

**Solution to Exercise 8.** 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 9.** *b*) and *g*) are illegal because `-a` and `7+a`, respectively, are `r` values and the left operand of an assignment must be an `l` value. 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 10.** *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)  $a-(a*(+(-b))) \rightarrow 5-(5*(+(-2))) \rightarrow 5-(5*(+-2)) \rightarrow 5-(5*-2) \rightarrow 5--10 \rightarrow 15$ .
- i)  $(b++)+(-a) \rightarrow 2+4 \rightarrow 6$ .

**Solution to Exercise 11.** 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  $x = n \text{ div } d$  and  $y = n \text{ mod } d$ .

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

$$a = pbc + q,$$

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

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

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

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

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

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 12.** a) 1111 b) 10101100 c) 101001001 d) 1111111110

**Solution to Exercise 13.** a) 55 b) 65 c) 233 d) 341

**Solution to Exercise 14.**

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

**Solution to Exercise 15.** 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.

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

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 16.** 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  $(n - b_0)/2 = n \text{ div } 2$ . It follows that the second to last digit is  $(n \text{ div } 2) \bmod 2$ , and

that the third to last one is  $((n \text{ div } 2) \text{ div } 2) \bmod 2$ , or (as one can prove; nice Exercise for next time)  $(n \text{ div } 4) \bmod 2$ .

---

```

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 17.** 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	...	$k - 2$	$k - 1$	$k$	...	$n - 1$
new numbering	$n - k$	$n - k + 1$	...	$n - 2$	-	0	...	$n - k - 1$

Here is the program that determines  $p(k)$  by iteratively computing the sequence  $p_1(k), p_2(k), \dots, p_{41}(k)$ , 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  $p_1(k), \dots, p_{41}(k)$ 
16
17  // each of the following 40 identical statements increments n and updates
18  // p according to the formula  $p_n(k) = (p_{n-1}(k) + k) \bmod n$ 
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  $p(1), \dots, p(41)$ , at least one of the 41 number  $\{0, \dots, 40\}$  appears twice. But this implies that there must also be at least one other number in  $\{0, \dots, 40\}$  that does not appear at all. The initial position with this number is therefore fatal for all  $k \in \{1, \dots, 41\}$ .

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