数据结构与算法作业

理论课

将顺序表中的整型元素重排，奇数在前偶数在后。

```
template <class RandomAccessIterator>
void fun (RandomAccessIterator first, RandomAccessIterator last) {
    while (first < last) {
        for (; first < last && (*first) % 2; first++);
        for (last--; first < last && (*last) % 2 == 0; last--);
        if (first < last)
            swap(*first, *last);
    }
}
```

设计无头结点的单链表的插入算法

参考实验课的单链表作业 02/03.h ，详细的实现过程见下述链接：
https://gitee.com/Czile/homework/blob/master/02/03.h

下面仅给出类声明与插入算法：

类声明

```
template <class T>
class LinkedListNode {
    public:
        T val;
        LinkedListNode * next;
};

// This is an easy linkedlist.
template <class T>
class linkedlist {
    public:
        // Constructs a linkedlist.
        linkedlist();
        // Constructor a linkedlist from an interator.
        template <class RandomAccessIterator>
        linkedlist(RandomAccessIterator, RandomAccessIterator);
        // TODO: operator=
        // linkedlist<T>& operator=(linkedlist &);

        // Destroys the linkedlist
        ~linkedlist();
        // Insert an element in the position,
```

```
        // return a pointer that points to the first of the new inserted
elements.
        LinkedListNode<T>* insert(LinkedListNode<T>* position, const
T&val);
        // Delete an element in the position,
        // return a pointer that points to the next element.
        // NOTE: we will NOT ensure whether the position is in the
linkedlist or not!!
        LinkedListNode<T>* erase(LinkedListNode<T>* position);
        // If there are no element in the list, return true.
        bool empty() const noexcept;
        // return the pointer that points to the first element.
        LinkedListNode<T>* begin() noexcept;
        // return the pointer that points to the last+1 element.
        LinkedListNode<T>* end() noexcept;
        // return the number of elements in the linkedlist.
        uint64_t size() const noexcept;
        // Print the linkedlist.
        void print() const noexcept;
        // reverse the element of the linkedlist
        void reverse();
        // sort the element of the linkedlist
        // NOTE: to use this function, you MUST write a funcion of the
operator <
        void sort(bool(*cmp)(T&a, T&b)=[](T&a, T&b) {return a<b;});
        // Find the element by its value.
        // The second Pars is to find the nth element.
        // Return a pointer points to the value or return this.end().
        LinkedListNode<T>* find(const T&, unsigned=1);
        // Return the element in the list.
        // If the position is out of range, return *this.end() and throw an
error.
        T& operator[](unsigned);
    private:
        LinkedListNode<T>* _first, *_last;
        uint64_t _size;
        // Quick sort
        void _sort(LinkedListNode<T>* first, LinkedListNode<T>* last,
bool(*cmp)(T&a, T&b));
};
```

插入算法

```
template <class T>
LinkedListNode<T>* linkedlist<T>::insert(LinkedListNode<T>* position, const
T&val) {
    auto tmp = new LinkedListNode<T>;
    tmp -> val = position -> val;
    tmp -> next = position -> next;     //new a node, and make the
position's be the node.
    position -> val = val;
```

```
    position -> next = tmp;
    if (position == _last) {
        _last = tmp;
        _last -> next = _last;
    }
    _size++;
    return position;
}
```

设计算法求顺序表与单链表的逆

顺序表

```
template <class BidirectionalIterator>
void reverse (BidirectionalIterator first, BidirectionalIterator last) {
    while ((first!=last)&&(first!=--last)) {
        std::iter_swap (first,last);
        ++first;
    }
}
```

单链表

对于单链表，当然可以使用上述代码去实现逆置，但第二题中声明的 `linkedlist` 类型并未定义双向迭代器，故对于第二题的声明，我们需要给出其他的算法：

```
template <class T>
void linkedlist<T>::reverse() {
    auto p = _last;
    for (auto i=0; i<_size; i++) {
        p = insert(p, _first->val);
        erase(_first);
    }
}
```

將單鏈表中的數字、字母和其他字符分別拆分到循環鏈表中並輸出

同樣的，單鏈表的代碼採用習題二的代碼，下面只給出循環鏈表的代碼以及主函數代碼：

`cyclelist.h`

```
#ifndef _cyclelist_H_
#define _cyclelist_H_

#include <iostream>
```

```
template <class T>
class cyclelistNode {
    public:
        T val;
        cyclelistNode * next;
};

// This is an easy cyclelist.
template <class T>
class cyclelist {
    public:
        // Constructs a cyclelist.
        cyclelist();
        // Constructor a cyclelist from an interator.
        template <class RandomAccessIterator>
        cyclelist(RandomAccessIterator, RandomAccessIterator);
        // Destroys the cyclelist
        ~cyclelist();
        // Insert an element in the position, return a pointer that points
to the first of the newly inserted elements.
        cyclelistNode<T>* insert(cyclelistNode<T>* position, const T&val);
        // Delete an element in the position, return a pointer that points
to the next element.
        // NOTE: we will NOT ensure whether the position is in the
cyclelist or not!!
        cyclelistNode<T>* erase(cyclelistNode<T>* position);
        // If there are no element in the list, return true.
        bool empty() const noexcept {
            return (!_size);
        }
        // return the pointer that points to the first element.
        cyclelistNode<T>* begin() noexcept {
            return _first;
        }
        cyclelistNode<T>* end() noexcept {
            return _last;
        }
        // return the number of elements in the cyclelist.
        uint64_t size() const noexcept {
            return _size;
        }
        // Print the cyclelist.
        void print() const noexcept{
            for (auto p = _first; p != _last; p = p -> next)
                std::cout<<(p->val)<<" ";
            std::cout<<std::endl;
        }
        // reverse the element of the cyclelist
        void reverse();
        // sort the element of the cyclelist
        // NOTE: to use this function, you MUST write a funcion of the
operator <
        void sort(bool(*cmp)(T&a, T&b)=[](T&a, T&b) {return a<b;});
        // Find the element by its value.
```

```
        // The second Pars is to find the nth element.
        // Return a pointer points to the value or return this.end().
        cyclelistNode<T>* find(const T&, unsigned=1);
        // Return the element in the list.
        // If the position is out of range, return *this.end() and throw an
error.
        T& operator[](unsigned);
    private:
        cyclelistNode<T>* _first, *_last;
        uint64_t _size;
        // Quick sort
        void _sort(cyclelistNode<T>* first, cyclelistNode<T>* last,
bool(*cmp)(T&a, T&b));
};

template <class T>
cyclelist<T>::cyclelist() {
    _last = new cyclelistNode<T>;
    // This is to ensure the pointer is safe.
    _last->next = _first;
    _first = _last;
    _size = 0;
}

template <class T>
template <class RandomAccessIterator>
cyclelist<T>::cyclelist(RandomAccessIterator first, RandomAccessIterator
end) {
    for (auto i=0; first != end; first++)
        insert(_last, *first);
}

template <class T>
cyclelist<T>::~cyclelist() {
    while (!empty()) {
        erase(_first);
    }
    delete _first;
}

template <class T>
cyclelistNode<T>* cyclelist<T>::insert(cyclelistNode<T>* position, const
T&val) {
    auto tmp = new cyclelistNode<T>;
    tmp -> val = position -> val;
    tmp -> next = position -> next;     //new a node, and make the
position's be the node.
    position -> val = val;
    position -> next = tmp;
    if (position == _last) {
        _last = tmp;
        _last -> next = _first;
    }
    _size++;
```

```
    return position;
}

template <class T>
cyclelistNode<T>* cyclelist<T>::erase(cyclelistNode<T>* position) {
    if (position == _last)
        return _last;
    auto tmp = position -> next;
    position -> val = position -> next -> val;
    position -> next = position -> next -> next;
    delete tmp;
    if (tmp == _last) {
        _last = position;
        position -> next = _first;
    }
    _size--;
    return position;
}

template <class T>
void cyclelist<T>::reverse() {
    auto p = _last;
    for (auto i=0; i<_size; i++) {
        p = insert(p, _first->val);
        erase(_first);
    }
}

template <class T>
void cyclelist<T>::sort(bool(*cmp)(T&a, T&b)) {
    _sort(_first, _last, cmp);
}

template <class T>
void cyclelist<T>::_sort(cyclelistNode<T>* first, cyclelistNode<T>* last,
bool(*cmp)(T&a, T&b)) {
    if (first == last)
        return;
    auto mid = first;
    for (auto p = first -> next; p != _last && p != last;)
        if (cmp(p->val, mid->val)) {
            first = insert(first, p->val);
            if (p->next == last)
                last = p = erase(p);
            else
                p = erase(p);
        }
        else
            p = p -> next;
    _sort(first, mid, cmp);
    _sort(mid->next, last->next, cmp);
}

template <class T>
```

```
cyclelistNode<T>* cyclelist<T>::find(const T& val, unsigned k) {
    if (!k)
        return _last;
    auto p = _first;
    for (; p != _last && k;k && (p = p -> next))
        if (p->val == val)
            k--;
    return p;
}

template <class T>
T& cyclelist<T>::operator[] (unsigned k) {
    if (k < 0 || k >= _size) {
        throw "In delist::operator[]: the index is out of range.\n";
        return _last -> val;
    }
    auto p = _first;
    for (unsigned i=0; i<k; i++)
        p = p -> next;
    return p -> val;
}

#endif
```

main.cpp

```
#include <bits/stdc++.h>
#include "linkedlist.h"
#include "cyclelist.h"
using namespace std;

int main() {
    srand(time(0));
    linkedlist<char> p;
    for (auto i=0; i<30; i++)
        p.insert(p.end(), rand() % 127);
    try {
        cyclelist<char> q[3];
        for (auto z = p.begin(); z != p.end(); z = z -> next) {
            if (z -> val >= '0' && z -> val <= '9')
                q[0].insert(q[0].begin(), z -> val);
            else if (z -> val >= 'a' && z -> val <= 'z' || z -> val >= 'A'
&& z -> val <= 'Z')
                q[1].insert(q[1].begin(), z -> val);
            else
                q[2].insert(q[2].begin(), z -> val);
        }
        q[0].print();
        q[1].print();
        q[2].print();
    } catch (const char * e) {
        cout<<e<<endl;
```

```
    }

    return 0;
}
```

刪除鏈表的重復元素

以單鏈表 `linkedlist.h` 爲例， 即第二題定義的單鏈表爲例，現在給出刪除重復元素的代碼：

```
template <class T>
void delSameElement(linkedlist<T> p) {
    p.sort();
    for (auto first = p.begin(); first -> next != p.end();) {
        if (first -> val == first -> next -> val)
            first = p.erase(first);
        else
            first = first -> next;
    }
}
```

用鏈表實現集合的操作

本題如實驗題第四題所示。本題代碼如下：

```
#ifndef _SETT_H_
#define _SETT_H_

#include "02.h"

template <class T>
class sett{
    public:
        // Constructor
        sett() {}
        // Destroys the set.
        ~sett() {}
        // Return a pointer that points to the first element in the set.
        denode<T>* begin() const noexcept {return p.begin();}
        // Return a pointer that points to the last element's next location
in the set.
        denode<T>* end() const noexcept {return p.end();}
        // Return true if the size of the size is 0;
        bool empty() const noexcept {return p.empty();}
        // Return the size of the list
        unsigned size() const noexcept {return p.size();}
        // Sort the list by the cmp function
        void sort(bool(*cmp)(T&, T&) = [](T&a, T&b){return a<b;})
{p.sort(cmp);}
        // Insert the element in the location.
```

```
        // Return a pointer points to the new element.
        // NOTE: if the last element exists, this operate will delete the
last element.
        void insert(const T&t) {if (p.find(t) == p.end()) p.insert(p.end(),
t);}
        // Delete the element who has the value.
        // Return a pointer points to the next element.
        void erase(const T&t) {
            auto tmp = p.find(t);
            if (tmp != p.end())
                p.erase(tmp);
        }
        // Find the element by its value.
        // The second Pars is to find the nth element.
        // Return a pointer points to the value or return this.end().
        denode<T>* find(const T&t) {return p.find(t);}
        // Operator
        sett& operator*=(sett&t) {
            for (auto _t = p.begin(); _t != p.end();)
                if (t.find(_t -> val) == t.end())
                    _t = p.erase(_t);
                else
                    _t = _t -> next;
            return *this;
        }
        sett& operator+=(sett&t) {
            for (auto _t = t.begin(); _t != t.end(); _t = _t -> next)
                insert(_t->val);
            return *this;
        }
        sett& operator-=(sett&t) {
            for (auto _t = p.begin(); _t != p.end();)
                if (t.find(_t -> val) != t.end())
                    _t = p.erase(_t);
                else
                    _t = _t -> next;
            return *this;
        }
        // Print the set
        void print() {p.print();}
    private:
        delist<T> p;
};



#endif
```

解決約瑟夫問題

本題即使不用循環鏈表也相當簡單，此處給出單鏈表的解決方案

```
---
# 数据结构与算法作业
> 本作業所有代碼均可在 [https://gitee.com/Czile/homework]() 中找到。

> **陳梓樂** 19336035 數學學院
---
## 理论课
### 将顺序表中的整型元素重排，奇数在前偶数在后。

```cpp
template <class RandomAccessIterator>
void fun (RandomAccessIterator first, RandomAccessIterator last) {
    while (first < last) {
        for (; first < last && (*first) % 2; first++);
        for (last--; first < last && (*last) % 2 == 0; last--);
        if (first < last)
            swap(*first, *last);
    }
}
```

设计无头结点的单链表的插入算法

参考实验课的单链表作业 02/03.h ，详细的实现过程见下述链接：
https://gitee.com/Czile/homework/blob/master/02/03.h

下面仅给出类声明与插入算法：

类声明

```
template <class T>
class LinkedListNode {
    public:
        T val;
        LinkedListNode * next;
};

// This is an easy linkedlist.
template <class T>
class linkedlist {
    public:
        // Constructs a linkedlist.
        linkedlist();
        // Constructor a linkedlist from an interator.
        template <class RandomAccessIterator>
        linkedlist(RandomAccessIterator, RandomAccessIterator);
        // TODO: operator=
        // linkedlist<T>& operator=(linkedlist &);

        // Destroys the linkedlist
        ~linkedlist();
```
```

```
        // Insert an element in the position,
        // return a pointer that points to the first of the new inserted
elements.
        LinkedListNode<T>* insert(LinkedListNode<T>* position, const
T&val);
        // Delete an element in the position,
        // return a pointer that points to the next element.
        // NOTE: we will NOT ensure whether the position is in the
linkedlist or not!!
        LinkedListNode<T>* erase(LinkedListNode<T>* position);
        // If there are no element in the list, return true.
        bool empty() const noexcept;
        // return the pointer that points to the first element.
        LinkedListNode<T>* begin() noexcept;
        // return the pointer that points to the last+1 element.
        LinkedListNode<T>* end() noexcept;
        // return the number of elements in the linkedlist.
        uint64_t size() const noexcept;
        // Print the linkedlist.
        void print() const noexcept;
        // reverse the element of the linkedlist
        void reverse();
        // sort the element of the linkedlist
        // NOTE: to use this function, you MUST write a funcion of the
operator <
        void sort(bool(*cmp)(T&a, T&b)=[](T&a, T&b) {return a<b;});
        // Find the element by its value.
        // The second Pars is to find the nth element.
        // Return a pointer points to the value or return this.end().
        LinkedListNode<T>* find(const T&, unsigned=1);
        // Return the element in the list.
        // If the position is out of range, return *this.end() and throw an
error.
        T& operator[](unsigned);
    private:
        LinkedListNode<T>* _first, *_last;
        uint64_t _size;
        // Quick sort
        void _sort(LinkedListNode<T>* first, LinkedListNode<T>* last,
bool(*cmp)(T&a, T&b));
};
```

插入算法

```
template <class T>
LinkedListNode<T>* linkedlist<T>::insert(LinkedListNode<T>* position, const
T&val) {
    auto tmp = new LinkedListNode<T>;
    tmp -> val = position -> val;
    tmp -> next = position -> next;     //new a node, and make the
position's be the node.
```

```
    position -> val = val;
    position -> next = tmp;
    if (position == _last) {
        _last = tmp;
        _last -> next = _last;
    }
    _size++;
    return position;
}
```

设计算法求顺序表与单链表的逆

顺序表

```
template <class BidirectionalIterator>
void reverse (BidirectionalIterator first, BidirectionalIterator last) {
    while ((first!=last)&&(first!=--last)) {
        std::iter_swap (first,last);
        ++first;
    }
}
```

单链表

对于单链表，当然可以使用上述代码去实现逆置，但第二题中声明的 `linkedlist` 类型并未定义双向迭代器，故对于第二题的声明，我们需要给出其他的算法：

```
template <class T>
void linkedlist<T>::reverse() {
    auto p = _last;
    for (auto i=0; i<_size; i++) {
        p = insert(p, _first->val);
        erase(_first);
    }
}
```

将單鏈表中的数字、字母和其他字符分别拆分到循環鏈表中並輸出

同樣的，單鏈表的代碼採用習題二的代碼，下面只給出循環鏈表的代碼以及主函數代碼：

`cyclelist.h`

```
#ifndef _cyclelist_H_
#define _cyclelist_H_

#include <iostream>
```

```
template <class T>
class cyclelistNode {
    public:
        T val;
        cyclelistNode * next;
};

// This is an easy cyclelist.
template <class T>
class cyclelist {
    public:
        // Constructs a cyclelist.
        cyclelist();
        // Constructor a cyclelist from an interator.
        template <class RandomAccessIterator>
        cyclelist(RandomAccessIterator, RandomAccessIterator);
        // Destroys the cyclelist
        ~cyclelist();
        // Insert an element in the position, return a pointer that points
to the first of the newly inserted elements.
        cyclelistNode<T>* insert(cyclelistNode<T>* position, const T&val);
        // Delete an element in the position, return a pointer that points
to the next element.
        // NOTE: we will NOT ensure whether the position is in the
cyclelist or not!!
        cyclelistNode<T>* erase(cyclelistNode<T>* position);
        // If there are no element in the list, return true.
        bool empty() const noexcept {
            return (!_size);
        }
        // return the pointer that points to the first element.
        cyclelistNode<T>* begin() noexcept {
            return _first;
        }
        cyclelistNode<T>* end() noexcept {
            return _last;
        }
        // return the number of elements in the cyclelist.
        uint64_t size() const noexcept {
            return _size;
        }
        // Print the cyclelist.
        void print() const noexcept{
            for (auto p = _first; p != _last; p = p -> next)
                std::cout<<(p->val)<<" ";
            std::cout<<std::endl;
        }
        // reverse the element of the cyclelist
        void reverse();
        // sort the element of the cyclelist
        // NOTE: to use this function, you MUST write a funcion of the
operator <
        void sort(bool(*cmp)(T&a, T&b)=[](T&a, T&b) {return a<b;});
```

```
        // Find the element by its value.
        // The second Pars is to find the nth element.
        // Return a pointer points to the value or return this.end().
        cyclelistNode<T>* find(const T&, unsigned=1);
        // Return the element in the list.
        // If the position is out of range, return *this.end() and throw an
error.
        T& operator[](unsigned);
    private:
        cyclelistNode<T>* _first, *_last;
        uint64_t _size;
        // Quick sort
        void _sort(cyclelistNode<T>* first, cyclelistNode<T>* last,
bool(*cmp)(T&a, T&b));
};

template <class T>
cyclelist<T>::cyclelist() {
    _last = new cyclelistNode<T>;
    // This is to ensure the pointer is safe.
    _last->next = _first;
    _first = _last;
    _size = 0;
}

template <class T>
template <class RandomAccessIterator>
cyclelist<T>::cyclelist(RandomAccessIterator first, RandomAccessIterator
end) {
    for (auto i=0; first != end; first++)
        insert(_last, *first);
}

template <class T>
cyclelist<T>::~cyclelist() {
    while (!empty()) {
        erase(_first);
    }
    delete _first;
}

template <class T>
cyclelistNode<T>* cyclelist<T>::insert(cyclelistNode<T>* position, const
T&val) {
    auto tmp = new cyclelistNode<T>;
    tmp -> val = position -> val;
    tmp -> next = position -> next;     //new a node, and make the
position's be the node.
    position -> val = val;
    position -> next = tmp;
    if (position == _last) {
        _last = tmp;
        _last -> next = _first;
    }
```

```
    _size++;
    return position;
}

template <class T>
cyclelistNode<T>* cyclelist<T>::erase(cyclelistNode<T>* position) {
    if (position == _last)
        return _last;
    auto tmp = position -> next;
    position -> val = position -> next -> val;
    position -> next = position -> next -> next;
    delete tmp;
    if (tmp == _last) {
        _last = position;
        position -> next = _first;
    }
    _size--;
    return position;
}

template <class T>
void cyclelist<T>::reverse() {
    auto p = _last;
    for (auto i=0; i<_size; i++) {
        p = insert(p, _first->val);
        erase(_first);
    }
}

template <class T>
void cyclelist<T>::sort(bool(*cmp)(T&a, T&b)) {
    _sort(_first, _last, cmp);
}

template <class T>
void cyclelist<T>::_sort(cyclelistNode<T>* first, cyclelistNode<T>* last,
bool(*cmp)(T&a, T&b)) {
    if (first == last)
        return;
    auto mid = first;
    for (auto p = first -> next; p != _last && p != last;)
        if (cmp(p->val, mid->val)) {
            first = insert(first, p->val);
            if (p->next == last)
                last = p = erase(p);
            else
                p = erase(p);
        }
        else
            p = p -> next;
    _sort(first, mid, cmp);
    _sort(mid->next, last->next, cmp);
}
```

```
template <class T>
cyclelistNode<T>* cyclelist<T>::find(const T& val, unsigned k) {
    if (!k)
        return _last;
    auto p = _first;
    for (; p != _last && k;k && (p = p -> next))
        if (p->val == val)
            k--;
    return p;
}

template <class T>
T& cyclelist<T>::operator[] (unsigned k) {
    if (k < 0 || k >= _size) {
        throw "In delist::operator[]: the index is out of range.\n";
        return _last -> val;
    }
    auto p = _first;
    for (unsigned i=0; i<k; i++)
        p = p -> next;
    return p -> val;
}

#endif
```

main.cpp

```
#include <bits/stdc++.h>
#include "linkedlist.h"
#include "cyclelist.h"
using namespace std;

int main() {
    srand(time(0));
    linkedlist<char> p;
    for (auto i=0; i<30; i++)
        p.insert(p.end(), rand() % 127);
    try {
        cyclelist<char> q[3];
        for (auto z = p.begin(); z != p.end(); z = z -> next) {
            if (z -> val >= '0' && z -> val <= '9')
                q[0].insert(q[0].begin(), z -> val);
            else if (z -> val >= 'a' && z -> val <= 'z' || z -> val >= 'A'
&& z -> val <= 'Z')
                q[1].insert(q[1].begin(), z -> val);
            else
                q[2].insert(q[2].begin(), z -> val);
        }
        q[0].print();
        q[1].print();
        q[2].print();
    } catch (const char * e) {
```

```
        cout<<e<<endl;
    }

    return 0;
}
```

刪除鏈表的重復元素

以單鏈表 `linkedlist.h` 爲例， 即第二題定義的單鏈表爲例，現在給出刪除重復元素的代碼：

```
template <class T>
void delSameElement(linkedlist<T> p) {
    p.sort();
    for (auto first = p.begin(); first -> next != p.end();) {
        if (first -> val == first -> next -> val)
            first = p.erase(first);
        else
            first = first -> next;
    }
}
```

用鏈表實現集合的操作

本題如實驗題第四題所示。本題代碼如下：

```
#ifndef _SETT_H_
#define _SETT_H_

#include "02.h"

template <class T>
class sett{
    public:
        // Constructor
        sett() {}
        // Destroys the set.
        ~sett() {}
        // Return a pointer that points to the first element in the set.
        denode<T>* begin() const noexcept {return p.begin();}
        // Return a pointer that points to the last element's next location
in the set.
        denode<T>* end() const noexcept {return p.end();}
        // Return true if the size of the size is 0;
        bool empty() const noexcept {return p.empty();}
        // Return the size of the list
        unsigned size() const noexcept {return p.size();}
        // Sort the list by the cmp function
        void sort(bool(*cmp)(T&, T&) = [](T&a, T&b){return a<b;})
{p.sort(cmp);}
```

```
        // Insert the element in the location.
        // Return a pointer points to the new element.
        // NOTE: if the last element exists, this operate will delete the
last element.
        void insert(const T&t) {if (p.find(t) == p.end()) p.insert(p.end(),
t);}
        // Delete the element who has the value.
        // Return a pointer points to the next element.
        void erase(const T&t) {
            auto tmp = p.find(t);
            if (tmp != p.end())
                p.erase(tmp);
        }
        // Find the element by its value.
        // The second Pars is to find the nth element.
        // Return a pointer points to the value or return this.end().
        denode<T>* find(const T&t) {return p.find(t);}
        // Operator
        sett& operator*=(sett&t) {
            for (auto _t = p.begin(); _t != p.end();)
                if (t.find(_t -> val) == t.end())
                    _t = p.erase(_t);
                else
                    _t = _t -> next;
            return *this;
        }
        sett& operator+=(sett&t) {
            for (auto _t = t.begin(); _t != t.end(); _t = _t -> next)
                insert(_t->val);
            return *this;
        }
        sett& operator-=(sett&t) {
            for (auto _t = p.begin(); _t != p.end();)
                if (t.find(_t -> val) != t.end())
                    _t = p.erase(_t);
                else
                    _t = _t -> next;
            return *this;
        }
        // Print the set
        void print() {p.print();}
    private:
        delist<T> p;
};



#endif
```

解決約瑟夫問題

本題即使不用循環鏈表也相當簡單，此處給出單鏈表的解決方案

```
void fun(linkedlist<int> p, int m) {
    for (auto first = p.begin(); p.empty() == false; ) {
        m--;
        if (m)
            first = first == p.end() ? p.begin() : first -> next;
        else {
            cout << first -> val;
            first = p.erase(first);
        }
    }
}
```

```

```