数据结构实验报告

题目：　约瑟夫环的链表实现

班级：计科一班　　姓名：范文　　学号：　PB18111679　　完成日期：2019.10.17

一．实验要求

1. 通过链表来实现约瑟夫环

2. 掌握 C++语言命令行传参

二．设计思路

1. ADT

为了实现约瑟夫环的功能，我构造了一个带头结点的单向循环链表，其 ADT 如下。

```
template<class T>
struct Node
{
    T data1;
    T data2;
    Node *next;
};

template<class T>
class CircularLink
{
    private:
        Node<T>* head;
        int size;
    public:
        CircularLink();
        Node<T> *get_head() const { return head; }
        int get_size() const { return size; }
        void push_back( const T& insertData1,const T& insertData2 );
        void insert(int order, const T& insertData1, const T& insertData2 );
        void erase(int index);
        void erase(Node<T> *deletePtr);
        ~CircularLink();
};

```

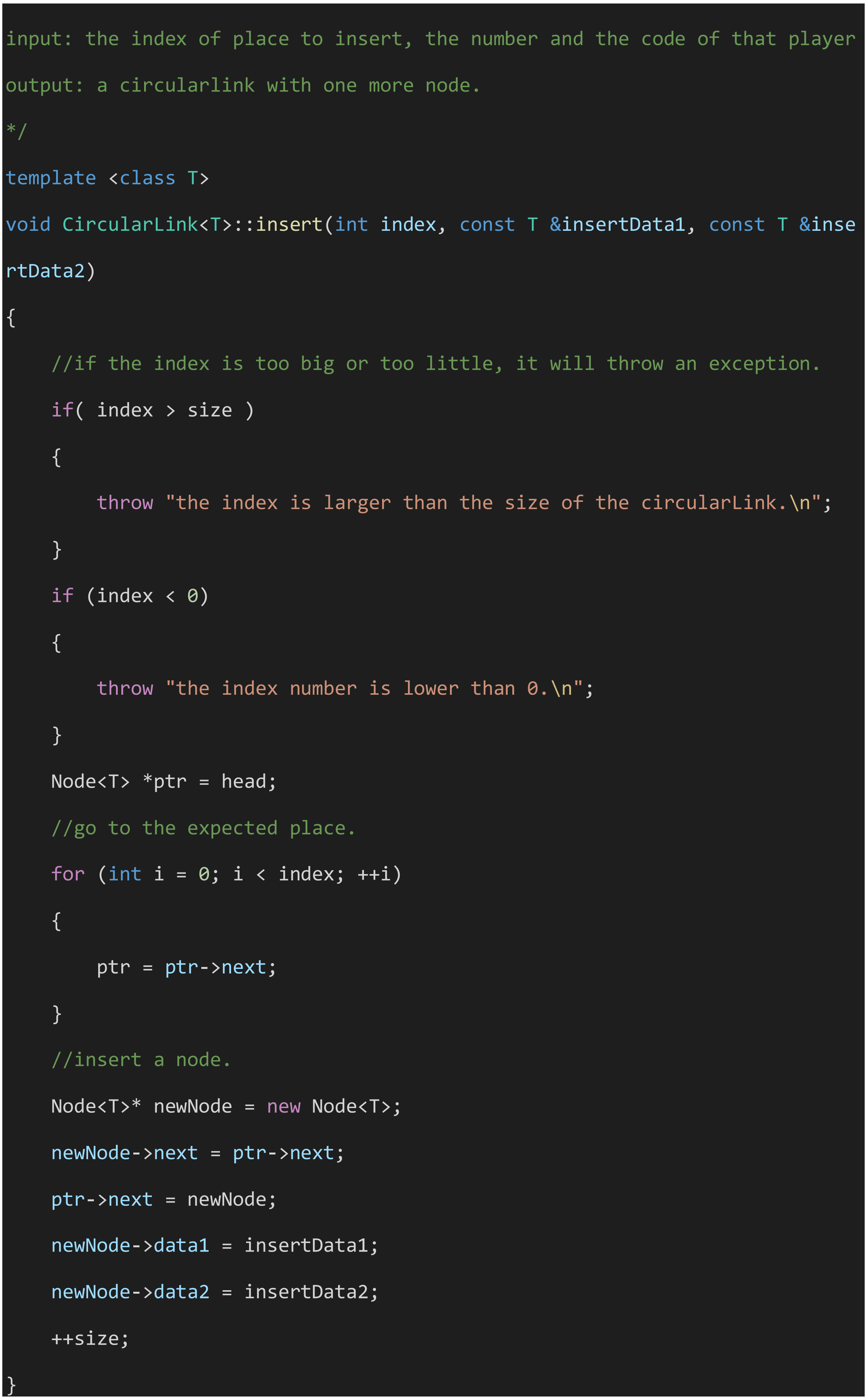
结点中有两个数据项，分别代表玩家的序号和代码。链表的方法包括构造函数，取头结点和大小的函数，插入函数，删除函数和析构函数。其具体方法如下：

（1）构造函数

```
template<class T>
CircularLink<T>::CircularLink()
{
    head = new Node<T>;
    head->next = head;
    size = 0;
}
```

（2）插入函数

```
/*
insert function
```



```
input: the index of place to insert, the number and the code of that player
output: a circularlink with one more node.
*/
template <class T>
void CircularLink<T>::insert(int index, const T &insertData1, const T &insertData2)
{
    //if the index is too big or too little, it will throw an exception.
    if( index > size )
    {
        throw "the index is larger than the size of the circularLink.\n";
    }
    if (index < 0)
    {
        throw "the index number is lower than 0.\n";
    }
    Node<T> *ptr = head;
    //go to the expected place.
    for (int i = 0; i < index; ++i)
    {
        ptr = ptr->next;
    }
    //insert a node.
    Node<T>* newNode = new Node<T>;
    newNode->next = ptr->next;
    ptr->next = newNode;
    newNode->data1 = insertData1;
    newNode->data2 = insertData2;
    ++size;
}
```

（3）其中可以利用 insert 函数写出插尾 push_back 函数。

```
template <class T>
void CircularLink<T>::push_back(const T& insertData1, const T& insertData2)
{
        insert(size , insertData1,insertData2);
}
```

（4）删除结点函数 erase

```

/*
erase the node that we anticipated.
input: a node that we want to erase.
output: a revised circularLink.
*/
template <class T>
void CircularLink<T>::erase( Node<T>* deletePtr)
{
    //try to traverse the link to find the anticipated node.
    Node<T>* ptr = head;
    while( ptr->next != head && ptr->next != deletePtr)
    {
        ptr = ptr->next;
    }
    //can not find the node.
    if( ptr -> next == head )
    {
        throw "fail to find the deletePtr to erase the node.\n";
    }
    //delete the node that we have found
    Node<T> *tempPtr = ptr -> next;
    ptr->next = ptr->next->next;
    delete tempPtr;
    --size;
}
```

（5）析构函数

```
template<class T>
CircularLink<T>::~CircularLink()
{
    Node<T> *frontPtr = head, *rearPtr = head->next;
    for (int i = 0; i < size ;++i)
    {
        frontPtr = rearPtr;
        rearPtr = rearPtr->next;
        delete frontPtr;
    }
    delete rearPtr;
}
```

2.程序模块

本程序中，先在 circularLink.h 中写好单向循环链表的结构。再向 main 函数进行命令行参数的传入，并将数据储存到构造的单向循环链表中。Joseph_simulation 进行约瑟夫环游戏的模拟。之后就可以输出结果了。

三．关键代码

核心函数 joseph_simulation 的代码如下：

```
/*simulating the process of the joseph ring
input: the original value of m, a single-direction circular link.
output: printing the player who is out in each round
        and showing the winner at last.
*/
void joseph_simulation(int originM,CircularLink<int> josephRing )
{
    ofstream mycout("tempResult.txt");
    int m = originM;
    int outOrder = 0;
    Node<int> *ptr = josephRing.get_head()->next;
    // more than 1 player, the game is still on.
    while( josephRing.get_size() > 1 )
    {
        //using mod to let m in the range of (1,size]
        // in order to reduce the repetition in the circulation.
        if( m % josephRing.get_size() != 0 )
            m = m % josephRing.get_size();
        else
            m = josephRing.get_size();

        // traverse the players in each round
        for (int i = 1; i < m;++i)
        {
            ptr = ptr->next;
            //skip the head of the circularLink.
            if( ptr == josephRing.get_head() )
            {
                ptr = ptr->next;
            }
        }
        // record the order and the code of the player who is out in the current round.
        m = ptr->data2;
        outOrder = ptr->data1;
        cout << outOrder << " is out.\n";
        mycout << outOrder << " is out.\n";
        // use a tempPtr to temporarily store the node to be erased later.
        Node<int> *tempPtr = ptr;
        //skip the head of the circularLink.
        do
        {
            ptr = ptr->next;
        } while (ptr == josephRing.get_head());
        josephRing.erase(tempPtr);

        // adjust m again to reduce repetition.
        if( m % josephRing.get_size() != 0 )
            m = m % josephRing.get_size();
        else
            m = josephRing.get_size();
    }
    cout << "the winner is " << josephRing.get_head()->next->data1 << endl;
    mycout << "the winner is " << josephRing.get_head()->next->data1 << endl;
    mycout.close();
}
```

其中需要包含<fstream>头文件来将结果输出到指定文本文件中。当约瑟夫环中只剩下一名玩家时，这名玩家就是 winner。

而 main 函数的代码如下：

```
/* pay attention: when using cmd to input the argument.
    1. the arguments are string.
    2.the first argument is the name of the cpp file.
*/
// the argument from the cmd:
// m represents the original code.
// n represents the amount of people
// n - numbers representing the code.
int main(int argc, char** argv)
{
    CircularLink<int> josephRing;
    int n  = atoi( argv[1] );
    int m = atoi(argv[2]);
    // constructing a circular link to simulate joseph ring.
    try
    {
        for (int i = 0; i < n;++i)
        {
            cout << "the argument is " << atoi(argv[i + 3]) << endl;
            //throw an exception when the amount of argument is not enough.
            if( atoi(argv[i + 3]) == 0 )
            {
                throw "the amount of arguments is not enough.\n";
            }
            josephRing.push_back( i+1, atoi(argv[i + 3]) );
        }
    }
    catch( const char * s)
    {
        cout << "EXCEPTION THROWN: \n";
        cout << s;
        exit(0);
    }
    try
    {
        joseph_simulation(m, josephRing);
    }
    catch( const char * s)
    {
        cout << "EXCEPTION THROWN: \n";
        cout << s;
        exit(0);
    }
    cout << "DONE!!!\n";
    return 0;
}
// the testing cmd input is  7 20 3 1 7 2 4 8 4
```

四．算法时空复杂度分析

在创建约瑟夫环的时候，由于尾插时通过遍历链表的方法找到尾结点，因此其时间复杂度为 O（n）。在删除节点的时候，由于采用了遍历链表找到了需要删除的结点，因此其时间复杂度为 O（n）。在约瑟夫环的模拟中，其时间复杂度为 O（m*n），其中 m 为输入的玩家数，n 为每位玩家的代

码大小。

五．代码测试

在编译了该 cpp 文件之后，在命令行中输入测试样例的数据

```
./joseph 7 20 3 1 7 2 4 8 4
```

之后在指定的文本文件 tempResult 中便出现了结果：

```
    6 is out.
    1 is out.
    4 is out.
    7 is out.
    2 is out.
    3 is out.
the winner is 5
```

与测试样例的结果相同。

六．实验总结

本次实验中，我掌握了单向循环链表的构造和应用，并学会了 M 命令行传参。但是，我有点好奇能不能先写一个带头结点的单向循环链表的抽象形式，然后让约瑟夫环继承它。另外，由于链表是单向的，因此尾插时时间复杂度较大，可以继续优化。

七．附件

CircularLink.h　存放单向循环链表类的成员和方法

Joseph.cpp　存放约瑟夫的模拟函数以及 main 函数

tempResult.txt　存放模拟后的结果。