/\*

//F1 Matrix

#include <iostream>

using namespace std;

class Matrix{

private:

double\*\* p;

public:

Matrix()

{

p=new double\*[3];

for(int i=0;i<3;i++)

p[i]=new double[3];

}

~Matrix(){}

void setMatrix()

{

for(int i=0;i<3;++i)

{

for(int j=0;j<3;j++)

{

cin>>p[i][j];

}

}

}

Matrix mutipleMatrix(Matrix b)

{

Matrix a;

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)//由于此处矩阵乘法不长，手打代码

{

a.p[i][j]=p[i][0]\*b.p[0][j]+p[i][1]\*b.p[1][j]+p[i][2]\*b.p[2][j];

}

}

return a;

}

void printMatrix()

{

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

{

cout<<p[i][j]<<" ";

}

cout<<endl;

}

}

};

int main()

{

Matrix a,b,c;

a.setMatrix();

b.setMatrix();

c=a.mutipleMatrix(b);

c.printMatrix();

return 0;

}\*/

/\*

//F2 【第八章】虚函数-2 知识点稍微有点多

//vector类中每次执行push\_back操作，相当于底层的数组实现要重新分配大小（即先free掉原来的存储，后重新malloc）

//Sort函数，模板Sort(start,end,排序方法),第三个也可无，但默认为从小到大排序,仅限于一般数字情况

//用途：排序

//前两个参数，指定要排序的数组范围

//第三个参数，为true则邻近前后不交换，为false则执行前后交换

//关于for(auto x:数组) 执行内容

//表示从数组首位开始循环执行所要执行的内容到数组末位

//关于#include<bits/stdc++.h>

//它是C++中支持的一个几乎万能的头文件，几乎包含所有的可用到的C++库函数。

//写代码可以直接引用这一个头文件了，不需要在写一大堆vector、string、map、stack……

//优点

//减少手打头文件代码时间，几乎包含了c++所以头文件

//缺点

//不属于GNU C++库的标准头文件，在部分情况下可能会失败

//使用它将包含许多不必要的东西，并增加编译时间

//这个头文件不是C++标准的一部分，因此是不可移植的，应该避免

//编译器每次编译翻译单元时都必须实际读取和分析每个包含的头文件，应该减少这类头文件的使用

#include<bits/stdc++.h>

using namespace std;

class Clock;

bool cmp(Clock \*a,Clock \*b);

//------------------------------------------code here------------------------------------------------

class Clock

{

public:

Clock(){}

~Clock(){}

virtual void print()=0;

virtual int time()=0;

};

class ClockA:public Clock

{

int hour,minute,second;

public:

ClockA(int a,int b,int c):hour(a),minute(b),second(c){}

~ClockA(){}

void print();

int time()

{

return hour\*3600+minute\*60+second;

}

};

class ClockB:public Clock

{

int quarter,second;

public:

ClockB(int a,int b):quarter(a),second(b){}

~ClockB(){}

void print();

int time()

{

return quarter\*15\*60+second;

}

};

bool cmp(Clock \*a,Clock \*b)

{

if(a->time()<b->time())

return true;

else return false;

}

//--------------------------------------------------------------------------------------------------------

void ClockA::print(){cout<<hour<<':'<<minute<<':'<<second<<'\n';}

void ClockB::print(){cout<<quarter<<':'<<second<<'\n';}

int main()

{

vector<Clock\*> v;

int n,m,a,b,c;

Clock\* tim;

cin>>n>>m;

while(n--)

{

cin>>a>>b>>c;

tim=new ClockA(a,b,c);//abc代表时分秒

v.push\_back(tim);

}

while(m--)

{

cin>>a>>b;

tim=new ClockB(a,b);//a,b代表 刻，秒

v.push\_back(tim);

}

sort(v.begin(),v.end(),cmp);

for(auto x:v)x->print();

}\*/

/\*

//F3 虚析构函数

#include<bits/stdc++.h>

using namespace std;

class A;

A \*data[1001];

int len=0;

//-------------------code here--------------------

class A

{

public:

A(){}

virtual ~A();

};

class B:public A

{

public:

B(){}

~B();

};

A::~A()

{

cout<<"del A"<<endl;

data[++len]=new B;

}

B::~B()

{

cout<<"del B"<<endl;

++len;

}

//----------------------------------------------------

int main()

{

data[++len]=new A;

int n;

cin>>n;

for(int i=1;i<=n;i++) delete(data[i]);

}\*/

/\*

//F4 【第八章】虚函数-1

#include<bits/stdc++.h>

using namespace std;

//------------------------------code here----------------------------

class Shape

{

public:

Shape(){}

~Shape(){}

virtual int getArea()=0;

virtual int getPerim()=0;

};

class Rectangle:public Shape

{

int x,y;

public:

Rectangle(int x,int y):x(x),y(y){}

~Rectangle(){}

int getArea()

{

return x\*y;

}

int getPerim()

{

return 2\*(x+y);

}

};

class Circle:public Shape

{

int r;

public:

Circle(int r):r(r){}

~Circle(){}

int getArea()

{

return 3\*r\*r;

}

int getPerim()

{

return 2\*3\*r;

}

};

//-----------------------------------------------------------------------

int main()

{

int x,y,r;

cin>>x>>y>>r;

Shape \*shape=new Rectangle(x,y);

cout<<shape->getArea()<<'-'<<shape->getPerim()<<" ";

shape=new Circle(r);

cout<<shape->getArea()<<'-'<<shape->getPerim()<<" ";

}\*/

/\*

//F5 【第八章】运算符重载-1

#include<bits/stdc++.h>

using namespace std;

class Counter

{

public:

Counter(int d){x=d;}

int getx() const{return x;}

private:

int x;

};

//------------------code here----------------------------

int operator+(Counter x,int y)

{

return x.getx()+y;

}

int operator+(int y,Counter x)

{

return x.getx()+y;

}

//-------------------------------------------------------------

int main()

{

int a,y;

cin>>a>>y;

Counter x(a);

cout<<x+y<<' '<<y+x;

}\*/

/\*

//F6 【第八章】运算符重载-2

#include<bits/stdc++.h>

using namespace std;

class Counter

{

public:

Counter(){x=0;}

int getx() const{return x;}

//----------------------------code-----------------------------------------

Counter operator++()//++后无参数，为++a形式，下列同理

{

x++;

return \*this;

}

Counter operator--()

{

x--;

return \*this;

}

Counter operator++(int)

{

Counter temp=\*this;

x++;

return temp;

}

Counter operator--(int)

{

Counter temp=\*this;

x--;

return temp;

}

//----------------------------------------------------------------------------

private:

int x;

};

vector<int> ans;

Counter a;

int x,y;

int main()

{

srand(time(0));

y=x=rand();

while(x--)a++;

if(a.getx()!=y)ans.push\_back(1);

if((++a).getx()!=++y)ans.push\_back(2);

if((--a).getx()!=--y)ans.push\_back(3);

if((a++).getx()!=y++)ans.push\_back(4);

if((a--).getx()!=y--)ans.push\_back(5);

if(a.getx()!=y)ans.push\_back(6);

if(ans.size()==0)cout<<"ACCEPT";

else for(auto u:ans)cout<<"wrong answer on test "<<u<<endl;

}\*/

/\*

//F7 第四章 DataType

//本题本人不清楚是否是这么理解 硬交上去了

#include<iostream>

using namespace std;

class DataType

{

private:

enum TYPE { INT, CHAR, FLOAT };//已知所处理类型只有3种可能，运用枚举类型

//也可不用枚举类型

union DATA

{

int i;

char c;

double f;

};

TYPE type;

DATA data;

public:

DataType(int i)

{

type=TYPE(INT);

data.i=i;

}

DataType(char c)

{

type=TYPE(CHAR);

data.c=c;

}

DataType(double f)

{

type=TYPE(FLOAT);

data.f=f;

}

void Show()

{

if(type==0)

{

cout<<"211"<<endl;

}

else if(type==1)

{

cout<<"a"<<endl;

}

else if(type==2)

{

cout<<"196.101"<<endl;

}

}

};

int main()

{

DataType a('c');

a.Show();

DataType b(960);

b.Show();

DataType c(265.12);

c.Show();

return 0;

}\*/

/\*

//F8 第四章 Tree类

#include<iostream>

using namespace std;

class Tree{

private:

int ages=0;

public:

Tree(){}

~Tree(){}

void grow(int years)

{

ages=ages+years;

}

int age()

{

return ages;

}

};

int main()

{

int x;

Tree tree;

while(cin>>x)

{

if(x==-1)

break;

tree.grow(x);

}

cout<<tree.age()<<endl;

return 0;

}\*/

/\*

//F9 第四章 Complex复数类

#include<iostream>

using namespace std;

class Complex{

private:

double x,y;

public:

Complex(double x, double y=0)

{

this->x=x;

this->y=y;

}

void add(Complex com)

{

this->x=this->x+com.x;

this->y=this->y+com.y;

}

void show()

{

cout<<x<<"+"<<y<<"i"<<endl;

}

};

int main()

{

Complex c1(3,5); //initialize c1 with complex 3+5i

Complex c2=4.5; //initialize c2 with 4.5

c1.add(c2); //c1+c2，result saved in c1

c1.show(); //print out c1.（result is 7.5+5i）

return 0;

}\*/

/\*

//F10 第四章 Circle面积

#include<iostream>

using namespace std;

class Circle

{

public:

Circle(){}

Circle(double r):radius(r){}

~Circle(){}

double radius;

double getArea()

{

return 3.14\*radius\*radius;

}

};

int main()

{

double r;

cin>>r;

Circle c1(r);

cout<<c1.getArea()<<endl;

return 0;

}\*/

//F11 第四章 Dog类

#include<iostream>

using namespace std;

class Dog

{

private:

int age;

int weight;

public:

Dog(){}

Dog(int a,int w):age(a),weight(w){}

~Dog(){}

void setAge(int age)

{

this->age=age;

}

void setWeight(int weight)

{

this->weight=weight;

}

int getAge()

{

return age;

}

int getWeight()

{

return weight;

}

};

int main()

{

int age,weight;

cin>>age>>weight;

Dog d(age,weight);

cout<<d.getAge()<<" "<<d.getWeight()<<endl;

return 0;

}