2020/04/03_第5课_goto语句、循环结构、折半查找

笔记本: C
创建时间: 2020/4/3 星期五 14:57
作者: ileemi
标签: 三种循环及折半查找

---

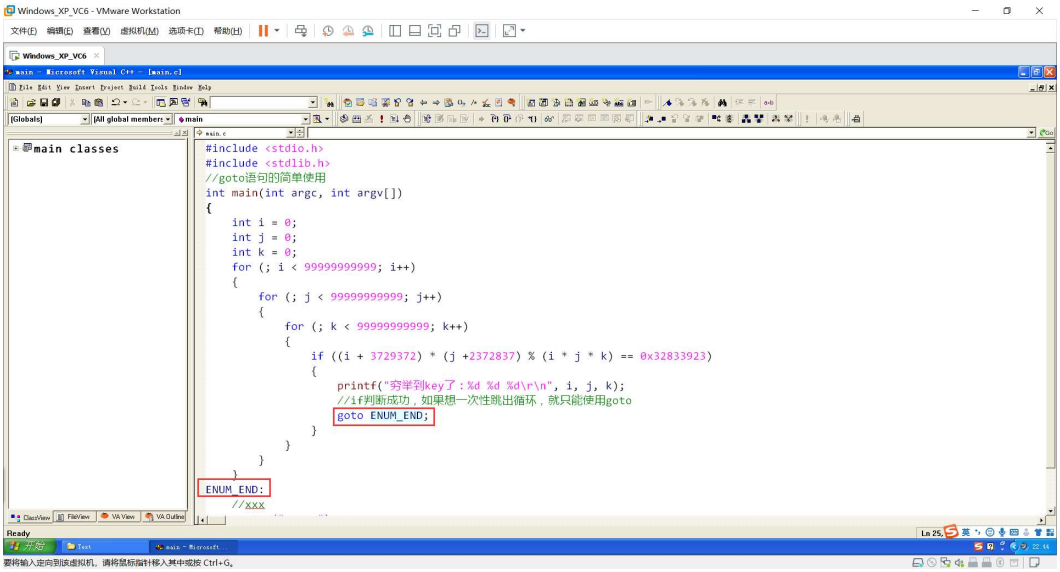
goto语句

优点:

- 可以更改程序执行的正常顺序,可以跳转到程序的任何部分
- 可以从嵌套很深的地方跳转出来（多层循环可以一口气从最内层跳到最外层）

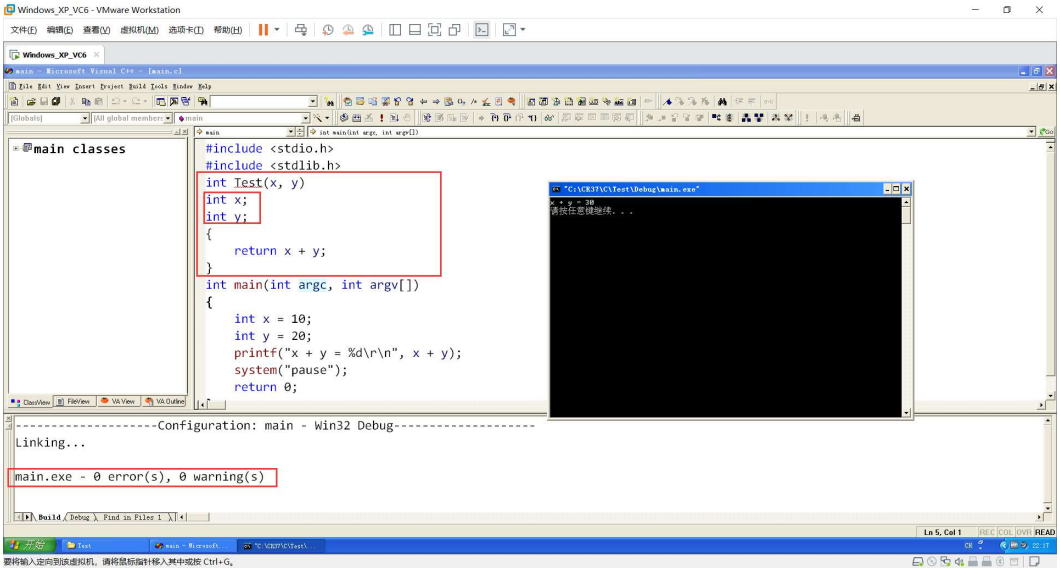
缺点：容易导致流程结构混乱

使用场景:



goto语句的使用，以后只能在多重循环需要跳出时才能使用，

VC++ 6.0时代的一个语法（现阶段只有VC++ 6.0IDE支持这样的写法）：



代码规划

一行只执行一条语句，每条语句不超过三句（含运算），对于长表达式，应该按运算逻辑或运算优先级作出拆分多条语句，原则时运算逻辑优与运算优先级。

三种循环的使用

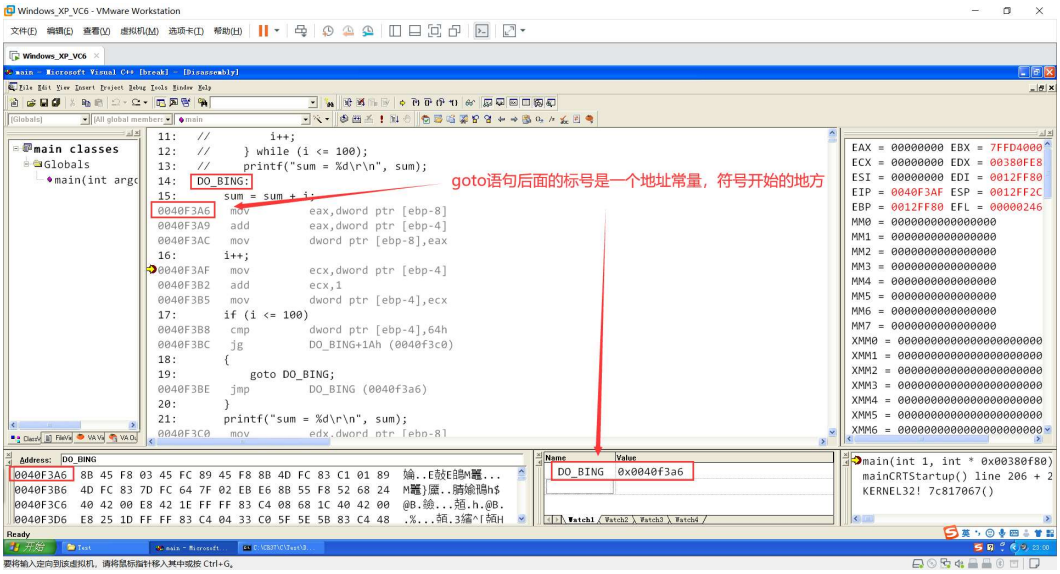
do while循环

先吃饭后买单

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i = 0;
    int sum = 0;
    do
    {
        sum = sum + i;
        i++;
    }while(i <= 100);
    printf("sum = %d\r\n", sun);
    system("pause");
    return 0;
}
```

使用goto语句模拟do while循环

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i = 0;
    int sum = 0;
    //使用goto语句模拟do while 循环
 DO_BING:
    sum = sum + i;
    i++;
    if(i <= 100)
    {
        goto DO_BING;    //DO_BING是个地址常量，DO_BING符号化一个地址常量 0x0040F3A6
    }
    printf("sum = %d\r\n", sun);
    system("pause");
    return 0;
}
```



while循环

先买单后吃饭

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i = 0;
    int sum = 0;
    while(i <= 100)      //条件为真，继续执行
```

```
    {
        sum = sum + i;
        i++;
    }
    printf("sum = %d\r\n", sun);
    system("pause");
    return 0;
}
```

使用goto语句模拟while循环

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i = 0;
    int sum = 0;
    //使用goto语句模拟do while 循环
WHILE_BING:
    if(i <= 100)
    {
        sum = sum + i;
        i++;
        goto WHILE_BING;
    }
    printf("sum = %d\r\n", sun);
    system("pause");
    return 0;
}
```

for循环

先买单后吃饭

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i = 0;
    int sum = 0;
    //参数1（初值）可以省略，需要在for循环开始之前进行定义
    //参数2（终值）没有终值，该程序就会一直执行下去
    //参数3（步长）的地址在循环体的上面
    for(; i <= 100; i++)
    {
```

```
        sum = sum + i;
    }
    printf("sum = %d\r\n", sun);
    system("pause");
    return 0;
}
```

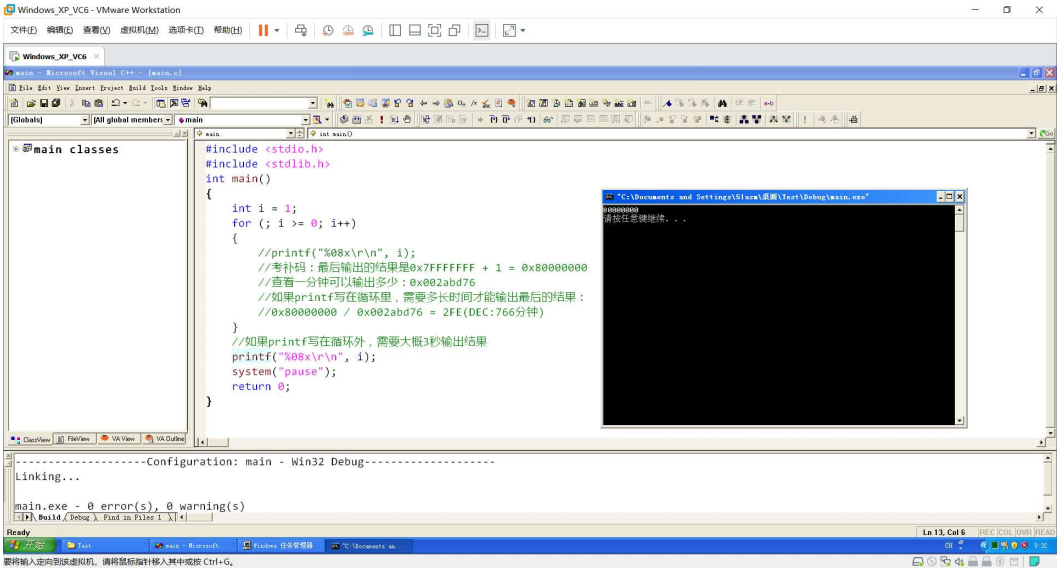
goto语句模拟 for 循环

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i = 0;
    int sum = 0;
FOE_BING:
    i++;
    if(i <= 100)
    {
        sum = sum + i;
        goto FOR_BING;
    }
    printf("sum = %d\r\n", sun);
    system("pause");
    return 0;
}
```

printf 执行效率问题

printf 需要做的事情比较多，格式化，输出，访问标准输出设备，统计返回值，可以发现printf 的效率"开支"是比较大的。
Debug 不会优化代码的执行效率

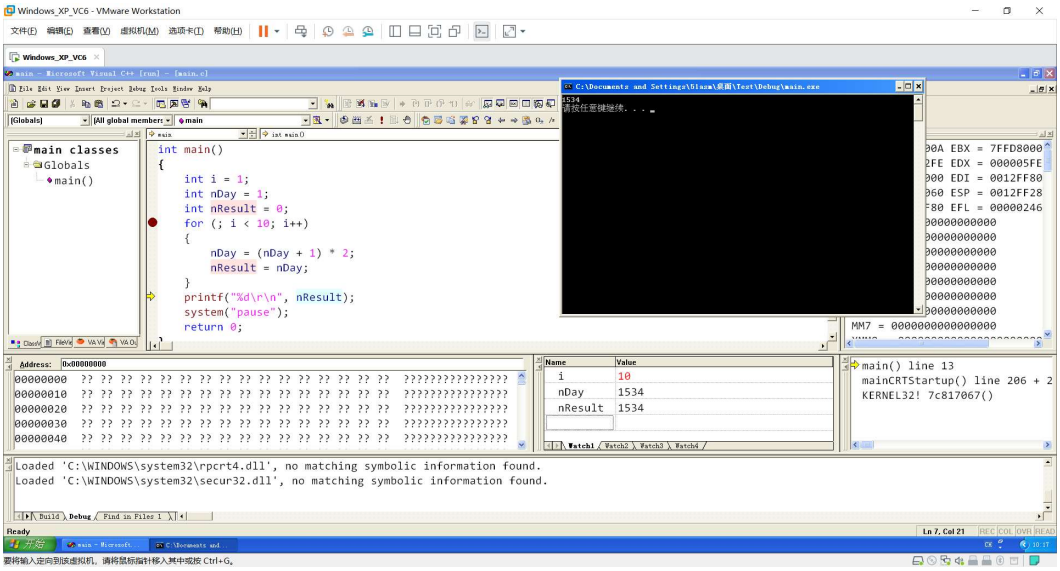
Release 会优化代码的执行效率



猴子吃桃

题目：猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个，第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃前一天剩下的一半零一个。到第10天早上想再吃时，见只剩下一个桃子了。求第一天共摘多少个桃子?

正常版：线性阶



用等比数列求：线性阶

运行高中数学中的等比数列可以知道，其公比为1/2

常量阶：不管规模有多大，代价永远恒定

公式法:

再对问题分析，假定第n天吃之前还有桃子F_n个我们有:

$F_{n-1} = 2 \times (F_n + 1)$

即:

$$
\begin{aligned}
F_n &= \frac{F_{n-1}}{2} - 1 \\
&= \frac{\frac{F_{n-2}}{2} - 1}{2} - 1 \\
&= \frac{F_{n-2}}{2^2} - (1 + \frac{1}{2^1}) \\
&= ... \\
&= \frac{F_1}{2^{n-1}} - (1 + \frac{1}{2^1} + ... + \frac{1}{2^{n-2}}) \\
&= \frac{F_1}{2^{n-1}} - 2 \times (1 - \frac{1}{2^{n-1}}) \\
&= \frac{F_1 + 2}{2^{n-1}} - 2
\end{aligned}
$$

所以: $F_1 = F_n \times 2^{n-1} + 2^n - 2$

代码实现:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
    //正常版，线性阶
    int nDay = 1;
    int i = 1;
    int nResult = 0;
    int nCount = 1;
    int nDayCount = 10;
    for (; i < 10; i++)
    {
        nDay = (nDay + 1) * 2;
        nResult = nDay;
    }
    printf("猴子第一天一共摘了 %d 个桃子\r\n\n", nResult);

    //常数阶
    while (nDayCount >= 1)
    {
        printf("猴子第 %d 天还有 %d 个桃子\r\n",nCount++,(int)pow(2, nDayCount-1) + (int)pow(2, nDayCount) - 2);
        nDayCount--;
    }
    system("pause");
```

```
    return 0;
}
```

折半查找

时间复杂度 O(logn) 对数阶

该算法的要求：

- 必须采用顺序存储结构
- 必须按关键字大小有序排列

代码实现：

```
/*
折半查找的简单使用
*/
#include <stdio.h>
int main()
{
    int nKey = 0;
    int nBegin = 0;
    int nMit = 0;
    int nEnd = 0;
    int i = 0;
    int nAry[10] = { 1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
    printf("数组内的个元素值分别为：\r\n");
    for (; i < 10; i++)
    {
        printf("%d  ", nAry[i]);
    }
    printf("\r\n");
```

```
    nEnd = sizeof(nAry) / sizeof(int) - 1;
    printf("请输入你要查找的数值：");
    scanf("%d", &nKey);
    while (nBegin <= nEnd)
    {
        nMit = (nBegin + nEnd) / 2;
        printf("nBegin = %d, nMit = %d, nEnd = %d\r\n", nBegin, nMit,
nEnd);
        if (nKey > nAry[nMit])
        {
            nBegin = nMit + 1;
        }
        else if (nKey < nAry[nMit])
        {
            nEnd = nMit - 1;
        }
        else
        {
            printf("%d 再数组索引为 %d 的位置上\r\n", nKey, nMit);
            break;
        }
    }
    if (nBegin > nEnd)
    {
        printf("你要查找的数值 %d 未找到\r\n", nKey);
    }
    return 0;
}
```

算法的五个基本特性

输入、输出、有穷性、确定性和可行性