BAHRIA UNIVERSITY KARACHI CAMPUS

DEPARTMENT OF COMPUTER ENGINEERING

Trading Software

Data Structures & Algorithms

CSL 221

Muneeza Iftikhar – 012
Hafsa Hafeez Siddiqui – 026

Haris Aamir - 034

Acknowledgements

We are utmost grateful for the opportunity that was provided to us and it wouldn't have been possible without our course teacher Dr.Samabia Tehsin and our lab engineer Rohma Najam. Both of them helped clarify our concepts throughout the course. Through their guidance we were able to execute our project to the best of our ability. Each group member participated and performed its duty in the right way. This project is the outcome of each group member and its instructors. The project has been assigned to each group member to play its role in accordance to their understanding and feasibility.

Abstract

Trading software is implemented on C++ using VSCode and Code Blocks. It's tries to simulate a trading platform using cryptocurrencies particularly bitcoin. The following project is implemented on the concepts of data structures.

Contents

List of Figures

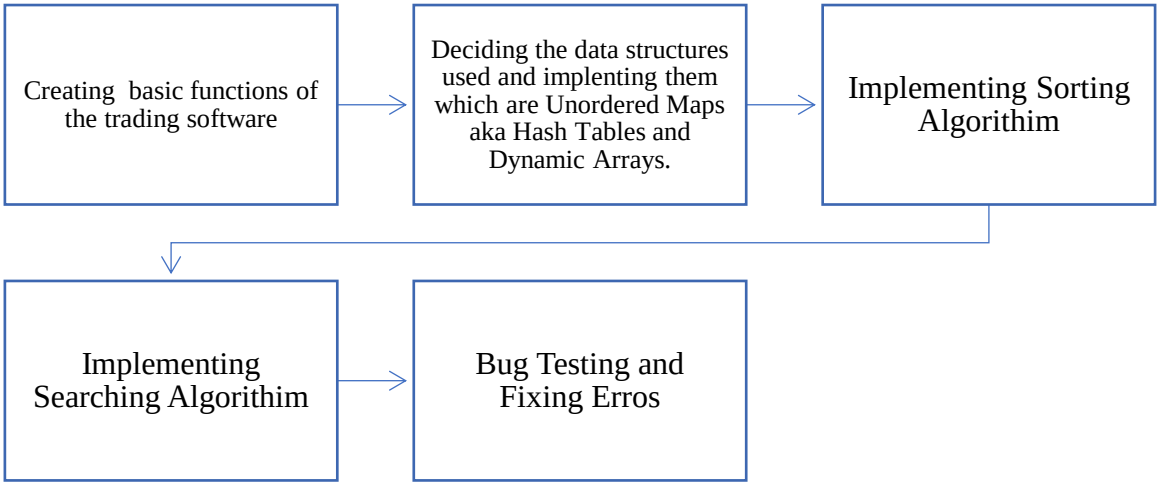
Introduction

Trading Software simulates a trading system with functionality to create an account, deposit and withdraw money, buy and sell bitcoins, and display account information. The program uses a map to store the accounts, where the key is the username and the value is the Account struct, which stores information such as the balance, transaction history, and bitcoin holdings. The program also has functionality to sort the accounts based on their bitcoin holdings and search for a specific account by username.

Problem and solution

The task at hand was to figure out what kind of data structures to implement in this scenario. One of data structure used to store the username is unordered maps. It has a built-in library in c++ and unsorted maps are hash tables with closed addressing. unordered_map is used to store the account information for each user. Unordered map is an associative container that contains key-value pairs with unique keys. Search, insertion, and removal of elements have average constant-time complexity. Another data structure we have used is vectors. a vector is a container class that is part of the Standard Template Library (STL). It is a dynamic array, meaning that its size can change during runtime. A vector is similar to an array, but with the added ability to dynamically grow and shrink in size, making it more flexible. a vector is used to temporarily store the accounts in order to sort them based on bitcoin holdings by storing the pair of username and Account struct.

Workflow



Overview of project

Concepts used:

- Unordered Maps aka Hash Tables. It uses unordered map to store the Account struct, which holds information about the account's balance, transaction history, and bitcoin holdings. The program also has a global variable bitcoin_price that is used to calculate the cost of buying or selling bitcoin. The program has a menu-driven interface that allows users to perform these actions through a series of user inputs.
- Vectors (Dynamic Arrays) is used to temporarily store the accounts in order to sort them based on bitcoin holdings by storing the pair of username and Account struct.
- Comparison Sorting which utilizes a hybrid sorting which includes combination of heap sort, insertion sort and quick sort. A new function called "sortAccounts" that takes a sorting criterion as a parameter (e.g. by bitcoin holdings).
- Linear search is a method of searching through a collection of data by iterating through each element one by one until the desired item is found. In this case, the searchAccount function iterates through each key-value pair in the accounts map and compares the inputted username with the keys of the map. When a match is found, the function calls the displayAccount function to display the account information.

Source Code:

```
#include <iostream>
#include <map>
#include <vector>
#include <algorithm>
using namespace std;

struct Account
{
    double balance;
    std::string transactionHistory;
    double bitcoin_holdings;
};

std::map<std::string, Account> accounts;
double bitcoin_price = 5000;
void createAccount(std::string username)
{
    if (accounts.count(username) == 0)
    {
        Account newAccount;
        newAccount.balance = 0;
        newAccount.transactionHistory = "";
        newAccount.bitcoin_holdings = 0;
        accounts[username] = newAccount;
        cout << "A new account was created successfully!" << endl;
    }
}
void deposit(std::string username)
{
    double amount;
    cout << "Enter amount to deposit: " << endl;
    cin >> amount;
    if (accounts.count(username) != 0)
    {
    }
    accounts[username].balance += amount;
    accounts[username].transactionHistory += "Deposited " + std::to_string(amount)
+ "\n";
    cout << "Deposited Successfully" << endl;
}

void withdraw(std::string username)
{
    double amount;
```

```
    cout << "Enter amount to withdraw: " << endl;
    cin >> amount;
    if (accounts.count(username) != 0 && accounts[username].balance >= amount)
    {
        accounts[username].balance -= amount;
                 accounts[username].transactionHistory   +=   "Withdrew   "   +
std::to_string(amount) + "\n";
        cout << "Withdrawn Successfully" << endl;
    }
    else
    {
        cout << "Insufficient funds" << endl;
    }
}
// accounts.count(username) != 0 &&
void buyBitcoin(std::string username, double amount)
{
    if (accounts[username].balance >= amount * bitcoin_price)
    {
        accounts[username].balance -= amount * bitcoin_price;
        accounts[username].bitcoin_holdings += amount;
       accounts[username].transactionHistory += "Bought " + std::to_string(amount)
+ " bitcoin\n";
         cout << "Bought " << to_string(amount) << " bitcoin(s) successfully" <<
endl;
    }
}

void sellBitcoin(std::string username, double amount)
{
     if (accounts.count(username) != 0 && accounts[username].bitcoin_holdings >=
amount)
    {
        accounts[username].balance += amount * bitcoin_price;
        accounts[username].bitcoin_holdings -= amount;
        accounts[username].transactionHistory += "Sold " + std::to_string(amount)
+ " bitcoin\n";
        cout << "Sold " << to_string(amount) << " bitcoin(s) successfully" << endl;
    }
}
void displayAccount(std::string username)
{
    if (accounts.count(username) != 0)
    {
        std::cout << "Balance: " << accounts[username].balance << std::endl;
```

```
        std::cout << "Bitcoin Holdings: " << accounts[username].bitcoin_holdings
<< std::endl;
                        std::cout     <<     "Transaction     History:     "     <<
accounts[username].transactionHistory << std::endl;
    }
}
void displaySortedAccount(std::string username)
{
    if (accounts.count(username) != 0)
    {
        std::cout << "Username: " << username << std::endl;
        std::cout << "Bitcoin Holdings: " << accounts[username].bitcoin_holdings
<< std::endl;
    }
}

void searchAccount(std::string username)
{
    if (accounts.count(username) != 0)
    {
        cout << "Account found." << endl;
    }
    else
    {
        cout << "Account not found." << endl;
    }
}
void sortAccounts()
{
    std::vector<std::pair<std::string, Account>> accounts_vec;
    for (auto it = accounts.begin(); it != accounts.end(); it++)
        accounts_vec.push_back(*it);

    sort(accounts_vec.begin(), accounts_vec.end(),
         [](auto &a, auto &b)
         { return a.second.bitcoin_holdings < b.second.bitcoin_holdings; });

    cout << "Sorted Accounts based on: bitcoin_holdings" << endl;
    for (auto it = accounts_vec.begin(); it != accounts_vec.end(); it++)
        displaySortedAccount(it->first);
}

void menu()
{
    int choice;
```

```
std::string username;
double amount;

while (true)
{
    cout << "-----------------------------------------" << endl;
    cout << "--------Welcome to Trading Software-------" << endl;
    cout << "1. Create Account" << endl;
    cout << "2. Deposit" << endl;
    cout << "3. Withdraw" << endl;
    cout << "4. Buy Cryptocurrency" << endl;
    cout << "5. Sell Cryptocurrency" << endl;
    cout << "6. Display Account Information" << endl;
    cout << "7. Search For Account" << endl;
    cout << "8. Sort Accounts" << endl;
    cout << "9. Exit" << endl;
    cout << "-----------------------------------------" << endl;
    cout << "\nEnter choice: ";
    cin >> choice;

    switch (choice)
    {
    case 1:
        cout << "Enter username: ";
        cin >> username;
        createAccount(username);
        break;
    case 2:
        cout << "Enter username: ";
        cin >> username;
        deposit(username);
        break;
    case 3:
        cout << "Enter username: ";
        cin >> username;
        withdraw(username);
        break;
    case 4:
        cout << "Enter username: ";
        cin >> username;
        cout << "Enter the amount to buy: ";
        cin >> amount;
        buyBitcoin(username, amount);
        break;
    case 5:
```

```
            cout << "Enter username: ";
            cin >> username;
            cout << "Enter the amount to sell: ";
            cin >> amount;
            sellBitcoin(username, amount);
            break;
        case 6:
            cout << "Enter username: ";
            cin >> username;
            displayAccount(username);
            break;
        case 7:
            cout << "Enter username: ";
            cin >> username;
            searchAccount(username);
            break;
        case 8:
            sortAccounts();
            break;
        case 9:
            cout << "Thank you for using our trading software!" << endl;
            return;
            break;
        default:
            cout << "Invalid choice. Please enter a valid option." << endl;
            break;
        }
    }
}
int main()
{
    menu();
    cout << "Do you wish to access the menu?" << endl;
    string choice;
    cin >> choice;
    if (choice == "Y")
    {
        menu();
    }
    else
    {
        cout << "Thankyou for using our trading software" << endl;
    }
}
```

Output

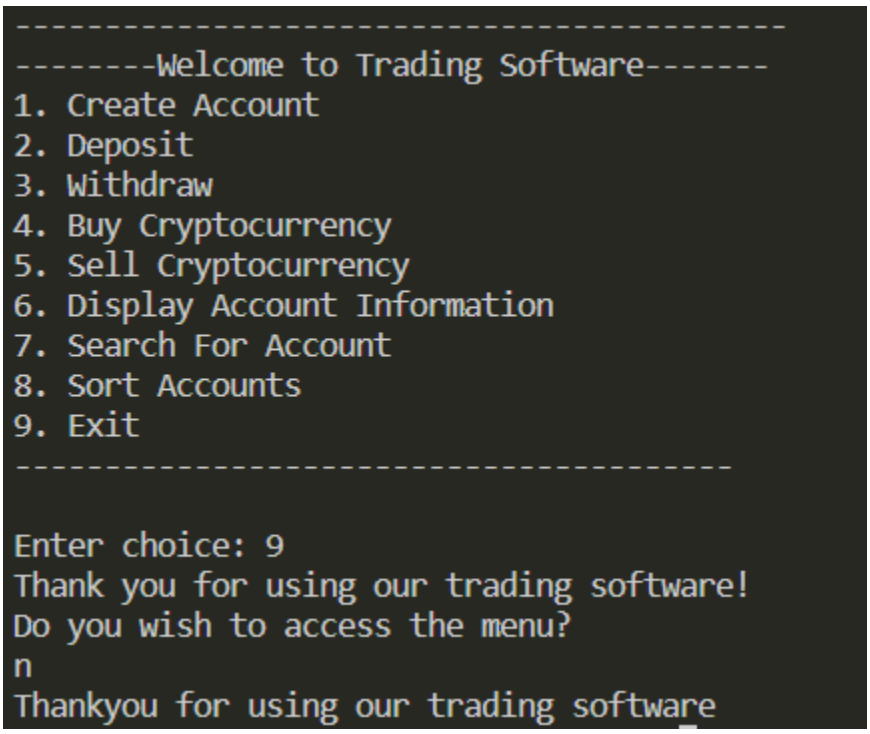


Figure i Main

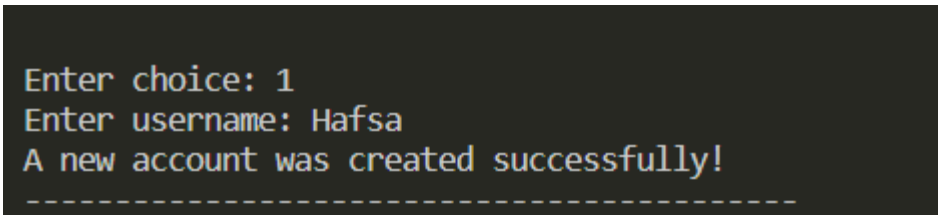


Figure ii Creating a new account

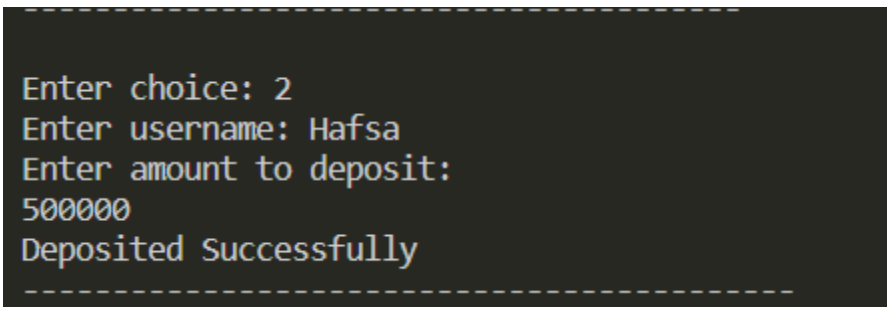


Figure iii Depositing Money to account "Hafsa"

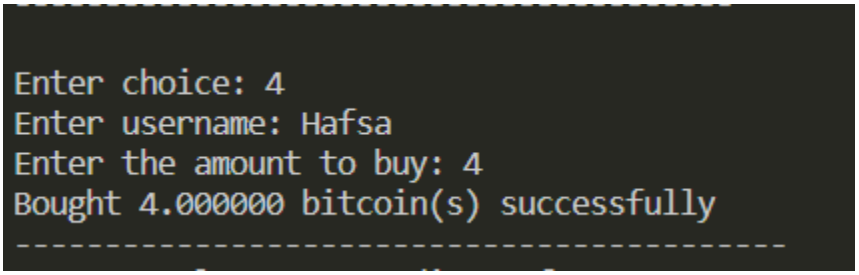


Figure iv Buying Bitcoins

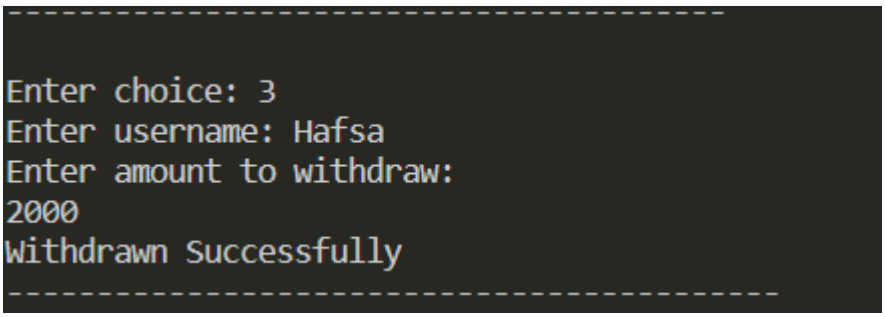


Figure v Withdrawing money

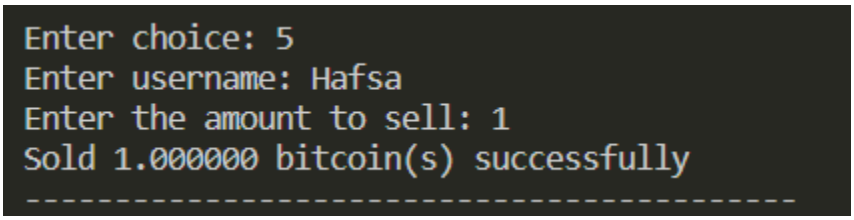


Figure vi Selling Bitcoins

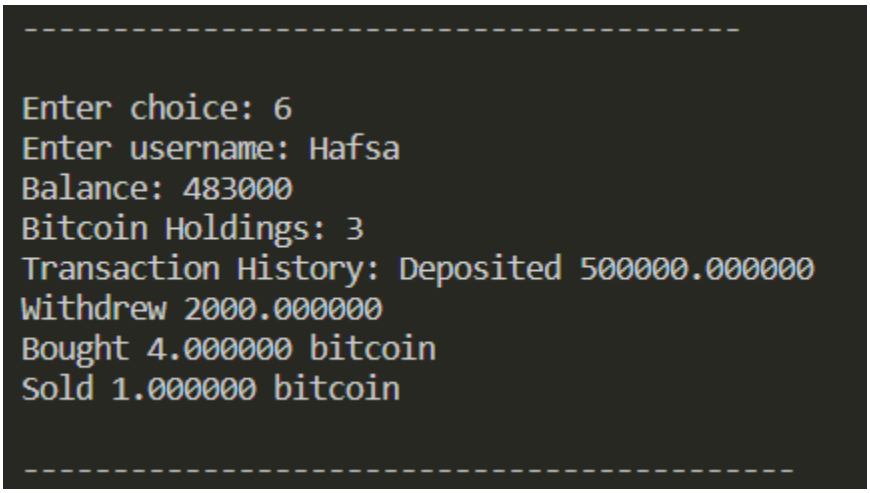


Figure vii Displaying Account

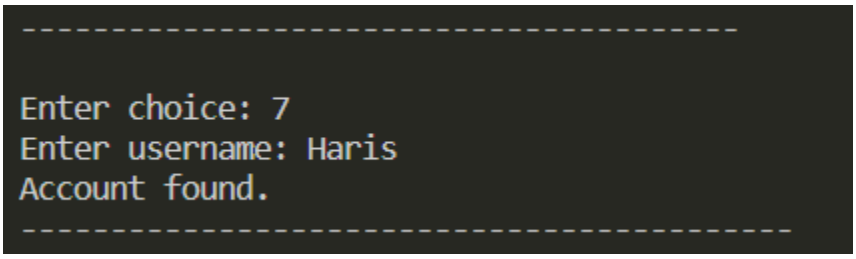


Figure viii Searching accounts

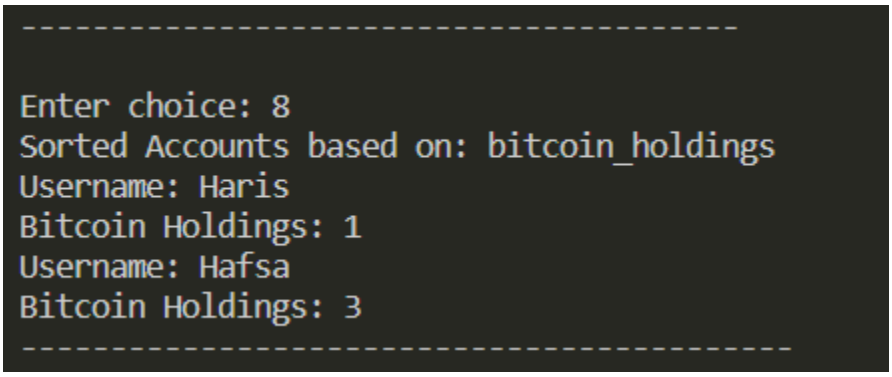


Figure ix Sorted Account Bitcoin Holding History

Conclusion

Trading Software is built through the concepts of data structures which has the main functions of buying and selling bitcoin and withdrawing or depositing money. Furthermore, the user can also view their own account information. The entire program can be accessed through a menu. Even though several data structures can be implemented on this hash maps or unordered maps and vectors are ideal because we require a key to access each account. That key is “username” and the storage of the vector is handled automatically, being expanded as needed.

References

FUN WITH MAPS IN C++. (2022, July 18). Retrieved from Quasar: https://quasar.ai/fun-with-maps-in-cpp/

Pradhan, S. (2022, December 13). *rand() and srand() in C++*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/

unordered_map. (2022, November 9). Retrieved from cppreference.com: https://en.cppreference.com/w/cpp/container/unordered_map

Vector. (2023, January 6). Retrieved from cppreference.com: https://en.cppreference.com/w/cpp/container/vector