Input 規則1.register一律使用R開頭的，例如R1,R11

2.label須單獨一行

3.輸入完畢後須輸入單獨一行-1

4.輸入類別後要空白一格，之後所有register要連續輸入用’,’隔開

5.只接受add addi beq 三種類型

Ex:

addi R1,R0,0

addi R2,R0,4

Loop:

beq R1,R2,End

addi R2,R2,-1

beq R0,R0,Loop

End:

-1

之後會詢問是需要多少bit的history predictor還有總共是多少entries

並在 ’:’ 之後輸入bit跟entries

Output會是 每個branch的第一行是對應到第幾個entry 後面會接這個branch的詳細code，第二行前面會是這個對應到的table的 history跟他的所有2BC，再來是他預測會taken(T)還是not taken(N)，跟實際他到底是taken(T)還是not taken(N)，最後面則是這個branch到目前為止錯誤預測了幾次

Ex:

entry: 3 beq R1,R2,End  
(00, SN, SN, SN, SN) N N misprediction: 0

簡略:

首先，先讀入所有的code到vector<string>code 裡面，並且針對label存放他在第幾行到map<string,int>label 裡面去，另外設了一個map<string,int>Reg當作register的容器來使用

讀取完所有code之後會先把所有register歸0，之後開始主體bitHistory

首先先詢問需求多少bit 的predictor，之後詢問多少entries，另外創建一個struct entries 類別來當作每個entries裡面會放些啥，例如他的2BC跟他的history還有miss的次數，並用一個vector<entries> entry來當作history table

之後分別分析每一行在vector<string> code裡面的code，且他們每行都用單一空格分別，所以透過stringstream分割前半部進去string s跟後半部分進去string s2，因為s只有add,addi,beq三種，而他們的格式都很像s2都是用兩個’,’分別的，所以用個string arr[3]分別存放兩個’,’分別出的三個部分

之後分別呼叫add addi beq的function來實際運作各個型別，其中beq的if裡面會先透過這是第k行code去%輸入的entriesNum(也就是輸入要多少entries)

之後會先輸出這個entry的其他資料(就是output裡面直到預測是T還是N之前所有資料)，並透過historyToBC function來找出現在所對應的的2BC得知目前預測是T還是N 並輸出預測的結果，之後根據beq function回傳的T或N來輸出真實會taken還是not taken並且判斷有沒有miss如果有就讓miss次數+1，如果實際是T就讓目前指向的2BC state上升(讓數字++)，但狀態已經在ST(BC=3)在++會超過，就讓BC=3(也就是保持在ST的狀態)，反之如果是N就讓state下降(讓數字--)，並確認狀態是不是再SN(BC=0)，再減下去就太小，所以讓BC=0(保持在SN的狀態)，並把這次實際結果push\_back到當前entries的history裡邊，最後再輸出這個entries總共mis了幾次

Void resetReg() 用來把所有register歸0

Void add(string arr[])用來算rs2+rs1放到rd裡面去

Void addi(string arr[])跟add同道理

Bool beq(string arr[])用來回傳rs1跟rs2有沒有equal  
int historyToBc(string s,int bitNum)用來根據bitNum找出現在history是要幾個bit並且回傳這個history目前指向第幾個2BC

Void outputEntries(entries\* ptr, string s, int howManyBit)用來輸出output裡面再預測結果前的所有需要輸出的東西

剩餘詳細請見code的註解