Solution to Exercise 11. Types of intermediate results are either `int` or `unsigned int` (indicated with `u`).

a) `-2-4*3` is parenthesized as `(-2)-(4*3)` since unary minus binds more strongly than binary minus, and since multiplicative operators bind more strongly than additive ones. Evaluation: `(-2)-(4*3)` $\rightarrow$ `(-2)-12` $\rightarrow$ `-14`.

b) `10%6*8%3` is parenthesized as `((10%6)*8)%3` since binary (multiplicative) arithmetic operators are left-associative. Evaluation: `((10%6)*8)%3` $\rightarrow$ `(4*8)%3` $\rightarrow$ `(32)%3` $\rightarrow$ `2`.

c) `6-3+4*5` is parenthesized as `(6-3)+(4*5)` since multiplicative operators bind more strongly than additive ones, and since (additive) arithmetic operators are left-associative. Evaluation: `(6-3)+(4*5)` $\rightarrow$ `3+(4*5)` $\rightarrow$ `3+20` $\rightarrow$ `23`.

d) `5+5*3u` is parenthesized as `5+(5*3u)` since multiplicative operators bind more strongly than additive ones. Evaluation: `5+(5*3u)` $\rightarrow$ `5+15u` $\rightarrow$ `20u`. Here, `int` values are implicitly converted to `unsigned int`.

e) `31/4/2` is parenthesized as `(31/4)/2` since (multiplicative) arithmetic operators are left-associative. Evaluation: `(31/4)/2` $\rightarrow$ `7/2` $\rightarrow$ `3`.

f) `-1-1u+1-(-1)` is parenthesized as `(((-1)-1u)+1)-(-1)` since (additive) arithmetic operators are left-associative. Evaluation: `(((-1)-1u)+1)-(-1)` $\rightarrow$ `(((`2^b-1`)u-1u)+1)-(-1)` $\rightarrow$ `((`2^b-2`)u+1)-(-1)` $\rightarrow$ `(`2^b-1`)u-(-1)` $\rightarrow$ `(`2^b-1`)u-(`2^b-1`)u` $\rightarrow$ `0u`, due to the conversion rules.

Solution to Exercise 12. *b)* and *g)* are illegal because `-a` and `7+a`, respectively, are rvalues and the left operand of an assignment must be an lvalue. For the others, the paranthese are as follows.

a)	`c=((a+7)+(--b))`			*c)*	`c=(a=(-b))`
d)	`a-((a/b)*b)`	*e)*	`b*=((++a)+b)`	*f)*	`a-(a*(+(-b)))`
		h)	`(a+(3*(--b)))+(a++)`	*i)*	`(b++)+(--a)`

Solution to Exercise 13. *h)* induces unspecified behavior because it is not specified whether the increment of `a` happens before or after the first `a` is evaluated. The remaining expressions can be evaluated as follows.

a) `c=((a+7)+(--b))` $\rightarrow$ `c=((5+7)+1)` $\rightarrow$ `c=(12+1)` $\rightarrow$ `c=13` $\rightarrow$ 13.

c) `c=(a=(-b))` $\rightarrow$ `c=(a=(-2))` $\rightarrow$ `c=(a=`-2`)` $\rightarrow$ `c=`-2 $\rightarrow -2$.

d) `a-((a/b)*b)` $\rightarrow$ `5-((5/2)*2)` $\rightarrow$ `5-(2*2)` $\rightarrow$ `5-(4)` $\rightarrow$ 1.

e) `b*=((++a)+b)` $\rightarrow$ `b*=(6+2)` $\rightarrow$ `b*=`8 $\rightarrow$ 16.

f) `a-(a*(+(-b)))` $\to$ `5-(5*(+(-2)))` $\to$ `5-(5*(+-2))` $\to$ `5-(5*-2)` $\to$ `5--10` $\to$ 15.

i) `(b++)+(--a)` $\to$ `2+4` $\to$ 6.

Solution to Exercise 14. For any positive d, we can uniquely write n in the form

$$n = xd + y,$$

where $x, y \in \mathbb{N}$ and $y < d$. In fact, these define div and mod via $x = n \operatorname{div} d$ and $y = n \bmod d$.

For $n = a, d = bc$, we get

$$a = pbc + q,$$

where $p, q \in \mathbb{N}$ and $q < bc$. In particular, $p = a \operatorname{div}(bc)$. It remains to prove that

$$(a \operatorname{div} b) \operatorname{div} c = p.$$

To this end, we use $n = q, d = b$ to get

$$q = rb + s,$$

where $r, s \in \mathbb{N}, s < b$ and $r < c$ (since $rb \leq q < cb$). Then we have

$$a = (pc + r)b + s, s < b,$$

and this means that

$$a \operatorname{div} b = pc + r, r < c.$$

This in turn means that $p = (a \operatorname{div} b) \operatorname{div} c$.

The implication holds as long as the mathematical value of the expression `b*c` is representable as an `unsigned int` value. This holds since the other operations involving `/` are always error-free. If there is an overflow in `b*c`, the expressions `a/b/c` and `a/(b*c)` might yield different values.

Solution to Exercise 15. *a)* 1111 *b)* 10101100 *c)* 101001001 *d)* 1111111110

Solution to Exercise 16. *a)* 55 *b)* 65 *c)* 233 *d)* 341

Solution to Exercise 17. We first need the fact that a natural number $n \geq 1$ has $\lfloor \log_{10} n \rfloor + 1$ decimal digits ($\lfloor x \rfloor$ is x rounded down to the next smaller integer). This is not hard to see: if n has decimal expansion

$$n = \sum_{i=0}^{\infty} b_i 10^i,$$

the largest i with nonzero b_i occurs for the unique value i_0 such that $10^{i_0} \leq n < 10^{i_0+1}$. By taking logarithms, this is equivalent to $i_0 \leq \log_{10} n < i_0 + 1$, meaning that $i_0 = \lfloor \log_{10} n \rfloor$. Since we now have

$$n = \sum_{i=0}^{i_0} b_i 10^i,$$

with $b_{i_0} \neq 0$, n has exactly $\lfloor \log_{10} n \rfloor + 1$ decimal digits.

Now for the actual question: first, it is not hard to see that the two numbers $2^{43,112,609} - 1$ and $2^{43,112,609}$ have the same number of decimal digits, since a power of two always has a last decimal digit $2, 4, 8$, or 6.

The number of digits is therefore

$$\begin{aligned} \lfloor \log_{10} 2^{43,112,609} \rfloor + 1 &= \lfloor 43,112,609 \cdot \log_{10} 2 \rfloor + 1 \\ &= \lfloor 43,112,609 \cdot 0.3010299957 \rfloor + 1 \\ &= \lfloor 12,978,188.5 \rfloor + 1 \\ &= 12,978,189. \end{aligned}$$

We have to be a bit careful here: if we compute $\log_{10} 2$ with insufficient precision, we may easily get a wrong answer. In fact, how can we be sure that the above answer is correct? Assuming that 0.3010299957 is correct up to possibly a difference of one in the last digit, we have an additive error of at most 10^{-10} in this approximation of $\log_{10} 2$. Even if we multiply this approximation with the large number $43,112,609$, the additive error is therefore at most around 0.0043 which is not enough to change the value of $\lfloor 12,978,188.5 \rfloor$.

These observations imply that we need to have $\log_{10} 2$ rounded to eight digits in order to guarantee the correct value. Most pocket calculators will be able to compute $\log_{10} 2$ up to that precision.

Solution to Exercise 18.

a) 0110 *b)* 1100 *c)* 1000 *d)* not representable *e)* 1101

Solution to Exercise 19. We start by solving the Celsius-to-Fahrenheit equation for Degrees Celsius and get

$$\text{Degrees Celsius} = \frac{5(\text{Degrees Fahrenheit} - 32)}{9}.$$

The critical operation is the multiplication by 5. If $\underline{m}, \overline{m}$ are the smallest and largest representable `int` values, we must guarantee that

$$\underline{m} \leq 5(\text{Degrees Fahrenheit} - 32) \leq \overline{m}$$

in order to avoid over- and underflows. Solving this for Degrees Fahrenheit yields the following bounds.

$$\frac{m}{5} + 32 \leq \text{Degrees Fahrenheit} \leq \frac{\overline{m}}{5} + 32.$$

Since integer division rounds down for positive numerators, a valid upper bound is obtained from the expression

```
std::numeric_limits<int>::max() / 5 + 32
```

For the lower bound, we have assumed for this exercise that the integer division rounds towards zero (up) for negative numerators. Therefore, the value

```
std::numeric_limits<int>::min() / 5 + 32
```

is a valid lower bound. The resulting code is given below (the output of the mixed number simply employs the div and mod operators).

```
1  // Program: celsius.C
2  // Convert temperatures from Fahrenheit to Celsius.
3
4  #include<iostream>
5  #include<limits>
6
7  int main ()
8  {
9    // Input
10   std::cout << "Temperature in degrees Fahrenheit =? \n"
11             << " (from the interval ["
12             << std::numeric_limits<int>::min() / 5 + 32
13             << ", "
14             << std::numeric_limits<int>::max() / 5 + 32
15             << "])\n";
16   int fahrenheit;
17   std::cin >> fahrenheit;
18
19   // Computation
20   int ncelsius = 5 * (fahrenheit - 32); // numerator
21
22   // Output as mixed number
23   std::cout << fahrenheit << " degrees Fahrenheit are "
24             << ncelsius / 9 << " " << ncelsius % 9
25             << "/9 degrees Celsius.\n";
26
27   return 0;
28 }
```

Solution to Exercise 20. According to Section 2.2.8, the least significant binary digit b_0 of n is $n \bmod 2$, and the remaining digits are obtained by applying the same technique to $(n - b_0)/2 = n \operatorname{div} 2$. It follows that the second to last digit is $(n \operatorname{div} 2) \bmod 2$, and that the third to last one is $((n \operatorname{div} 2) \operatorname{div} 2) \bmod 2$, or (as one can prove; nice Exercise for next time) $(n \operatorname{div} 4) \bmod 2$.

```
1  // Program: threebin.C
2  // Output the last three binary digits of a number n.
```

```
 3
 4  #include<iostream>
 5
 6  int main ()
 7  {
 8    // input
 9    std::cout << "Last three binary digits of n =? ";
10    unsigned int n;
11    std::cin >> n;
12
13    // computation and output
14    std::cout << "The digits are "
15              << n / 2 / 2 % 2 << n / 2 % 2 << n % 2 << ".\n";
16
17    return 0;
18  }
```

Solution to Exercise 21. For part a), we first have to find a theoretical way of computing the survivor $p(k)$. One way of doing it is to generalize the problem first. Let $p_n(k)$ be the last survivor in a circle of n people when every k-th person is killed. Then $p(k) = p_{41}(k)$. Here is the crucial insight that leads to a recursive formula for $p_n(k)$:

$$p_n(k) = (p_{n-1}(k) + k) \bmod n, \quad n \geq 2,$$

where $p_1(k) = 0$ (in a circle of 1, person 0 is the only person and therefore the last survivor). To see that this formula holds, we consider the n-circle after the first person (numbered $k-1$) has been killed. We now have an $(n-1)$-circle, and if we renumber the positions in such a way that the position formerly being k is now position 0, $p_{n-1}(k)$ tells us the last survivor in this new numbering. To get the survivor $p_n(k)$ in the old numbering, we simply have to add k, *but modulo* n. This takes a little thought but is easily deduced from the following table.

old numbering	0	1	$\cdots$	$k-2$	$k-1$	k	$\cdots$	$n-1$
new numbering	$n-k$	$n-k+1$	$\cdots$	$n-2$	$-$	0	$\cdots$	$n-k-1$

Here is the program that determines $p(k)$ by iteratively computing the sequence $p_1(k), p_2(k), ..., p_{41}(k)$, according to the formula above.

```
 1  // Program: josephus.C
 2  // determines the survivor number in the Josephus Problem
 3
 4  #include<iostream>
 5
 6  int main ()
 7  {
 8    // input
 9    std::cout << "Every k-th person is killed for k =? ";
10    unsigned int k;
11    std::cin >> k;
12
13    // computation of p_41(k)
14    unsigned int n = 1; // circle size, runs through 1,...,41
15    unsigned int p = 0; // survivor number, runs through p_1(k),...,p_{41}(k)
```

```
16
17    // each of the following 40 identical statements increments n and updates
18    // p according to the formula p_n(k) = (p_{n-1}(k) + k) mod n
19    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
20    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
21    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
22    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
23    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
24    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
25    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
26    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
27    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
28    p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n; p = (p + k) % ++n;
29
30    // now p = p_{41}(k) = p(k); output
31    std::cout << "The survivor is the person number " << p << ".\n";
32
33    return 0;
34  }
```

Part b) is solved by playing with the program a little, and the answer is no. There are 41 possible initial positions for Josephus, and 41 ways of choosing k. Both for $k = 3$ and for $k = 7$, the survivor number is 30, so in the sequence $p(1), \ldots, p(41)$, at least one of the 41 number $\{0, \ldots, 40\}$ appears twice. But this implies that there must also be at least one other number in $\{0, \ldots, 40\}$ that does not appear at all. The initial position with this number is therefore fatal for all $k \in \{1, \ldots, 41\}$.

Interestingly, if we allow k to be *any* natural number, then it is possible to survive from any initial position.

Solution to Exercise 22.

```
1   // Prog: pythagoras.C
2   // helper program for computing all positive integers a,b,c
3   // such that
4   //      a^2 + b^2 = c^2 and a + b + c = 1000
5   #include<iostream>
6
7   // Here is our 'Ansatz':
8   // Substituting c = 1000 - (a+b) into a^2 + b^2 = c^2 implies
9   //
10  //                0 = 1000000 - 2000(a+b) + 2ab,
11  //    <=>    ab/1000 = a+b - 500 (*)
12  //
13  // This means that ab must be divisible by 1000 = 2^3*5^3. Therefore,
14  // one of a and b must be divisible by 5^2 = 25. Because the situation
15  // is completely symmetric in a and b, we will assume that it's a. In
16  // other words, we must only check the 39 values a=25,50,75,...,975.
17  //
18  // Given a, (*) can be used to compute b = 1000 - 500000/(1000-a).
19  // Since b must be integer, 500000/(1000-a) must be integer as well.
20  //
21  // The following outputs the four values
22  //    r = 500000 mod (1000-a),
23  //    a,
24  //    b = 1000 - 500000/(1000-a),
25  //    c = 1000 - a - b
26  // for all 39 values of a that need to be considered. (a,b,c) is
27  // a solution if and only if r = 0 and b,c > 0.
28
```

```
29  int main()
30  {
31    int a = 25;
32    int r;
33    int b;
34    int c;
35    // now come 39 copies of the same two lines
36    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
37    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
38    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
39    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
40    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
41    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
42    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
43    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
44    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
45    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
46    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
47    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
48    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
49    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
50    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
51    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
52    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
53    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
54    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
55    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
56    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
57    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
58    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
59    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
60    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
61    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
62    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
63    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
64    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
65    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
66    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
67    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
68    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
69    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
70    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
71    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
72    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
73    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
74    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
75    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
76    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
77    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
78    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
79    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
80    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
81    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
82    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
83    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
84    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
85    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
86    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
87    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
88    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
89    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
90    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
91    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
92    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
93    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
```

```
 94    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
 95    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
 96    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
 97    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
 98    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
 99    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
100    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
101    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
102    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
103    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
104    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
105    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
106    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
107    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
108    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
109    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
110    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
111    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
112    r = 500000 % (1000-a); b = 1000 - 500000 / (1000 - a); c = 1000 - a - b;
113    std::cout << r << ", " << a << ", " << b << ", " << c << "\n"; a +=25;
114
115    // The unique solution is {a, b} = {375, 200}, c = 425
116    return 0;
117 }
```