Table 1: Referring to classes, methods and data members

1	`class Son extends Father`
	`{`
	`  int bar;`
	`}`
	`class Father`
	`{`
	`  void foo() { PrintInt(8); }`
	`}`
		ERROR

2	`class Edge`
	`{`
	`  Vertex u;`
	`  Vertex v;`
	`}`
	`class Vertex`
	`{`
	`  int weight;`
	`}`
		ERROR

3	`class UseBeforeDef`
	`{`
	`  void foo() { bar(8); }`
	`  void bar(int i) { PrintInt(i); }`
	`}`
		ERROR

4	`class UseBeforeDef`
	`{`
	`  void foo() { PrintInt(i); }`
	`  int i;`
	`}`
		ERROR

Table 2: Method overloading and variable shadowing are both illegal in **L**.

1	`class Father`
	`{`
	`  int foo() { return 8; }`
	`}`
	`class Son extends Father`
	`{`
	`  void foo() { PrintInt(8); }`
	`}`
		ERROR

2	`class Father`
	`{`
	`  int foo(int i) { return 8; }`
	`}`
	`class Son extends Father`
	`{`
	`  int foo(int j) { return j; }`
	`}`
		OK

3	`class IllegalSameName`
	`{`
	`  void foo() { PrintInt(8); }`
	`  void foo(int i) { PrintInt(i); }`
	`}`
		ERROR

4	`class IllegalSameName`
	`{`
	`  int foo;`
	`  void foo(int i) { PrintInt(i); }`
	`}`
		ERROR

5	`class Father`
	`{`
	`  int foo;`
	`}`
	`class Son extends Father`
	`{`
	`  string foo;`
	`}`
		ERROR

6	`class Father`
	`{`
	`  int foo;`
	`}`
	`class Son extends Father`
	`{`
	`  void foo() { }`
	`}`
		ERROR

Table 3: Class Son is a semantically valid input for foo.

`class Father { int i; }`
`class Son extends Father { int j; }`
`void foo(Father f) { PrintInt(f.i); }`
`void main(){ Son s; foo(s); }`
	OK

Table 4: nil sent instead of a (Father) class is semantically allowed.

`class Father { int i; }`
`void foo(Father f){ PrintInt(f.i); }`
`void main(){ foo(nil); }`
	OK

Table 5: Non interchangeable array types.

`array gradesArray = int[];`
`array IDsArray = int[];`
`void F(IDsArray ids){ PrintInt(ids[6]); }`
`void main()`
`{`
`  IDsArray ids := new int[8];`
`  gradesArray grades := new int[8];`
`  F(grades);`
`}`
	ERROR

Table 6: nil sent instead of an integer array is semantically allowed.

`array IntArray = int[];`
`void F(IntArray A){ PrintInt(A[8]); }`
`void main(){ F(nil); }`
	OK

Table 7: Assignments.

1	`class Father { int i; }` `Father f := nil;`	OK

2	`class Father { int i; }`
	`class Son extends Father { int j; }`
	`Father f := new Son;`
		OK

3	`class Father { int i; }` `class Son extends Father { int j := 8; }`	OK
4	`class Father { int i := 9; }` `class Son extends Father { int j := i; }`	ERROR
5	`class Father { int foo() { return 90; } }` `class Son extends Father { int j := foo(); }`	ERROR

6	`class IntList`
	`{`
	`  int head := -1;`
	`  IntList tail := new IntList;`
	`}`
		ERROR

7	`class IntList`
	`{`
	`  IntList tail;`
	`  void Init() { tail := new IntList; }`
	`  int head;`
	`}`
		OK

8	`array gradesArray  = int[];`
	`array IDsArray  = int[];`
	`IDsArray  i := new int[8];`
	`gradesArray g := new int[8];`
	`void foo() { i := g; }`
		ERROR

9	`string s := nil;`	ERROR

Table 8: Equality testing.

1	`class Father { int i; int j; }`
	`int Check(Father f)`
	`{`
	`  if (f = nil)`
	`  {`
	`  return 800;`
	`  }`
	`  return 774;`
	`}`
		OK

2	`int Check(string s)`
	`{`
	`  return s = “LosPollosHermanos”;`
	`}`
		OK

3	`array gradesArray = int[];`
	`array IDsArray  = int[];`
	`IDsArray  i:= new int[8];`
	`gradesArray g:=new int[8];`
	`int j := i = g;`
		ERROR

4	`string s1;`
	`string s2 := “HankSchrader”;`
	`int i := s1 = s2;`
		OK

Table 9: Binary Operations.

1	`class Father`
	`{`
	`  int foo() { return 8/0; }`
	`}`
		ERROR

2	`class Father { string s1; string s2; }`
	`void foo(Father f)`
	`{`
	`  f.s1 := f.s1 + f.s2;`
	`}`
		OK

3	`class Father { string s1; string s2; }`
	`void foo(Father f)`
	`{`
	`  int i := f.s1 < f.s2;`
	`}`
		ERROR

4	`class Father { int j; int k; }`
	`int foo(Father f)`
	`{`
	`  int i := 620;`
	`  return i < f.j;`
	`}`
		OK

Table 10: Scope Rules.

1	`int salary := 7800;`
	`void foo()`
	`{`
	`  string salary := “six”;`
	`}`
		OK

2	`int salary := 7800;`
	`void foo(string salary)`
	`{`
	`  PrintString(salary);`
	`}`
		OK

3	`void foo(string salary)`
	`{`
	`  int salary := 7800;`
	`  PrintString(salary);`
	`}`
		ERROR

4	`string myvar := “ab”;`
	`class Father`
	`{`
	`  Father myvar := nil;`
	`  void foo()`
	`  {`
	`  int myvar := 100;`
	`  PrintInt(myvar);`
	`  }`
	`}`
		OK

5	`int foo(string s) { return 800;}`
	`class Father`
	`{`
	`  string foo(string s)`
	`  {`
	`  return s;`
	`  }`
	`  void Print()`
	`  {`
	`  PrintString(foo(“Jerry”));`
	`  }`
	`}`
		OK