```
/*
  Author : Abhishek Yadav
  Roll NO: CS12B032

    this programs calculates the quotient of a division upto nth places of decimal
    So as a part of input you have to enter first numerator which can be anything(fraction, some
invalid string
    , which, after validation check will be rejected), and second input as denominator and third is the
number of places
    after decimal you want to print the result.
*/



#include<iostream>
#include<cmath>
using namespace std;
#include"double_linked_list.cpp"

void divide(string &str1,string &str2,unsigned long long n)
{
    int flag1=0,flag2=0,i=0,j=0;
    if(str1[0]=='-'){
        flag1++; i++;
      }
      if(str2[0]=='-'){
          flag2++;j++;
      }
    //counts how many digits are there in numerator after decimal
    int count_after_decimal_num=0;
    //counts digits after decimal in denominator
    int count_after_decimal_denom=0;
    int decimal_flag=0,k=0;
    int l1=str1.length();
     int temp=l1;
    l1--;

    unsigned long long num=0;
    unsigned long long denom=0;
    while(l1>=i){
        if(isdigit(str1[l1]))
            num+=pow(10,k++)*(str1[l1]-'0');
         else decimal_flag++;
         if(decimal_flag) count_after_decimal_num++;
         l1--;
        }
        if(count_after_decimal_num){
          if(str1[0]!='-')
          count_after_decimal_num=temp-count_after_decimal_num;
          else count_after_decimal_num=temp-count_after_decimal_num-1;
          }
```

```
  // cout <<num << " "<<count_after_decimal_num<<endl;
    l1=str2.length();
    temp=l1;
    l1--;k=0;
    decimal_flag=0;
    while(l1>=j){
        if(isdigit(str2[l1]))
            denom+=pow(10,k++)*(str2[l1]-'0');
         else decimal_flag++;
         if(decimal_flag) count_after_decimal_denom++;
         l1--;
        }
if(count_after_decimal_denom){
              if(str2[0]!='-')
         count_after_decimal_denom=temp-count_after_decimal_denom;
         else count_after_decimal_denom=temp-count_after_decimal_denom-1;
    //cout <<denom << " "<<count_after_decimal_denom<<endl;
    }
   if(denom==0){

   cout <<"infinity\n";
   return ;}
   else if(num==0){
   cout <<num <<endl;return;}
    if(num==denom){
         cout << num/denom <<" " <<endl;
         return;
      }

           temp=num;
           int temp_=denom;
           unsigned long long counter=0;
           int count_upto_decimal=0;
           int count_num=0;
            list* div_list=NULL;


           if(temp<temp_){
              while(temp<temp_)
              {
                temp*=10;
                count_num++;
              }
           }
           if(temp>temp_){
                int local=temp/temp_;
               while(local>0){
                 div_list=add_list_in_reverse(div_list,local%10);
                   count_upto_decimal++;
                   local/=10;
                 }
```

```
            temp%=temp_;
          }

        while(temp<temp_){
          if(counter==n|| temp==0)
           break;
            int flag=0;
              while(temp<temp_){
                temp*=10;
                if(flag>=1)
                  div_list=add_node_in_list(div_list,0);
              flag++;
           }
           div_list=add_node_in_list(div_list,temp/temp_);
           temp%=temp_;
           counter++;

            }

            if(flag1^flag2)
            cout <<"-";
           int temp3= count_after_decimal_denom - count_after_decimal_num - count_num +
count_upto_decimal;
           // cout <<temp3  <<" " <<count_after_decimal_denom <<"
"<<count_after_decimal_num
            //<<" " << count_num <<" " <<count_upto_decimal <<endl;
            if(temp3<0){
            cout <<".";
                while(temp3<0){
                cout <<"0" ;
                temp3+=1;
                }
                 list *flag_list=div_list;
           while(flag_list!=NULL){
           cout <<flag_list->data;
           flag_list=flag_list->rear;
           }

         }else {
           list *flag_list=div_list;
           while(flag_list!=NULL){
           if(temp3==0)
           cout <<".";
           cout <<flag_list->data;
           flag_list=flag_list->rear;
           --temp3;
           }

         }
           cout <<endl;
           return ;
 }
```

```
 bool verify_num(string &str)
 {
    int l=str.length();
    int flag=0;
     int i=0;
     if(str[i]=='-')
     i++;
    while(i<l){
      if(isdigit(str[i])){i++;continue;}
         else if(str[i]=='.' && flag==0)
            flag++;
         else break;
        i++;
     }

     if(i<l) return false ;
     else return true;

 }
int main()
{
   string str1,str2;
    cout <<"enter Numerator : ";
    getline(cin,str1);
    if(!verify_num(str1)){
    cout <<"Input not consistent:\n";
     return 0;
     }
    cout <<"Enter denominator: ";
    getline(cin,str2);
    if(!verify_num(str2)){
    cout <<"Input not consistent:\n";
     return 0;
     }

    cout <<"After Decimal, upto how many digits do you want the result:? \n";
    unsigned long long n;
    cin >>n;
     cout <<"result of division is: \n";
   divide(str1,str2,n);
   return 0;
}

// Library implementation of double linked list used for this program


// double_linkedList.cpp

using namespace std;
typedef struct linked_list list;
struct linked_list{
       list* rear;
```

```
      int data;
      list* front;
};
typedef struct list_ptr ptrs;
struct list_ptr{
   list* lst1;
   list* lst2;
};


list* list_new(int num){
   list *new_node=new list;
    new_node->rear=NULL;
    new_node->data=num;
    new_node->front=NULL;
   return new_node;
}

list *add_node_in_list(list*l,int data_element){
    list*flag_node=l;
    list*new_node=list_new(data_element);
        if(l==NULL)
      l=new_node;
    else{
        while(flag_node->rear!=NULL)
        flag_node=flag_node->rear;

        new_node->front=flag_node;
        flag_node->rear=new_node;
    }
    return l;
}

list * add_list_in_reverse(list* l,int data_element)
{
   list*flag_node=l;
    list *new_node=new list;
    new_node->rear=NULL;
    new_node->data=data_element;
    new_node->front=NULL;
    if(l!=NULL){
        new_node->rear=flag_node;
        flag_node->front=new_node;
        l=new_node;
    }
    else if(l==NULL)
      l=new_node;
      return l;

}

int list_size(list*l)
```

```
{
  list*temp_list=l;
  int counter=0;
  while(temp_list!=NULL)
  {
     temp_list=temp_list->rear;
     counter++;
  }
  return counter;
}
list* string_to_int(string& str)
{
  int l=str.length();
  list* long_num1=NULL ;
  int p,i=0;
  while(str[i]=='0' && i<l)
  i++;
  if(i==l){
  long_num1=add_node_in_list(long_num1,0);
  return long_num1;
  }else{
  while(i<l){
    if(isdigit(str[i])){
       p=str[i++]-'0';
       long_num1=add_list_in_reverse(long_num1,p);
       }else if(str[0]=='-')
           i++;
       else break;
  }
 }
  if(i<l)
  return NULL;
  else return long_num1;

}
list**string_to_int_for_division(string &str){
         int len=str.length();
  list** l=new list* [2];
  l[0]=NULL;
  l[1]=NULL;
  int p,i=0;
  while(str[i]=='0' && i<len)
  i++;

  if(i==len){
   l[0]=add_node_in_list(l[0],0);
   l[1]=add_list_in_reverse(l[1],0);
   return l;
  }else{
  int temp=len-1;
  int q;
  while(i<len){
```

```
    if(isdigit(str[i])){
       p=str[i++]-'0';
       q=str[temp--]-'0';
        l[0]=add_node_in_list(l[0],p);
        l[1]=add_list_in_reverse(l[1],q);
       }else if(str[0]=='-')
           i++;
       else break;
  }
 }
  if(i<len)
  return NULL;
  else return l;

}
```