Informatik - AS19

Exercise 8: Two-Dimensional vectors, Characters, Recursion

Handout: 4. Nov. 2019 06:00

Due: 11. Nov. 2019 18:00

Task 1: Vector and matrix operations

Open Task

Note: All occurrences of the word *vector* in this assignment without an explicit `std::` prefix mean vector in the mathematical sense.

Task:

Write a program that implements the multiplication of integer vectors and matrices using `std::vector`s. The program must support the following three possible operations:

1. Dot product of two vectors (also called scalar product or inner product)
2. Matrix-vector product (application of a matrix to a column vector)
3. Matrix product

Implement the following program structure:

1. Query for the two operands from the input: the left operand first, then the right operand.
2. If no operations can be applied, output `error`. This happens either if the left operand is a vector and the right operand is a matrix, or if the dimensions do not match.
3. Output the result of the operation.

Specific rules for this task:

1. Use one dimensional `std::vector`s to represent vectors and two dimensional `std::vector`s to represent matrices.
2. Use functions to implement input, output and the three different operations. Use references with proper constness to access `std::vector`s from functions.
3. You may assume that all input vector/matrices have non-zero dimensions, and that input will not contain values big enough to cause overflow.

Format:

The format to represent scalars/vectors/matrices in input/output is the following:

1. A character among `s`, `v` and `m` to denote whether the data represent a scalar, a vector or a matrix (scalar cannot occur in input for this task).
2. A sequence of integer giving the dimensions of the value:
    1. nothing for scalar
    2. a single integer giving the length for vectors
    3. and two integers giving respectively the number of rows and the number of columns for matrices
3. a newline
4. A sequence of integer giving the content of the value:
    1. The scalar itself for scalars
    2. the content of the vector in order for vectors, on a single line
    3. the content of the matrix in row-major order for matrices (the content of the first row in column order first, then the second row, etc). Each row should be placed on a distinct line.

Input:

Two values according to the above format.

Examples:

1. ```
   v 3
   -1 1 2
   v 3
   2 5 2
   ```

2. ```
   m 2 3
    1  1 -5
   -1  0  4
   v 3
   3 5 7
   ```

3. ```
   m 3 2
   2  0
   1 -1
   1  1
   m 2 4
   9  7 -2  4
   5 -2 -1 -3
   ```

4. ```
   v 3
   ```

```
1 2 3
m 1 1
4
```

Output:

The resulting value, according to the above format.

Examples:

1. `s 7`

2. ```
   v 2
   -27 25
   ```

3. ```
   m 3 4
   18 14 -4 8
   4  9  -1 7
   14 5  -3 1
   ```

4. `error`

Note: The scalar values should be returned only by scalar products. Even if a matrix-vector multiplication or a matrix-matrix multiplication results in a single value, it should be represented by a single-element vector or matrix. Similarly, if the result of a matrix-matrix multiplication looks like a vector, it should still be represented as a matrix, with one dimension equal to 1.

Task 2: Decode binary NZZ front page

Open Task

On 8th June 2012, *Neue Zürcher Zeitung* went completely digital, and what they did to visualize this was to encode the whole cover page in binary in the way that each 8-bit binary number represented a single ASCII character (e.g., 01001110 01011010 01011010 encodes NZZ):

The file `nzz.in` provided contains the transcript of the NZZ binary cover page (converted to pure ASCII). The binary data includes newline characters.

Task: Write a program that asks for filename, loads and decodes this file, and outputs the decoded text.

Recall: reading from a file:

1. Include file stream library: `include <fstream>`
2. Opening an input stream from a file: if filename is stored in variable `filename` of type `std::string`,

   ```
   std::ifstream in(filename)
   ```

   declare an input stream named `in`, initialized from file named with the content of `filename`
3. Use `in >> value` to read content from input stream `in`. If stream is empty, attempting to read put the stream in a 'bad' state.
4. Convert the stream to bool (`(bool)(in)`) to test if a stream is in an error state. Alternatively, `(bool)(in >> value)` returns false iff the stream was empty when trying the operation. This is because `in >> value` returns a reference to `in` itself.

Note: If you want to pass a variable of type `ifstream` as parameter to a method, always pass it by reference. Passing it by value may lead to unexpected behavior.

Input

A filename

Example:

```
hello.in
```

Output

The decoded content of the filename

Example:

```
Hello World!
```

Task 3: Recursive function analysis

Open Task

This task is a text based task. You do not need to write any program/C++ file: the answer should be written in main.md *(and might include code fragments if questions ask for them).*

Task

For each of the following recursive functions:

1.

```
bool f(const int n) {
  if (n == 0) return false;
  return !f(n - 1);
}
```

2.

```
void g(const int n) {
  if (n == 0) {
    std::cout << "*";
    return;
  }
  g(n - 1);
  g(n - 1);
}
```

i) Formulate pre- and post conditions.

ii) Show that the function terminates. **Hint:** No proof expected, proceed similar as with the lecture example of the factorial function.

iii) Determine the number of functions calls as mathematical function of parameter n. **Note:** include the first non-recursive function call.