

# 计算机系统基础 代码优化实验报告

## 1. 实验内容:

完成代码优化任务,

## 2. 实验步骤:

1. 分析代码架构与任务

2. 优化 `calculate1` 函数

3. 在 `calculate1` 基础上使用 `simd` 指令尝试编写速度更快的 `calculate2`

## 3. 解题步骤:

阅读原 `calculate1` 函数, 发现多处可优化位置, 如原函数多次调用 `getdata` 函数, 造成不必要的时间开销; 按变量进行多次迭代, 浪费了每次迭代的计算效率; 使用多个 `for` 循环内局部变量, 造成空间初始化与释放的时间浪费。

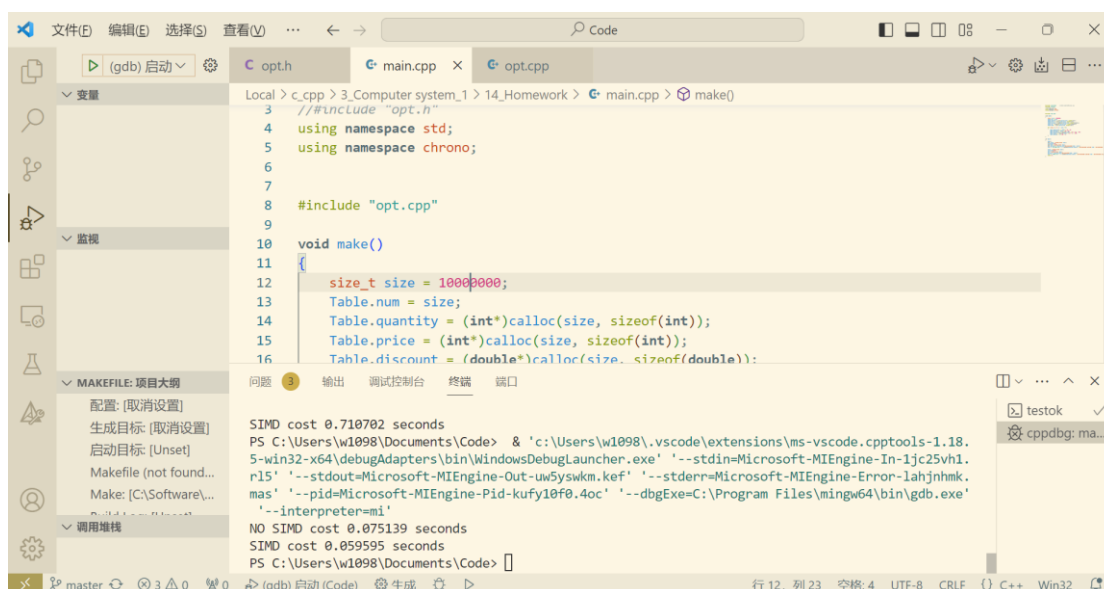
因此, 在原函数基础上将多个循环进行整合, 使用以整个函数为作用域的变量对 `gatdata` 值进行存储并利用, 从而削减了不必要的时间开销, 得到初步优化的函数, 并将运行速度从 `0.254453s` 减少至 `0.074149s`。

但初步优化后的函数运行速度仍然不能令人满意, 经过时间效率与函数编写复杂度的取舍, 采取 `2*2` 循环展开, 及使用步长为 `2` 的循环进行迭代, 并且每次处理两个元素, 成功将运行速度进一步小幅压缩, 达到 `0.064809 s`。

在 `calculate1` 函数基础上进行 `calculate2` 的编写。首先仅使用 `simd` 指令对初步优化后的、未经循环展开的 `calculate1` 函数进行优化, 达到 `0.070253s` 的运行速度, 但与未使用 `simd` 指令的 `calculate1` 函数差异不大。于是尝试在循环展开后的 `calculate1` 函数基础上采用 `simd` 指令, 但运行时

间反而延长至 0.101284 s，猜测为高频的数据类型转换所需要的时间开销带来的影响。

最终，结合运行速度与代码复杂度进行综合考虑，我们选择使用未经循环展开的，使用 simd 指令的 calculate1 函数作为 calculate2 函数，在保持运行速度基本不变的基础上降低了代码复杂度，多次测验中所得到的最快运行速度如下：



```
Local > c.cpp > 3_Computer system_1 > 14_Homework > main.cpp > make()
3 // #include "opt.h"
4 using namespace std;
5 using namespace chrono;
6
7
8 #include "opt.cpp"
9
10 void make()
11 {
12     size_t size = 1000000;
13     Table.num = size;
14     Table.quantity = (int*)calloc(size, sizeof(int));
15     Table.price = (int*)calloc(size, sizeof(int));
16     Table.discount = (double*)calloc(size, sizeof(double));
17 }
```

SIMD cost 0.710702 seconds  
PS C:\Users\w1098\Documents\Code> & 'c:\Users\w1098\.vscode\extensions\ms-vscode.cpptools-1.18.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-1jc25vh1.r15' '--stdout=Microsoft-MIEngine-Out-uw5yswkm.kef' '--stderr=Microsoft-MIEngine-Error-lahjnhmk.mas' '--pid=Microsoft-MIEngine-Pid-kufy10f0.4oc' '--dbgExe=C:\Program Files\mingw64\bin\gdb.exe' '--interpreter=mi'  
NO SIMD cost 0.075139 seconds  
SIMD cost 0.059595 seconds  
PS C:\Users\w1098\Documents\Code>

#### 4. 实验收获与心得：

收获：初步掌握了 simd 指令与减少函数调用、循环展开等常见的代码优化手段。

心得：高级的代码并不一定会增加程序运行效率，有时普通、简洁的代码反而可以以较高效率运作；电脑性能及当前 cpu 运行频率会对代码运行速度产生较大影响。

相关函数如下：

```
void calculate1(int n)           //经过循环展开的
{
    long long sum_qty = 0, sum_base_price = 0;
    double sum_disc_price = 0, sum_charge = 0, sum_discount = 0, avg_qty
= 0;
    double avg_price = 0, avg_disc = 0;

    size_t count_order = 0;
    int table_num = Table.num;
    int price = 0;
    double discount = 0;
    double oneMdiscount = 0;
    double PandOneDis = 0;

    int price1 = 0;
    double discount1 = 0;
    double oneMdiscount1 = 0;
    double PandOneDis1 = 0;

    int price2 = 0;
    double discount2 = 0;
    double oneMdiscount2 = 0;
    double PandOneDis2 = 0;
    //double tax = 0;

    // 2*2 循环展开的思想是每次迭代处理两个元素，从而减少循环次数和循环控制的
    开销
    // 为了保证正确性，需要考虑 table_num 是否为偶数，以及循环边界的处理
    // 另外，为了避免重复计算，可以将一些常量和中间变量提取出来
    int half_table_num = table_num / 2; // 计算 table_num 的一半，向下取整
    //double one_plus_tax = 1 + getdata(&Table, Table.tax, 0); // 假设税
    率是固定的，只需获取一次
    int a = 0; // 循环变量
    // 处理前半部分的元素，每次两个
    for (; a < half_table_num * 2; a += 2)
    {
        int date1 = getdata(&Table, Table.date, a); // 获取第一个元素的日
        期
        int date2 = getdata(&Table, Table.date, a + 1); // 获取第二个元素
        的日期
    }
```

```

if (date1 <= n && date2 <= n) // 如果两个元素都满足条件
{
    // 获取两个元素的价格和折扣，并计算相关的值
    price1 = getdata(&Table, Table.price, a);
    price2 = getdata(&Table, Table.price, a + 1);

    discount1 = getdata(&Table, Table.discount, a);
    discount2 = getdata(&Table, Table.discount, a + 1);

    oneMdiscount1 = 1 - discount1;
    oneMdiscount2 = 1 - discount2;

    PandOneDis1 = price1 * oneMdiscount1;
    PandOneDis2 = price2 * oneMdiscount2;

    // 累加相关的值
    sum_qty += getdata(&Table, Table.quantity, a) +
getdata(&Table, Table.quantity, a + 1);
    sum_base_price += price1 + price2;
    sum_discount += discount1 + discount2;
    sum_disc_price += PandOneDis1 + PandOneDis2;
    sum_charge += PandOneDis1 * (1 + getdata(&Table, Table.tax,
a))
                                + PandOneDis2 * (1 + getdata(&Table, Table.tax,
a+1));

    count_order += 2; // 增加两个订单的数量
}
else if (date1 <= n) // 如果只有第一个元素满足条件
{
    // 获取第一个元素的价格和折扣，并计算相关的值
    int price1 = getdata(&Table, Table.price, a);
    double discount1 = getdata(&Table, Table.discount, a);
    double oneMdiscount1 = 1 - discount1;
    double PandOneDis1 = price1 * oneMdiscount1;

    // 累加相关的值
    sum_qty += getdata(&Table, Table.quantity, a);
    sum_base_price += price1;
    sum_discount += discount1;
    sum_disc_price += PandOneDis1;
    sum_charge += PandOneDis1 * (1 + getdata(&Table, Table.tax,
a));

    count_order++; // 增加一个订单的数量
}

```

```

else if (date2 <= n) // 如果只有第二个元素满足条件
{
    // 获取第二个元素的价格和折扣，并计算相关的值
    int price2 = getdata(&Table, Table.price, a + 1);
    double discount2 = getdata(&Table, Table.discount, a + 1);
    double oneMdiscount2 = 1 - discount2;
    double PandOneDis2 = price2 * oneMdiscount2;

    // 累加相关的值
    sum_qty += getdata(&Table, Table.quantity, a + 1);
    sum_base_price += price2;
    sum_discount += discount2;
    sum_disc_price += PandOneDis2;
    sum_charge += PandOneDis2 * (1 + getdata(&Table, Table.tax,
a+1));

    count_order++; // 增加一个订单的数量
}
}
// 处理剩余的元素，如果有的话
for (; a < table_num; a++)
{
    int date = getdata(&Table, Table.date, a);
    if (date <= n)
    {
        price = getdata(&Table, Table.price, a);
        discount = getdata(&Table, Table.discount, a);
        oneMdiscount = 1 - discount;
        PandOneDis = price * oneMdiscount;
        //tax =

        sum_qty += getdata(&Table, Table.quantity, a);
        sum_base_price += price;
        sum_discount += discount;
        sum_disc_price += PandOneDis;
        sum_charge += PandOneDis * (1 + getdata(&Table, Table.tax,
a));

        count_order++;
    }
}
SumQuantity = sum_qty;
SumBasePrice = sum_base_price;
SumDiscPrice = sum_disc_price;
SumCharge = sum_charge;
AvgQuantity = SumQuantity / table_num;

```

```

    AvgPrice = SumBasePrice / table_num;
    AvgDiscount = sum_discount / table_num;
    total = count_order;
}

// void calculate2(int n)      //经过循环展开的calculate1 因为加了simd 反而不好
// {
//     long long sum_qty = 0, sum_base_price = 0;
//     double sum_disc_price = 0, sum_charge = 0, sum_discount = 0,
//     avg_qty = 0;
//     double avg_price = 0, avg_disc = 0;
//     size_t count_order = 0;
//     int table_num = Table.num;
//     int price = 0;
//     double discount = 0;
//     double oneMdiscount = 0;
//     double PandOneDis = 0;
//     int price1 = 0;
//     double discount1 = 0;
//     double oneMdiscount1 = 0;
//     double PandOneDis1 = 0;
//     int price2 = 0;
//     double discount2 = 0;
//     double oneMdiscount2 = 0;
//     double PandOneDis2 = 0;
//     //double tax = 0;
//     // 2*2 循环展开的思想是每次迭代处理两个元素，从而减少循环次数和循环控制的开销
//     // 为了保证正确性，需要考虑table_num 是否为偶数，以及循环边界的处理
//     // 另外，为了避免重复计算，可以将一些常量和中间变量提取出来
//     int half_table_num = table_num / 2; // 计算table_num 的一半，向下取整
//     //double one_plus_tax = 1 + getdata(&Table, Table.tax, 0); // 假设税率是固定的，只需获取一次
//     int a = 0; // 循环变量
//     // 处理前半部分的元素，每次两个
//     for (; a < half_table_num * 2; a += 2)
//     {
//         int date1 = getdata(&Table, Table.date, a); // 获取第一个元素的日期
//         int date2 = getdata(&Table, Table.date, a + 1); // 获取第二个元素的日期
//         if (date1 <= n && date2 <= n) // 如果两个元素都满足条件

```

```

//      {
//          // 获取两个元素的价格和折扣, 并计算相关的值
//          price1 = getdata(&Table, Table.price, a);
//          price2 = getdata(&Table, Table.price, a + 1);
//          discount1 = getdata(&Table, Table.discount, a);
//          discount2 = getdata(&Table, Table.discount, a + 1);
//          oneMdiscount1 = 1 - discount1;
//          oneMdiscount2 = 1 - discount2;
//          PandOneDis1 = price1 * oneMdiscount1;
//          PandOneDis2 = price2 * oneMdiscount2;
//          // 累加相关的值
//          sum_qty += getdata(&Table, Table.quantity, a) +
getdata(&Table, Table.quantity, a + 1);
//          sum_base_price += price1 + price2;
//          sum_discount += discount1 + discount2;
//          sum_disc_price += PandOneDis1 + PandOneDis2;
//          sum_charge += PandOneDis1 * (1 + getdata(&Table,
Table.tax, a))
//                          + PandOneDis2 * (1 + getdata(&Table,
Table.tax, a+1));
//          count_order += 2; // 增加两个订单的数量
//      }
//      else if (date1 <= n) // 如果只有第一个元素满足条件
//      {
//          // 获取第一个元素的价格和折扣, 并计算相关的值
//          int price1 = getdata(&Table, Table.price, a);
//          double discount1 = getdata(&Table, Table.discount, a);
//          double oneMdiscount1 = 1 - discount1;
//          double PandOneDis1 = price1 * oneMdiscount1;
//          // 累加相关的值
//          sum_qty += getdata(&Table, Table.quantity, a);
//          sum_base_price += price1;
//          sum_discount += discount1;
//          sum_disc_price += PandOneDis1;
//          sum_charge += PandOneDis1 * (1 + getdata(&Table,
Table.tax, a));
//          count_order++; // 增加一个订单的数量
//      }
//      else if (date2 <= n) // 如果只有第二个元素满足条件
//      {
//          // 获取第二个元素的价格和折扣, 并计算相关的值
//          int price2 = getdata(&Table, Table.price, a + 1);
//          double discount2 = getdata(&Table, Table.discount, a +
1);

```

```

//      double oneMdiscount2 = 1 - discount2;
//      double PandOneDis2 = price2 * oneMdiscount2;
//      // 累加相关的值
//      sum_qty += getdata(&Table, Table.quantity, a + 1);
//      sum_base_price += price2;
//      sum_discount += discount2;
//      sum_disc_price += PandOneDis2;
//      sum_charge += PandOneDis2 * (1 + getdata(&Table,
Table.tax, a+1));
//      count_order++; // 增加一个订单的数量
//  }
// }
// // 处理剩余的元素，如果有的话
// for (; a < table_num; a++)
// {
//     int date = getdata(&Table, Table.date, a);
//     if (date <= n)
//     {
//         price = getdata(&Table, Table.price, a);
//         discount = getdata(&Table, Table.discount, a);
//         oneMdiscount = 1 - discount;
//         PandOneDis = price * oneMdiscount;
//         //tax =
//         sum_qty += getdata(&Table, Table.quantity, a);
//         sum_base_price += price;
//         sum_discount += discount;
//         sum_disc_price += PandOneDis;
//         sum_charge += PandOneDis * (1 + getdata(&Table,
Table.tax, a));
//         count_order++;
//     }
// }
// SumQuantity = sum_qty;
// SumBasePrice = sum_base_price;
// SumDiscPrice = sum_disc_price;
// SumCharge = sum_charge;
// AvgQuantity = SumQuantity / table_num;
// AvgPrice = SumBasePrice / table_num;
// AvgDiscount = sum_discount / table_num;
// total = count_order;
// }

```

```

void calculate2(int n)      // 未经过循环展开的 simd 算法 不如不加 simd
{

```



```

// 定义一些变量，用于存储累加和平均值的结果
long long sum_qty = 0, sum_base_price = 0;
double sum_disc_price = 0, sum_charge = 0, sum_discount = 0, avg_qty
= 0;
double avg_price = 0, avg_disc = 0;
size_t count_order = 0;
int table_num = Table.num; // 获取表格的行数
int price = 0; // 用于存储每一行的价格
double discount = 0; // 用于存储每一行的折扣
double oneMdiscount = 0; // 用于存储1-折扣的值
double PandOneDis = 0; // 用于存储价格乘以1-折扣的值
//double tax = 0; // 用于存储每一行的税率，但是这里没有用到，所以注释掉了

// 使用OpenMP 的指令来开启并行区域，并指定共享变量和私有变量
#pragma omp parallel shared(sum_qty, sum_base_price, sum_disc_price,
sum_charge, sum_discount, count_order) private(price, discount,
oneMdiscount, PandOneDis)
{
    // 使用OpenMP 的指令来对循环进行并行化，并指定循环变量、调度策略和归
约操作

    #pragma omp for schedule(static) reduction(+:sum_qty,
sum_base_price, sum_disc_price, sum_charge, sum_discount, count_order)
    for (int a = 0; a < table_num; a++) // 遍历表格的每一行
    {
        int date = getdata(&Table, Table.date, a); // 获取第a 行的日
期

        if (date <= n) // 如果日期小于等于n，说明符合条件
        {
            price = getdata(&Table, Table.price, a); // 获取第a 行的
价格

            discount = getdata(&Table, Table.discount, a); // 获取第
a 行的折扣

            oneMdiscount = 1 - discount; // 计算1-折扣的值
            PandOneDis = price * oneMdiscount; // 计算价格乘以1-折扣
的值

            //tax =
            sum_qty += getdata(&Table, Table.quantity, a); // 将第a
行的数量累加到sum_qty 中
            sum_base_price += price; // 将第a 行的价格累加到
sum_base_price 中
            sum_discount += discount; // 将第a 行的折扣累加到
sum_discount 中
            sum_disc_price += PandOneDis; // 将第a 行的价格乘以1-折扣
的值累加到sum_disc_price 中

```

```

        sum_charge += PandOneDis * (1 + getdata(&Table,
Table.tax, a)); // 将第a 行的价格乘以 1-折扣再乘以 1+税率的值累加到
sum_charge 中
        count_order++; // 将符合条件的订单数量加一
    }
}
}
// 将累加的结果赋值给全局变量
SumQuantity = sum_qty;
SumBasePrice = sum_base_price;
SumDiscPrice = sum_disc_price;
SumCharge = sum_charge;
// 计算平均值，并赋值给全局变量
AvgQuantity = SumQuantity / table_num;
AvgPrice = SumBasePrice / table_num;
AvgDiscount = sum_discount / table_num;
// 将符合条件的订单数量赋值给全局变量
total = count_order;
}

// void calculate2(int n)    //拥有循环展开的simd 效果很差，不如不改
// {
//     Long Long sum_qty = 0, sum_base_price = 0;
//     double sum_disc_price = 0, sum_charge = 0, sum_discount = 0,
avg_qty = 0;
//     double avg_price = 0, avg_disc = 0;
//     size_t count_order = 0;
//     int table_num = Table.num;
//     int price = 0;
//     double discount = 0;
//     double oneMdiscount = 0;
//     double PandOneDis = 0;
//     //double tax = 0;
//     // 使用OpenMP 的指令来开启并行区域，并指定共享变量和私有变量
//     #pragma omp parallel shared(sum_qty, sum_base_price,
sum_disc_price, sum_charge, sum_discount, count_order) private(price,
discount, oneMdiscount, PandOneDis)
//     {
//         // 使用OpenMP 的指令来对循环进行并行化，并指定循环变量、调度策略和
归约操作
//         #pragma omp for schedule(static) reduction(+:sum_qty,
sum_base_price, sum_disc_price, sum_charge, sum_discount, count_order)
//         // 2*2 循环展开的思想是每次迭代处理两个元素，从而减少循环次数和循环
控制的开销

```

```

//          // 为了保证正确性，需要考虑 table_num 是否为偶数，以及循环边界的处理
//          // 另外，为了利用 simd 指令，需要将一些变量定义为 __m128d 类型，表示
//          // 一个 128 位的向量，可以存储两个双精度浮点数
//          int half_table_num = table_num / 2; // 计算 table_num 的一半，
//          向下取整
//          double one_plus_tax = 1 + getdata(&Table, Table.tax, 0); //
//          假设税率是固定的，只需获取一次
//          __m128d one_plus_tax_v = _mm_set1_pd(one_plus_tax); // 将税率
//          转换为向量类型
//          int a = 0; // 循环变量
//          // 处理前半部分的元素，每次两个
//          for (; a < half_table_num * 2; a += 2)
//          {
//          int date1 = getdata(&Table, Table.date, a); // 获取第一个
//          元素的日期
//          int date2 = getdata(&Table, Table.date, a + 1); // 获取第
//          二个元素的日期
//          if (date1 <= n && date2 <= n) // 如果两个元素都满足条件
//          {
//          //          // 获取两个元素的价格和折扣，并转换为向量类型
//          //          __m128d price_v = _mm_set_pd(getdata(&Table,
//          Table.price, a), getdata(&Table, Table.price, a + 1));
//          //          __m128d discount_v = _mm_set_pd(getdata(&Table,
//          Table.discount, a), getdata(&Table, Table.discount, a + 1));
//          //          // 计算 1- 折扣的值和价格乘以 1- 折扣的值
//          //          __m128d oneMdiscount_v = _mm_sub_pd(_mm_set1_pd(1),
//          discount_v);
//          //          __m128d PandOneDis_v = _mm_mul_pd(price_v,
//          oneMdiscount_v);
//          //          // 累加相关的值，使用 simd 指令进行加法和乘法
//          //          sum_qty += getdata(&Table, Table.quantity, a) +
//          getdata(&Table, Table.quantity, a + 1);
//          //          sum_base_price += _mm_cvtsd_f64(_mm_hadd_pd(price_v,
//          price_v)); // 使用水平加法指令将向量中的两个元素相加，然后转换为标量类型
//          //          sum_discount += _mm_cvtsd_f64(_mm_hadd_pd(discount_v,
//          discount_v));
//          //          sum_disc_price +=
//          _mm_cvtsd_f64(_mm_hadd_pd(PandOneDis_v, PandOneDis_v));
//          //          sum_charge +=
//          _mm_cvtsd_f64(_mm_hadd_pd(_mm_mul_pd(PandOneDis_v,
//          _mm_set1_pd(1+getdata(&Table, Table.tax, a))), _mm_mul_pd(PandOneDis_v,
//          _mm_set1_pd(1+getdata(&Table, Table.tax, a)))));
//          //          count_order += 2; // 增加两个订单的数量

```

```

//      }
//      else if (date1 <= n) // 如果只有第一个元素满足条件
//      {
//          // 获取第一个元素的价格和折扣，并计算相关的值
//          price = getdata(&Table, Table.price, a);
//          discount = getdata(&Table, Table.discount, a);
//          oneMdiscount = 1 - discount;
//          PandOneDis = price * oneMdiscount;
//          //tax =
//          // 累加相关的值
//          sum_qty += getdata(&Table, Table.quantity, a);
//          sum_base_price += price;
//          sum_discount += discount;
//          sum_disc_price += PandOneDis;
//          sum_charge += PandOneDis * one_plus_tax;
//          count_order++; // 增加一个订单的数量
//      }
//      else if (date2 <= n) // 如果只有第二个元素满足条件
//      {
//          // 获取第二个元素的价格和折扣，并计算相关的值
//          price = getdata(&Table, Table.price, a + 1);
//          discount = getdata(&Table, Table.discount, a + 1);
//          oneMdiscount = 1 - discount;
//          PandOneDis = price * oneMdiscount;
//          //tax =
//          // 累加相关的值
//          sum_qty += getdata(&Table, Table.quantity, a + 1);
//          sum_base_price += price;
//          sum_discount += discount;
//          sum_disc_price += PandOneDis;
//          sum_charge += PandOneDis * one_plus_tax;
//          count_order++; // 增加一个订单的数量
//      }
//      }
//      // 处理剩余的元素，如果有的话
//      for (; a < table_num; a++)
//      {
//          int date = getdata(&Table, Table.date, a);
//          if (date <= n)
//          {
//              price = getdata(&Table, Table.price, a);
//              discount = getdata(&Table, Table.discount, a);
//              oneMdiscount = 1 - discount;
//              PandOneDis = price * oneMdiscount;

```

```
//          //tax =
//          sum_qty += getdata(&Table, Table.quantity, a);
//          sum_base_price += price;
//          sum_discount += discount;
//          sum_disc_price += PandOneDis;
//          sum_charge += PandOneDis * (1 + getdata(&Table,
Table.tax, a));
//          count_order++;
//      }
//  }
//  SumQuantity = sum_qty;
//  SumBasePrice = sum_base_price;
//  SumDiscPrice = sum_disc_price;
//  SumCharge = sum_charge;
//  AvgQuantity = SumQuantity / table_num;
//  AvgPrice = SumBasePrice / table_num;
//  AvgDiscount = sum_discount / table_num;
//  total = count_order;
// }
```