Solution 1

Before starting with the actual solution notice that whenever $h$ is of type `int` the function `foo` swaps its arguments. And whenever $h$ is of type `int&` the function `foo` is equivalent to

```cpp
void foo (A a, B b) {
    a = b;
    b = a;
}
```

a) 23, since the `foo`-function only changes local variables.

b) 23, since again the `foo`-function has no reference to non-local variables.

c) 22 Since $h$ is of non-reference type the function `foo` simply swaps the values of its arguments $a$ and $b$. But since only $b$ is of reference type, the variable $a$ in the `main`-function is not changed. This has the effect that `foo` simply copies the value of $a$ into $b$.

d) 23 Since $h$ is a reference, the body of `foo` is equivalent to

```cpp
    a = b;
    b = a;
```

And here we immediately see that $b$ is assigned the value it had before. And since $b$ is a reference to $b$ in the `main`-function, we are overwriting $b$ in the `main`-function with its own value.

e) 33 Because $h$ is of non-reference type, this case behaves the same way as c), with the exception that this time the other argument is a reference to $a$ in the `main`-function.

f) 33 Again since $h$ is a reference to $a$, the function body is equivalent to

```cpp
    a = b;
    b = a;
```

In the first line of this reduced function body we are assigning the value of $b$ to $a$, the latter of which just represents $a$ from the `main`-function.

g) 32, since `foo` just swaps its arguments which are both references.
h) 33 Again since \( h \) is a reference to \( a \), the function body is equivalent to
\[
\begin{align*}
a &= b; \\
b &= a;
\end{align*}
\]

And here we see that we first assign the value of \( b \) to \( a \), the latter of which just represents \( a \) from the main-function and the former of which just represents \( b \) from the main-function. Then in the second line of the reduced function body we simply overwrite the value of \( b \) with the value it had before (since \( a \) was assigned the value of \( b \) in the line before).

Solution 2

a) Note that instead of using a switch-statement, you could for example also implement an approach which is based on multiple if-statements. However, this is a typical use case for a switch-statement.

```cpp
// PRE: dir is one of 'u', 'd', 'l', 'r'
// POST: robot position (x, y) is updated according to dir
void move (int & x, int & y, const char dir)
{
    switch (dir) {
    case 'u':
        // move up
        ++y; break;
    case 'd':
        // move down
        --y; break;
    case 'l':
        // move left
        --x; break;
    case 'r':
        // move right
        ++x; break;
    default:
        assert (false);
    }
}
```

b) 1 // Informatik – Serie 7 – Aufgabe 2
2 // Programm: robopath.cpp
3 // Autor: ... (Gruppe ...)
4 // Prints the target position of a simple robot that starts
5 // at (0, 0) and walks in horizontal and vertical direction.
6
7 //#include "tests.h"
8 #include <iostream>
```cpp
#include <cassert>

// PRE: dir is one of 'u', 'd', 'l', 'r'
// POST: robot position (x, y) is updated according to dir
void move (int& x, int& y, const char dir)
{
    switch (dir) {
    case 'u':
        // move up
        ++y; break;
    case 'd':
        // move down
        --y; break;
    case 'l':
        // move left
        --x; break;
    case 'r':
        // move right
        ++x; break;
    default:
        assert (false);
    }
}

int main ()
{
    // Input
    unsigned int n;
    std::cin >> n;

    // Position
    int x = 0;
    int y = 0;
    for (unsigned int i = 0; i < n; ++i) {
        char dir;
        std::cin >> dir;
        move (x, y, dir);
    }

    // Output
    std::cout << "Robot is now at (" << x << ", " << y << ")\n";

    return 0;
}
```
Solution 3

a) Output: true false true false true

Explanation: In the loop we simply read each entry of the array and output its negation.

b) Output: 15

Explanation: In the loop we go over the whole array and sum up the entries.

c) Output: 0 1 2 3 4 3 3 3 3 3

Explanation: First we define the vector vec to have 10 elements each with value 3. Thus at the moment, vec looks as follows:
3 3 3 3 3 3 3 3 3 3
In the first loop we then overwrite the entries with indices 0, 1, 2, 3, 4 each with its index and leave the remaining entries untouched. Thus at the moment, vec looks as follows:
0 1 2 3 4 3 3 3 3 3
Finally, in the last loop we output vec.

d) Output: 128 64 32 16 8 4 2 1

Explanation: We first define the vector vec to have n entries (i.e. 8 according to the exercise description) which are all initialized with 0. Thus at the moment, vec looks as follows:
0 0 0 0 0 0 0 0
Then we set the last element (i.e. with index 7) to 1. Thus at the moment, vec looks as follows:
0 0 0 0 0 0 0 1
In the first loop we iterate through vec from the end to the beginning and write in each element 2 times the value of the element whose index is larger by 1 (the next one towards the end). Filling up vec looks as follows:
0 0 0 0 0 0 1
0 0 0 0 0 2 1
0 0 0 0 4 2 1
0 0 0 8 4 2 1
...
128 64 32 16 8 4 2 1
Finally, in the last loop we output vec.

Solution 4

Please keep in mind that once again there are multiple ways to solve each subtask.

a)

```
// Informatik − Serie 7 − Aufgabe 4a
```
```cpp
#include "tests.h"
#include <iostream>

int main()
{
    // Input
    int a[10];
    for (unsigned int i = 0; i < 10; ++i)
        std::cin >> a[i];

    // Output what we have stored
    std::cout << "* ";
    for (unsigned int i = 0; i < 10; ++i)
        std::cout << a[i] << " ";
    std::cout << " *
";

    return 0;
}
```

```cpp
#include "tests.h"
#include <iostream>

int main()
{
    // Input
    int arr[10];
    for (unsigned int i = 0; i < 10; ++i)
        std::cin >> arr[i];

    // Count number of appearances
    const int last = arr[9];
    unsigned int counter = 0;
    for (unsigned int i = 0; i < 10; ++i)
        if (arr[i] == last)
            ++counter;

    // Output
    std::cout << counter << "\n";

    return 0;
}
```
c)

```cpp
#include "tests.h"
#include <iostream>

int main ()
{
    // Input
    int arr[10];
    for (unsigned int i = 0; i < 10; ++i)
        std::cin >> arr[i];

    // Output
    std::cout << "* ";
    for (unsigned int i = 0; i <= 4; ++i)
        std::cout << (arr[i] + arr[9-i]) << " ";
    std::cout << " *
";
    return 0;
}
```