STRUKTUR DATA

JULIO ADISANTOSO
Departemen Ilmu Komputer IPB

Pertemuan 2 : 21 Juni 2016

Array

Array

- Struktur data linier yang dapat menyimpan lebih dari satu buah nilai, umumnya bertipe data sama
- Datanya tersimpan secara terurut di memory
- Memiliki indeks (*random access*)
- Abstraksi (contoh):

$$X = \{X_0, X_1, \ldots, X_{(n-1)}\}$$

$$Y = \begin{pmatrix} Y_{00} & Y_{01} & \cdots & Y_{0(n-1)} \\ Y_{10} & Y_{11} & \cdots & Y_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{(m-1)0} & Y_{(m-1)1} & \cdots & Y_{(m-1)(n-1)} \end{pmatrix}$$

Multidimensional Array

- Array dengan dimensi lebih dari 1, tetap disimpan dalam memory dengan 1 dimensi
- Contoh:

$$Y = \begin{pmatrix} 4 & 2 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{pmatrix}$$

dapat ditulis sebagai $Y = \{\{4, 2, 1, 0\}, \{0, 0, 3, 1\}\}$

- Jadi, nilai $Y_{00} = -1$ disimpan dalam memory pada indeks ke-0, dan $Y_{12} = 3$ pada indeks ke-6.

Multidimensional Array

```
#define M 2
#define N 4
int main () {
   int Y[M][N]={{4,2,1,0},{0,0,3,1}};
   for (int i=0; i<M; i++)
      for (int j=0; j<N; j++) {
         cout << Y[i][j];
         if (j==N-1) cout << endl;
      }
   return 0;
}
```

```
#define M 2
#define N 4
int main () {
   int Y[M][N]={{4,2,1,0},{0,0,3,1}};
   int *p = &Y[0][0];
   for (int i=0; i<M; i++)
      for (int j=0; j<N; j++) {
         cout << *p;
         if (j==N-1) cout << endl;
         else cout << " ";
         p++;
      }
   return 0;
}
```

STL Array

Implementasi array dalam C++ dapat menggunakan STL Array

language built-in array	container library array

`#include <iostream>`
 	
`using namespace std;`
 	
`int main()`
`{`
`  int myarray[3] = {10,20,30};`
 	
`  for (int i=0; i<3; ++i)`
`  ++myarray[i];`
 	
`  for (int elem : myarray)`
`  cout << elem << '\n';`
`}`
	`#include <iostream>`
	`#include <array>`
	`using namespace std;`
	
	`int main()`
	`{`
	`  array<int,3> myarray {10,20,30};`
	
	`  for (int i=0; i<myarray.size(); ++i)`
	`  ++myarray[i];`
	
	`  for (int elem : myarray)`
	`  cout << elem << '\n';`
	`}`

TUGAS #1

Buat ADT MATRIK yang memiliki kemampuan sama seperti matrik dua dimensi. Memiliki fungsi transpose, penjumlahan, pengurangan, dan perkalian matrik. Ukuran maksimum matrik adalah 100×100. Dapat dijalankan dengan *driver* seperti contoh berikut:

```
int main() {
    array<array<int, 100>, 100> arr={{{5,8,2},{8,3,1}}};
    int brr[M][N]={{1,2,1},{0,1,1}};
    MATRIK A,B,C,D,BT;
    A.make(arr,2,3); B.make(brr,2,3);
    BT=B.transpose();
    C=A+B; C.print(); // menjumlah 2 matrik
    D=A*BT; D.print(); // mengalikan matrik
    return 0;
}
```

TUGAS #1

PRIMITIF dari ADT MATRIK kira-kira seperti berikut:

```
#define M 100
#define N 100
typedef array<array<int, N>, M> AR;
class MATRIK {
   AR dt;              // array 2D
   int nRow, nCol;     // ukuran matrik
   public:
   void make(...);     // membuat matrik
   void print(...);    // mencetak matrik
   ...;                // fungsi pendukung lainnya
   MATRIK transpose();  // memutar matrik
   friend MATRIK operator+(MATRIK a, MATRIK b);
   friend MATRIK operator-(MATRIK a, MATRIK b);
   friend MATRIK operator*(MATRIK a, MATRIK b);
}
```

Vector

Vector

- Array memiliki kelemahan, antara lain, kurang fleksibel pada masalah ukuran array (harus didefinisikan ukuran maksimum array)
- Vector merupakan implementasi dari array, tetapi lebih fleksibel, tidak memiliki batasan ukuran array
- Dalam C++, STL Vector dapat saling dipertukarkan dengan array, tidak demikian halnya dengan Java (ada fungsi konversi **toArray**()).
- Tugas: pelajari kembali materi Vector dalam Bahasa Pemrograman sebelumnya.

Struct

Struct

- Merupakan suatu struktur data dari serangkaian nilai yang dimungkinkan berbeda tipe datanya. Ingat, nilai dalam array umumnya memiliki tipe data yang sama.
- Struct dalam C++:

```
struct type_name {
    member_type1 member_name1;
    member_type2 member_name2;
    .
    .
};
typedef type_name alias_name;
```

Linked List

Linked List (LL)

- Daftar dari elemen yang saling berkait
- Elemennya sering disebut **node**
- Node-node pembentuk LL tidak disimpan secara terurut -¿ tidak memiliki indeks
- Akses node secara sekuensial
- Node tersusun atas dua komponen data: informasi dan pointer
- Jenis Linked List:
  - Single Linked List (SLL),
  - Double Linked List (DLL).

Node

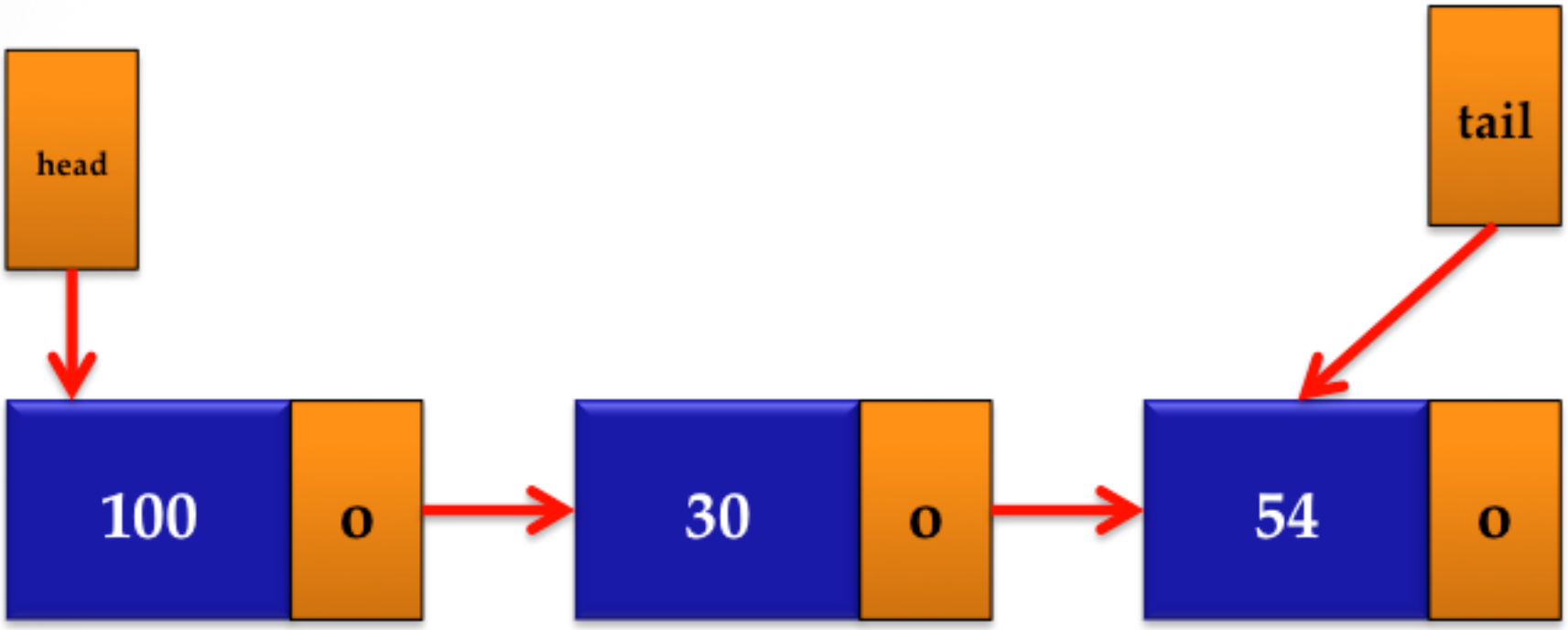
- Abstraksi logik SLL

- Spesifikasi TYPE dari SLL:

```
struct nama_node {
   komponen_informasi;
   komponen_pointer;
};
```

SLL

- Contoh SLL:



- Mengapa ada pointer **head** dan **tail**?

Spesifikasi SLL

- Spesifikasi TYPE:

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

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

Spesifikasi PRIMITIF SLL (P02b.cpp)

```
class SLL {
   Node *head, *tail;
   public:
     void make() { head=NULL; tail=NULL;}
     Node* make(int val);  // make a node
     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);
     Node *find_before(int val);
     void del(int val);
     void print();
};
```

Contoh DRIVER

- Driver:

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

Fungsi push_back(value)

- Menambahkan node bernilai ke bagian belakang SLL
- Kode program:

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

Fungsi push_front(value)

- Menambahkan node bernilai ke bagian depan SLL
- Kode program:

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

Fungsi print()

- Menampilkan setiap nilai node
- Kode program:

```
void SLL::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 pertama kali, atau NULL jika tidak ditemukan
- Kode program:

```
Node* SLL::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 bernilai value setelah suatu node yang bernilai after.
- Kode program:

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

Fungsi find_before (value)

- Mencari node sebelum node yang memiliki nilai info=value. Bermanfaat untuk fungsi **del**.
- Output berupa pointer ke node yang ditemukan
- Kode program:

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

Fungsi del(value)

- Menghapus node yang memiliki nilai info=value.
- Kode program:

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

TUGAS #2

Buat ADT SLL2 yang memiliki kemampuan sama seperti SLL pada contoh, tetapi informasi yang ditangani adalah nim dan ipk (*float*). Dapat dijalankan dengan *driver* seperti contoh berikut:

```
int main() {
    SLL2 list;
    list.make();
    list.push_back("G64204100", 3.14);
    list.push_back("G64204050", 3.67);
    list.push_front("G6420075", 2.05);
    list.print();
    list.push_after("G6420035", 2.89, "G64204100");
    list.print(); list.del("G64204100");
    list.print();
    return 0;
}
```

TUGAS #2

- Output program kira-kira sebagai berikut:

```
(G6420075,2.05)->(G64204100,3.14)->(G64204050,3.67)->NULL
(G6420075,2.05)->(G64204100,3.14)->(G6420035,2.89)->(G64204050,3.67)->NULL
(G6420075,2.05)->(G6420035,2.89)->(G64204050,3.67)->NULL
```

- Tugas 1 dan 2 dikumpulkan melalui LX sesuai batas waktu yang ada, dan akan di-*grader* secara manual.