Benjamin Jakubowski

Programming Languages HW1

C++ Language Standard Questions

1. There are five kinds of tokens: identifiers, keywords, literals, operators, and other separators. Blanks, horizontal and vertical tabs, newlines, formfeeds, and comments (collectively, “white space”) are ignored except as they serve to separate tokens [Page 21].

2. There are three kinds of selection statements described in 6.4 [Page 138]. Each choose a different flow of control:

- if ( condition ) statement

- if ( condition ) statement else statement

- switch ( condition ) statement

3. There are four kinds of iteration statements that specify looping described in 6.5 [Page 139]:

- while ( condition ) statement

- do statement while ( expression ) ;

- for ( for-init-statement conditionopt; expressionopt) statement

- for ( for-range-declaration : for-range-initializer ) statement

4. An example of dis-allowed function overloading are two function declarations that differ only in the return type [Page 300] (ex: two functions with the same name and parameters, but one returns a char and the other an int).

5. Two examples of features that work differently in C++ than in C are [Page 1232]:

- A large number of new keywords added to C++ that are not in C. This is a backwards compatibility issue since C code using these keywords as identifiers are not valid C++ programs.

- The type of a character literal is changed from int to char. This change disambiguates ints from chars, and allows for improved function overloading, but C code that is dependent on "sizeof('x') == sizeof(int)" will break.

6. The grammar fragment presented in section 2.13.4 (Floating literals) [Page 27] is:

floating-literal:

fractional-constant exponent-partopt floating-suffixopt

digit-sequence exponent-part floating-suffixopt

fractional-constant:

digit-sequenceopt. digit-sequence

digit-sequence .

exponent-part:

e signopt digit-sequence

E signopt digit-sequence

sign: one of

+ -

digit-sequence:

digit

digit-sequence ’opt digit

floating-suffix: one of

f l F L

In BNF, this grammar is:

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