STRUKTUR DATA

JULIO ADISANTOSO
Departemen Ilmu Komputer IPB

Pertemuan 3 : 27 Juni 2016

Standard Template Library

Mengapa STL?

- STL = ST L (L huruf ke-12). Jadi ST L = ST 12 ;)
- Mengapa menggunakan STL?
  - Reduce development time $\Leftarrow$ Data-structures already written and debugged.
  - Code readability $\Leftarrow$ Fit more meaningful stuff on one page.
  - Robustness $\Leftarrow$ STL data structures grow automatically.
  - Portable code.
  - Maintainable code
  - Easy

Implementasi SLL dengan STL

SLL lebih mudah diimplementasikan dengan STL forward_list

Contoh

```
#include <iostream>
#include <forward_list>
using namespace std;
int main () {
  forward_list<int> mylist;
  mylist.push_front (19); mylist.push_front (34);
  forward_list<int>::iterator it;
  for(it=mylist.begin();it!=mylist.end();++it)
    cout << *it << " -> ";
  cout << "NULL" << endl;
  return 0;
}
```

Implementasi SLL dengan STL

- Tentunya, STL forward_list belum memenuhi semua kebutuhan yang diinginkan, misalnya tidak ada fungsi print, push_back, push_after, dsb.
- Solusi: dapat dibuat class baru sebagai turunan dari container forward_list.

Contoh Class SLL dengan STL

Contoh

```
template<typename T>
class SLL : public forward_list<T> {
public:
  void print() {
    typename forward_list<T>::iterator it;
    for (it=forward_list<T>::begin();
      it!=forward_list<T>::end();++it)
        cout << *it << " -> ";
    cout << "NULL" << endl;
  }
};
```

SLL::push_back dengan STL

Karena tidak ada pointer tail, maka fungsi push_back harus dibuat khusus:

Fungsi push_back

```
void push_back(T val) {
  typename forward_list<T>::iterator it, before_end;
  if (forward_list<T>::empty())
    forward_list<T>::insert_after(
      forward_list<T>::before_begin(), val);
  else {
    for (it=forward_list<T>::begin();
      it!=forward_list<T>::end(); ++it)
        before_end=it;
    forward_list<T>::insert_after(before_end, val);
  }
}
```

Implementasi SLL dengan STL

- Ambil contoh kasus TUGAS #2 (SLL dengan info berupa pasangan nilai NIM dan IPK) $\Rightarrow$ membutuhkan **utility** *pair*

Spesifikasi TYPE (mengapa begini?)

```
#include <utility>
#include <iostream>
#include <forward_list>
#include <string>
using namespace std;
typedef pair<string, float> P;
```

Spesifikasi PRIMITIF

ADT SLL menggunakan STL

```
class SLL {
    SLLP dt;
    public:
      void push_front(string nim, float ipk);
      void push_back(string nim, float ipk);
      void push_after(string nim, float ipk,
         string after);
      void del(string nim);
      SLLP::iterator find(string nim);
      void print();
};
```

Fungsi print()

Menuliskan isi SLL ke standard output

Kode program

```
void SLL::print() {
    SLLP::iterator it;
    for (it=dt.begin(); it!=dt.end(); ++it) {
      cout << "(" << it->first
           << "," << it->second << ")->";
    }
    cout << "NULL" << endl;
}
```

Fungsi push_front(value)

Menyisipkan value di node paling depan

Kode program

```
void SLL::push_front(string nim, float ipk) {
    P t=make_pair(nim, ipk);
    dt.push_front(t);
}
```

Fungsi `make_pair` membuat dua nilai dengan tipe data tradisional (nim dan ipk) menjadi suatu nilai pasangan (*pair*).

Fungsi push_back(value)

Menyisipkan value di node paling belakang

Kode program

```
void SLL::push_back(string nim, float ipk) {
   P t=make_pair(nim, ipk);
   if (dt.empty()) dt.push_front(t);
   else {
     SLLP::iterator before=dt.begin();
     SLLP::iterator it=dt.begin();
     for (; it!=dt.end(); before=it, ++it);
     dt.insert_after(before,t);
   }
}
```

Fungsi lainnya harap dikembangkan sendiri.

Double Linked List

Double Linked List (DLL)

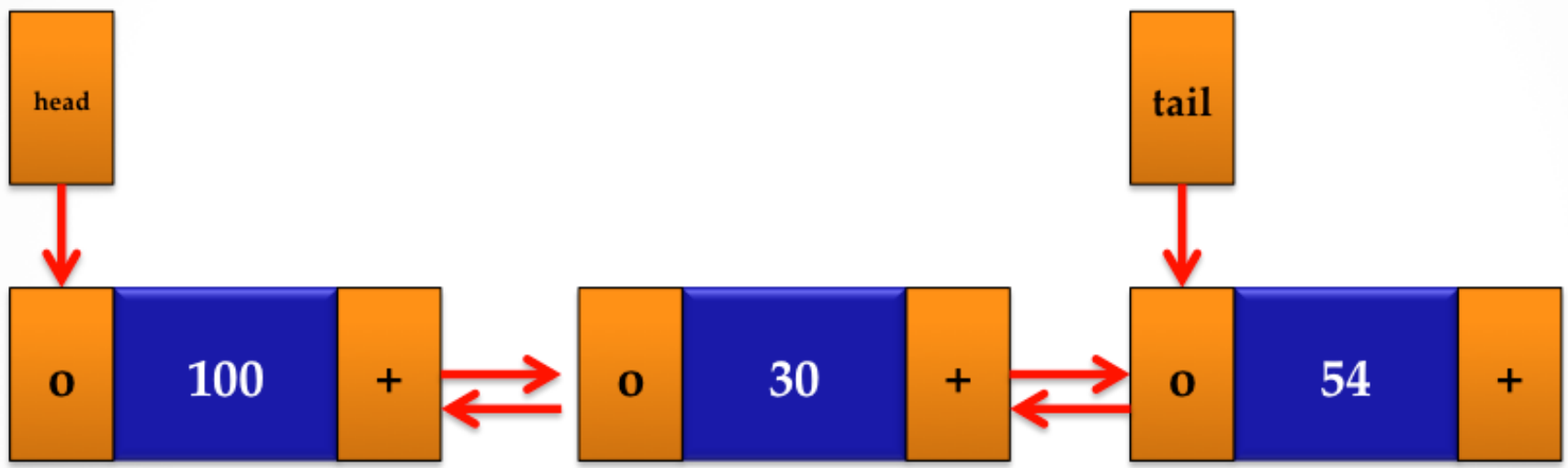
- Linked list dengan dua buah komponen pointer:
  - Ke node sebelumnya (**prev**)
  - Ke node berikutnya (**next**)
- Akses tetap sekuensial, sama dengan Single Linked List (SLL), hanya DLL dapat maju maupun mundur

SLL vs DLL

- Ukuran bytes DLL lebih besar dari SLL. Berapa?
- Penelusuran node pada DLL lebih fleksibel dibanding SLL. Mengapa?
- Penyisipan node baru ke dalam DLL dan penghapusan node dari DLL membutuhkan sumberdaya yang lebih rendah dibanding SLL. Mengapa?

Contoh DLL

- Contoh DLL:



- Bagaimana spefisikasi DLL ini?

Spesifikasi DLL

- Spesifikasi TYPE:

```
struct node {
    int info;
    struct node *prev, *next;
};
typedef struct node Node;
```

- Spesifikasi PRIMITIF: make, push (front, after, back), search, update (delete), isEmpty, etc.

Spesifikasi PRIMITIF DLL (P03b.cpp)

ADT DLL

```
class DLL {
   Node *head, *tail;
   public:
      void make() { head=NULL; tail=NULL;}
      Node* make(int val);
      int isEmpty() { return (head==NULL); }
      void push_back(int val);
      void push_front(int val);
      void push_after(int val, int after);
      Node *find(int val);
      void del(int val);
      void print();
};
```

Contoh DRIVER

Driver

```
int main() {
   DLL list; list.make();
   list.push_back(100); list.push_back(50);
   list.push_front(75); list.print();
   list.push_after(35,100); list.print();
   list.del(50); list.print();
   return 0;
}
```

Output

```
   75 -> 100 -> 50 -> NULL
   75 -> 100 -> 35 -> 50 -> NULL
   75 -> 100 -> 35 -> NULL
```

Fungsi make(value)

- Membuat sebuah node bernilai yang ditunjuk oleh pointer (ptr)
- Nilai yang dikembalikan adalah sebuah pointer ke node yang dibuat

Kode Program

```
Node* DLL::make(int val) {
    Node *temp = new(Node);
    temp->info=val;
    temp->next=NULL; temp->prev=NULL;
    return temp;
}
```

Fungsi push_back(value)

Menambahkan node bernilai ke bagian belakang DLL

Kode Program

```
void DLL::push_back(int val) {
   Node *ptr=make(val);
   if (isEmpty()) {
      head=ptr; ptr->prev=head; tail=ptr;
   } else {
      tail->next=ptr; ptr->prev=tail;
      tail=ptr;
   }
}
```

Fungsi push_front(value)

Menambahkan node bernilai ke bagian depan DLL

Kode Program

```
void DLL::push_front(int val) {
   Node *ptr=make(val);
   if (isEmpty()) {
      head=ptr; ptr->prev=head; tail=ptr;
   } else {
      ptr->next=head; ptr->prev=head;
      head=ptr;
   }
}
```

Fungsi print()

Menampilkan setiap nilai node

Kode Program

```
void DLL::print() {
   Node *ptr=head;
   for (; ptr!=NULL; ptr=ptr->next) {
      cout << ptr->info << " -> ";
   }
   cout << "NULL" << endl;
}
```

Fungsi find(value)

Mencari node yang memiliki nilai info=value. Output berupa pointer ke node yang ditemukan. NULL jika tidak ditemukan.

Kode program

```
Node* DLL::find(int val) {
   Node *ptr=head;
   if (isEmpty()) return NULL;
   else {
      while (ptr->next!=NULL && ptr->info!=val) {
         ptr=ptr->next;
      }
      if (ptr->info==val) return ptr;
      else return NULL;
   }
}
```

Fungsi push_after(value)

Menyisipkan node setelah suatu node yang bernilai after.

Kode program

```
void DLL::push_after(int val, int after) {
    Node *ptr = find(after);
    if (ptr!=NULL) {
        Node *temp = make(val);
        temp->next = ptr->next;
        (ptr->next)->prev=temp;
        ptr->next = temp;
        temp->prev=ptr;
    }
}
```

Fungsi del(value)

Menghapus node yang memiliki nilai info=value.

Kode program

```
void DLL::del(int val) {
   Node *ptr = find(val);
   if (ptr==head) {
      head=head->next; head->prev=head;
   } else if (ptr==tail) {
      tail=tail->prev; tail->next=NULL;
   } else if (ptr!=NULL) {
      (ptr->next)->prev=ptr->prev;
      (ptr->prev)->next=ptr->next;
   }
}
```

Implementasi DLL dengan STL (p03c.cpp)

- DLL dapat diimplementasikan dengan STL `list`
- STL `forward_list` vs `list`:

No	Operasi	`forward_list`	`list`
1	Sisip depan	push_front	push_front
	Sisip belakang	n/a	push_back
	Sisip tengah	insert_after	insert
2	Hapus depan	pop_front	pop_front
	Hapus belakang	n/a	pop_back
	Hapus setelah	erase_after	erase
3	Pencarian	n/a	n/a
4	Update	n/a	n/a

- `forward_list` $\Leftrightarrow$ SLL; `list` $\Leftrightarrow$ DLL

Spesifikasi DLL menggunakan STL

```
#include <iostream>
#include <list>
using namespace std;
typedef list<int> LI;
class DLL {
   LI dt;
   public:
     int isEmpty() { return dt.empty(); }
     void push_back(int val) { dt.push_back(val); };
     void push_front(int val) { dt.push_front(val); };
     void push_after(int val, int after);
     LI::iterator find(int val);
     void del(int val);
     void print();
};
```

Fungsi print() dan find(value)

Fungsi print()

```
void DLL::print() {
   LI::iterator it;
   for (it=dt.begin(); it!=dt.end(); ++it)
      cout << (*it) << "->";
   cout << "NULL" << endl; }
```

Fungsi find(value)

```
LI::iterator DLL::find(int val) {
   LI::iterator it;
   for (it=dt.begin(); it!=dt.end(); ++it)
      if ((*it)==val) return it;
   return it; }
```

Fungsi push_after(value) dan del(value)

Fungsi push_after(value)

```
void DLL::push_after(int val, int after) {
   LI::iterator it=find(after);
   if (it!=dt.end()) {
      ++it;
      dt.insert(it, val);
   }
}
```

Fungsi del(value)

```
void DLL::del(int val) {
   LI::iterator it=find(val);
   if (it!=dt.end()) dt.erase(it);
}
```

TUGAS #3

Diketahui n ayam berdiri dalam satu baris dari Utara ke Selatan. Setiap ayam dicatat tinggi badannya. Selanjutnya datang m ayam baru yang berjalan beriringan dari Utara ke Selatan yang ingin ikut dalam barisan. Ternyata ada aturan, bahwa setiap ayam yang baru datang akan menempati sebelum ayam dalam barisan yang pertama kali lebih tinggi darinya. Tuliskan susunan akhir dari barisan ayam tersebut.

Contoh input

```
5 10 40 15 20 35   (n=5)
3 30 50 7          (m=3)
```

Contoh output

```
7->10->30->40->15->20->35->50->NULL
```