第 3 章 选择与循环

（视频讲解：1.5 小时）

裁缝做衣服有尺子和圆规，程序员控制程序执行逻辑使用选择和循环，接下来我们来看下 C 语言为我们提供的选择语句，循环语句。学习本章，你可以掌握

- **选择结构程序设计**
- **循环结构程序设计**

3.1 选择结构程序设计

3.1.1 关系表达式与逻辑表达式

在讲选择语句之前，我们首先来练习一下关系表达式与逻辑表达式，**算术运算符优先级高于关系运算符，关系运算符优先级高于逻辑与与逻辑或，相同优先级运算符从左至右进行结合**，下面有一个这样的例子，**5>3&&8<4-!0** ，这个表达式最终的值为多少呢，计算如图 3.1.1-1 所示

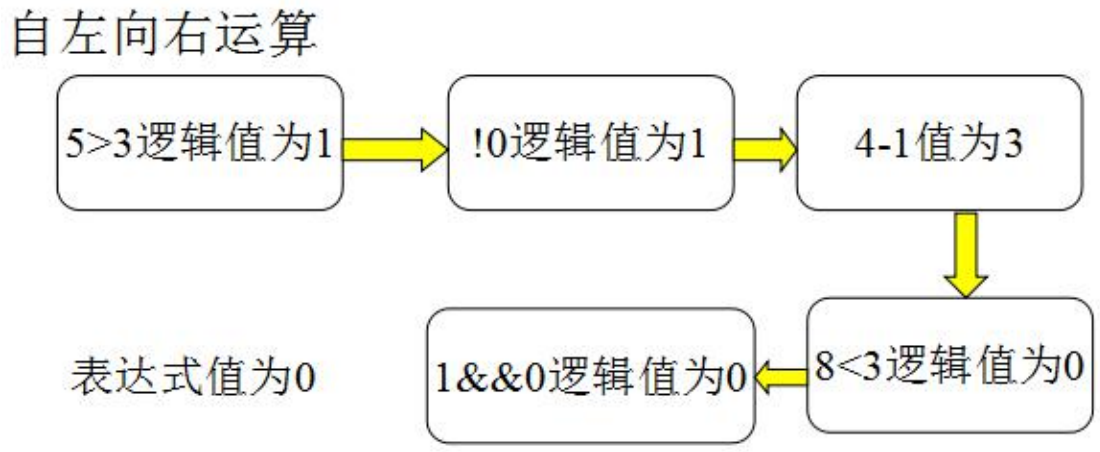

图 3.1.1-1

下面我们再来看一个计算闰年的例子，闰年是能被 4 整除，但不能被 100 整除或者能被 4 整除，又能被 400 整除。year%4==0&&year%100!=0||year%400==0，网上一些人写成这样(year%4==0&&year%100!=0)||year%400==0，虽然可以，但是括号是多余的。

3.1.2 if 语句

假如你打开你的衣柜，拿出最上面一件衣服，你会去判断是否是你想穿的，是就做穿上的动作，不是，就会去找其他衣服。在计算机中，我们用 if 判断语句来实现这样的效果，if 判断条件（表达式）为真，就执行某个语句，如果为假，就不执行这个语句。当然也可以 if 判断条件（表达式）为真，就执行某个语句，如果为假，用 else 分支执行另一个语句，原理图如图 3.1.2-1 和 3.1.2-2

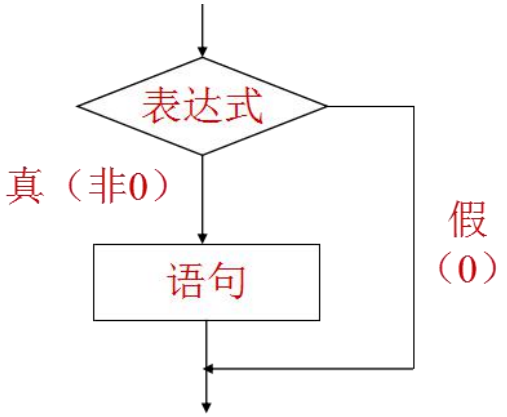

图 3.1.2-1

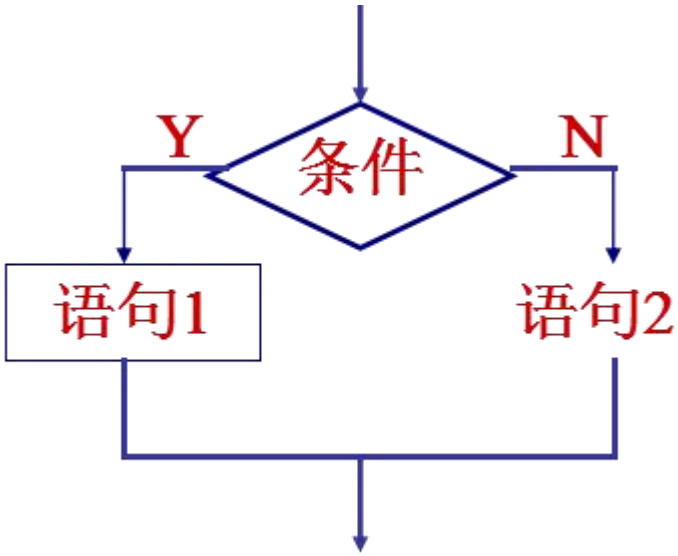

图 3.1.2-2

下面我们来看一个实际的例子，输入值大于 0 时，会打印 i is bigger than 0，当输入值小于等于 0 时，会打印 i is not bigger than 0。但是 if 后是不可以加分号的，如果有 else，加分号会编译不通，如果没有 else，加分号会导致 i 无论为何值，都会执行语句 i is bigger than 0。

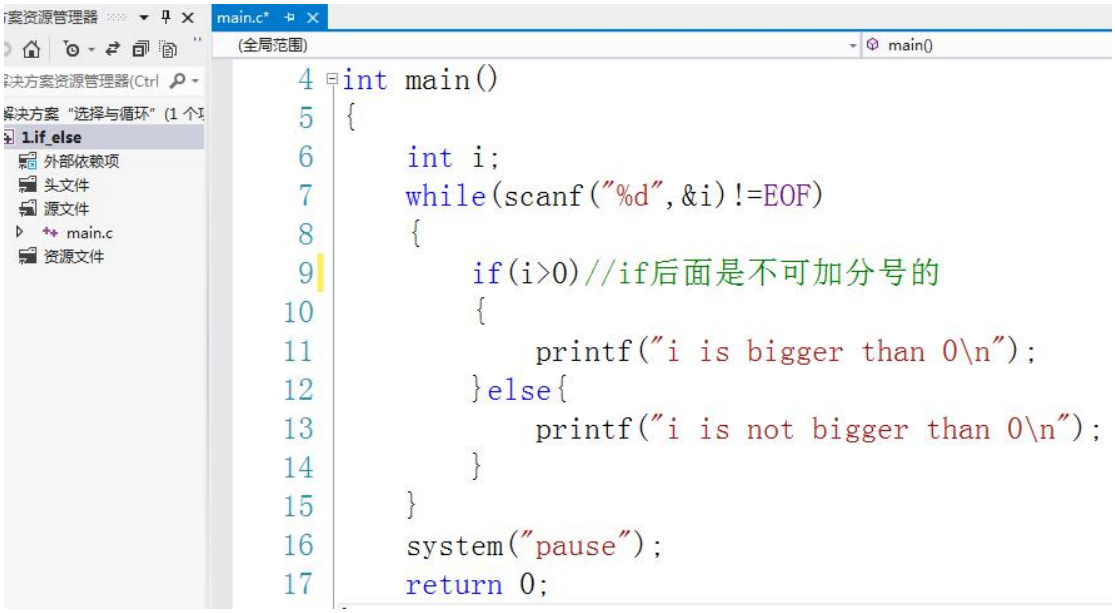

图 3.1.2-3

同时 if 也支持 if 与 else if 功能，如图 3.1.2-4，无论有多少个语句，或者 else if，程序只会执行其中一个语句，例如下面一个用电量的例子，如果用的度数越多，则单价越高，但是最终 cost 只会被赋值一次。同时 if 也支持多层嵌套，在 if 语句中又包含一个或多个 if 语句称为 if 语句的嵌套，如图 3.1.2-5。

```
if (number>500)cost=0.15;
        else if(number>300)cost=0.10;
                else if(number>100)cost=0.075;
                        else if(number>50)cost=0.05;
                            else    cost=0;
```

```
if（表达式1）语句1
else if(表达式2)语句2
else if(表达式3)语句3
    ……
else if(表达式m)语句m
else  语句n
```

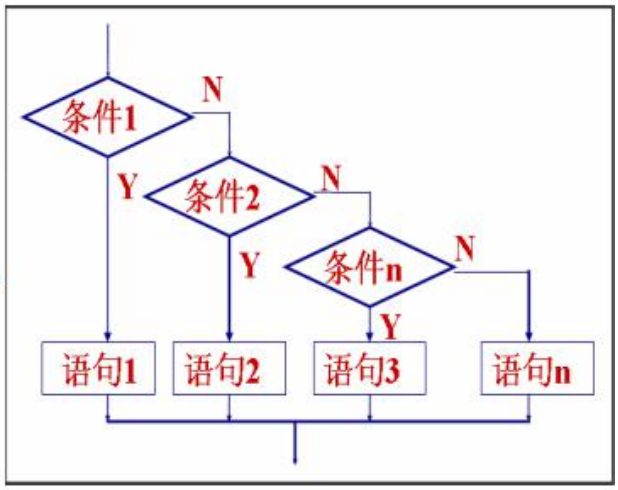

图 3.1.2-4

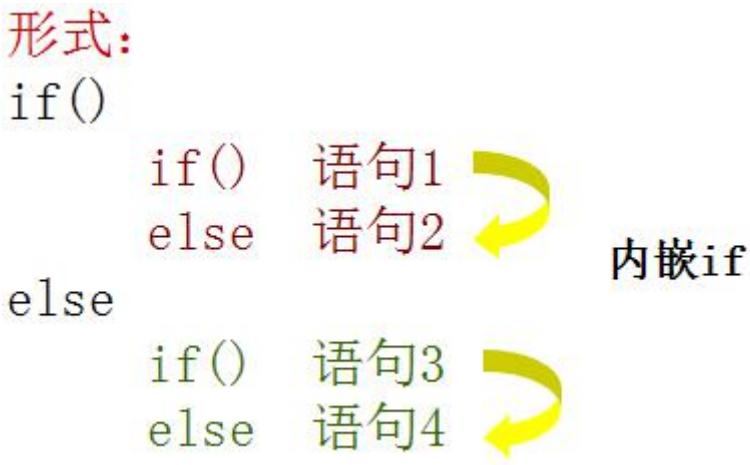

图 3.1.2-5

容易出错场景：

当 if 语句嵌套出现时，就会出现“悬空的 else”问题。例如，在下面的例子中，你认为 else 子句从属于哪一个 if 语句呢？

```
if(i>1)
        If(i<10)
           Printf(“i>1 and i<10\n”);
      else
```

```
            Printf(“no,they are not\n”);
```

这里故意把 else 子句以奇怪的方式缩进，就是不给你任何提示。这个问题和其他绝大多数语言一样，就是 else 子句从属于最靠近它的不完整的 if 语句。如果你想让它从属于第一个子句，你可以用一个花括号把他包围在一个代码块之内，如下所示：

```
if(i>1){
        if(i<10)
            Printf(“i>1 and i<10\n”);
 }
    else
        Printf(“no,they are not\n”);
```

提醒：在 if 语句中的语句列表前后加上大括号，防止后面维护代码的人，不小心加了一句代码后，实际并没有被包含在 if 语句中。

3.1.3 switch 语句

当一个变量我们判断其等于几个值，或者几十个值时，使用 if 和 else if 会导致 else if 分支非常多，这种情况我们将使用 switch 语句，switch 语句语法格式如下：

```
switch （表达式）
   {case  常量表达式 1 ： 语句 1
    case  常量表达式 2 ： 语句 2
    …
    case  常量表达式 n ： 语句 n
          default       ： 语句 n ＋ 1
         }
```

我们来看一个日期的例子【例 3.1.3-1】，输入一个年份，月份，判断是几月，然后打印对应月份的天数，如果是闰年，2 月输出 29 天，代码如下，对应电子附件项目名称为《switch 月份 1》，为什么不是从 1 月到 12 月的顺序呢，是想让大家明白 switch 匹配并不需要值的大小要从小到大，等于哪个值，就会执行对应 case 后的语句，每一句后需要加 break，代表不在匹配下面的 case，switch 结束。

【例 3.1.3-1】switch 的使用

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int mon,year;
    while(scanf("%d%d",&year,&mon)!=EOF)
    {
        switch (mon)
        {
        case 2:printf("mon=%d is %d
days\n",mon,28+(year%4==0&&year%100!=0||year%400==0));break;
        case 1:printf("mon=%d is 31days\n",mon);break;
        case 3:printf("mon=%d is 31days\n",mon);break;
        case 5:printf("mon=%d is 31days\n",mon);break;
```

```
        case 7:printf("mon=%d is 31days\n",mon);break;
        case 8:printf("mon=%d is 31days\n",mon);break;
        case 10:printf("mon=%d is 31days\n",mon);break;
        case 12:printf("mon=%d is 31days\n",mon);break;
        case 4:printf("mon=%d is 30days\n",mon);break;
        case 6:printf("mon=%d is 30days\n",mon);break;
        case 9:printf("mon=%d is 30days\n",mon);break;
        case 11:printf("mon=%d is 30days\n",mon);break;
        default:
            printf("error mon\n");
        }
    }
    system("pause");
    return 0;
    }
```

基于 switch 不加 break，会继续匹配下面的 case，我们对【例 3.1.3-1】进行了优化，代码如【例 3.1.3-2】，对应电子附件项目名称为《switch 月份 2》，输入年份月份，执行效果和上面代码执行效果一致，原理是匹配到 1,3,5,7,8,10,12 的任何一个，就不会再拿 mon 与 case 后的值比较，而是执行语句 printf("mon=%d is 31days\n",mon)，执行完毕后，break，跳出 switch，每一个 switch 最后加入 default 的目的是加入所有 case 都未匹配，进行错误打印输出，或者一些提醒，能够让程序员快速掌握执行的执行情况。

【例 3.1.3-2】日期实例改进版

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int mon,year;
    while(scanf("%d%d",&year,&mon)!=EOF)
    {
        switch (mon)
        {
        case 2:printf("mon=%d is %d
days\n",mon,28+(year%4==0&&year%100!=0||year%400==0));break;
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:printf("mon=%d is 31days\n",mon);break;
        case 4:
        case 6:
        case 9:
```

```
        case 11:printf("mon=%d is 30days\n",mon);break;
        default:
            printf("error mon\n");
        }
    }
    system("pause");
    return 0;
    }
```

3.2 循环结构程序设计

3.2.1 goto 语句

在学校老师都讲不用掌握 goto，这种说法是不对的，goto 才是循环的本质，对应于汇编中的 jmp 跳转，C 中的 while，do while 和 for 在程序编译时，都要拆解为汇编的 jmp。**goto 语句——无条件转向语句，使用方法为 goto 语句标号；**语句标号的命名规则与 C 语言变量的命名规则一致，例如：goto label_1; 合法; goto 123；不合法。goto 的使用场景分为两种，一种是**向上跳转**，实现循环，一种是**向下跳转**，实现中间部分代码不执行。首先我们来看下 goto 向上跳转实现从 1 加到 100 的例子（图 3.2.1-1），因为 goto 是无条件跳转，所以我们会用 if 与 goto 配合使用，通过 if 判断，当 i 大于 100 时，就不再进行无条件跳转，所以在 goto lable 与 lable 之间，需要有 i++，让判断趋近于假。如果 i 小于等于 100，那么代码走到第 15 行时，就会跳转到第 10 行再次开始执行。

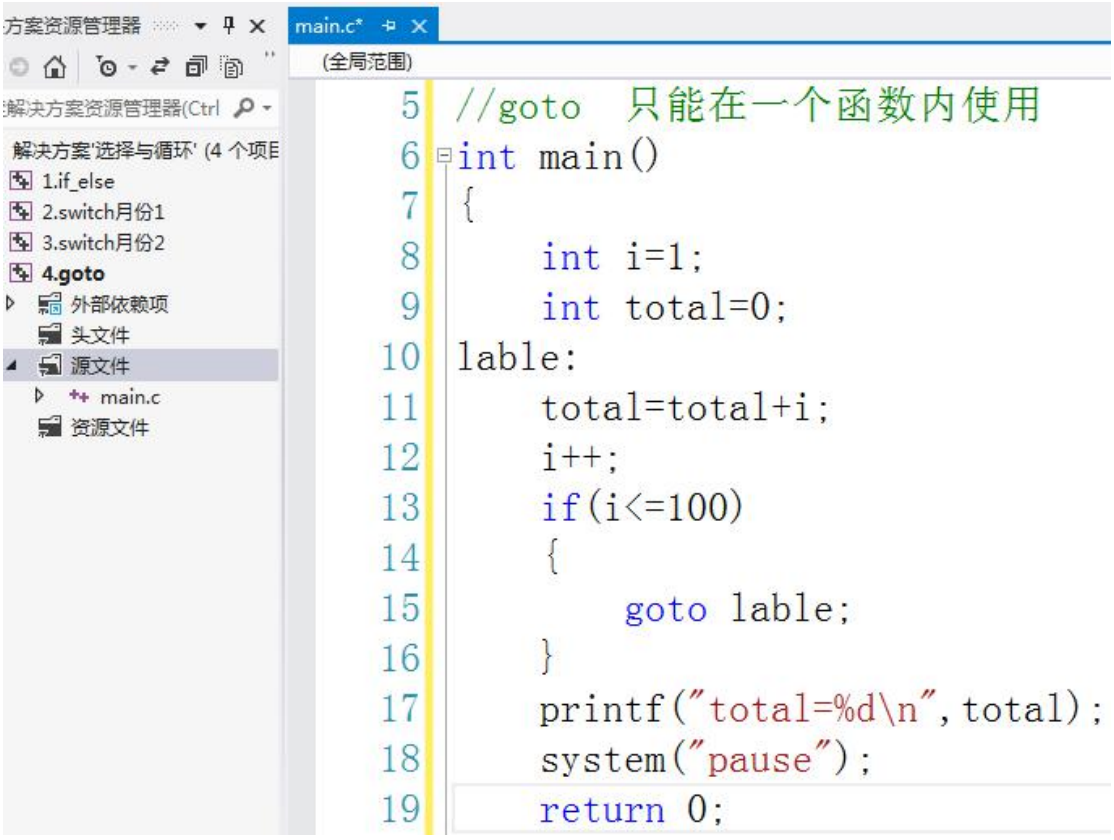

图 3.2.1-1

下面我们来看下 goto 向下跳转，如图 3.2.1-2，当判断 disk 输入值为 0 时，代表磁盘发生异常，就直接跳转到 lable_disk_error，Linux 内核编写中，大量采用这种手法，因为这样

编写首先代码编写设计难度低，同时编译或者执行效率高。

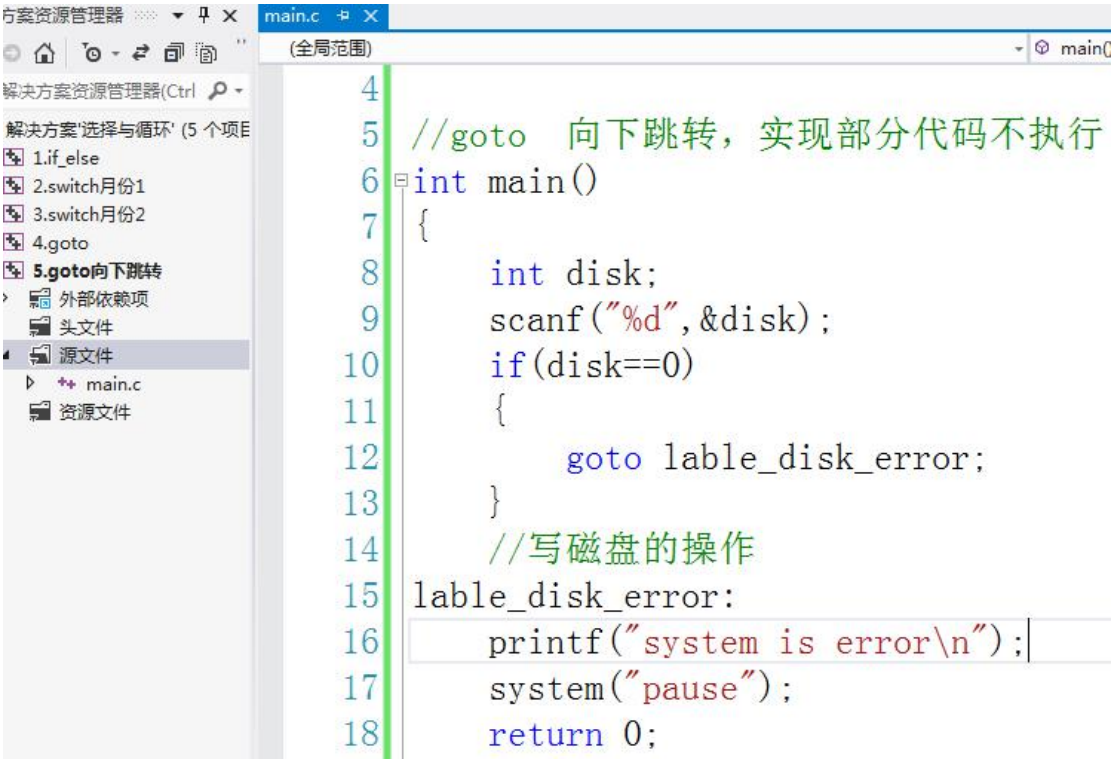

图 3.2.1-2

3.2.2 while 循环

while 语句用来实现“当型”循环结构，一般形式：while (表达式) 语句，当表达式为非 0 值时，执行 while 语句中的内嵌语句。其特点是:**先判断表达式，后执行语句**。如图 3.2.2-1 所示，表达式非 0，就会执行语句，从而实现语句多次执行的效果，但是程序不能死循环，在语句中需要有让表达式趋近于假的操作。

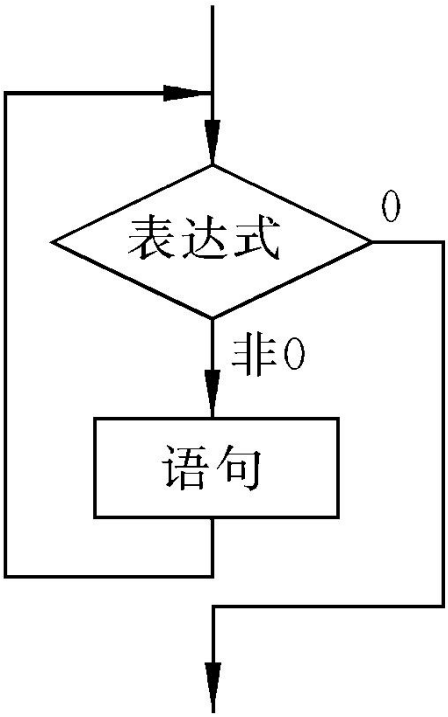

图 3.2.2-1

下面我们来看 while 实现从 1 加到 100 的效果，如图 3.2.2-2 所示，while 后面不能加分号，否则会编译通过，但是执行会发送死循环。通常我们会将 while 的语句用大括号括起来，就算语句只有一句，也会用大括号括起来，因为程序往往都会修改，使用括号可以让程序更加清晰，避免修改者往循环内添加语句发生出错。

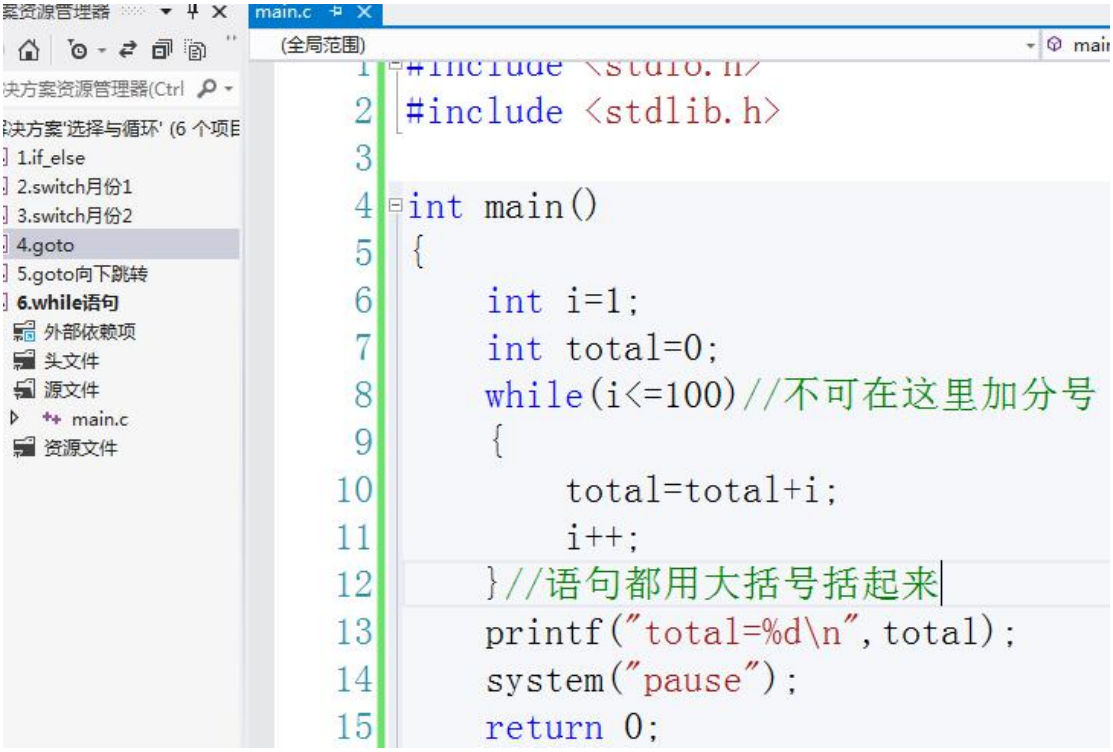

图 3.2.2-2

在 windows 下的 VS 集成开发环境我们可以用 fflush 或者 rewind 清空标准输入缓冲区，但是这些接口在 Linux 是不行的，我们如何自己写一个清空缓冲区的方法呢？答案如下：

While((ch=getchar())!=EOF && ch!='\n');

3.2.3 do while 循环

do-while 语句的特点:**先执行循环体，然后判断循环条件是否成立。**

一般形式: do

{

循环体语句

}

while (表达式);

执行过程：先执行一次指定的循环体语句，然后判别表达式，当表达式的值为非零(“真”)时，返回重新执行循环体语句，如此反复，直到表达式的值等于 0 为止，此时循环结束。图 3.2.3-1 是 do while 实现从 1 加到 100，do while 与 while 的差别是第一次不会判断表达式，也就是如果 i 的初值为 101，第一次依然会进入循环体。工作中用 do while 较少。

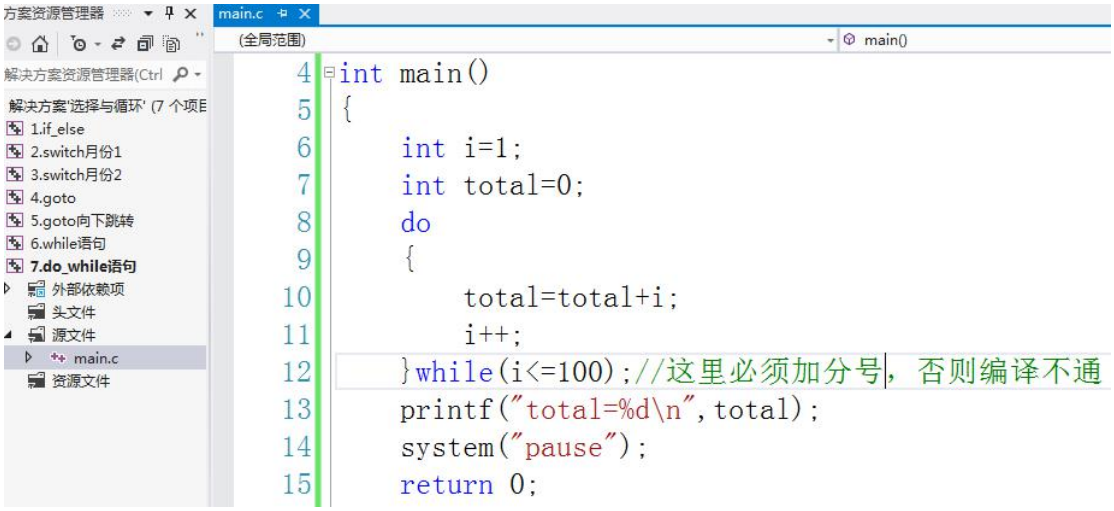

图 3.2.3-1

下面是某公司面试题：

【例 3.2.3-1】do while 的面试题

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i=1;
    do{
        printf("%d\n",i);
        i++;
        if(i<15)
        {
            continue;
        }
    }while(0);
    system("pause");
```

}

请问【例 3.2.3-1】执行输出是多少？执行输出如图 3.2.3-2 所示，只会输出 1，为什么呢？这是很多同学容易犯错的地方，以为 continue 就会跳过 while 内的表达式判断，其实并不会，continue 只会跳过其下面的代码部分，依然要进行 while 内表达式的判断，因为 while 内表达式为假，所以只会执行一次。

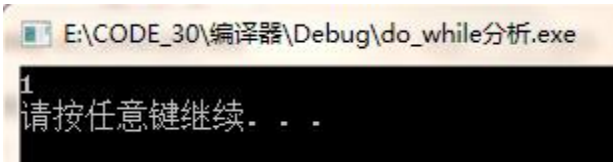

图 3.2.3-2

3.2.4 for 循环

C 语言中的 for 语句使用最为灵活，不仅可以用于循环次数已经确定的情况，而且可以用于循环次数不确定而只给出循环结束条件的情况，它完全可以代替 while 语句。

一般形式:

for(表达式 1；表达式 2；表达式 3) 语句

for 语句的执行过程（如图 3.2.4-1）：

(1) 先求解表达式 1。

(2) 求解表达式 2，若其值为真(值为非 0)，则执行 for 语句中指定的内嵌语句，然后执行下面第(3)步。若为假(值为 0)，则结束循环，转到第(5)步。

(3) 求解表达式 3。

(4) 转回上面第(2)步骤继续执行。

(5) 循环结束，执行 for 语句下面的语句

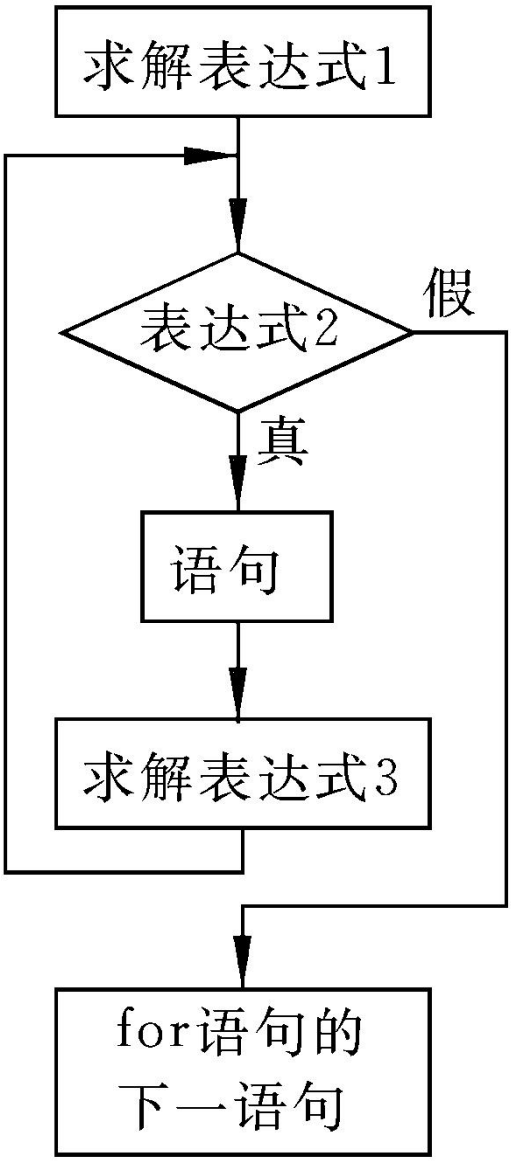

图 3.2.4-1

接下来我们来看一下通过 for 循环实现从 1 加到 100，for 中必须且只能有两个分号，用于分割表达式 1，表达式 2，表达式 3，当然表达式 1，表达式 2，表达式 3 均可省略，省略写法用的较少，如图 3.2.4-2，我们的 i=1,total=0 是表达式 1，也就是表达式 1 我们可以使用逗号运算符初始化多个变量；表达式 3 的操作作用是使表达式 2 趋近于假。

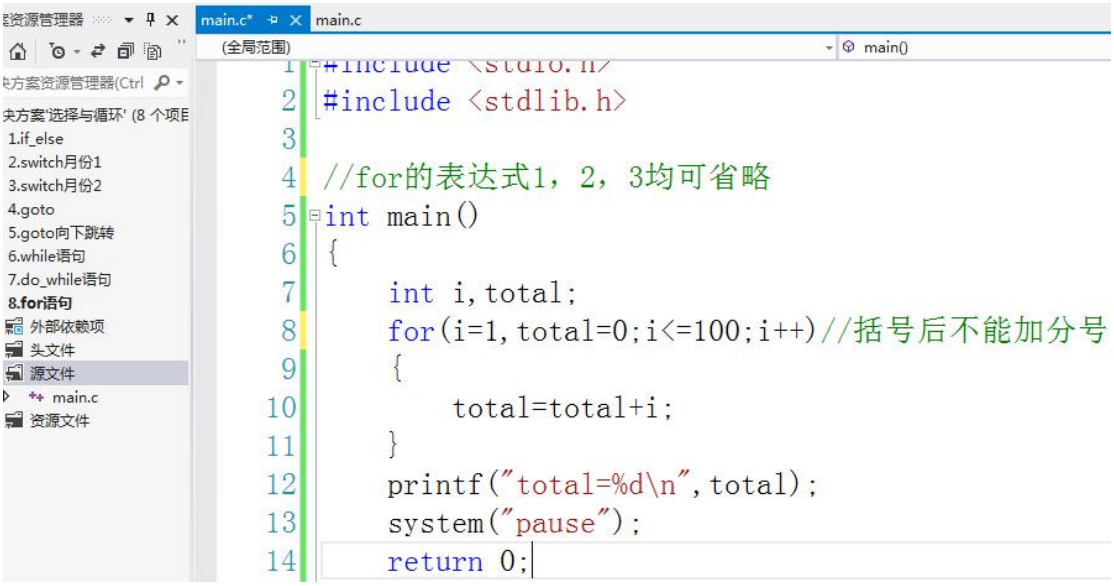

图 3.2.4-2

提醒：for 循环的可读性比 while 循环好，所以能用 for 的，不要强制改为 while

3.2.5 continue 语句

作用为结束本次循环，即**跳过循环体中下面尚未执行的语句**，接着进行下一次是否执行循环的判定.一般形式：continue;下面通过一个实例，例子的作用是对 1 到 100 之间的奇数进行求和，我们通过对图 3.2.4-1 的 for 循环进行改造，continue 后，执行的语句是 i++。**当 continue 使用于 while 和 do while 循环时，注意不要跳过让循环趋近于假的语句**。

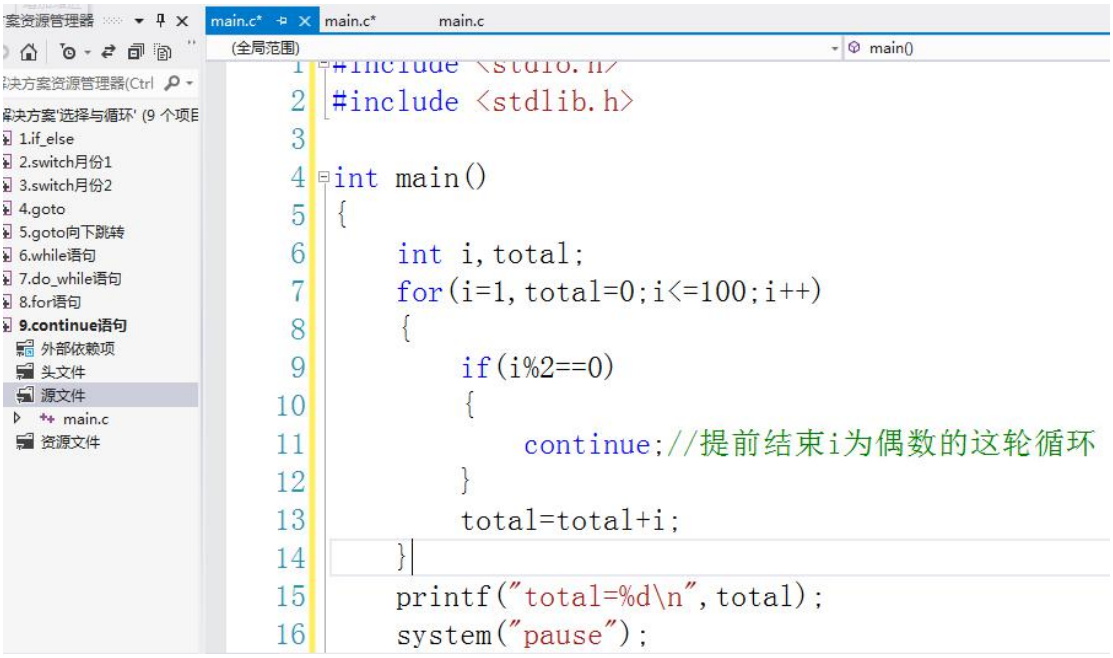

图 3.2.5-1

3.2.6 break 语句

与 continue 相比，break 语句则是**结束整个循环过程**，不再判断执行循环的条件是否成立。我们通过下面的实例（如图 3.2.6-1）来感受 break，例子的作用是从 1 开始加，当加的和大于 2000 时，就不再进行加，也就是结束 for 循环，同时打印这时 total 的值和 i 的值。

一旦执行 break，那么下一句要执行的是 printf("total=%d,i=%d\n",total,i)。break 也可以用于 while 和 do while，结束对应循环。

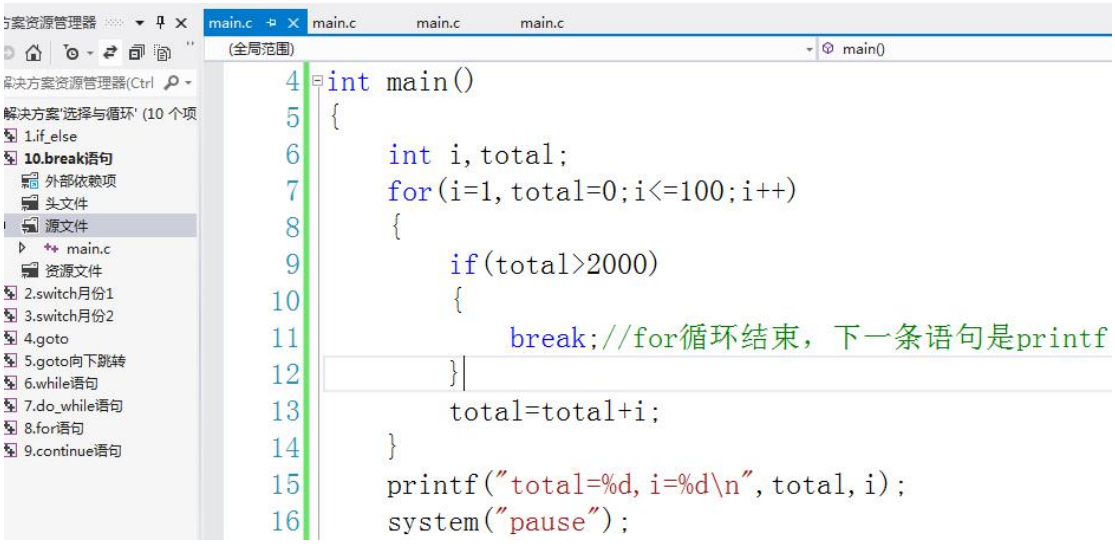

图 3.2.6-1