Solutions 4

Solution 1

a) Output: 5 4 3 2 1

b) Output: 4 11 12 1 5
   Notice that the number 0 - which ends the loop - is not output anymore.

c) Output: 2 6 10

d) Output: (1,1) (1,2) (1,3) (2,1) (2,2) (2,3) (3,1) (3,2) (3,3)

Solution 2

Please keep in mind that once again there are multiple ways to solve each subtask. Provided below are example code snippets you can insert in each Codeboard template that solve the respective task. Also given is a short explanation for why the particular loops are chosen.

a) A while-loop works well here as we don’t need a variable that is used only in the loop, i.e. we don’t need a loop-specific init-statement.

```cpp
// Input
unsigned int n;
std::cin >> n;

// Computation of the reduced number
while (n >= 1000)
    n -= 1000;

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

Remark: The result is the same as for n % 1000.

b) A for-loop works well because the summation-index i iterates through the numbers \{1, 2, ..., n\} but it is not used after the loop anymore (the sum’s value is used afterwards).```
c) A do-loop works well because the loop-condition depends on the things we repeatedly do (we verify a condition based on the input), i.e. we have to get an input first before we can verify it.

```cpp
unsigned int input;
unsigned int counter = 0;
do {
    std::cin >> input;
    ++counter;
} while (input != 0);
std::cout << counter << "\n";
```

d) Two nested for-loops do the job elegantly here. The outer one indicates the row index and the inner one then prints the stars based on the row index from the outer loop.

```cpp
// Input
unsigned int n;
std::cin >> n;

// Output
for (unsigned int i = 1; i <= n; ++i) {
    for (unsigned int j = 0; j < i; ++j) {
        std::cout << "*";
    }
    std::cout << "\n";
}
```

Solution 3
Solution 4

// Informatik – Serie 4 – Aufgabe 4
// Programm: pi.cpp
// Autor: ... (Gruppe ...)
// Approximate pi according to first n terms of both formulas.

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

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

    // loop over all numbers in the range [1, 1000]
    for (unsigned int n = 1; n <= 1000; ++n) {
        // count the number of divisors of n
        unsigned int divisors = 0;
        for (unsigned int m = 1; m <= n; ++m)
            if (n % m == 0)
                ++divisors;

        // output n if the number of divisors is equal to k
        if (divisors == k)
            std::cout << n << " ";
    }
    std::cout << "\n";
    return 0;
}
unsigned int n;
std::cin >> n;

// computation — formula 1
double pif1 = 0.0;
for (unsigned int i = 1; i < 2*n; i += 2)
    if (i % 4 == 1)
        // case: 1/1 + 1/5 + 1/9 + ...
pif1 += 4.0 / i;
    else
        // case: -1/3 - 1/7 - 1/11 - ...
pif1 -= 4.0 / i;

// computation — formula 2

// auxiliary variables
// initialized for first term of forward sum (i=0)
double numer = 2.0; // numerator i-th term
double denom = 1.0; // denominator i-th term
double pif2 = 2.0; // value after term i (i=0 initially, then i←
                    // 1, 2, ..., n-1)
for (unsigned int i = 1; i < n; ++i) {
    numer *= i;
    denom *= (2*i + 1);
    pif2 += numer / denom; // update to term i
}

// output
std::cout << "Formula 1: " << pif1 << "\n";
std::cout << "Formula 2: " << pif2 << "\n";

return 0;

Notice that when running the first formula for \( n = 10,000 \), it gives on our platform the approximation 3.14149 (still off in the fourth digit after the decimal point). For \( n = 100,000 \), we get 3.14158 (still off in the fifth digit after the decimal point). For \( n = 1,000,000 \), finally, the result is correct to five digits after the decimal point: 3.14159.

On the other hand, the approximation based on the second formula already yields the result 3.14159 for \( n = 17 \) on our platform, so this version is obviously preferable.