Implementasi List Menggunakan C++

Yohanes Nugroho

Overview

- Materi ini membahas mengenai implementasi List di C++
- Peserta kuliah dianggap sudah mengerti konsep List di C

Elemen List

- Di C, elemen list didefinisikan dengan struct, di C++ elemen list dapat didefinisikan sebagai struct atau class
- Penggunaan class lebih disarankan karena akan membungkus elemen list (enkapsulasi)

Deklarasi Elemen List

- Suatu kelas boleh merefer dirinya sendiri (tidak perlu ada typedef seperti di C)

```
class ListElement {
   int info;
   ListElement *next;
};
```

Deklarasi Elemen List (lengkap)

- Gunakan accessor set/get

```
#include <stdlib.h> //untuk deklarasi NULL
class ListElement {
    private: int info; ListElement *next;
    public:
    ListElement(){ this->info=0;this->next=NULL; }
    ListElement(int info, ListElement*
     next){ this->info=info;this->next=next; }
    void setInfo(int info){this->info = info; }
    int getInfo(){ return info;}
    void setNext(ListElement *next){ this->next = next; }
    ListElement *getNext(){ return next;}
};
```

Implementasi List

- List diimplementasikan sebagai kelas:
  - Property yang dimiliki adalah head dan atau tail

```
class SingleLinkedList {
   private: ListElement *head;
};
```

Konstruktor

- Konstruktor digunakan untuk membuat suatu list kosong
  - Biasanya mengisi head/tail dengan NULL
- Contoh:

```
public: SingleLinkedList(){
   head=NULL;
}
```

Method lain

- Algoritma untuk method lain sama dengan C
- Contoh:

```
public: void addFirst(int i){
   if (head==NULL){
      head = new ListElement();
      head->setInfo(i); head->setNext(NULL);
   } else {
      ListElement *e = new ListElement();
      e->setInfo(i); e->setNext(head);
      head = e;
   }
}
```

`addFirst()` menggunakan konstruktor

- AddFirst dapat diimplementasikan menggunakan konstruktor

```
public void addFirst(int i){
      head = new ListElement(i, head);
   }
```

Dealokasi

- Dealokasi elemen harus dilakukan dengan delete
- Contoh:

```
/*I.S: list pasti tidak kosong*/
/*F.S: elemen pertama terhapus*/
public: int delFirst(int i){
    int  v = head->getInfo();
    ListElement *tmp = head;
    head = head->getNext();
    delete tmp;
    return v;
}
```

Inner Class

- Kelas ListElement boleh ada di dalam kelas List, seperti ini:

```
class List {
   class ListElement {
      /*isi kelas list element*/
   }
  /*isi kelas List*/
}
```

Mengakses Inner Class

- Dengan sintaks sebelumnya, kelas ListElement tidak bisa diakses dari luar kelas List (karena private), agar bisa diakses, harus dibuat menjadi public

```
class List {
    public: class ListElement { /*isi kelas list
      element*/ }
    /*isi kelas List*/
}
```

- Dengan access modifier public, kelas ListElement memiliki bisa diakses dari luar kelas :

```
SingleLinkedList::ListElement = new
  SingleLinkedList::ListElement();
```

Kapan Memakai Inner class

- *Inner class* sebaiknya digunakan dibanding kelas terpisah jika hanya ada satu jenis list (*coupling* lebih terlihat)
- Jika ada banyak jenis list dengan satu jenis elemen (misal single linked list dengan head, dan single linked list dengan head dan tail) maka kelas ListElement boleh (dan sebaiknya) dipisah

List Rekursif

- List bisa diimplementasikan seperti pada bahasa fungsional
- List memiliki info (car) dan sisanya adalah list tail (cdr)

```
class List {
  /*m_ menandakan member variable*/
   private: int m_car; List* m_cdr;
   /*accessor*/
   public: int car() { return m_car;}
   List* cdr() { return m_cdr;}
};
```

Konstruktor List Rekursif

```
class List {
/*…*/
/*properti dan method lain */
/*…*/
  public: List(int car, List *cdr) {
  m_car = car;
  m_cdr = cdr;
 }
}
```

Definisi List Kosong

- List kosong adalah list yang menunjuk ke null (tidak ada operasi yang bisa dilakukan):

```
List *l = null; /*list kosong*/
l.print(); /*illegal*/
```

Algoritma Rekursif

- Menghitung jumlah elemen:

```
int countElement(){
   return 1 + ((cdr()==NULL)?0:
   cdr()->countElement());
}
```

- Print (traversal):

```
void print(){
   cout << car();
   if (cdr()!=NULL) cdr()->print();
}
```

Menghapus List

- List harus dihapus secara rekursif seperti ini:

```
public: ~List(){
   if (m_cdr!=NULL){ delete m_cdr;
     m_cdr=NULL;}
}
```

- Meskipun kode di atas tidak terlihat rekursif, pemanggilan delete akan secara implisit memanggil destruktur dari sisa List

Contoh Pemakaian List Rekursif

```
int main(int argc, char *argv[]){
   List *l = new List(123, new
  List(456, NULL));
 cout << "Count=“ <<
    l->countElement();
   l->print();
    return 0;
}
```

Catatan Penutup

- List rekursif harus diimplementasikan secara hati-hati karena C++ tidak memiliki managemen memori otomatis ("sampah memori" tidak otomatis dibuang)