تمرين (2) _ برنامه نويسى پيشرفته

طراحي سيستم بانكي

مدرس

مهندس بطحاييان

دانشجو

مهرناز جيريايي _ 9812358013

پاييز 1400

شرح کلی برنامه:

در یک سیستم بانکی، چون با پول سروکار داریم همه چیز باید خیلی دقیق و منظم باشد. شما در این تمرین یک سیستم بانکی ساده را تولید میکنید که حسابهای مشتریان را کنترل و ویرایش میکند، درصورت امکان وام میدهد و به مشتری ها شود بانکی میپردازد، و یا درصورت بروز فاجعه، اعلام ورشکستگی میکند. این سیستم ماند خط فرمان عمل میکند و دستورهای ما را پردازش میکند. در ادامه دستورها به اختصار معرفی میشوند.

شرح فایل Bank.h :

این فایل شامل یک کلاس بنام customer است که دارای دیتا ممبر های پرایوت در تصویر زیر است.

```
private:
    std::string username;          // account username
    int card_number;               // a four digit number
    long int opening_date;         // not sure about its type
    long int expiration_date;      // not sure about its type
    unsigned long int balance;     // account balance
    std::string ip;                // first ip of account
    std::vector<std::string> ips; // account ips
    // class transaction;           // account transaction
```

همچنین شامل متد ها و توابع عضو در تصویر زیر است :

```
public:
    customer(std::string, std::string);
    ~customer();
    class transaction; // account transaction
    std::string get_username();
    void set_ip(std::string);
    void ip_validation(std::string);
    void set_opening_date(int);
    void check_expiration_date_for_renewal();
    void set_expiration_date(int);
    unsigned long int get_balance();
    void set_balance(unsigned long int);
    int get_expiration_date();
    int get_opening_date();
    void check_expiration_date_for_transaction();
    bool get_ips(std::string);
```

این فایل دارای توابعی به صورت عمومی نیز میباشد:

```
unsigned int convert(std::string &);
unsigned int convert(char &);
void add_ip(std::string, std::vector<customer> &, std::string);
void renewal(std::vector<customer> &, size_t &);
customer check_existance_account(std::vector<customer> &, std::string);
void withdraw(customer &, unsigned int &);
```

شرح فایل Transaction.h :

این فایل شامل یک کلاس بنام Transaction میباشد که دارای دیتا ممبرهای پرایوت زیر است :

```
private:
    customer beginning;
    customer destination;
    unsigned int payment;
    int date;
```

و توابع عضو زیر :

```
public:
    transaction(customer, customer, unsigned int);
    ~transaction();
```

شرح فایل Bank.cpp :

این فایل شامل چندین تابع است که به شرح انها میپردازیم:

شرح تابع convert :

این تابع به صورت سربارگذاری (function overloading) پیاده سازی شده که یکی آرگومان عدد از نوع رشته گرفته و از طریق تابع stoi که عضو کتابخانه string است به نوع عددی تبدیل میکند و دیگری ارگومان عدد کاراکتری گرفته و به نوع عددی تبدیل میکند (از این طریق که کاراکتر را منهای عدد 48 کرده و نوع عددی ان کاراکتر عددی به دست می اید) و انتخاب هر کدام ازین توابع بر حسب نوع ورودی است که رشته باشد یا کاراکتر!

```
unsigned int convert(string &str)
{
    int number;
    number = stoi(str);
    return number;
}

unsigned int convert(char &ch)
{
    int number;
    number = ch - 48;
    return number;
}
```

شرح کانستراکتور customer :

```
customer::customer(string UN, string IP) // constructor with two parameter
    : username(UN), ip(IP)
{
    // if (!(is_Empty))
    // {
    //     for (size_t i = 0; i < usernames.size(); i++)
    //     {
    //         if (UN == usernames[i])
    //         {
    //             throw("Repeated username!! - try another username.\n");
    //         }
    //     }
    // }
    // is_Empty = false;
    if (!(isalpha(UN[0])))
    {
        throw 505; // an error
    }
    char c;
    for (size_t i = 1; i < UN.size(); i++) // i = 0 was checked recently so it starts from 1
    {
        if (!((UN[i] >= 48 && UN[i] <= 57) ||
              (isalpha(UN[i]))))
        {
            throw(c); // an error
        }
    }
    ip_validation(IP);
    ips.push_back(IP);

    default_random_engine eng(static_cast<unsigned int>(time(0)));
    uniform_int_distribution<unsigned int> myrand(1000, 9999);
    card_number = myrand(eng);
    std::cout << "Your card number is : " << card_number << endl;
    balance = 25000;
}
```

این کانستراکتور دو ارگومان از نوع رشته میگیرد یکی UN و دیگری IP .

UN به username انتساب میشود و IP هم به ip .

در بدنه این کانستراکتور بخشی از کد کامنت شده که جهت چک کردن عدم تکراری بودنِ یوزر نیم وارد شده توسط کاربر بوده است اما بدلیل رخ دادن برخی ارور ها کامنت شد. عملکرد این بخش به این صورت است

که ایتدا از طریق متغیر is_Empty چک میکند این خانه از عنصر خالی است یا پر شده؟ سپس اگر پر بود وارد حلقه for میشد که دراین حلقه هم همه یوزرنیم های از قبل ذخیره شده را با یوزرنیم وارد شده مقایسه کرده و در صورت تکراری بودن یک استثنا به صورت یک پیغام پرتاب میکند و در اخر ان خانه مقدار false را به متغیر is_Empty انتساب میدهد.

مرحله بعدی حرف اول یوزر نیم را از طریق تابع isalpha که عضو کتابخانه ctype.h است اعتبار سنجی میکند که حتما حرف انگلیسی باشد نه کاراکتر دیگری . سپس در مرحله بعدی با حلقه for بقیه حروف یوزرنیم را بررسی میکند که بجز عدد و حروف انگلیسی کاراکتر دیگری نباشند. در غیر این صورت یک استثنا بصورت ارور پرتاب میشود .

UN[i] >= 48 && UN[i] <= 57

در خط بالا نشان میدهد که کاراکتر ها باید بین مقدار 48 تا 57 باشند که کد اسکی اعداد 0 تا 9 میباشند.

سپس در مرحله بعد مقدار ای پی به تابع ip_validation پاس داده میشود تا درستی ان بررسی شود در ادامه این تابع شرح داده میشود.

در صورت صحت ای پی، این ایپی در وکتور (ips) ای پی های حساب بانک ان مشتری ذخیره میشود.

در مرحله بعد از یک روش تولید اعداد تصادفی
(default_random_engine) استفاده کرده ایم تا برای هر حساب یک

کد چهار رقمی تولید کندو در متغیر card_number قرار میدهد.

همچنین برای هر حسابی که ایجاد میشود بصورت پیش فرض مبلغ 25000 تومان در نظر گرفته میشود.

شرح تابع get_username :

این تابع یوزر نیم حساب کاربر را که از نوع رشته است برمیگرداند.

```
string customer::get_username()
{
    return username;
}
```

شرح تابع set_ip :

این تابع ای پی جدید را در وکتور ایپی ها حساب ذخیره میکند. و سپس همه ایپی های متعلق به ان حساب را نمایش میدهد.

```
void customer::set_ip(string IP)
{
    ips.push_back(IP);
    for (size_t i = 0; i < ips.size(); i++)
    {
        std::cout << ips[i] << "\t";
    }
}
```

شرح تابع set_opening_date :

```
void customer::set_opening_date(int open_date)
{
    opening_date = open_date;
}
```

این تابع سال افتتاح این حساب را از ورودی میگیرد (در متن تمرین بیان شده بود که در کانستراکتور با تایم سیستم ست شود که به دلیل عدم استفاده از مبحث فایل در تمرین و سرعت بخشیدن به اجرا و بررسی برنامه تصمیم بر این شد که سال افتتاح و انقضا از ورودی گرفته شود.)

شرح تابع set_expiration_date :

```
void customer::set_expiration_date(int exp)
{
    expiration_date = exp;
}
```

این تابع سال انقضای این حساب را از ورودی میگیرد.(در متن تمرین بیان شده بود که در کانستراکتور با تایم سیستم ست شود که به دلیل عدم استفاده از مبحث فایل در تمرین و سرعت بخشیدن به اجرا و بررسی برنامه تصمیم بر این شد که سال افتتاح و انقضا از ورودی گرفته شود.)

شرح تابع get_expiration_date و get_opening_date:

```
int customer::get_expiration_date()
{
    return expiration_date;
}

int customer::get_opening_date()
{
    return opening_date;
}
```

این دو تابع عملکرد مشابهی دارن با این تفاوت که اولی مقدار سال افتتاح حساب را برمیگرداند و دومی مقدار سال انقضای حساب را بر میگرداند.

شرح تابع check_expiration_date_for_renewal:

```
void customer::check_expiration_date_for_renewal()
{
    // std::cout << expiration_date - opening_date << endl;
    // std::cout << get_expiration_date() - get_opening_date() << endl;
    if ((get_expiration_date() - get_opening_date()) <= 2 && (get_expiration_date() - get_opening_date()) >= 0)
    {
        throw runtime_error("Your account has not expired yet!!\n");
    }
}
```

این تابع تاریخ انقضای حساب را برای زمانی که مشتری درخواست تمدید میدهد را بررسی میکند که در صورتی که هنوز تاریخ نگذشته باشد یک استثنا به شکل پیغام چاپ میکند که هنوز حساب تاریخش نگذشته .

شرح تابع

check_expiration_date_for_transaction :

```
void customer::check_expiration_date_for_transaction()
{
    if (!((get_expiration_date() - get_opening_date()) > 2))
    {
        throw runtime_error("Your account has not expired!!\n");
    }
}
```

این تابع تاریخ انقضای حساب را برای زمانیکه کاربر میخواهد تراکنش مانند واریز یا انتقال انجام دهد را بررسی میکند که در صورتی که تاریخ گذشته باشد به مشتری پیغام میدهد.

شرح تابع get_balance :

این تابع مقدار موجودی حساب را برمیگرداند.

شرح تابع set_balance :

این تابع مقدار موجودی حساب را دریافت میکند.

```
unsigned long int customer::get_balance()
{
    return balance;
}

void customer::set_balance(unsigned long int blnc)
{
    balance = blnc;
}
```

شرح تابع ip_validation :

این تابع ایپی را بصورت استرینگ گرفته و انرا از طریق شناسایی کاراکتر نقطه و حلقه while به 4 بخش تقسیم میگند و در متغیر های جداگانه که از نوع رسته هستند میریزد (IP1/IP2/IP3/IP4)

```
void customer::ip_validation(string IP)
{
    size_t i = 0;

    string IP1;
    string IP2;
    string IP3;
    string IP4;

    int dot = 0; // counts the number of dots in ip

    while (IP[i] != '.' && IP[i])
    {
        IP1 += IP[i];
        i++;
    }
    if (IP[i] == '.')
    {
        dot++;
    }
    i++;

    while (IP[i] != '.' && IP[i])
    {
        IP2 += IP[i];
        i++;
    }
```

سپس در انتهای تقسیم بندی در برنامه چک میکند که اگر تعداد نقاط موجود در ایپی غیر از سه تا باشد یک استثنا پرتاب کند که نشانگر عدم صحیح بودن فرمت ایپی است. سپس هر کدام از چهار قسمت ایپی به تابع convert پاس داده میشود تا تبدیل به نوع عددی شوند تا بهتر بتوان عمل اعتبار سنجی ایپی را انجام دادو در مرحله بعد هر قسمت از ایپی بررسی میشود که بین 0 تا 255 باشد و در غیر این صورت استثنا پرتاب میشود تا به کاربر نشان داده شود ایپی صحیح نیست.

```
int part1 = convert(IP1);
int part2 = convert(IP2);
int part3 = convert(IP3);
int part4 = convert(IP4);

float b;
if (!(part1 >= 0 && part1 <= 255) || !(part2 >= 0 && part2 <= 255) || !(part3 >= 0 && part3 <= 255) || !(part4 >= 0 && part4 <= 255))
{
    throw(b);
}
```

شرح تابع add_ip :

```
void add_ip(string username, vector<customer> &moshtari, string ip)
{
    for (size_t i = 0; i < moshtari.size(); i++)
    {
        if (username == moshtari[i].get_username())
        {
            try
            {
                moshtari[i].ip_validation(ip);
                moshtari[i].set_ip(ip);
            }

            catch (float f)
            {
                std::cout << "Invalid ip - each part of ip must be between 0 and 255.\n";
            }

            catch (double d)
            {
                std::cout << "Invalid ip - ip must include for parts and exactly 3 dots.\n";
            }
        }
    }
```

این تابع بررسی میکند که اگر یوزر نیم وارد شده در یوزر نیم های ذخیره شده در بانک باشد یک ایپی جدید به لیست ایپی های ان اضافه کند. حساب های ذخیره شده در بانک در یک وکتور از نوع customer و بنام moshtari ذخیره شده اند. سپس که حساب کاربر پیدا شد ابتدا اعتبار سنجی ایپی انجام میشود و در صورت صحت در لیست ایپی های حساب ذخیره میشود. این بخش از کد در یک بلاک try _catch انجام شد که در هر مرحله از اعتبار سنجی مورد دور از انتظاری رخ داد استثنایی پرتاب شود.

شرح تابع get_ip :

```
bool customer::get_ips(string ip) // this function checks if an ip belong to an account or not
{
    for (size_t i = 0; i < ips.size(); i++)
    {
        if (ip == ips[i])
        {
            std::cout << "true\n";
            return true;
        }
    }
}
```

این تابع بررسی میکند که ایا ایپی وارد شده به حساب مورد نظر تعلق دارد یا نه به این روش که ایپی وارد شده ر با تمام ایپی های موجود در لیست ایپی های حساب مقایسه میکند و در صورت وجود مقدار true را برمیگرداند.

شرح تابع renewal :

```
void renewal(vector<customer> &moshtari, size_t &i)
{
    char ch;
    std::cout << "Your expiration time of your account has been finished!! Do you wanna renewal your account? (For renewal you have to pa
    cin >> ch;
    if (ch == 'y' || ch == 'Y')
    {
        // here does a validation for when balance is zero and have to get loan
        moshtari[i].set_balance(moshtari[i].get_balance() - 5000);              // here decrease the renewal cost from account balance
        moshtari[i].set_expiration_date(moshtari[i].get_expiration_date() + 2); // change // this updates the expiration account
        std::cout << moshtari[i].get_expiration_date() << endl;
        std::cout << "congrats" << endl;
    }
    else if (ch == 'n' || ch == 'N')
    {
        return;
    }
}
```

این تابع زمانی که انقضای حساب به پایان رسیده باشد و مشتری درخواست تمدید داده باشد یا در حین انجام تراکنش مانند واریز یا انتقال برنامه متوجه انقضای تاریخ حساب شده باشد کار میکند. به این شکل که در ابتدا از مشتری پرسیده میشود که تاریخ انقضا به پایان رسیده و ایا مطمئنا ست که میخواهد حساب را تمدید کند؟ و کاربر با وارد کردن حرف y یا n بصورت بزرگ یا کوچک پاسخ خود را نشان میدهد. درصورتی که کاربر y را وارد کرده باشد مبلغ 5000 تومان از حساب کاربر کم شده (هزینه تمدید) و مبلغ جدید در حساب قرار میگیرد (از طریق توابع set_balance و get_balance) و تاریخ انقضای جدید برای کاربر به همراه یک پیغام تبریک نمایش داده میشود. در صورتی کاربر n وارد کرده باشد برنامه از تابع خارج میشود.

شرح تابع check_existance_account :

```
customer check_existance_account(vector<customer> &moshtari, string s)
{
    for (size_t i = 0; i < moshtari.size(); i++)
    {
        if (moshtari[i].get_username() == s)
        {
            return moshtari[i];
            // throw invalid_argument("The destination account does not exist with these username or ip!\n");
        }
        else
        {
            cout << "fuch\n";
            return moshtari[i];
        }
    }
}
```

این تابع وجود و عدم وجود حساب را از طریق یوزرنیم بررسی میکند که در صورت وجود ان حساب را برمیگرداند.

شرح تابع withdraw :

```
void withdraw(customer &moshtari, unsigned int &s)
{
    moshtari.set_balance(moshtari.get_balance() - s);
}
```

این تابع مبلغی از حساب کاربر برداشت میکند.

شرح فایل Transaction.cpp :

در این فایل شرح کلاس تراکنش یا Transaction نوشته شده است .

```cpp
#include <iostream>
#include "Transaction.h"
using namespace std;

transaction::transaction(customer begin_acc, customer destin_acc, unsigned int transfer_money)
    : beginning(begin_acc), destination(destin_acc), payment(transfer_money)
{

    begin_acc.set_balance(begin_acc.get_balance() - transfer_money);
    destin_acc.set_balance(destin_acc.get_balance() + transfer_money);
    cout << begin_acc.get_balance() << endl;
    cout << destin_acc.get_balance() << endl;
}

transaction::~transaction()
{
}
```

کانستراکتور این کلاس دو ارگومان از نوع customer و یک ارگومان از نوع عدد صحیح بی علامت میگیرد. همانطور که میدانیم این کلاس عمل انتقال را از حساب مبدا به حساب مقصد انتقال میدهد که مبلغ را (transfer_money) از حساب مبدا (از طریق تابع get_balance) کم کرده و از طریق تابع set_balance موجودی جدید را ثبت میکند و از طریق همین توابع اما برای حساب مقصد این مبلغ را اضافه و ذخیره میکند.

شرح تابع main :

```
cout << "WELCOME TO GHOZAH BANK\n"; //بانک قُـزَه
vector<customer> moshtari;          // a vector that save the accounts
vector<transaction> taraconesh;     // a vector that save the transactions
// std::vector<customer>::iterator it;
```

در ابتدای این تابع یک پیغام خوشامدگویی چاپ میشود سپس دو وکتور یکی از نوع customerو دیگری از نوع transaction، اولی برای ذخیره حساب ها و دومی برای ذخیره تراکنش ها ساخته شده است.

```
string str = "";
string command = "";  // customer request
string username = ""; // account username
string ip = "";       // account ip

cout << "Enter your request:\n";

getline(cin, str);
```

سپش چند رشته بنام های command برای ذخیره دستور ، رشته str برای کل رشته ورودی، username برای ذخیره یوزرنیم، ip بریا ذخیره ایپی، که به هر کدام مقدار تهی داده میشود تا مقدار اضافه ای در انها ذخیره نشود.

سپس یک پیغام برای وارد کردن رشته نمایش داده میشود. کل رشته ورودی از طریق تابع getline دریافت میشود رشته str از طریق حلقه های

while به قسمت های مختلف مانند command, username, ip و .... میشود. رشته هم برای مواقعیست که بقیه قسمت های رشته str را ذخیره کند.

```
size_t i = 0;

while (str[i] != ' ' && str[i])
{
    command += str[i]; // create command word
    i++;
}

i++;

while (str[i] != ':' && str[i])
{
    username += str[i]; // create username word
    i++;
}

i++;

while (str[i] != ':' && str[i])
{
    ip += str[i]; // create ip
    i++;
}
```

رشته payment هم برا موقعیست که کاربر میخواهد مبلغی را انتقال دهد یا واریز کند که مقدار مبلغ بصورت رشته در ان ذخیره میشود.

```
string s = "";
if (str[i])
{
    i++;
    while (str[i] != ':' && str[i])
    {
        s += str[i];
        i++;
    }
    cout << "s :" << s << endl;
}

unsigned int transfer_money = 0;
string payment = "";

if (str[i])
{
    i++;
    while (str[i])
    {
        payment += str[i];
        i++;
    }
    cout << "payment : " << payment << endl;
    transfer_money = convert(payment);
    cout << "ts : " << transfer_money << endl;
}
```

در مرحله بعدی در صورتیکه دستور وارد شده برابر با create بود وارد بدنه شرط if میشویم و یک حساب بنام account از نوع customer با ارگومان های username و ip ساخته میشود. سپس از کاربر سال افتتاح و انقضای حساب را دریافت میکند . بعد از ان که حساب ساخته شد ان را در لیست (وکتور) حساب های بانگ ذخیره میکند و یک پیغام که حساب با

موفقیت افتتاح شده است نمایش داده میشود. در انتها برای حذف کاراکتر اضافی که بصورت ناخواسته در بافر ذخیره شده از تابع cin.ignore استفاده کردیم.

همچینین این بخش از کد در یک بلاک try_catch انجام شد که در هر مرحله از اعتبار سنجی یوزرنیم و یا ایپی مورد دور از انتظاری رخ داد استثنایی پرتاب شود و یک پیغام خطا نمایش داده شود.

اگر command برابر با کلمه add_ip بود برنامه وارد تابع add_ip میشود که عملکرد ان را قبلا توضیح داده ایم. سپس در صورتیکه رسته S برابر با کلمه another بود کاربر همزمان میتواند دوتا ایپی برای یک حساب بصورت دستی بسازد.

اگر command برابر با کلمه renewal بود برنامه یوزرنیم وارد شده با یوزرنیم حساب های قبلی که ساخته شده اند را مقایسه کرده تا ان را پیدا کند و در صورت یافتن تابع check_expiration_date_for_renewal فراخوانی میشود که عملکرد ان را قبلا توضیح داده ایم که در صورت عدم اتمام انقضای حساب پیغامی چاپ میشود.

در صورتی که انقضای حساب تمام شده بود برنامه وارد تابع renewal شده و حساب را با تایید مشتری تمدید میکند.

اگر command برابر با کلمه deposit بود عملکرد مانند دستور renewal است با این تفاوت که پس از تمدید یا عدم تمدید مشتری مبلغی را به حساب واریز میکند.

اگر command برابر با کلمه transfer بود بود عملکرد مانند دستور renewal است اما پس از تمدید یا عدم تمدید برنامه از طریق تابع check_existance_account بررسی میکند که ایا حساب مقصد موجو میباشد یا نه در صورت وجود انتقال مبلغ از حساب مبدا به حساب مقصدر از طریق transaction transition صورت میگیرد و این تراکنش در وکتور Taraconesh ذخیره میشود . در صورت عدم وجود حساب مقصد پیغام خطا برای مشتری چاپ میشود.

اگر command برابر با کلمه withdraw بود بود عملکرد مانند دستور renewal است اما پس از تمدید یا عدم تمدید برنامه وارد تابع withdraw مشتری میتواند مبلغی از حساب خود برداشت کند.