Mittuniversitetet
MID SWEDEN UNIVERSITY

DT019G Objektbaserad programmering i C++

Laboration:
Länkade listor

Martin Kjellqvist

container.tex 131 2016-06-02 14:06:24Z martin

Innehåll

1 Introduktion

Vi har tidigare använt oss av datastrukturen vector. En vector har som största förtjänst att man kan accessa elementen med hjälp av index utan kostnad. Vi har sett i föreläsningsexemplet Kjellqvist [2] att det är kostsamt att förändra storleken på en vector-liknande struktur. Det krävs omallokering + kopiering av samtliga element för att förändra storleken.

Ofta är det önskvärt att ha en struktur som kan växa och krympa utan extrra kostnad. Exemepel på sådana strukturer är stack och kö. Implementationen av sådana strukturer baserar sig ofta på en länkad lista. Länkade listor växer och krymper utan att påverka den övriga strukturen och är därför enkla att förändra storleken på.

Uppgiften är avsedd att ta två labbtillfällen i anspråk.

2 Syfte

Vi kommer i labben att i detalj undersöka hur länkade listor fungerar och hur de implementeras.

En beskrivning av länkade listor finner du både i kurslitteraturen och på Wikipedia [3].

Notera att länkade listor redan finns implementerade i standardbiblioteket i form av `std::list`. Vi skriver inte en länkad lista för att den behövs utan för att undersöka hur den är uppbyggd.

Som i labb 2 används operatorerna `new`, `delete` för att konstruera noderna i den länkade listan.

Innan du fortsätter med instruktionen bör du läsa igenom en beskrivning av länkade listor.

3 Terminologi

För att beskriva en länkad lista används flera olika termer.

nod En struct som innehåller datum och länkar. Hädanefter hävisad till som typen `node_t`.

head Det första elementet i den länkade listan. Deklareras som `node_t *`.

tail Det sista elementet i en länkad lista. Finns oftast endast i en dubbellänkad lista. Deklareras som `node_t *`.

länk En pekare, deklarerad som medlem i `node_t` som `next` och `prev`

enkellänkad lista En länkad lista vars noder endast innehåller en länk till nästa element, `next`, en fortsättning på head.

dubbellänkad lista En länkad lista vars noder innehåller länkarna `next` och `prev`. Möjliggör att man går igenom listan framlänges och baklänges.

4 Uppgift

Du skall konstruera en klass som beskriver en dubbellänkad lista. Separera header och implementation i .h och .cpp-fil.

Din header ska ha en klassdefinition med minst följande medlemmar.

```
class linked_list{
  public:
  linked_list();
  linked_list(const linked_list& src);

  ~linked_list();

  linked_list&  operator=(const linked_list& rhs);

  // adds elements to the back.
  linked_list& operator+=(const linked_list& rhs);

  // inserting elements
  void insert(double value, size_t pos);
  // in front
  void push_front(double value);
  // in back
```

```
  void push_back(double value);



  // accessing elements
  double front() const;
  double back() const;
  double at(size_t pos) const;

  // removing elements
  void remove(size_t pos);
  // remove and access
  double pop_front();
  double pop_back();

  // informational
  size_t size() const;
  bool is_empty() const;

  // output
  void print() const;
  void print_reverse() const;

  private:
  struct node_t{
    node_t(double value, node_t* next=0, node_t* prev=0);
    double value;
    node_t* next;
    node_t* prev;
  };
  node_t* head;
  node_t* tail;
};
```

Du bör titta på Kjellqvist [2] och följa den stilen i ditt program.

Konstrueraren använder sig av sk. "defaultargument". `=0` anges endast i headern, ej i implementationsfilen.

Borttagningsfunktionerna `pop*` tar bort elementet och returnerar värdet hos det borttagna elementet.

Structen `node_t` är en struct som är nästlad i klassen `linked_list`. Det gör att skrivsättet för definitionerna blir lite annorlunda.

```
linked_list::node_t::node_t(node_t* next, node_t* prev){
  // code...
}
```

Man kan också använda sig av en using-deklaration `using node_type = linked_list::node_t;` för att underlätta skrivsättet för implementationen.

Planera. Innan du skriver kod, rita med penna och papper hur länkarna måste följa varandra, vilka tilldelningar som måste ske då ett element stoppas in eller tas bort. Detta är helt nödvändigt för en lyckad implementation.

Olikt IntArrayexemplet. Det är inte meningsfullt att i konstrueraren ta ett antal element som listan kommer att innehålla.

Deep copy. Kopieringskonstrueraren ska göra en fullständig och oberoende kopia av originalet.

Kommentera Motivera knepiga pekaroperationer med dina kommentarer. Kod kan vara lättläst även om den innehåller mycket trix med pekare.

Tilldelningsoperatorn. Du kan för tilldelningsoperatorn antingen använda idiomet Copy&Swap eller författar du en egen direkt implementation. Du måste då undvika self-assignment.

Räds inte tillägg. Om du anser dig behöva fler / andra medlemmar skapar du dem efter eget huvud.

Skrid fram med små steg, testa ofta. Gör små tillägg till klassen, testkör ofta. Förslagsvis börjar du med node_t-structen och defaultkonstrueraren. Testa. Därefter implementerar du push_front. Testa. Implementera print(). Testa; och så vidare. Testen gör du genom att anropa funktionen och skriva ut resultat så du kan kontrollera riktigheten.

4.1 main.cpp

I huvudprogrammet ska du demonstrera funktionen hos samtliga medlemmar.

1. `push_back` Fyll två länkade listor med 100 stigande slumptal vardera. Du kan till exempel beräkna det nästkommande värdet med `ll.push_back( ll.back() + rand()% 20);`
2. `at` Kontrollera vilken lista var 50:e element är störst.
3. `remove` Ta bort det större av elementen.
4. `operator =` Deklarera en tredje lista. Tilldela därefter lista 1 till den tredje listan.
5. `pop_back, push_front` Ta bort varannat element genom att iterera

   ```
   l3.pop_back();
   l3.push_front( l3.pop_back());
   ```

   50 gånger.
6. `print` Skapa och anropa en global funktion

   ```
   void print_list(linked_list l);
   ```

   som skriver ut innehållet i den tredje listan.
7. Skapa en fjärde lista. Lista ut en metod som kombinerar lista 1 och 2 till den fjärde listan så att den fjärdes element ligger i ordning. Lista 1 och 2 behöver inte vara intakta efter operationen. Man behöver endast använda medlemmarna `front(), pop_front(), is_empty() och push_back()`

```
linked_list merge(linked_list list1, linked_list list2);
```

8. Skriv programkod som kontrollerar att listans element faktiskt ligger i ordning.

5 Examination

Din lösning ska redovisas muntligen för en lärare vid något av kursens redovisningstillfällen. När du redovisat och fått godkänt laddar du upp din källkod (och byggskript) till inlämningslådan i lärplattformen.

Referenser

[1] *C++ Reference*. URL `http://www.cplusplus.com/reference/`.

[2] Kjellqvist, Martin. Föreläsningsexempel, en dynamisk arrayimplementation. URL: `http://ver.miun.se/courses/dt019g/lect/IntArrays/`, 2013.

[3] Wikipedia. Wikipedia, länkade listor. URL: `http://en.wikipedia.org/wiki/Linked_list/`.