Hey Vergil! Your matrix-computing days are over.

> Author: Artanisax
>
> Keywords: reference & pointer, multi-dimensional operation simulation, shallow/deep copy, ROI

Description

JUNE 15, 05:14 PM

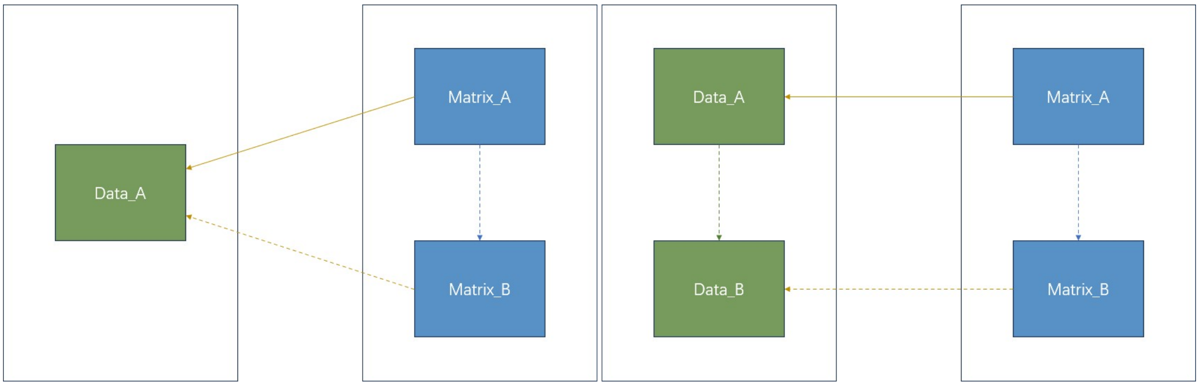
Vergil misses his son Neuro and decides to open a portal with Yamato to see him. Now he needs you to implement a matrix library based on `C++` to help calculate the path.

Since Vergil has little modern knowledge, especially math, you only need to support matrices with `int` entries. You are also required to implement both **shallow/deep copy** and rectangular **ROI** (Region of Interest) features because Vergil desperately wants to save memory for pursuing power.

Dante offers some assistance so the data structures and function declarations have been made and you are only required to complete the function definitions.

Hint

- In this question, you are supposed to use `new/delete` to manage data. Using C-style functions like `malloc/calloc/free` will lead to **Runtime Error** when testing. For this question specifically, you only need to use sentences resemble `mat.data = new Data(row, col);` and `delete mat.data;` to manage `Data` pointers as the jobs inside `Data` have been done by the constructor and destructor.
- You are allowed to implement other functions to help to implement required functions. But we will only invoke the functions we declared during the test process.
- The following picture will roughly illustrate the difference between **shallow copy** and **deep copy**:



- For **ROI**, you may try to understand `print_matrix()` to obtain some insight.

Template

```
//PREPEND BEGIN
#include <iostream>
#include <cstdlib>
#include <cstring>
//PREPEND END

struct Data
{
    int *entry;
    size_t row, col;
    size_t ref_cnt;

    Data(size_t row, size_t col):
        row(row), col(col), ref_cnt(0)
    { entry = new int[row * col]{}; }

    ~Data()
    { delete[] entry; }
};

struct Matrix
{
    Data *data;         // the ptr pointing to the entries
    size_t start;       // the starting index of ROI
    size_t row, col;    // the shape of ROI

    Matrix():
        data(nullptr), start(0), row(0), col(0) {}

    ~Matrix()
    {} // something invisible
};

//TEMPLATE BEGIN
void print_matrix(Matrix &mat)
{
    for (size_t r = 0; r < mat.row; r++)
    {
        size_t head = mat.start+r*mat.data->col;
```

```cpp
        for (size_t c = 0; c < mat.col; c++)
            std::cout << mat.data->entry[head + c] << ' ';
        std::cout << '\n';
    }
    std::cout << std::endl;
}

void unload_data(Matrix &mat)
{
    // TODO
    // Noted that `mat.data` could be `nullptr` here
}

void load_data(Matrix &mat, Data *data, size_t start, size_t row, size_t col)
{
    // TODO
}

void shallow_copy(Matrix &dest, Matrix &src)
{
    // TODO
}

void deep_copy(Matrix &dest, Matrix &src)
{
    // TODO
}

bool equal(Matrix &a, Matrix &b)
{
    // TODO
}

void add(Matrix &dest, Matrix &a, Matrix &b)
{
    // TODO
}

void minus(Matrix &dest, Matrix &a, Matrix &b)
{
    // TODO
}

void multiply(Matrix &dest, Matrix &a, Matrix &b)
{
    // TODO
}
//TEMPLATE END

//APPEND BEGIN
int main()
{
    // Sample code on how to use your library
    Data *da = new Data(3, 2), *db = new Data(2, 3);
    for (size_t i = 0; i < 6; i++)
```

```
        da->entry[i] = db->entry[i] = i;

    Matrix a, b, c;
    load_data(a, da, 0, 3, 2);  // the ROI is the whole matrix
    load_data(b, db, 0, 2, 3);
    print_matrix(a);
    /*
        0 1
        2 3
        4 5
    */
    print_matrix(b);
    /*
        0 1 2
        3 4 5
    */

    multiply(c, a, b);
    print_matrix(c);
    /*
        3 4 5
        9 14 19
        15 24 33
    */

    Matrix d, e, f;
    shallow_copy(d, c); // d, c -> (the same) data
    deep_copy(e, c);    // e->data (that have the exactly same content with) c->data
                        // but their addresses are different and ref_cnts are possibly
    load_data(f, c.data, 1, 3, 2);
    print_matrix(f);
    /*
        4 5
        14 19
        24 33
    */
    add(b, a, f);   // notice that the original b.data->ref_cnt becomes 0 and should be deleted
    print_matrix(b);
    /*
        4 6
        16 22
        28 38
    */

    std::cout << a.data->ref_cnt << ' ' << b.data->ref_cnt << ' '
        << c.data->ref_cnt << ' ' << d.data->ref_cnt << ' '
        << e.data->ref_cnt << ' ' << f.data->ref_cnt << std::endl;
    /*
        1 1 3 3 1 3
    */
    return 0;
}
//APPEND END
```

Test Cases

We guarantee that all the parameters are valid, i.e. all the matrices are matched in dimensions.

There are `10` test cases in total testing the following features of your implementation:

- Case 1-4: Basic matrix arithmetic operations
- Case 5-9: Operations with ROI
- Case 10: Memory management (`shallow_copy()` and `ref_cnt` are only checked in this case)