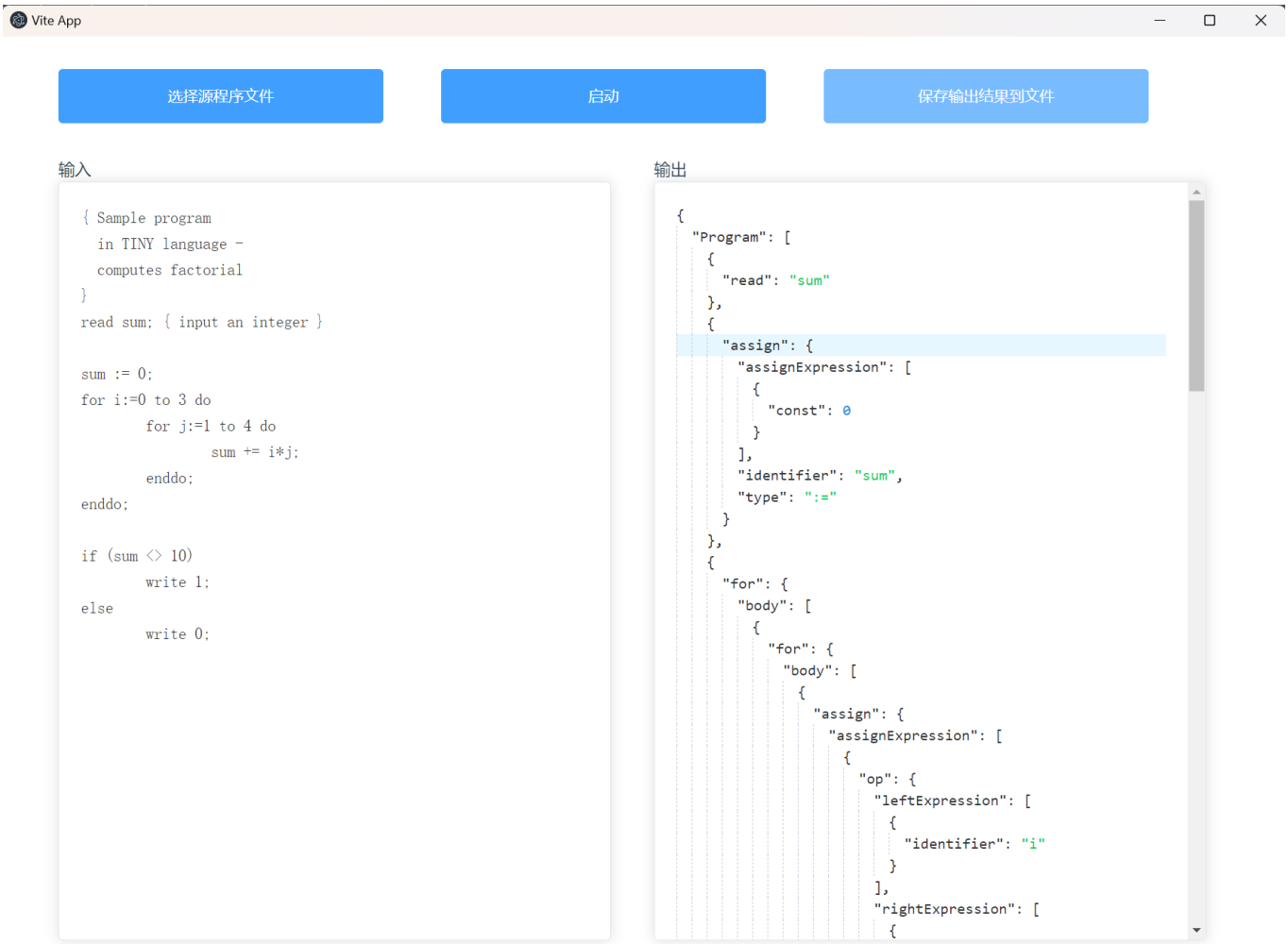
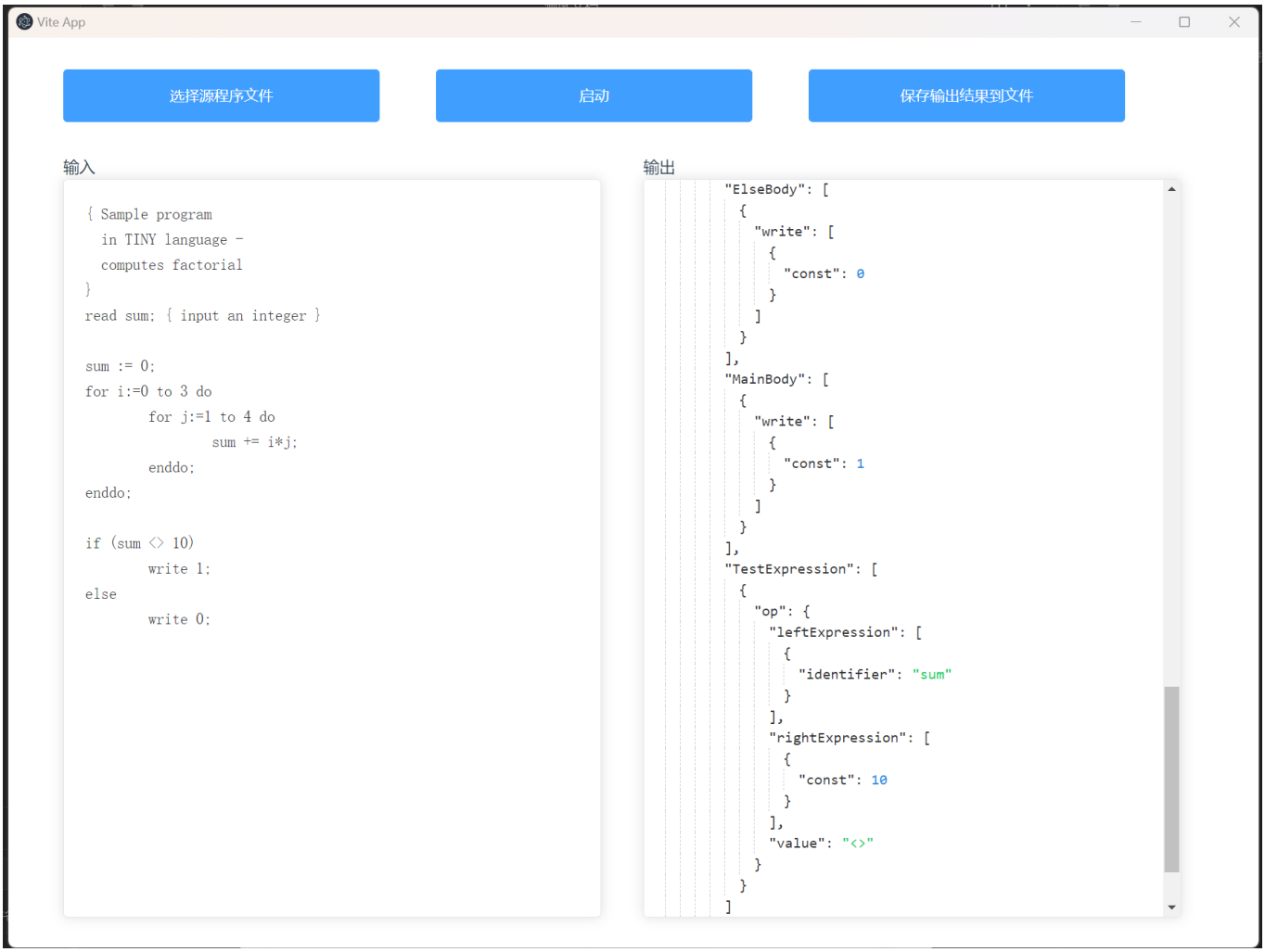
测试文档

测试数据一

```
{ Sample program
  in TINY language -
  computes factorial
}
read sum; { input an integer }

sum := 0;
for i:=0 to 3 do
        for j:=1 to 4 do
                sum += i*j;
        enddo;
enddo;

if (sum <> 10)
        write 1;
else
        write 0;
```

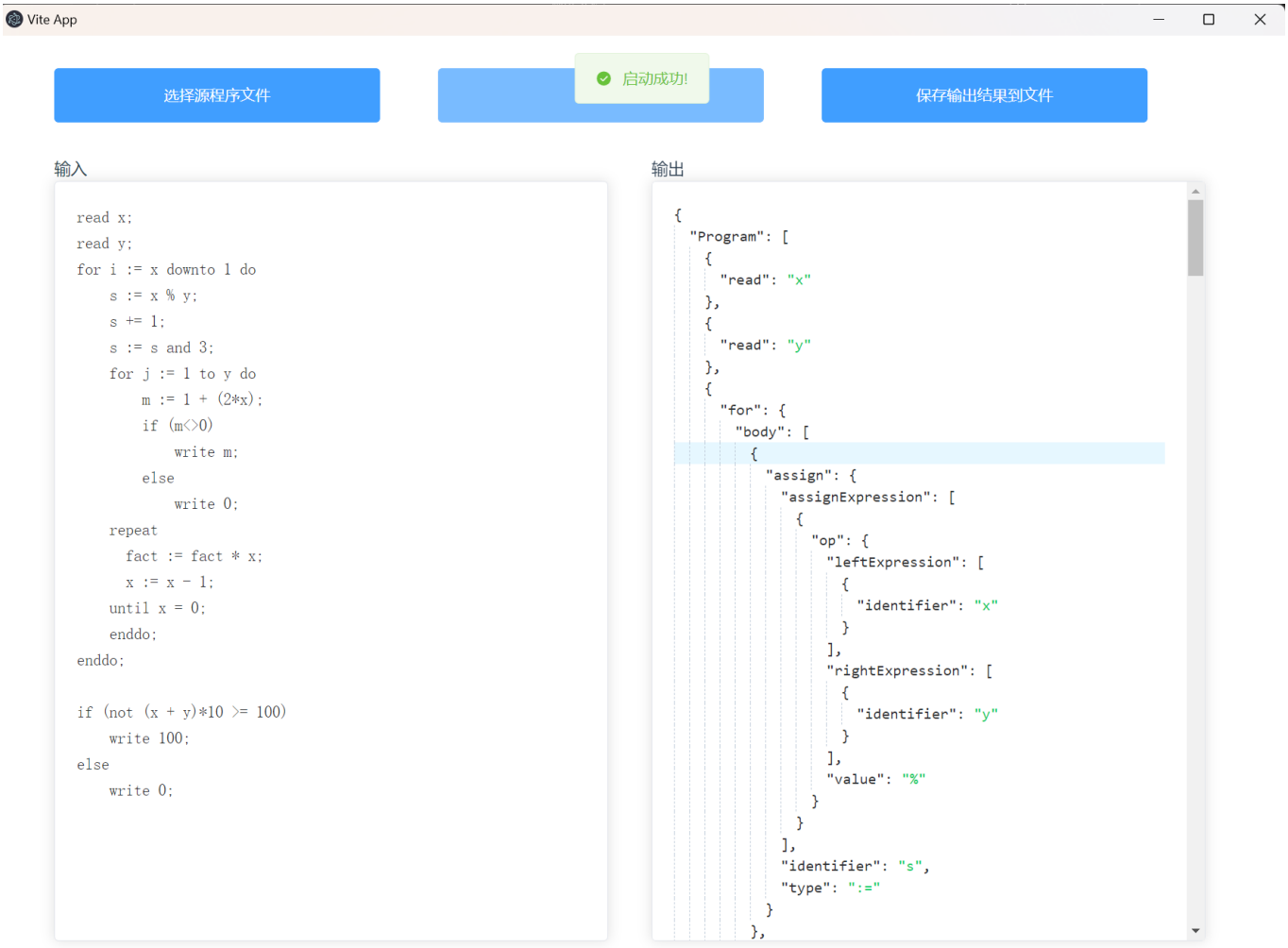

Vite App
选择源程序文件
启动
保存输出结果到文件
输入
{ Sample program
in TINY language -
computes factorial
}
read sum; { input an integer }
sum := 0;
for i:=0 to 3 do
for j:=1 to 4 do
sum += i*j;
enddo;
enddo;
if (sum <> 10)
write 1;
else
write 0;
输出
{
"Program": [
{
"read": "sum"
},
{
"assign": {
"assignExpression": [
{
"const": 0
}
],
"identifier": "sum",
"type": ":="
}
},
{
"for": {
"body": [
{
"for": {
"body": [
{
"assign": {
"assignExpression": [
{
"op": {
"leftExpression": [
{
"identifier": "i"
}
],
"rightExpression": [
{

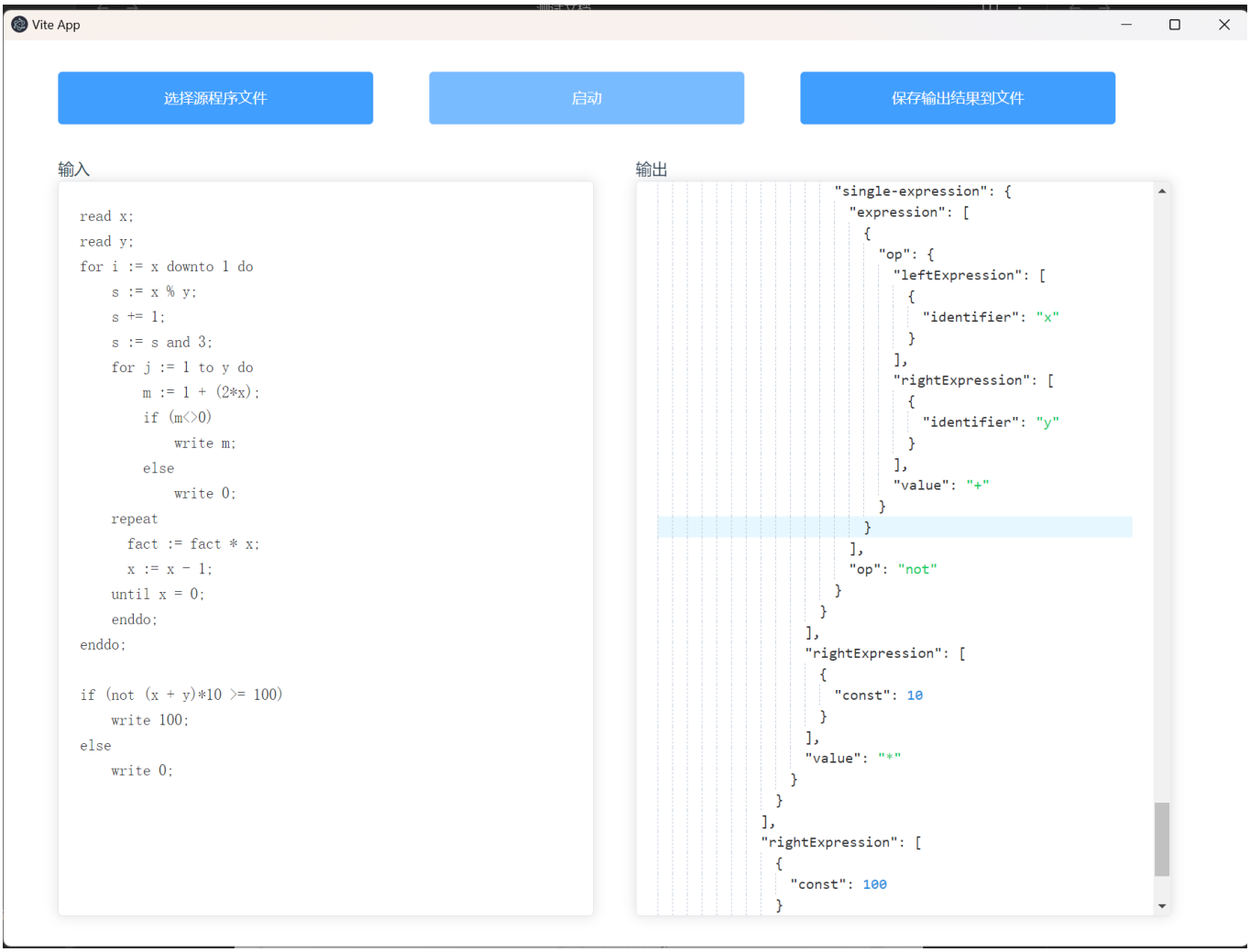


测试数据二

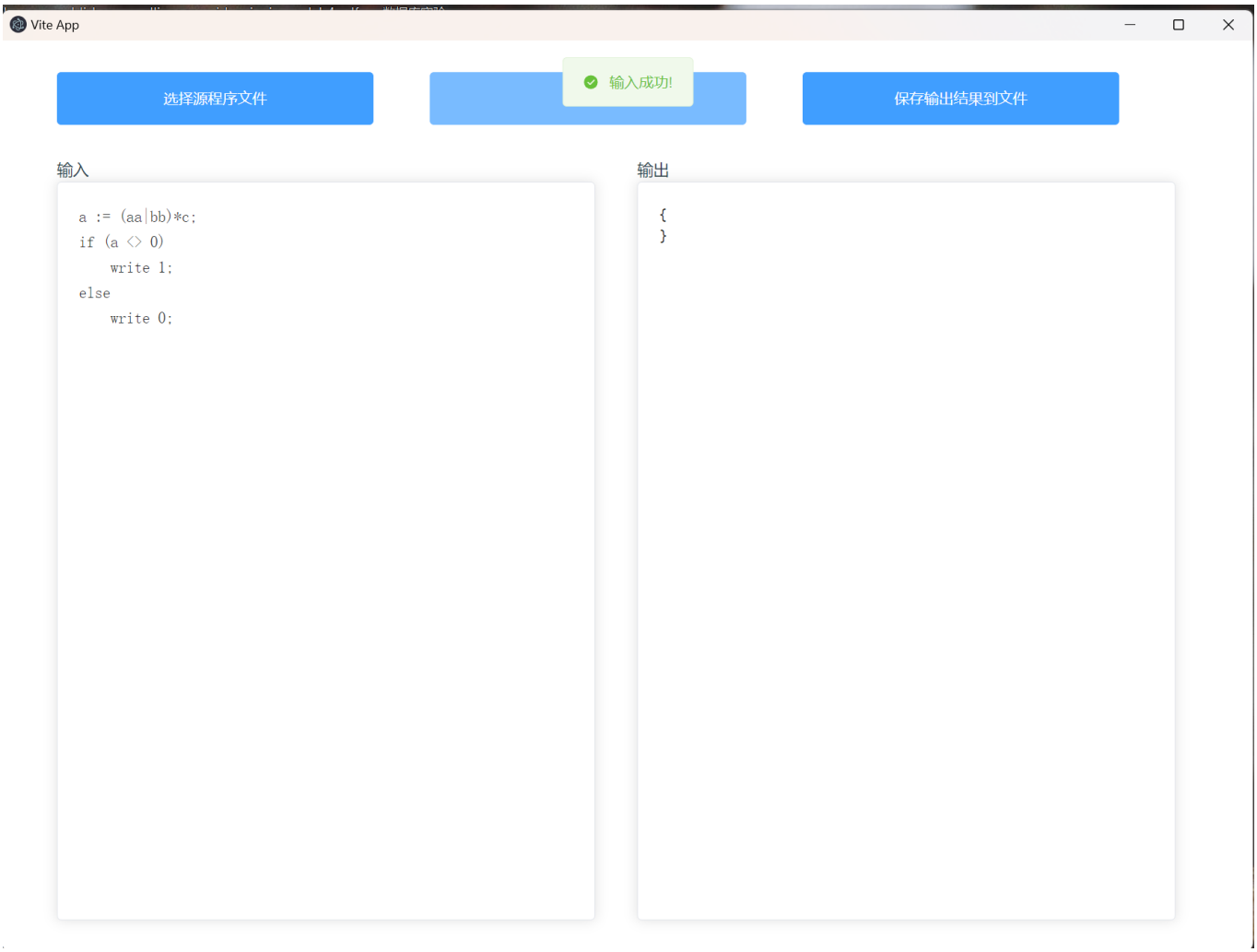
```
read x;
read y;
for i := x downto 1 do
    s := x % y;
    s += 1;
    s := s and 3;
    for j := 1 to y do
        m := 1 + (2*x);
        if (m<>0)
            write m;
        else
            write 0;
    repeat
      fact := fact * x;
      x := x - 1;
    until x = 0;
    enddo;
enddo;

if (not (x + y)*10 >= 100)
```

```
    write 100;
else
    write 0;
```




Vite App
选择源程序文件
启动
保存输出结果到文件
输入
read x;
read y;
for i := x downto 1 do
s := x % y;
s += 1;
s := s and 3;
for j := 1 to y do
m := 1 + (2*x);
if (m<>0)
write m;
else
write 0;
repeat
fact := fact * x;
x := x - 1;
until x = 0;
enddo;
enddo;
if (not (x + y)*10 >= 100)
write 100;
else
write 0;
输出
"single-expression": {
"expression": [
{
"op": {
"leftExpression": [
{
"identifier": "x"
}
],
"rightExpression": [
{
"identifier": "y"
}
],
"value": "+"
}
}
],
"op": "not"
}
}
],
"rightExpression": [
{
"const": 10
}
],
"value": "*"
}
}
],
"rightExpression": [
{
"const": 100
}

测试数据三



总结

可以正确的识别单元运算符（ `not` ）和 `+=` ， `-=` 等扩充的运算符号。也可以正确的识别 `for` 循环和扩充的比较运算符号（ `>=` ， `<>` ， `<=` ），也支持位运算符。但是这里识别正则表达式有些问题，如果遇到正则表达式会报错。