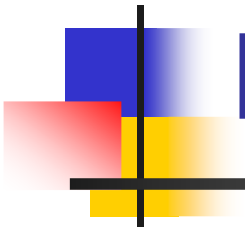
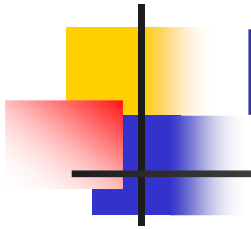


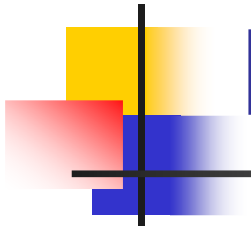
ИТЕРАТОРЫ

Примеры реализаций
на языке C++



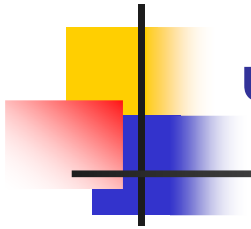
Итератор – это

- объект, который может перемещаться по коллекции однотипных объектов, перебирая их один за другим.



Внешние итераторы

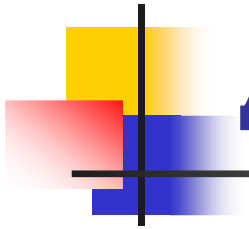
- имеют преимущество по сравнению со встроенными итераторами



Обычно итераторы реализуются через дружественные классы.

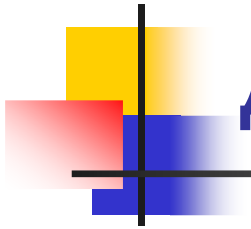
Для того чтобы коллекция могла использовать итератор, он должен объявить класс коллекции дружественным.

Перегруженные операции для итераторов



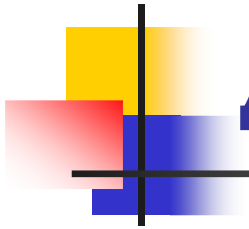
- Как правило, итератор имеет операцию доступа к элементу коллекции по ссылке. Такая операция может быть реализована путем перегрузки операции разыменования *.

Перегруженные операции для итераторов



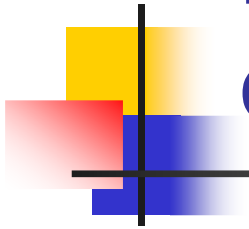
- Для итераторов предусматриваются также операции перемещения вперед и назад по коллекции объектов. Чаще всего для этого перегружаются операции ++ и –-

Перегруженные операции для итераторов



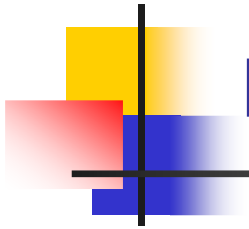
- Кроме того, операции = = и ! = обычно перегружаются для проверки равенства итераторов.

В классе-коллекции для использования итератора обычно создаются две функции:



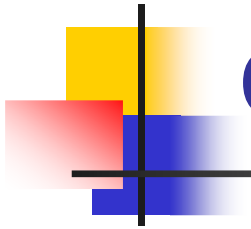
- iterator begin() – инициализация итератора ссылкой на первый элемент коллекции;
- iterator end() – значение сообщающее, что итератор достиг конца коллекции.

Очень важно оговорить правила поведения итератора



- после выполнения таких операций, как, например, вставка элемента после позиции итератора или удаление элемента в позиции итератора.

Итератор для линейного односвязного списка



Класс список spis предполагается уже описанным.

Список содержит элементы, являющиеся структурой el:

```
struct el
{
 int item;
 el* next;
};
```

Интерфейс класса итератора для списка

```
class spisIterator
{
    private:
    const spis *collection;
          el   *cur;
    public:
    spisIterator (const spis *s, el *e): collection(s), cur(e){}
    const int &operator *();
    spisIterator operator++(); // префиксный ++
    int operator == (const spisIterator &ri) const;
    int operator != (const spisIterator &ri) const;
    friend class spis;
};
```

Реализация методов итератора:

```
const int & spisIterator:: operator *()
{
        return cur -> item;
}

spisIterator spisIterator:: operator++()
{
    cur = cur->next;
    return *this;
}
```

Реализация методов итератора:

```
int spisIterator:: operator==(const spisIterator &ri) const
{
    return ((collection == ri.collection) &&
            (cur == ri.cur));
}

int spisIterator:: operator!=(const spisIterator &ri) const
{
    return ! (*this==ri);
}
```

Фрагмент класса Список

```
class spis
{
 private:
    el* head;
    el* tail;
 public:
 ...
 spisIterator begin() const;
 spisIterator end() const;
 spisIterator next (spisIterator it) throw (IterException);
};
```

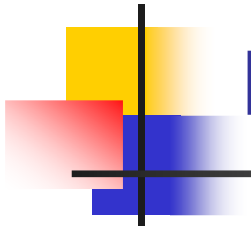
Инициализация итератора ссылкой на первый элемент списка:

```
spisIterator spis::begin() const
{
    spisIterator iter(this,head);
    return iter;
}
```

Метод, возвращающий признак того, что итератор достиг конца списка:

```
spisIterator spis::end() const
{
    spisIterator iter(this,NULL);
    return iter;
}
```

Метод, реализующий перемещение итератора



```
spisIterator spis::next(spisIterator it)
//throw (IterException)
{
    //if (it.collection != this)
        //throw IterException();
    //if (it == end())
        //throw IterException();
    return (++it);
}
```

Использование итератора для линейного односвязного списка

```
int main()
{
    spis s1;
    int n, a, i;
    cout<<"длина=\n"; cin>>n;
    cout<<"элементы списка=\n";
    for( i=0;i<n; i++)
    {
          cin>>a; s1+=a; // предполагается, что
          //операция += перегружена
          // для добавления элемента в конец списка
    }
```

Использование итератора для линейного односвязного списка

```
    spisIterator it = s1.begin();
    cout<<” список \n”;
    while (it!= s1.end())
    {
        cout << *it<<" ";
        it=s1.next(it); // ++it;
    }
    return 0;
}
```