

Outline

- 1 Tujuan
- 2 Konsep Dasar
- 3 Pengalokasian Memori Secara Dinamik
- 4 Linked-List menggunakan struct
- 5 Doubly Linked-List

Tujuan

- Setelah menyelesaikan modul ini mahasiswa diharapkan:
 - Memahami konsep pengalokasian memori secara dinamik
 - Dapat mengalokasikan array secara dinamik
 - Memahami konsep struktur data struct
 - Dapat mengimplementasikan konsep *linked-list*
 - Dapat mengimplementasikan konsep *doubly linked-list*



Konsep Dasar

- Mengalokasikan memori sebanyak yang diperlukan.
- Mengurangi jumlah perubahan yang harus dibuat terhadap program apabila kebutuhan program berubah.



Pengalokasian Memori Secara Statik

- Sebelum memperkenalkan konsep alokasi memori dinamik, diberikan terlebih dahulu contoh alokasi memory statik.
- Program berikut mengalokasikan memori sebanyak 20 bytes ke array integer kemudian mengisi elemen-elemen array dengan bilangan acak.
- Bilangan acak yang dihasilkan adalah berdasarkan benih (*seed*) yang diberikan berdasarkan fungsi waktu *time()*.
- Fungsi *bubble_sort()* kemudian akan mensortir elemen-elemen array tersebut secara menaik (*ascending*).
- Contoh Program 8.1:



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

Pengalokasian Memori Secara Statik

```
47  int main()
48  {
49      Array myArr;
50
51      myArr.printArray("Sebelum_disortir:_");
52      myArr.bubble_sort();
53      myArr.printArray("Setelah_disortir:_");
54
55      return 0;
56  }
```



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Pengalokasian Memori Secara Dinamik

```
1 //Contoh8_2.cpp
2 #include <iostream>
3 #include <string>
4 #include <cstdlib>
5 #include <ctime>
6
7 using namespace std;
8
9 class Array
10 {
11     private:
12         int *array;
13         int size;
```





```
28 void Array::generate()
29 {
30     if ((array = new int[size]) == NULL)
31         cerr << "Error_alokasi_memori..." << endl;
32     else
33     {
34         for(int i=0; i < size; i++)
35             array[i] = rand() % size;
36     }
37 }
```



```
49 void Array::printArray(string str)
50 {
51     cout << str << endl;
52     for(int i=0; i < size; i++)
53         cout << array[i] << "_";
54     cout << endl;
55 }
```


Pengalokasian Memori Secara Dinamik

```
56 int main()
57 {
58     Array myArr(100);
59
60     //Generate random array
61     myArr.generate();
62
63     //Print unsorted array
64     myArr.printArray("Sebelum_disortir:");
65     //Sort array
66     myArr.bubble_sort();
67     //Print sorted array
68     myArr.printArray("Sesudah_disortir:");
69
70     return 0;
71 }
```

Transformation
Towards the Ultimate

Linked-List menggunakan struct

- Linked-list menggunakan struktur data struct dalam membentuk simpul-simpul (*nodes*) list.
- Bentuk umum struktur data struct:

```
struct nama_struktur
{
    jenis_data elemen_data1;
    jenis_data elemen_data2;
    ...
    jenis_data elemen_dataN;
};
```


Linked-List menggunakan struct

- Linked list dapat dibuat pada *stack*.
- Ketika objek itu meninggalkan *scope* fungsi *main()*, secara otomatis akan di keluarkan dari stack (*popped off*), dan destructor akan dipanggil.
- Contoh:
 LinkedList myList;
 myList.clear();
- Contoh Program 8.3:



A set of navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

Linked-List menggunakan struct

```
15 class LinkedList
16 {
17     private:
18         Node *first;
19     public:
20         LinkedList();
21         void show_list();
22         void append_value(int);
23         void clear();
24         ~LinkedList() {}
25 };
```



Linked-List menggunakan struct

```
26 LinkedList::LinkedList()
27 {
28     srand(time(NULL));
29     first = NULL;
30 }
31
32 void LinkedList::show_list()
33 {
34     Node *node=first;
35     while(node)
36     {
37         cout << node->value << "_";
38         node = node->next;
39     }
40     cout << endl;
41 }
```



```

42 void LinkedList::append_value(int value)
43 {
44     Node *tmp, *tail;
45     tmp = new Node;
46     tmp->value = value;
47     tmp->next = NULL;
48     if (first == NULL)
49         first = tmp;
50     else
51     {
52         tail = first;
53         while (tail->next)
54             tail = tail->next;
55         tail->next = tmp;
56     }
57 }

```

Transformation
Towards the Ultimate

11

```
58 void LinkedList::clear() {
59     Node* temp;
60
61     cout << "deleting_..._";
62
63     while (first != NULL) {
64         cout << first->value << "_";
65         temp = first->next;
66         delete first;
67         first = temp;
68     }
69     cout << endl;
70     first = NULL;
71 }
```

```
72 int main()
73 {
74     LinkedList myList;
75
76     cout << "Linked_list:" << endl;
77     for(int i=0; i < N; i++)
78         myList.append_value(rand() % N);
79     myList.show_list();
80
81     myList.clear();
82
83     return 0;
84 }
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

Linked-List menggunakan struct

```

1 //Contoh8_4.cpp
2 #include <iostream>
3 #include <iomanip>
4 #include <cstdlib>
5 #include <ctime>
6 #define N 100
7
8 using namespace std;
9
10 struct Node
11 {
12     int value;
13     Node *next;
14 };
  
```



Linked-List menggunakan struct

```
15 class LinkedList
16 {
17     private:
18         Node *first;
19     public:
20         LinkedList();
21         void show_list();
22         void append_value(int);
23         void clear();
24         ~LinkedList() {}
25 };
```



Linked-List menggunakan struct

```
29 void Array::bubble_sort()
30 {
31     for(int i=0; i < N; i++)
32         for(int j=0; j < N; j++)
33             if(array[i] < array[j])
34             {
35                 int temp = array[i];
36                 array[i] = array[j];
37                 array[j] = temp;
38             }
39 }
```



Linked-List menggunakan struct

```

42 void LinkedList::append_value(int value)
43 {
44     Node *tmp, *tail;
45     tmp = new Node;
46     tmp->value = value;
47     tmp->next = NULL;
48     if (first == NULL)
49         first = tmp;
50     else
51     {
52         tail = first;
53         while(tail->next)
54             tail = tail->next;
55         tail->next = tmp;
56     }
57 }
  
```



Linked-List menggunakan struct

```
58 void LinkedList::clear() {
59     Node* temp;
60
61     cout << "deleting_..._";
62
63     while (first != NULL) {
64         cout << first->value << "_";
65         temp = first->next;
66         delete first;
67         first = temp;
68     }
69     cout << endl;
70     first = NULL;
71 }
```



Linked-List menggunakan struct

```
72 int main()
73 {
74     LinkList *myList = new LinkList();
75
76     cout << "Linked_list:" << endl;
77     for(int i=0; i < N; i++)
78         myList->append_value(rand() % N);
79     myList->show_list();
80
81     myList->clear();
82     delete myList;
83
84     return 0;
85 }
```



Linked-List menggunakan struct

• Contoh Program 8.5: reverse list

```
1 //Contoh8_5.cpp
2 #include <iostream>
3 #include <iomanip>
4 #include <cstdlib>
5 #include <ctime>
6 #define N 100
7
8 using namespace std;
9
10 struct Node
11 {
12     int value;
13     Node *next;
14 };
```



Linked-List menggunakan struct

```
15 class LinkedList
16 {
17     private:
18         Node *first;
19     public:
20         LinkedList();
21         void show_list();
22         void append_value(int);
23         void reverseList();
24         ~LinkedList() {}
25 };
```




```
26 LinkList::LinkList()
27 {
28     srand(time(NULL));
29     first = NULL;
30 }
31
32 void LinkList::show_list()
33 {
34     Node *node = first;
35     while(node)
36     {
37         cout << node->value << "_";
38         node = node->next;
39     }
40     cout << endl;
41 }
```

Linked-List menggunakan struct

```

42 void LinkedList::append_value(int value)
43 {
44     Node *tmp, *tail;
45
46     tmp = new Node;
47     tmp->value = value;
48     tmp->next = NULL;
49     if (first == NULL)
50         first = tmp;
51     else
52     {
53         tail = first;
54         while(tail->next)
55             tail = tail->next;
56         tail->next = tmp;
57     }
58 }

```



Transformation
Towards the Ultimate

Navigation icons: back, forward, search, etc.

Linked-List menggunakan struct

```
59 void LinkedList::reverseList()
60 {
61     Node* temp = first;
62     Node* nextnode = NULL;
63     Node* revnode = NULL;
64
65     while (temp != NULL)
66     {
67         first = temp;
68         nextnode = temp->next;
69         temp->next = revnode;
70         revnode = temp;
71         temp = nextnode;
72     }
73 }
```




Linked-List menggunakan struct

```

74  int main()
75  {
76      LinkedList myList;
77
78      cout << "Linked_list_awal:" << endl;
79      for(int i=0; i < N; i++)
80          myList.append_value(rand() % N);
81      myList.show_list();
82
83      myList.reverseList();
84      cout << "Linked_list_setelah_dibalik:"
85          << endl;
86      myList.show_list();
87
88      return 0;
89  }

```



Transformation
Towards the Ultimate

Linked-List menggunakan struct

- Penjelasan algoritma reverse pada program 8.5:

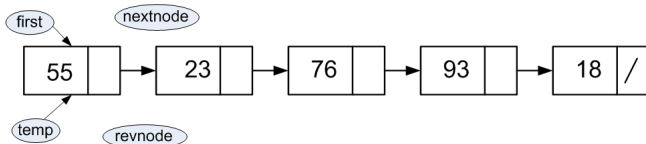
```

59 void LinkedList::reverseList()
60 {
61     Node* temp = first;
62     Node* nextnode = NULL;
63     Node* revnode = NULL;
64
65     while (temp != NULL)
66     {
67         first = temp;
68         nextnode = temp->next;
69         temp->next = revnode;
70         revnode = temp;
71         temp = nextnode;
72     }
73 }
  
```



Linked-List menggunakan struct

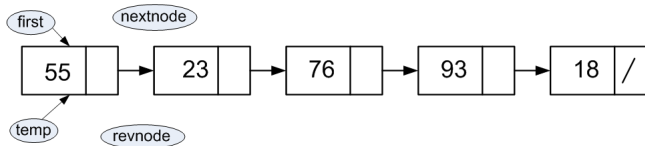
- Keadaan awal sebelum loop:



Line 61-63
temp = first;
nextnode = NULL;
revnode = NULL;

Linked-List menggunakan struct

• Iterasi 1:



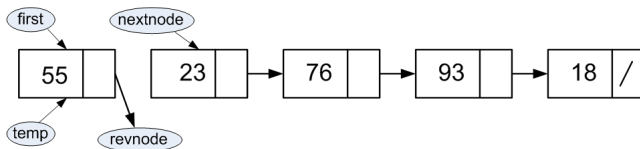
Line 67

first = temp;

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

Linked-List menggunakan struct

- Iterasi 1:



Line 69
temp->next = revnode;

- Iterasi 1:



- Iterasi 1:



Linked-List menggunakan struct

● Contoh Program 8.6:

```
1 // Contoh8_6.cpp
2 // Program untuk menyimpan titik koordinat
  dalam linked list
3 #include <iostream>
4
5 using namespace std;
6
7 struct Simpul
8 {
9     int absis;
10    int ordinat;
11    Simpul *berikut;
12 };
```



Linked-List menggunakan struct



```
13 class titikKoord
14 {
15     private:
16         Simpul *first;
17     public:
18         titikKoord();
19         void tambahTitik();
20         void buangTitik();
21         void cetakDaftar();
22 };
23
24 titikKoord::titikKoord()
25 {
26     first = NULL;
27 }
```















◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

```
56     else
57     {
58         ekor = first;
59         while (ekor->berikut)
60             ekor = ekor->berikut;
61         ekor->berikut = baru;
62     }
63 }
64
65 void titikKoord::buangTitik()
66 {
67     int x, y, ada = 0;
68     Simpul *cari, *hapus;
69
70     cout << "Titik, mana?" << endl;
```

```
72     cin >> x >> y;
```

73

```
74     cari = first;
```

```
75 while (cari && !ada)
```

76 {

```
77     if ((cari->absis == x) &&
```

```
78      (cari->ordinat == y))
```

```
79      ada = 1;
```

```
80         else
```

```
81      cari = cari->berikut;
```

82 }

```
83     if (ada)
```

84 {

```
85     hapus = first;
```



```

86     if(hapus == cari)
87     {
88         first = first->berikut;
89         cari->berikut = NULL;
90         delete cari;
91     }
92     else
93     {
94         while ((hapus->berikut) &&
95                (hapus->berikut != cari))
96             hapus = hapus->berikut;
97         hapus->berikut = cari->berikut;
98         cari->berikut = NULL;
99         delete cari;
100    }
101 }

```

Transformation
Towards the Ultimate

Kampus Merdeka

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Linked-List menggunakan struct

```
117 cout << "3. Mencetak daftar." << endl;
118 cout << "4. Selesai." << endl;
119 cout << "Pilihan: ";
120 cin >> pilih;
121 switch(pilih)
122 {
123     case '1': titikA.tambahTitik();
124         break;
125     case '2': titikA.buangTitik();
126         break;
127     case '3': titikA.cetakDaftar();
128         system("pause");
129         break;
130 }
131 } while (pilih < '4');
132 return 0;
```



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ≡ ≡ ↺ 🔍 ↻

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.



```

27 DoubleLinkedList::DoubleLinkedList()
28 {
29     srand(time(NULL));
30     first = NULL;
31     end = NULL;
32 }
33
34 void DoubleLinkedList::show_list()
35 {
36     Node *node;
37
38     node = first;
39     while(node)
40     {
41         cout << node->value << " ";
42         node = node->next;
43     }

```

Doubly Linked-List

```
46 void DoubleLinkedList::show_prevlist ()
47 {
48     Node *node;
49
50     node = end;
51     while (node)
52     {
53         cout << node->value << "_";
54         node = node->prev;
55     }
56     cout << endl;
57 }
```



Doubly Linked-List

```

58 Node *DoublyLinkedList::append_value(int value)
59 {
60     Node *ptr = end;
61     end = new Node;
62     if (first == NULL)
63         first = end;
64     else
65         ptr->next = end;
66     if(end)
67     {
68         end->next = NULL;
69         end->prev = ptr;
70         end->value = value;
71     }
72     return end;
73 }
  
```


Transformation
Towards the Ultimate