作业5  
1131190111-唐川淇

注：请本周日24:00前提交至钉钉群相应文件夹中

任务：村庄里住着ABCD四个人，他们每天将各自的转账和存款信息打包成区块：

1. ABCD四人每人拥有100元；
2. A向B支付了20元购买了一些牛肉，B向C支付了5元购买了一桶矿泉水；
3. C向A支付了10元用于理发，A向B支付了2元购买了一些芹菜；
4. B向C支付了8元购买了一瓶啤酒，D向A支付了10元用于理发 … …

示意图如下。请编写一个简单的区块链记账程序，注意这里采用的是地址Addr而不是哈希值Hash，因而实际完成的效果类似于一个反向的链表。

当前Addr

当前Addr

当前Addr

当前Addr

B-08 C+08

D-10 A+10

A 98

B 109

C 103

D 90

C-10 A+10

A-02 B+02

A 88

B 117

C 95

D 100

A-20 B+20

B-05 C+05

A 80

B 115

C 105

D 100

A 100

B 100

C 100

D 100

前块Addr

前块Addr

前块Addr

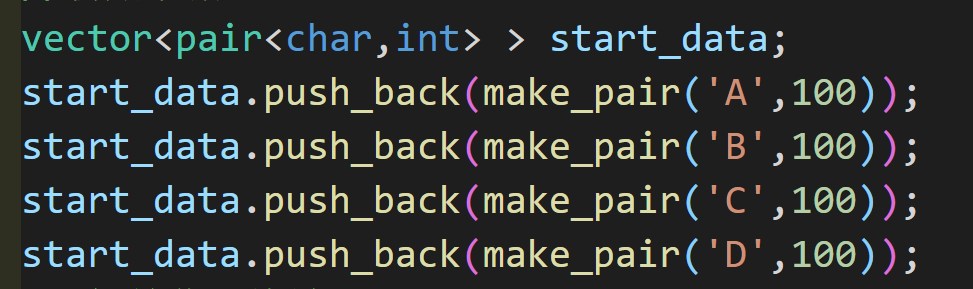
-

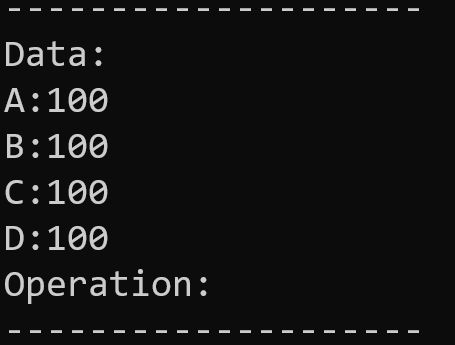
要求：

1. 程序的每个函数独立，有新增记录、打包区块等功能
2. 可以打印记账流水，也就是，可以按时间顺序或逆序打印账本的交易和余额信息
3. 请将代码粘贴在下方，并将运行结果截图

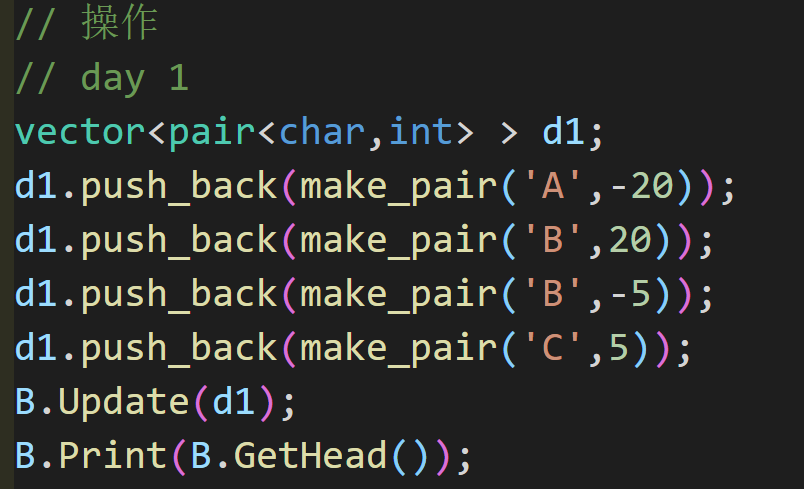
|  |
| --- |
| Exp6.cpp |
| #include<iostream>  #include<vector>  using namespace std;  struct Node{  vector<pair<char,int> > Data; // 数据域  vector<pair<char,int> > Operation; // 操作域  Node\* next;  };  class BlockChain{  private:  Node\* head;  public:  BlockChain(vector<pair<char,int> > N); // 初始化区块链  void Update(vector<pair<char,int> > N); // 插入区块  Node\* GetHead(); // 得到头区块  void Print(Node\* temp); // 打印单个区块  void Print\_All(); // 打印全部区块  };  int main(){  //初始数据  vector<pair<char,int> > start\_data;  start\_data.push\_back(make\_pair('A',100));  start\_data.push\_back(make\_pair('B',100));  start\_data.push\_back(make\_pair('C',100));  start\_data.push\_back(make\_pair('D',100));  // 初始化区块链  BlockChain B(start\_data);  B.Print(B.GetHead());  // 操作  // day 1  vector<pair<char,int> > d1;  d1.push\_back(make\_pair('A',-20));  d1.push\_back(make\_pair('B',20));  d1.push\_back(make\_pair('B',-5));  d1.push\_back(make\_pair('C',5));  B.Update(d1);  B.Print(B.GetHead());  //B.Print\_All();  // day 2  vector<pair<char,int> > d2;  d2.push\_back(make\_pair('C',-10));  d2.push\_back(make\_pair('A',10));  d2.push\_back(make\_pair('A',-2));  d2.push\_back(make\_pair('B',2));  B.Update(d2);  B.Print(B.GetHead());  // day 3  vector<pair<char,int> > d3;  d3.push\_back(make\_pair('B',-8));  d3.push\_back(make\_pair('C',8));  d3.push\_back(make\_pair('D',-10));  d3.push\_back(make\_pair('A',10));  B.Update(d3);  B.Print(B.GetHead());  // 打印所有区块  B.Print\_All();  }  BlockChain::BlockChain(vector<pair<char,int> > N){  head=new Node; // 初始化头节点  int s=N.size();  for(int i=0;i<s;i++)  head->Data.push\_back(N[i]);  head->next=NULL;  }  void BlockChain::Update(vector<pair<char,int> > N){  Node\* temp=new Node;// 创建新区块  temp->next=head;  for(int i=0;i<N.size();i++){  temp->Data.push\_back(head->Data[i]);  }  for(int i=0;i<N.size();i++){  temp->Operation.push\_back(N[i]);  char op=temp->Operation[i].first;  int change\_num=temp->Operation[i].second;  temp->Data[op-'A'].second+=change\_num;  }  head=temp;  }  void BlockChain::Print(Node\* temp){  cout<<"--------------------"<<endl;  cout<<"Data:"<<endl; // 数据域  for(int i=0;i<temp->Data.size();i++){  cout<<temp->Data[i].first<<":"<<temp->Data[i].second<<endl;  }  cout<<"Operation:"<<endl; // 操作域  for(int i=0;i<temp->Operation.size();i++){  cout<<temp->Operation[i].first<<":"<<temp->Operation[i].second<<endl;  }  cout<<"--------------------"<<endl;  }  void BlockChain::Print\_All(){  Node\* temp=head;  cout<<"All Blocks:"<<endl;  while(temp!=NULL){  Print(temp);  temp=temp->next;  }  }  Node\* BlockChain::GetHead(){  return head;  } |

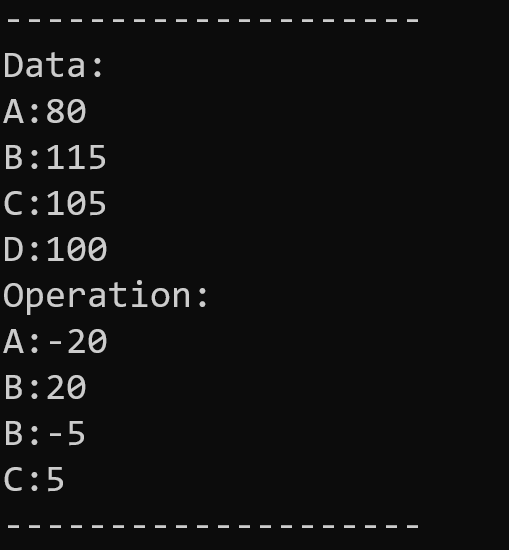
初始区块：



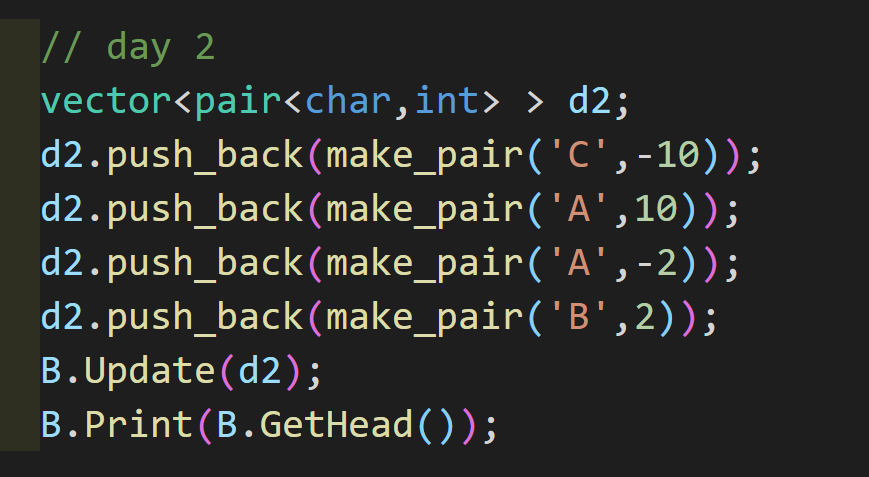


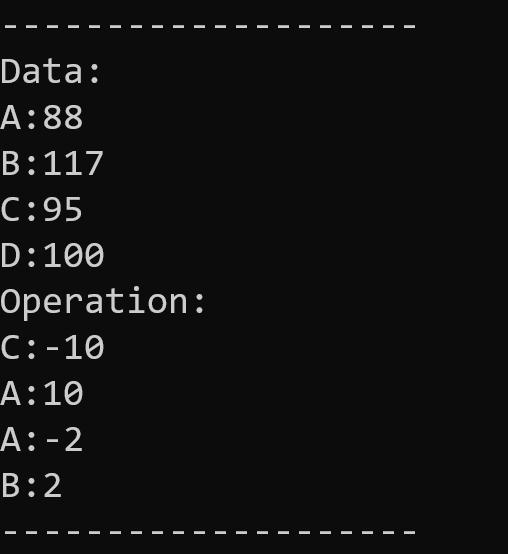
第一天操作：



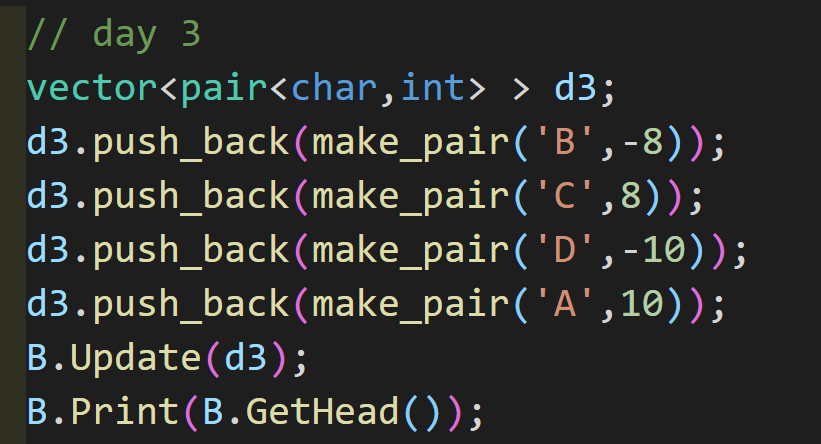


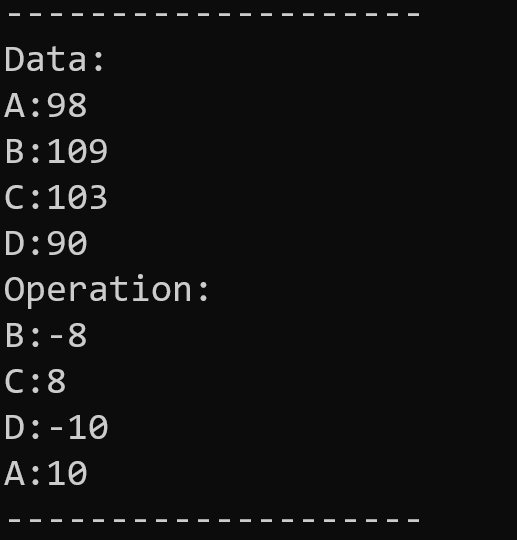
第二天：





第三天：





打印全部区块：

