

第 2 章 循环和计数

刘 卉

huiliu@fudan.edu.cn

前言

□ 改写第1章的程序

- 更灵活: 不需要重写程序, 就可以改变框架的大小.
- C++中的算术问题
- C++中的循环和条件语句
- 循环不变式(loop invariant)

2.1 问题

- “为名字装框输出”程序的主要缺点
 - 输出的每一行都需要与一条语句和一个变量相对应.
 - 输出格式发生微小变化, 都需要重写程序.
- 解决方法
 - 除greeting外, 单独输出每个字符(' '和'*').
 - 输出的字符不再保存到变量, 因为输出后就不需要了.

2.2 程序的总体结构

```
#include <iostream>
#include <string>
int main()
{
    std::cout << "Please enter your first name: ";
    std::string name;
    std::cin >> name;
    // build the message that we intend to write
    const std::string greeting = "Hello, " + name + "!";
    // we have to rewrite this part...
    return 0;
}
```

2.3 输出任意多行

- 输出: 长方形的字符矩阵, 一次输出一行.
- 计算输出的行数

```
const int pad = 1; // the number of blanks surrounding the greeting
const int rows = pad * 2 + 3; // the number of rows to write
// 变量pad控制上下左右的空白数,即框架的大小
std::cout << std::endl; // separate the output from the input
int r = 0; // invariant: we have written r rows so far
while (r != rows) {
    // write a row of output (as we will describe in § 2.4)
    std::cout << std::endl;
    ++r;
}
```

while语句

□ 循环不变式——设计循环的有效工具

- 每轮循环,
 1. 测试循环条件时, 不变式为真.
 2. 结束之前, 不变式也为真.
- 寻找不变式的策略: 能说明循环语句所包含变量的某个性质.
- 循环语句体的作用: 保持不变式为真的情况下, 操作相关变量, 最终使循环条件为假.

```
// invariant: we have written  $r$  rows so far
int r = 0; // setting  $r$  to 0 makes the invariant true
while (r != rows) {
    // we can assume that the invariant is true here
    // writing a row of output makes the invariant false
    std::cout << std::endl;
    // incrementing  $r$  makes the invariant true again
    ++r;
}
// we can conclude that the invariant is true here
```


2.4 输出一行

□ 计算输出的列数

```
const std::string::size_type cols = greeting.size() + pad * 2 + 2;
```

- 与名字空间和复合语句相同, 类也定义了自己的作用域.
- `std::string::size_type`: 类型名, 无符号整型.
- 局部变量`cols`: 表示一个string对象的长度.

`cols`能包含`greeting`的字符数, 无论这个数有多大.

 **良好的习惯:** 定义变量用来保存标准库中某个特定数据结构的大小时, 应使用标准库定义的相应类型.

□ 输出每行字符

```
std::string::size_type c = 0;  
// invariant: we have written c characters so far in the current row  
while (c != cols) {  
    // write one or more characters  
    // adjust the value of c to maintain the invariant  
}
```

□ C++的内置类型bool

- bool类型的值只有两种: 真(true)或假(false).

2.4.1 输出边界字符

□ 根据循环不变式决定何时输出'*'

```
// invariant: we have written c characters so far in the current row
while (c != cols) {
    if (r == 0 || r == rows - 1 || c == 0 || c == cols - 1) {
        std::cout << "*";
        ++c;
    } else {
        // write one or more non-border characters
        // adjust the value of c to maintain the invariant
    }
}
```

2.4.2 输出非边界字符

□ 空白符或问候语

```
if (r == pad + 1 && c == pad + 1) {  
    cout << greeting;  
    c += greeting.size();  
} else {  
    cout << " ";  
    ++c;  
}
```

2.5 完整的程序框架

□ 三种方式简化程序

1) using声明

e.g. `using std::cout;`

□ 一次声明, 此后可用`cout`表示`std::cout`.

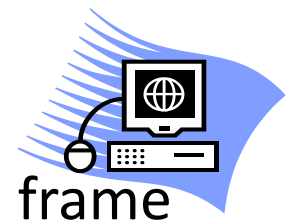
2) 用for语句取代while语句.

```
for (int r = 0; r != rows; ++r) {  
    // stuff that doesn't change the value of r  
}
```

□ 半开区间[开始值, 越界值)

3) 合并相同操作的条件语句

```
if (we are about to write the greeting) {  
    cout << greeting;  
    c += greeting.size();  
}  
else {  
    if (we are in the border)  
        cout << "*";  
    else  
        cout << " ";  
    ++c;  
}
```



```
#include <iostream>
#include <string>
// say what standard-library names we use
using std::cin;          using std::endl;
using std::cout;         using std::string;
int main()
{
    cout << "Please enter your first name: ";
    string name;
    cin >> name;
    const string greeting = "Hello, " + name + "!";
    const int pad = 3; // the number of blanks surrounding the greeting
    const int rows = pad * 2 + 3; // the number of rows and columns to write
    const string::size_type cols = greeting.size() + pad * 2 + 2;
    cout << endl; // write a blank line to separate the output from the input
    // write rows rows of output

    cout << endl;
    return 0;
}
```

```
// invariant: we have written r rows so far
for (int r = 0; r != rows; ++r) {
    string::size_type c = 0;
    // invariant: we have written c characters so far in the current row
    while (c != cols) {
        // is it time to write the greeting?
        if (r == pad + 1 && c == pad + 1) {
            cout << greeting;
            c += greeting.size();
        } else {
            // are we on the border?
            if (r == 0 || r == rows - 1 || c == 0 || c == cols - 1)
                cout << "*";
            else
                cout << " ";
            ++c;
        }
    }
}
```

2.6 计 数

□ 从0开始计数

```
for(int r=0; r!=rows; ++r){  
    // write a row  
}
```

vs

```
for(int r=1; r<=rows; ++r){  
    // write a row  
}
```

1) 不对称区间比对称区间容易使用

- $[m,n)$ 包含 $n-m$ 个元素, $[m,n]$ 包含 $n-m+1$ 个元素
- 空区间: 不对称区间 $[n,n)$, 对称区间 $[n,n-1]$

2) 更容易表达循环不变式

```
for(int r=0; r!=rows; ++r){  
    // write a row  
}
```

vs

```
for(int r=1; r<=rows; ++r){  
    // write a row  
}
```

- 从0开始计数, 使不变式直接明了.
- 从1开始计数的不变式?

将要输出第r行: while语句最后一次测试条件时, 不变式为假.

3) 选择!=而不是<=作比较操作符

- 一个循环结束时, 它会影响我们对程序状态的了解.
- 从0开始计数:
想确保循环重复执行rows次, 使用 $r \neq \text{rows}$ 作为条件;
循环可能重复执行rows次或更多次, 使用 $r < \text{rows}$.
- 从1开始计数?
循环可能重复执行rows次或更多次时, 使用 $r \leq \text{rows}$;
想确保循环重复执行rows次, 必须使用 $r \neq \text{rows} + 1$ 作为条件.

小 结

□ 表达式

- C++从C继承了丰富的操作符集合. 通过重新定义内置操作符在类对象中的意义, 扩展了语言核心.

□ 隐式类型转换——为了保持精度

- 小类型转换为大类型, 有符号转换为无符号.
- bool: 内置类型, 表示真假值.
- size_t: 无符号整数类型(<cstdlib>), 可包含任意对象的长度.

- `string::size_type`: 无符号整数类型, 可包含任意string对象的长度

□ 语句

- `using namespace-name::name`: 定义name为namespace-name::name的同义词.
- `type-name name(args)`: 定义type-name类型的变量name, 并用参数args初始化(构造).

□ 半开区间