1. What data type in the C programming language allows for the largest values of factorial to be computed?

Answer：
In the C programming language, there is no specific data type specifically designed to compute the maximum value of a factorial. The results of factorial can quickly become very large, and even the largest built-in data types (such as unsigned long long) may not hold very large factorial values.

For example, 20 factorial (20!) It is already more than 200 million numbers, which is beyond the scope of the int type. Larger factorial values, such as 30, will far exceed the maximum value of the unsigned long long type.

Therefore, when you need to calculate factorials, you usually need to use some special strategy or data structure to deal with large numbers. This may include using arrays or strings to represent large numbers, or using specialized mathematical libraries to handle large number operations.

If you just want to calculate the maximum value of the factorial, you need to be clear about what you mean by "maximum". If you mean the maximum factorial value that can be represented, then it depends on the data structure and method you use. If you are referring to the maximum input value that the factorial operation can handle, then this also depends on your implementation and available memory.

In general, the C language itself does not directly provide a data type to deal exclusively with the maximum value of factorial, because this requires special handling of large numbers, which is usually beyond the scope of the language itself.

2. At what input value for the recursive factorial function does your computer start to 'crash' or really slow down when you try to compute a factorial? Is it the same value as the iterative function? Experiment and report your result.

Answer:

**Use a bigger data type--unsigned long long int**

23, same as the iterative function.

```
factorial(10) = 3628800
factorial(10) = 3628800
```

```
factorial(20) = 2432902008176640000
factorial(20) = 2432902008176640000
```

```
factorial(21) = 14197454024290336768
factorial(21) = 14197454024290336768
```

```
factorial(22) = 17196083355034583040
factorial(22) = 17196083355034583040
```

```
factorial(23) = 8128291617894825984
factorial(23) = 8128291617894825984
```

**Use int type**

32, same as the iterative function.

```
[biology000@login-students Lab06]$ ./factorial
factorial(23) = 862453760
factorial_recursion(23) = 862453760
```

```
factorial(30) = 1409286144
factorial_recursion(30) = 1409286144
```

```
factorial(31) = 738197504
factorial_recursion(31) = 738197504
```

```
factorial(32) = -2147483648
factorial_recursion(32) = -2147483648
```

```
factorial(33) = -2147483648
factorial_recursion(33) = -2147483648
```

```
factorial(35) = 0
factorial_recursion(35) = 0
```

```
factorial(40) = 0
factorial_recursion(40) = 0
```

```
factorial(50) = 0
factorial_recursion(50) = 0
```

3. In 2-3 sentences, describe why you believe you saw or didn't see differences between the iterative and recursive versions of factorial.

Answer:

The iterative and recursive versions of factorial are similar in computational logic, but they differ in how they are implemented and in memory usage. The iterative version uses loops to compute factorial incrementally, while the recursive version incrementally reduces the size of the problem through function calls themselves. I think I (or others) may not see the reason for the difference between them, probably because the two approaches are mathematically equivalent, but in programming practice, the recursive version is generally more concise, while the iterative version may be easier to understand and debug. In addition, for very large input values, the iterative version may be more efficient because it avoids the additional overhead of recursive calls.