#include<iostream>

using namespace std;

/\*一个刚出锅的山芋，在围成一圈的n（以编号1，2，3...n分别表示 ）个孩子间传递。

大家一起数数，从编号为1的孩子开始报数，每数一次，当前拿着山芋的孩子就把山芋交给紧邻其右的孩子。

一旦数到事先约定的某个数k ， 那个人就出列，他的下一个人又从1开始报数，数到数k的那个人就出列。

如此反复，最后剩下的那个孩子就是幸运者。

试实现算法josephus（int n, int k），输出孩子们出列的次序

程序输入：n k

程序输出：孩子们出列的次序

样例输入输出

样例1

输入:

8 3

输出:

3->6->1->5->2->8->4->7\*/

struct node

{

int data;

struct node\* next;

};

int main() {

int n, m;//孩子个数，约定的数

int i;

struct node\* head, \* tail, \* p, \* q;

head = new node;

head->next = NULL;//创建头结点

while (1) {

cin >> n >> m;

if (n == 0 || m == 0)

{

delete head;

break;

}

else {

tail = head;

//采用尾插法创建链表

for (i = 0; i < n; i++) {

p = new node;

p->data = i + 1;//填写孩子序号

tail->next = p;//和头结点连在一起，尾插

p->next = head->next;//循环

tail = p;

}

p = head->next;//对pq初始化，使其一前一后，方便删除节点，q是p的前驱

q = tail;

i = 1;

while (p != q) {//等于的时候只剩下一个了

if (i == m) {

cout << p->data << "->";

q->next = p->next;

delete p;

p = q->next;

i = 1;

}

else {

q = p;

p = p->next;

i++;

}

head->next = q;//其他链表题需要，这个不需要嘟

cout << p->data;

}

delete p;

head->next = NULL;

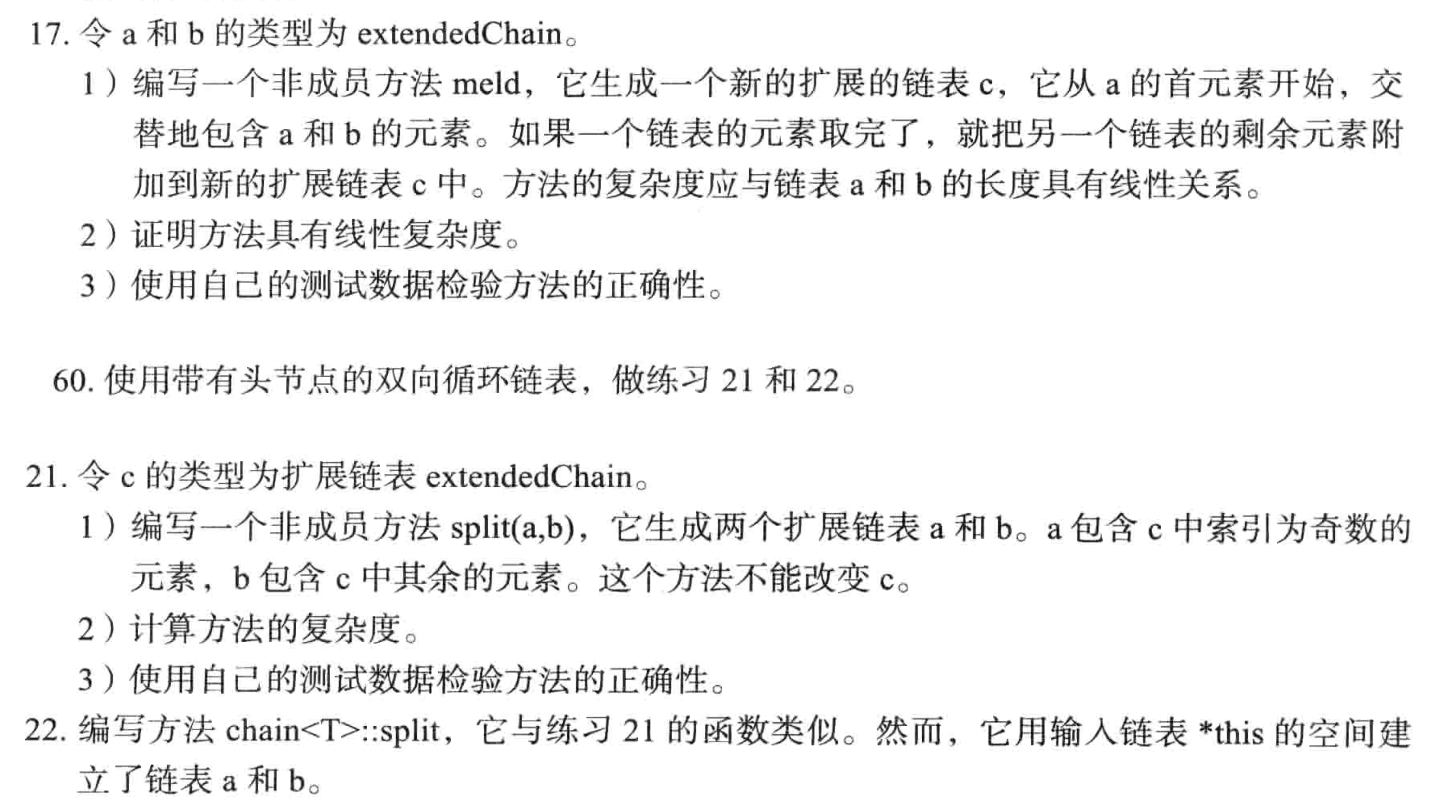
}

delete head;

return 0;

}

}



17

#include<iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int dat) : data(dat), next(NULL) {}

};

class extendedChain {

public:

Node\* head, \* tail;

extendedChain() : head(NULL), tail(NULL) {}

void insert(int val) {

Node\* newNode = new Node(val);

if (!head) {

head = tail = newNode;

return;

}

tail->next = newNode;

tail = newNode;

tail->next = NULL;

}

void print() {

Node\* curr = head;

while (curr) {

cout << curr->data << " ";

curr = curr->next;

}

cout << endl;

}

int count() {

int ans = 0;

Node\* curr = head;

while (curr) {

ans++;

curr = curr->next;

}

return ans;

}

~extendedChain() {

Node\* curr = head;

while (curr) {

Node\* temp = curr;

curr = curr->next;

delete temp;

}

}

};

void meld(extendedChain &a, extendedChain &b, extendedChain &c) {

Node\* curr1 = a.head;

Node\* curr2 = b.head;

while (curr1 && curr2) {

c.insert(curr1->data);

c.insert(curr2->data);

curr1 = curr1->next;

curr2 = curr2->next;

}

if (a.count() == b.count()) {

return;

}

else if (a.count() > b.count()) {

while (curr1) {

c.insert(curr1->data);

curr1 = curr1->next;

}

return;

}

else {

while (curr2) {

c.insert(curr2->data);

curr2 = curr2->next;

}

return;

}

}

int main(){

extendedChain a,b,c;

a.insert(1);

a.insert(2);

a.insert(3);

b.insert(1);

b.insert(2);

b.insert(3);

a.insert(4);

a.insert(5);

a.print();

b.print();

meld(a, b, c);

c.print();

cout << a.count() << endl;

cout << b.count() << endl;

cout << c.count() << endl;

return 0;

}

21

#include<iostream>

using namespace std;

class Node {

public:

int data;

Node\* next,\*pre;

Node(int dat) : data(dat), next(NULL) , pre(NULL) {}

};

class extendedChain {

public:

Node\* head, \* tail;

extendedChain() : head(NULL), tail(NULL) {}

void insert(int val) {

Node\* newNode = new Node(val);

if (!head) {

head = tail = newNode;

tail->next = head;

head->pre = tail;

return;

}

newNode->pre = tail;

tail->next = newNode;

tail = newNode;

tail->next = head;

head->pre = tail;

}

void append(int val) {

Node\* newNode = new Node(val);

if (!head) {

head = tail = newNode;

tail->next = head;

head->pre = tail;

return;

}

newNode->next = head;

head->pre = newNode;

newNode->pre = tail;

tail->next = newNode;

head = newNode;

}

void print() {

Node\* curr = head;

while (curr!=tail) {

cout << curr->data << " ";

curr = curr->next;

}

cout << curr->data << " ";

//cout << curr->next->data << " ";

//cout << head->pre->data << " ";

cout << endl;

}

int count() {

int ans = 0;

Node\* curr = head;

while (curr) {

ans++;

curr = curr->next;

}

return ans;

}

~extendedChain() {

Node\* curr = head;

while (curr) {

Node\* temp = curr;

curr = curr->next;

delete temp;

}

}

};

void split(extendedChain& a, extendedChain& b, extendedChain& c) {

Node\* curr = c.head;

while (curr->next != c.head && curr->next->next != c.head) {

a.insert(curr->data);

curr = curr->next->next;

}

a.insert(curr->data);

if (curr->next == c.head) {

curr = curr->pre;

while (curr->pre != c.head) {

b.append(curr->data);

curr = curr->pre->pre;

}

b.append(curr->data);

}

else if (curr->next->next == c.head) {

curr = curr->next;

while (curr->pre != c.head) {

b.append(curr->data);

curr = curr->pre->pre;

}

b.append(curr->data);

}

return;

}

int main() {

extendedChain a, b, c;

for (int i = 1; i <= 8; i++) {

c.insert(i);

}

split(a, b, c);

a.print();

b.print();

c.print();

cout << a.count() << endl;

cout << b.count() << endl;

cout << c.count() << endl;

return 0;

}

22

#include<iostream>

using namespace std;

class Node {

public:

int data;

Node\* next, \* pre;

Node(int dat) : data(dat), next(NULL), pre(NULL) {}

};

class extendedChain {

public:

Node\* head, \* tail;

extendedChain() : head(NULL), tail(NULL) {}

void insert(int val) {

Node\* newNode = new Node(val);

if (!head) {

head = tail = newNode;

tail->next = head;

head->pre = tail;

return;

}

newNode->pre = tail;

tail->next = newNode;

tail = newNode;

tail->next = head;

head->pre = tail;

}

void append(int val) {

Node\* newNode = new Node(val);

if (!head) {

head = tail = newNode;

tail->next = head;

head->pre = tail;

return;

}

newNode->next = head;

head->pre = newNode;

newNode->pre = tail;

tail->next = newNode;

head = newNode;

}

void print() {

Node\* curr = head;

while (curr->next != head) {

cout << curr->data << " ";

curr = curr->next;

}

cout << curr->data << " ";

//cout << curr->next->data << " ";

//cout << head->pre->data << " ";

cout << endl;

}

int count() {

int ans = 0;

Node\* curr = head;

while (curr) {

ans++;

curr = curr->next;

}

return ans;

}

/\*~extendedChain() {

Node\* curr = head;

while (curr) {

Node\* temp = curr;

curr = curr->next;

delete temp;

}

}\*/

};

void split(extendedChain& a, extendedChain& b, extendedChain& c) {

Node\* curr = c.head;

Node\* curr2 = c.head->next;

a.head = c.head;

b.head = curr2;

Node\* tmp = curr2->next;

while (curr2->next != c.head && curr2->next->next != c.head) {

curr2->next = curr2->next->next;

curr->next = tmp;

curr = curr->next;

curr2 = curr2->next;

tmp= curr2->next;

}

if (curr2->next->next == c.head) {

curr->next = curr2->next;

curr = curr->next;

}

curr->next = a.head;

curr2->next = b.head;

/\*if (curr->next == c.head) {

curr->next = c.head;

c.head->pre = curr;

curr = curr2;

while (curr->next->next != c.head) {

curr->next = curr->next->next;

curr = curr->next;

curr->pre = curr->pre->pre;

curr = curr->pre;

}

curr->next = c.head;

curr2->pre = curr;

}

else if (curr->next->next == c.head) {

curr->next = c.head;

c.head->pre = curr;

curr = curr2;

while (curr->next != c.head) {

curr->next = curr->next->next;

curr = curr->next;

curr->pre = curr->pre->pre;

curr = curr->pre;

}

curr->next = c.head;

curr2->pre = curr;

}\*/

return;

}

int main() {

extendedChain a, b, c;

for (int i = 1; i <= 7; i++) {

c.insert(i);

}

split(a, b, c);

a.print();

b.print();

//c.print();

//cout << a.count() << endl;

//cout << b.count() << endl;

//cout << c.count() << endl;

return 0;

}