointer which was the only reference to a memory location dynamically allocated (and not freed) which points somewhere else now.)

Scopes:

1. *Static Scoping*: Value of b = 4; So, the output is 10.
2. *Dynamic Scoping:* Value of b = 3; The value is set by the function g(). So, the output is 9.

Short-Circuit Evaluation:

1. Yes, short-circuiting and evaluation order are required for operators || and && in both C and C++ standards.

   C++ standard says

   In the evaluation of the following expressions:
   a && b
   a || b
   a ? b : c
   a , b

using the built-in meaning of the operators in these expressions, there is a sequence point after the evaluation of the first expression.

2. **Output:**

for (int x=0,y=0; x++ < 5 || y++ > 3;) std::cout << "Output " << x << " " << y << std::endl;

```
1 0
2 0
3 0
4 0
5 0
```

The x value is incremented until it reaches 5. Then, the first condition fails and (y=0) is compared to 3 which again fails and the execution stops.

3.

- The conditional-or operator || operator is like | (§15.22.2), but evaluates its righthand operand only if the value of its left-hand operand is false.

ConditionalOrExpression:
  ConditionalAndExpression
  ConditionalOrExpression || *ConditionalAndExpression*

The conditional-or operator is syntactically left-associative (it groups left-to-right).

- At run-time, the left-hand operand expression is evaluated first; if the result has type Boolean, it is subjected to unboxing conversion (§5.1.8). If the resulting value is true, the value of the conditional-or expression is true and the right-hand operand expression is not evaluated. If the value of the left-hand operand is false, then the right-hand expression is evaluated; if the result has type Boolean, it is subjected to unboxing conversion (§5.1.8). The resulting value becomes the value of the conditional-or expression.

Bottom-Up Parsing:

Input : (+ 5 6 (6 2))*

Actual computation will be: (+ (+ 5 6) (6 2))*

Current Stack	Operation
(	shift

(+	shift
(+ 5	shift
(+ TERM	REDUCE
(+ TERM 6	shift
(+ TERM TERM	REDUCE
(+ TERM FACTOR	REDUCE
(+ EXPR	REDUCE
(+ EXPR (	shift
(+ EXPR (*	shift
(+ EXPR (* 6	shift
(+ EXPR (* TERM	REDUCE
(+ EXPR (* TERM 2	Shift
(+ EXPR (* TERM TERM	REDUCE
(+EXPR (EXPR	REDUCE
(+EXPR (EXPR)	Shift
(+EXPR EXPR	REDUCE
(+EXPR EXPR)	Shift
(EXPR)	REDUCE
EXPR	REDUCE
PROG	REDUCE

5) **Precedence**

```
expr : expr = expr { $1 = $2; }
   | '*' POST_EXPR {$$ = VALUE($2); }
  ;
POST_EXPR : SYMBOL '++'    {$$ = $1; $1 = $1 + 1;}
```

SYMBOL is the terminal character returned by FLEX.

* SHIFT
*p - SHIFT
*SYMBOL - REDUCE
*SYMBOL ‘++’ -- SHIFT
*POST_EXPR -- REDUCE
expr -- REDUCE
expr = -- SHIFT
expr = * SHIFT
expr = *q SHIFT
expr = *SYMBOL - REDUCE
expr = *SYMBOL ++ - SHIFT
expr = *POST_EXPR - REDUCE
expr = expr - REDUCE
expr = REDUCE