3. Определите АТД для хранения информации о таблице базы данных: столбцы (количество и названия столбцов), строки со значениями (все значения в строке таблицы имеют тип string), ключи для получения строк таблицы в некотором порядке (для упрощения ключ включает только один столбец, т.е. ключ - это имя или номер столбца, а строки таблицы можно получать последовательно по одной в порядке возрастания значения в указанном столбце). Перечислите методы АТД, обеспечивающие последовательный доступ к информации и её изменение, аргументы и возвращаемые значения каждого метода с комментариями.

```
#include <iostream>
#include <vector>

using namespace std;

class Table
{
private:
    int columns_count;
    vector<string> columns_name;

public:
    Table();
    ~Table();
    int get_column_count();
    vector<string> get_column_names();
};
```

Получить кол-во строк

Получить кол-во столбцов

Добавить строку (список значений)

Удалить строку (номер строки)

Получить всю строку (номер строки) - (возврат списка значений в строке)

Получить название всех столбцов - (возврат списка имен столбцов по порядку)

Добавить столбец (имя столбца)

Удалить столбец (имя столбца или его номер)

Получить значение у указанной колонки и строки (имя столбца или его номер, номер строки) - возвращает значение в соответсвующем поле

Изменить значение у строки (номер строки, вектор значений)

Изменить значение у указанного столбца и строки

Отсортировать таблицу по указанному столбцу (имя столбца)

8. Используя set из STL напишите решение следующей задачи с эффективностью O(NlogN)$O(N\log N)$.
Дана последовательность из nn различных целых чисел, которые постепенно добавляются в множество. После каждого добавления числа выведите ближайшие числа из множества, меньшее и большее добавленного. Если какого-либо числа не существует, выведите символ '*'.

```
#include <iostream>
#include <set>

using namespace std;

int main()
{
  int n;
  cin >> n;

  set<int> numbers;

  for (int i = 0; i < n; ++i)
```

```
  {
    int num;
    cin >> num;

    auto it = numbers.lower_bound(num);

    if (it != numbers.end())
    {
      cout << *prev(it) << " ";
    }
    else
    {
      cout << "* ";
    }

    if (it != numbers.begin())
    {
      cout << *prev(it) << endl;
    }
    else
    {
      cout << "*\n";
    }

    numbers.insert(num);
  }

  return 0;
}
```

9. Сравните время работы set и unordered_set из STL для операций поиска с количеством элементов N=100,10000,106,107N=100,10000,106,107 (например, измерить время поиска 10 существующих значений в наборе и 10 несуществующих). Ключами являются строки из случайных букв от a до z длиной ровно 16. Результат оформить в виде таблицы, время в ns. Привести код, использованный для измерения времени для одного значения NN.

```
#include <iostream>
#include <algorithm>
#include <unordered_set>
#include <set>
#include <string>
#include <vector>
#include <ctime>
#include <chrono>

using namespace std;

void create(int size, set<string> &str, unordered_set<string> &u_str,
vector<string> &tests)
{
   for (int i = 0; i < size; i++)
   {
      int number = rand() % (int)1e9;
      str.insert(number);
      u_str.insert(number);
      if (tests.size() < 50)
         tests.push_back(number);
      s.clear();
   }
```

```
    for (int i = 0; i < 50; i++)
        tests.push_back(rand());
}

void speed(set<string> &str, unordered_set<string> &u_str, vector<string> &tests)
{
    std::chrono::steady_clock::time_point begin, end;
    long long unset_count = 0, set_count = 0;
    for (int i = 0; i < tests.size(); i++)
    {
        begin = std::chrono::steady_clock::now();
        str.find(tests[i]);
        end = std::chrono::steady_clock::now();
        set_count += chrono::duration_cast<std::chrono::nanoseconds>(end -
begin).count();
    }

    for (int i = 0; i < tests.size(); i++)
    {
        begin = std::chrono::steady_clock::now();
        u_str.find(tests[i]);
        end = std::chrono::steady_clock::now();
        unset_count += chrono::duration_cast<std::chrono::nanoseconds>(end -
begin).count();
    }

    cout << "set: " << set_count / tests.size() << '\n';
    cout << "unordered_set: " << unset_count / tests.size() << '\n';
}
```

```
int main()
{
    srand(time(nullptr));

    set<string> str;
    unordered_set<string> u_str;
    vector<string> tests;
    int size = 100;

    cout << "size: " << size << '\n';

    create(size, str, u_str, tests);
    speed(str, u_str, tests);

    str.clear();
    u_str.clear();
    tests.clear();

    size = (int)1e4;
    cout << "size: " << size << '\n';

    create(size, str, u_str, tests);
    speed(str, u_str, tests);

    str.clear();
    u_str.clear();
    tests.clear();
```

```
    size = (int)1e6;
    cout << "size: " << size << '\n';

    create(size, str, u_str, tests);
    speed(str, u_str, tests);

    str.clear();
    u_str.clear();
    tests.clear();

    size = (int)1e7;
    cout << "size: " << size << '\n';

    create(size, str, u_str, tests);
    speed(str, u_str, tests);

    str.clear();
    u_str.clear();
    tests.clear();

    return 0;
}
```

11,Определить АТД Разреженная матрица, обеспечивающий метод get(i,ji,j) для получения элемента матрицы и set(i,j,vi,j,v) для изменения (добавления) ненулевого элемента. В конструкторе задаются размеры матрицы. Реализовать АТД через список списков vector<list<pair<int,double>>>. Определить эффективность операций + и * в зависимости от количества ненулевых элементов KK.

```
#include <iostream>
```

```
#include <vector>
#include <list>

using namespace std;

class SparseMatrix
{
private:
  int rows, cols;
  vector<list<pair<int, double>>> matrix;

public:
  SparseMatrix(int rows, int cols) : rows(rows), cols(cols)
  {
    matrix.resize(rows);
  }

  double get(int i, int j);
  void set(int i, int j, double v);
};

double SparseMatrix::get(int i, int j)
{
  for (const auto &elem : matrix[i])
```

```
    {
        if (elem.first == j)
        {
            return elem.second;
        }
    }
    return 0;
}

void SparseMatrix::set(int i, int j, double v)
{
    bool found = false;
    for (auto &elem : matrix[i])
    {
        if (elem.first == j)
        {
        elem.second = v;
            found = true;
            break;
        }
    }
    if (!found)
    {
        matrix[i].push_back(make_pair(j, v));
```

```
   }
}
```

```
int main()
{
   return 0;
}
```

```
//Эффективность кода O(cols * rows)
```

12.Определить АТД Матрица, обеспечивающий метод [i,ji,j] для доступа к элементам матрицы. В конструкторе задаются размеры матрицы.Реализовать матрицу через vector размером N·M$N \cdot M$. Определить операцию +. Сравнить время сложения матриц размером 1000×10001000×1000, меняя порядок циклов (строки/столбцы и столбцы/строки) для уровня оптимизации O3. Результаты записать в таблицу, в которой будет указан порядок выполнения циклов и время выполнения в мкс.

```
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

class Matrix
{
    vector<double> v;
    int rows, colums;
```

```
public:
    Matrix(int rows, int colums) : rows(rows), colums(colums)
    {
        v.resize(rows * colums);
    }
    double &operator[](pair<int, int> index)
    {
        auto [i, j] = index;
        if (i < 0 || i > rows || j < 0 || j > colums)
        {
            throw runtime_error("Invalid Index");
        }

        return v[i * rows + j];
    }
    Matrix operator+(const Matrix &other)
    {
        if (rows != other.rows || colums != other.colums)
        {
            throw runtime_error("Invalid Sizes");
        }

        Matrix answer(rows, colums);
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < colums; j++)
           {
                answer[{i, j}] = v[i * rows + j] + other.v[i * rows + j];
            }
```

```
        }
        return answer;
    }
};

int main()
{
    Matrix m(1000, 1000), m2(1000, 1000);
    long long cnt = 0;
    for (int i = 0; i < 100; i++)
    {
        chrono::steady_clock::time_point begin = chrono::steady_clock::now();
        Matrix ans = m + m2;
        chrono::steady_clock::time_point end = chrono::steady_clock::now();
        cnt += chrono::duration_cast<std::chrono::microseconds>(end -
begin).count();
    }
    cout << cnt / 100 << " microseconds\n";
    return 0;
}
```

10. Определить АТД Полином, обеспечивающий метод calc для вычисления значения полинома в точке xx (используйте схему Горнера или барицентрическую форму интерполяционного многочлена Лагранжа). Реализовать полином через представление в виде вектора коэффициентов. В конструкторе задается набор коэффициентов a0,a1,...,an−1$a0,a1,...,an$-1. Определить операции + и *.

```
#include <iostream>
```

```
#include <vector>
using namespace std;
class Polynom
{
private:
    vector<double> coef;

public:

    Polynom(vector<double> coef) : coef(coef) {}

    // Метод для вычисления значения полинома в точке x
    double calc(double x)
    {
        double result = 0;
        for (int i = coef.size() - 1; i >= 0; --i)
       {
            result = result * x + coef[i];
        }
        return result;
    }

    // Оператор сложения двух полиномов
    Polynom operator+(const Polynom& other)
    {
        vector<double> resultCoef(max(coef.size(), other.coef.size()), 0);
        for (size_t i = 0; i < coef.size(); ++i)
       {
            resultCoef[i] += coef[i];
```

```
        }
        for (size_t i = 0; i < other.coef.size(); ++i)
        {
            resultCoef[i] += other.coef[i];
        }
        return Polynom(resultCoef);
    }

    // Оператор умножения двух полиномов
    Polynom operator*(const Polynom& other)
    {
        vector<double> resultCoef(coef.size() + other.coef.size() - 1, 0);
        for (size_t i = 0; i < coef.size(); ++i)
       {
            for (size_t j = 0; j < other.coef.size(); ++j)
          {
                resultCoef[i + j] += coef[i] * other.coef[j];
            }
        }
        return Polynom(resultCoef);
    }
};
```