Summarized:

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>int 2</td>
</tr>
<tr>
<td>(ii)</td>
<td>bool false</td>
</tr>
<tr>
<td>(iii)</td>
<td>bool true</td>
</tr>
<tr>
<td>(iv)</td>
<td>bool true</td>
</tr>
<tr>
<td>(v)</td>
<td>bool true</td>
</tr>
</tbody>
</table>

Now about how to obtain these: here we do it in more detail than you’re usually asked for, if they appear in an exam. We first show you the fully-parenthesized expressions indicating in which order subexpressions are evaluated. The parentheses are applied according to the rules of precedence and associativity.

a) The parenthesized expressions are:

(i) \((n / f) * d\)
(ii) \(u == (4 * (++d))\)
(iii) \(((0 < d) < 2) || (2 > f)\)
(iv) \(((0 < 1) && (1 != 1)) || ((++x) > 3)\)
(v) \(((3 * d) > d) || (((1 / z) != 0) && ((3 + 4) >= 7))\)

b) Possible step-by-step evaluations:

(i) \((n / f) * d\) →
   \((n / f) * 2\) →
   \((9 / f) * 2\) →
   \((9 / 5) * 2\) →
   \(1 * 2\) →
   \(2\)

Type: int
(ii) (Note: this exercise was slightly changed compared to the one in the exam.)

\[ u == (4 * (++d)) \rightarrow \]
\[ u == (4 * 3) \rightarrow \]
\[ u == 12 \rightarrow \]
\[ 4 == 12 \rightarrow \]
\[ false \]
Type: bool

(iii) \((0 < d) < 2) || (2 > f) \rightarrow
(0 < 2) || (2 > f) \rightarrow
(true < 2) || (2 > f) \rightarrow
(1 < 2) || (2 > f) \rightarrow
true || (2 > f) \rightarrow
true
Type: bool

Note: Due to short circuit evaluation, the right part of the or statement will not be executed (ever!). If you tried to do so, your solution is wrong (independent of the result you might have found).

(iv) \((0 < 1) && (1 != 1)) || ((++x) > 3) \rightarrow
(true && (1 != 1)) || ((++x) > 3) \rightarrow
(true && false) || ((++x) > 3) \rightarrow
false || ((++x) > 3) \rightarrow
false || (4 > 3) \rightarrow
false || true \rightarrow
true
Type: bool

(v) \((3 * d) > d) || (((1 / x) != 0) && ((3 + 4) >= 7)) \rightarrow
((3 * 2) > d) || (((1 / x) != 0) && ((3 + 4) >= 7)) \rightarrow
(6 > d) || (((1 / x) != 0) && ((3 + 4) >= 7)) \rightarrow
(6 > 2) || (((1 / x) != 0) && ((3 + 4) >= 7)) \rightarrow
true || (((1 / x) != 0) && ((3 + 4) >= 7)) \rightarrow
true
Type: bool

Note: Due to short circuit evaluation, the right part of the or statement will not be executed (ever!). If you tried to do so, your solution is wrong (independent of the result you might have found).

Solution 2

```cpp
// Informatik - Serie 3 - Aufgabe 2
// Programm: cross_sum.cpp
```
// Compute the cross sum of a natural number.

#include "tests.h" // remove slashes at beginning of line to test ← or submit

#include <iostream>

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

    // computation
    unsigned int cross = 0;
    for (unsigned int m = n; m > 0; m /= 10)
        cross += m % 10;

    // output
    std::cout << "Cross sum of " << n << " is: "
              << cross << "\n";

    return 0;
}

Solution 3

// Informatik – Serie 3 – Aufgabe 3
// Programm: dec2bin.cpp
// Autor: ... (Gruppe ...)
// Output reverse binary representation of a number n.

#include "tests.h" // remove slashes at beginning of line to test ← or submit

#include <iostream>

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

    // computation and output
    std::cout << "Reverse binary representation is: ";
    if (n == 0) // special case!
```cpp
std::cout << n;

for (; n > 0; n /= 2) // no output for n==0
    std::cout << n % 2;
std::cout << std::endl;

return 0;
```