Appendix B

Solutions

Solution to Exercise 1. (d) and (f) are not identifiers, since they do not start with a letter. (g) is not an identifier, since it contains the character `#`. (b) is not allowed as a variable name, but it is a valid identifier.

Solution to Exercise 2. (c) is not an expression, since the first operand of the assignment operator must be an lvalue, but `1` is a literal, hence an rvalue. (f) is not an expression, since there is no closing parenthesis for the opening one. (h) is invalid, since `(a*3)` is an rvalue, but the left operand of the assignment operator must be an lvalue.

Solution to Exercise 3. (a), (e), and (g) are rvalues by definition of the binary multiplication operator. (b) and (d) are lvalues by definition of the assignment operator.

Solution to Exercise 4. (a) has value 6, obtained by multiplying the value of the primary expression `1` with the value of the composite expression `(2*3)`. The latter value is 6, for the same reason. (b) has value 5, obtained by assigning value 5 to `b` first (right assignment), and then to `a` (left assignment). (d) has value 1, by definition of the assignment operator. (e) has value 35, since the operands `(a=5)` and `(b=7)` have values 5 and 7, respectively.

In case of (g), the value is unspecified. If the right operand is evaluated first, we get value 25, but if the left operand comes first, `b` may have some value other than 5, and the left operand evaluates to this other value. The final result will not be 25, then.

Solution to Exercise 5.

```
// Program: multhree.C
// Compute the product of three numbers.

#include <iostream>

int main()
{
  // input of a, b and c
  std::cout << "Compute a * b * c for a =? ";
  int a;
```



```
  std::cin >> a;

  std::cout << "... and b =? ";
  int b;
  std::cin >> b;

  std::cout << "... and c =? ";
  int c;
  std::cin >> c;

  // output a * b * c,
  std::cout << a << " * " << b  << " * " << c << " = "
            << a * b * c << ".\n";
  return 0;
}
```

Solution to Exercise 6.

```
// Program: power20.C
// Raise a number to the power twenty.

#include <iostream>

int main()
{
  // input
  std::cout << "Compute a^20 for a =? ";
  int a;
  std::cin >> a;

  // computation
  int b = a * a; // b = a^2
  int c = b * b; // c = a^4
  int d = c * c; // d = a^8
  int e = d * d; // e = a^16

  // output e * c, i.e. a^20
  std::cout << a << "^20 = " << e * c << ".\n";
  return 0;
}
```

Solution to Exercise 7. For the lower bound in a), we argue by induction that $a_i \leq a^{2^i}$ for all i (we can't do more than double the number in each step). In order to get $a_t = n$, we therefore must have

$$a^{2^t} \geq a_t = a^n,$$

or $2^t \geq n$. It follows that

$$t \geq \lg n \geq \lfloor \lg n \rfloor = \lambda(n).$$

For the upper bound, we have to come up with a computation for a^n that needs at most $\lambda(n) + \nu(n) - 1$ steps. This is simple (and called the binary method). By doubling a $\lambda(n)$ times, we can compute in $\lambda(n)$ steps all powers of the form a^{2^i} that are less or equal to a^n. For example, in $\lambda(20) = 4$ steps, we can get the values a, a^2, a^4, a^8, a^{16}.

Since n is the sum of exactly $\nu(n)$ of these 2^i, a^n is the *product* of exactly $\nu(n)$ of these a^{2^i} (this is a simple consequence of the formula $a^{n+m} = a^n \cdot a^m$). It follows that we can obtain a^n by simply multiplying these $\nu(n)$ values together, and since we already have them, this can be done in $\nu(n) - 1$ further multiplications.

For b), we give an example where the upper bound is not tight. Consider $n = 15$ (1111 in binary). We have $\lambda(15) = 3$ and $\nu(15) = 4$, so the binary method would need 6 multiplications. But we can do it with five multiplications, as follows:

```
a1 = a0 * a0   // a^2
a2 = a1 * a0   // a^3
a3 = a2 * a2   // a^6
a4 = a3 * a3   // a^12
a5 = a4 * a2   // a^15
```

In general, no exact formula for $\ell(n)$ is known.