Code:-

```
#include <iostream>
#include <cctype>
#include <regex>
#include<map>
#include <stack>
#include<string.h>
#include <ctype.h>
using namespace std;
int fg=0;
int ghgh=0;
//this will make the conversion from infix to postfix
string postfix(string pp){
   map<char,int> preced;
   preced['*']=2;
   preced['/']=2;
   preced['-']=1;
   preced['+']=1;
   stack<char> st;
   string fin;
   reverse(pp.begin(), pp.end());
   for(auto x:pp){
      if(x=='*'||x=='/'||x=='+'||x=='-'){
         if(st.empty()==true){
            st.push(x);
         }
         else{
            int a = preced[st.top()];
            int b = preced[x];
```

```
            if(a>b){
               fin=fin+st.top();
               st.pop();
               st.push(x);
            }
            else if(a<=b){
               st.push(x);
            }
          }

       }
       else{
       fin=fin+x;
       }
    }
    while(st.empty()==false){
    fin=fin+st.top();
    st.pop();
    }
    return fin;
}
//this will evaluate the string
//the method is if we get an operator we will pop the top two operator and do operation
int eval(string s,map<string,int> finall){
    stack<float> st;
    for(auto x:s){
        if(x=='*'){
            float a = st.top();
            st.pop();
```

```
    float b = st.top();
    st.pop();
    st.push(a*b);
 }
else if(x=='/'){
    float a = st.top();
    st.pop();
    float b = st.top();
    st.pop();
    st.push(a/b);
 }
else if(x=='-'){
    float a = st.top();
    st.pop();
    float b = st.top();
    st.pop();
    st.push(a-b);
 }
else if(x=='+'){
    float a = st.top();
    st.pop();
    float b = st.top();
    st.pop();
    st.push(a+b);
 }
else{
string p;
p.push_back(x);
if(isdigit(x)){
```

```
            st.push(stoi(p));
            continue;
        }
        st.push(finall[p]);
        }
    }
    return st.top();

}
//this is used to check the if statement
//we will pass to postfix then check
bool letmecheck(string a,map<string,int> finall){
    int aa=0;
    int bb=0;
    string ll;
    string pp;
    string op;
    bool flag = false;
    int i=0;
    for(auto l:a){
        if(l==' '){
          i++;
          if(i==1){
          pp=postfix(pp);
          aa = eval(pp,finall);
          pp="";
          continue;
          }
          if(i==2){
```

```
    if(pp==">"){
        op = "greater";
    }
    else if(pp=="<"){
        op = "less";
    }
    else if(pp==">="){
        op="greaterequal";
    }
    else if(pp=="<="){
        op="lessqual";
    }
    else if(pp=="="){
        op="equal";
    }
    else if(pp=="!"){
        op="not";
    }
    pp="";
    continue;
  }
  if(i==3){
  pp=postfix(pp);
  bb = eval(pp,finall);
  pp="";
  continue;
  }
}
else{
```

```
    pp = pp + l;
    }
}
 if(op=="greater"){
    if(aa>bb){
        flag=true;
    }
    return flag;
 }
 else if(op=="less"){
    if(aa<bb){
        flag=true;
    }
    return flag;
 }
 else if(op=="greaterequal"){
    if(aa>=bb){
        flag=true;
    }
    return flag;
 }
 else if(op=="lessqual"){
    if(aa<=bb){
        flag=true;
    }
    return flag;
 }
 else if(op=="equal"){
     if(aa==bb){
```

```
            flag=true;
        }
        return flag;
    }
    else if(op=="not"){
        if(aa!=bb){
            flag=true;
        }
        return flag;
    }
   return flag;
}
void parser(map<int,map<pair<int,string>,string>,less<int>> ques,int n,vector<int>
check,map<string,int> &finall,int argc, char** argv){
stack<string> ss;
int flag;
auto x = ques.begin();
for(;x!=ques.end();x++){
auto expr = x->second;
flag = x->first;
auto vt = expr.begin();
string pp;
for(;vt!=expr.end();vt++){
//first if it is let and it is initilizing
if(fg==1){
   break;
}
if((vt->first).second=="let"&&expr.size()==4){
for(int z=1;z<=4;z++){
```

```
if((vt->second)=="keyword"){
   vt++;
}
else if(vt->second=="identifier"){
   auto kk = (vt->first).second;
   ss.push(kk);
   vt++;
}
else if(vt->second=="equalsign"){
   vt++;
}
else if(vt->second=="value"){
   auto l = (vt->first).second;
   int pp=stoi(l);
   auto top = ss.top();
   ss.pop();
   finall[top]=pp;
}
}
}
//second if it is print
else if((vt->first).second=="print"){
   for(int z=1;z<=2;z++){
   if((vt->first).second=="print"){
   vt++;
   }
   else {
      if((vt->second)=="direct"){
         //cout<<"yaha pe toh nahi aagya???";
```

```
    cout<<(vt->first).second<<endl;
    continue;
}
if((vt->second)=="directto"){
    auto z = (vt->first).second;
    string mm;
    string a;
    string b;
    int flag=0;
    for(auto ll:z){
        if(ll=='*'){
            a=mm;
            if(finall.find(mm) != finall.end()){
            a=to_string(finall[mm]);
            }
            mm="";
            flag=1;
            continue;
        }
        else if(ll=='/'){
            a=mm;
            if(finall.find(mm) != finall.end()){
            a=to_string(finall[mm]);
            }
            mm="";
            flag=2;
            continue;
        }
        else if(ll=='+'){
```

```
        a=mm;
        if(finall.find(mm) != finall.end()){
        a=to_string(finall[mm]);
        }
        mm="";
        flag=3;
        continue;
    }
    else if(ll=='-'){
        a=mm;
        if(finall.find(mm) != finall.end()){
        a=to_string(finall[mm]);
        }
        mm="";https://www.onlinegdb.com/#tab-stdout
        flag=4;
        continue;
    }
   mm=mm+ll;
}
b=mm;
if(flag==1){
    int xx=stoi(a);
    int yy=stoi(b);
    cout<<(xx*yy)<<endl;
}
if(flag==2){
    int xx=stoi(a);
    int yy=stoi(b);
    cout<<(xx/yy)<<endl;
```

```
        }
        if(flag==3){
            int xx=stoi(a);
            int yy=stoi(b);
            cout<<(xx+yy)<<endl;
        }
        if(flag==3){
            int xx=stoi(a);
            int yy=stoi(b);
            cout<<(xx-yy)<<endl;
        }
        continue;
      }
      else{
      cout<<finall[(vt->first).second]<<endl;
      }
   }
   }
}
//third if it is let but it is for evaluation
else if((vt->first).second=="let"&&expr.size()>4){
for(int z=1;z<=expr.size();z++){
if((vt->second)=="keyword"){
   vt++;
}
else if(vt->second=="identifier"&&z<=2){
   auto kk = (vt->first).second;
   ss.push(kk);
   vt++;
```

```
}
else if(vt->second=="equalsign"){
   vt++;
}
else{
  pp=pp+(vt->first).second;
  if(z==expr.size()){
     pp=postfix(pp);
     int kl = eval(pp,finall);
     auto top = ss.top();
     ss.pop();
     finall[top]=kl;
     continue;
  }
  vt++;
}
}
}
//fourth if it is if stattemtn
else if((vt->first).second=="if"){
   int lol = 0;
   string ap;
   int dd=0;
   int zt = 0;
   while(dd<=(expr.size())){
      dd++;
      if((vt->first).second=="if"){
         vt++;
         continue;
```

```
}
else if(vt->second!="comp"&&zt==0&&(vt->first).second!="then"){
    ap=ap+(vt->first).second;
    vt++;
    continue;
}
else if(vt->second=="comp"){
    ap=ap+" ";
    ap=ap+(vt->first).second;
    ap=ap+" ";
    vt++;
    continue;
}
else if((vt->first).second=="then"){
   zt=1;
   ap=ap+" ";
   if(letmecheck(ap,finall)==true){
    lol=1;
   }
   else{
      lol=0;
   }
   vt++;
   continue;
}
else if((vt->first).second=="print"&&lol==1){
for(int z=1;z<=2;z++){
if((vt->first).second=="print"){
vt++;
```

```
    }
    else {
    if((vt->second)=="direct"){
       //cout<<"isit";
       cout<<(vt->first).second<<endl;
       vt++;
       continue;
    }
    else{
    cout<<finall[(vt->first).second]<<endl;
    vt++;
    continue;
    }
    }
    }
    }
else if((vt->first).second=="goto"&&lol==1){
int dd=0;
int ff = 0;
while(dd<=2){
    dd++;
    if((vt->first).second=="goto"){
       vt++;
       continue;
    }
    if(vt->second=="value"){
       auto zzz= (vt->first).second;
       auto xxx = ques.begin();
       auto zll=xxx;
```

```
        for(;xxx!=ques.end();xxx++){
            auto eee = xxx->second;
            auto at = eee.begin();
           if(xxx->first==stoi(zzz)){
              x=zll;
              vt++;
              break;
           }
           zll = xxx;
           }
           }
       }
    }
       vt++;

}
}
//fifth if it is directly an goto statement
else if((vt->first).second=="goto"){
    int dd=0;
    int ff = 0;
    while(dd<=2){
       dd++;
       if((vt->first).second=="goto"){
          vt++;
          continue;
       }
       if(vt->second=="value"){
          auto zzz= (vt->first).second;
```

```
            auto xxx = ques.begin();
            auto zll=xxx;
          for(;xxx!=ques.end();xxx++){
             auto eee = xxx->second;
             auto at = eee.begin();
            if(xxx->first==stoi(zzz)){
               x=zll;
               vt++;
               break;
            }
            zll = xxx;
            }
            }
        }
}
//sixth(initilize x,y,z)
else if((vt->first).second=="integer"){
    int dd=0;
    while(dd<=expr.size()){
        dd++;
        if((vt->first).second=="integer"){
            vt++;
            continue;
        }
        else if(vt->second=="value"){
            finall[(vt->first).second];
        }
    }
}
```

```
//seventh
else if((vt->first).second=="input"){
   int dd=0;
   while(dd<=expr.size()){
      dd++;
      if((vt->first).second=="input"){
         vt++;
         continue;
      }
      else if(vt->second=="identifier"){
         ghgh++;
         finall[(vt->first).second]=stoi(argv[ghgh]);
         vt++;
      }

   }
}

}

}
}
int main(int argc, char** argv) {
   map<int,map<pair<int,string>,string>,less<int>> ques;//this will map the line number with the lexical analysis of that line
   //map<pair<int,string>,string> expr;
   map<int,string,less<int>> help;// this is basically used for storing the line number and the string
   int ll;
   string s;
```

```
string para;
vector<int> check;
map<string,int> finall;
//here i am storing in memory
while(getline(cin,s)){
   auto first_token = s.substr(0, s.find(' '));
   s=s.substr(s.find_first_of(" ")+1);// removing the line number from the string
   s=s+" "; // giving blank spaces
   para=para+s+" ";
   help[stoi(first_token)]=s;
}
string word;
int i=0;
int z=0;
string zzz;
// //basically here i am tokenizing for diffffernt tokenz
///lexer
for(auto lmn:help){
string paraa = lmn.second;
for(auto x:paraa){
   if(x==' '){
        i++;
        if((word.find('0') != string::npos || word.find('1') != string::npos ||word.find('2') != string::npos
||word.find('3') != string::npos ||word.find('4') != string::npos ||word.find('5') != string::npos
||word.find('6') != string::npos ||word.find('7') != string::npos ||word.find('8') != string::npos
||word.find('9') != std::string::npos)){
        if(zzz=="print"){
        ques[lmn.first].insert(make_pair(make_pair(i,word),"directto"));
        }
        }
```

```
        if(word=="integer"){
           ques[lmn.first].insert(make_pair(make_pair(i,word),"init"));
           zzz="init";
        }
        if(word=="input"){
           ques[lmn.first].insert(make_pair(make_pair(i,word),"input"));
           zzz="input";
        }
        if(word.find_first_not_of("0123456789") == string::npos){
        ques[lmn.first].insert(make_pair(make_pair(i,word),"value"));
        zzz="value";
        }
        else if(word=="if"){
        ques[lmn.first].insert(make_pair(make_pair(i,word),"comparsion"));
        zzz="comparision";
        }
        else if(word=="goto"){
        ques[lmn.first].insert(make_pair(make_pair(i,word),"travel"));
        zzz="travel";
        }
        else if(word.find("\"")!=string::npos){
        ques[lmn.first].insert(make_pair(make_pair(i,word),"direct"));
        zzz="direct";
        }
        else
if(word=="LET"||word=="let"||word=="print"||word=="println"||word=="PRINT"||word=="PRINTLN"||
|word=="print"){
        if(word=="println"){
           word="print";
```

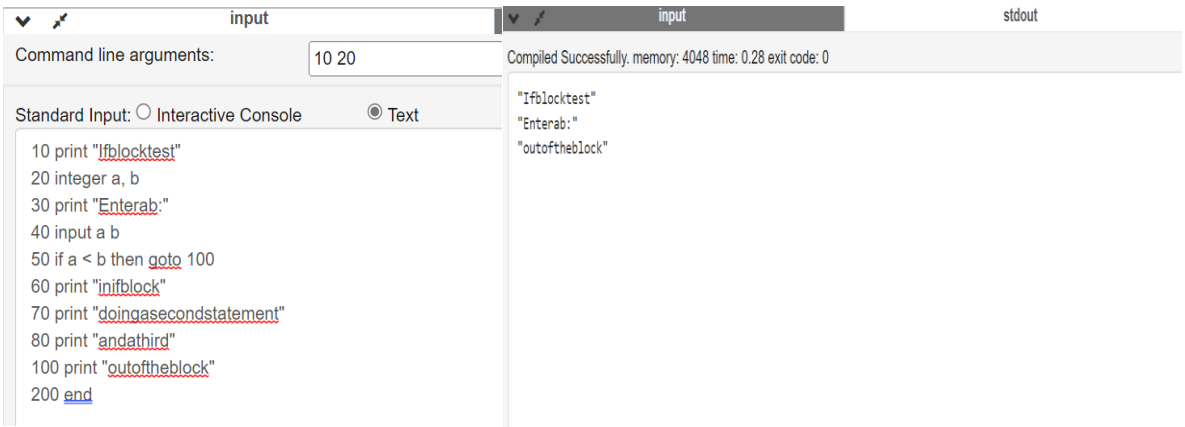
```
}
ques[lmn.first].insert(make_pair(make_pair(i,word),"keyword"));
check.push_back((i-z));
z=i;
ll=z;
if(word=="print"){
zzz="print";
}
else{
   zzz="random";
}
 }
else if(regex_match(word,regex("^[A-Za-z]+$"))&&word.length()==1){
ques[lmn.first].insert(make_pair(make_pair(i,word),"identifier"));
zzz="identifier";
 }
else if(word=="then"){
ques[lmn.first].insert(make_pair(make_pair(i,word),"then"));
zzz="then";
}
else if(word=="/"||word=="*"||word=="+"||word=="-"){
ques[lmn.first].insert(make_pair(make_pair(i,word),"operator"));
zzz="operator";
}
else if(word=="!"||word=="<"||word==">"){
ques[lmn.first].insert(make_pair(make_pair(i,word),"comp"));
zzz="comp";
}
else if(word=="="){
```

```
            ques[lmn.first].insert(make_pair(make_pair(i,word),"equalsign"));
            zzz="equalsign";
            }
            word="";
          continue;
       }
       word=word+x;
   }
   }
   check[0]=0;
   check.push_back((i-ll+1));
   i=i+1;
   //basically sending to parser which will take care of everything
   parser(ques,i,check,finall,argc,argv);
   return 0;
}
```

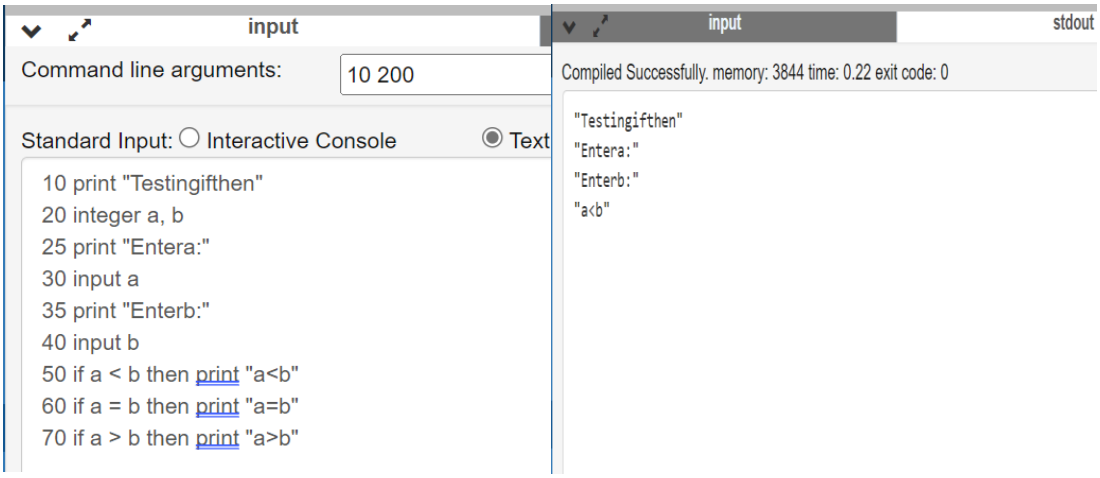
Sample I/O:-

(Inside print statement in my code having spaces will have problem and while taking input having commas will create problem so not using that )

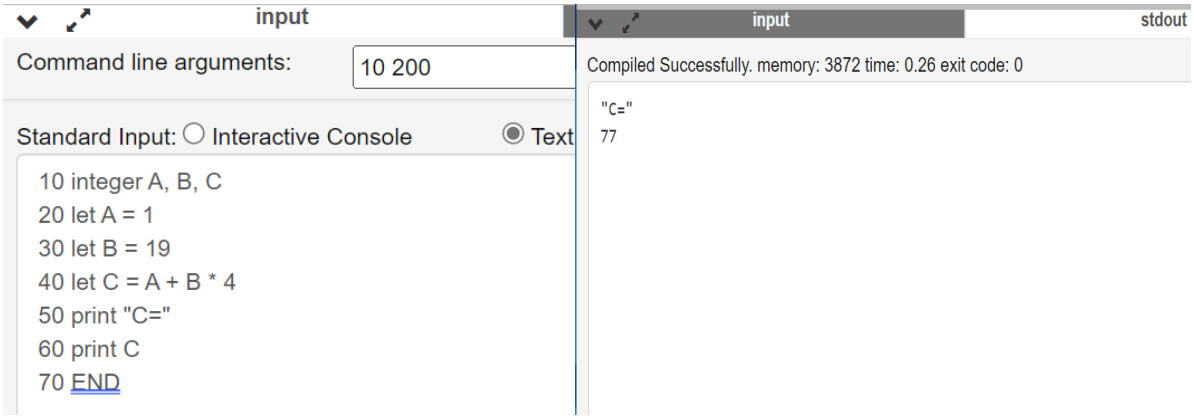
Input :- (here the input is given through command line)



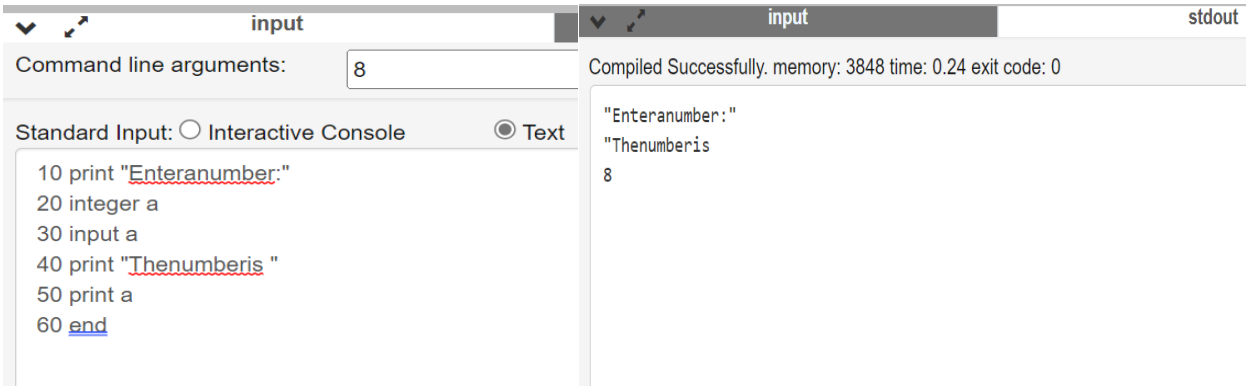
Input 2 :-



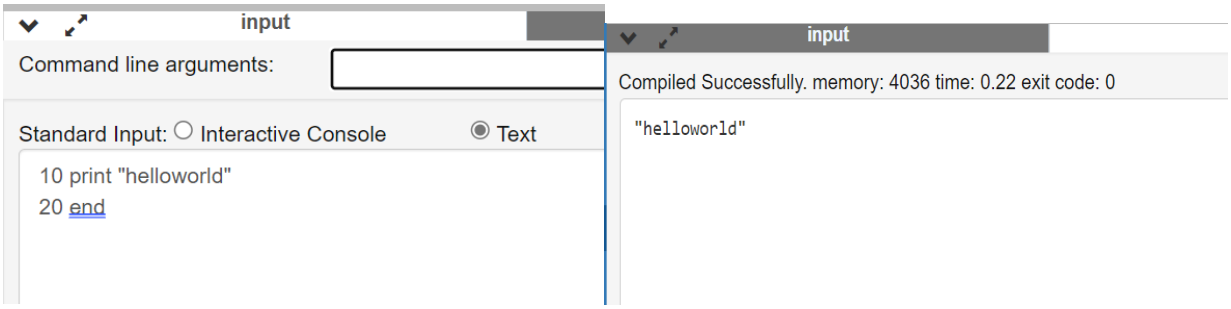

Input 3 :-



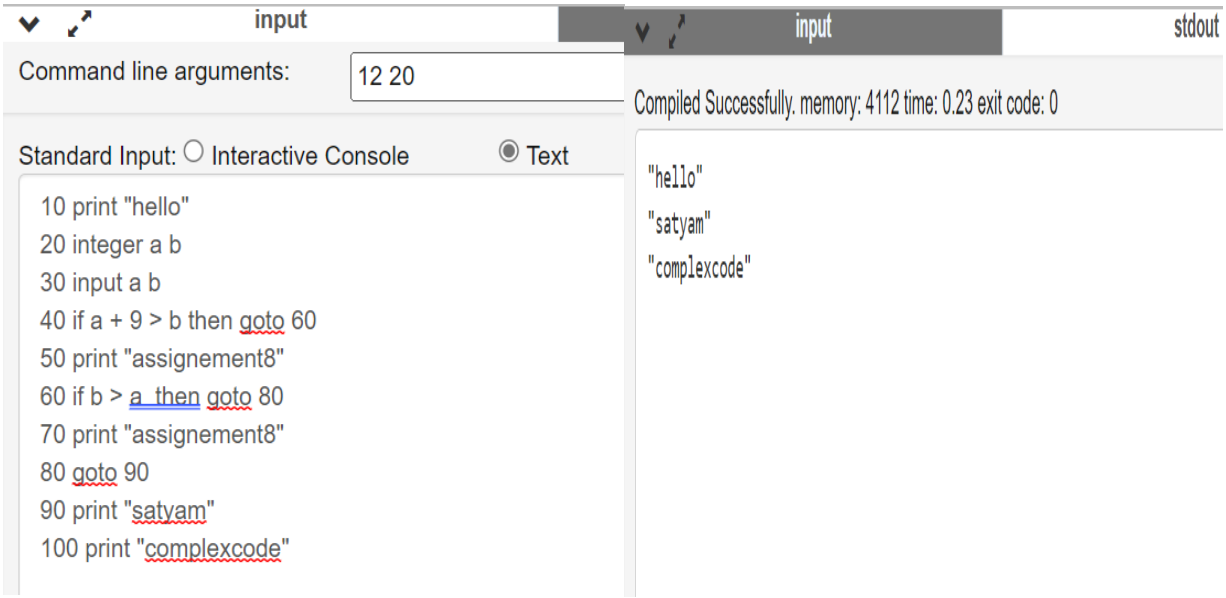

Input 4:-




Input 5:-




Input 6 :- (to show against complicated cases to show the efficiency and proper code)




My methodology :-

Firstly, I have stored the input in the memory and my memory is a map with key as integer and value as map<pair<int,string>,string>.

The map<pair<int,string>,string> is from my previous code (Assignment 7). Over here, inside the pair of key, the token number and the word is stored and they are mapped to whether that particular word is a keyword, variable or equalsign. All of this was done in the lexical analysis part .

For getting the input from the user I used command line.

The memory is then sent to the parser in which the syntax is checked and it is parsed only if the syntax is correct. In this block of code majority of the tasks are performed like goto, evaluation of expression and mapping of variables.

For goto, I searched my memory and changed the iterator in which the key is equal to int next to goto.

For if, I used the same expressions from my previous code and sent it to that postfix and evaluation was performed.