Programování v C++ 2, 7. cvičení

spojový seznam

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 a polymorfismus

- potomek může zastoupit předka
- ukazatel na potomka může zastoupit ukazatel na předka
- časná a pozdní vazba
- virtuální metody
- čistě virtuální metody a abstraktní třídy
- grafické objekty
  - abstraktní třída `Go` jako společný předek
  - obsahuje čistě virtuální metodu `nakresli()`
  - odvozené třídy (`Bod`, `Usecka`) ji musí implementovat
  - použití ukazatele na společného předka `Go*`

Zadání

Zadání příkladu

Napište třídu `List`, která bude implementovat obousměrně zřetězený seznam. Do třídy doplňte metody pro vložení a odebrání prvku na koncích seznamu a metodu, která odstraní ze seznamu všechny prvky. Dále napište abstraktní třídu `Iterator` a od ní odvoďte třídu `ListIterator`, která bude implementovat iterátor nad seznamem.

Prvek seznamu

- potřeba použít deklaraci **`friend`**
- vytvoření nového typu přejmenování

```
1  typedef int Data;
2  class Element{
3    Data data;
4    Element *prev. *next;
5  public:
6    Element(Data d);
7    void setData(Data &d);
8    Data getData() const;
9    void print() const;
10   friend class List;
11 };
```

Vytvoření seznamu

```
1  class List{
2    Element *first, *last;
3    unsigned int counter; //pocet prvku
4    void init(); //inicializace seznamu
5  public:
6    List(); //konstruktor
7    List(const List &other); //kopirovaci
8    //dalsi verejne metody...
9  };
```

1. vytvoř nový prvek a jeho adresu ulož do hlavy
2. vytvoř nový prvek a jeho adresu ulož do zarážky
3. hlavu zapoj před zarážku, za hlavu zapoj zarážku
4. nastav počet prvků na 0
5. využití inicializační části konstruktoru

Vytvoření seznamu

```
1  void List::init(){
2    first->next = last;
3    last->prev = first;
4  }
5
6  List::List()
7  : first(new Element(0)), last(new Element(0)),
8   counter(0)
9  {
10   init();
11 }
```

Kopírovací konstruktor

- implicitní kopírovací konstruktor vytváří mělkou kopii
- potřeba hluboká kopie - zkopírovat data prvek po prvku z originálu

```
1  List::List(const List &other)
2  : first(new Element(0)), last(new Element(0)), counter(0)
3  {
4    init();
5    copyFrom(other);
6  }
7
8  void List::copyFrom(const List &source){
9    Element *curr = source.first->next;
10   while(curr!=source.last){
11     this->pushBack(curr->data);
12     curr=curr->next;
13   }
14 }
```

Přidání prvku na konec seznamu

1. data ulož do zarážky
2. vytvoř nový prvek a zapoj jej za zarážku
3. před nově vytvořený prvek zapojte zarážku
4. zarážku posuň na nově vytvořený prvek

```
1 void List::pushBack(Data &d){
2   last->setData(d);
3   last->next = new Element(0);
4   last->next->prev = last;
5   last=last->next;
6   counter++;
7 }
```

- alokace paměti může selhat!

Odstranění posledního prvku

1. posuň zadní zarážku na předchůdce
2. smaž původní zadní zarážku
3. v nové zarážce vynuluj ukazatel na následníka

```
1 void List::popBack(){
2   last = last->prev;
3   delete last->next;
4   last->next = nullptr;
5   counter--;
6 }
```

Přidání na začátek a smazání ze začátku:

- nahradit `first` za `last` a naopak
- nahradit `prev` za `next` a naopak

Tisk seznamu

1. pomocný ukazatel `p` nastavte na hlavu seznamu
2. dokud je `p` různé od zarážky:
   1. tiskni data z prvku `p`
   2. prvek `p` posuň na následníka

```
1 void List::print() const{
2   for(Element *p = first->next; p!=last;
3         p = p->next){
4     p->print();
5   }
6 }
```

Čištění seznamu

- čištění = vymazání všech prvků s užitečnými daty
- dokud není seznam prázdný, maž první prvek

```
1 bool List::empty() const{
2   return (counter == 0);
3 }
4
5 void List::clear(){
6   while(!empty())
7     popFront();
8 }
```

Destruktor:

- smaž všechny prvky s užitečnými daty
- smaž obě zarážky

Implementace fronty

- možnosti implementace fronty:
  1. pomocí pole
  2. pomocí spojového seznamu
- návrhový vzor *adaptér*
- využití seznamu jako podkladové datové struktury
- skládání objektů
  - seznam jako soukromý atribut fronty
  - metody fronty využívají metody seznamu

Implementace fronty

```
1  class Queue{
2    List *list;
3  public:
4    Queue(){list = new List;}
5    ~Queue(){delete list;}
6    bool empty() const{
7      return list->empty();
8    }
9    void push(Data &d){
10     list->pushBack(d);
11   }
12   Data front() const{
13     return list->front();
14   }
15   void pop(){
16     list->popFront();
17   }
18 };
```

Závěr

Shrnutí

- objektová implementace spojového seznamu
- konstruktor, kopírovací konstruktor
- destruktor
- přidání prvků na konec a začátek seznamu
- odstranění prvků z konce a začátku seznamu
- tisk seznamu
- implementace fronty
  - návrhový vzor adaptér
  - využití skládání objektů