数据结构与算法 I 实验报告

实验 3：区块链 (2)

龚舒凯 2022202790 应用经济-数据科学实验班

https://github.com/GONGSHUKAI

2023 年 10 月 27 日

区块链 (2):Script—基于堆栈的智能合约

1 需求分析

问题描述：在此前搭建的区块链中，交易产生的每个 output 都有一个 script(脚本) 字段，该字段的用途是“锁”住这个 output，使其只能被特定用户使用。

发布此交易的用户先选择一个字符串 x，计算其哈希值并放到脚本中。假设后续某个交易的 input 尝试使用这个 output，这个 input 的 scriptSig 内容必须为字符串 x。在验证交易时，系统将 input 的 scriptSig 和所引用的 output 的 script 合并，利用堆栈对该字符串序列进行验证，如果运行完毕验证通过，说明这个 input 可以使用相应 output。本实验要求基于栈实现简单的比特币脚本解析和交易 script 验证。

在本实验中，交易涉及到的脚本有以下三种类型：

1. **提供一个字符串得到指定哈希 (Puzzle):** 例如

```
Output = OP_HASH160 4389813341121533778 OP_EQUALVERIFY\\
Input: HELLOBLOCKCHAIN
```

只有当`std::hash("HELLOBLOCKCHAIN")`和 input 的 ScriptSig(这里是4389813341121533778) 相等时才能验证通过。

2. **中缀表达式计算:**

```
Output: OP_BEGIN_CALC 1 + 2 OP_END_CALC OP_EQUALVERIFY
Input: 3
```

这要求我们计算中缀表达式1 + 2，如果答案等于 input 中的 scriptSig(这里是3)，那么验证通过

3. **P2PKH 交易**：

```
Output: OP_DUP OP_HASH160 cbc20a7664f2 OP_EQUALVERIFY OP_CHECKSIG
Input: <signature> < public key>
```

这里执行了一个双重验证：首先要求 public key 在`std::hash()`加密后与 Output 的 Script 中对应部分匹配，其次要求 public key 经签名后能与 signature 匹配。两个要求都满足时才能验证通过。

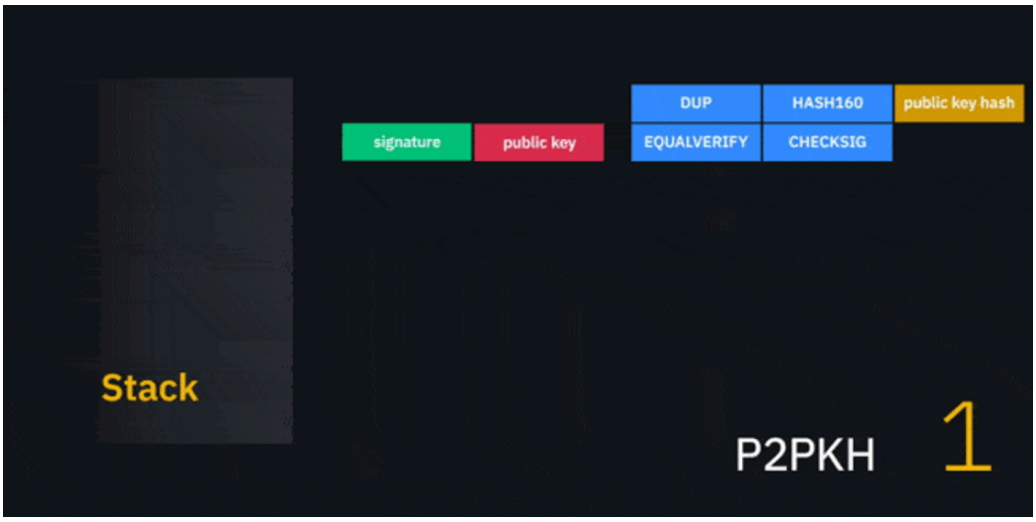

图 1: P2PKH

本次试验所需用到的操作码如下表所示:

操作码	含义
OP_BEGIN_CALC	定义一个四则运算表达式的开始。
OP_END_CALC	定义一个四则运算表达式的结束，计算表达式并将结果压入栈中。
OP_HASH160	弹出栈顶的数，调用 std::hash 计算，并将结果压入栈中。
OP_EQUALVERIFY	弹出栈顶两个操作数，判断它们是否相等。
OP_DUP	复制（非弹出）栈顶元素的值，并将复制的结果压入栈中。
OP_CHECKSIG	弹出栈顶两个操作数，其中第一个必须是公钥，第二个是对公钥的签名。验证该签名是否有效。

表 1: script-scriptSig 验证涉及到的操作码

基本要求：在实验 2 的基础上，增加验证交易的 Script 检测实验数据中交易合法性：

- 验证交易 input 与其使用的 output 的完整脚本，若通过则该 input 可以使用此 output，否则不能。
- 给定一组区块，以及一组交易，执行这些交易所引用的脚本，并输出脚本运行结果。

输出形式:

1. 打印区块总数、合法交易总数、不合法的交易总数。
2. 从键盘输入区块高度 (Height)，输出该区块内容。
3. 从键盘输入交易号 txid，输出该交易内容。

2 具体实现

2.1 区块链的定义

此前对区块链各要素的结构体定义依旧沿用，这里只增加一个全局栈`std::stack <std::string> SmartContract;`用来存放操作码 (如 OP_HASH160)，数字 (如 3)，字符串 (如 Welcome-to-RUC)，用于验证 script 和 scriptSig 是否能匹配上。

2.2 操作码、数与字符串的提取

将 input 的 scriptSig 和 output 的 script 拼接得到一个长字符串后，我们根据空格将长字符串切割成一个个部分。例如

`Welcome-to-RUC OP_HASH160 3528693918439319417 OP_EQUALVERIFY`

可以分割为下列几个部分`"Welcome-to-RUC","OP_HASH160","3528693918439319417","OP_EQUALVERIFY"`，然后将这一个个提取出来的部分入栈`SmartContract`，再根据操作码的规则对栈中元素进行运算。代码实现如下：

```
bool CheckIOScript(std::string scriptSig, std::string script){
    keywords.push_back("OP_HASH160");
    keywords.push_back("OP_BEGIN_CALC");
    keywords.push_back("OP_DUP");
    keywords.push_back("OP_EQUALVERIFY");
    keywords.push_back("OP_CHECKSIG");
    SmartContract.push(scriptSig);
    size_t pos = 0;
    flag = 1;
    while (pos < script.length()) {
        std::string keyword;
        size_t spacePos = script.find(' ', pos);
        if (spacePos != std::string::npos) {
            keyword = script.substr(pos, spacePos - pos);
        }
        else {
            keyword = script.substr(pos);
        }
        Execution(keyword, script, scriptSig);

        if (keyword == "OP_BEGIN_CALC") {
            size_t endCalcPos = script.find("OP_END_CALC", pos);
            if (endCalcPos != std::string::npos) {
                pos = endCalcPos + sizeof("OP_END_CALC") - 1;
            }
            else {
                std::cout << "OP_BEGIN_CALC found, but OP_END_CALC is missing" << std::endl;
```

```
                break;
            }
        }
        else {
            pos = spacePos != std::string::npos ? spacePos + 1 : script.length();
        }
    }
    if (flag == 1) return true;
    else return false;
}
```

这是 script-scriptSig 验证的基本函数，其核心思想如下：

1. 这里用到一个`std::vector <std::string> keywords`存放操作码 (即 OP_HASH160 等)。
2. 先将属于 inputs 的 scriptSig 压栈。
3. 遍历并按照空格分割`script`。再调用一个函数`Execution(keyword, script, scriptSig)`函数，将提取出来的操作码/数/字符串传入`Execution()`函数后，`Execution()`根据传入的`keyword`执行对应的操作。
   (a) 这里执行了一个特殊判断：如果扫描到的操作码 (keyword) 是 OPEN_CALC，说明从 OP_BEGIN_CALC 到 OP_END_CALC 都是中缀表达式。于是可以让分割的光标直接跳到 OP_END_CALC 之后，节省运算时间并避免不必要的压栈操作。
   (b) `Execution(keyword, script, scriptSig)`的具体实现在”**2.3 6 个操作码的函数实现**” 中详述。
   (c) 如果在遍历`script`过程中，任何一个操作码 (keyword) 下的验证不成立，则置符号变量`flag = 0`为 0，表示使用某条 output 的某条 input 是不合法的，进而这条交易是不合法的。
4. 最后根据符号变量`flag`的取值判断该条交易的 script-scriptSig 验证是否合法。

2.3 6 个操作码的函数实现

根据表1对六种操作码实现功能的叙述，编写六个操作码的函数实现：

1. `OP_HASH160` 弹出栈顶的 scriptSig，调用`std::hash`计算，并将结果压入栈中。

```
void OP_HASH160(std::string script){
    //弹出栈顶的数，调用 std::hash 计算，并将结果压入栈中
    std::string calc = SmartContract.top();//取栈顶元素
    SmartContract.pop();
    std::hash <std::string> h;//调用哈希函数
    SmartContract.push(std::to_string(h(calc)));//计算 calc 的哈希值，并压入栈
    /* 观察到 OP_HASH160 后面总会再跟一个验证串 (例如: OP_HASH160 17730432883776450158)
    把这个串也顺便压进栈内，从而便于后续计算 (OP_EQUALVERIFY)*/
    //先找到 OP_HASH160 后面空格的位置
    std::string substr = "OP_HASH160";
```

```
        size_t pos = script.find(substr);
        size_t startPos = pos + substr.length();//OP_HASH160 后第一个空格的位置
        std::string substr2 = "OP_EQUALVERIFY";
        size_t endPos = script.find(substr2, startPos);
        std::string VerifyStr = script.substr(startPos + 1, endPos - startPos - 2);
        SmartContract.push(VerifyStr);
    }
```

2. OP_EQUALVERIFY 弹出栈顶两个操作数，判断它们是否相等。

```
    void OP_EQUALVERIFY(){//弹出栈顶两个操作数，判断它们是否相等
        std::string top1 = SmartContract.top();
        SmartContract.pop();
        std::string top2 = SmartContract.top();
        SmartContract.pop();
        if (top1 == top2) flag = 1;
        else flag = 0;
    }
```

3. OP_DUP 复制（非弹出）栈顶元素的值，并将复制的结果压入栈中。

```
    void OP_DUP(){//复制（非弹出）栈顶元素的值，并将复制的结果压入栈中
        std::string sig = SmartContract.top();//先将之前压入的 input 的 scriptSig 取出
        std::string signature;
        std::string public_key;
        //将 scriptSig 分解为 signature 和 public key 两部分
        unsigned long long pos = sig.find(' ');
        if (pos != std::string::npos){
        //如果没有返回一个无效的位置，即 sig 中存在空格（例如：22308B989987AEE01DED1BAC4C84497F 202210
            signature = sig.substr(0,pos);//22308B989987AEE01DED1BAC4C84497F 是 signature
            public_key = sig.substr(pos + 1, sig.length());//20221009 是 public key
            SmartContract.push(signature);//将 signature 压入栈中
            SmartContract.push(public_key);//将 public_key 压入栈中
            SmartContract.push(public_key);//将 public_key 再压入栈中（相当于复制）
        }
        else{
        //如果返回一个无效的位置（scriptSig 中根本没有空格，例如 2023）
            SmartContract.push(sig);
            SmartContract.push(sig);//DUP 指令将 sig 复制了一遍再压进栈
        }
    }
```

4. OP_BEGIN_CALC 定义一个四则运算表达式的开始，计算表达式并将结果压入栈中。

```
void OP_BEGIN_CALC(std::string scriptSig, std::string script){
    //先找到 OP_BEGIN_CALC 后面第一个空格的位置
    std::string substr = "OP_BEGIN_CALC";
    unsigned long long startPos = script.find(substr);
    startPos += substr.length();
    //再找到 OP_END_CALC 的位置
    unsigned long long endPos = script.find("OP_END_CALC", startPos);
    std::string infix_expression = script.substr(startPos + 1,endPos - startPos - 2);
    //str.substr(a,b) 返回从 a 开始，长度为 b 的子串
    int idx = 0;
    char converted_infix[MAXSIZE];
    //把 script 的子串 infix_expression 转换为 Infix_Calculation.cpp 能读取的形式
    for (int i = 0 ; i < infix_expression.size() ; i++){
        if (infix_expression[i] != ' '){
            converted_infix[idx] = infix_expression[i];
            idx++;
        }
    }
    converted_infix[idx] = '=';
    Mystack *OPND = InitStack();
    char_stack *OPTR = InitCharStack();
    int ans = InfixCalculation(converted_infix, OPND, OPTR);
    SmartContract.push(std::to_string(ans));//把中缀表达式运算结果压入栈中
}
```

需要说明的是，这里调用了实验 (1) 中缀表达式计算的程序InfixCalculation(infix, OPND, OPTR)。将实验 (2)“中缀表达式计算”的程序封装为头文件"InfixCalculation.h"后，在区块链程序中直接调用函数即可。

5. OP_CHECKSIG 弹出栈顶两个操作数，其中第一个必须是公钥，第二个是对公钥的签名。验证该签名是否有效。

```
void OP_CHECKSIG(){
    std::string public_key = SmartContract.top();
    SmartContract.pop();
    std::string signature = SmartContract.top();
    SmartContract.pop();
    BigInt MySignature(signature);

    Rsa rsaSign; //rsa 对象，用于签名
```

```
        Rsa rsaVerify; //rsa 对象，用于验证签名
        rsaVerify.init(0); //初始化
        rsaSign.setPu(std::stoi(public_key));
        flag = verify(rsaVerify, MySignature, public_key);
    }
```

这里调用了头文件"rsa.h"中的大整数类型 (BigInt) 与Rsa rsaVerify来对公钥进行签名。

最后，在定义完这六个函数后，用函数Execution(keyword, script, scriptSig)来作分支判断：

```
void Execution(std::string keyword, std::string script, std::string scriptSig){
    if (keyword == "OP_HASH160"){
        OP_HASH160(script);
    }
    else if (keyword == "OP_BEGIN_CALC"){
        OP_BEGIN_CALC(scriptSig, script);
    }
    else if (keyword == "OP_DUP"){
        OP_DUP();
    }
    else if (keyword == "OP_EQUALVERIFY"){
        OP_EQUALVERIFY();
    }
    else if (keyword == "OP_CHECKSIG"){
        OP_CHECKSIG();
    }
    else{
    }
}
```

2.4 交易合法性的判断

只需在实验 2 区块链 (1) 的交易合法性判断函数IOCheck(firstblock, currentblock, transaction t)中增添对 script-scriptSig 的判断CheckIOScript(std::string scriptSig, std::string script)即可

```
ol IOCheck(block *firstblock, block *currentblock, transaction t){
  int i = 0;
  while (t.inputs[i].scriptSig != ""){
      transaction *PrevOutput = FindPrevOutput(t.inputs[i].prevTxID, firstblock, currentblock);
      if (PrevOutput != nullptr){
          if (PrevOutput->outputs[t.inputs[i].prevTxOutIndex].IsUse == NotUsed &&
              PrevOutput->valid == yes &&
              CheckIOScript(t.inputs[i].scriptSig, PrevOutput->outputs[t.inputs[i].prevTxOutIndex].scri
```

```
9               int sum_in = 0;
10              int sum_out = 0;
11              for (int j = 0 ; j < t.input_count ; j++) sum_in += PrevOutput->outputs[t.inputs[j].prevT
12              for (int j = 0 ; j < t.output_count ; j++) sum_out += t.outputs[j].value;
13              if (sum_in >= sum_out){
14                  PrevOutput->outputs[t.inputs[i].prevTxOutIndex].IsUse = Used;
15                  return true;
16              }
17              else return false;
18          }
19      }
20      i++;
21  }
22  return false;
23
```

3 使用说明与程序测试样例

为了更直观地展示，这里我们选取一个交易的一个输入和输出来查看 script-scriptSig 的验证结果：区块Height = 4中的交易"591e91f809d716912ca1d4a9295e70c3e78bab077683f79350f101da64588073"的**输入**用到了区块Height = 3的交易"5a916d9e74946ed6f3c2aec1acea20ae59a2af216eb9b33f91a0771f20678bed"的**0 号输出**。当前输入的 scriptSig 和用到输出的 script 如下所列

```
(outputs) script: OP_DUP OP_BEGIN_CALC ( 100 * 10 + 24 ) / ( 2 + 2 ) + 1767 OP_END_CALC
                  OP_EQUALVERIFY OP_HASH160 3858365116623476756 OP_EQUALVERIFY
(inputs) scriptSig: 2023
```

我们先手动的检验该 scriptSig 能否和 script 匹配上：

1. 将scriptSig(也就是 2023) 压入栈，开始扫描script。
2. 执行OP_DUP，将 2023 复制一遍再压入栈。
3. 执行OP_BEGIN_CALC，计算中缀表达式 (100 * 10 + 24) / (2 + 2) + 1767，得 2023，并跳到OP_END_CALC之后。
4. 执行OP_EQUALVERIFY，弹出栈顶两个元素 (2023, 2023)，判断相等。
5. 执行OP_HASH160，弹出栈顶元素 (2023)，计算其哈希值 (h(2023) = "3858365116623476756")，然后压入栈中，并将OP_HASH160后的字符串"3858365116623476756"压入栈中。
6. 执行OP_EQUALVERIFY，弹出栈顶两个元素 ("3858365116623476756","3858365116623476756")，判断相等。

经过这 6 轮判断，发现该 scriptSig 能和 script 匹配上，故此条 inputs-outputs 组合合法！程序运行结果如下，符合我们的预期：

```
void CheckIOScript_Test(){
    std::string scriptSig = "2023";
    std::string script = "OP_DUP OP_BEGIN_CALC ( 100 * 10 + 24 ) / ( 2 + 2 ) + 1767 OP_END_CALC OP_EQ
    std::cout << CheckIOScript(scriptSig, script) << std::endl;
}
```

输出结果

```
1
```

在输出完脚本运行结果后，仍然可以键盘输入区块高度 (Height) 和交易号 (txid) 来查询区块具体信息和交易号所对应交易信息：

```
Please input block's height:
```

此时输入一个区块的 Height(以输入 114 为例)，将显示该区块的信息：

```
Block Height: 114
Block Hash: 000000005d7dd40c24c4b6334812f48d4386c7756fb9006d6c74300837c91ebd
Block prevHash: 0000000019176838de40606d70738084f2fbc48a50548eeeac3ceb857677c6d
Block merkleRoot: ca7b0295546806c9b6631f8bc89ed0b55434ee5f8ce6198a918af18c6da9237c
Block nonce: 2979771436

Block Transaction 0 txid: ca7b0295546806c9b6631f8bc89ed0b55434ee5f8ce6198a918af18c6da9237c
Block Transaction 0 input count: 1
Block Transaction 0 output count: 1
Block Transaction 0 Coinbase: 1
Transaction 0 in Block: 114
Transaction 0 input count: 1
Transaction 0 output count: 1
Transaction 0 Coinbase: 1
Transaction 0 output 0 txid: ca7b0295546806c9b6631f8bc89ed0b55434ee5f8ce6198a918af18c6da9237c
Transaction 0 output 0 index: 0
Transaction 0 output 0 value: 500000
Transaction 0 output 0 script: 4104cd10ce592a9c4918948a5b4e92e9702e6c36eed1a627594f67066792b3a227cedb
```

接着显示

```
Please input transaction's txid:
```

此时输入一个交易的 txid(以输入 d9df26d62a3ce4855f3282462ba5581b23dc51ca3595de810115c0e7176722d3 为例)，将显示该交易所属的区块，该交易共有几个 input，几个 output，coinbase 是什么：

```
Transaction 0 in Block: 514
Transaction 0 input count: 1
Transaction 0 output count: 1
Transaction 0 Coinbase: 1
```

4 附录

根目录下所有文件(包括rsa.h, InfixCalculation.h)详见https://github.com/GONGSHUKAI/Data_Structure/tree/main/Lab_Code/Lab_2/Oct.13_Lab，这里只展示区块链的主程序代码Block_Chain_2.cpp。

```
nclude <iostream>
nclude <stack>
nclude <vector>
nclude <fstream>
nclude <sstream>
nclude <string>
nclude <ctime>
nclude <cmath>
nclude "rsa.h"
nclude "Infix_Calculation.h"
efine yes 100
efine no -100
efine NotUsed 200
efine Used -200
efine TransactionExist 999

efine MAXTRANS 100//一个块内最高有 100 条交易信息
efine MAXINPUT 100//一条交易信息中最高有 100 个输入
efine MAXOUTPUT 100//一条交易信息中最高有 100 个输出
efine MAXSIZE 200

pedef struct output{
  std::string txid;//该 output 所属的交易
  unsigned long long index;//该 output 在所属交易中的索引值
  unsigned long long value;//该 output 的价值 (数据已乘 10^8, 避免浮点误差)
  std::string script;//脚本

  int IsUse = NotUsed;
utput;

pedef struct input{
  unsigned long long pre_block;//该 input 所引用的 output 所在区块的高度
  std::string prevTxID;//该 input 所引用的 output 所在交易的 txID
  unsigned long long prevTxOutIndex;//该 input 所引用的 output 位于所在交易 output 集合中的索引
  std::string scriptSig;//脚本和签名
nput;

pedef struct transaction{
```

```
39   std::string txid;//交易的编号，具有唯一性
40   unsigned long long input_count;//inputs 的数量
41   unsigned long long output_count;//outputs 的数量
42   input inputs[MAXINPUT];//一组 input 的集合，表示当前交易的输入所用到的输出
43   output outputs[MAXOUTPUT];//一组 output 的集合，表示当前交易的输出
44   int is_coinbase;//1 为 coinbase 交易，0 为非 coinbase 交易
45
46   int valid = yes;
47 ransaction;
48
49 pedef struct block{
50   unsigned long long height;//当前块的高度，一条链上每个区块的 Height 均不相同
51   std::string hash;//本区块的哈希值
52   std::string prevHash;//前一个区块的哈希值
53   std::string merkleRoot;//本区块中所有交易的默克尔树根
54   unsigned long long nonce;//神秘数
55   transaction transactions[MAXTRANS];//一组 transaction 的集合
56
57   struct block *next;
58 lock;
59
60 ock* FileToBlock();
61 id FileToTransaction(block *currentBlock);
62 id FileToInput(block *currentBlock);
63 id FileToOutput(block *currentBlock);
64 t BlocksLength(block *firstblock);
65 id BlockInfo(int height, block *firstblock);
66 t TransactionInfo(std::string txid, block *firstblock, block *endblock);
67 ock* InitBlockChain();
68 id CheckValidTransaction(block *firstblock);
69 ol IOCheck(block *firstblock, block *currentblock, transaction t);
70 ansaction* FindPrevOutput(std::string My_txid, block *firstblock, block *endblock);
71 ol CheckIOScript(std::string scriptSig, std::string script);
72
73 d::stack <std::string> SmartContract;
74 d::vector <std::string> keywords;
75 t flag = 1;
76
77 ock* FileToBlock() {
78   block *firstBlock = new block;
79   std::ifstream file("demo/blocks.csv"); //打开 CSV 文件
80   if (!file) {
```

```
81          std::cout << " 无法打开文件" << std::endl;
82          return nullptr;
83      }
84      std::string line;
85      std::getline(file, line); // 读取第一行标题（忽略）
86      /*std::getline 函数返回一个布尔值，表示是否成功读取一行内容。
87      如果读取成功，则返回 true;
88      如果已到达文件末尾或发生错误，则返回 false。*/
89      block *currentBlock = nullptr;
90      block *prevBlock = nullptr;
91      while (std::getline(file, line)) {
92          std::stringstream ss(line);//创建一个 std::std::stringstream 对象，字符串 ss 作为初始输入。
93          std::string cell;
94          int column = 0;
95          block *newBlock = new block();// 创建新的 block
96
97          while (std::getline(ss, cell, ',')){
98          //使用逗号作为分隔符，从 ss 中提取每个字段的内容，并将其存储到字符串变量 cell 中
99              switch (column) {
100                 case 0:
101                     newBlock->height = stoull(cell);//stoi 函数将字符串 cell 转换为 unsigned long long
102                     break;
103                 case 1:
104                     newBlock->hash = cell;
105                     break;
106                 case 2:
107                     newBlock->prevHash = cell;
108                     break;
109                 case 3:
110                     newBlock->merkleRoot = cell;
111                     break;
112                 case 4:
113                     newBlock->nonce = stoull(cell);
114                     break;
115                 default://当没有匹配到任何 case 标签时执行的代码块。
116                     break;
117             }
118             column++;
119         }
120         // 如果是第一个 block，则将其设置为第一个 block
121         if (prevBlock == nullptr) {
122             firstBlock = newBlock;
```

```
123             currentBlock = newBlock;
124         } else {
125             prevBlock->next = newBlock; // 将前一个 block 的 next 指针指向新的 block
126             newBlock->next = nullptr;
127             currentBlock = newBlock;
128         }
129         prevBlock = currentBlock;
130     }
131     file.close(); // 关闭文件
132     return firstBlock;
133 
134 
135 id FileToTransaction(block *currentBlock){
136     std::ifstream file("demo/transactions.csv");
137     if (!file) {
138         std::cout << " 无法打开文件" << std::endl;
139         return;
140     }
141     std::string line;
142     std::getline(file, line);
143     int transIndex = 0;
144 
145     while (std::getline(file, line)) {
146         std::stringstream ss(line);
147         std::string cell;
148         int column = 0;
149         int transHeight = 0;
150         transaction currentTrans;
151         currentTrans.valid = yes;
152         while (std::getline(ss, cell, ',')) {//扫描一行的数据，以逗号为分隔符
153             switch (column) {
154                 case 0:
155                     transHeight = stoull(cell);
156                     break;
157                 case 1:
158                     currentTrans.txid = cell;
159                     break;
160                 case 2:
161                     currentTrans.is_coinbase = stoi(cell);
162                     break;
163                 case 3:
164                     currentTrans.input_count = stoull(cell);
```

```
165                     break;
166                 case 4:
167                     currentTrans.output_count = stoull(cell);
168                     break;
169                 default:
170                     break;
171             }
172             column++;
173         }
174         if (transHeight == currentBlock->height){
175             currentBlock->transactions[transIndex] = currentTrans;
176             transIndex++;
177         }
178         else{
179             while (currentBlock->height != transHeight){
180                 currentBlock = currentBlock->next;
181             }
182             transIndex = 0;
183             currentBlock->transactions[transIndex] = currentTrans;
184             transIndex++;
185         }
186     }
187     file.close();
188 
189 
190 id FileToInput(block *currentBlock){
191     std::ifstream file("demo/inputs.csv");
192     if (!file) {
193         std::cout << " 无法打开文件" << std::endl;
194         return;
195     }
196     std::string line;
197     std::getline(file, line);
198     int transIndex = 0;
199     int inputsIndex = 0;
200     while (std::getline(file, line)) {
201         std::stringstream ss(line);
202         std::string cell;
203         int column = 0;
204         int inputHeight = 0;
205         std::string input_txid;
206         input currentInput;
```

```
207         while (std::getline(ss, cell, ',')) {//扫描一行的数据，以逗号为分隔符
208             switch (column) {
209                 case 0:
210                     inputHeight = stoull(cell);
211                     break;
212                 case 1:
213                     input_txid = cell;
214                     break;
215                 case 2:
216                     currentInput.pre_block = stoi(cell);
217                     break;
218                 case 3:
219                     currentInput.prevTxID = cell;
220                     break;
221                 case 4:
222                     currentInput.prevTxOutIndex = stoull(cell);
223                     break;
224                 case 5:
225                     currentInput.scriptSig = cell;
226                 default:
227                     break;
228             }
229             column++;
230         }
231         if (inputHeight == currentBlock->height && input_txid == currentBlock->transactions[transIndex].t
232             currentBlock->transactions[transIndex].inputs[inputsIndex] = currentInput;
233             inputsIndex++;
234         }
235         else{
236             while (currentBlock->height != inputHeight){
237                 currentBlock = currentBlock->next;
238             }
239             while (currentBlock->transactions[transIndex].txid != input_txid && currentBlock->transaction
240                 transIndex++;
241             }
242             inputsIndex = 0;
243             currentBlock->transactions[transIndex].inputs[inputsIndex] = currentInput;
244             inputsIndex++;
245         }
246     }
247     file.close();
248
```

```
249
250 id FileToOutput(block *currentBlock){
251   std::ifstream file("demo/outputs.csv");
252   if (!file) {
253       std::cout << " 无法打开文件" << std::endl;
254       return;
255   }
256   std::string line;
257   std::getline(file, line);
258   int transIndex = 0;
259   int outputsIndex = 0;
260   while (std::getline(file, line)) {
261       std::stringstream ss(line);
262       std::string cell;
263       int column = 0;
264       int outputHeight = 0;
265       std::string output_txid;
266       output currentOutput;
267       currentOutput.IsUse = NotUsed;
268       while (std::getline(ss, cell, ',')) {//扫描一行的数据，以逗号为分隔符
269           switch (column) {
270               case 0:
271                   outputHeight = stoull(cell);
272                   break;
273               case 1:
274                   currentOutput.txid = cell;
275                   break;
276               case 2:
277                   currentOutput.index = stoi(cell);
278                   break;
279               case 3:
280                   currentOutput.value = stoull(cell);
281                   break;
282               case 4:
283                   currentOutput.script = cell;
284                   break;
285               default:
286                   break;
287           }
288           column++;
289       }
290       if (outputHeight == currentBlock->height && currentOutput.txid == currentBlock->transactions[tran
```

```
291             currentBlock->transactions[transIndex].outputs[outputsIndex] = currentOutput;
292             outputsIndex++;
293         }
294         else{
295             while (currentBlock->height != outputHeight){
296                 currentBlock = currentBlock->next;
297             }
298             while (currentBlock->transactions[transIndex].txid != currentOutput.txid && currentBlock->tra
299                 transIndex++;
300             }
301             outputsIndex = 0;
302             currentBlock->transactions[transIndex].outputs[outputsIndex] = currentOutput;
303             outputsIndex++;
304         }
305     }
306     file.close();
307 
308 
309 id BlockInfo(int height, block *firstblock){
310     //输入区块高度，输出该区块内容
311     block *p = firstblock;
312     while (height != p->height && p != nullptr){
313         p = p->next;
314     }//找到 height 值对应的 block
315     std::cout << "Block Height: " << p->height << std::endl;
316     std::cout << "Block Hash: " << p->hash << std::endl;
317     std::cout << "Block prevHash: " << p->prevHash << std::endl;
318     std::cout << "Block merkleRoot: " << p->merkleRoot << std::endl;
319     std::cout << "Block nonce: " << p->nonce << std::endl;
320     int i = 0;
321     std::cout << std::endl;
322     while (p->transactions[i].txid != ""){
323         std::cout << "Block Transaction " << i << " txid: " << (p->transactions[i]).txid << std::endl;
324         std::cout << "Block Transaction " << i << " input count: " << (p->transactions[i]).input_count <<
325         std::cout << "Block Transaction " << i << " output count: " << (p->transactions[i]).output_count
326         std::cout << "Block Transaction " << i << " Coinbase: " << (p->transactions[i]).is_coinbase << st
327         TransactionInfo(p->transactions[i].txid, firstblock, nullptr);
328         std::cout << std::endl;
329         i++;
330     }
331     std::cout << std::endl;
332 
```

```
333
334 t BlocksLength(block *firstblock){
335   block *p = firstblock;
336   int len = 0;
337   while (p != nullptr){
338       len++;
339       p = p->next;
340   }
341   return len;
342
343
344 t TransactionInfo(std::string My_txid, block *firstblock, block *endblock){
345   block *p = firstblock;
346   int find = 0;//找到交易信息则 find = 1, 否则 find = 0
347   int i = 0;
348   int j = 0;
349   int k = 0;
350   while (p != endblock){
351       while (p->transactions[i].txid != ""){
352           if (p->transactions[i].txid == My_txid){
353               std::cout << "Transaction "<< i <<" in Block: " << p->height << std::endl;
354               std::cout << "Transaction "<< i <<" input count: " << (p->transactions[i]).input_count <<
355               std::cout << "Transaction "<< i <<" output count: " << (p->transactions[i]).output_count
356               std::cout << "Transaction "<< i <<" Coinbase: " << (p->transactions[i]).is_coinbase << st
357               while (p->transactions[i].inputs[j].scriptSig != ""){
358                   std::cout << "Transaction "<< i <<" input " << j << " pre_block: " << (p->transaction
359                   std::cout << "Transaction "<< i <<" input " << j << " prevTxID: " << (p->transactions
360                   std::cout << "Transaction "<< i <<" input " << j << " prevTxOutIndex: " << (p->transa
361                   std::cout << "Transaction "<< i <<" input " << j << " scriptSig: " << (p->transaction
362                   j++;
363               }
364               while (p->transactions[i].outputs[k].script != ""){
365                   std::cout << "Transaction "<< i << " output " << k << " txid: " << (p->transactions[i
366                   std::cout << "Transaction "<< i << " output " << k << " index: " << (p->transactions[
367                   std::cout << "Transaction "<< i << " output " << k << " value: " << (p->transactions[
368                   std::cout << "Transaction "<< i << " output " << k << " script: " << (p->transactions
369                   k++;
370               }
371               find = 1;//找到这条交易记录
372               break;
373           }
374           i++;
```

```
375         }
376         if (find == 1) break;
377         else{
378             i = 0;
379             p = p->next;
380         }
381     }
382     if (find == 1) return find;
383     else{
384         std::cout << "Transaction Not Found!" << std::endl;
385         return find;
386     }
387 
388 
389 ock* InitBlockChain(){
390     block* firstBlock = FileToBlock();
391     FileToTransaction(firstBlock);
392     FileToInput(firstBlock);
393     FileToOutput(firstBlock);
394     return firstBlock;
395 
396 
397 id CheckValidTransaction(block *firstblock){
398     block *p = firstblock;
399     int i = 0;
400 
401     int valid = 0;
402     int invalid = 0;
403     //规则 1: 每个 input 所使用的 output 能够找到。
404     //规则 2: 每个 input 所使用的 output 没有被之前的交易用过。
405     //规则 3: 该交易所有 input 所引用的 output 的价值（value）之和大于等于该交易所有 output 的价值（value）之
406 
407     //a、有一类特殊交易，其 is_coinbase 字段为 true，该类交易的特点是没有 input，只有 output。该类交易是合法
408     //b、每一个 output 只能被使用一次，即便还有剩余的 value 没有被使用。
409     //c、如果某个交易是非法的，那么引用了该交易作为 input 的交易也同样是非法的（非法交易不会被包括在区块内，如
410     while (p != nullptr){
411         while (p->transactions[i].txid != ""){
412             //规则 1: 其 is_coinbase 字段为 true，该类交易的特点是没有 input，只有 output。该类交易是合法的，
413             if (p->transactions[i].is_coinbase == 1){
414                 valid++;
415                 i++;
416                 continue;
```

```
417             }
418             //规则 2: 如果每个 input 所使用的 output 找不到, 交易不合法;
419             //规则 3: 每个 input 所使用的 output 被之前的交易用过, 交易不合法
420             //规则 4: 如果引用的 output 来自不合法的交易, 交易不合法
421             //规则 5: 所引用的 output 的 value 之和 < 该所有 output 的 value 之和, 交易不合法
422             else if (IOCheck(firstblock, p, p->transactions[i]) == false){
423                 p->transactions[i].valid = no;
424                 invalid++;
425                 i++;
426                 continue;
427             }
428             else{//
429                 valid++;
430                 i++;
431                 continue;
432             }
433         }
434         i = 0;
435         p = p->next;
436     }
437 
438     std::cout << " 不合法交易数: " << invalid << std::endl;
439     std::cout << " 合法交易数: " << valid << std::endl;
440 
441 
442 ol IOCheck(block *firstblock, block *currentblock, transaction t){
443     int i = 0;
444     while (t.inputs[i].scriptSig != ""){
445         transaction *PrevOutput = FindPrevOutput(t.inputs[i].prevTxID, firstblock, currentblock);
446         if (PrevOutput != nullptr){
447             if (PrevOutput->outputs[t.inputs[i].prevTxOutIndex].IsUse == NotUsed &&
448                 PrevOutput->valid == yes &&
449                 CheckIOScript(t.inputs[i].scriptSig, PrevOutput->outputs[t.inputs[i].prevTxOutIndex].scri
450                 int sum_in = 0;
451                 int sum_out = 0;
452                 for (int j = 0 ; j < t.input_count ; j++) sum_in += PrevOutput->outputs[t.inputs[j].prevT
453                 for (int j = 0 ; j < t.output_count ; j++) sum_out += t.outputs[j].value;
454                 if (sum_in >= sum_out){
455                     PrevOutput->outputs[t.inputs[i].prevTxOutIndex].IsUse = Used;
456                     return true;
457                 }
458                 else return false;
```

```
459             }
460         }
461         i++;
462     }
463     return false;
464 
465 
466 ansaction* FindPrevOutput(std::string My_txid, block *firstblock, block *endblock){
467     //根据 PrevTxid 找到当前 input 所用的 output 的 Txid，再根据当前 prevTxOutIndex 找到所用的 output
468     block *p = firstblock;
469     int i = 0;
470     while (p != endblock){
471         while (p->transactions[i].txid != ""){
472             if (p->transactions[i].txid == My_txid) return &p->transactions[i];
473             else i++;
474         }
475         i = 0;
476         p = p->next;
477     }
478     return nullptr;
479 
480 
481 id OP_HASH160(std::string script){
482     //弹出栈顶的数，调用 std::hash 计算，并将结果压入栈中
483     std::string calc = SmartContract.top();//取栈顶元素
484     SmartContract.pop();
485     std::hash <std::string> h;//调用哈希函数
486     SmartContract.push(std::to_string(h(calc)));//计算 calc 的哈希值，并压入栈
487     /* 观察到 OP_HASH160 后面总会再跟一个验证串 (例如: OP_HASH160 17730432883776450158)
488     把这个串也顺便压进栈内，从而便于后续计算 (OP_EQUALVERIFY)*/
489     //先找到 OP_HASH160 后面空格的位置
490     std::string substr = "OP_HASH160";
491     size_t pos = script.find(substr);
492     size_t startPos = pos + substr.length();//OP_HASH160 后第一个空格的位置
493     std::string substr2 = "OP_EQUALVERIFY";
494     size_t endPos = script.find(substr2, startPos);
495     std::string VerifyStr = script.substr(startPos + 1, endPos - startPos - 2);
496     SmartContract.push(VerifyStr);
497 
498 
499 id OP_BEGIN_CALC(std::string scriptSig, std::string script){
500     //先找到 OP_BEGIN_CALC 后面第一个空格的位置
```

```
501   std::string substr = "OP_BEGIN_CALC";
502   unsigned long long startPos = script.find(substr);
503   startPos += substr.length();
504   //再找到 OP_END_CALC 的位置
505   unsigned long long endPos = script.find("OP_END_CALC", startPos);
506   std::string infix_expression = script.substr(startPos + 1,endPos - startPos - 2);
507   //str.substr(a,b) 返回从 a 开始，长度为 b 的子串
508   int idx = 0;
509   char converted_infix[MAXSIZE];
510   //把 script 的子串 infix_expression 转换为 Infix_Calculation.cpp 能读取的形式
511   for (int i = 0 ; i < infix_expression.size() ; i++){
512       if (infix_expression[i] != ' '){
513           converted_infix[idx] = infix_expression[i];
514           idx++;
515       }
516   }
517   converted_infix[idx] = '=';
518   Mystack *OPND = InitStack();
519   char_stack *OPTR = InitCharStack();
520   int ans = InfixCalculation(converted_infix, OPND, OPTR);
521   SmartContract.push(std::to_string(ans));//把中缀表达式运算结果压入栈中
522
523
524 id OP_DUP(){//复制（非弹出）栈顶元素的值，并将复制的结果压入栈中
525   std::string sig = SmartContract.top();//先将之前压入的 input 的 scriptSig 取出
526   std::string signature;
527   std::string public_key;
528   //将 scriptSig 分解为 signature 和 public key 两部分
529   unsigned long long pos = sig.find(' ');
530   if (pos != std::string::npos){
531   //如果没有返回一个无效的位置，即 sig 中存在空格（例如: 22308B989987AEE01DED1BAC4C84497F 20221009）
532       signature = sig.substr(0,pos);//22308B989987AEE01DED1BAC4C84497F 是 signature
533       public_key = sig.substr(pos + 1, sig.length());//20221009 是 public key
534       SmartContract.push(signature);//将 signature 压入栈中
535       SmartContract.push(public_key);//将 public_key 压入栈中
536       SmartContract.push(public_key);//将 public_key 再压入栈中（相当于复制）
537   }
538   else{
539   //如果返回一个无效的位置（scriptSig 中根本没有空格，例如 2023）
540       SmartContract.push(sig);
541       SmartContract.push(sig);//DUP 指令将 sig 复制了一遍再压进栈
542   }
```

```
543
544
545 id OP_EQUALVERIFY(){//弹出栈顶两个操作数，判断它们是否相等
546   std::string top1 = SmartContract.top();
547   SmartContract.pop();
548   std::string top2 = SmartContract.top();
549   SmartContract.pop();
550   if (top1 == top2) flag = 1;
551   else flag = 0;
552
553
554 id OP_CHECKSIG(){
555   std::string public_key = SmartContract.top();
556   SmartContract.pop();
557   std::string signature = SmartContract.top();
558   SmartContract.pop();
559   BigInt MySignature(signature);
560
561   Rsa rsaSign; //rsa 对象，用于签名
562   Rsa rsaVerify; //rsa 对象，用于验证签名
563   rsaVerify.init(0); //初始化
564   rsaSign.setPu(std::stoi(public_key));
565   flag = verify(rsaVerify, MySignature, public_key);
566
567
568 id Execution(std::string keyword, std::string script, std::string scriptSig){
569   if (keyword == "OP_HASH160"){
570       OP_HASH160(script);
571   }
572   else if (keyword == "OP_BEGIN_CALC"){
573       OP_BEGIN_CALC(scriptSig, script);
574   }
575   else if (keyword == "OP_DUP"){
576       OP_DUP();
577   }
578   else if (keyword == "OP_EQUALVERIFY"){
579       OP_EQUALVERIFY();
580   }
581   else if (keyword == "OP_CHECKSIG"){
582       OP_CHECKSIG();
583   }
584   else{
```

```
585   }
586
587
588 ol CheckIOScript(std::string scriptSig, std::string script){
589   keywords.push_back("OP_HASH160");
590   keywords.push_back("OP_BEGIN_CALC");
591   keywords.push_back("OP_DUP");
592   keywords.push_back("OP_EQUALVERIFY");
593   keywords.push_back("OP_CHECKSIG");
594   SmartContract.push(scriptSig);
595   size_t pos = 0;
596   flag = 1;
597   while (pos < script.length()) {
598       std::string keyword;
599       size_t spacePos = script.find(' ', pos);
600       if (spacePos != std::string::npos) {
601           keyword = script.substr(pos, spacePos - pos);
602       }
603       else {
604           keyword = script.substr(pos);
605       }
606       Execution(keyword, script, scriptSig);
607
608       if (keyword == "OP_BEGIN_CALC") {
609           size_t endCalcPos = script.find("OP_END_CALC", pos);
610           if (endCalcPos != std::string::npos) {
611               pos = endCalcPos + sizeof("OP_END_CALC") - 1;
612           }
613           else {
614               std::cout << "OP_BEGIN_CALC found, but OP_END_CALC is missing" << std::endl;
615               break;
616           }
617       }
618       else {
619           pos = spacePos != std::string::npos ? spacePos + 1 : script.length();
620       }
621   }
622   if (flag == 1) return true;
623   else return false;
624
625
626 id CheckIOScript_Test(){
```

```
627   std::string scriptSig = "2023";
628   std::string script = "OP_DUP OP_BEGIN_CALC ( 100 * 10 + 24 ) / ( 2 + 2 ) + 1767 OP_END_CALC OP_EQUALV
629   std::cout << CheckIOScript(scriptSig, script) << std::endl;
630
631
632
633
634 t main(){
635   /*clock_t startTime = clock();//计时开始
636   block* firstBlock = InitBlockChain();
637   clock_t endTime = clock();//计时结束
638   std::cout << "Time spent reading data: " <<(double)(endTime - startTime) / CLOCKS_PER_SEC << "s" << s
639
640   std::cout << "Block count: " << BlocksLength(firstBlock) << std::endl;
641   CheckValidTransaction(firstBlock);
642   std::cout << std::endl;
643   int height;
644   std::cout << "Please input block's height: " << std::endl;
645   std::cin >> height;
646   std::cout << std::endl;
647   BlockInfo(height, firstBlock);
648
649   std::string txid;
650   std::cout << "Please input transaction's txid: " << std::endl;
651   std::cin >> txid;
652   std::cout << std::endl;
653   TransactionInfo(txid, firstBlock, nullptr);
654
655   return 0;*/
656   CheckIOScript_Test();
657
658
```