Списък

Трифон Трифонов

Структури от данни и програмиране, спец. Компютърни науки, 2 поток, 2024/25 г.

10–24 ноември 2022 г.

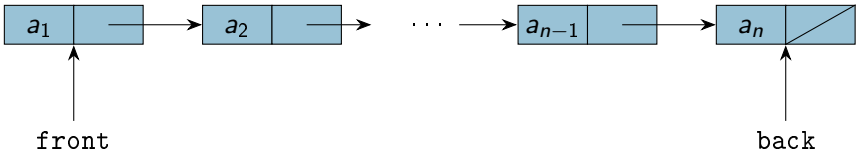
АТД: списък

Хомогенна линейна структура с последователен достъп до елементите

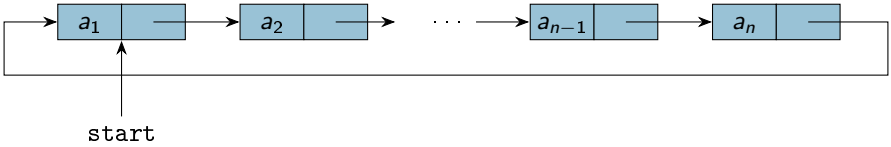
Операции:

- `create()` — създаване на празен списък
- `empty()` — проверка за празен списък
- `insert(x, p)` — включване на елемент `x` на дадена позиция `p`
- `delete(p)` — изключване на елемент на дадена позиция `p`
- `get(p)` — достъп до елемент на дадена позиция `p`

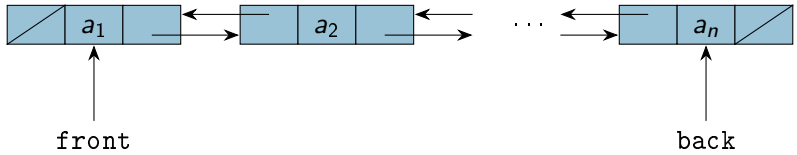
Едносвързано представяне



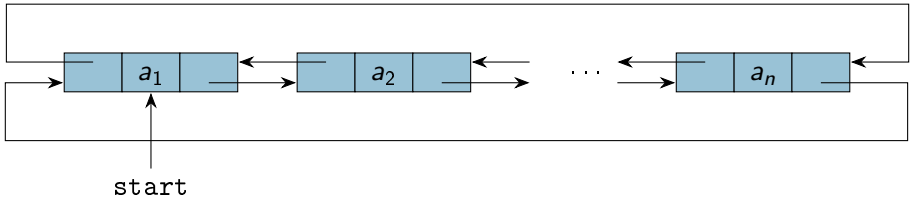
Едносвързано циклично представяне



Двусвързано представяне



Двусвързано циклично представяне



АТД: итератор

Абстракция на позиция, позволяваща обхождането на данни в дадена колекция

Операции:

- `begin()` – инициализация в началото на списъка
- `end()` – инициализация в края на списъка
- `next()` – преместване напред
- `prev()` – преместване назад
- `get()` – достъп до елемент на дадената позиция
- `valid()` – проверка за валидност

Итератор: физическо представяне

- при свързано представяне — указател към двойна или тройна кутия
- при последователно представяне — индекс на пореден елемент

Цикъл върху контейнери

В C++11 имаме възможност да итерираме по елементите в даден контейнер със следния синтаксис:

`for(`<тип> <елемент> `:` <израз>`)` <оператор>

Цикъл върху контейнери

В C++11 имаме възможност да итерираме по елементите в даден контейнер със следния синтаксис:

`for(`<тип> <елемент> `:` <израз>`)` <оператор>

- <израз> трябва да се оценява до контейнер

Цикъл върху контейнери

В C++11 имаме възможност да итерираме по елементите в даден контейнер със следния синтаксис:

`for(`<тип> <елемент> `:` <израз>`)` <оператор>

- <израз> трябва да се оценява до контейнер
- <тип> трябва съвпада с типа на елементите в контейнера

Цикъл върху контейнери

В C++11 имаме възможност да итерираме по елементите в даден контейнер със следния синтаксис:

`for(`<тип> <елемент> `:` <израз>`)` <оператор>

- <израз> трябва да се оценява до контейнер
- <тип> трябва съвпада с типа на елементите в контейнера
- <оператор> е тялото на цикъла, в което може да се използва <елемент>

Цикъл върху контейнери

В C++11 имаме възможност да итерираме по елементите в даден контейнер със следния синтаксис:

`for(`<тип> <елемент> `:` <израз>`)` <оператор>

- <израз> трябва да се оценява до контейнер
- <тип> трябва съвпада с типа на елементите в контейнера
- <оператор> е тялото на цикъла, в което може да се използва <елемент>
- ако <тип> е препратка, то <елемент> може да се използва за промяна на стойността на съответната позиция в контейнера

Цикъл върху контейнери

В C++11 имаме възможност да итерираме по елементите в даден контейнер със следния синтаксис:

for(<тип> <елемент> : <израз>) <оператор>

- <израз> трябва да се оценява до контейнер
- <тип> трябва съвпада с типа на елементите в контейнера
- <оператор> е тялото на цикъла, в което може да се използва <елемент>
- ако <тип> е препратка, то <елемент> може да се използва за промяна на стойността на съответната позиция в контейнера

Превежда се до:

```
<подходящ тип> const& c = <израз>;
for(<подходящ тип> it = c.begin(); it != c.end(); ++it) {
  <тип> <елемент> = *it;
  <оператор>
}
```

Достъп до производен клас на шаблон

Проблем: Какъв резултат трябва да връщат операциите ++, преместващи итераторите?

- `Iterator& operator++();`
  - връщаме псевдоним към обект от абстрактен клас, OK
- `Iterator operator++(int);`
  - връщаме обект от абстрактен клас, грешка!

Достъп до производен клас на шаблон

Проблем: Какъв резултат трябва да връщат операциите ++, преместващи итераторите?

- `Iterator& operator++();`
  - връщаме псевдоним към обект от абстрактен клас, OK
- `Iterator operator++(int);`
  - връщаме обект от абстрактен клас, грешка!

Трябва абстрактният базов клас да знае кой е конкретния му наследник! Възможно ли е това?

Достъп до производен клас на шаблон

Проблем: Какъв резултат трябва да връщат операциите ++, преместващи итераторите?

- `Iterator& operator++();`
  - връщаме псевдоним към обект от абстрактен клас, OK
- `Iterator operator++(int);`
  - връщаме обект от абстрактен клас, грешка!

Трябва абстрактният базов клас да знае кой е конкретния му наследник! Възможно ли е това?

Да! Базовият клас трябва да е **шаблон, приемащ производния си клас за параметър**

Curiously Recurring Template Pattern (CRTP)

```
template <typename Derived>
class Base {
  ...
};

class Derived : public Base<Derived> {
  ...
};
```

По този начин шаблонът Base може да се обръща към наследяващия го клас.

Решение на проблема с operator++

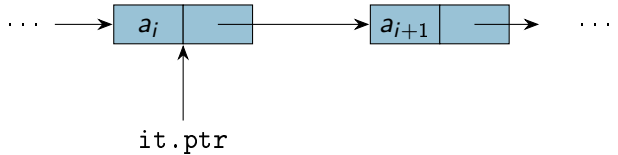
```
template <typename T, typename ConcreteIterator>
class Iterator {
  ...
  ConcreteIterator operator++(int) {
    ConcreteIterator save = (ConcreteIterator&)*this;
    ++(*this);
    return save;
  }
};
template <typename T>
class LinkedListIterator :
        public Iterator<T, LinkedListIterator<T> > {
  ...
};
```

Решение на проблема с operator++

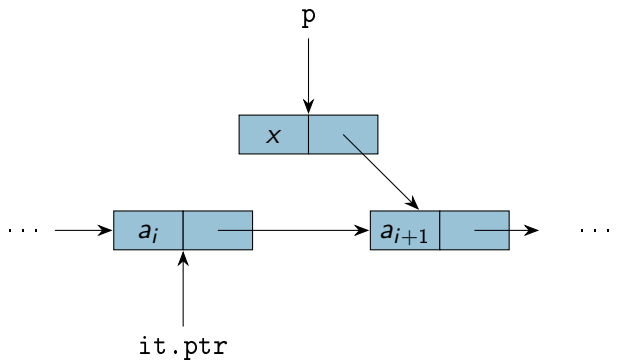
```
template <typename T, typename ConcreteIterator>
class Iterator {
  ...
  ConcreteIterator operator++(int) {
    ConcreteIterator save = (ConcreteIterator&)*this;
    ++(*this);
    return save;
  }
};
template <typename T>
class LinkedListIterator :
        public Iterator<T, LinkedListIterator<T> > {
  ...
};
```

Експлицитното преобразуване на типа се налага, понеже `*this` е от тип `Iterator&`, а конструкторът за копиране на `ConcreteIterator` очаква тип `ConcreteIterator&`.

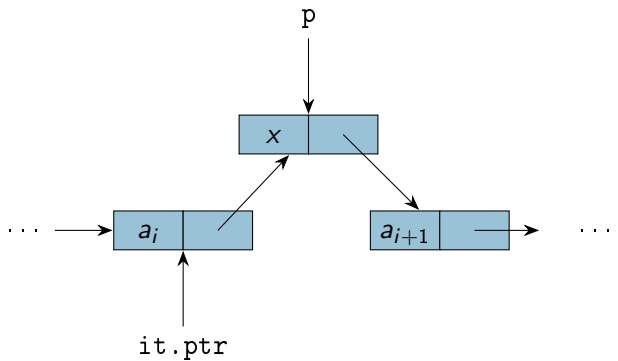
Включване на елемент в едносвързан списък



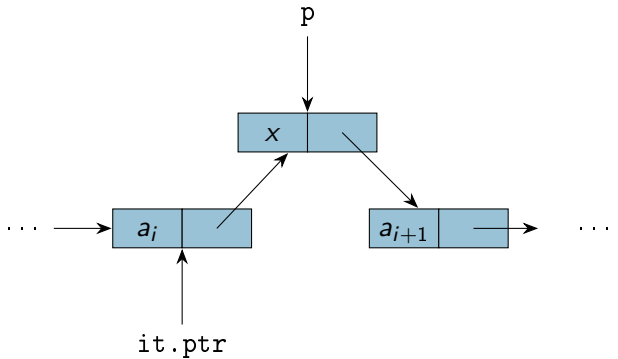
Включване на елемент в едносвързан списък



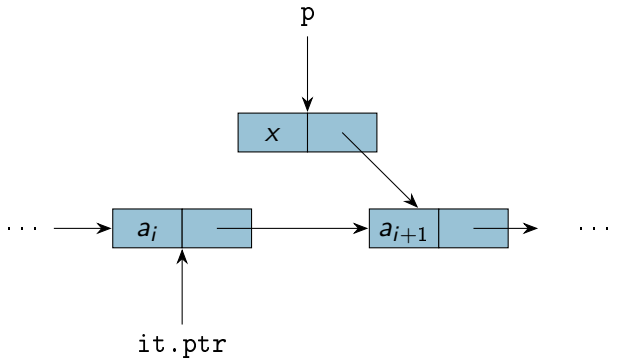
Включване на елемент в едносвързан списък



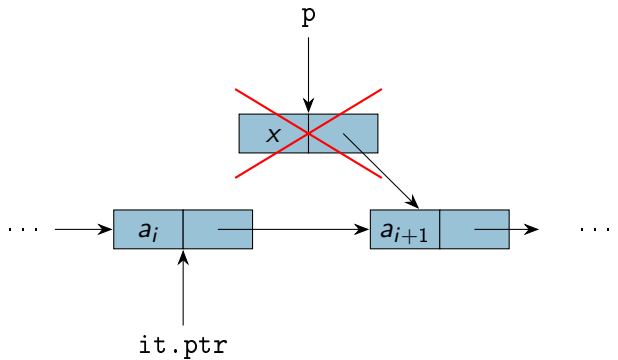
Изключване на елемент от едносвързан списък



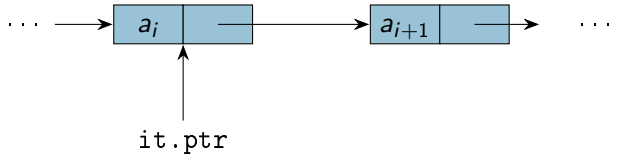
Изключване на елемент от едносвързан списък



Изключване на елемент от едносвързан списък



Изключване на елемент от едносвързан списък



Сложност на операции за едносвързан списък

Бързи операции (сложност $O(1)$ по време и памет):

- `insertAfter`
- `deleteAfter`
- `insertFirst`
- `insertLast`
- `deleteFirst`

Сложност на операции за едносвързан списък

Бързи операции (сложност $O(1)$ по време и памет):

- `insertAfter`
- `deleteAfter`
- `insertFirst`
- `insertLast`
- `deleteFirst`

Бавни операции (сложност $O(n)$ по време и $O(1)$ по памет):

- `insertBefore`
- `deleteLast`
- `deleteAt`
- `deleteBefore`

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък
- Включване на елемент преди първия

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък
- Включване на елемент преди първия
- Включване на елемент след последния

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък
- Включване на елемент преди първия
- Включване на елемент след последния
- Изключване на единствения елемент на списък

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък
- Включване на елемент преди първия
- Включване на елемент след последния
- Изключване на единствения елемент на списък
- Изключване на първия елемент

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък
- Включване на елемент преди първия
- Включване на елемент след последния
- Изключване на единствения елемент на списък
- Изключване на първия елемент
- Изключване на последния елемент

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък
- Включване на елемент преди първия
- Включване на елемент след последния
- Изключване на единствения елемент на списък
- Изключване на първия елемент
- Изключване на последния елемент
- Опит за изключване на елемент от празен списък

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък
- Включване на елемент преди първия
- Включване на елемент след последния
- Изключване на единствения елемент на списък
- Изключване на първия елемент
- Изключване на последния елемент
- Опит за изключване на елемент от празен списък
- Опит за изключване преди първия елемент

Гранични случаи

Случаи, които изискват особено внимание:

- Включване на елемент в празен списък
- Включване на елемент преди първия
- Включване на елемент след последния
- Изключване на единствения елемент на списък
- Изключване на първия елемент
- Изключване на последния елемент
- Опит за изключване на елемент от празен списък
- Опит за изключване преди първия елемент
- Опит за изключване след последния елемент

Задачи за списъци

Задача. Да се залепи на края на даден списък втори даден списък.

Задачи за списъци

Задача. Да се залепи на края на даден списък втори даден списък.

Решение №1 (абстрактно):
Добавяме елементите на втория списък на края на първия.

Задачи за списъци

Задача. Да се залепи на края на даден списък втори даден списък.

Решение №1 (абстрактно):
Добавяме елементите на втория списък на края на първия.

Решение №2 (конкретно):
Завързваме началото на втория списък за края на първия.

Задачи за списъци

Задача. Да се залепи на края на даден списък втори даден списък.

Решение №1 (абстрактно): $O(n)$
Добавяме елементите на втория списък на края на първия.

Решение №2 (конкретно): $O(1)$
Завързваме началото на втория списък за края на първия.

Задачи за списъци

Задача. Да се залепи на края на даден списък втори даден списък.

Решение №1 (абстрактно): $O(n)$
Добавяме елементите на втория списък на края на първия.

Решение №2 (конкретно): $O(1)$
Завързваме началото на втория списък за края на първия.

Задача. Да се обърне реда на елементите в даден списък.

Задачи за списъци

Задача. Да се залепи на края на даден списък втори даден списък.

Решение №1 (абстрактно): $O(n)$
Добавяме елементите на втория списък на края на първия.

Решение №2 (конкретно): $O(1)$
Завързваме началото на втория списък за края на първия.

Задача. Да се обърне реда на елементите в даден списък.

Решение №1 (абстрактно):
Преместваме последователно елементите след първия елемент на списъка в началото на списъка.

Задачи за списъци

Задача. Да се залепи на края на даден списък втори даден списък.

Решение №1 (абстрактно): $O(n)$
Добавяме елементите на втория списък на края на първия.

Решение №2 (конкретно): $O(1)$
Завързваме началото на втория списък за края на първия.

Задача. Да се обърне реда на елементите в даден списък.

Решение №1 (абстрактно):
Преместваме последователно елементите след първия елемент на списъка в началото на списъка.

Решение №2 (конкретно):
Разместване на указателите “на място”.

Сортиране чрез сливане

Задача. Да се раздели даден списък на два други с приблизително равна дължина.

Сортиране чрез сливане

Задача. Да се раздели даден списък на два други с приблизително равна дължина.

Задача. Да се слеят два възходящо подредени списъка в един.

Сортиране чрез сливане

Задача. Да се раздели даден списък на два други с приблизително равна дължина.

Задача. Да се слеят два възходящо подредени списъка в един.
Решение: Винаги избираме по-малкия елемент от двата списъка.

Сортиране чрез сливане

Задача. Да се раздели даден списък на два други с приблизително равна дължина.

Задача. Да се слеят два възходящо подредени списъка в един.
Решение: Винаги избираме по-малкия елемент от двата списъка.

Задача. Да се сортира дадена списък чрез сливане.

Сортиране чрез сливане

Задача. Да се раздели даден списък на два други с приблизително равна дължина.

Задача. Да се слеят два възходящо подредени списъка в един.
Решение: Винаги избираме по-малкия елемент от двата списъка.

Задача. Да се сортира дадена списък чрез сливане.
Решение:

1. Разделяме дадения списък на две.
2. Всеки от получените два списъка сортираме рекурсивно.
3. Сливаме двата сортирани списъка в един.

Функции от по-висок ред за списъци

Нека е даден списък $l = (a_1\ a_2\ a_3\ \ldots\ a_n)$.

- `foldr` — свиване надясно

$$a_1 \oplus \Big(a_2 \oplus \big(\ldots \oplus (a_n \oplus \bot)\ldots\big)\Big),$$

Функции от по-висок ред за списъци

Нека е даден списък $l = (a_1\ a_2\ a_3\ \ldots\ a_n)$.

- `foldr` — свиване надясно

$$a_1 \oplus \Big(a_2 \oplus \big(\ldots \oplus (a_n \oplus \bot)\ldots\big)\Big),$$

- `foldl` — свиване наляво

$$\Big(\ldots\big((\bot \oplus a_1) \oplus a_2\big) \oplus \ldots\Big) \oplus a_n$$

Функции от по-висок ред за списъци

Нека е даден списък $l = (a_1\ a_2\ a_3\ \ldots\ a_n)$.

- `foldr` — свиване надясно

$$a_1 \oplus \Big(a_2 \oplus \big(\ldots \oplus (a_n \oplus \bot)\ldots\big)\Big),$$

- `foldl` — свиване наляво

$$\Big(\ldots\big((\bot \oplus a_1) \oplus a_2\big) \oplus \ldots\Big) \oplus a_n$$

- `map` — изобразяване

$$f(a_1), f(a_2), \ldots, f(a_n)$$

Функции от по-висок ред за списъци

Нека е даден списък $l = (a_1\ a_2\ a_3\ \ldots\ a_n)$.

- `foldr` — свиване надясно

$$a_1 \oplus \Big(a_2 \oplus \big(\ldots \oplus (a_n \oplus \bot)\ldots\big)\Big),$$

- `foldl` — свиване наляво

$$\Big(\ldots\big((\bot \oplus a_1) \oplus a_2\big) \oplus \ldots\Big) \oplus a_n$$

- `map` — изобразяване

$$f(a_1), f(a_2), \ldots, f(a_n)$$

- `filter` — филтриране

$$a_{k_1}, a_{k_2}, \ldots, a_{k_m}, \text{ където } \begin{cases} p(a_{k_i}) = \texttt{true за } i \in \{1, \ldots, m\}, \\ p(a_k) = \texttt{false за } k \notin \{k_1, \ldots k_m\} \end{cases}$$

Задачи за функции от по-висок ред

Задача. Да се намери сумата от нечетните квадрати на числата в даден списък.

Задачи за функции от по-висок ред

Задача. Да се намери сумата от нечетните квадрати на числата в даден списък.

Задача. Да се намери произведението от най-малките положителни елементи на списък от списъци от числа.

λ-функции в C++11

- `[](<параметри>) -><тип> {<тяло>}`

λ-функции в C++11

- `[](<параметри>) -><тип> {<тяло>}`
- създава анонимна (λ) функция, дефинирана като: `<тип>` λ`(<параметри>) {<тяло>}`

λ-функции в C++11

- `[](<параметри>) -><тип> {<тяло>}`
- създава анонимна (λ) функция, дефинирана като:
  `<тип>` λ`(<параметри>) {<тяло>}`
- типът на израза е специален системен клас `ClosureType`, който има операция за преобразуване на типа до указател към функция с горната сигнатура, т.е.
  `<тип> (*)(<параметри>);`

λ-функции в C++11

- `[](<параметри>) -><тип> {<тяло>}`
- създава анонимна (λ) функция, дефинирана като:
  `<тип> λ(<параметри>) {<тяло>}`
- типът на израза е специален системен клас `ClosureType`, който има операция за преобразуване на типа до указател към функция с горната сигнатура, т.е.
  `<тип> (*)(<параметри>);`
- **Примери:**

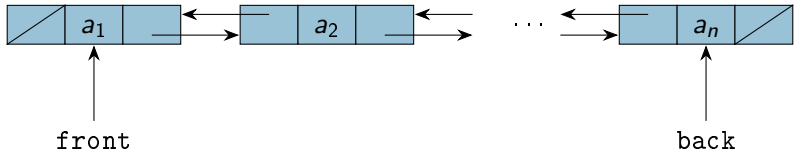
λ-функции в C++11

- `[](<параметри>) -><тип> {<тяло>}`
- създава анонимна (λ) функция, дефинирана като:
  `<тип> λ(<параметри>) {<тяло>}`
- типът на израза е специален системен клас `ClosureType`, който има операция за преобразуване на типа до указател към функция с горната сигнатура, т.е.
  `<тип> (*)(<параметри>);`
- **Примери:**
- `map([](int x) -> int { return x * x; }, l)`

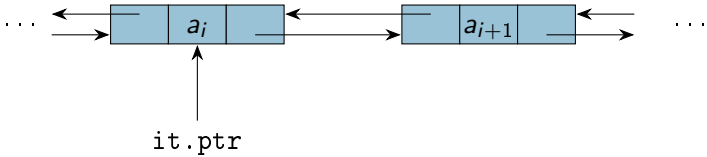
λ-функции в C++11

- `[](<параметри>) -><тип> {<тяло>}`
- създава анонимна (λ) функция, дефинирана като:
  `<тип> λ(<параметри>) {<тяло>}`
- типът на израза е специален системен клас `ClosureType`, който има операция за преобразуване на типа до указател към функция с горната сигнатура, т.е.
  `<тип> (*)(<параметри>);`
- **Примери:**
- `map([](int x) -> int { return x * x; }, l)`
- `foldr([](int x, int y) -> int { return x + y; }, 0, l)`

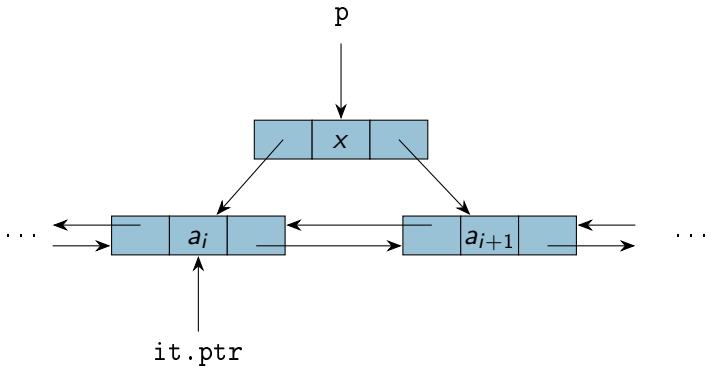
Двусвързано представяне



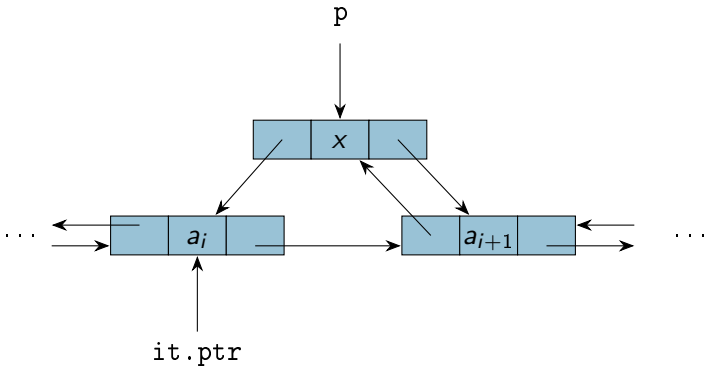
Включване на елемент в двусвързан списък



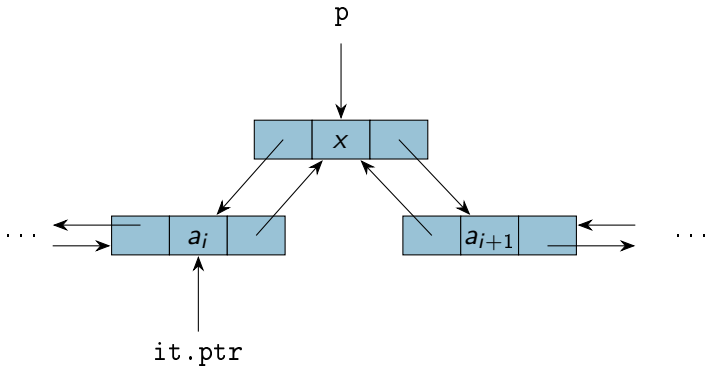
Включване на елемент в двусвързан списък



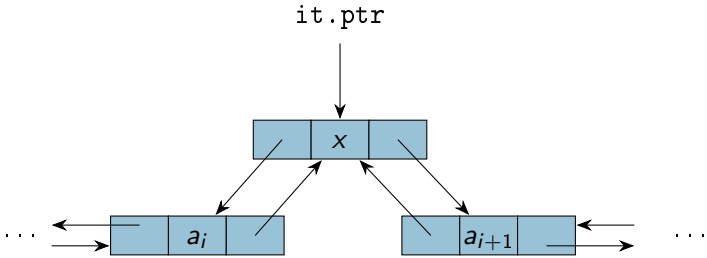
Включване на елемент в двусвързан списък



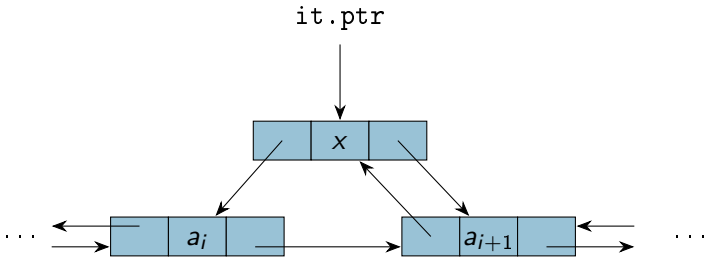
Включване на елемент в двусвързан списък



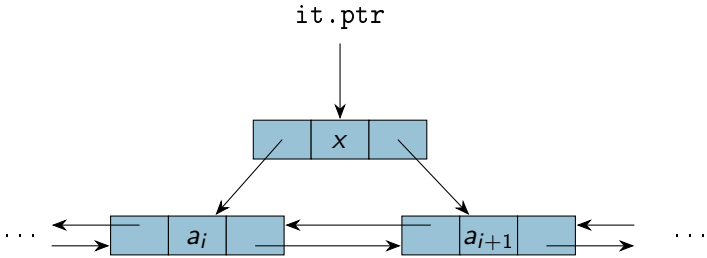
Изключване на елемент от двусвързан списък



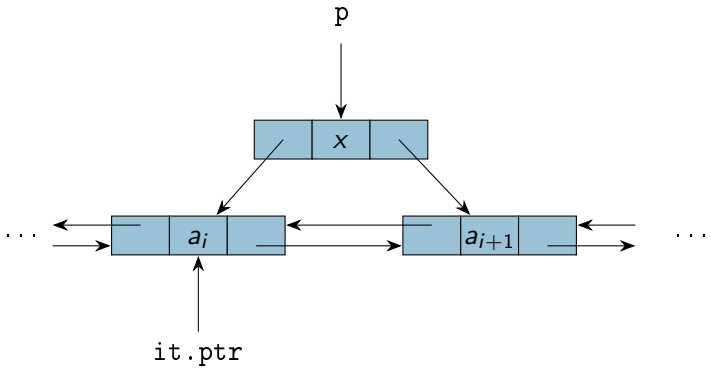
Изключване на елемент от двусвързан списък



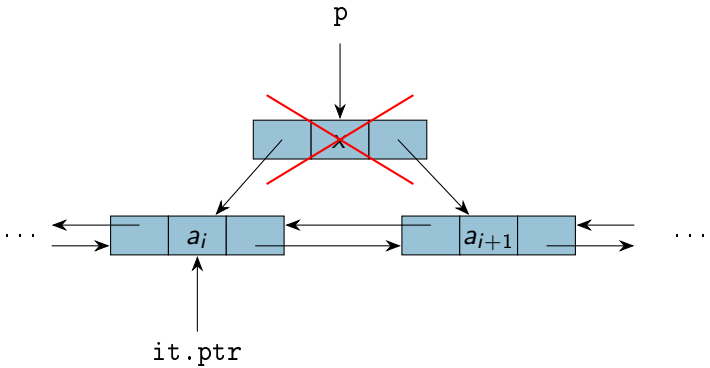
Изключване на елемент от двусвързан списък



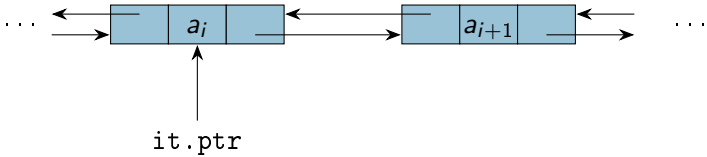
Изключване на елемент от двусвързан списък



Изключване на елемент от двусвързан списък



Изключване на елемент от двусвързан списък



Задачи за двусвързан списък

Задача. Да се провери дали даден двусвързан списък е палиндром.

Задачи за двусвързан списък

Задача. Да се провери дали даден двусвързан списък е палиндром.

Решение: Обхождаме едновременно в двете посоки.

std::list<T>

Реализацията на `std::list` в STL е двусвързана.

- `front()`, `back()` — първи и последен елемент
- `begin()`, `end()` — итератори към началото и края
- `rbegin()`, `rend()` — итератори за обратно обхождане
- `push_front()`, `push_back()` — вмъкване в началото/края
- `pop_front()`, `pop_back()` — изтриване от началото/края
- `insert()`, `erase()` — вмъкване/изтриване на позиция
- `splice()` — прехвърляне на елементи от един списък в друг
- `remove()`, `remove_if()` — филтриране по стойност/условие
- `merge()` — сливане на подредени списъци
- `sort()` — сортиране на списък (на място)
- `reverse()` — обръщане на списък
- `==,!=,<,>,<=,>=` — лексикографско сравнение на два списъка