#include <iostream>

/\*template<class T>

class Rows {

public:

Rows(size\_t row)

{

\_row = row;

tab = new T\*[\_row];

}

const T operator[](int index)

{

return this->tab[index];

}

~Rows()

{

delete[] tab;

}

protected:

T\*\* tab;

size\_t \_row;

};\*/

template<class T>

class Table{

public:

// Table(size\_t row, size\_t col) : \_row{row}, \_col{col}

Table(size\_t row, size\_t col)

{

\_row = row;

\_col = col;

//tab = new T\*[\_row];

tab = new Row\* [\_row];

for (size\_t i = 0; i < this->\_row; ++i)

{

//this->tab[i] = new T[\_col];

tab[i] = Row(col);

}

}

/\*T& operator[](int index)

{

return tab[index];

}\*/

const Row\* operator[](int index)

{

return this->tab[index];

}

/\*void operator=(T value)

{

this->tab = value;

}\*/

~Table()

{

for (size\_t i = 0; i < this->\_row; ++i)

{

delete[] this->tab[i];

}

delete[] tab;

}

class Row {

public:

Row(size\_t col)

{

row = new T[col]; // создание массива целочисленных данных

for (int i = 0; i < col; i++)//инициализация полей нулями

{

row[i] = 0;

}

}

T& operator[](int index)

{

return row[index];

}

T\* row;

};

private:

//T\*\* tab; // указатель на двумерный динамич массив

Row\*\* tab;

size\_t \_row;

size\_t \_col;

};

int main()

{

auto test = Table<int>(2, 3);

std::cout << test[0][0];

//test[0][0] = 4;

//std::cout << test[0][0]; // выводит 4

}