数据结构 ： 2/8 进制转换器 实验报告

毛子恒 李臻 张梓靖

2020 年 10 月 12 日

小组成员

班级：2019211309	姓名：毛子恒	学号：2019211397	分工：代码 文档
班级：2019211310	姓名：李臻	学号：2019211458	分工：测试 文档
班级：2019211308	姓名：张梓靖	学号：2019211379	分工：文档

目录

1 需求分析

1.1 题目描述

在由序号为 1 至 n 的 n 个元素依次排列并且首尾相接而组成的环中，规定初始时从序号 1 开始依次经过 $2, 3, ...$ 元素走到第 n 个元素的方向为正方向。

初始时以第 x 个元素为起点 st，重复以下过程 $n-1$ 次：以 st 为第 1 个元素，沿正方向找到第 y 个元素 del，从环中删除 del 元素，再将原 del 的下一个元素作为新的 st。

求经过 $n-1$ 次操作之后，环中仅剩的一个元素的序号是否是 1。

1.2 输入描述

程序从标准输入中读入数据。输入一行三个整数，用空格分隔，分别表示 n, x, y。

其中各个值的范围需要满足 $1 < n \leq 10^4 \quad 0 < x \leq n \quad 0 < y \leq 5 \times 10^4$。

1.3 输出描述

程序向标准输出中输出结果。

输出分为三种情况：

1. 输入合法，程序正常运行结束。此时输出两行，第一行一个字符串"Yes" 或者"No"（不带引号），分别表示最后一个元素是/不是 1，第二行一个数字，表示最后一个元素的序号。
2. 输入不合法。此时输出一行一个字符串"Please check your input."（不带引号）。
3. 程序发生运行时错误，比如内存分配失败。此时程序没有输出。

1.4 样例输入输出

1.4.1 样例输入输出 1

【输入】

```
10 1 3
```

【输出】

```
No
4
```

1.4.2 样例输入输出 2

【输入】

```
10 3 7
```

【输出】

```
Yes
1
```

1.4.3 样例输入输出 3

【输入】

```
100 87 305
```

【输出】

```
No
50
```

1.4.4 样例输入输出 4

【输入】

```
1000 725 801
```

【输出】

```
No
798
```

1.4.5 样例输入输出 5

【输入】

```
1 1 3
```

【输出】

```
Please check your input.
```

1.4.6 样例输入输出 6

【输入】

```
5 6 3
```

【输出】

```
Please check your input.
```

1.5 程序功能

程序通过给定的 n, x, y 计算出最后环中仅剩的元素序号，并且与 1 比较。

2 概要设计

2.1 问题解决的思路

使用单循环链表维护此约瑟夫环，首先在链表中依次插入 n 个结点表示 n 名队员，以 now 指针模拟计数过程。从头结点找到第 x 个结点，此后执行以下操作 $n-1$ 次：找到当前结点之后的第 $y-1$ 个结点，删除这个结点。此题中单循环链表实现了初始化、判空、在指定位置增加节点、删除指定位置的节点、释放空间这五种操作。

2.2 栈的定义

2.3 主程序的流程

1. 输入
2. 初始化链表
3. 在链表中依次插入 n 个结点
4. 找到第 x 个结点
5. 循环 $n-1$ 次：找到当前节点之后的第 $y-1$ 个结点，删除这个结点
6. 输出
7. 释放空间

2.4 各程序模块之间的层次关系

函数调用关系图如图 1。

图 1: 函数的调用关系

3 详细设计

3.1 栈的实现

链表设计种基本操作的伪代码算法如下：

3.2 函数的调用关系图

如 2.4 所示。

4 调试分析报告

4.1 调试过程中遇到的问题和思考

初步实现后，测试样例时发现对于前导零的情况没有处理，遂增加查询栈顶操作，并且在主程序中增加弹出栈顶 0 元素的循环。

之后对串全为 0 的情况增加特判。

对于规模较大的数据测试时发现指针会访问到无效位置，发现是 realloc 操作之后没有更新 *top* 指针的位置所致。遂增加更新 *top* 指针的语句。

4.2 设计实现的回顾讨论

由于链表的删除操作实现是删除给定结点的后继，所以 now 指针始终指向当前正在计数元素的前驱。

由于单循环链表中存在一个特殊的头结点，所以另实现一个函数，返回某个结点的后继（跳过头结点）。

删除操作的细节：由于 now 指向正在计数结点的前驱，删除某个结点之后 now 仍然指向原来被删节点的前驱，之后执行 $y-1$ 次寻找后继操作，now 便指向下一个待删除结点的前驱。

由于主函数对函数的调用足够严密，所以链表的实现没有考虑不符合前件的情况。

由于链表元素均为 int 类型，所以链表的实现中没有对元素类型进行抽象，并且多次使用赋值运算符更改元素值。

4.3 算法复杂度分析

initStack, isStackEmpty, pushStack, getStackTop, popStack, destroyStack 函数的时间复杂度均为 $O(1)$。

主程序复杂度为 $O(n^2)$，整体时间复杂度为 $O(n^2)$。

4.4 改进设想的经验和体会

4.4.1 改进 1

在主程序的这一部分：

```
for (int i = 1; i <= n; ++i) // 逐个添加元素
{
   addNode(now, i);
   now = nextNode(now);
}
now = list;
for (int i = 1; i < x; ++i) // 找到第x个元素的前驱
   now = nextNode(now);
```

可以另用一个指针变量在向链表逐个添加元素的同时记录第 $x-1$ 个元素的位置，以省去第二个循环。优化后的实现如下：

```
List temp = NULL;
for (int i = 1; i <= n; ++i)
{
   addNode(now, i);
   now = nextNode(now);
   if (i == x - 1) temp = now;
}
now = temp;
```

5 用户使用说明

使用 gcc 编译生成可执行文件。

```
gcc -o main -std=c11 main.c list.c
```

执行可执行文件：

```
./main
```

在 Windows cmd 下：

```
main
```

之后通过标准输入输入数据，输入格式参考 1.2 节的输入描述，结果通过标准输出返回。如果输入合法并且程序正常运行结束，主函数返回值为 0。

6 测试结果

测试环节分为三个步骤。

6.1 测试第一部分

对 1.4 节给出的样例进行测试。

6.2 测试第二部分

测试非法输入和边界条件。

【输入】

```
5 -1 2
```

【输出】

```
Please check your input.
```

6.3 测试第三部分

使用 Python 实现 2/8 进制转换器 (test.py) 如下：

```python
a = int(input()[:-1], 2)
print(oct(a)[2:])
```

将原解法与此解法比对。

测试在 macOS Catalina 10.15.6 下进行。

在 $LEN <= 10$，$LEN <= 1000$，$LEN <= 1000000$ 的范围下分别随机生成 1000 组测试数据，分别传入 main 和 test.py，并且比对两程序的输出。

3000 组数据中两程序的输出均相同。

数据生成程序 (data.cpp) 如下：

```cpp
#include <bits/stdc++.h>

using namespace std;

const int LEN = 1e4;

int main()
{
   srand(time(0));
   int n = rand() % LEN + 1;
```

```
   for (int i = 1; i <= n; ++i)
      printf("%d", rand() % 2);
   puts("#");
   return 0;
}
```

比对脚本 (chk.sh) 如下：

```
for i in {1..100}
do
   sleep 1
   ./data >in.in
   ./main <in.in >out.out
   python ./test.py <in.in >out1.out
   if ! diff out.out out1.out
   then
      break
   fi
   echo "Correct"
done
```