۱- الف) می خواهیم یک کلاس برای نمایش و کار با آرایه ای از زوج های مرتب حافظه‌دار بنویسیم. هر زوج مرتب شامل یک کلید و یک مقدارفعلی و یک مقدار قبلی است. نوع کلید و مقادیر می توانند متفاوت و از هر نوع دلخواه باشند. مثلا یک آرایه از زوج های مرتب می‌تواند دارای کلید string و مقدارfloat باشد. مثلا آرایه ۵ تایی زیر یک آرایه از نوع مورد نظر است که نمره‌های نهایی(با ارفاق ;)) و نمره ابتدایی دانشجویان را مشخص می‌کند.

Sara	۲۰	۱۹.۲
Ali	۱۸.۵	۱۸
Arman	۱۰	۹.۳
Shadi	۱۱	۱۰.۵

دقت کنید که این فقط یک مثال است. عناصر هر زوج مرتب می تواند از هر نوع باشد اما برای همه عناصر آرایه از یک نوع است. یعنی کلیدها همه از یک نوع و مقادیر همه از یک نوع هستند. دقت کنید که دسترسی مستقیم به این ساختمان داده از بیرون کلاس امکان پذیر نیست.

کلاسی را برای این ساختمان داده پیاده سازی کنید. این کلاس غیر از ساختمان داده مورد نظر دارای یک سازنده و دو تابع عضو public به شرح زیر است:

- سازنده یک عدد صحیح به عنوان طول می گیرد و به اندازه آن آرایه از زوج های مرتب را allocate می کند.
- تابع insert یک کلید و یک مقدار دریافت می‌کند و به انتهای آرایه زوج های مرتب اضافه می کند. در این حالت مقدار قبلی کلید مربوطه، یک مقدار نامعلوم است.

تابع update یک کلید و یک مقدار دریافت می‌کند و با جستجوی کلید داده‌شده، مقدار زوج مرتب مربوطه را تغییر می‌دهد و مقدار قبلی این کلید را نیز ذخیره می‌کند. در صورتی که کلید داده‌شده قبلاً اضافه نشده باشد، این تابع یک exception با پیام unknown key تولید می‌کند. آبجکت exception موردنظر غیر از پیام دارای رشته ای است که کلیدی که وجود نداشته است و آپدیت آن باعث اکسپشن شده است را نگه داری میکند و در یک متد get باز میگرداند. بنابراین در این قسمت لازم است یک کلاس exception جدید بنویسید که از یکی از کلاسهای exception موجود در c++ ارث بری داشته باشد.

در پیاده‌سازی این سؤال می‌توانید از STL استفاده کنید یا از ساختمان داده‌های پایه مثل آرایه استفاده کنید.

ب) یک تابع main بنویسید که یک شئ از نوع کلاس تعریف شده بسازد (با استفاده از سازنده کلاس). این شئ دارای زوج مرتب هایی از نوع (string, float) با طول آرایه ۵ باشد. سپس در یک حلقه مقادیر ۵ زوج مرتب را از ورودی دریافت کرده و به شئ ساخته شده اضافه کند.

۲- فرض کنید یک سرور داریم که با یک کلاینت در ارتباط است. این سرور یک ساختمان داده از شماره تلفن افراد در حافظه دارد. این شماره تلفنها از طریق کلاینت وارد یا حذف میشود. بدین صورت که سرور منتظر اتصال کلاینت است و پس از اتصال هر کلاینت، منتظر دریافت پیام از کلاینت میماند. این پیامها یا برای درخواست اضافه کردن یک شماره تلفن و یا برای حذف یک شماره تلفن هستند.

پیام کلاینت به سرور شامل ۳۴ یا ۲۳ بایت، مطابق با شکل زیر است:

"ADD" (3 bytes)	TellNo (11 bytes)	Name (20 bytes)

"REM" (3 bytes)	Name (20 bytes)

در صورتی که نوع پیام ADD باشد، سرور پس از اعتبارسنجی فرد موردنظر با شماره داده‌شده (با استفاده از فراخوانی تابع آماده Authenticate(name, telNo)، شماره تلفن و نام را در حافظه خود ذخیره‌سازی می‌کند و برعکس اگر نوع پیام REM باشد سرور باید شماره فرد موردنظر را از حافظه خود حذف کند.

سرور در یک thread پیامها را پشت سرهم دریافت میکند و بدون رسیدگی به آنها، آن‌ها را در یک آرایه از کلاس message_handler ذخیره‌سازی می‌کند (فرض کنید نرخ دریافت پیام بالاست و رسیدگی به پیامها زمان بر است. پس برای اطمینان از اینکه بافر دریافت پیام سرور پر نمیشود و هیچ پیامی را از دست نمیدهد سریعاً آن‌ها را ذخیره سازی میکند تا در thread دیگری به پردازش آن‌ها بپردازد)

هر کلاس message_handler دارای یک متد handle_message است. از طرفی دو نوع کلاس message_handler داریم که یکی برای پیامهای ADD و دیگری برای پیامهایREM به کار می‌رود و هر یک داده‌های موردنیاز خود را دارند. تابع handle_message نیز در هر یک از دو نوع کلاس، وظیفه اضافه کردن یا حذف شماره را به عهده دارد. ضمناً در صورتی که شماره موردنظر برای اضافه کردن تأیید نشود یک exception از نوع bad_type_id با پیام "authentication failed" و در صورتی که نام موردنظر برای حذف قبلاً در حافظه ذخیره نشده باشد یک exception از نوع جدید not_found_id با پیام "there is no contact with this name" برگردانده می‌شود.

سرور در thread پردازش پیامها، آرایه message_handlerها را دریافت کرده و با فراخوانی handle_message به صورت متوالی پیامها را پردازش می‌کند.دقت کنید که به نحو مناسبی exception ها را هندل کنید.

(threads, Polymorphsim, STL, network, exception)

جهت برنامه نویسی سرور و کلاینت از کلاسهای سرور و کلاینت که در ادامه آمده است استفاده کنید.

نمره اضافه: برنامه را برای بیش از یک کلاینت بنویسید. یعنی سرور پیامهای اضافه و حذف شماره تلفنها را از بیش از یک کلاینت به صورت همزمان دریافت میکند.

```cpp
#include <QTcpServer>
#include <QTcpSocket>
class Client
{
    QTcpSocket socket;
public:
    Client(char* ServerAddr, int port){
        qDebug()<< "Connecting to server...\n";
        socket.connectToHost(serverAddr, port);
        if (socket.waitForConnected(1000))
            qDebug("Connected!");
        else
            throw runtime_error("Can not connect to the server");
    }
    void send(QByteArray){
        if(socket.state() == QAbstractSocket::ConnectedState){
            socket.write(data);
            socket.waitForBytesWritten();
        }
        else throw runtime_error("Disconnected from server");
    }
    QByteArray recieve(){
        if(socket.state() == QAbstractSocket::ConnectedState){
            if(socket.waitForReadyRead(-1))
                return socket.readAll();
        }
        else throw runtime_error("Disconnected from server");
    }
    ~Client(){
        socket.close();
    }
};
```

```
class Server
{
protected:
    QTcpServer tcpserver;
    QTcpSocket *client;
public:
    Server(char* IPaddr, int port){
        if(!tcpserver.listen(QHostAddress(QString(IPaddr)),port))
            throw runtime_error{"can not listen on this IP and port "};
    }
    void wait_for_connection(){
        tcpserver.waitForNewConnection(-1);
        client = tcpserver.nextPendingConnection();
        if(client!=NULL){
            qDebug() << "New connection from: " << client->peerAddress()<<":"<<client->peer

        }
    }
    void send(QByteArray buff){
        if(client->state() == QAbstractSocket::ConnectedState){
            client->write(data);
            client->waitForBytesWritten();
        }
        else
            throw runtime_error("Client is disconnected");
    }
    QByteArray recieve(){
        if(client->state() == QAbstractSocket::ConnectedState){
            if(client->waitForReadyRead(-1))
                return client->readAll();
        }
        else
            throw runtime_error("Client is disconnected");
    }
    ~Server(){
        tcpserver.close();
        client->close();
    }
};
```

راهنما

<fstream>
std::ofstream: Stream class to write on files
std::ifstream: Stream class to read from files
std::fstream: Stream class to both read and write from/to files.
open (filename, mode): Open a file with possible modes:

- ios::in, Open for input operations.
- ios::out, Open for output operations.
- ios::binary, Open in binary mode.
- ios::ate, Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
- ios::app, All output operations are performed at the end of the file, appending the content to the current content of the file.
- ios::trunc, If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

is_open(): Check if a file is open
close(): Close file
operator>> Extract formatted input (istream and input mode)
operator<< InsertInsert formatted output (ostream and output mode)
bad(): Returns true if a reading or writing operation fails.
Fail(): Returns true in the same cases as bad(), but also in the case that a format error happens
Eof(): Returns true if a file open for reading has reached the end.
Good(): It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true.

<cstdio>
rename (const char * oldname, const char * newname): Changes the name of the file or directory specified by *oldname* to *newname*.

<thread>
thread() : default constructor, Construct a thread object that does not represent any thread of execution.
thread (Fn&& fn, Args&&... args): initialization constructor, Construct a thread object that represents a new *joinable* thread of execution. The new thread of execution calls fn passing args as arguments
join(): Join thread

<exception>
std::exception: exception class
std::bad_type_id: bad_type_id class
std::logic_error: logic_error exception class
exception::what(): returns a null-terminated character sequence (of type char *) and that can be overwritten in derived classes to contain some sort of description of the exception.
throw(exception e) – throwing the exception e
catch(exception e) – catching an exception with an exception handler e
try – A **try** block identifies a block of code for which particular exceptions will be activated.

STL

std::vector<Typename>: default constructor, construct an empty vector
std::map<key_Typename,Val_Typename>: default constructor, construct an empty map
vector::operator[], map::::operator[]: access specified element
vector::begin(), **map::begin**(): returns an iterator to the beginning
vector::end(), **map::end**(): returns an iterator to the end
vector::size(), map::size(): returns the number of elements
vector::push_back() : adds an element to the end
vector::pop_back(): removes the last element
map::insert(std::pair<key_Typename,Val_Typename>(key, val)): Insert a pair
map:: count(key): find number of elements with a specific key
std::vector<Typename>::iterator: an iterator for a vector
std::map<key_Typename,Val_Typename>::iterator: an iterator for a map