CSCI 1540 Introduction to Computing Using C++

Tutorial 4

Xinyi Hu

SHB 1005

xyhu@cse.cuhk.edu.hk

Outline

- Assignment 3

- Formatting output

Outline

- Assignment 3
- Formatting output

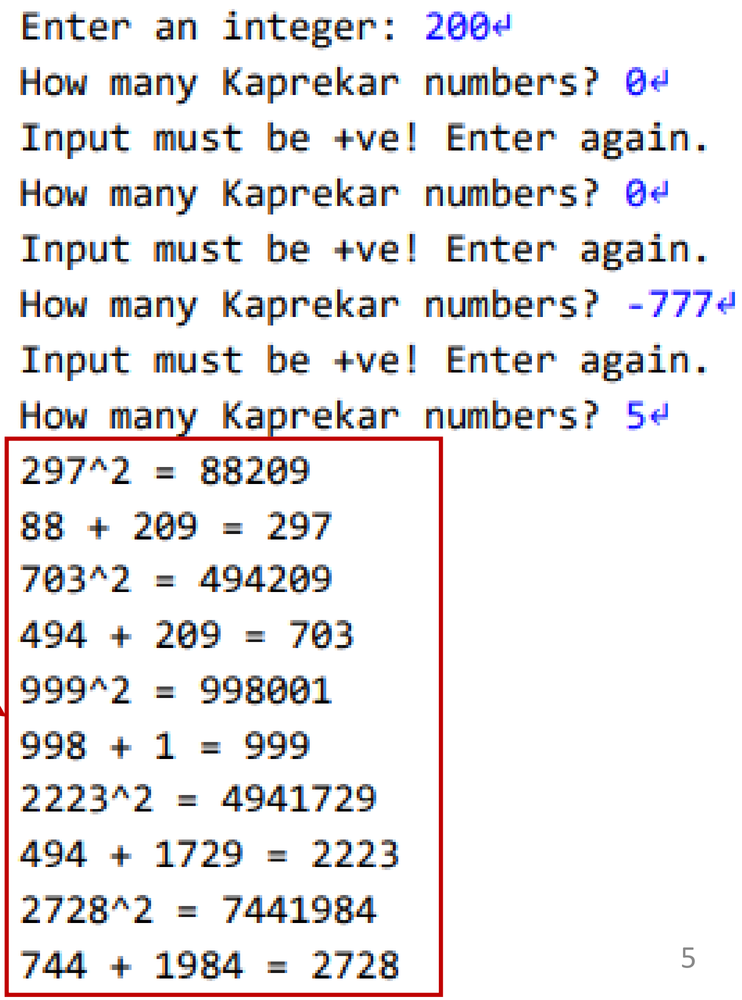
Assignment 3: Introduction

- A Kaprekar number is a positive integer x if the digits of x^2 can be split into two parts that add up to x, where the part formed from the low-order (rightmost) digits of x^2 must be non-zero (although leading zeroes are allowed in the part).
- 31 is not *Kaprekar number*:
  - 31^2 = 961
  - 96 + 1 ≠ 31, 9 + 61 ≠ 31, and 0 + 961 ≠ 31
- 45 is a *Kaprekar number*:
  - 45^2 = 2025
  - 20 + 25 = 45

Assignment 3: Introduction

- Requirement:
  - Input: A positive integer *s*, a positive integer *n* that presents how many Kaprekar numbers you want to look for;
  - Output: the square and the split of the two parts of the Kaprekar number

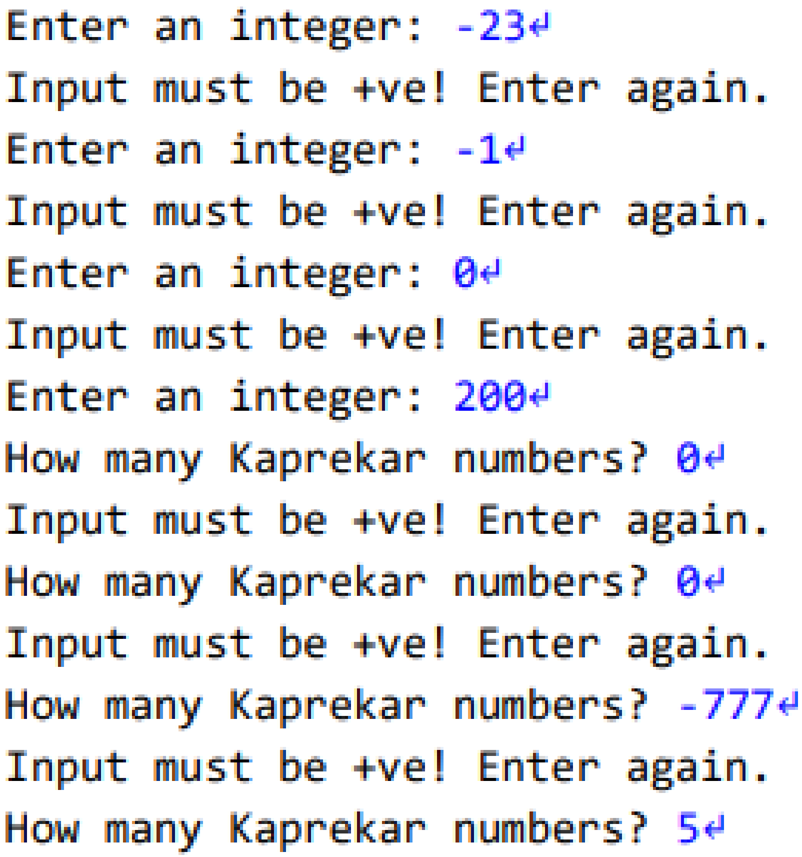
- Example:



```
Enter an integer: 200↵
How many Kaprekar numbers? 0↵
Input must be +ve! Enter again.
How many Kaprekar numbers? 0↵
Input must be +ve! Enter again.
How many Kaprekar numbers? -777↵
Input must be +ve! Enter again.
How many Kaprekar numbers? 5↵
297^2 = 88209
88 + 209 = 297
703^2 = 494209
494 + 209 = 703
999^2 = 998001
998 + 1 = 999
2223^2 = 4941729
494 + 1729 = 2223
2728^2 = 7441984
744 + 1984 = 2728
```

Assignment 3: Specification

- Input: always an integer;
- Input check: non-positive is not allowed;
- When *s*, *n* is not positive, you should display a warning message and ask for another input, until *s*, *n* is positive.



```
Enter an integer: -23↵
Input must be +ve! Enter again.
Enter an integer: -1↵
Input must be +ve! Enter again.
Enter an integer: 0↵
Input must be +ve! Enter again.
Enter an integer: 200↵
How many Kaprekar numbers? 0↵
Input must be +ve! Enter again.
How many Kaprekar numbers? 0↵
Input must be +ve! Enter again.
How many Kaprekar numbers? -777↵
Input must be +ve! Enter again.
How many Kaprekar numbers? 5↵
```

Assignment 3: Specification

- Note: You are not allowed to use any functions in the <cmath> library in this assignment.

So think about how to get number of digits of given x and how to spilt the given x

Maybe we can use loop, % and / ?

Assignment 3: Specification

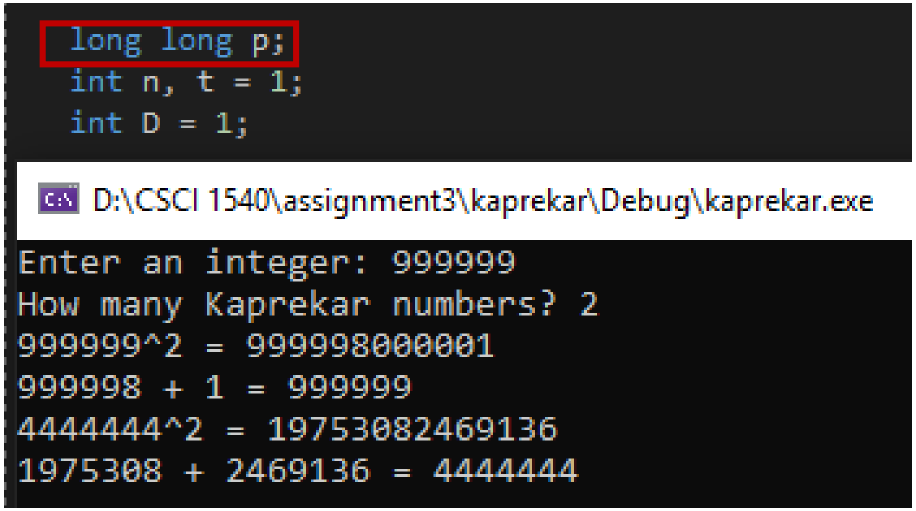
- Note: you are required to use the data type long long instead of int for all integer variables in this assignment

Why?

int -2147483648 ~ +2147483647 (4 Bytes) 2*10^9

long long -9223372036854775808 ~ +9223372036854775807 (8 Bytes) 9*10^18

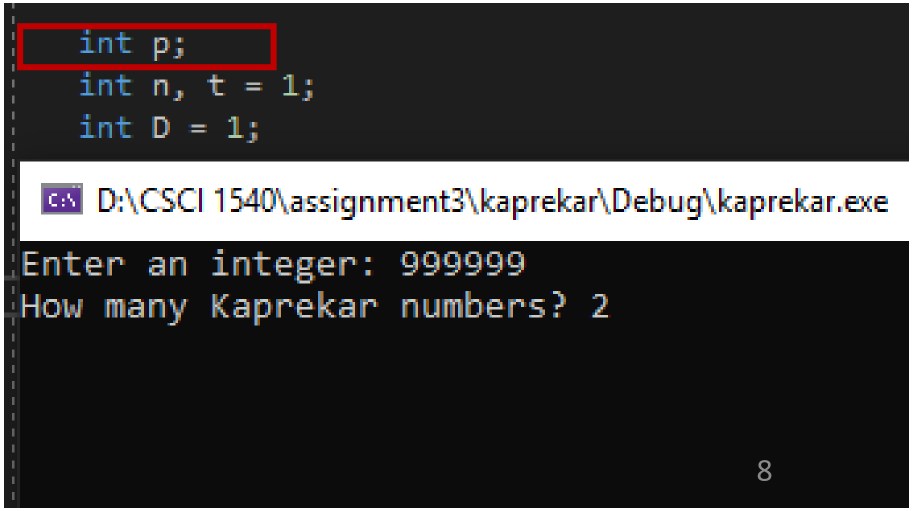
A square can easily go overflow with the int type.



```
long long p;
int n, t = 1;
int D = 1;
```

D:\CSCI 1540\assignment3\kaprekar\Debug\kaprekar.exe

```
Enter an integer: 999999
How many Kaprekar numbers? 2
999999^2 = 999998000001
999998 + 1 = 999999
4444444^2 = 19753082469136
1975308 + 2469136 = 4444444
```



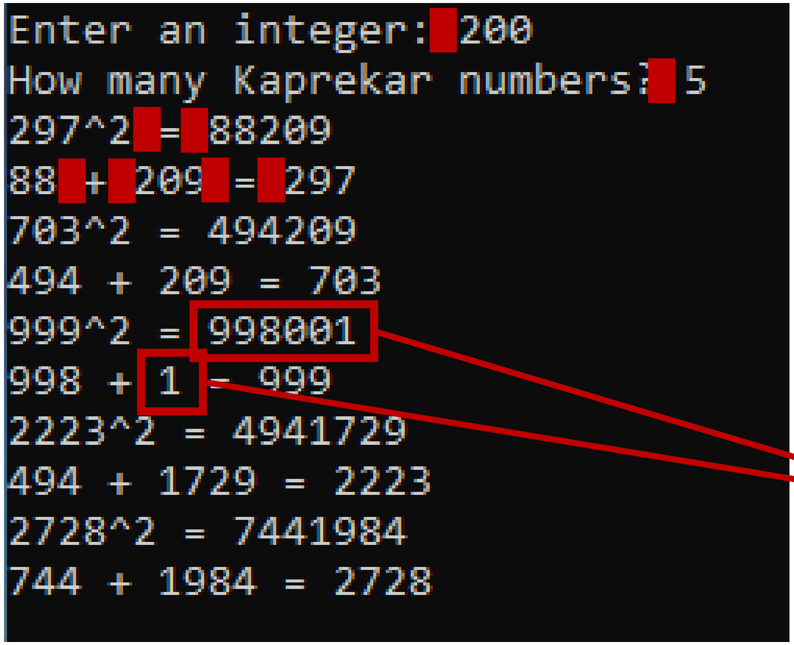
```
int p;
int n, t = 1;
int D = 1;
```

D:\CSCI 1540\assignment3\kaprekar\Debug\kaprekar.exe

```
Enter an integer: 999999
How many Kaprekar numbers? 2
```

Assignment 3: Specification

- The output should be the **100% same** with sample output. (i.e., same text, same symbols, same letter case, same number of spaces, etc.)



```
Enter an integer: 200
How many Kaprekar numbers? 5
297^2 = 88209
88 + 209 = 297
703^2 = 494209
494 + 209 = 703
999^2 = 998001
998 + 1 = 999
2223^2 = 4941729
494 + 1729 = 2223
2728^2 = 7441984
744 + 1984 = 2728
```

Again, pay attention to the **Spaces**!

998001:
Be spilt into 998 and 001, but the output should be 998 and 1

Assignment 3: Steps

Necessary Steps:

- User input;
- Check Kaprekar number;
- Output;

Step 1 – User Input

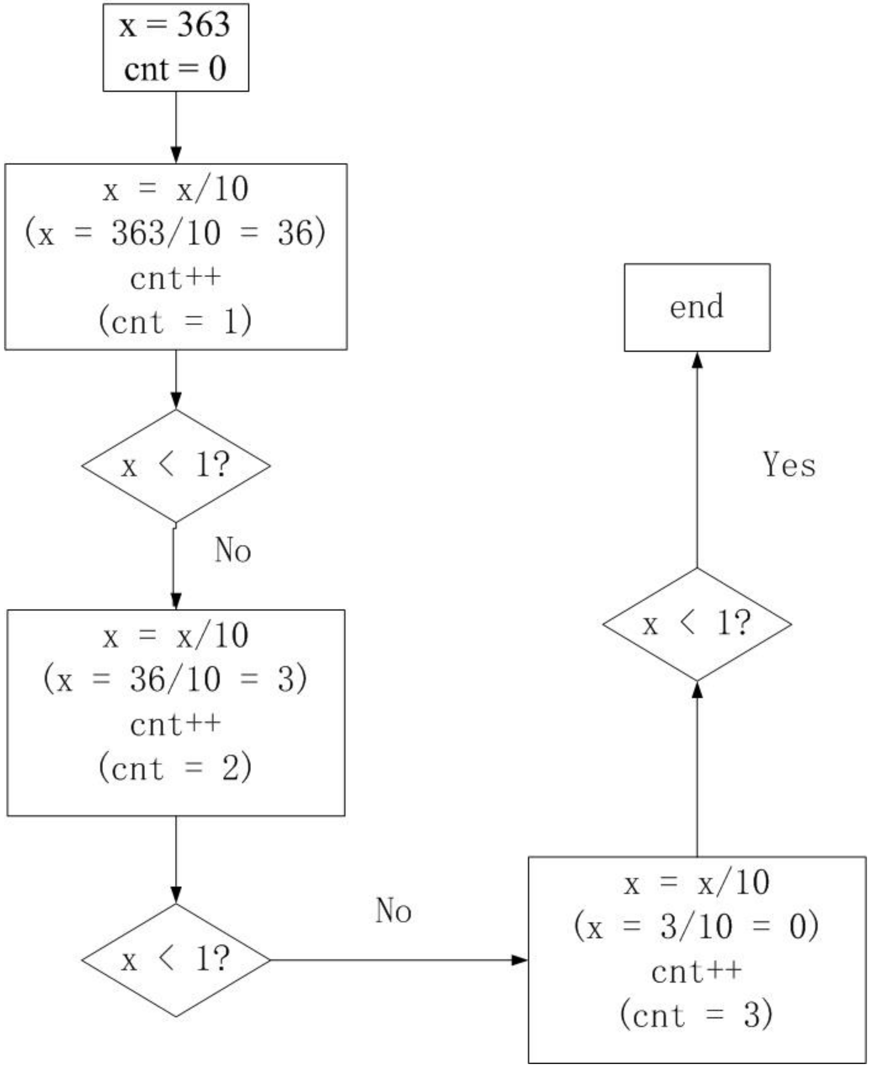
- **Inputs**:
  - A positive integer *s*, a positive integer *n* that presents how many Kaprekar numbers you want to look for;
  - Use *cin* to read the inputs;
- **Input check**:
  - Is it a non-positive integer number?
  - Use *cout* to print messages if input is non-positive;

```
while (x is non-positive){
        cout<<“Invalid. Try again!”;
        cin>> x;
}
```

Step 2 – Check Kaprekar Number

- **Get digit number of x**

- Repeatedly applying x = x/10 with a counter;
- Stop until x<1;
- Eg. Given the number x = 363, you want to get 3:



Step 2 – Check Kaprekar Number

- **Get each digit of x**
  - Repeatedly applying y = mod(x, 10);
  - Stop until x < 1;

  - Eg. x = 325
  - d = x % 10 = 325 % 10 = 5, x / 10 = 325 / 10 =32
  - d = x % 10 = 32 % 10 = 2, x / 10 = 32 / 10 = 3
  - d = x % 10 = 3 % 10 =3, x / 10 = 3 / 10 = 0

  - So you can get 5 2 3

Step 2 – Check Kaprekar Number

- **Check whether x is a Kaprekar number**
  - split into two parts
  - compute the sum of the two parts

Eg. x = 45^2 = 2025, we want to get a = 20, b = 25

- First, compute the number of digits of x, which is 4 (the method is in p12)
- while(digit!=0) {

```
while(digit!=0) {
      digit--;
      long long tmp = 1;
      int time = digit;
      while (time!=0) {
                tmp *= 10;
                time--;
      }
}
```

Test digit from digit-1 to 1, and try all possible split positions

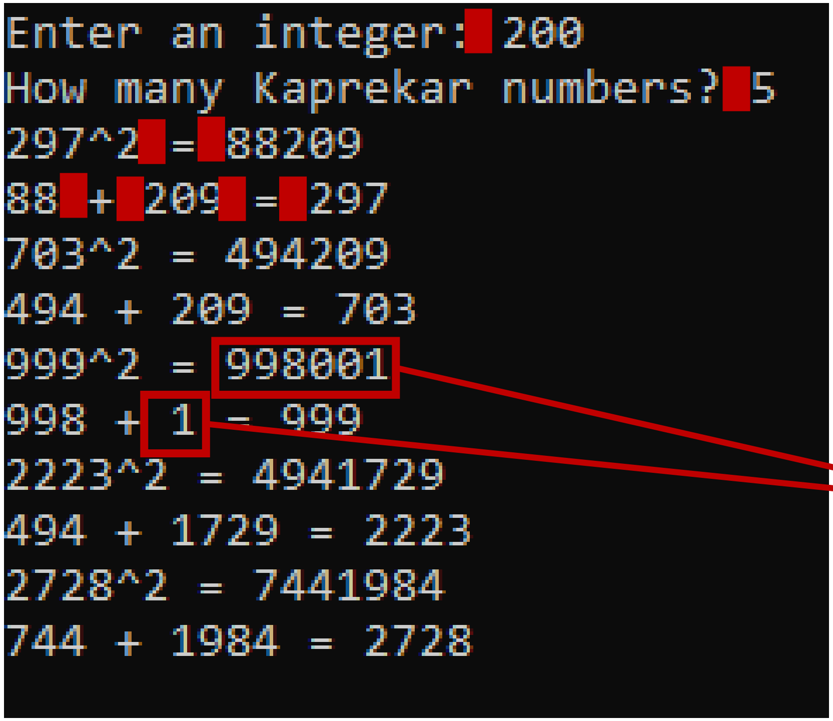
When digit = 3, we can get tmp = 1000;
a = x / tmp = 2025 / 1000 =2;
b = x % tmp = 2025 % 1000 = 25

When digit = 2, we can get tmp = 100;
a = x / tmp = 2025 / 100 =20;
b = x % tmp = 2025 % 100 = 25

When digit = 1, we can get tmp = 10;
a = x / tmp = 2025 / 10 =202;
b = x % tmp = 2025 % 10 = 5

Step 3 – Output

- **Output**:
  - the square and the split of the two parts of the Kaprekar number;
  - Use *cout* to print the results;

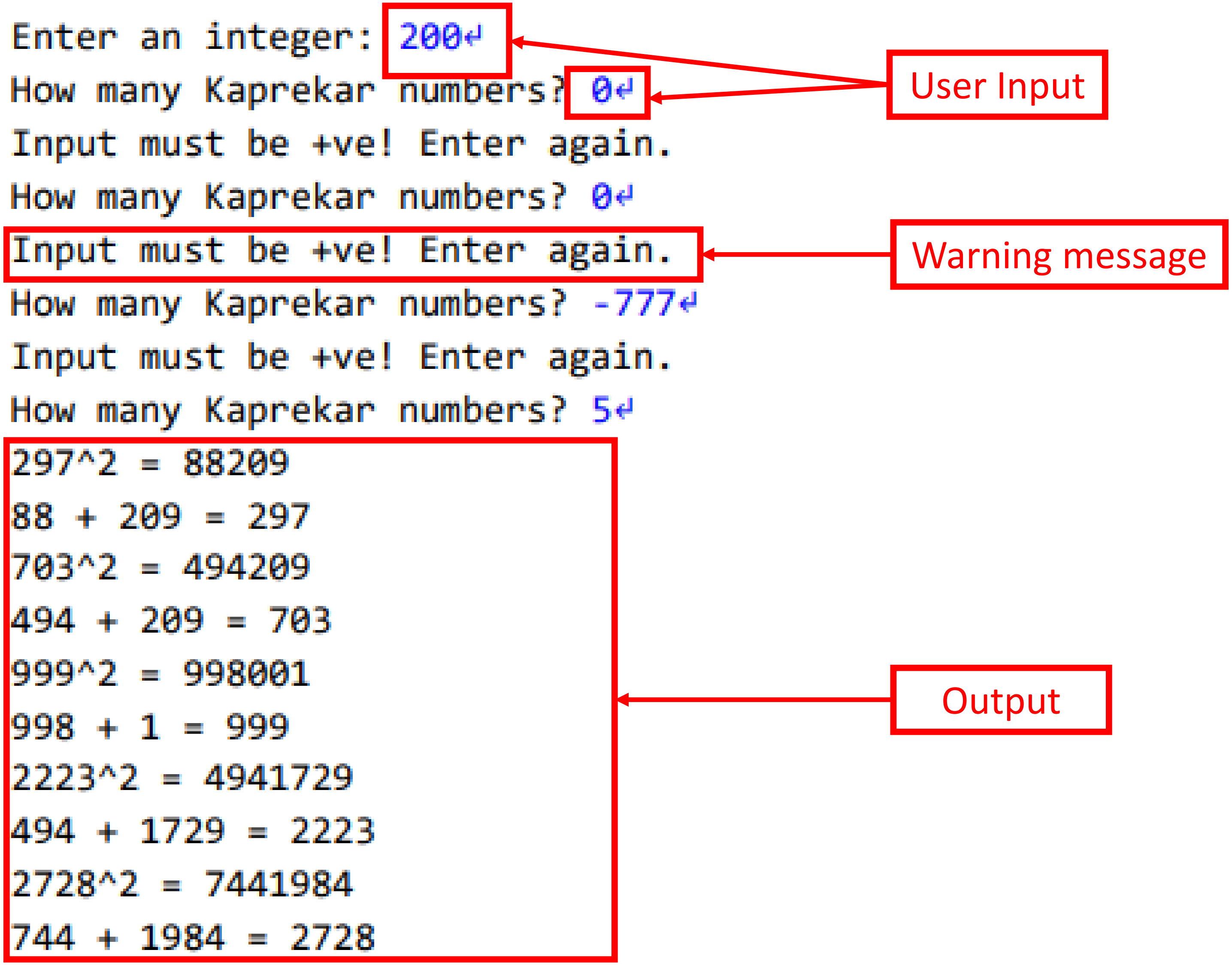


```
Enter an integer: 200
How many Kaprekar numbers? 5
297^2 = 88209
88 + 209 = 297
703^2 = 494209
494 + 209 = 703
999^2 = 998001
998 + 1 = 999
2223^2 = 4941729
494 + 1729 = 2223
2728^2 = 7441984
744 + 1984 = 2728
```

Again, pay attention to the **Spaces**!

998001:
Be spilt into 998 and 001, but the output should be 998 and 1

Sample Output



```
Enter an integer: 200↵
How many Kaprekar numbers? 0↵
Input must be +ve! Enter again.
How many Kaprekar numbers? 0↵
Input must be +ve! Enter again.
How many Kaprekar numbers? -777↵
Input must be +ve! Enter again.
How many Kaprekar numbers? 5↵
297^2 = 88209
88 + 209 = 297
703^2 = 494209
494 + 209 = 703
999^2 = 998001
998 + 1 = 999
2223^2 = 4941729
494 + 1729 = 2223
2728^2 = 7441984
744 + 1984 = 2728
```

User Input

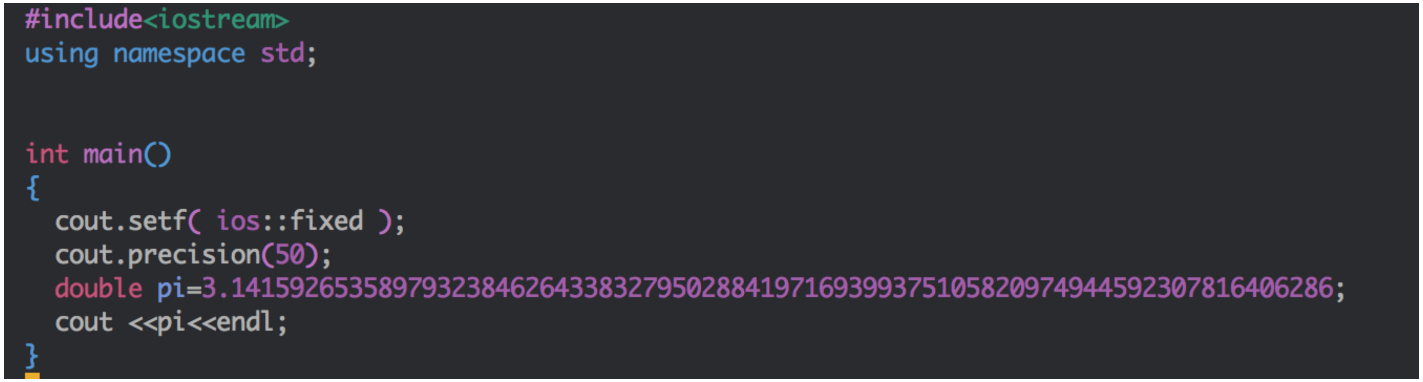
Warning message

Output

Outline

- Assignment 3
- Formatting output

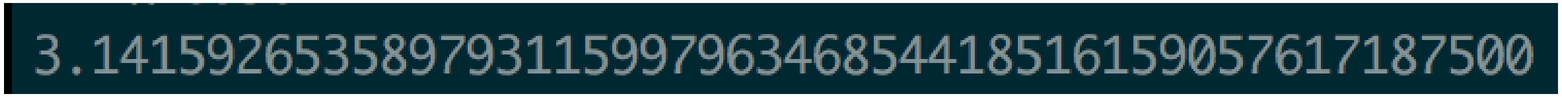
Why we want to format output?



```
#include<iostream>
using namespace std;


int main()
{
  cout.setf( ios::fixed );
  cout.precision(50);
  double pi=3.141592653589793238462643383279502884197169399375105820974944592307816406286;
  cout <<pi<<endl;
}
```

- Output



```
3.14159265358979311599796346854418516159057617187500
```

- Do we really care about the 40-th digit of pi?
  - ➢ No! – so why not make the output **cleaner**, for example, output just two decimal place only? (3.14)

Format flags

- Flags are like switches (which can be turned on/off).

- Based on the values of the formatting flags, an output **stream** object decides how to output a value.

```
int number = 1023;

cout.setf( ios::dec );
cout << number << endl;
cout.unsetf( ios::dec );

cout.setf( ios::hex );
cout << number << endl;

cout.setf( ios::uppercase );
cout << number << endl;
cout.unsetf( ios::hex );

cout.setf( ios::oct );
cout << number << endl;
cout.unsetf( ios::oct );

cout << true << endl;
cout.setf( ios::boolalpha );
cout << true << endl;
```

```
1023
3ff
3FF
1777
1
true
```

Using format flags to format integers

Format flags

- **`boolalpha`** -- Boolean values can be input/output using the words "true" and "false".

- **`dec`** / **`oct`** / **`hex`**– Numeric values are displayed in decimal / octal / hexadecimal.

- **`fixed`** – Display floating point values using normal notation (as opposed to scientific).
- **`scientific`** – Display floating point values using scientific notation

- **`internal`** – Numeric value is padded to fill a field, spaces are inserted between the sign and base character.

- **`left`** / **`right`** – Output is left/right justified

- **`showbase`** – Display the base of all numeric values

- **`showpoint`** – Display a decimal and extra zeros, even when not needed

- **`showpos`** – Display a leading plus sign before positive numeric values

- **`skipws`** – Discard whitespace characters (spaces, tabs, newlines) when reading from a stream

- **`unitbuf`** – Flush the buffer after each insertion

- **`uppercase`** – Display the "e" of scientific notation and the "x" of hexidecimal notation as capital letters

Formatting Floating Points

- **fixed or scientific**
  - **showpoint** flag has no effect (always assume **showpoint** is on)

- **Default** (When both **fixed** and **scientific** flags are off)
  - If **showpoint** is off, won't print unnecessary trailing zeros after decimal places.
  - Automatically switch between fixed and scientific notation depends on the magnitude of the value.

```cpp
double num = 1.234;
// Default precision is 6
cout.setf( ios::showpoint );
cout << 100.0 << endl;

cout.setf( ios::scientific );
cout << num << endl;
cout.unsetf( ios::scientific );

cout.setf( ios::fixed );
cout << num << endl;
cout.precision(12);          // Set precision to 12
cout << num << endl;
cout.unsetf( ios::fixed );

// Use the "default" format for floating point numbers
cout.unsetf( ios::showpoint );
cout << num << endl;
cout << 100.0 << endl;
cout << 100000000000000.0 << endl;
```

```
100.000
1.234000e+00
1.234000
1.234000000000
1.234
100
1e+14
```

Example: Using format flags to format floating point numbers

```cpp

int main() {

  cout << 123.0 << endl;
  cout.setf(ios::showpoint);
  cout << 1.0 << endl;
  cout << 123.0 << endl;

  return 0;
}
```

```
123
1.00000
123.000
```

Flag states are carried along with the stream object.

It is important to make sure the flag states remain unchanged after local use of the stream object.

I/O Manipulators

- We can also use *manipulators* to manipulate flags indirectly
- For example, to set the "dec" flag, we can write

cout << dec;

- Some of the manipulators are defined in <**iostream**> and some are defined in **<iomanip>**

```
int number = 0x03ff;

cout << dec << number << endl;
cout << hex << number << endl;
cout << uppercase << number << endl;
cout << oct << number << endl;

cout << noboolalpha << true << endl;
cout << boolalpha << true << endl;
cout << endl;

```

```
1023
3ff
3FF
1777
1
true
```

Example: Using manipulators to format integers

```
double num = 1.234;

cout << showpoint << 100.0 << endl;
cout << scientific << num << endl;
cout << fixed << num << endl;

cout << setprecision(12) << num << endl;

// Reset to the "default" floating point format
cout.unsetf( ios::scientific | ios::fixed );

cout << noshowpoint << num << endl;
cout << 100.0 << endl;
```

```
100.000
1.234000e+00
1.234000
1.234000000000
1.234
100
```

- Example: Using manipulators to format floating point numbers

```
cout << "----------==========" << endl;

// Left justified the value in the reserved space
cout << left;
cout << setw(10) << 123 << setw(10) << "ABC" << endl;

// Right justified the value in the reserved space
cout << right;
cout << setw(10) << 123 << setw(10) << "ABC" << endl;
```

```
----------==========
123       ABC
       123       ABC
```

- **setw(field_width)** only applies to the next value inserted to the stream.
- Without **setw(field_width)** or when **field_width** is too small, left/right justification has no effect.

Manipulators defined in <iostream>			
Manipulator	Description	Input	Output
boolalpha	Turns on the boolalpha flag	X	X
dec	Turns on the dec flag	X	X
endl	Output a newline character, flush the stream		X
ends	Output a null character		X
fixed	Turns on the fixed flag		X
flush	Flushes the stream		X
hex	Turns on the hex flag	X	X
internal	Turns on the internal flag		X
left	Turns on the left flag		X
noboolalpha	Turns off the boolalpha flag	X	X
noshowbase	Turns off the showbase flag		X
noshowpoint	Turns off the showpoint flag		X
noshowpos	Turns off the showpos flag		X

noskipws	Turns off the skipws flag	X	
nounitbuf	Turns off the unitbuf flag		X
nouppercase	Turns off the uppercase flag		X
oct	Turns on the oct flag	X	X
right	Turns on the right flag		X
scientific	Turns on the scientific flag		X
showbase	Turns on the showbase flag		X
showpoint	Turns on the showpoint flag		X
showpos	Turns on the showpos flag		X
skipws	Turns on the skipws flag	X	
unitbuf	Turns on the unitbuf flag		X
uppercase	Turns on the uppercase flag		X
ws	Skip any leading whitespace	X	

Manipulators defined in <iomanip>			
Manipulator	Description	Input	Output
resetiosflags(long f)	Turn off the flags specified by *f*	X	X
setbase(int base)	Sets the number base to *base*		X
setfill(char ch)	Sets the fill character to *ch*		X
setiosflags(long f)	Turn on the flags specified by *f*	X	X
setprecision(int p)	Sets the number of digits of precision		X
setw(int w)	Sets the field width to *w*		X

Q & A