Programování v C++ 2, 8. cvičení

návrhový vzor iterátor

Vladimír Jarý[1]

[1]Fakulta jaderná a fyzikálně inženýrská
České vysoké učení technické v Praze

Zimní semestr 2020/2021

Přehled

Shrnutí minule procvičené látky

- dědění
- potomek může zastoupit předka
- časná a pozdní vazba
- virtuální metody
- čistě virtuální metody a abstraktní třídy
- příklad: spojový seznam
  - iterátor seznamu

Zadání

Do třídy `List` přidejte veřejnou metodu, která ze seznamu odstraní prvek se zadanými daty.

Nalezení prvku se zadanými daty

- hledaná data ulož do zarážky
- prohledávej prvky seznamu od hlavy:
  - dokud nenajdeš hledaná data, posuň se na další prvek
- našla-li se data v zarážce, vrať `nullptr`, jinak vrať ukazatel na nalezený prvek

```
1 Element* List::find(Data &what){
2   last->setData(what);
3   Element *p = first->next;
4   for(;p->getData()!=what;p=p->next);
5   return (p == last ? nullptr : p);
6 }
```

Smazání prvku se zadanými daty

- nalezni prvek, který má být smazán
- za předchůdce nalezeného prvku napoj jeho následníka
- před následníka nalezeného prvku napoj jeho předchůdce
- prvek smaž a sniž počítadlo prvků

```
1  bool List::remove(Data &what){
2    Element *p = find(what);
3    if(!p){ return false; }
4    p->prev->next = p->next;}
5    p->next->prev = p->prev;
6    delete p;
7    counter--;
8    return true;
9  }
```

Iterátor seznamu

- zobecnění indexovacího operátoru
- objekt, který poskytuje konzistentní prostředek pro procházení různých typů kontejnerů
- odstiňuje uživatele kontejneru od detailu implementace
- umožňuje psát znovupoužitelný kód
- základní operace třídy `Iterator`:
  1. `reset`: vrátí iterátor před začátek datové struktury
  2. `next`: posune se na další prvek a oznámí, zda je k dispozici další prvek
  3. `data`: vráti data z aktuálního prvku
- tyto metody jsou ve třídě `Iterator` čistě virtuální
- implementace závisí na konkrétní datové struktuře, proto napsána až v odvozených třídách

Třídy `Iterator` a `ListIterator`

```
1  class Iterator{ //abstraktni trida
2  public:
3    Iterator();
4    virtual ~Iterator(){}
5    virtual void reset() = 0; //ciste virtualni metody
6    virtual Data& data() = 0; //referencni funkce
7    virtual bool next() = 0;
8  };
9
10 class List; //dopredna deklarace
11 class Element;
12
13 class ListIterator: public Iterator{
14   List *list;
15   Element *current;
16 public:
17   ListIterator(List *l);
18   virtual ~ListIterator(){}
19   virtual void reset();
20   virtual Data& data();
21   virtual bool next();
22 };
```

Implementace metod

```
1  ListIterator::ListIterator(List *l)
2  : list(l), current(list->first)
3  {}
4
5  void ListIterator::reset(){
6    current = list->first;
7  }
8
9  Data& ListIterator::data(){
10   return current->data;
11 }
12
13 bool ListIterator::next(){
14   if(current==list->last){
15     current = list->first->next;
16   } else{
17     current = current->next;
18   }
19   if(current==list->last){ return false; }
20   else{ return true; }
21 }
```

Použití iterátoru

Do seznamu doplníme metodu, která vytvoří iterátor:

```
1  ListIterator* List::getIterator(){
2    return new IteratorSeznamu(this);
3  }
```

Použití:

```
1   void processSequence(Iterator *i){ //parametrem je
2    i->reset();                       //ukazatel na predka
3    while(i->next())
4      cout << i->data() << endl;
5  }
6  int main(){
7    List l; ListIterator *it = l.getIterator();
8    //zadej do seznamu data a seznam pak zpracuj
9    processSequence(it);
10   delete it;
```

Funkce `processSequence` umí pracovat s libovolnou posloupností, ke které existuje iterátor

Příklad: dynamické pole

```
1  class Array{
2    unsigned int capacity;
3    Data *data;
4  public:
5    Array(unsigned int N);
6    Array(const Array &original);
7    ~Array();
8    Data& at(unsigned int idx){return data[idx];}
9    unsigned int size() const {return capacity;}
10   ArrayIterator* getIterator();
11 };
```

- vhodné přetížit operátor indexování

Implementace metod

```
1  Array::Array(unsigned int N)
2  : capacity(N), data(new Data[capacity])
3  {
4    memset(this->data, 0, capacity*sizeof(Data));
5  }
6
7  Array::Array(const Array &original)
8  : capacity(original.capacity),
9    data(new Data[capacity])
10 {
11   std::memcpy(this->data, original.data, capacity*sizeof(Data));
12 }
13
14 Array::~Array(){ delete [] data; }
15
16 ArrayIterator* Array::getIterator(){
17   return new ArrayIterator(this);
18 }
```

- funkce `memset` vyplní blok paměti zadanou hodnotou
- funkce `memcpy` zkopíruje blok paměti

Iterátor pole

```
1  ArrayIterator::ArrayIterator(Array *a)
2  : array(a), index(array->size()){}
3
4  void ArrayIterator::reset(){
5    index = array->size();
6  }
7
8  Data& ArrayIterator::data(){
9    return array->at(index);
10 }
11
12 bool ArrayIterator::next(){
13   if(index == array->size()){
14     index = 0;
15   } else{
16     index++;
17   }
18   return (index < array->size());
19 }
```

Příklad: hledání nejmenšího prvku

```
1  #include <climits>
2  #include "iterator.h"
3
4  Data findMin(Iterator *it){
5      Data min = INT_MAX;
6      while(it->next()){
7          if(it->data() < min)
8              min = it->data();
9      }
10     it->reset();
11     return min;
12 }
```

- opět bude fungovat pro libovolnou posloupnost, pro kterou je implementován iterátor

Závěr

Shrnutí

- práce se spojovým seznamem
  - přidání prvku na začátek seznamu
  - smazání všech prvků
  - destruktor
  - nalezení prvku
  - odstranění prvku
- návrhový vzor iterátor
  - abstraktní iterátor
  - iterátor seznamu
  - použití iterátoru: nalezení nejmenšího prvku posloupnosti