第3章 选择与迭代算法

【主要内容】

本章介绍处理问题时的选择思想和迭代思想，以及如何使用 C++ 实现选择和迭代算法。通过本章的学习，读者能够根据自己的问题，判断出是否应该采用选择或迭代处理方法，能够设计出相应的算法，并能够使用 C++ 提供的相应语句编写程序实现算法。本章还介绍 C++ 中实现程序流程转移的转向语句，并给出部分应用程序实例，这些实例涉及程序设计时迭代和二分法的基本思想，给读者以启发，激发读者利用计算思维解决更多、更复杂的问题的兴趣。同时，本章围绕“智能五子棋人机对战程序”中棋盘交互功能的实现，引导读者将选择与迭代结构应用于鼠标事件响应、坐标映射、落子有效性判断及回合状态管理等实际问题，体会程序控制结构在图形交互程序中的具体作用。

【重点】

- 选择思想和选择算法。
- 迭代思想和迭代算法。
- C++ 实现选择算法的 if 语句和 if…else 语句。
- C++ 实现迭代算法的 for 语句、while 语句。
- 在五子棋项目中运用选择与迭代结构实现交互逻辑。

【难点】

- 正确描述选择条件或循环条件。
- 综合运用选择与迭代结构完成具有实际交互功能的程序模块。

3.1 单路选择算法及其 C++ 实现

3.1 节
讲解视频

在解决一些问题时，人们都是按照一定的顺序进行的。有时，处理过程中的一些步骤需要根据不同的情况进行不同的处理，这种情况就是选择。

3.1.1 单路选择问题

单路选择问题是：如果某种情况发生，则就进行相应的处理；否则，不需要做任何处理。例如，天黑了，如果房间里的灯是关着的，就将灯打开。

【问题 3-1】 求一个实数的绝对值问题。

问题求解思路：变量 x 是用户输入的实数值，如果 $x<0$，$x=-x$。当 $x\geqslant 0$ 时，x 的值就是其绝对值，不需要做任何处理。解决该问题需要三步，其中第 2 步需要根据 x 的值选择进

行处理或不进行处理。

解决该问题的算法如表 3-1 所示。

表 3-1 求解问题 3-1 的算法

步骤	处　　理
1	输入 x 的值
2	如果 x<0,则 x 的绝对值为－x
3	输出 x 的绝对值

3.1.2 用 C++ 的 if 语句编程解决单路选择问题

对于需要进行单路选择处理的问题,C++ 提供了相应的 if 语句,用户可以使用该语句编写程序,让计算机完成问题的求解。

if 语句的语法格式如下:

```
if (<测试条件>)
    <分支语句>
```

if 语句的执行过程:首先计算<测试条件>的值,如果其值为 true(非 0),则表示满足某种条件,执行<分支语句>;否则,表示不满足某种条件,不执行<分支语句>,而直接执行分支语句后面的语句。图 3-1 是 if 语句流程图。

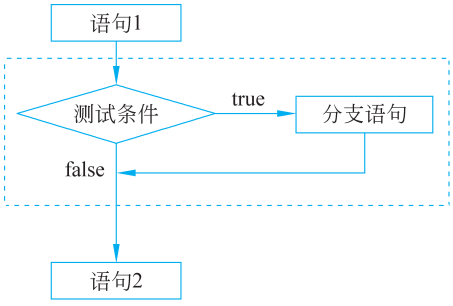

图 3-1 if 语句流程图

【例 3-1】 根据表 3-1 给出的算法编写程序,用 C++ 提供的 if 语句实现求一个实数的绝对值问题。

```
#include <iostream>
using namespace std;
int main()
{
    double x;
    cout<<"请输入 1 个实数:"<<endl;
    cin>>x;
    cout<<x<<"的绝对值是:";
    if (x<0)          // 当 x 是负数时,需要进行处理
        x=-x;
    cout<<x<<endl;
    return 0;
}
```

【提示】 if 语句中的(<测试条件>)不能缺少,<测试条件>应是逻辑型或计算结果可以自动转换为逻辑型的表达式。

3.2　双路选择算法及其 C++ 实现

3.2 节
讲解视频

双路选择问题是：如果某种情况发生，则进行一种处理；否则，进行另一种处理。例如，如果还有作业，则完成作业；否则，去电影院看电影。

3.2.1　双路选择问题

【问题 3-2】　判断某一年是否为闰年。

问题求解思路：能被 400 整除的年以及能被 4 整除但不能被 100 整除的年是闰年。假设 year 表示年，如果满足上述条件，则该年是闰年，否则该年不是闰年。

解决该问题的算法如表 3-2 所示。

表 3-2　求解问题 3-2 的算法

步骤	处　　理
1	输入 year 的值
2	如果 year 能被 400 整除或者能被 4 整除但不能被 100 整除，则输出“是闰年”； 否则，输出“不是闰年”

根据第 2 章学习的 C++ 知识，假设 year 表示年份，year 是闰年则下面的条件为真：

```
(year %4 == 0 && year %100 != 0) || (year %400 ==0)
```

【问题 3-3】　求解一元二次方程 $ax^2+bx+c=0$ 的两个不相等的实根。要求：在有两个不相等的实根的情况下，求解并输出两个实根；在没有两个实根的情况下，则输出“此方程无两个不相等的实根”的信息。

问题求解思路：这是一个典型的双路选择问题。需要判断该一元二次方程是否有两个实根，根据判断结果，做出相应的处理。

解决该问题的算法如表 3-3 所示。

表 3-3　求解问题 3-3 的算法

步骤	处　　理
1	输入一元二次方程，即输入系数 a、b、c
2	如果该方程有两个不相等的实根，则计算这两个实根并输出；否则，输出“此方程无两个不相等的实根”

根据第 2 章学习的 C++ 知识，如果一元二次方程 $ax^2+bx+c=0$ 有两个不相等的实根，则下列条件为真：

```
b*b-4*a*c>0&&a!=0
```

3.2.2　用 C++ 提供的 if…else 语句编程解决双路选择问题

对于需要进行双路选择处理的问题，C++ 提供了相应的 if…else 语句，用户可以使用该语句编写程序，让计算机完成问题的求解。

if…else 语句的语法格式如下：

```
if (<测试条件>)
    <分支语句 1>
else
    <分支语句 2>
```

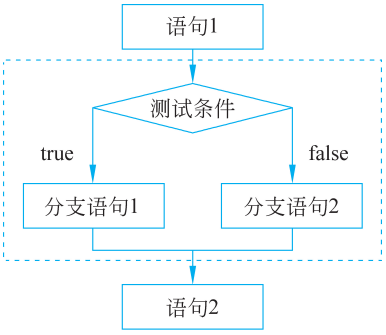

图 3-2 if…else 语句流程图

if…else 语句的执行过程：首先计算＜测试条件＞的值，如果其值为“真”(非 0)，则表示满足测试条件，执行＜分支语句 1＞；否则，执行＜分支语句 2＞。图 3-2 是 if…else 语句流程图。

同 if 语句一样，if…else 语句中的(＜测试条件＞)不能缺少，＜测试条件＞应是逻辑型或计算结果可以自动转换为逻辑型的表达式。

【例 3-2】 根据表 3-2 给出的算法编写程序，用 C++ 提供的 if…else 语句实现判断用户输入的年份是否为闰年的问题。

```
#include <iostream>
using namespace std;
int main()
{
    int year;
    bool isLeapYear;
    cout<<"请输入一个年数: "<<endl;
    cin>>year;
    isLeapYear=(year %4 == 0 && year %100 != 0) || (year %400 ==0);
    if(isLeapYear)
        cout<<year<<"年是闰年!"<<endl;
    else
        cout<<year<<"年不是闰年!"<<endl;
    return 0;
}
```

【例 3-3】 根据表 3-3 给出的算法编写程序，用 C++ 提供的 if…else 语句实现“求解一元二次方程 $ax^2+bx+c=0$ 的两个不相等的实根”问题。

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int a,b,c;
    double x1,x2;
    cout<<"请输入一元二次方程的系数: "<<endl;
    cin>>a>>b>>c;
    if (b*b-4*a*c>0&&a!=0)
    {
        x1=(-b+sqrt(b*b-4.0*a*c))/(2*a);
        x2=(-b-sqrt(b*b-4.0*a*c))/(2*a);
        cout<<"方程的两个实数根分别是: "<<x1<<"和"<<x2<<endl;
    }
```

```
    else
        cout<<"此方程无两个不相等的实根"<<endl;
    return 0;
}
```

【提示】 if 语句和 if…else 语句在逻辑上都是一条语句，它们的分支语句在逻辑上也都是一条语句，当一个分支功能需要多条语句才能完成时，如例 3-3，就需要使用复合语句。

3.3　嵌套选择算法及其 C++ 实现

3.3 节
讲解视频

在根据某种情况选择了一种处理后，在进行该处理时，还需要根据情况进行选择，这就构成了选择的嵌套，这样的问题就是嵌套选择问题。

【问题 3-4】 找出三个人中年龄最小的那个人。要求给出寻找过程。

问题求解思路：假设三个人的年龄分别是 age1、age2、age3，首先比较 age1 和 age2，找出第 1 和第 2 个人中年龄较小的那个人；然后再用这个人的年龄与第 3 个人的年龄 age3 进行比较，根据比较结果，找出年龄最小的那个人。由于第一次的比较结果决定第二次哪两个年龄进行比较，第二次的比较结果决定最后是谁最小，所以，这样的问题就是嵌套选择问题。解决该问题的算法如表 3-4 所示。

表 3-4　求解问题 3-4 的算法

步骤	处　　理
1	输入三个人的年龄，分别放到 age1、age2 和 age3 中
2	如果 age1＜age2 　输出“前两个人中第 1 个人较年轻，下面比较第 1 个人和第 3 个人的年龄”的信息 　如果 age1＜age3 　　则找到了年龄最小的人是第 1 个人，输出“第 1 个人年龄最小”的信息 　否则 　　找到了年龄最小的人是第 3 个人，输出“第 3 个人年龄最小”的信息 否则 　输出“前两个人中第 2 个人较年轻，下面比较第 2 个人和第 3 个人的年龄”的信息 　如果 age2＜age3 　　则找到了年龄最小的人是第 2 个人，输出“第 2 个人年龄最小”的信息 　否则 　　找到了年龄最小的人是第 3 个人，输出“第 3 个人年龄最小”的信息

【例 3-4】 根据表 3-4 的算法编写程序，找出用户输入的三个人中年龄最小的那个人。

用 C++ 提供的 if 语句或 if…else 语句，就能够解决嵌套的问题。根据表 3-4 对算法的描述可以发现，第一次（即外层）选择是一个双路选择，使用 if…else 语句即可解决。对于外层选择后的进一步处理也是一个双路选择，所以继续使用 if…else 语句来解决内层选择问题。

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

```
int main()
{
    int age1, age2, age3;
    cout<<"请分别输入三个人的年龄: "<<endl;
    cin>>age1>>age2>>age3;
    if(age1<age2)                                          // 外层 if
        {
            cout<<"前两个人中第 1 个人较年轻"<<endl
                <<"下面比较第 1 个人和第 3 个人的年龄"<<endl;
            if(age1<age3)                                  // 内层 if
                cout<<"第 1 个人年龄最小"<<endl;
            else                                           // 内层 else
                cout<<"第 3 个人年龄最小"<<endl;

        }
    else                                                   // 外层 else
        {
            cout<<"前两个人中第 2 个人较年轻: "<<endl
                <<"下面比较第 2 个人和第 3 个人的年龄"<<endl;
            if(age2<age3)                                  // 内层 if
                cout<<"第 2 个人年龄最小"<<endl;
            else                                           // 内层 else
                cout<<"第 3 个人年龄最小"<<endl;
        }
    return 0;
}
```

【提示】 C++ 编译器总是将 else 与其前面最近的那个尚未配对的 if 匹配成一个 if …else 结构。

3.4 节 讲解视频

3.4 多路选择算法及其 C++ 实现

多路选择问题也是根据不同的情况进行不同的处理。双路选择问题是多路选择问题中可能出现两种情况的一类问题。实际上,有的问题存在的情况会超过两种,这类问题就是多路选择问题。对于多路选择问题,可以直接使用 C++ 提供的 if 或 if…else 语句,通过多次判断来完成问题的求解。使用 if…else 语句处理多路选择问题,会因为嵌套的层次太多而导致程序的可读性下降,且容易出错。C++ 还提供了一个专门处理多路选择的 switch 语句。

3.4.1 多路选择问题

【问题 3-5】 用户输入一个数字字符 0～9,输出相应的 ASCII 码。

问题求解思路:如果用户输入的是字符 0,则输出 48;如果用户输入的是字符 1,则输出 49;依次类推;直到如果用户输入的是字符 9,则输出 57。这是一个多路选择问题。

解决该问题的算法如表 3-5 所示。

表 3-5 求解问题 3-5 的算法

步骤	处 理
1	用户输入数字字符 ch
2	如果用户输入的是字符 0,则输出 48; 如果用户输入的是字符 1,则输出 49; 如果用户输入的是字符 2,则输出 50; 如果用户输入的是字符 3,则输出 51; 如果用户输入的是字符 4,则输出 52; 如果用户输入的是字符 5,则输出 53; 如果用户输入的是字符 6,则输出 54; 如果用户输入的是字符 7,则输出 55; 如果用户输入的是字符 8,则输出 56; 如果用户输入的是字符 9,则输出 57

3.4.2　用 C++ 提供的 switch 语句编程解决多路选择问题

C++ 提供了适合处理多路选择情况的 switch 语句。

switch 语句的语法格式如下：

```
switch (<测试条件>)
{
    case <常量表达式 1>: [<分支语句序列 1>]
                         [break;]
    case <常量表达式 2>: [<分支语句序列 2>]
                         [break;]
         ⋮
    case <常量表达式 n>: [<分支语句序列 n>]
                         [break;]
    [default:           <分支语句序列 n+1>]
}
```

switch 语句的执行过程：首先计算＜测试条件＞的值，然后将该值逐个与各常量表达式的值进行比较。当＜测试条件＞的值与某个常量表达式的值相等时，就执行其后面的分支语句序列，如果遇到 break 语句则跳出 switch 语句，否则继续执行其后的分支语句序列。如果＜测试条件＞的值与所有常量表达式的值都不相等，则执行 default 后面的分支语句序列，如果缺省 default，则跳出 switch 语句。

switch 语句中的＜测试条件＞表达式只能是整型、字符型或枚举型的表达式。各＜常量表达式＞的值要互不相同，与条件表达式应为同一数据类型。各＜常量表达式＞后面的语句序列不需要用一对花括号“{}”括起来。执行某个＜常量表达式＞后面的分支语句序列时，只有遇到 break 语句时才跳出 switch 语句，否则将顺序执行后面的分支语句序列。图 3-3 是每一个分支语句序列都没有 break 语句的 switch 语句流程图。图 3-4 是每一个分支语句序列都有 break 语句的 switch 语句流程图。

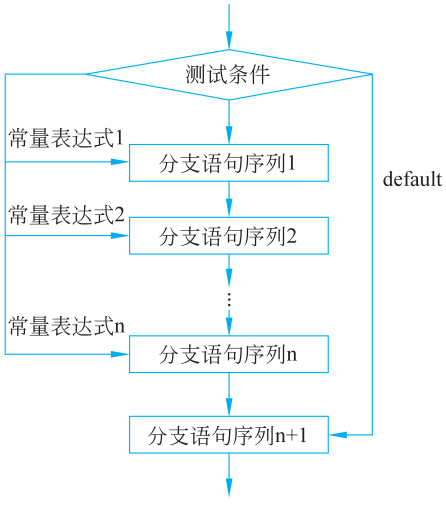

图 3-3　没有 break 语句的 switch 语句流程图

default 分支可以缺省。如果＜测试条件＞的值与所有常量表达式的值都不相等，则直接跳出 switch 语句。

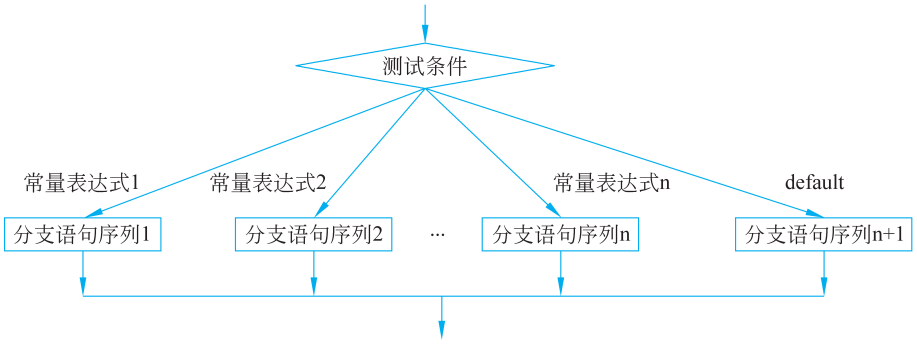

图 3-4 有 break 语句的 switch 语句流程图

【例 3-5】 根据表 3-5 给出的算法编写程序,根据用户输入的一个数字字符 0～9,输出相应的 ASCII 码。

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cout<<"请输入 1 个数字字符(0～9): "<<endl;
    cin>>ch;
    switch(ch)
    {
        case '0': cout<<"该数字字符的 ASCII 码是: "<<48<<endl; break;
        case '1': cout<<"该数字字符的 ASCII 码是: "<<49<<endl; break;
        case '2': cout<<"该数字字符的 ASCII 码是: "<<50<<endl; break;
        case '3': cout<<"该数字字符的 ASCII 码是: "<<51<<endl; break;
        case '4': cout<<"该数字字符的 ASCII 码是: "<<52<<endl; break;
        case '5': cout<<"该数字字符的 ASCII 码是: "<<53<<endl; break;
        case '6': cout<<"该数字字符的 ASCII 码是: "<<54<<endl; break;
        case '7': cout<<"该数字字符的 ASCII 码是: "<<55<<endl; break;
        case '8': cout<<"该数字字符的 ASCII 码是: "<<56<<endl; break;
        case '9': cout<<"该数字字符的 ASCII 码是: "<<57<<endl; break;
        default: cout<<"输入的不是数字字符"<<endl;
    }
    return 0;
}
```

3.5 节 讲解视频

3.5 迭代算法及其 for 语句的实现

很多计算问题需要重复一组特殊的计算步骤,直到达到某一状态才停止。例如要求“计算 1～5 的累加和”的问题,可如下设计一个简单的算法。

第 1 步:将总和 sum 清 0。

第 2 步:将 1 累加到 sum 中。

第 3 步:将 2 累加到 sum 中。

第 4 步:将 3 累加到 sum 中。

第 5 步:将 4 累加到 sum 中。

第 6 步:将 5 累加到 sum 中。

第 7 步：输出 1～5 的累加和 sum。

这个算法很简单。如果问题变成设计“计算 1～100 的累加和”的算法，则需要 100 行类似“将 1 累加到 sum 中”的操作；如果问题变成设计“计算 1～100 000 的累加和”的算法，则需要 100 000 行类似“将 1 累加到 sum 中”的操作。显然，这种算法在处理大规模问题时是不可行的。

3.5.1　迭代算法

使用计算思维中经常使用的迭代思想，设计出迭代算法，可以将算法中重复步骤简要、清楚地描述出来，使算法变短从而增加其可读性。

设计迭代算法时，首先要确定需要重复的操作或操作集合，然后确定需要进行多少次这样的操作。对于上面的问题，可设计如下一个非常短的迭代算法。

第 1 步：将总和 sum 清 0。

第 2 步：i 的取值范围是 1～5，进行如下操作：

将 i 累加到 sum 中。

第 3 步：输出 1～5 的累加和 sum。

如果问题变为“设计计算 1～n 的累加和的算法”时，仅需将上面迭代算法的第 2 步“i 的取值范围是 1～5”修改为“i 的取值范围是 1～n”即可。

【问题 3-6】 设计“计算 1＋2＋3＋…＋n”的迭代算法。

问题求解思路： 可以发现该问题需要重复的操作是将一个整数累加，这个操作需要重复 n 次。

解决该问题的迭代算法如表 3-6 所示。

表 3-6　求解问题 3-6 的迭代算法

步骤	处　　理
1	输入用户要累加的最大整数 n
2	将存放累加和的 sum 清 0
3	i 的取值范围是 1～n，进行如下操作： 将 i 累加到 sum 中
4	输出结果 sum

3.5.2　用 C++ 提供的 for 语句实现迭代算法

所有的高级程序设计语言都有实现迭代算法的循环语句。所以，可以使用这些循环语句来实现一个语句（或一组语句）的重复处理，直到某个条件满足才停止。

C++ 提供了专门用于循环的 for 语句。for 语句的语法格式如下：

```
for ([<表达式 1>]; [<表达式 2>]; [<表达式 3>])
    <循环体>
```

for 语句的执行过程：首先执行＜表达式 1＞，然后执行＜表达式 2＞。当＜表达式 2＞的值为“真”（非 0），执行＜循环体＞。每执行完＜循环体＞后，执行＜表达式 3＞，再执行＜表达式 2＞，当＜表达式 2＞的值为“真”（非 0）则重复执行循环体，再执行＜表达式 3＞……直到＜表达式 2＞等于“假”（0）为止。图 3-5 是 for 语句流程图。

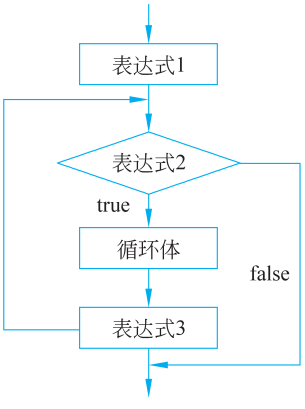

图 3-5　for 语句流程图

for 语句中的＜表达式 1＞的主要作用是初始化循环变量；＜表达式 2＞的主要作用是控制循环；＜表达式 3＞的主

要作用是修改循环变量。

【例 3-6】 根据表 3-6 给出的迭代算法编写程序,用 C++ 提供的 for 语句实现“计算 1+2+3+…+n”的问题。

```
#include <iostream>
using namespace std;
int main()
{
    int n,sum;
    cout<<"请输入要累加的最大整数: "<<endl;
    cin>>n;
    sum=0;
    for(int i=1; i<=n; i++)
        sum=sum+i;
    cout<<"1+2+3+…+n 的结果为: "<<sum<<endl;
    return 0;
}
```

3.6 节 讲解视频

3.6 迭代算法及其 while 语句的实现

3.6.1 用 C++ 提供的 while 语句实现迭代算法

【问题 3-7】 设计“计算 2^n”的迭代算法。

问题求解思路:可以发现该问题需要重复的操作是乘 2,这个操作需要重复 n 次。

解决该问题的迭代算法如表 3-7 所示。

表 3-7 求解问题 3-7 的迭代算法

步骤	处　　理
1	输入 2 的幂次 n
2	将存放结果的 power 置为 1
3	重复 n 次下面的操作:power=power×2
4	输出结果

C++ 提供了专门用于循环的 while 语句。while 语句的语法格式如下:

```
while (<测试条件>)
    <循环体>
```

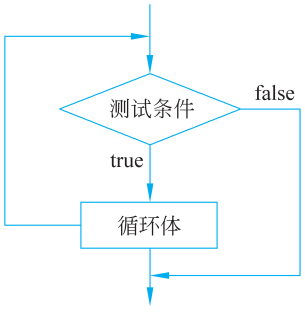

图 3-6 while 语句流程图

while 语句的执行过程:首先计算<测试条件>的值,如果其值为“真”(非 0),表示满足循环条件,则执行<循环体>;如果其值为“假”(0),则结束循环。每执行完一次循环体后再次计算<测试条件>的值,如果其值为“真”(非 0),则继续执行<循环体>;如果其值为“假”(0),则结束循环。图 3-6 是 while 语句流程图。

【例 3-7】 根据表 3-7 给出的迭代算法编写程序,用 C++ 提供的 while 语句实现“计算 2^n”的问题。

```
#include <iostream>
using namespace std;
int main()
{
    int i,n,power;
```

```
    cout<<"请输入幂次: "<<endl;
    cin>>n;
    power=1;
    i=1;
    while (i<=n)
    {
        power=power * 2;
        i++;
    }
    cout<<"2 的 n 次幂为: "<<power<<endl;
    return 0;
}
```

【提示】

- while 语句中的(＜测试条件＞)不能缺省,应该是逻辑型或计算结果可以自动转换为逻辑型的表达式。
- while 语句中,＜循环体＞在语法逻辑上是一条语句,如果多条语句的情况下则要使用复合语句。
- 在循环语句的＜循环体＞或＜测试条件＞部分必须有改变循环条件,使＜测试条件＞表达式最终成为假的语句,如例 3-12 和例 3-13 中的“i＋＋;”,否则＜测试条件＞永远为“真”,造成无法退出循环,即出现所谓的“死循环”。

3.6.2　用 C++ 提供的 do…while 语句实现迭代算法

C++ 提供了专门用于循环的 do…while 语句。do…while 语句的语法格式如下:

```
do
    <循环体>
while (<测试条件>);
```

do…while 语句的执行过程:首先执行＜循环体＞,然后计算＜测试条件＞的值,如果其值为“真”(非 0),表示满足循环条件,则重复上述过程;如果其值为“假”(0),则结束循环。图 3-7 是 do…while 语句流程图。

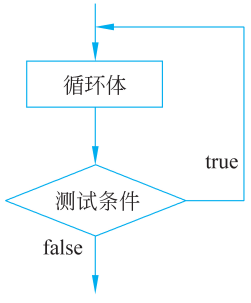

图 3-7　do…while 语句流程图

【例 3-8】　根据表 3-6 给出的迭代算法,编写程序,用 C++ 提供的 do…while 语句实现“计算 1＋2＋3＋…＋n”的问题。

```
#include <iostream>
using namespace std;
int main()
{
    int i=1,sum=0;
    do
    {
        sum=sum+i;
        i++;
    }
    while (i<=100);          // 分号不能缺省
    cout<<"1+2+3+…+100 的结果为: "<<sum<<endl;
    return 0;
}
```

3.7节
讲解视频

3.7 迭代嵌套及其C++实现

如果在一个迭代过程中又包含了迭代过程,就构成了迭代嵌套。实际的应用程序常常采用循环的嵌套结构,即C++程序的循环体中还可以包含各种循环语句,这就构成了循环的嵌套,通常称为多重循环。

【问题3-8】 设计"计算 $2^1+2^2+2^3+\cdots+2^n$"的算法,并用C++程序实现该算法。

问题求解思路:该问题是一个n项累加的问题,需要重复n次完成。其中,要累加的每一项也需要进行重复处理,即计算第i项的值时,也需要重复i次乘以2的操作。因此,是一个迭代嵌套的算法。

解决该问题的嵌套迭代算法如表3-8所示。

表3-8 求解问题3-8的算法

步骤	处　　理
1	输入2的幂次n
2	将各项的累加和sum清0
3	将i的范围设置为1～n,进行如下操作: 　将存放第i项结果的power置为1 　重复i次下面的操作: 　　power=power×2 　将power累加到sum中
4	输出结果sum

【例3-9】 根据表3-8给出的迭代嵌套算法编写程序,用C++提供的循环语句实现"计算 $2^1+2^2+2^3+\cdots+2^n$"的算法。

```
#include <iostream>
using namespace std;
int main()
{
    int n, sum, power, i, j;
    cout<<"请输入最高幂次 n: ";
    cin>>n;
    sum=0;
    for(i=1; i<=n; i++)
    {
        power=1;
        j=1;
        while (j<=i)
        {
            power=power * 2;
            j++;
        }
        sum=sum+power;
    }
    cout<<"2 的 1~"<< n <<"次幂的和为: "<<sum<<endl;
    return 0;
}
```

【提示】

- 不同结构的循环语句也可以构成循环嵌套,例如for循环里面可以嵌套while循环。
- 循环嵌套的执行顺序是外层循环执行时,若遇到内层循环,则先完成所有内层循环,然后再开始外层的下一次循环。
- 循环体里可以嵌套多个内层循环。

【例3-10】 "水仙花数"是指一个三位整数,其各位数字立方的和等于该数本身。例如,$153=1^3+5^3+3^3$,所以153是水仙花数。编写程序,求所有的"水仙花数"。

问题求解思路：因为“水仙花数”是三位整数，所以一定都在 100～999 内。利用三重循环，外层循环变量 i 控制百位数字从 1 变化到 9，中层循环变量 j 控制十位数字从 0 变化到 9，内层循环变量 k 控制个位数字从 0 变化到 9，穷举判断所有的三位整数是否是水仙花数。

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

int main()
{
    int i, j, k, m, n;
    for(i=1; i<=9; i++)                          // 外层循环，控制百位
        for(j=0; j<=9; j++)                      // 中层循环，控制十位
            for(k=0; k<=9; k++)                  // 内层循环，控制个位
            {
                n=100*i+10*j+k;                  // 求三位数
                m=i*i*i+j*j*j+k*k*k;             // 求其各位数字立方的和
                if(m==n)   cout<<m<<endl;
            }
    return 0;
}
```

3.8　迭代与选择嵌套及其 C++ 实现

3.8 节
讲解视频

3.8.1　迭代与选择嵌套及其 C++ 实现

在一个迭代算法中，如果包括一个选择处理，则构成了迭代和选择的嵌套。使用 C++ 的循环语句，在循环体采用选择语句来实现嵌套和选择的嵌套算法。

【例 3-11】　可以根据用户输入的年份判断该年是否为闰年，直到用户输入 0 年停止处理。

问题求解思路：在学习双路选择问题时已经实现了判断某某年是否是闰年的问题。现在的问题是需要重复若干次这样的判断，直到用户输入的年份为 0，这又是一个迭代算法。

解决该问题的迭代与选择嵌套算法如表 3-9 所示。

表 3-9　求解例 3-11 的算法

步骤	处　　理
1	输入一个年份 year
2	当 year 不等于 0 时重复下面的操作： 　如果 year 满足闰年条件 　　输出“是闰年” 　否则 　　输出“不是闰年” 　输入一个新的年份 year

```
#include <iostream>
using namespace std;
int main()
{
    int year;
    bool isLeapYear;
    cout<<"请输入一个年份(0 退出程序)："<<endl;
    cin>>year;
    while (year!=0)
    {
        isLeapYear=(year %4 == 0 && year%100 != 0)||(year%400 ==0);
        if(isLeapYear)
            cout<<year<<"年是闰年!"<<endl;
```

```
        else
            cout<<year<<"年不是闰年!"<<endl;
        cout<<"请输入一个年份(0退出程序):"<<endl;
        cin>>year;
    }
    return 0;
}
```

3.8.2 选择与迭代嵌套及其C++实现

在一个选择算法中,如果某个分支又包含了迭代处理,则就构成了选择和迭代的嵌套。使用C++的选择语句,在选择语句中采用循环语句来实现选择和迭代的嵌套算法。

【例3-12】 编程实现可以根据用户的选择进行求"1～10的和"(用户输入1)或"求10!"操作(用户输入2)。

问题求解思路: 当用户输入1时,进入求解"1～10的和"的分支,当用户输入2时,进入求解"求10!"的分支。其中,每一个分支又是一个迭代算法。

解决该问题的选择与迭代嵌套算法如表3-10所示。

表3-10 求解例3-12的算法

步骤	处　　理
1	输入一个选择m
2	如果m等于1 　　则迭代求解"1～10的和",并输出结果 如果m等于2 　　则迭代求解"10!",并输出结果

```
#include <iostream>
using namespace std;
int main()
{
    int m, sum=0, fac=1;
    cout <<"请输入你的选择(1: 1~10的和,2: 求10!): ";
    cin >> m;
    if(m == 1)
    {
        for(int i=1; i<=10; i++)
            sum = sum + i;
        cout <<"1~10的和: "<< sum << endl;
    }
    else
    {
        for(int i=2; i<=10; i++)
            fac=fac * i;
        cout <<"10!的结果为: "<< fac << endl;
    }
    return 0;
}
```

3.9节 讲解视频

3.9 C++中的转向语句

C++提供的转向语句包括break语句、continue语句、goto语句和return语句。使用这些语句可以使程序简练,或者减少循环次数,或者跳过那些没有必要再去执行的语句,以提高程序执行效率。但对转向语句使用不当也容易造成程序的混乱,甚至出现错误。所以,要理解各转向语句的功能,恰当地使用它们。

3.9.1　break 语句

C++ 提供的 break 语句又称跳出语句。它的语法格式是关键字 break 加分号，即

```
break;
```

break 语句可用在 switch 语句或三种循环语句中。break 语句的功能是：结束当前正在执行的循环(for、while、do…while)或多路分支(switch)程序结构，转而执行这些结构后面的语句。在 switch 语句中，break 语句用来使流程跳出 switch 语句，继续执行 switch 后的语句。在循环语句中，如果 break 语句位于多重循环的内层循环中，则只能跳出它所在的内层循环。

【例 3-13】　编写程序，找出 5～100 内能同时被 3 和 5 整除的最小的整数。

问题求解思路：由于需要搜索 5～100 的整数，所以可以采用 for 循环语句，依次判断 5、6、7……100，是否能同时被 3 和 5 整除。由于问题只需找到能同时被 3 和 5 整除的最小的那个整数，所以在循环中，找到第一个满足条件的数即退出循环，退出循环适合使用 break 语句。整型变量 n 能同时被 3 和 5 整除的逻辑表达式为“n%5==0&&n%3==0”。

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    for(n=5; n<=100; n++ )
    {
        if(n%5==0&&n%3==0)   // 如果 n 能被 3 和 5 同时整除,退出循环
            break;
    }
    cout <<"能同时被 3 和 5 整除的最小整数是: "<< n << endl;
    return 0;
}
```

3.9.2　continue 语句

C++ 提供的 continue 语句的语法格式为关键字 continue 加上分号，即

```
continue;
```

continue 语句的功能是：根据某个判断条件结束本次循环，即跳过循环体中尚未执行的语句，接着进行下一次是否执行循环的判定。

【例 3-14】　编写程序，找出 5～100 内能同时被 3 和 5 整除的所有整数。

问题求解思路：由于需要搜索 5～100 的整数，所以可以采用 for 循环语句，依次判断 5、6、7……100，是否能同时被 3 和 5 整除。由于问题需要找到能同时被 3 和 5 整除的所有整数，所以在循环中，如果当前的这个整数不满足条件，则开始下一次查找；否则，输出此数。结束本次循环的语句适合使用 continue 语句。

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
```

```
{
    int n;
    for(n=5; n<=100; n++ )
    {
        if(n%5!=0 || n%3!=0)        // 如果 n 不能被 3 和 5 整除,则开始下一次循环
            continue;
        cout<<' '<<n;               // 如果 n 能同时被 3 和 5 整除,则输出
    }
    cout<<endl;
    return 0;
}
```

3.9.3 return 语句

C++ 中的 return 语句也称函数返回语句。它的语法格式如下：

```
return [<表达式>];
```

return 语句的功能是：停止当前函数，程序转去执行当前函数调用后面的语句。

对于非 void 类型的函数，必须有 return 语句，其中＜表达式＞的类型要与函数返回值的类型一致；当函数的返回类型为 void 型时，不需要写＜表达式＞，也可以不写 return 语句。

3.9.4 goto 语句

goto 语句的也称无条件转向语句，它的语法格式如下：

```
goto <标号>;
```

goto 语句的功能是：将程序无条件跳转到＜标号＞指定的语句处继续执行。其中，＜标号＞是一个 C++ 的标识符，放在要跳转到的语句前面，标号说明格式如下：

```
<标号>:<语句>
```

【例 3-15】 编写程序，使用 goto 语句，实现找出从 5 到 100 的能同时被 3 和 5 整除的最小的整数。

问题求解思路： 与例 3-14 思考方法相同。在本例中使用 goto 语句代替 break 语句。

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    for(n=5; n<=100; n++ )
    {
        if(n%5==0&&n%3==0)   // 如果 n 能被 3 和 5 同时整除,则退出循环
            goto tt;
    }
    tt : cout <<"能同时被 3 和 5 整除的最小整数是: "<< n << endl;
    return 0;
}
```

【提示】

- continue 语句和 break 语句的区别是：continue 语句只结束本次循环，而不是终止整

个循环的执行；break 语句则是结束本次循环，不再进行条件判断。

- 在多重循环语句中，break 语句只能跳转到循环的上一层，即结束它所在的那一层循环。goto 语句可以直接从内层循环跳转到多重循环外，即结束所有多重循环。
- 如果不加限制地使用 goto 语句，则会导致程序流程的混乱，降低程序的可读性。一般情况下，应尽量减少或不使用 goto 语句。

3.10　应用案例——智能五子棋人机对战程序

在第 2 章实现静态棋盘的基础上，下面学习如何让程序具备“判断”与“重复”的能力，实现棋盘交互与绘制优化，为后续实现完整对弈逻辑打下基础。

3.10.1　Excitation——提出感兴趣的话题

第 2 章我们系统地解决了“棋盘如何被看见”的问题，成功地将五子棋棋盘从抽象数据转化为屏幕上可视的图形界面，并定义了管理游戏状态的基础变量。至此，我们已经完成了智能五子棋人机对战程序“静”的部分——一个绘制完整的、等待对弈开始的棋盘。

然而，一个真正的游戏程序必须是“动”的，它需要能够响应用户的操作，并按照规则更新状态。目前我们的程序尚不具备这个能力：当我们单击棋盘上期望落子的位置时，程序不会有任何反应；棋盘上预设的棋子是固定的，我们无法添加新的棋子，更无法实现双方轮流落子的基本流程。

仔细思考实现轮流落子这一过程，其核心涉及程序两种最基本的“行为”能力：一是**判断**，即程序需要能够判断当前该哪一方落子、玩家的单击位置是否有效等条件，并据此决定执行何种操作；二是**重复**，例如绘制棋盘上 15 条横线和 15 条竖线，本质上是同一绘图动作的重复执行，而双方轮流落子本身也是一个循环交替的过程。

因此，在拥有了静态棋盘的基础上，本章我们要面对的新问题是：如何让程序“学会”判断与重复，从而实现对鼠标单击的响应，并优化程序的构建方式？

换言之，**如何利用程序的基本逻辑结构，让五子棋棋盘“活”起来，实现最基本的交互？**

3.10.2　Exploration——探索发现问题本质

明确了“让棋盘响应用户单击，实现基本交互与绘制优化”这一目标后，我们需要深入剖析这一任务所包含的核心挑战。这一过程要求我们系统思考程序如何从接收用户输入到完成反馈的完整流程，并重新审视程序构建方式中存在的优化空间。

1. 分解“单击-落子”交互的流程

从用户单击鼠标到棋盘相应位置显示棋子，程序需要依次完成一系列逻辑严密的处理步骤。每个步骤都对应着程序需要具备的一种特定能力。

(1) 如何感知用户的单击操作

在图形界面程序中，用户的鼠标单击、移动等操作被系统捕获并转化为事件——一种通知程序“有用户操作发生”的信号。程序需要建立一种机制来持续检查这类事件的发生。具体到五子棋程序，我们需要让程序能够检测到“鼠标左键被按下”这个事件，并获取按下时的位置坐标。这引出了一个具体的技术问题：在使用 EasyX 图形库时，如何编写代码来实现

这种事件的检测与获取?

(2) 如何将屏幕坐标转换为棋盘逻辑坐标

程序获取到的鼠标位置是基于整个窗口的像素坐标(如(320,180))。但程序内部管理棋盘状态使用的是行列索引(如第7行、第7列)。因此,必须设计一个转换算法:根据棋盘绘制的起始位置(margin)和网格间距(chessSize),将像素坐标(x,y)准确映射为棋盘上的行列索引(row,col)。这一步是连接用户交互与程序内部数据的关键桥梁。

(3) 如何判断落子位置的有效性

获得行列索引后,程序并不能直接落子,而需要进行两步关键的条件判断,这正是选择结构的典型应用场景。

① 范围有效性判断。

转换得到的row和col是否在棋盘的有效索引范围内(如对于15×15棋盘,是否均在0~14内)?只有在有效索引内才有效,这确保了单击位置在棋盘网格内。

② 状态合法性判断。

该位置当前的棋子状态是什么?是"空位",还是已被黑子或白子占据?这需要程序查询其内部记录该位置状态的数据。只有当状态为"空位"时,此处才允许落子。

(4) 如何管理回合并实现自动切换

五子棋对弈要求双方轮流落子。程序内部必须维护一个状态变量(如currentPlayer)来记录"当前轮到哪一方"。每次成功落子后,程序必须能自动更新该状态,实现黑白双方的回合切换。这通常通过简单的状态翻转逻辑实现,是程序维持对弈流程的基础。

(5) 如何更新状态并完成视觉呈现

所有条件满足后,程序需执行两项收尾工作。

① 更新内部棋盘状态数据,将对应位置标记为当前玩家的棋子。

② 调用图形绘制函数,在对应的屏幕坐标上绘制出相应颜色(黑或白)的棋子图形,从而完成从内部状态变更到外部视觉反馈的完整闭环。

2. 优化重复性任务的处理机制

在分析交互逻辑的同时,我们也需要审视现有代码的构建方式。在第2章绘制棋盘网格时,我们写了15条几乎相同的line()语句来画横线,又写了15条几乎相同的语句来画竖线。这种编写方式带来了明显的弊端。

- 代码冗长:大量相似的语句使得程序文件变得冗长。
- 难以维护:如果需要调整棋盘大小(如改为19×19),或者调整线条样式,我们必须手动修改数十行代码,极易出错或遗漏。
- 不符合计算机思维:计算机最擅长执行精确的、重复性的任务。我们目前的写法并未充分利用这一优势,反而像是在用手工的方式指挥机器。

这暴露出我们当前工具集的不足:我们急需一种语法结构,能够让我们只描述一次"画一条线"这个动作,然后指挥计算机:"将这个动作重复执行N次,每次参数有一点规律变化"。这种结构就是循环(迭代)。通过循环,我们可以将数十行重复代码压缩成寥寥数行,并且将棋盘大小等参数定义为变量,使得未来修改变得轻而易举,这正体现了程序设计的核心优势之一——通过抽象与自动化提升效率与灵活性。

通过以上探索,我们将"棋盘交互与绘制优化"这一目标,具体转化为对事件处理、坐标

转换、条件判断、状态管理、循环控制等一系列基础编程概念与技能的掌握需求。这为本章后续的系统学习与实践提供了明确的技术路线图。

3.10.3　Enhancement——拓展学习求解问题必备的知识和能力

在明确了“让棋盘响应用户单击，实现基本交互与绘制优化”这一目标后，我们需要系统地学习支撑这一目标所必需的编程知识与技能。本节将通过一系列贴近实际应用的示例，详细讲解实现棋盘交互功能所需的核心概念与编程方法，包括事件处理机制、坐标转换原理、状态管理策略以及循环结构的应用等。这些知识不仅对实现五子棋项目至关重要，更是解决各类图形交互问题的通用基础。

1. 事件处理机制——程序如何“感知”用户操作

在图形界面程序中，用户通过鼠标、键盘等设备进行的任何操作都被称为事件。事件是程序外部发生的事情给程序内部发送的一种“通知”，例如鼠标单击、键盘按键等。要让程序能够响应这些用户操作，我们需要学习如何让程序“感知”并处理这些事件。

(1) 事件的基本概念

在 EasyX 图形库中，用户的鼠标操作(单击、移动、释放)会被封装成特定格式的消息。程序通过检测和处理这些消息来实现与用户的交互。为了完整描述一个鼠标事件，需要同时记录事件的类型(如左键按下)和发生位置(坐标)，这就需要一种能够将多个相关数据组合在一起的数据组织形式。

(2) 结构体的初步认识

在 C++ 中，结构体(struct)是一种用户自定义的数据类型，它允许将多个不同类型的数据项组合成一个整体。第 4 章将系统学习结构体的定义和使用，这里可先了解其基本概念。结构体就像一个“数据容器”，可以将相关的数据打包在一起。

在 EasyX 中，鼠标事件的相关信息被打包在一个名为 MOUSEMSG 的预定义结构体中。该结构体包含了描述鼠标事件所需的关键信息，主要包括：

- uMsg，事件类型标识，表示发生了何种鼠标事件。
- x，事件发生时鼠标在窗口中的横坐标(像素)。
- y，事件发生时鼠标在窗口中的纵坐标(像素)。

(3) 事件监测的基本方法

在 EasyX 图形库中，有多种方法可以处理鼠标事件。对于初学者来说，最直观的方式是使用 MouseHit()函数和 GetMouseMsg()函数的组合。

① 检测鼠标事件。MouseHit()函数用于检查当前是否有鼠标事件发生。如果有，则该函数返回“真”(true)；否则，该函数返回“假”(false)。通常将这个检查放在程序的主循环中，让程序持续关注是否有鼠标操作。

② 获取鼠标消息。当检测到有鼠标事件时，我们需要进一步获取这个事件的详细信息。GetMouseMsg()函数就是用来完成这个任务的。该函数返回一个 MOUSEMSG 类型的结构体，其中包含了鼠标事件的完整信息。调用该函数后，我们可以通过访问返回结构体的成员来获取具体的事件详情。关于结构体成员的访问方法第 4 章还会详细介绍，此处先做了解即可。

例如，执行语句“MOUSEMSG msg＝GetMouseMsg()；”后：

- msg.uMsg 存储了事件类型,通过判断其值可以确定是哪种鼠标事件。
- msg.x 和 msg.y 存储了事件发生时鼠标在窗口中的坐标位置。

(4) 常用鼠标事件类型

为了便于识别不同类型的鼠标事件,EasyX 预定义了一系列事件类型常量。这些常量实际上是 Windows 操作系统定义的消息标识符,EasyX 直接引用了这些标准定义。最常用的鼠标事件类型常量包括:

- WM_LBUTTONDOWN,表示鼠标左键被按下。
- WM_LBUTTONUP,表示鼠标左键被释放(弹起)。
- WM_RBUTTONDOWN,表示鼠标右键被按下。
- WM_MOUSEMOVE,表示鼠标移动。

这些常量的命名具有明确的含义:WM 代表 Windows Message(Windows 消息),LBUTTON 代表"左键",DOWN 代表"按下"。因此,WM_LBUTTONDOWN 整体表示"鼠标左键按下消息"。

在程序中,通常使用 if 语句(选择结构)来判断具体的事件类型,例如:

```
if (msg.uMsg==WM_LBUTTONDOWN) {
…    // 处理鼠标左键按下事件
}
```

(5) 事件处理的基本框架

一个典型的图形界面程序需要持续运行以响应用户操作,这通常通过一个循环结构(如 while 循环)来实现。在循环内部,程序不断检查是否有新的事件发生,并根据事件类型进行相应处理。这种"循环检查-处理事件"的模式是事件驱动型程序的基本框架。

【例 3-16】 编写程序,实现简单的鼠标事件检测。

问题求解思路: 本示例旨在演示如何在图形程序中检测并响应鼠标单击事件。程序将创建一个 400×300 像素的窗口,当用户在窗口内按下鼠标左键时,程序会在控制台输出单击位置的坐标信息。

```
#include <graphics.h>
#include <conio.h>
#include <iostream>
using namespace std;
int main() {
    // 初始化图形窗口
    initgraph(400, 300);
    // 设置背景色
    setbkcolor(WHITE);
    cleardevice();
    // 显示操作提示
    settextcolor(BLUE);
    settextstyle(16, 0, _T("宋体"));
    outtextxy(50, 120, _T("单击窗口任意位置观察坐标输出"));
    outtextxy(50, 150, _T("按 ESC 键退出程序"));
    // 事件处理循环
    while (true) {
        // 检测鼠标事件
        if (MouseHit()) {
            // 获取鼠标消息
```