**Review Questions**

**2.      What is a ternary operator?**

ternary operator is  conditional expressions that can be used anywhere in a program (in a C-based language) where any other expression can be used. In addition to the C-based languages, conditional expressions are provided in Perl, JavaScript, and Ruby.

**3.      What is a prefix operator?**

prefix operators is the operators that precede the operands.

**5.      What is a nonassociative operator?**

Operator with illegal expression.

**9.      What is a coercion?**

coercion is an automatic conversion. For example, if an int variable and a float variable are added in Java, the value of the int variable is coerced to float and a floating-point add is done.

**18.  What is short circuit evaluation?**

A short-circuit evaluationof an expression is one in which the result is determined without evaluating all of the operands and/or operators. For example, the value of the arithmetic expression.

**25.  What mixed mode assignments are allowed in Ada?**

Ada does not allow mixed-mode assignment.

**26.  What mixed mode assignments are allowed in Java?**

C++, Java and C# allow mixed-mode assignment only if the required coercion is widening. So, an **int**value can be assigned to a **float**variable, but not vice versa. Disallowing half of the possible mixedmode assignments is a simple but effective way to increase the reliability of Java and C#, relative to C and C++.

**28.  What is a cast?**

Cast is explicit type conversions. To specify a cast, the desired type is placed in parentheses just before the expression to be converted.

**Problem Set**

**1.      When might you want the compiler to ignore type differences in an expression?**

Suppose Type1 is a subrange of Integer. It may be useful for the difference between Type1 and Integer to be ignored by the compiler in an expression.

**7.      Describe a situation in which the add operator in a programming language would not be commutative.**

 An expression such as a + fun(b)

**8.      Describe a situation in which the add operator in a programming language would not be associative.**

Consider the integer expression A + B + C. Suppose the values of A, B, and C are 20,000, 25,000, and -20,000, respectively. Further suppose that the machine has a maximum integer value of 32,767. If the first addition is computed first, it will result in overflow. If the second addition is done first, the whole expression can be correctly computed.

**9.      Assume the following rules of associativity and precedence for expressions:**

***Precedence Highest*\*, /, not**

**+, –, &, mod**

**– (unary)**

**=, /=, < , <=, >=, >**

**and**

***Lowest*or, xor**

***Associativity Left to right***

**Show the order of evaluation of the following expressions by parenthesizing**

**all subexpressions and placing a superscript on the right parenthesis**

**to indicate order. For example, for the expression**

**a + b \* c + d**

**the order of evaluation would be represented as**

**((a + (b \* c)1)2 + d)3**

**a. a \* b – 1 + c**

**b. a \* (b – 1) / c mod d**

**c. (a – b) / c & (d \* e / a – 3)**

**d. -a or c = d and e**

**e. a > b xor c or d <= 17**

**f. -a + b**

(a) ( ( ( a \* b )1 – 1 )2 + c )3

(b) ( ( ( a \* ( b – 1 )1 )2 / c )3 mod d )4

(c) ( ( ( a – b )1 / c )2 & ( ( ( d \* e )3 / a )4 – 3 )5 )6

(d) ( ( ( – a )1 or ( c = d )2 )3 and e )4

(e) ( ( a > b )1 xor ( c or ( d <= 17 )2 )3 )4

(f) ( – (a + b)1 )2

**10.  Show the order evaluation of the expression of problem 9, assuming that there are no precedence rules and all operators associate right to left.**

(a) ( a \* ( b – ( 1 + c )1 )2 )3

(b) ( a \* ( ( b – 1 )2 / ( c mod d )1 )3 )4

(c) ( ( a – b )5 / ( c & ( d \* ( e / ( a – 3 )1 )2 )3 )4 )6

(d) ( – ( a or ( c = ( d and e )1 )2 )3 )4

(e) ( a > ( xor ( c or ( d <= 17 )1 )2 )3 )4

(f) ( – ( a + b )1 )2

**11.  Write a BNF description of the precedence and associativity rules defined for the expressions in Problem 9. Assume the only operands are the names a,b,c,d, and e.**

<expr> → <expr> or <e1> | <expr> xor <e1> | <e1>

<e1> → <e1> and <e2> | <e2>

<e2> → <e2> = <e3> | <e2> /= <e3> | <e2> < <e3>

| <e2> <= <e3> | <e2> > <e3> | <e2> >= <e3> | <e3>

<e3> → <e4>

<e4> → <e4> + <e5> | <e4> – <e5> | <e4> & <e5> | <e4> mod <e5> | <e5>

<e5> → <e5> \* <e6> | <e5> / <e6> | not <e5> | <e6>

<e6> → a | b | c | d | e | const | ( <expr> )