# DAFTAR ISI

[DAFTAR ISI 1](#_Toc441134375)

[DAFTAR GAMBAR 5](#_Toc441134376)

[DAFTAR PROGRAM 7](#_Toc441134377)

[DAFTAR TABEL 8](#_Toc441134378)

[Modul 0 : Running Modul 9](#_Toc441134379)

[0.1 Tujuan 9](#_Toc441134380)

[0.2 Peraturan Praktikum 9](#_Toc441134381)

[0.3 Penilaian Praktikum 10](#_Toc441134382)

[Modul 1 : ADT 11](#_Toc441134383)

[1.1 Tujuan 11](#_Toc441134384)

[1.2 Fungsi 11](#_Toc441134385)

[1.2.1 Paramater Formal dan Parameter Aktual 12](#_Toc441134386)

[1.2.2 Cara melewatkan Parameter 12](#_Toc441134387)

[1.3 Prosedur 14](#_Toc441134388)

[1.4 Rekursif 14](#_Toc441134389)

[1.5 Array 15](#_Toc441134390)

[1.5.1 Array Satu Dimensi 15](#_Toc441134391)

[1.5.2 Array Dua Dimensi 15](#_Toc441134392)

[1.5.3 Array Berdimensi Banyak 16](#_Toc441134393)

[1.6 String 16](#_Toc441134394)

[1.7 Abstract Data Type (ADT) 17](#_Toc441134395)

[1.8 Latihan 19](#_Toc441134396)

[Modul 2 : SINGLE LINKED LIST 21](#_Toc441134397)

[2.1 Tujuan 21](#_Toc441134398)

[2.2 Pointer 21](#_Toc441134399)

[2.2.1 Pointer dan Alamat 21](#_Toc441134400)

[2.3 Linked List dengan Pointer 22](#_Toc441134401)

[2.4 Single Linked List 23](#_Toc441134402)

[2.4.1 Pembentukan Komponen-Komponen List 24](#_Toc441134403)

[2.4.2 Insert 25](#_Toc441134404)

[2.4.3 Delete 27](#_Toc441134405)

[2.4.4 Update 29](#_Toc441134406)

[2.4.5 View 29](#_Toc441134407)

[2.4.6 Searching 29](#_Toc441134408)

[2.5 Latihan 36](#_Toc441134409)

[Modul 3 : DOUBLE LINKED LIST 38](#_Toc441134410)

[3.1 Tujuan 38](#_Toc441134411)

[3.2 Double Linked List 38](#_Toc441134412)

[3.2.1 Insert 39](#_Toc441134413)

[3.2.2 Delete 42](#_Toc441134414)

[3.3 Latihan 46](#_Toc441134415)

[Modul 4 : MULTI LINKED LIST 48](#_Toc441134416)

[4.1 Tujuan 48](#_Toc441134417)

[4.2 Multi Linked List 48](#_Toc441134418)

[4.2.1 Insert 49](#_Toc441134419)

[4.2.2 Delete 50](#_Toc441134420)

[4.3 Latihan 55](#_Toc441134421)

[Modul 5 : STACK (TUMPUKAN) 56](#_Toc441134422)

[5.1 Tujuan 56](#_Toc441134423)

[5.2 Pengertian 56](#_Toc441134424)

[5.3 Komponen-Komponen dalam Stack 56](#_Toc441134425)

[5.4 Operasi-Operasi dalam Stack 57](#_Toc441134426)

[5.4.1 Push 57](#_Toc441134427)

[5.4.2 Pop 57](#_Toc441134428)

[5.5 Primitif-Primitif dalam Stack 58](#_Toc441134429)

[5.6 Stack (Representasi Tabel) 58](#_Toc441134430)

[5.6.1 Operasi-operasi Dalam Stack 59](#_Toc441134431)

[5.6.1.1 Push 59](#_Toc441134432)

[5.6.1.2 Pop 59](#_Toc441134433)

[5.6.2 Primitif-primitif Dalam Stack 59](#_Toc441134434)

[5.7 Latihan 61](#_Toc441134435)

[Modul 6 : QUEUE (Representasi Pointer) 64](#_Toc441134436)

[6.1 Tujuan 64](#_Toc441134437)

[6.2 Pengertian 64](#_Toc441134438)

[6.3 Operasi-Operasi dalam Queue 65](#_Toc441134439)

[6.3.1 Insert (Enqueue) 65](#_Toc441134440)

[6.3.2 Delete (Dequeue) 66](#_Toc441134441)

[6.4 Primitif-Primitif dalam Queue 66](#_Toc441134442)

[6.5 Priority Queue 68](#_Toc441134443)

[6.5.1 Pengertian 68](#_Toc441134444)

[6.5.2 Bentuk Priority Queue 68](#_Toc441134445)

[6.5.2.1 Menggunakan Satu Queue 68](#_Toc441134446)

[6.5.2.1.1 EnQueue 68](#_Toc441134447)

[6.5.2.1.2 DeQueue 69](#_Toc441134448)

[6.5.2.2 Menggunakan beberapa Queue 69](#_Toc441134449)

[6.5.2.2.1 EnQueue 69](#_Toc441134450)

[6.5.2.2.2 DeQueue 70](#_Toc441134451)

[6.5.3 Primitif-primitif dalam Priority Queue 70](#_Toc441134452)

[6.6 Latihan 71](#_Toc441134453)

[Modul 7 : QUEUE (Representasi Tabel) 73](#_Toc441134454)

[7.1 Tujuan 73](#_Toc441134455)

[7.2 Pengertian 73](#_Toc441134456)

[7.3 Macam-macam Bentuk Queue 73](#_Toc441134457)

[7.3.1 Alternatif 1 73](#_Toc441134458)

[7.3.2 Alternatif 2 73](#_Toc441134459)

[7.3.3 Alternatif 3 74](#_Toc441134460)

[7.4 Primitif-Primitif dalam Queue 75](#_Toc441134461)

[7.5 Latihan 76](#_Toc441134462)

[Modul 8 : TREE 77](#_Toc441134463)

[8.1 Tujuan 77](#_Toc441134464)

[8.2 Binary Search Tree 77](#_Toc441134465)

[8.3 Operasi-Operasi dalam Binary Search Tree 77](#_Toc441134466)

[8.4 Traversal pada Binary Tree 79](#_Toc441134467)

[8.5 Latihan 81](#_Toc441134468)

[Modul 9 : GRAPH TAK BERARAH 82](#_Toc441134469)

[9.1 Tujuan 82](#_Toc441134470)

[9.2 Pengertian Graph 82](#_Toc441134471)

[9.3 Pengertian Graph tak Berarah 82](#_Toc441134472)

[9.4 Representasi Graph 82](#_Toc441134473)

[9.4.1 Array 2 Dimensi 83](#_Toc441134474)

[9.4.2 Multi Linked List 83](#_Toc441134475)

[9.5 Metode-Metode Penulusuran Graph 84](#_Toc441134476)

[9.5.1 Breadth First Search (BFS) 84](#_Toc441134477)

[9.5.2 Depth First Search (DFS) 85](#_Toc441134478)

[9.6 Latihan 85](#_Toc441134479)

[Modul 10 : GRAPH BERARAH 87](#_Toc441134480)

[10.1 Tujuan 87](#_Toc441134481)

[10.2 Pengertian 87](#_Toc441134482)

[10.3 Representasi Graph 87](#_Toc441134483)

[10.3.1 Multi Linked List 87](#_Toc441134484)

[10.4 Topological Sort 88](#_Toc441134485)

[10.4.1 Pengertian 88](#_Toc441134486)

[10.4.2 Representasi Topological Sort 90](#_Toc441134487)

[10.5 Latihan 90](#_Toc441134488)

[Modul 11 : Responsi Tugas Besar 1 92](#_Toc441134489)

[Modul 12 : Responsi Tugas Besar 2 93](#_Toc441134490)

[Modul 13 : Presentasi Tugas Besar 94](#_Toc441134491)

[DAFTAR PUSTAKA 95](#_Toc441134492)

# DAFTAR GAMBAR

[Gambar 2‑1 Output Pointer 18](#_Toc408777776)

[Gambar 2‑2 Elemen Single Linked List 19](#_Toc408777777)

[Gambar 2‑3 Single Linked List dengan Elemen Kosong 19](#_Toc408777778)

[Gambar 2‑4 Single Linked List dengan 3 Elemen 19](#_Toc408777779)

[Gambar 2‑5 Circular Single Linked List 20](#_Toc408777780)

[Gambar 2‑7Single Linked List Insert First 2 21](#_Toc408777781)

[Gambar 2‑8Single Linked List Insert First 3 21](#_Toc408777782)

[Gambar 2‑9Single Linked List Insert Last 1 22](#_Toc408777783)

[Gambar 2‑10Single Linked List Insert Last 2 22](#_Toc408777784)

[Gambar 2‑11Single Linked List Insert Last 3 22](#_Toc408777785)

[Gambar 2‑12Single Linked List Insert After 1 23](#_Toc408777786)

[Gambar 2‑13Single Linked List Insert After 2 23](#_Toc408777787)

[Gambar 2‑14Single Linked List Insert After 3 23](#_Toc408777788)

[Gambar 2‑15 Single Linked List Delete First I 23](#_Toc408777789)

[Gambar 2‑16Single Linked List Delete First 2 24](#_Toc408777790)

[Gambar 2‑17Single Linked List Delete First 3 24](#_Toc408777791)

[Gambar 2‑18Single Linked List Delete Last 1 24](#_Toc408777792)

[Gambar 2‑19Single Linked List Delete Last 2 24](#_Toc408777793)

[Gambar 2‑20Single Linked List Delete Last 3 24](#_Toc408777794)

[Gambar 2‑21Single Linked List Delete After 1 25](#_Toc408777795)

[Gambar 2‑22Single Linked List Delete After 2 25](#_Toc408777796)

[Gambar 2‑23Single Linked List Delete After 3 25](#_Toc408777797)

[Gambar 3‑1Double Linked List dengan Elemen Kosong 34](#_Toc408777798)

[Gambar 3‑3 Circular Double Linked List 34](#_Toc408777800)

[Gambar 3‑5 Double Linked List Insert First 2 35](#_Toc408777801)

[Gambar 3‑6 Double Linked List Insert First 3 36](#_Toc408777802)

[Gambar 3‑7 Double Linked List Insert First 4 36](#_Toc408777803)

[Gambar 3‑9 Double Linked List Insert Last 2 36](#_Toc408777804)

[Gambar 3‑10Double Linked List Insert Last 3 36](#_Toc408777805)

[Gambar 3‑11 Double Linked List Insert Last 4 37](#_Toc408777806)

[Gambar 2‑10 Double Linked List Insert After 1 37](#_Toc408777807)

[Gambar 3‑13 Double Linked List Insert After 2 37](#_Toc408777808)

[Gambar 3‑14 Double Linked List Insert After 3 37](#_Toc408777809)

[Gambar 3‑15 Double Linked List Delete First 1 38](#_Toc408777810)

[Gambar 3‑16 Double Linked List Delete First 2 38](#_Toc408777811)

[Gambar 3‑17 Double Linked List Delete First 3 38](#_Toc408777812)

[Gambar 3‑18 Double Linked List Delete Last 1 38](#_Toc408777813)

[Gambar 3‑19 Double Linked List Delete Last 2 38](#_Toc408777814)

[Gambar 3‑20 Double Linked List Delete Last 3 38](#_Toc408777815)

[Gambar 3‑21 Double Linked List Delete After 1 39](#_Toc408777816)

[Gambar 3‑22 Double Linked List Delete After 2 39](#_Toc408777817)

[Gambar 3‑23 Double Linked List Delete After 3 39](#_Toc408777818)

[Gambar 4‑1 Muti Linked List 44](#_Toc408777819)

[Gambar 4‑2 Multi Linked List Insert Anak 1 45](#_Toc408777820)

[Gambar 4‑3 Multi Linked List Insert Anak 2 45](#_Toc408777821)

[Gambar 4‑4 Multi Linked List Delete Anak 1 46](#_Toc408777822)

[Gambar 4‑5 Multi Linked List Delete Anak 2 46](#_Toc408777823)

[Gambar 4‑6 Multi Linked List Delete Induk 1 47](#_Toc408777824)

[Gambar 5‑1Stack dengan 3 Elemen 52](#_Toc408777825)

[Gambar 5‑2 Stack Push 1 53](#_Toc408777826)

[Gambar 5‑3 Stack Push 2 53](#_Toc408777827)

[Gambar 5‑4 Stack Push 4 53](#_Toc408777828)

[Gambar 5‑5 Stack Pop 1 54](#_Toc408777829)

[Gambar 5‑6 Stack Pop 2 54](#_Toc408777830)

[Gambar 5‑7 Stack Pop 3 54](#_Toc408777831)

[Gambar 6‑2 Queue Insert 1 61](#_Toc408777833)

[Gambar 6‑3 Queue Insert 2 61](#_Toc408777834)

[Gambar 6‑4 Queue Insert 3 61](#_Toc408777835)

[Gambar 6‑5 Queue Delete 1 62](#_Toc408777836)

[Gambar 6‑6 Queue Delete 2 62](#_Toc408777837)

[Gambar 6‑7 Queue Delete 3 62](#_Toc408777838)

[Gambar 7‑1 arrayQueue.h 72](#_Toc408777839)

[Gambar 8‑1 Heap Tree 73](#_Toc408777840)

[Gambar 8‑2 Binary Search Tree Insert 74](#_Toc408777841)

[Gambar 8‑3 Binary Search Tree sebelum di-Delete 74](#_Toc408777842)

[Gambar 8‑4 Binary Search Tree setelah di-Delete 74](#_Toc408777843)

[Gambar 8‑5 Traversal pada Binary Tree 1 75](#_Toc408777844)

[Gambar 8‑6 traversal pada Binary Tree 2 75](#_Toc408777845)

[Gambar 9‑1 Graph Kost dan Common Lab 78](#_Toc408777846)

[Gambar 9‑2 Graph Tidak Berarah (Undirected Graph) 78](#_Toc408777847)

[Gambar 9‑3 Graph 79](#_Toc408777848)

[Gambar 9‑4 Representasi Graph Array 2 Dimensi 79](#_Toc408777849)

[Gambar 9.5 Representasi Graph Multi Linked List 79](#_Toc408777850)

[Gambar 9‑5 Graph Jarak Antar kota 79](#_Toc408777851)

[Gambar 9‑6 Graph Breadth First Search (BFS) 81](#_Toc408777852)

[Gambar 9‑7 Graph Depth First Search (DFS) 81](#_Toc408777853)

[Gambar 10‑1 Graph Berarah (Directed Graph) 83](#_Toc408777854)

[Gambar 10‑2 Graph Berarah 2 84](#_Toc408777855)

[Gambar 10‑3 Ilustrasi Topological Sort 85](#_Toc408777856)

# DAFTAR PROGRAM

[Program 1‑1 Fungsi 8](#_Toc378324154)

[Program 1‑2 Pemanggilan dengan Nilai (call by value) 9](#_Toc378324155)

[Program 1‑3 Pemeanggilan dengan Referensi (call by reference) 10](#_Toc378324156)

[Program 1‑4 Prosedur 10](#_Toc378324157)

[Program 1‑5 Rekursif 11](#_Toc378324158)

[Program 1‑6 Copy String dengan Array 1 13](#_Toc378324159)

[Program 1‑7 Copy String dengan Array 2 13](#_Toc378324160)

[Program 2‑1 Pointer 18](#_Toc378324161)

[Program 3‑1 Deklarasi Struktur Data Double Linked List 35](#_Toc378324162)

[Program 4‑1 multilist.h 50](#_Toc378324163)

[Program 4‑2 adtmulti.h 51](#_Toc378324164)

[Program 5‑1 Deklarasi Struktur Data Stack 52](#_Toc378324165)

[Program 5‑2 Stack Representasi Tabel 56](#_Toc378324166)

[Program 5‑3 Hanoi.h 59](#_Toc378324167)

[Program 6‑1 Deklarasi Struktur Data Queue 61](#_Toc378324168)

[Program 6‑2 Queue.h 64](#_Toc378324169)

[Program 6‑3 queuebarang.h 68](#_Toc378324170)

[Program 8‑1 tree.h 77](#_Toc378324171)

[Program 9‑1 Deklarasi Struktur Data Graph dengan Representasi Array 2 Dimensi 80](#_Toc378324172)

[Program 9‑2 graph.h 82](#_Toc378324173)

[Program 10‑1 topological.h 87](#_Toc378324174)

# DAFTAR TABEL

[Tabel 1‑1 Ilustrasi Array Dua Dimensi 12](#_Toc378324175)

[Tabel 9‑1 Container Queue BFS 81](#_Toc378324176)

[Tabel 9‑2 Container Stack DFS 81](#_Toc378324177)

# Running Modul

## Tujuan

Setelah mengikuti Running Modul mahasiswa diharapkan dapat:

1. Memahami peraturan kegiatan praktikum.
2. Memahami Hak dan Kewajiban praktikan dalam kegiatan praktikum.
3. Memhami komponen penilaian kegiatan praktikum.

## Peraturan Praktikum

1. Praktikum diampu oleh **Dosen Kelas** dan dibantu oleh **Asisten Laboratorium** dan **Asisten Praktikum**.
2. Praktikum dilaksanakan di Gedung FIT lantai 2 (**PRIDE LAB dan GEAR LAB**) sesuai jadwal yang ditentukan.
3. Praktikan wajib membawa **modul praktikum, kartu praktikum, dan alat tulis**.
4. Praktikan wajib mengisi **daftar hadir** dan **BAP praktikum** dengan bolpoin **bertinta hitam**.
5. Durasi kegiatan praktikum **D3 = 4 jam (200 menit)**.
6. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
7. 60 menit untuk penyampaian materi
8. 125 menit untuk pengerjaan jurnal dan tes akhir
9. Jumlah **pertemuan praktikum**:
   * 10 kali di lab (praktikum rutin)
   * 3 kali di luar lab (terkait Tugas Besar dan/atau UAS)
   * 1 kali berupa presentasi Tugas Besar dan/atau pelaksanaan UAS
10. Praktikan **wajib hadir minimal 75%** dari seluruh pertemuan praktikum di lab.Jika total kehadiran kurang dari 75% maka nilai UAS/ Tugas Besar = 0.
11. Praktikan yang datang terlambat :

* <= 30 menit : diperbolehkan mengikuti praktikum tanpa tambahan waktu Tes Awal
* > 30 menit : tidak diperbolehkan mengikuti praktikum

1. Saat praktikum berlangsung, asisten praktikum dan praktikan:
   * Wajib menggunakan **seragam** sesuai aturan Institusi.
   * Wajib mematikan/ men-silent semua **alat komunikasi**(smartphone, tab, iPad, dsb).
   * Dilarang membuka **aplikasi yang tidak berhubungan** dengan praktikum yang berlangsung.
   * Dilarang mengubah **setting software maupun hardware** komputer tanpa ijin.
   * Dilarang **membawa makanan maupun minuman** di ruang praktikum.
   * Dilarang **memberikan jawaban ke praktikan lain** (pre-test, TP, jurnal, dan post-test).
   * Dilarang **menyebarkan soal pre-test, jurnal, dan post-test**.
   * Dilarang**membuang sampah/sesuatu apapun** di ruangan praktikum.
2. Setiap praktikan dapat mengikuti praktikum susulan maksimal 2 modul untuk satu praktikum.
   * Praktikan yang dapat mengikuti praktikum susulan hanyalah praktikan yang memenuhi syarat sesuai ketentuan Institusi, yaitu rawat inap di Rumah Sakit (menunjukkan bukti rawat inap dan resep obat dari RS), tugas dari Institusi (menunjukkan surat dinas dari Institusi), atau mendapat musibah (menunjukkan surat keterangan dari orangtua/ wali mahasiswa).
   * Persyaratan untuk praktikum susulan diserahkan sesegera mungkin ke Asisten Laboratorium untuk keperluan administrasi.
   * Praktikan yang diijinkan menjadi peserta praktikum susulan ditetapkan oleh Pembina Lab& Bengkel D3 Teknik Informatika dan tidak dapat diganggu gugat.
3. Pelanggaran terhadap peraturan praktikum ini akan ditindak secara tegas secara berjenjang di lingkup Kelas, Laboratorium, Departemen, Fakultas, hingga Institusi.

## Penilaian Praktikum

1. Komponen penilaian praktikum:

**60%** nilai permodul dan **40%** nilai Tugas Besar (atau UAS praktek)

1. Seluruh komponen penilaian beserta pembobotannya ditentukan oleh dosen **PJMP**
2. Penilaian permodul dilakukan oleh **asisten praktikum**, sedangkan nilai Tugas Besar/ UAS diserahkan kepada **dosen kelas,** dilaporkan ke **PJMP.**
3. Baik praktikan maupun asisten tidak diperkenankan meminta atau memberikan **tugas tambahan** untuk perbaikan nilai.
4. Standar **indeks dan range nilai** ditentukan oleh dosen PJMP atas sepengetahuan KaBengkel Fakultas Informatika

# ADT

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Menggunakan operator-operator dalam program.
2. Memahami cara membuat program sederhana dalam bahasa C++.
3. Memahami penggunaan type data dan variable dalam bahasa C++.
4. Menggunakan operator-operator input/output dengan tepat.
5. Memahami dan meggunakan fungsi kondisional dalam program.

## Fungsi

Fungsi merupakan blok dari kode yang dirancang untuk melaksanakan tugas khusus dengan tujuan:

1. Program menjadi terstruktur, sehingga mudah dipahami dan mudah dikembangkan. Program dibagi menjadi beberapa modul yang kecil.
2. Dapat mengurangi pengulangan kode (duplikasi kode) sehingga menghemat ukuran program.

Pada umumnya fungsi memerlukan masukan yang dinamakan sebagai parameter. Masukan ini selanjutnya diolah oleh fungsi. Hasil akhir fungsi berupa sebuah nilai (nilai balik fungsi).

Bentuk umum sebuah fungsi:

*tipe\_keluarannama\_fungsi*(*daftar\_parameter*) {

*blok pernyataan fungsi* ;

}

Jika penentu\_tipe fungsi merupakan tipe dari nilai balik fungsi, bila tidak disebutkan maka akan dianggap (default) sebagai int.

|  |
| --- |
| /\*----------------------------------------------------------------\*/  /\* contoh penulisan fungsi \*/  /\*----------------------------------------------------------------\*/  #include <conio.h>  #include <iostream>  #include <stdlib.h>  Using namespace std;  int maks3(int a, int b, int c); /\*mendeklarasikan prototype fungsi \*/  int main(){  system(“cls”);  int x,y,z;  cout<<"masukkan nilai bilangan ke-1 =";  cin>>x;  cout<<"masukkan nilai bilangan ke-2 =";  cin>>y;  cout<<"masukkan nilai bilangan ke-3 =";  cin>>z;  cout<<"nilai maksimumnya adalah = "<<maks3(x,y,z);  getch();  return 0;  }  int maks3(int a, int b, int c) { /\*badan fungsi \*/  int temp\_max=a; /\*deklarasi variabel lokal dalam fungsi \*/    if(b>temp\_max)  temp\_max=b;  if(c>temp\_max)  temp\_max=c;    return (temp\_max);  } |

Program ‑ Fungsi

### Paramater Formal dan Parameter Aktual

Parameter formal adalah variabel yang ada pada daftar paramerter ketika mendefinisikan fungsi. Pada fungsi maks3() contoh diatas, a, b dan merupakan parameter formal.

*float perkalian (float x, float y) {*

*return (x\*y);*

*}*

Pada contoh diatas x dan y adalah parameter formal.

Adapun parameter aktual adalah parameter (tidak selamanya menyatakan variabel) yang dipakai untuk memanggil fungsi.

*X = perkalian(a,b);*

*Y = perkalian(10,30);*

Dari pernyataan diatas a dan b merupakan parameter aktual, begitu pula 10 dan 30. Parameter aktual tidak harus berupa variabel, melainkan bisa berupa konstanta atau ungkapan.

### Cara melewatkan Parameter

#### Pemanggilan dengan Nilai (call by value)

Pada pemanggilan dengan nilai, nilai dari parameter actual akan disalin kedalam parameter formal, jadi parameter aktual tidak akan berubah meskipun parameter formalnya berubah. Untuk lebih jelasnya perhatikan contoh berikut:

|  |
| --- |
| /\*------------------------------\*/  /\*contoh melewatkan parameter \*/  /\* call by value \*/  /\*------------------------------\*/  #include <iostream>  #include <conio.h>  #include <stdlib.h>  Using namespace std;  void tukar(int x, int y); /\*prototype fungsi \*/  int main () {  int a, b;  system (“cls”);  a=4;  b=6;  cout<<"kondisi sebelum ditukar \n";  cout<<" a = "<<a<<" b = "<<b<<endl;  tukar(a,b);  cout<<"kondisi setelah ditukar \n";  cout<<" a = "<<a<<" b = "<<b<<endl;  getch();    return 0;  }  void tukar (int x, int y) {  int temp;  temp = x;  x = y;  y = temp;  cout<<"nilai akhir pada fungsi tukar \n";  cout<<" x = "<<x<<" y = "<<y<<endl;  } |

Program ‑ Pemanggilan dengan Nilai (call by value)

Hasil eksekusi :

Kondisi sebelum tukar

*a = 4, b = 6*

nilai akhir pada fungsi tukar

*x = 6, y = 4*

kondisi sesudah tukar

*a = 4, b = 6*

Jelas bahwa pada pemanggilan fungsi tukar, yang melewatkan variabel a dan b tidak merubah nilai dari variabel tersebut. Hal ini dikarenakan ketika pemanggilan fungsi tersebut nilai dari a dan b disalin ke variabel formal yaitu x dan y.

#### Pemanggilan dengan Referensi (call by reference)

Pemanggilan dengan referensi merupakan cara untuk melewatkan alamat suatu variabel kedalam suatu fungsi. Dengan cara ini dapat merubah nilai dari variabel aktual yang dilewatkan ke dalam fungsi. Jadi cara ini dapat merubah nilai variabel yang ada diluar fungsi. Cara penulisan :

*tukar(int \*px, int \*py) {*

*int temp;*

*temp = \*px;*

*\*px = \*py;*

*\*py = temp;*

*… …*

*}*

Cara pemanggilan:

*tukar(&a, &b);*

Pada ilustrasi diatas \*px merupakan suatu variabel pointer yang menunjuk ke suatu variabel integer. Pada pemanggilan fungsi tukar(), &a dan &b menyatakan “alamat a” dan “alamat b”. Dengan cara diatas maka variabel yang diubah dalam fungsi tukar() adalah variabel yang dilewatkan dalam fungsi itu juga, karena yang dilewatkan dalam fungsi adalah alamat dari variabel tersebut, jadi bukan sekedar disalin.

|  |
| --- |
| /\*------------------------------\*/  /\*contoh melewatkan parameter \*/  /\* call by references \*/  /\*------------------------------\*/  #include <iostream>  #include <conio.h>  #include <stdlib.h>  Using namespace std;  void tukar(int \*x, int \*y); /\*prototype fungsi \*/  int main () {  int a, b;  system(“cls”);  a=4;  b=6;  cout<<"kondisi sebelum ditukar \n";    cout<<" a = "<<a<<" b = "<<b<<endl;  tukar(&a,&b);  cout<<"kondisi setelah ditukar \n";  cout<<" a = "<<a<<" b = "<<b<<endl;  getch();  return 0;  }  void tukar (int \*x, int \*y) {  int temp;  temp = \*x;  \*x = \*y;  \*y = temp;  cout<<"nilai akhir pada fungsi tukar"<<endl;  cout<<" x = "<<x<<" y = "<<y<<endl;  } |

Program ‑ Pemeanggilan dengan Referensi (call by reference)

## Prosedur

Dalam C sebenarnya tidak ada prosedure, semua berupa fungsi, termasuk main() pun adalah sebuah fungsi. Jadi prosedur dalam C merupakan fungsi yang tidak mengembalikan nilai, biasa diawali dengan kata kunci void di depan nama prosedur.

Bentuk umum sebuah prosedur:

void *nama\_prosedur* (*daftar\_parameter*) {

*blok pernyataan prosedur* ;

}

Program ‑ Prosedur

|  |
| --- |
| /\*------------------------------\*/  /\* contoh prosedur \*/  /\*------------------------------\*/  #include <iostream>  #include <conio.h>  #include <stdlib.h>  Using namespace std;  void tulis(int x); /\*prototype fungsi\*/  int main(){  system(“cls”);  int jum;  cout<<" jumlah baris kata=";  cin>>jum;  tulis(jum);    getch();  return 0;  }  void tulis(int x) { /\*badan prosedur\*/  for (int i=0;i<x;i++)  cout<<"baris ke-"<<i+1<<endl;  } |

## Rekursif

Rekursif berarti suatu fungsi dapat memanggil fungsi yang merupakan dirinya sendiri.

|  |
| --- |
| #include <conio.h>  #include <iostream>  #include <stdlib.h>  Using namespace std;  int pangkat(int x, int y); /\* prototype fungsi rekursif\*/  int main(){ /\*fungsi utama\*/  system(“cls”);  int bil, bil\_pkt;  cout<<"menghitung x^y \n";  cout<<"x=";  cin>>bil;  cout<<"\ny=";  cin>>bil\_pkt;  cout<<"\n<<bil<<"^"<<bil\_pkt<<"="<<pangkat(bil,bil\_pkt));  /\*pemanggilan fungsi rekursif\*/    getch();  return 0;  }  int pangkat(int x, int y) { /\*badan fungsi rekursif\*/  if (y==1)  return(x);  else  return(x\*pangkat(x,y-1)); /\*bentuk penulisan rekursif\*/  } |

Program ‑ Rekursif

## Array

Array merupakan kumpulan data dengan nama yang sama dan setiap elemen bertipe data sama. Untuk mengakses setiap komponen / elemen array berdasarkan indeks dari setiap elemen.

### Array Satu Dimensi

Adalah array yang hanya terdiri dari satu larik data saja. Cara pendeklarasian array satu dimensi:

*tipe\_data*nama\_var[*ukuran*]

Keterangan:

Tipe\_data 🡪 menyatakan jenis elemen array (int, char, float, dll).

Ukuran 🡪 menyatakan jumlah maksimum array.

Contoh:

*int nilai[10];*

Menyatakan bahwa array nilai mengandung 10 elemen dan bertipe integer.

Dalam C++ data array disimpan dalam memori pada lokasi yang berurutan. Elemen pertama memiliki indeks 0 dan selemen selanjutnya memiliki indeks 1 dan seterusnya. Jadi jika terdapat array dengan 5 elemen maka elemen pertama memiliki indeks 0 dan elemen terakhir memiliki indeks 4.

***nama\_var[indeks]***

*nilai[5] 🡪 elemen ke-5 dari array nilai.*

Contoh memasukkan data kedalam array :

*nilai[4] = 90; /\*memasukkan 90 kedalam array nilai indeks ke-4\*/*

*cin>>nilai[4]; /\*membaca inputan dari keyboard\*/*

### Array Dua Dimensi

Bentuk array dua dimensi ini mirip seperti tabel. Jadi array dua dimensi bisa digunakan untuk menyimpan data dalam bentuk tabel. Terbagi menjadi dua bagian, dimensi pertama dan dimensi kedua.

Misalkan pendeklarasian data nilai siswa tiap tahun.

*int data\_nilai[4][3];*

Nilai 3 menyatakan banyaknya tahun, dan nilai 4 menyatakan banyaknya siswa.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 90 | 70 | 40 |
| 1 | 56 | 43 | 33 |
| 2 | 65 | 67 | 78 |
| 3 | 96 | 81 | 55 |

Tabel ‑ Ilustrasi Array Dua Dimensi

Ilustrasi tersebut dapat dibaca sebagai berikut.

Pada tahun ke-0, siswa ke-0 nilainya 90, siswa ke-1 nilainya 70, dan siswa ke-2 memiliki nilainya 40.

Pada tahun ke-1, siswa ke-0 nilainya 56, siswa ke-1 nilainya 43, dan siswa ke-2 memiliki nilainya 33.

Pada tahun ke-2, siswa ke-0 nilainya 65, siswa ke-1 nilainya 67, dan siswa ke-2 memiliki nilainya 78.

Pada tahun ke-3, siswa ke-0 nilainya 96, siswa ke-1 nilainya 81, dan siswa ke-2 memiliki nilainya 55.

Cara akses, deklarasi, inisialisasi, dan menampilkan data sama dengan array satu dimensi, hanya saja indeks yang digunakan ada dua.

Contoh :

int nilai[4][3];

### Array Berdimensi Banyak

Merupakan array yang mempunyai indeks banyak, lebih dari dua. Indeks inilah yang menyatakan dimensi array. Array berdimensi banyak lebih susah dibayangkan, sejalan dengan jumlah dimensi dalam array.

Cara deklarasi:

*tipe\_data* nama\_var[*ukuran1*][*ukuran2*]…[*ukuran-N*];

Contoh:

int data\_rumit[4][6][6];

Array sebenarnya masih banyak pengembangannya untuk penyimpanan berbagai betuk data, pengem bangan array misalnya untuk array tak berukuran.

## String

String merupakan bentuk data yang sering digunakan dalam bahasa pemrograman untuk mengolah data teks atau kalimat. Dalam bahasa C pada dasarnya string merupakan kumpulan dari karakter atau array dari karakter.

Deklarasi variabel string:

*char nama[50];*

50 🡪 menyatakan jumlah maksimal karakter dalam string.

Memasukkan data string dari keyboard:

***gets(nama\_array);***

*gets(nama);*

Menggunakan cin

*cin>>nama;*

Inisialisasi string:

*char nama[]= {‘s’,’t’,’r’,’u’,’k’,’d’,’a’,’t’,’\0’};*

Merupakan variabel nama dengan isi data string “strukdat”.

Bentuk inisialisasi yang lebih singkat:

*char nama[]=”strukdat”;*

Menampilkan string bisa menggunakan puts() atau printf() :

*puts(nama);*

*cout<< nama;*

Untuk mengakses data string sepertihalnya mengakses data pada array, pengaksesan dilakukan perkarakter sesuai dengan indeks setiap karakter dalam string.

Contoh :

*Cout<< nama[3]); /\*menampilkan karakter ke-3 dari string\*/*

Kita akan mengilustrasikan aspek dari array dengan mempelajari dari fungsi yang diambil dari library standar <string.h>. Fungsi tersebut contohnya adalah strcpy (s,t), merupakan fungsi yang akan meng-copy-kan string t ke string s. Dapat dikatakan juga bahwa s=t. Akan tetapi, di sini akan mengcopykan pointer bukan karakter. Untuk mengcopykan karakter kita membutuhkan perulangan, berikut ini pengcopyan string menggunakan array.

|  |
| --- |
| /\*strcpy: mengcopy t ke s dengan menggunakan array\*/  void strcpy (char \*s, char \*t) {  int i;  i=0;  while ((s[i]=t[i] != ‘\0’)) {  i++;  }  } |

Program ‑ Copy String dengan Array 1

|  |
| --- |
| /\*strcpy: mengcopy t ke s dengan menggunakan array\*/  void strcpy (char \*s, char \*t) {  int i;  i=0;  while ((s[i]=t[i] != ‘\0’)) {  s++;  t++;  }  } |

Program ‑ Copy String dengan Array 2

Karena argument dilewatkan melalui passing by value,strcpy dapat menggunakan parameter t dimanapun ditempatkan.Hal ini sesuai dengan inisialisasi pointer, dimana akan membuat array menjadi karakter-karakter sampai pada karakter null yang merupakan batas dari t setelah di copykan ke s.

## Abstract Data Type (ADT)

ADT adalah TYPE dan sekumpulan PRIMITIF (operasi dasar) terhadap TYPE tersebut.Selain itu, dalam sebuah ADT yang lengkap, disertakan pula definisi invarian dari TYPE dan aksioma yang berlaku.ADT merupakan definisi STATIK.

Definisi type dari sebuah ADT dapat mengandung sebuah definisi ADT lain.

Misalnya :

1. ADT waktu yang terdiri dari ADT JAM dan ADT DATE
2. Garis terdiri dari duah buah ADT POINT

SEGI4 yang terdiri dari pasangan dua buah POINT (Top,Left) dan (Bottom,Right)

TYPE diterjemahkan menjadi type terdefinisi dalam bahasa yang bersangkutan. Jika dalam bahasa C++ menggunakan struct PRIMITIF, dalam konteks procedural, diterjemahkan menjadi fungsi atau prosedur. PRIMITIF dikelompokan menjadi :

1. Konstruktor/Kreator, pembentuk nilai type.Semua objek (variabel) bertype tersebut harus melalui konstruktor.Biasanya namanya diawali Make.
2. Selektor, untuk mengakses komponen type (biasanya namanya diawali Get).
3. Prosedur pengubah nilai komponen (biasanya namanya diawali Get).
4. Validator komponen type, yang dipakai untuk mentest apakah dapat membentuk type sesuai dengan batasan.
5. Destruktor/Dealokator yaitu untuk “menghancurkan” nilai objek/variable (sekaligus memory penyimpanannya).
6. Baca/Tulis, untuk interface dengan input/output device.
7. Operator relational, terhadap type tersebut untuk mendefinisikan lebih besar, lebih kecil, sama dengan dan sebagainya.
8. Aritmatika terhadap type tersebut , karena biasanya aritmatika dalam bahasa C hanya terdefinisi untuk bilangan numeric.
9. Konversi dari type tersebut ke type dasar dan sebaliknya.

ADT biasanya diimplementasikan menjadi dua buah modul utama dan 1 modul interface program utama(driver). Dua modul tersebut adalah sebagai berikut:

1. Definisi/Spesifikasi Type dan Primitif / Header fungsi (.h)

* Spesifikasi type sesuai dengan kaidah bahasa yang dipakai
* Spesifikasi dari primitive sesuai dengan kaidah dalam konteks procedural,yaitu:
* Fungsi : nama, domain, range, dan prekondisi jika ada
* Prosedur : Initial state, Final state, dan proses yang dilakukan

1. Body/relisasi dari primitive (.c)

Berupa kode program dalam bahasa yang bersangkutan (dalam praktikum ini berarti dengan kaidah bahasa C. Realisasi fungsi dan prosedur harus sedapat mungkin memanfaatkan selector dan konstruktor.

|  |
| --- |
| /\* file : point.h \*/  /\* contoh ADT point \*/  #ifndef POINT\_H  #define POINT\_H  #include <stdio.h>  #include <math.h>  #include "boolean.h"  point MakePoint(int x,int y);  /\*mengembalikan nilai point dengan absis x dan ordinat y\*/  int GetAbsis(point p);  /\*mengembalikan nilai absis dari point p\*/  int GetOrdinat(point p);  /\*mengembalikan nilai ordinat dari point p\*/  void SetAbsis(point \*p,int newX);  /\*IS:nilai absis point p masih kosong\*/  /\*FS:nilai absis point p=newx\*/  void SetOrdinat(point \*p,int newY);  /\*IS:nilai ordinat point p masih kosong\*/  /\*FS:nilai ordinat point p=newx\*/  void BacaPoint(point \*p);  /\*IS:nilai point p masih belumada\*/  /\*FS:memperoleh nilai point p dari inputan user\*/  void TulisPoint(point p);  /\*IS:nilai point p sudah ada\*/  /\*FS:nilai point ditampilkan di layar\*/  #endif |

Program ‑ 8 point.h

|  |
| --- |
| /\* file : point.c \*/  /\* tempat mendefinisikan semua badan fungsi dari file point.h \*/  /\* contoh ADT point \*/  #include <iostream>  #include <math.h>  #include "boolean.h"  #include “point.h”  Using namespace std;  typedef struct titik {  int x;  int y;  }point;  point MakePoint(int x,int y){  point p;  p.x=x;  p.y=y;  return p;  }  int GetAbsis(point p) {  return(p.x);  }  int GetOrdinat(point p) {  return(p.y);  }  void SetAbsis(point \*p,int newX) {  cin>>newX;  (\*p).x=newX;  }  void SetOrdinat(point \*p,int newY) {  cin>>newY;  (\*p).y=newY;  }  void BacaPoint(point \*p) {  int x,y;  cout<<"Nilai x: ";  cin>>x;  cout<<"Nilai y: ";  cin>>y;  \*p=MakePoint(x,y);  }  void TulisPoint(point p) {  cout<<"("<<p.x<<","<<p.y<<")";  } |

Program ‑ 9 point.c

Program diatas tidak akan dapat dijalankan tanpa dibuat drivernya di .c.Untuk driver aplikasikan sendiri-sendiri sesuai kreatifitas masing-masing yang penting dapat mengaplikasikan semua fungsi diatas.

## Latihan

1. Buatlah FUNGSI atau PROSEDUR untuk menghitung faktorial dari suatu bilangan secara rekursif.

Contoh:

input bilangan: 5 <inputan>

faktorial dari 5 adalah 120 <outputan>

1. Buat program yang dapat menyimpan data mahasiswa (max. 10) kedalam sebuah array dengan field nama, nim, uts, uas, tugas, dan nilai akhir. Nilai akhir diperoleh dari FUNGSI dengan rumus 0.3\*uts+0.4\*uas+0.3\*tugas.
2. Studi Kasus

Buatlah sebuah program pemrosesan data mahasiswa sederhana yang mengimplementasikan konsep ADT.

Ketentuannya :

* + - Tipe bentukan Mahasiswa berupa Nama, NIM, dan Nilai
    - Terdiri dari 3 file, main.c, mahasiswa.c, mahasiswa.h
    - Menu program terdiri dari Input Data Mahasiswa, View Data Mahasiswa, dan Cari Data Mahasiswa
    - View Data Mahasiswa termasuk menampilkan Nilai Rata-rata, Tertinggi, dan Terrendah
    - Menu program tambahan terdiri dari Update Data Mahasiswa dan Delete Data Mahasiswa (nilai bonus)

file mahasiswa.h harus berisi :

|  |
| --- |
| typedef struct {  // isi sendiri  } Mahasiswa;  typedef struct {  // isi sendiri  } TabMahasiswa;  void InputMahasiswa (Mahasiswa M, TabMahasiswa \*T);  void ViewMahasiswa (TabMahasiswa T);  void CariMahasiswa (char NIM[10], TabMahasiswa T);  float NilaiRata2 (TabMahasiswa T);  float NilaiTertinggi (TabMahasiswa T);  float NilaiTerrendah (TabMahasiswa T); |

Program ‑ 10 mahasiswa.h

# SINGLE LINKED LIST

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Memahami penggunaan linked list dengan menggunakan pointer operator- operator dalam program.
2. Memahami operasi-operasi dasar dalam linked list.
3. Membuat program dengan menggunakan linked list dengan prototype yang ada

## Pointer

### Pointer dan Alamat

Variabel pointer sering dikatakan sebagai variable yang menunjuk ke objek lain. Akan tetapi, sebenarnya pointer berisi alamat dari suatu objek lain.

Misalkan px adalah variabel pointer dan x adalah variabel yang ditunjuk oleh px. Jika x berada pada alamat memory 1000, maka px akan berisi 1000. Cara pendeklarasian variable pointer adalah sebagai berikut:

type \*nama\_variabel;

Contoh:

int x;

int \*px;

/\*px merupakan variable pointer yang menunjuk ke data bertipe int \*/

Agar suatu pointer menunjuk ke variable lain, mula-mula pointer harus diisi dengan alamat memory yang ditunjuk.Untuk menyatakan alamat dari suatu variable operator “&” bisa digunakan dengan cara menempatkan di depan nama variable. Sebagai contoh bila **x** merupakan variable yang dideklarasikan dengan type int maka penulisannya :

&x

Artinya “alamat dari variable x”. Adapun contoh dari pemberian alamat ke suatu variable **px** yaitu:

px=&x

Pernyataan diatas berarti bahwa **px** diberi nilai berupa alamat dari variable **x**.Setelah pernyataan tersebut di eksekusi maka dapat dikatakan bahwa px menunjuk ke variable **x**. Jika suatu variable sudah ditunjuk oleh pointer. Maka, variable yang ditunjuk oleh pointer dapat diakses melalui variable itu sendiri ataupun melalui pointer.

Contoh:

*int y;*

*px=&x;*

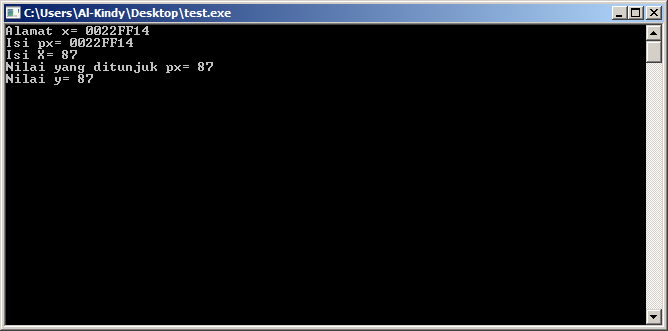
*y=\*px;*

Maka variable **y** akan berisi nilai sesuai dengan nilai **x**.

Berikut ini contoh program sederhana menggunakan pointer:

|  |
| --- |
| #include <iostream>  #include <conio.h>  using namespace std;  int main(){  int x,y; //x dan y bertipe int  int \*px; //px merupakan varibel pointer menunjuk ke variabel int  x=87;  px=&x;  y=\*px;  cout<<"Alamat x= "<<&x<<endl;  cout<<"Isi px= "<<px<<endl;  cout<<"Isi X= "<<x<<endl;  cout<<"Nilai yang ditunjuk px= "<<\*px<<endl;  cout<<"Nilai y= "<<y<<endl;  getch();  return 0;  } |

Program ‑ Pointer



Gambar ‑ Output Pointer

Pada program di atas pernyataan :

px=&x;

y=\*px;

sebenarnya bisa digantikan dengan :

y=x;

## Linked List dengan Pointer

Linked List (biasa disebut list saja) adalah salah satu bentuk struktur data (representasi penyimpanan) berupa serangkaian elemen data yang saling berkait (berhubungan) dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Data yang disimpan dalam Linked List bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variable), misalnya : nama bertipe string. Sedangkan data majemuk merupakan sekumpulan data (record) yang di dalamnya terdiri dari berbagai tipe data, misalnya : Data Mahasiswa, yang terdiri dari dari Nama bertipe string, NIM bertipe long integer, dan Alamat bertipe string.

Linked List dapat diimplementasikan menggunakan Array dan Pointer (Linked List).

Yang akan kita gunakan adalah pointer, karena beberapa alasan, yaitu :

1. Array bersifat statis, sedangkan pointer dinamis.
2. Pada linked list bentuk datanya saling bergandengan (berhubungan) sehingga lebih mudah memakai pointer.
3. Sifat linked list yang fleksibel lebih cocok dengan sifat pointer yang dapat diatur sesuai kebutuhan.
4. Karena array lebih susah dalam menangani linked list, sedangkan pointer lebih mudah.
5. Array lebih cocok pada kumpulan data yang jumlah elemen maksimumnya sudah diketahui dari awal.

Dalam implementasinya, pengaksesan elemen pada Linked List dengan pointer bisa menggunakan -> atau tanda titik (.).

Model-model dari ADT Linked List yang kita pelajari adalah :

1. Single Linked List
2. Double Linked List
3. Circular Linked List
4. Multi Linked List
5. Stack (Tumpukan)
6. Queue (Antrian)
7. Tree
8. Graph

Setiap model ADT Linked List di atas memiliki karakteristik tertentu dan dalam penggunaannya disesuaikan dengan kebutuhan.

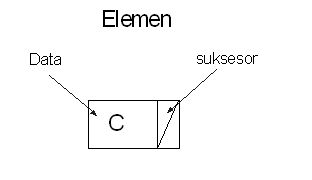
Secara umum operasi-operasi ADT pada Linked List, yaitu :

1. Penciptaan dan inisialisasi list (Create List).
2. Penyisipan elemen list (Insert).
3. Penghapusan elemen list (Delete).
4. Penelusuran elemen list dan menampilkannya (View).
5. Pencarian elemen list (Searching).
6. Pengubahan isi elemen list (Update).

## Single Linked List

Single Linked List merupakan model ADT Linked List yang hanya memiliki satu arah pointer.

Komponen elemen dalam single linked list:



Gambar ‑ Elemen Single Linked List

Keterangan:

* Elemen : segmen-segmen data yang terdapat dalam suatu list.
* Data : informasi utama yang tersimpan dalam sebuah elemen.
* Suksesor : bagian elemen yang berfungsi sebagai penghubung antar elemen.

Sifat dari Single Linked List:

1. Hanya memerlukan satu buah pointer.
2. Node akhir menunjuk ke Nil kecuali untuk list sircular.
3. Hanya dapat melakukan pembacaan maju.
4. Pencarian sequensial dilakukan jika data tidak terurut.
5. Lebih mudah ketika melakukan penyisipan atau penghapusan di tengah list.

Istilah-istilah dalam Single Linked List :

1. first/head: pointer pada list yang menunjuk alamat elemen pertama list.
2. next: pointer pada elemen yang berfungsi sebagai suksesor (penunjuk) alamat elemen didepannya.
3. Null/Nil: artinya tidak memiliki nilai, atau tidak mengacu ke mana pun, atau kosong.
4. Node/simpul/elemen: merupakan tempat penyimpanan data pada suatu memori tertentu.

Gambaran sederhana single linked list dengan elemen kosong:



Gambar ‑ Single Linked List dengan Elemen Kosong

Gambaran sederhana single linked list dengan 3 elemen:



Gambar ‑ Single Linked List dengan 3 Elemen

Contoh deklarasi struktur data single linked list:

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

|  |
| --- |
| /\*file : list.h\*/  #ifndef list\_H  #define list\_H  #include "boolean.h"  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define first(L) ((L).first)  /\*deklarasi record dan struktur data list\*/  typedef int infotype;  typedef struct tElmlist \*address;  typedef struct tElmlist{  infotype info;  address next;  }elmlist;  /\* definisi list : \*/  /\* list kosong jika Fisrt(L)=Nil \*/  typedef struct{  address first;  }list;  #endif |

Program 2.1 Deklarasi Struktur Data Single Linked List

Pada single linked list yang standar suksesor next pada elemen terakhir menunjuk Nil. Pengembangan single linked list adalah circular linked list, yaitu single linked list yang suksesor next elemen terakhirnya tidak bernilai Nil, tetapi menunjuk pada elemen awal list. Jadi sifat pada circular list adalah bisa kembali mengakses elemen awal list tanpa harus melakukan backtracking, tetapi dengan melakukan iterasi maju terus sampai kembali ke elemen awal. Gambar circular linked list:



Gambar ‑ Circular Single Linked List

Operasi – operasi pada circular linked list pada dasarnya sama dengan single linked list biasa. Perbedaannya hanya pada suksesor next elemen terakhir yang menunjuk ke elemen awal list.

### Pembentukan Komponen-Komponen List

#### Pembentukan List

Adalah sebuah proses untuk membetuk sebuah list baru. Biasanya nama fungsi yang digunakan createList(). Fungsi ini akan mengeset nilai awal list yaitu first(list) dan last(list) dengan nilai Nil.

#### Pengalokasian Memori

Adalah proses untuk mengalokasikan memori untuk setiap elemen data yang ada dalam list. Fungsi yang biasanya digunakan adalah nama fungsi yang biasa digunakan alokasi(). Fungsi ini akan mengalokasikan memori sesuai dengan besarnya elemen yang dialokasikan.

Sintak alokasi:

P = (address)malloc(sizeof(elm\_list));

Keterangan:

* P : variabel pointer yang mengacu pada elemen yang dialokasikan.
* Address : tipe data pointer dari tipe data elemen yang akan dialokasikan.
* elm\_list : tipe data atau record elemen yang dialokasikan.

Fungsi akan mengembalikan nilai bertipe address jika alokasi berhasil dan mengembalikan nilai Nil jika alokasi gagal.

#### Pengecekan List

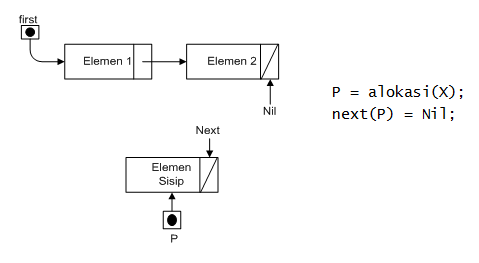
Adalah fungsi untuk mengecek apakah list tersebut kosong atau tidak. Akan mengembalikan nilai true jika list kosong dan nilai false jika list tidak kosong. Nama fungsi yang bisanya digunakan adalah isEmty().

### Insert

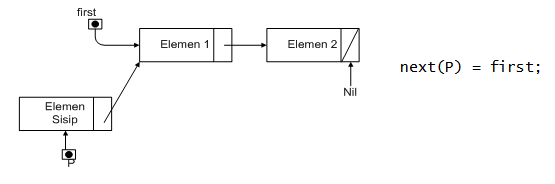
#### Insert First

Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada awal list.

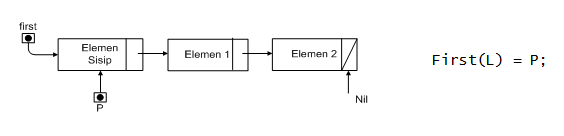
Langkah-langkah dalam proses insert first:



Gambar ‑6 Single Linked List Insert First I



Gambar ‑Single Linked List Insert First 2

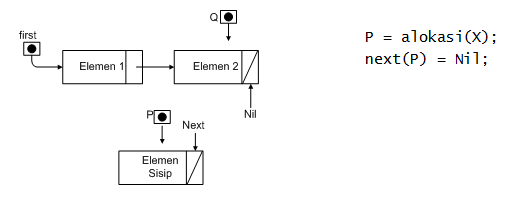


Gambar ‑Single Linked List Insert First 3

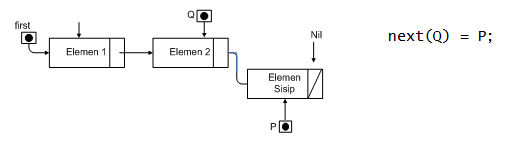
#### Insert Last

Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada akhir list.

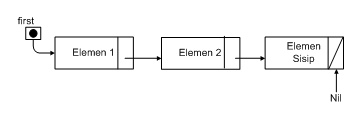
Langkah dalam insert last :



Gambar ‑Single Linked List Insert Last 1



Gambar ‑Single Linked List Insert Last 2

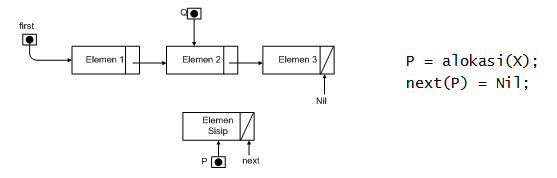


Gambar ‑Single Linked List Insert Last 3

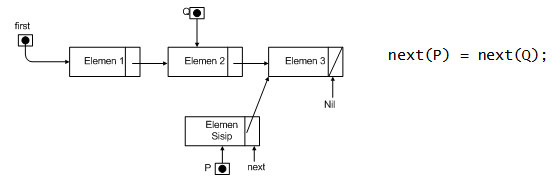
#### Insert After

Merupakan metode memasukkan data ke dalam list yang diletakkan setelah node tertentu yang ditunjuk oleh user.

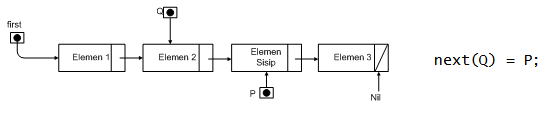
Langkah dalam insert after:



Gambar ‑Single Linked List Insert After 1



Gambar ‑Single Linked List Insert After 2



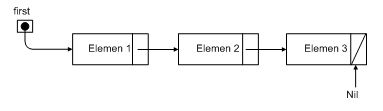
Gambar ‑Single Linked List Insert After 3

### Delete

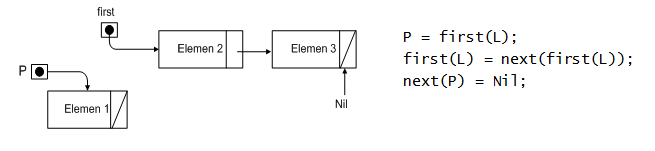
#### Delete First

Adalah pengambilan atau penghapusan sebuah elemen pada awal list.

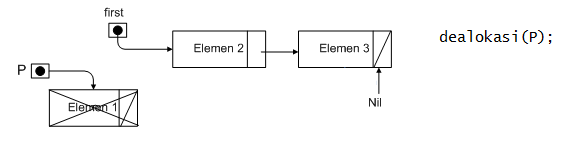
Langkah-langkah dalam delete first:



Gambar ‑Single Linked List Delete First I



Gambar ‑Single Linked List Delete First 2

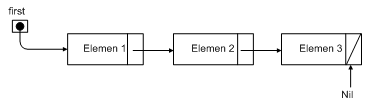


Gambar ‑Single Linked List Delete First 3

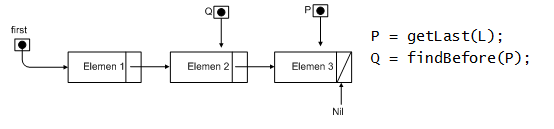
#### Delete Last

Merupakan pengambilan atau penghapusan suatu elemen dari akhir list.

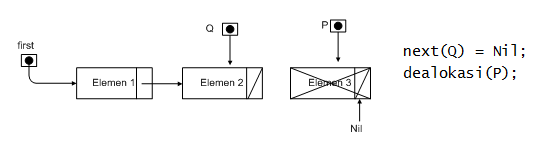
Langkah-langkah dalam delete last:



Gambar ‑Single Linked List Delete Last 1



Gambar ‑Single Linked List Delete Last 2

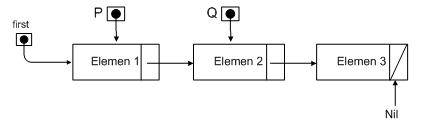


Gambar ‑Single Linked List Delete Last 3

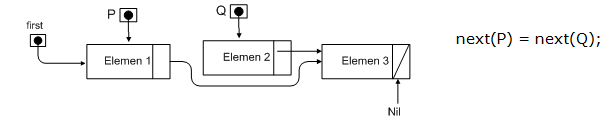
#### Delete After

Merupakan pengambilan atau penghapusan node setelah node tertentu.

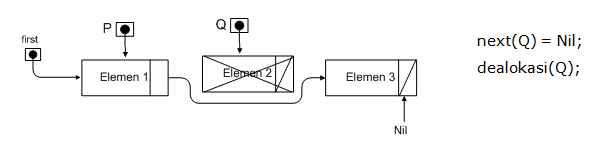
Langkah-langkah dalam delete after:



Gambar ‑Single Linked List Delete After 1



Gambar ‑Single Linked List Delete After 2



Gambar ‑Single Linked List Delete After 3

#### Delete Elemen

Adalah operasi yang digunakan untuk menghapus dan membebaskan memori yang dipakai oleh elemen tersebut.

Fungsi yang biasanya dipakai:

1. Fungsi dealokasi(P): membebaskan memori yang dipakai oleh elemen P.
2. Fungsi delAll(L): membebaskan semua memori yang dipakai elemen – elemen yang ada pada list L. Hasil akhir list L menjadi kosong.

Semua operasi-operasi dasar list biasa disebut dengan operasi primitif. Primitif-primitif dalam list ini merupakan bagian dari ADT list yang tersimpan dalam file \*.h dan file \*.c, dengan rincian file \*.h untuk menyimpan prototipe primitive-primitif atau fungsi-fungsi dan menyimpan tipedata-tipedata yang dipergunakan dalam primitif list tersebut.

Untuk bisa mengakses semua primitif tersebut yaitu dengan meng-include terhadap file \*.h-nya.

### Update

Merupakan operasi dasar pada list yang digunakan untuk mengupdate data yang ada di dalam list. Dengan operasi update ini kita dapat mengupdate data-data node yang ada di dalam list. Proses update biasanya diawali dengan proses pencarian terhadap data yang akan diupdate.

### View

Merupakan operasi dasar pada list yang menampilkan isi node/simpul dengan suatu penelusuran list. Mengunjungi setiap node kemudian menampilkan data yang tersimpan pada node tersebut.

### Searching

Searching merupakan operasi dasar list dengan melakukan aktivitas pencarian terhadap node tertentu. Proses ini berjalan dengan mengunjungi setiap node dan berhenti setelah node yang dicari ketemu. Dengan melakukan operasi searching, operasi-operasi seperti insert after, delete after, dan update akan lebih mudah.

Semua fungsi dasar diatas merupakan bagian dari ADT dari singgle linked list, dan aplikasi pada bahasa pemrograman C semua ADT tersebut tersimpan dalam file \*.cpp dan file \*.h.

|  |
| --- |
| /\*file : list.h\*/  /\* contoh ADT list berkait dengan representasi fisik pointer\*/  /\* representasi address dengan pointer\*/  /\* info tipe adalah integer \*/  #ifndef list\_H  #define list\_H  #include "boolean.h"  #include <stdio.h>  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define first(L) ((L).first)  /\*deklarasi record dan struktur data list\*/  typedef int infotype;  typedef struct tElmlist \*address;  typedef struct tElmlist{  infotype info;  address next;  }elmlist;  /\* definisi list : \*/  /\* list kosong jika Fisrt(L)=Nil \*/  /\* setiap elemen address P dapat diacu info(P) atau next(P) \*/  typedef struct{  address first;  }list;  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah list kosong \*\*\*\*\*\*\*\*\*/  boolean ListEmpty(list L);  /\* mengembalikan nilai true jika list kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void CreateList(list \*L);  /\* I.S. sembarang \*/  /\* F.S. terbentuk list kosong \*/  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotype X);  /\* mengirimkan address dari alokasi sebuah elemen \*/  /\* jika aloksi berhasil maka nilai address tidak Nil dan jika gagal nilai address Nil \*/  void dealokasi(address P);  /\* I.S. P terdefinisi \*/  /\* F.S. memori yang digunakan P dikembalikan ke sistem \*/  /\*\*\*\*\*\*\*\*\* pencarian sebuah elemen list \*\*\*\*\*\*\*\*\*/  address findElm(list L, infotype X);  /\* mencari apakah ada elemen list dengan info(P) = X\*/  /\* jika ada mengembalikan address elemen tsb, dan Nil jika sebaliknya\*/  address findBefore(list L, infotype X);  /\* mengembalikan address elemen sebelum Prec dimana info(Prec) = X\*/  /\* jika elemen dengan info = X tidak ada maka mengembalikan nilai Nil\*/  /\* jika Prec berada pada awal list maka mengembalikan nilai Nil\*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan nilai \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen pada list \*/  void insertNFirst(list \*L, infotype X);  /\* I.S. List mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada awal list dengan nilai X \*/  void insertNLast(list \*L, infotype X);  /\* I.S. List mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada akhir list dengan nilai X \*/  /\*\*\*\*\*\*\*\*\* penghapusan elemen pada list \*/  void delNFirst(list \* L, infotype \* X);  /\* I.S. List tidak kosong \*/  /\* F.S. elemen pertama dihapus: nilai info disimpan pada X dan elemen pertama di-dealokasi \*/  void delNLast(list \* L, infotype \* X);  /\* I.S. List tidak kosong \*/  /\* F.S. elemen terakhir dihapus: nilai info disimpan pada X dan elemen terakhir di-dealokasi \*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan alamat \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen berdasarkan alamat \*\*\*\*\*\*\*\*\*/  void insertFirst(list \*L, address P);  /\* I.S. sembarang, P sudah dialokasikan \*/  /\* F.S. menempatkan elemen beralamat P pada awal list\*/  void insertAfter(list \*L, address P, address Prec);  /\* I.S. sembarang, P dan Prec alamat salah satu elemen list \*/  /\* F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec \*/  void insertLast(list \*L, address P);  /\* I.S. sembarang, P sudah dialokasikan \*/  /\* F.S. menempatkan elemen beralamat P pada akhir list\*/  /\*\*\*\*\*\*\*\*\* penghapusan sebuah elemen \*\*\*\*\*\*\*\*\*/  void delFirst(list \* L, address \* P);  /\* I.S. list tidak kosong \*/  /\* F.S. P adalah alamat dari alamat elemen pertama list sebelum elemen pertama lsit dihapus \*/  /\* elemen pertama list hilang dan list mungkin mjd kosong\*/  /\* first elemen yang baru adalah suksesor first elemen yang lama\*/  void delP (list \*L, infotype X);  /\* I.S. sembarang \*/  /\* F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P dihapus \*/  /\* dan P di-dealokasi, jika tidak ada maka list tetap\*/  /\* list mungkin menjadi kosong karena penghapusan \*/  void delLast(list \* L, address \* P);  /\* I.S. list tidak kosong \*/  /\* F.S. P adalah alamat dari alamat elemen terakhir list sebelum elemen terakhir lsit dihapus \*/  /\* elemen terakhir list hilang dan list mungkin mjd kosong\*/  /\* last elemen yang baru adalah alamat elemen sebelum last elemen yang lama\*/  void delAfter(list \*L, address \* P, address Prec);  /\* I.S. list tidak kosong, Prec alamat salah satu elemen list \*/  /\* F.S. P adalah alamat dari next(Prec), menghapus next(Prec) dari list \*/  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void printfInfo(list L);  /\* I.S. list mungkin kosong \*/  /\* F.S. jka list tidak kosong menampilkan semua info yang ada pada list \*/  int nbList(list L);  /\* mengembalikan jumlah elemen pada list\*/  infotype maks(list L);  /\* mengembalikan nilai elemen list yang paling besar nilainya\*/  address adrMaks(list L);  /\* mengembalikan address P dimana info(P) nilai yang paling besar\*/  infotype min(list L);  /\* mengembalikan nilai elemen list yang paling kecil nilainya\*/  address adrMin(list L);  /\* mengembalikan address P dimana info(P) nilai yang paling kecil\*/  infotype rata(list L);  /\* mengembalikan nilai rata - rata dari semua elemen list\*/  /\*\*\*\*\*\*\*\*\* proses terhadap list \*\*\*\*\*\*\*\*\*/  void delAll(list \*L);  /\* menghapus semua elemen list dan semua elemen di-dealokasi \*/  void inverstList(list \*L);  /\* I.S. sembarang\*/  /\* F.S. elemen - elemen list dibalik \*/  void copyList(list L1, list \*L2);  /\* I.S. L1 sembarang\*/  /\* F.S. L1 = L2 , L1 dan L2 menunjuk pada elemen yang sama\*/  list fCopyList(list L);  /\* mengembalikan list yang merupakan salinan dari L\*/  void koncat (list L1, list L2, list \*L3);  /\* I.S. L1 dan L2 sembarang\*/  /\* F.S. L3 dialokasikan baru, L3 merupakan concatation dari L1 dan L2\*/  void concat1(list \*L1, list \*L2, list \*L3);  /\* I.S. L1 dan L2 sembarang\*/  /\* F.S. L3 tidak perlu dialokasikan baru, L3 merupakan penggabungan L1 dan L2\*/  /\* L1 dan L2 menjadi kosong\*/  void pecahList(list \*L1, list \*L2, list L);  /\* I.S. L mungkin kosong \*/  /\* L1 dan L2 harus dialokasikan baru, \*/  /\* F.S. terbetuk L1 merupakan 1/2 bagian awal dari L, dan L2 merupakan 1/2 bagian akhir dari L\*/  #endif |

Program .2 list.h

|  |
| --- |
| /\*file : list.cpp\*/  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah list kosong \*\*\*\*\*\*\*\*\*/  boolean ListEmpty(list L){  bool x;  if(first(L) ==NULL)  x=false;  else  x=true;  return x;  }  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void CreateList(list \*L){  first(\*L) = NULL;  }  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotype X){  address p;  p = (address) malloc (sizeof(elmlist));  if (p!=NULL){  info(p)=X;  next(p)=NULL:  }  return p;  }  void dealokasi(address P){  free(P);  }  /\*\*\*\*\*\*\*\*\* pencarian sebuah elemen list \*\*\*\*\*\*\*\*\*/  address findElm(list L, infotype X){  address q;  address p;  bool found = false;  q = first(L);  p = NULL;    while (q!=NULL && found==false){  if(info(q)==x){  found = true;  p = q;  }  else{  q = next(q);  }  }  return p;  }  address findBefore(list L, infotype X){  address q;  address p;  bool found = false;  q = first(L);  p = NULL;  while (q!=NULL && found==false){  if(info(q)==x){  found = true;  p = q;  }  else  {  q = next(q);  }  }  return q;  }  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan nilai \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen pada list \*/  void insertNFirst(list \*L, infotype X){  address p;  p = alokasi(X);    if(first(\*L)==NULL){  createList(L);  }  else{  next(p)=first(\*L);  first(\*L)=p;  }  }  void insertNLast(list \*L, infotype X){  address p,q;  p = alokasi(X);  q = first(\*L);  if(isEmpty(\*L) == true){  insertNFirst(&(\*L), X)  }  else{  while(next(q)!=NULL){  q = next(q);  }  next(q)=p;  }  }  /\*\*\*\*\*\*\*\*\* penghapusan elemen pada list \*/  void delNFirst(list \* L, infotype \* X);  /\* I.S. List tidak kosong \*/  /\* F.S. elemen pertama dihapus: nilai info disimpan pada X dan elemen pertama di-dealokasi \*/  void delNLast(list \* L, infotype \* X);  /\* I.S. List tidak kosong \*/  /\* F.S. elemen terakhir dihapus: nilai info disimpan pada X dan elemen terakhir di-dealokasi \*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan alamat \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen berdasarkan alamat \*\*\*\*\*\*\*\*\*/  void insertFirst(list \*L, address P){  next(p) = first(\*L);  first(\*L) = p;  }  void insertAfter(list \*L, address P, address Prec){  q = findElm(&(\*L), info(p))  next(p)=next(q);  next(q)=p;  }  void insertLast(list \*L, address P){  while(next(q)!=NULL){  q = next(q);  }  next(q)=p;  }  /\*\*\*\*\*\*\*\*\* penghapusan sebuah elemen \*\*\*\*\*\*\*\*\*/  void delFirst(list \* L, address \* P){  p = first(\*L);  first(\*L) = next(p);  next(p) = NULL;  dealokasi(p);  }  void delP (list \*L, infotype X){  address P;  address Q;    if (isEmpty(\*L) == 0) {  cout<<"\n\nList masih kosong, tidak ada element yang bisa dihapus";  }  else {  P = findElm(\*L,X);  if (P == NULL) {  cout<<X<<" tidak ditemukan di dalam List";  }  else if (P == first(\*L)){  delFirst(&(\*L));  cout<<X<<" sudah terhapus dari List");  }  else{  Q = findBefore(\*L,x);  delAfter(P,Q);  cout<<X<<" sudah terhapus dari List";  }  }  }  void delLast(list \* L, address \*P){  address q;  \*P= first(\*L);  while(next(p)!=NULL){  \*P = next(\*P);  q=\*P;  }  next(q)=NULL;  dealokasi(\*P)  }  void delAfter(list \*L, address \* P, address Prec){  Prec = findElm(&(\*L), info(\*P))  next(Prec) = next(\*P);  next(\*P) = NULL;  dealokasi(\*P);  }  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void printfInfo(list L){  address Q;  if (isEmpty(L) == false){  cout<< “List masih kosong";  }  else {  cout<<”Element di dalam List :\n";  Q = first(L);  while (Q != NULL){  Cout<<info(Q);  Q = next(Q);  }  }  }  int nbList(list L);  /\* mengembalikan jumlah elemen pada list\*/  infotype maks(list L);  /\* mengembalikan nilai elemen list yang paling besar nilainya\*/  address adrMaks(list L);  /\* mengembalikan address P dimana info(P) nilai yang paling besar\*/  infotype min(list L);  /\* mengembalikan nilai elemen list yang paling kecil nilainya\*/  address adrMin(list L);  /\* mengembalikan address P dimana info(P) nilai yang paling kecil\*/  infotype rata(list L);  /\* mengembalikan nilai rata - rata dari semua elemen list\*/  /\*\*\*\*\*\*\*\*\* proses terhadap list \*\*\*\*\*\*\*\*\*/  void delAll(list \*L);  /\* menghapus semua elemen list dan semua elemen di-dealokasi \*/  void inverstList(list \*L);  /\* I.S. sembarang\*/  /\* F.S. elemen - elemen list dibalik \*/  void copyList(list L1, list \*L2);  /\* I.S. L1 sembarang\*/  /\* F.S. L1 = L2 , L1 dan L2 menunjuk pada elemen yang sama\*/  list fCopyList(list L);  /\* mengembalikan list yang merupakan salinan dari L\*/  void koncat (list L1, list L2, list \*L3);  /\* I.S. L1 dan L2 sembarang\*/  /\* F.S. L3 dialokasikan baru, L3 merupakan concatation dari L1 dan L2\*/  void concat1(list \*L1, list \*L2, list \*L3);  /\* I.S. L1 dan L2 sembarang\*/  /\* F.S. L3 tidak perlu dialokasikan baru, L3 merupakan penggabungan L1 dan L2\*/  /\* L1 dan L2 menjadi kosong\*/  void pecahList(list \*L1, list \*L2, list L);  /\* I.S. L mungkin kosong \*/  /\* L1 dan L2 harus dialokasikan baru, \*/  /\* F.S. terbetuk L1 merupakan 1/2 bagian awal dari L, dan L2 merupakan 1/2 bagian akhir dari L\*/  #endif |

Program .2 list.cpp

## Latihan

1. Buatlah file \*.cpp dari ADT singgle linked list diatas! Untuk memulainya buat terlebih dahulu file \*.h-nya sesuai dengan ADT diatas, kemudian buat file \*.cpp-nya dengan nama yang sama.
2. Untuk mengaplikasikan ADT pada nomor satu, buatlah sebuah program sistem informasi kepegawaian yang mengandung ADT tersebut.

Note : Asumsi info data, tergantung setiap mahasiswa

1. Studi Kasus

Buatlah subuah sistem informasi kepegawaian yang dapat menerima input data pegawai, melihat data pegawai menghapus data pegawai serta mengupdate data pegawai.

Keterangan :

* + - Program di buat dalam 3 file, \*.h , \*.c , \*.c
    - Isi dari data pegawai adalah, nama, nip, jam kerja serta gaji yang dihitung berdasarkkan jam kerja, gaji=10000\*jam kerja
    - Fungsi-funsi utama yang diperlukan antara lain insert data pegawai, view data pegawai, delete data pegawai serta update data pegawai.

Contoh file .h dalam kasus ini.

|  |
| --- |
| #ifndef pegawai\_H  #define pegawai\_H  #include <stdio.h>  #include <conio.h>  #include <stdlib.h>  #include <string.h>  #define nil NULL  #define info(p) (p)->info  #define next(p) (p)->next  #define first(l) ((l).first)  typedef struct{  char nama[20];  char nip[10];  int jkerja;  int gaji;  }infotype;  typedef struct tElmlist \*address;  typedef struct tElmlist{  infotype info;  address next;  }elmlist;  typedef struct{  address first;  }list;  void createEmpty(list \*l);  void insertFirst(list \*l, address p);  void insertLast(list \*l, address p);  void insertAfter(address q, address p);  void deleteFirst(list \*l, address \*p);  void deleteLast(list \*l, address \*p);  void deleteAfter(address q, address \*p);  address searchData(list l, infotype x);  void updateData(list l, infotype x);  void printInfo(list l);  void dealokasi(address p);  address alokasi(infotype x);  #endif |

Program .3 pegawai.h

# DOUBLE LINKED LIST

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat :

1. Praktikan dapat memahami konsep modul linked list.
2. Praktikan dapat mengaplikasikan konsep double linked list dengan menggunakan pointer dan dengan bahasa C++

## Double Linked List

Double Linked List adalah linked list yang masing – masing elemen nya memiliki 2 suksesor, yaitu suksesor yang menunjuk pada elemen sebelumnya (prev) dan suksesor yang menunjuk pada elemen sesudahnya(next).

Gambar berikut menunjukan bentuk Double Linked List dengan elemen kosong:



Gambar ‑Double Linked List dengan Elemen Kosong

Gambar berikut menunjukan bentuk Double Linked List dengan 3 elemen:

Gambar ‑ Double Linked List dengan 3 Elemen

Pada double linked list juga menggunakan dua buah suksesor utama yang terdapat pada list, yaitu first (suksesor yang menunjuk elemen pertama) dan last (susesor yang menunjuk elemen terakhir list).

Komponen-komponen dalam double linked list:

1. First : pointer pada list yang menunjuk pada elemen pertama list.
2. Last : pointer pada list yang menunjuk pada elemen terakhir list.
3. Next : pointer pada elemen sebagai suksesor yang menunjuk pada elemen didepannya.
4. Prev : pointer pada elemen sebagai suksesor yang menunjuk pada elemen dibelakangnya.

Sama dengan singke linked list, pengembangan double linked list bisa dibuat circular. Yaitu list yang elemen awal dan elemen akhirnya bisa saling berhubungan. Pada circular double linked list, suksesor prev pada elemen awal mengacu pada elemen akhir list dan suksesor next pada elemen akhir mengacu pada elemen awal list.



Gambar ‑ Circular Double Linked List

Contoh pendeklarasian struktur data untuk double linked list:

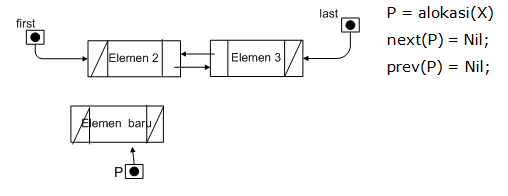
|  |
| --- |
| /\*file : doublelist.h\*/  #ifndef doublelist\_H  #define doublelist\_H  #include "boolean.h"  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define prev(P) (P)->prev  #define first(L) ((L).first)  #define last(L) ((L).last)  /\*deklarasi record dan struktur data double linked list\*/  typedef int infotype;  typedef struct tElmlist \*address;  typedef struct tElmlist {  infotype info;  address next;  address prev;  } elmlist;  /\* definisi list : \*/  /\* list kosong jika Fisrt(L)=Nil \*/  typedef struct {  address first;  address last;  } list;  #endif |

Program ‑ Deklarasi Struktur Data Double Linked List

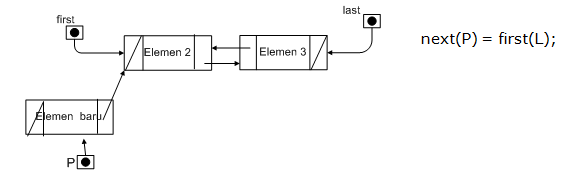
### Insert

#### Insert First

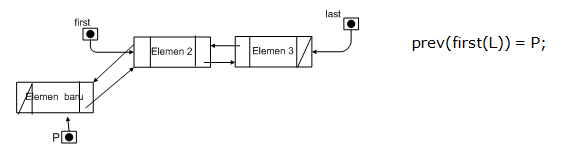
Langkah-langkah dalam proses insert first:



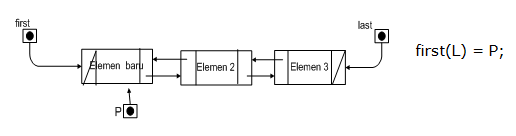
Gambar 3‑4 Double Linked List Insert First I



Gambar ‑ Double Linked List Insert First 2



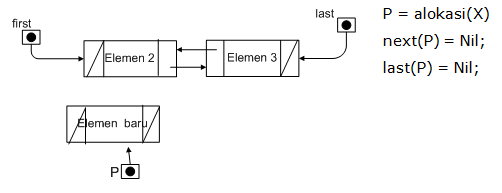
Gambar ‑ Double Linked List Insert First 3



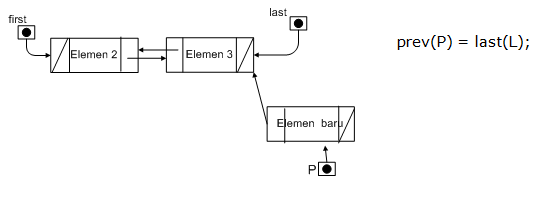
Gambar ‑ Double Linked List Insert First 4

#### Insert Last

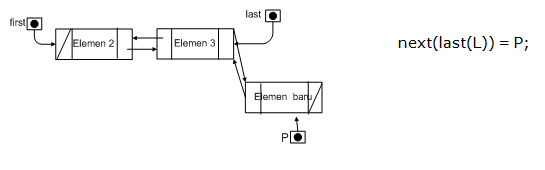
Langkah-langkah dalam proses insert last:



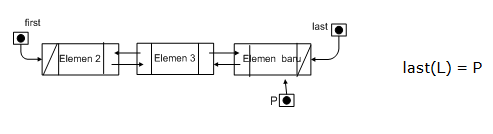
Gambar 3‑8 Double Linked List Insert Last 1



Gambar ‑ Double Linked List Insert Last 2



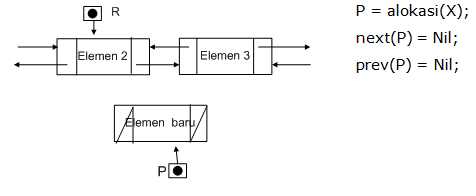
Gambar ‑Double Linked List Insert Last 3



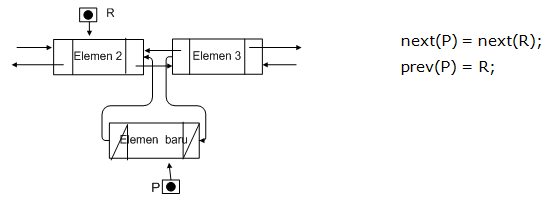
Gambar ‑ Double Linked List Insert Last 4

#### Insert After

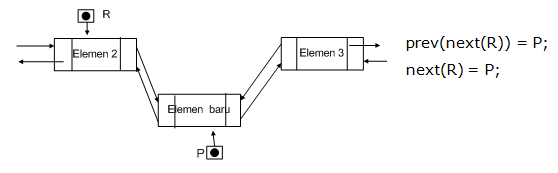
Langkah-langkah dalam proses insert after:



Gambar ‑ Double Linked List Insert After 1



Gambar ‑ Double Linked List Insert After 2



Gambar ‑ Double Linked List Insert After 3

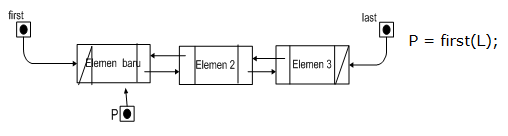
#### Insert Before

Diatas hanya dijelaskan tentang insert after. Insert before hanya kebalikan dari insert after. Perbedaan Insert After dan Insert Before terletak pada pencarian elemennya.

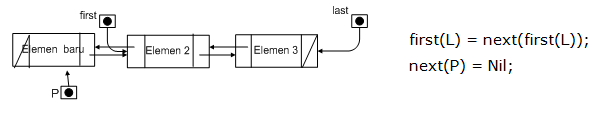
### Delete

#### Delete First

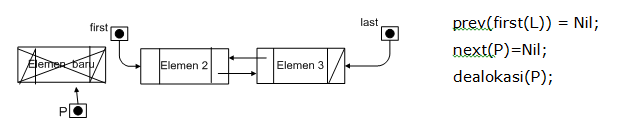
Langkah-langkah dalam proses delete first:



Gambar ‑ Double Linked List Delete First 1



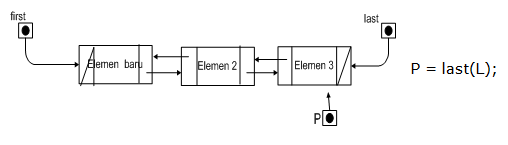
Gambar 3‑Double Linked List Delete First 2



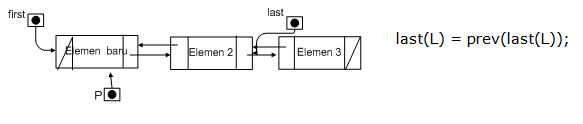
Gambar ‑ Double Linked List Delete First 3

#### Delete Last

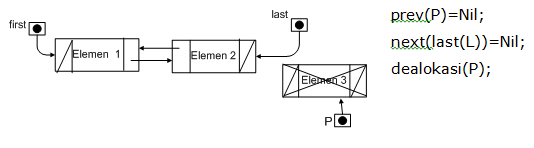
Langkah-langkah dalam proses delete last:

:

Gambar ‑ Double Linked List Delete Last 1



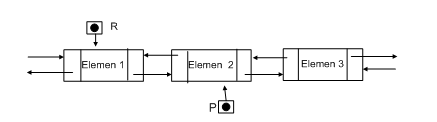
Gambar ‑ Double Linked List Delete Last 2



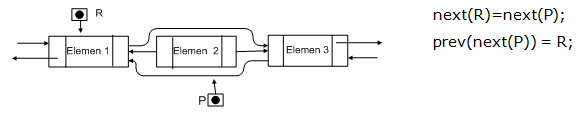
Gambar ‑ Double Linked List Delete Last 3

#### Delete After

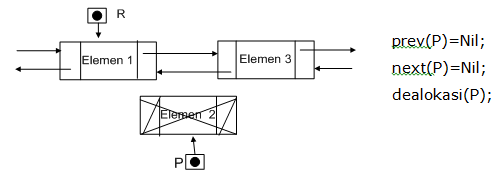
Langkah-langkah dalam proses delete after:



Gambar ‑ Double Linked List Delete After 1



Gambar ‑ Double Linked List Delete After 2



Gambar ‑ Double Linked List Delete After 3

#### Delete Before

Diatas hanya dijelaskan tentang delete after. Delete before hanya kebalikan dari delete after. Perbedaan Delete After dan Delete Before terletak pada pencarian elemennya.

#### Update, View, dan Searching

Proses pencarian, update data dan view data pada dasarnya sama dengan proses pada single linked list. Hanya saja pada double linked list lebih mudah dalam melakukan proses akses elemen, karena bisa melakukan iterasi maju dan mundur.

Seperti halnya single linked list, double linked list juga mempunyai ADT yang pada dasarnya sama dengan ADT yang ada pada single linked list.

|  |
| --- |
| /\*file : doublelist.h\*/  /\* contoh ADT list berkait dengan representasi fisik pointer\*/  /\* representasi address dengan pointer\*/  /\* info tipe adalah integer \*/  #ifndef doublelist\_H  #define doublelist\_H  #include "boolean.h"  #include <stdio.h>  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define prev(P) (P)->prev  #define first(L) ((L).first)  #define last(L) ((L).last)  typedef int infotype;  typedef struct tElmlist \*address;  /\* pendefinisian tipe data bentukan elemen list\*/  /\* dengan dua suksesor, yaitu next dan prev\*/  typedef struct tElmlist {  infotype info;  address prev;  address next;  }elmlist;  /\* definisi double linked list : \*/  /\* list kosong jika Fisrt(L)=Nil \*/  /\* setiap elemen address P dapat diacu info(P) atau next(P) \*/  /\* element terakhir adalah last \*/  /\* nama tipe list yang dipakai adalah 'list', sama dengan pada single list\*/  typedef struct {  address first, last;  }list;  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah list kosong \*\*\*\*\*\*\*\*\*/  boolean ListEmpty(list L);  /\* mengembalikan nilai true jika list kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void CreateList(list \*L);  /\* I.S. sembarang \*/  /\* F.S. terbentuk list kosong \*/  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotype X);  /\* mengirimkan address dari alokasi sebuah elemen \*/  /\* jika aloksi berhasil maka nilai address tidak Nil dan jika gagal nilai address Nil \*/  void Dealokasi(address P);  /\* I.S. P terdefinisi \*/  /\* F.S. memori yang digunakan P dikembalikan ke sistem \*/  /\*\*\*\*\*\*\*\*\* pencarian sebuah elemen list \*\*\*\*\*\*\*\*\*/  address findElm(list L, infotype X);  /\* mencari apakah ada elemen list dengan info(P) = X\*/  /\* jika ada mengembalikan address elemen tsb, dan Nil jika sebaliknya\*/  boolean fFindEml(list L, address P);  /\* mencari apakah ada elemen list dengan alamat P\*/  /\* mengembalikan true jika ada dan false jika tidak ada \*/  address findBefore(list L, infotype X, address Prec);  /\* mengembalikan address elemen sebelum Prec dimana info(Prec) = X\*/  /\* jika elemen dengan info = X tidak ada maka mengembalikan nilai Nil\*/  /\* jika Prec berada pada awal list maka mengembalikan nilai Nil\*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan nilai \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen pada list \*/  void insertNFirst(list \*L, infotype X);  /\* I.S. List mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada awal list dengan nilai X \*/  void insertNLast(list \*L, infotype X);  /\* I.S. List mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada akhir list dengan nilai X \*/  /\*\*\*\*\*\*\*\*\* penghapusan elemen pada list \*/  void delNFirst(list \* L, infotype \* X);  /\* I.S. List tidak kosong \*/  /\* F.S. elemen pertama dihapus: nilai info disimpan pada X dan elemen pertama di-dealokasi \*/  void delNLast(list \* L, infotype \* X);  /\* I.S. List tidak kosong \*/  /\* F.S. elemen terakhir dihapus: nilai info disimpan pada X dan elemen terakhir di-dealokasi \*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan alamat \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen berdasarkan alamat \*\*\*\*\*\*\*\*\*/  void insertFirst(list \*L, address P);  /\* I.S. sembarang, P sudah dialokasikan \*/  /\* F.S. menempatkan elemen beralamat P pada awal list\*/  void insertAfter(list \*L, address P, address Prec);  /\* I.S. sembarang, P dan Prec alamat salah satu elemen list \*/  /\* F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec \*/  void insertLast(list \*L, address P);  /\* I.S. sembarang, P sudah dialokasikan \*/  /\* F.S. menempatkan elemen beralamat P pada akhir list\*/  /\*\*\*\*\*\*\*\*\* penghapusan sebuah elemen \*\*\*\*\*\*\*\*\*/  void delFirst(list \* L);  /\* I.S. list tidak kosong \*/  /\* F.S. P adalah alamat dari alamat elemen pertama list sebelum elemen pertama lsit dihapus \*/  /\* elemen pertama list hilang dan list mungkin mjd kosong\*/  /\* first elemen yang baru adalah suksesor first elemen yang lama\*/  void delP (list \*L, infotype X);  /\* I.S. sembarang \*/  /\* F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P dihapus \*/  /\* dan P di-dealokasi, jika tidak ada maka list tetap\*/  /\* list mungkin menjadi kosong karena penghapusan \*/  void delLast(list \* L);  /\* I.S. list tidak kosong \*/  /\* F.S. P adalah alamat dari alamat elemen terakhir list sebelum elemen terakhir lsit dihapus \*/  /\* elemen terakhir list hilang dan list mungkin mjd kosong\*/  /\* last elemen yang baru adalah alamat elemen sebelum last elemen yang lama\*/  void delAfter(list \*L, address \* P, address Prec);  /\* I.S. list tidak kosong, Prec alamat salah satu elemen list \*/  /\* F.S. P adalah alamat dari next(Prec), menghapus next(Prec) dari list \*/  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void printfInfo(list L);  /\* I.S. list mungkin kosong \*/  /\* F.S. jka list tidak kosong menampilkan semua info yang ada pada list \*/  int nbList(list L);  /\* mengembalikan jumlah elemen pada list\*/  infotype maks(list L);  /\* mengembalikan nilai elemen list yang paling besar nilainya\*/  address adrMaks(list L);  /\* mengembalikan address P dimana info(P) nilai yang paling besar\*/  infotype min(list L);  /\* mengembalikan nilai elemen list yang paling kecil nilainya\*/  address adrMin(list L);  /\* mengembalikan address P dimana info(P) nilai yang paling kecil\*/  infotype rata(list L);  /\* mengembalikan nilai rata - rata dari semua elemen list\*/  /\*\*\*\*\*\*\*\*\* proses terhadap list \*\*\*\*\*\*\*\*\*/  void delAll(list \*L);  /\* menghapus semua elemen list dan semua elemen di-dealokasi \*/  void copyList(list L1, list \*L2);  /\* I.S. L1 sembarang\*/  /\* F.S. L1 = L2 , L1 dan L2 menunjuk pada elemen yang sama\*/  list fCopyList(list L);  /\* mengembalikan list yang merupakan salinan dari L\*/  void cpAlokList(list lin, list \* lout);  /\* I.S. Lin sembarang\*/  /\* F.S. lout dialokasikan baru, jika alokasi berhasil maka mengkopikn semua Lin ke lout\*/  void koncat (list L1, list L2, list \*L3);  /\* I.S. L1 dan L2 sembarang\*/  /\* F.S. L3 dialokasikan baru, L3 merupakan concatation dari L1 dan L2\*/  void concat1(list \*L1, list \*L2, list \*L3);  /\* I.S. L1 dan L2 sembarang\*/  /\* F.S. L3 tidak perlu dialokasikan baru, L3 merupakan penggabungan L1 dan L2\*/  /\* L1 dan L2 menjadi kosong\*/  void pecahList(list \*L1, list \*L2, list L);  /\* I.S. L mungkin kosong \*/  /\* L1 dan L2 harus dialokasikan baru, \*/  /\* F.S. terbetuk L1 merupakan 1/2 bagian awal dari L, dan L2 merupakan 1/2 bagian akhir dari L\*/  #endif |

Program .2 doublelist.h

## Latihan

1. Cobalah buat ADT (file \*.c) untuk double linked list dengan spesifikasi sesuai dengan file \*.h diatas!
2. implementasikan ADT yang dibentuk untuk membuat progam yang menangani Data Pegawai seperti pada latihan soal modul 3 Single List. Asumsikan parameter dan inputan sesuai bayangan anda.
3. Cobalah untuk menampilkan doble list yang telah dibuat pada nomer 2 secara terbalik.
4. Studi Kasus

Buatlah sebuah sistem informasi kepegawaian yang dapat menerima input data pegawai, melihat data pegawai menghapus data pegawai serta mengupdate data pegawai.

Keterangan :

* + - Program di buat dalam 3 file, \*.h , \*.c , \*.c
    - Isi dari data pegawai adalah, nama, nip, jam kerja serta gaji yang dihitung berdasarkkan jam kerja, gaji=10000\*jam kerja
    - Fungsi-funsi utama yang diperlukan antara lain insert data pegawai, view data pegawai, delete data pegawai serta update data pegawai.

Berikut file .h untuk membantu pengerjaan, boleh juga menggunakan asumsi anda sendiri.

|  |
| --- |
| #ifndef DOUBLE\_H  #define DOUBLE\_H  #include <stdio.h>  #include <conio.h>  #include <stdlib.h>  #include <string.h>  #define nil NULL  #define info(p) (p)->info  #define next(p) (p)->next  #define prev(p) (p)->prev  #define first(l) ((l).first)  #define last(l) ((l).last)  typedef struct{  char nama[20];  char nip[10];  int jkerja;  int gaji;  }infotype;  typedef struct tElmlist \*address;  typedef struct tElmlist{  infotype info;  address next,prev;  }elmlist;  typedef struct{  address first,last;  }list;  boolean ListEmpty(list L);  void CreateList(list \*L);  address alokasi(infotype X);  void Dealokasi(address P);  address findElm(list L, char x[10]);  void insertFirst(list \*L, address P);  void insertLast(list \*L, address P);  void delP(list \*L, char X[10]);  void delAfter(list \*L, address \* P, address Prec);  void printfInfo(list L);  #endif |

Program .3 double.h

# MULTI LINKED LIST

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat :

1. Mengetahui tentang Multi Linked List.
2. Memahami penggunaan Multi Linked List.
3. Menerapkan penggunaan Multi Linked List dalam beberapa studi kasus.

## Multi Linked List

Multi List merupakan sekumpulan list yang berbeda yang memiliki suatu keterhubungan satu sama lain. Tiap elemen dalam multi link list dapat membentuk list sendiri.Biasanya ada yang bersifat sebagai list induk dan list anak .

Contoh Multi Linked List dapat dilihat pada gambar berikut.



Gambar ‑ Muti Linked List

Jadi, dari implementasi diatas akan terdapat dua buah list, list pegawai dan list anak.Dimana untuk list pegawai menunjuk satu buah list anak.Disini list induknya adalah list pegawai dan list anaknya adalah list anak.

### Insert

#### Insert Anak

Dalam penginsertan elemen anak harus diketahui dulu elemen induknya.

Berikut ini ilustrasi insert anak dengan konsep insert last:



Gambar ‑ Multi Linked List Insert Anak 1



Gambar ‑ Multi Linked List Insert Anak 2

#### Insert Induk

Untuk insert elemen induk sama dengan konsep insert pada single, double dan circular linked list.

### Delete

#### Delete Anak

Sama dengan insert anak untuk delete anak maka harus diketahui dulu induknya.Berikut ini Gambar ilustrasinya untuk delete last pada induk peg 1:



Gambar ‑ Multi Linked List Delete Anak 1



Gambar ‑ Multi Linked List Delete Anak 2

#### Delete Induk

Untuk delete elemen induk maka saat di hapus maka seluruh anak dengan induk tersebut juga harus dihapus.Berikut ini gambar ilustrasinya:





Gambar ‑ Multi Linked List Delete Induk 1

Berikut ini primitif untuk ADT Multi List.

|  |
| --- |
| /\* file : multilist.h\*/  /\* contoh ADT list berkait dengan representasi fisik pointer\*/  /\* representasi address dengan pointer\*/  /\* info tipe adalah integer \*/  #ifndef multilist\_H  #define multilist\_H  #include "boolean.h"  #include <stdio.h>  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define first(L) ((L).first)  #define last(L) ((L).last)  #define first(Lanak) ((Lanak).first)  #define last(Lanak) ((Lanak).last)  typedef int infotypeanak;  typedef int infotypeinduk;  typedef struct element\_list\_induk \*address;  typedef struct element\_list\_anak \*address\_anak;  /\* definisi list : \*/  /\* list kosong jika Fisrt(L)=Nil \*/  /\* setiap elemen address P dapat diacu info(P) atau next(P) \*/  /\* element terakhir list jika adrressnya last, maka next(last) = Nil\*/  typedef struct element\_list\_anak{  /\* struct ini untuk menyimpan elemen anak dan pointer penunjuk elemen tetangganya \*/  infotypeanak info;  address\_anak nextAnak;  address\_anak prevAnak;  };  typedef struct listanak{  /\* struct ini digunakan untuk menyimpan list anak itu sendiri \*/  address\_anak first;  address\_anak last;  };  struct element\_list\_induk {  /\* struct ini untuk menyimpan elemen pegawai dan pointer penunjuk elemen tetangganya \*/  infotypeinduk info;  address\_anak firstAnak;  address nextInduk;  address prevInduk;  };  struct listinduk{  /\* struct ini digunakan untuk menyimpan list pegawai itu sendiri \*/  address first;  address last;  };  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah list kosong \*\*\*\*\*\*\*\*\*/  boolean ListEmpty(listinduk L);  /\* mengembalikan nilai true jika list induk kosong \*/  boolean ListEmptyAnak(listanak Lanak);  /\* mengembalikan nilai true jika list anak kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void CreateList(list \*L);  /\* I.S. sembarang \*/  /\* F.S. terbentuk list induk kosong \*/  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotypeinduk X);  /\* mengirimkan address dari alokasi sebuah elemen induk \*/  /\* jika aloksi berhasil maka nilai address tidak Nil dan jika gagal  nilai address Nil \*/  address\_anak alokasiAnak(infotypeanak X);  /\* mengirimkan address dari alokasi sebuah elemen anak \*/  /\* jika aloksi berhasil maka nilai address\_anak tidak Nil dan jika  gagal nilai address\_anak Nil \*/  void Dealokasi(address P);  /\* I.S. P terdefinisi \*/  /\* F.S. memori yang digunakan P dikembalikan ke sistem \*/  void DealokasiAnak(address\_anak P);  /\* I.S. P terdefinisi \*/  /\* F.S. memori yang digunakan P dikembalikan ke sistem \*/  /\*\*\*\*\*\*\*\*\* pencarian sebuah elemen list \*\*\*\*\*\*\*\*\*/  address findElm(listinduk L, infotypeinduk X);  /\* mencari apakah ada elemen list dengan info(P) = X\*/  /\* jika ada mengembalikan address elemen tsb, dan Nil jika sebaliknya \*/  address findBefore(listinduk L, infotypeinduk X, address Prec);  /\* mengembalikan address elemen sebelum Prec dimana info(Prec) = X\*/  /\* jika elemen dengan info = X tidak ada maka mengembalikan nilai Nil\*/  /\* jika Prec berada pada awal list maka mengembalikan nilai Nil \*/  address\_anak findElmAnak(listanak Lanak, infotypeanak X);  /\* mencari apakah ada elemen list anak dengan info(P) = X\*/  /\* jika ada mengembalikan address\_anak elemen tsb, dan Nil jika  sebaliknya \*/  address\_anak findBeforeAnak(listanak Lanak, infotypeanak X,  address\_anak Prec);  /\* mengembalikan address elemen sebelum Prec dimana info(Prec) = X\*/  /\* jika elemen dengan info = X tidak ada maka mengembalikan nilai Nil\*/  /\* jika Prec berada pada awal list anak maka mengembalikan nilai Nil\*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan nilai \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen pada list \*/  void insertFirst(listinduk \*L, infotypeinduk X);  /\* I.S. List induk mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada  awal list dengan nilai X \*/  void insertLast(listinduk \*L, infotypeinduk X);  /\* I.S. List induk mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada  akhir list dengan nilai X \*/  void insertFirstAnak(listinduk \*L, address p, infotypeanak X);  /\* I.S. List anak mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada  awal list dengan nilai X \*/  void insertLastAnak(listinduk \*L, address p, infotypeanak X);  /\* I.S. List anak mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada  akhir list dengan nilai X \*/  /\*\*\*\*\*\*\*\*\* penghapusan elemen pada list \*/  void delFirst(listinduk \*L, infotypeinduk X);  /\* I.S. List induk tidak kosong \*/  /\* F.S. elemen pertama dihapus: nilai info disimpan pada X dan  elemen pertama di-dealokasi \*/  void delLast(listinduk \*L, infotypeinduk X);  /\* I.S. List induk tidak kosong \*/  /\* F.S. elemen terakhir dihapus: nilai info disimpan pada X dan  elemen terakhir di-dealokasi \*/  void delFirstAnak(listinduk \*L, address p, infotypeanak X);  /\* I.S. List anak tidak kosong \*/  /\* F.S. elemen pertama dihapus: nilai info disimpan pada X dan  elemen pertama di-dealokasi \*/  void delLastAnak(listinduk \*L, address p, infotypeanak X);  /\* I.S. List anak tidak kosong \*/  /\* F.S. elemen terakhir dihapus: nilai info disimpan pada X dan  elemen terakhir di-dealokasi \*/  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void printfInfo(listinduk L);  /\* I.S. list mungkin kosong \*/  /\* F.S. jika list tidak kosong menampilkan semua info yang ada pada  list \*/  int countList(listinduk L);  /\* mengembalikan jumlah elemen pada list\*/  address adrMaks(listinduk L);  /\* mengembalikan address P dimana info(P) nilai yang paling besar\*/  address adrMin(listinduk L);  /\* mengembalikan address P dimana info(P) nilai yang paling kecil\*/  infotypeinduk rata(listinduk L);  /\* mengembalikan nilai rata - rata dari semua elemen list\*/  void printfInfo(listinduk L);  /\* I.S. list mungkin kosong \*/  /\* F.S. jika list tidak kosong menampilkan semua info yang ada pada  list \*/  int countListAnak(listinduk L);  /\* mengembalikan jumlah elemen anak pada list\*/  infotypeanak maksAnak(listinduk L);  /\* mengembalikan nilai elemen list yang paling besar nilainya\*/  infotypeanak minAnak(listindukL);  /\* mengembalikan nilai elemen list yang paling kecil nilainya\*/  infotypeanak rataAnak(listinduk L);  /\* mengembalikan nilai rata - rata dari semua elemen list\*/  /\*\*\*\*\*\*\*\*\* proses terhadap list \*\*\*\*\*\*\*\*\*/  void delAll(listinduk \*L);  /\* menghapus semua elemen list induk dan semua elemen di-dealokasi \*/  void inverstList(listinduk \*L);  /\* I.S. sembarang\*/  /\* F.S. elemen - elemen list dibalik \*/  listinduk CopyList(listinduk L);  /\* mengembalikan list yang merupakan salinan dari L\*/  void delAllanak(listinduk \*L, address p);  /\* menghapus semua elemen anak pada address p dan semua elemen di-dealokasi \*/  void inverstListAnak(listinduk \*L, address p);  /\* I.S. sembarang\*/  /\* F.S. elemen - elemen list dibalik \*/  void copyListAnak(listinduk L, listanak \*Lanak);  /\* I.S. L sembarang\*/  /\* F.S. Lanak memiliki semua elemen anak pada L \*/  #endif |

Program ‑ multilist.h

## Latihan

* 1. Buatlah implementasi dari list.h diatas dalam bentuk file list.c!
  2. Buatlah sebuah program yang menggunakan ADT list diatas dengan nama driver.c!
  3. Studi Kasus

Buatlah sebuah program dalam multi link list yang di implementasikan dalam sebuah kasus dimana ada list truk berupa double link list, yang masing-masing truk memunyai list yaitu list barang yang berupa single link list.Di mana barang pada setiap truk mempunyai kode barang yang berbeda.

* + - Program di buat dalam 3 file, \*.h , \*.c , \*.c
    - Pada struktur datanya terdapat stuktur data element barang element truk dan list truk dimana pada struktur element truk terdapat sebuah alamat dengan tipe addressbarang sebagai first dari sebuah list barang.

Berikut file .h untuk membantu pembuatan program. Boleh juga berbeda.

|  |
| --- |
| #ifndef adtmulti\_H  #define adtmulti\_H  #include...........  #include...........  #include...........  typedef struct truk{  char NoPolisi[10];  }InfoTruk;  typedef struct barang {  char KodeBarang[10];  char NamaBarang[10];  int BeratBarang;  }InfoBarang;  typedef struct {  ........... ..........  ........... ..........  };  typedef struct{  ........... ..........  ........... ..........  };  typedef struct ListTruk{  AddressTruk first;  AddressTruk last;  }List;  AddressTruk alokasiTruk (InfoTruk x);  AddressBarang alokasiBarang(InfoBarang x);  void CreateList(list \*L);  void InsertTruk(List \*L,AddressTruk P);  void InsertElementBarang(AddressTruk \*P,AddressBarang Q);  void DeleteElementTruk(List \*L, InfoTruk x);  void DeleteElementBarang(List \*L,InfoBarang x);  AddressTruk FindElemtTruk(List \*L, InfoTruk x);  AddressBarang FindElementBarang(List \*L, InfoBarang x);  AddressTruk FindElementTrukBarang(List \*L, InfoBarang x);  void ViewBarang(List \*L,AddressTruk P);  void View(List \*L);  #endif |

Program ‑ adtmulti.h

# STACK (TUMPUKAN)

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Memahami konsep stack.
2. Mengaplikasikan stack dengan menggunakan pointer.

## Pengertian

Stack merupakan salah satu bentuk struktur data dimana prinsip operasi yang digunakan seperti tumpukan. Seperti halnya tumpukan, elemen yang bisa diambil terlebih dahulu adalah elemen yang paling atas, atau elemen yang pertama kali masuk, prinsip ini biasa disebut LIFO (Last In First Out).

## Komponen-Komponen dalam Stack

Komponen – komponen dalam stack pada dasarnya sama dengan komponen pada single linked list. Hanya saja akses pada stack hanya bisa dilakukan pada awal stack saja.



Gambar ‑Stack dengan 3 Elemen

Seperti terlihat pada gambar diatas bentuk stack mirip seperti list linier, yang terdiri dari elemen – elemen yang saling terkait. Komponen utama dalam stack yang berfungsi untuk mengakses data dalam stack adalah elemen paling awal saja yang disebut “Top”.

Pendeklarasian tipedata stack:

|  |
| --- |
| #ifndef stack\_H  #define stack\_H  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define Top(S) ((S).Top)  typedef int infotype; /\* tipe data dalam stack\*/  typedef struct tElm \*address; /\* tipe data pointer untuk elemen stack\*/  typedef struct tElm {  infotype info;  address next;  }elmStack; /\* tipe data elemen stack \*/  /\* pendeklarasian tipe data stack\*/  typedef struct {  address Top;  } stack;  #endif |

Program ‑Deklarasi Struktur Data Stack

Keterangan:

1. Dalam stack hanya terdapat TOP.
2. Tipe address adalah tipe elemen stack yang sama dengan elemen dalam list lainnya.

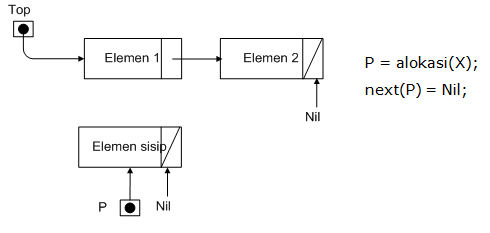
## Operasi-Operasi dalam Stack

Dalam stack ada dua operasi utama, yaitu operasi penyisipan(Push) dan operasi pengambilan (Pop).

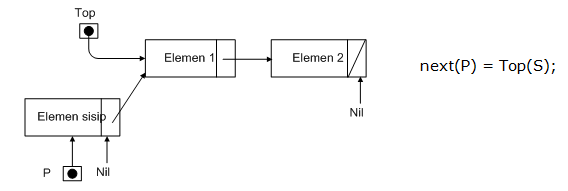
### Push

Adalah operasi menyisipkan elemen pada tumpukan data. Fungsi ini sama dengan fungsi insert first pada list biasa.

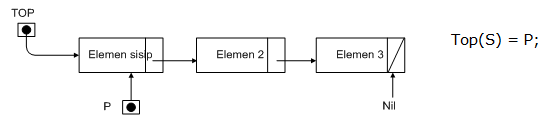
Langkah – langkah dalam proses Push:



Gambar ‑ Stack Push 1



Gambar ‑ Stack Push 2



Gambar ‑ Stack Push 4

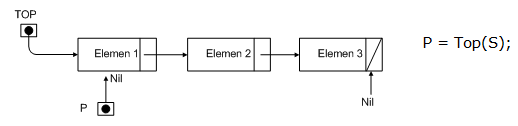
### Pop

Adalah operasi pengambilan data dalam list. Operasi ini mirip dengan operasi delete first dalam list linear, karena elemen yang paling pertama kali diakses adalah elemen paling atas atau elemen paling awal saja.

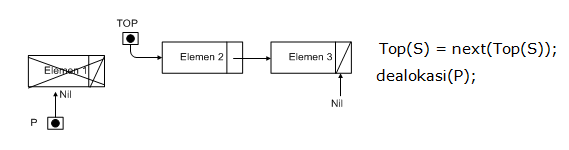
Langkah – langkah dalam proses Pop:



Gambar ‑ Stack Pop 1



Gambar ‑ Stack Pop 2



Gambar ‑ Stack Pop 3

## Primitif-Primitif dalam Stack

Primitif-primitif dalam stack pada dasarnya sama dengan primitive-primitif pada list lainnya. Malahan primitif dalam stack lebih sedikit, karena dalam stack hanya melakukan operasi-operasi terhadap elemen paling atas.

Primitif -primitif dalam stack :

1. createStack().
2. isEmpty().
3. alokasi().
4. dealokasi().
5. Fungsi – fungsi pencarian.
6. Dan fungsi – fungsi primitif lainnya.

Seperti halnya pada model list yang lain, primitif-primitifnya tersimpan pada file \*.c dan file \*.h.

## Stack (Representasi Tabel)

Pada prinsipnya representasi menggunakan table sama halnya dengan menggunakan pointer. Perbedaannya terletak pada pendeklarasian strukturdatanya, menggunakan array berindeks dan jumlah tumpukan yang terbatas.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Top |  |  |  |  |  |  |  |  |  | Imax |
| |  | | --- | |  | |  |  |  |  |  |  |  |  |  | |  | | --- | |  | |
|  |  |  |  |  |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Gambar di atas menunjukan stack maksimum terdapat pada indeks Imax=10, sedangkan Stack masih kosong karena Top = 0.

### Operasi-operasi Dalam Stack

Operasi-Operasi dalam stack representasi tabel pada dasarnya sama dengan representasi pointer, yaitu PUSH dan POP.

### Push

Push merupakan operasi penyisipan data ke dalam stack, penyisipan dilakukan dengan menggeser indeks dari TOP ke indeks berikutnya. Perhatikan contoh dibawah ini :

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Top |  |  |  |  |  |  |  |  |  | Imax |
| |  | | --- | |  | |  |  |  |  |  |  |  |  |  | |  | | --- | |  | |
|  |  |  |  |  |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

**Top=Top+1**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Top |  |  |  |  |  |  |  |  | Imax |
|  | |  | | --- | |  | |  |  |  |  |  |  |  |  | |  | | --- | |  | |
|  | X |  |  |  |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

### Pop

Pop merupakan operasi pengambilan datadi posisi indeks TOP berada dalam sebuah stack. Setelah data diambil, indeks TOP akan bergeser ke indeks sebelum TOP tanpa menghilangkan info dari indeks TOP sebelumnya. Perhatikan contoh dibawah ini:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Top |  |  |  |  |  | Imax |
|  |  |  |  | |  | | --- | |  | |  |  |  |  |  | |  | | --- | |  | |
|  | X | X | X | X |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Temp = S.Info[TOP]

TOP=TOP-1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Top |  |  |  |  |  |  | Imax |
|  |  |  | |  | | --- | |  | |  |  |  |  |  |  | |  | | --- | |  | |
|  | X | X | X | X |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

### Primitif-primitif Dalam Stack

Primitif-primitif pada stack representasi tabel pada dasarnya sama dengan representasi pointer., Perbedaanya hanya pada Management memori, pada representasi tabel tidak diperlukan Management memory. antara lain sebagai berikut,

1. createStack().
2. isEmpty().
3. Fungsi – fungsi pencarian.
4. Dan fungsi – fungsi primitif lainnya.

Seperti halnya pada model list yang lain, primitif-primitifnya tersimpan pada file \*.c dan file \*.h.

Untuk lebih memahami struktur data dari stack representasi tabel, berikut ini contoh ADT stack representasi tabel dalam file \*.h:

|  |
| --- |
| /\*file : stack.h\*/  /\* contoh ADT stack dengan representasi tabel\*/  #ifndef STACK\_H\_INCLUDED  #define STACK\_H\_INCLUDED  #include <stdio.h>  #include <conio.h>  typedef struct {  char nim[20];  char nama[20];  } infotype;  typedef struct {  infotype info[10];  int Top;  } Stack;  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah Stack kosong \*\*\*\*\*\*\*\*\*/  int isEmpty(stack S);  /\* mengembalikan nilai 0 jika stack kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan Stack \*\*\*\*\*\*\*\*\*/  void createStack(Stack \*S);  /\* I.S. sembarang \*/  /\* F.S. terbentuk stack dengan TOP=0 \*/  /\*\*\*\*\*\*\*\*\* penambahan elemen pada Stack \*/  void push(Stack \*S, infotype X);  /\* I.S. stack mungkin kosong \*/  /\* F.S. menambahkan elemen pada stack dengan nilai X, TOP=TOP+1 \*/  /\*\*\*\*\*\*\*\*\* penghapusan elemen pada list \*/  void pop(Stack \*S, Infotype \*X);  /\* I.S. stack tidak kosong \*/  /\* F.S. nilai info pada indeks TOP disimpan pada Xkemudian TOP=TOP-1 \*/  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void viewStack(Stack S);  /\* I.S. stack mungkin kosong \*/  /\* F.S. jka stack tidak kosong menampilkan semua info yang ada pada stack \*/  #endif  #endif // STACK\_H\_INCLUDED |

Program ‑ Stack Representasi Tabel

## Latihan

Contoh ADT stack representasi pointer dalam file \*.h:

|  |
| --- |
| /\*file : stack.h\*/  /\* contoh ADT stack dengan representasi fisik pointer\*/  /\* representasi address dengan pointer\*/  /\* info tipe adalah integer \*/  #ifndef stack\_H  #define stack\_H  #include "boolean.h"  #include <stdio.h>  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define Top(S) ((S).Top)  typedef int infotype;  typedef struct tElm \*address;  typedef struct tElm {  infotype info;  address next;  }elmStack;  /\*definisi stack\*/  /\* setiap elemen address P dapat diacu info(P) atau next(P) \*/  typedef struct {  address Top;  }stack;  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah list kosong \*\*\*\*\*\*\*\*\*/  boolean isEmpty(stack S);  /\* mengembalikan nilai true jika list kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void CreateStack(stack \*S);  /\* I.S. sembarang \*/  /\* F.S. terbentuk list kosong \*/  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotype X);  /\* mengirimkan address dari alokasi sebuah elemen \*/  /\* jika aloksi berhasil maka nilai address tidak Nil dan jika gagal nilai address Nil \*/  void Dealokasi(address P);  /\* I.S. P terdefinisi \*/  /\* F.S. memori yang digunakan P dikembalikan ke sistem \*/  /\*\*\*\*\*\*\*\*\* pencarian sebuah elemen list \*\*\*\*\*\*\*\*\*/  boolean fFindEml(stack S, address P);  /\* mencari apakah ada elemen stack dengan alamat P\*/  /\* mengembalikan true jika ada dan false jika tidak ada \*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan nilai \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen pada list \*/  void pushN(stack \*S, infotype X);  /\* I.S. stack mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada Awal stack dengan nilai X \*/  /\*\*\*\*\*\*\*\*\* penghapusan elemen pada list \*/  void popN(stack \* S, infotype \* X);  /\* I.S. stack tidak kosong \*/  /\* F.S. elemen pertama dihapus: nilai info disimpan pada X dan elemen pertama di-dealokasi \*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan alamat \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen berdasarkan alamat \*\*\*\*\*\*\*\*\*/  void push(stack \*S, address P);  /\* I.S. sembarang, P sudah dialokasikan \*/  /\* F.S. menempatkan elemen beralamat P pada awal stack\*/  /\*\*\*\*\*\*\*\*\* penghapusan sebuah elemen \*\*\*\*\*\*\*\*\*/  void pop(stack \* S, address \* P);  /\* I.S. stack tidak kosong \*/  /\* F.S. P adalah alamat dari alamat elemen pertama stack sebelum elemen pertama stack dihapus \*/  /\* elemen pertama stack hilang dan stack mungkin mjd kosong\*/  /\* top elemen yang baru adalah suksesor top elemen yang lama\*/  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void printfInfo(stack S);  /\* I.S. stack mungkin kosong \*/  /\* F.S. jka stack tidak kosong menampilkan semua info yang ada pada stack \*/  int nbList(stack S);  /\* mengembalikan jumlah elemen pada stack\*/  infotype maks(stack S);  /\* mengembalikan nilai elemen stack yang paling besar nilainya\*/  infotype min(stack S);  /\* mengembalikan nilai elemen stack yang paling kecil nilainya\*/  /\*\*\*\*\*\*\*\*\* proses terhadap list \*\*\*\*\*\*\*\*\*/  void delAll(stack \*S);  /\* menghapus semua elemen stack dan semua elemen di-dealokasi \*/  #endif |

1. Dari contoh ADT Stack diatas buatlah file \*.c – nya! Langkah pertama salin terlebih dahulu file \*.h-nya yang isinya sama dengan contoh diatas. Kemudian buat file \*.c – nya dengan nama sesuai dengan file \*.h-nya.
2. Setelah selesai nomor 1, kemudian implementasikan ADT yang telah dibuat dengan membuat sebuah program yang bisa meminta masukan, menghapus dan menampilkan semua data stack! Proses memasukkan dan menghapus data dilakukan satu – persatu.
3. Studi Kasus

Buatlah sebuah program permainan MENARA HANOI dengan mengimplementasikan struktur data stack. MENARA HANOI adalah sebuah game memindahkan disk-disk dari menara ke menara yang lain. Aturannya adalah sebagai berikut.

* + - Terdapat tiga menara, satu menara sudah berisi tumpukan sejumlah disk
    - Disk-disk tersebut berukuran/berbobot berbeda-beda
    - Disk dapat ditumpuk jika menara tersebut masih kosong atau disk teratasnya pada menara tersebut ukurannya/bobotnya lebih besar dari disk yang akan ditumpuk.
    - Satu menara awal yang berisi tumpukan sejumlah disk, disknya berurutan dari ukuran/bobot paling besar sampai paling kecil (dari terbawah sampai teratas)
    - Permainan berakhir jika semua tumpukan disk berada pada satu menara selain menara awal.

Implementasi yang harus ada:

|  |
| --- |
| void CreateStack(Stack \*S);  // I.S. Sembarang  // F.S. Terbentuk Stack kosong  bool IsEmpty(Stack S);  // Menghasilkan True jika Stack kosong, dan False jika Stack tidak kosong  Address Allocate(InfoType X);  // Menghasilkan Address dari alokasi sebuah elemen dengan InfoType X  // Jika alokasi berhasil maka nilai Address tidak Nil dan jika  // gagal nilai Address Nil  void DeAllocate(Address P);  // I.S. P terdefinisi  // F.S. Memori yang digunakan oleh P dikembalikan ke system  void Push(Stack \*S, Address P);  // I.S. Sembarang, P terdefinisi  // F.S. Menempatkan P pada Top dari S  void Pop(Stack \*S, Address \*P);  // I.S. Stack tidak kosong  // F.S. Mengambil P dari Top dari S  void ViewStack(Stack S);  // I.S. Sembarang  // F.S. Menampilkan semua Info dari masing-masing elemen dari Stack |

Program ‑ Hanoi.h

# QUEUE (Representasi Pointer)

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Memahami konsep queue.
2. Mengaplikasikan queue dengan menggunakan pointer.

## Pengertian

Queue merupakan struktur data yang dapat diumpamakan seperti sebuah antrian. Misalkan antrian pada loket pembelian tiket Kereta Api. Orang yang akan mendapatkan pelayanan yang pertama adalah orang pertamakali masuk dalam antrian tersebut dan yang terakhir masuk dia akan mendapatkan layanan yang terakhir pula. Jadi prinsip dasar dalam Queue adalah FIFO (First in Fisrt out), proses yang pertama masuk akan diakses terlebih dahulu.Dalam pengimplementasian struktur Queue dalam C++ dapat menggunakan tipe data array dan linked list.

Dalam praktikum ini hanya akan dibahas pengimplementasian Queue dalam bentuk linked list. Implementasi Queue dalam linked list sebenarnya tidak jauh berbeda dengan operasi list biasa, malahan lebih sederhana. Karena sesuai dengan sifat FIFO dimana proses delete hanya dilakukan pada bagian Head (depan list) dan proses insert selalu dilakukan pada bagian Tail (belakang list) atau sebaliknya, tergantung dari persepsi masing - masing. Dalam penerapannya Queue dapat diterapkan dalam single linked list dan double linked list.

Gambar ‑ Queue

Contoh pendeklarasian struktur data queue:

|  |
| --- |
| #ifndef queue\_H  #define queue\_H  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define head(Q) ((Q).head)  #define tail(Q) ((Q).tail)  typedef int infotype; /\* tipe data dalam queue\*/  typedef struct tElm \*address; /\* tipe data pointer untuk elemen queue \*/  typedef struct tElm {  infotype info;  address next;  }elmStack; /\* tipe data elemen queue \*/  /\* pendeklarasian tipe data queue \*/  typedef struct {  address head, tail;  } queue;  #endif |

Program ‑ Deklarasi Struktur Data Queue

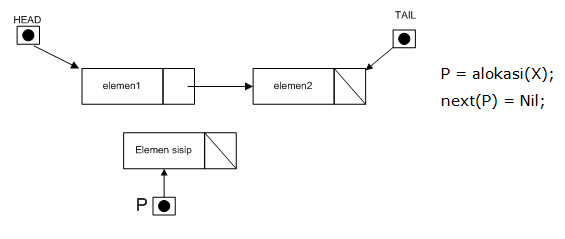
## Operasi-Operasi dalam Queue

Dalam queuE ada dua operasi utama, yaitu operasi penyisipan (Insert/Enqueue) dan operasi pengambilan (Delete/Dequeue).

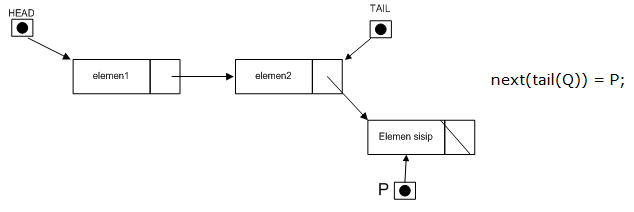
### Insert (Enqueue)

Operasi penyisipan selalu dilakukan pada akhir (tail).

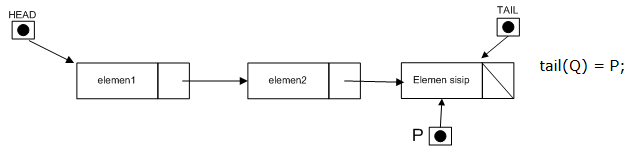
Langkah – langkah dalam proses Enqueue:



Gambar ‑ Queue Insert 1



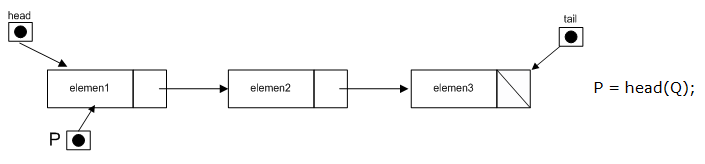
Gambar ‑ Queue Insert 2



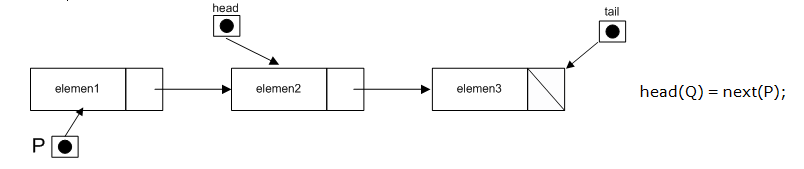
Gambar ‑ Queue Insert 3

### Delete (Dequeue)

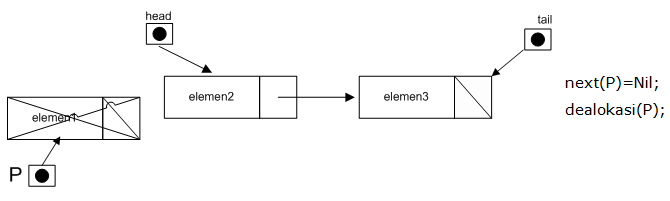
Operasi delete dilakukan pada awal (head).



Gambar ‑ Queue Delete 1



Gambar ‑ Queue Delete 2



Gambar ‑ Queue Delete 3

## Primitif-Primitif dalam Queue

Primitif-primitif pada queue tersimpan pada ADT queue, seperti pada materi sebelumnya, primitif-primitfnya tersimpan pada file \*.h dan \*.cpp.

File \*.h untuk ADT queue:

|  |
| --- |
| #ifndef queue\_H  #define queue\_H  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define head(S) ((S).head)  #define tail(S) ((S).tail)  #include <boolean.h>  #include <iostream>  #include <stdlib.h>  #include <conio.h>  using name space std;  typedef int infotype;  typedef struct tElm \*address;  typedef struct tElm {  infotype info;  address next;  }elmQ;  /\* deklarasi tipe data queue, terdiri dari haed dan tail, queue kosong jika head = Nil\*/  typedef struct {  address head, tail;  } queue;  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah list kosong \*\*\*\*\*\*\*\*\*/  boolean isEmpty(queue Q);  /\* mengembalikan nilai true jika list kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void CreateStack(queue \*Q);  /\* I.S. sembarang \*/  /\* F.S. terbentuk queue kosong \*/  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotype X);  /\* mengirimkan address dari alokasi sebuah elemen \*/  /\* jika aloksi berhasil maka nilai address tidak Nil dan jika gagal nilai address Nil \*/  void Dealokasi(address P);  /\* I.S. P terdefinisi \*/  /\* F.S. memori yang digunakan P dikembalikan ke sistem \*/  /\*\*\*\*\*\*\*\*\* pencarian sebuah elemen list \*\*\*\*\*\*\*\*\*/  address findElm(queue Q, infotype X);  /\* mencari apakah ada elemen queue dengan info(P) = X\*/  /\* jika ada mengembalikan address elemen tsb, dan Nil jika sebaliknya\*/  boolean fFindEml(queue Q, address P);  /\* mencari apakah ada elemen queue dengan alamat P\*/  /\* mengembalikan true jika ada dan false jika tidak ada \*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan nilai \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen pada list \*/  void Ninsert(queue \*Q, infotype X);  /\* I.S. queue mungkin kosong \*/  /\* F.S. mengalokasikan sebuah elemen, menambahkan elemen tsb pada akhir queue dengan nilai X \*/  /\*\*\*\*\*\*\*\*\* penghapusan elemen pada list \*/  void Ndelete(queue \*Q, infotype \* X);  /\* I.S. queue tidak kosong \*/  /\* F.S. elemen pertama dihapus: nilai info disimpan pada X dan elemen pertama di-dealokasi \*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan alamat \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen berdasarkan alamat \*\*\*\*\*\*\*\*\*/  void insert(queue \*Q, address P);  /\* I.S. sembarang, P sudah dialokasikan \*/  /\* F.S. menempatkan elemen beralamat P pada akhir queue \*/  /\*\*\*\*\*\*\*\*\* penghapusan sebuah elemen \*\*\*\*\*\*\*\*\*/  void delete(queue \*Q, address \* P);  /\* I.S. queue tidak kosong \*/  /\* F.S. P menunjuk elemen pertama queue, head dari queue menunjuk pada next elemen head yang lama\*/  /\* queue mngkin menjadi kosong\*/  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void printfInfo(queue Q);  /\* I.S. queue mungkin kosong \*/  /\* F.S. jka queue tidak kosong menampilkan semua info yang ada pada queue \*/  int nbList(queue Q);  /\* mengembalikan jumlah elemen pada queue \*/  infotype maks(queue Q);  /\* mengembalikan nilai elemen queue yang paling besar nilainya\*/  infotype min(queue Q);  /\* mengembalikan nilai elemen queue yang paling kecil nilainya\*/  /\*\*\*\*\*\*\*\*\* proses terhadap list \*\*\*\*\*\*\*\*\*/  void delAll(queue \*Q);  /\* menghapus semua elemen queue dan semua elemen di-dealokasi \*/    #endif |

Program ‑ Queue.h

## Priority Queue

### Pengertian

Priority queue merupakan pengembangan dari queue biasa. Keistimewaan dari queue jenis ini adalah kita bisa menambahkan ke dalam antrian sesuai dengan prioritas yang ada. Misalnya pasien dengan penyakit yang parah akan didahulukan dalam penanganan oleh rumah sakit, sedangkan pasien dengan penyakit yang biasa-biasa akan di tangani setelahnya. Dan masih banyak lagi penerapan priority queue dala kehidupan sehari-hari selain kasus di atas.

### Bentuk Priority Queue

Bentuk priority queue ada 2 macam berdasarkan jumlah queue yang digunakan, yang pertama menggunakan 1 queue dan yang ke dua menggunakan beberapa queue.

### Menggunakan Satu Queue

Dengan queue jenis ini kita bisa membuat priority queue dengan 1 queue saja dengan beberapa prioritas didalamnya. Pada saat dilakukan enQueue digunakan algoritma InsertAfter yang sebelumnya dilakukan pengecekan terhadap prioritas yang ada. Queue dengan prioritas yang lebih rendah tidak boleh berada di depan atau di belakang queue dengan prioritas yang lebih tinggi. Sedangkan proses deQueue dilakukan menggunakan DeleteFirst seperti Queue pada umumnya.

Misalkan terdapat 2 prioritas dalam suatu rumah sakit dalam penanganan pasien, prioritas A dan B. Prioritas A = sakit berat, Prioritas B = sakit Sedang. Untuk penganannya, pasien dengan Prioritas A akan didahulukan.

### EnQueue



Apabila dilakukan enQueue pasien dengan prioritas A, maka struktur Queue akan berubah menjadi seperti dibawah ini :



### DeQueue



Apabila dilakukan deQueue pasien, maka struktur Queue akan berubah menjadi seperti dibawah ini :



### Menggunakan beberapa Queue

Dengan queue jenis ini digunakan beberapa queue dalam implementasinya. Apabila dibutuhkan 2 prioritas maka dibuat 2 queue, apabila dibutuhkan 3 prioritas maka dibuat 3 queue dan seterusnya. Setiap queue yang dibuat mewakili setiap prioritas yang ada. Proses deQueue dilakukan terlebih dahulu terhadap queue degan prioritas yang lebih tinggi, sedangkan proses enQueue dilakukan di setiap queue yang dibuat tergantung dari prioritasnya.

Misalkan terdapat 2 prioritas dalam suatu rumah sakit dalam penanganan pasien, prioritas A dan B. Prioritas A = sakit berat, Prioritas B = sakit Sedang. Untuk penganannya, pasien dengan Prioritas A akan didahulukan.

### EnQueue





Proses enQueue dilakukan di masing-masing prioritas. Apabila kita melakukan enQueue pasien dengan prioritas B, maka pasien B di enQueu pada queue prioritas B. Hasilnya menjadi seperti dibawah ini,





### DeQueue





Proses deQueue dilakukan dari queue yang memiliki prioritas yang lebih tinggi terlebih dahulu, apabila proses yang lebih tinggi telah habis (Queue Kosong), barulah proses deQueue dilakukan pada level dibawahnya.





### Primitif-primitif dalam Priority Queue

Primitif-primitif pada priority Queue sama halnya dengan queue pada umumnya, perbedaannya hanya pada proses enQueue dan deQueue nya.

## Latihan

1. Buatlah file \*.cpp untuk ADT program 6.2 queue.h di atas.
2. Implementasikan untuk membaut suatu sistem antrian pada sistem posfik – infik dengan dipadukan menggunakan stack.
3. Studi Kasus

Diberikan sebuah kasus pengiriman barang dengan batas maksimal jumlah barang tertentu (diinputkan user) dengan 3 prioritas (1 untuk tinggi, 2 untuk prioritas sedang, 3 untuk prioritas rendah). Untuk setiap barang yang ditambahkan, diset prioritas dari barang tersebut, dan untuk barang yang dikirim lebih dulu adalah barang dengan prioritas pengiriman tertinggi. Bila nilai maksimum dari jumlah jenis barang sudah terlampaui, maka akan diminta konfirmasi untuk menghapus barang yang paling lama berada di prioritas tertinggi dan menggantikannya dengan barang yang baru.

Buatlah sebuah program yang dapat mengatur pengiriman barang dengan menggunakan implementasi queue pada bahasa c++. Berikut disediakan file \*.h untuk kasus di atas:

|  |
| --- |
| #ifndef queuebarang\_H  #define queuebarang\_H  #define nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define head(Q) ((Q).head)  #define tail(Q) ((Q).tail)  #include <boolean.h>  #include <iostream>  #include <conio.h>  Using namespace std;  typedef struct{  char namaBarang[10];  char kotaTujuan[10];  int jumlah;  int harga;  int total;  }infotype;  typedef struct tElm \*address;  typedef struct tElm {  infotype info;  address next;  }elmQ;  /\* deklarasi tipe data queue, terdiri dari haed dan tail, queue kosong jika head = Nil\*/  typedef struct {  address head, tail;  } queue;  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah list kosong \*\*\*\*\*\*\*\*\*/  boolean isEmpty(queue Q);  /\* mengembalikan nilai true jika list kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void createQueue(queue \*Q);  /\* I.S. sembarang \*/  /\* F.S. terbentuk queue kosong \*/  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotype X);  /\* mengirimkan address dari alokasi sebuah elemen \*/  /\* jika aloksi berhasil maka nilai address tidak Nil dan jika gagal nilai address Nil \*/  void dealokasi(address P);  /\* I.S. P terdefinisi \*/  /\* F.S. memori yang digunakan P dikembalikan ke sistem \*/  /\*\*\*\*\*\*\*\*\* Primitif berdasarkan alamat \*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\* penambahan elemen berdasarkan alamat \*\*\*\*\*\*\*\*\*/  void enqueue(queue \*Q, address P);  /\* I.S. sembarang, P sudah dialokasikan \*/  /\* F.S. menempatkan elemen beralamat P pada akhir queue \*/  /\*\*\*\*\*\*\*\*\* penghapusan sebuah elemen \*\*\*\*\*\*\*\*\*/  void dequeue(queue \*Q, address \*P);  /\* I.S. queue tidak kosong \*/  /\* F.S. P menunjuk elemen pertama queue, head dari queue menunjuk pada next elemen head yang lama\*/  /\* queue mngkin menjadi kosong\*/  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void printfInfo(queue Q);  /\* I.S. queue mungkin kosong \*/  /\* F.S. jka queue tidak kosong menampilkan semua info yang ada pada queue \*/  int isiQueue(queue Q);  /\* mengembalikan jumlah elemen pada queue \*/  #endif |

Program ‑ queuebarang.h

# QUEUE (Representasi Tabel)

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Memahami konsep queue representasi tabel.
2. Mengaplikasikan queue dengan menggunakan tabel.

## Pengertian

Pada dasarnya representasi queue menggunakan table sama halnya dengan menggunakan pointer. Perbedaan yang mendasar adalah pada management memori serta keterbatasan jumlah antriannya. Untuk lebih jelasnya perhatikan perbedaan representasi tabel dan pointer pada queue dibawah ini.

|  |  |  |
| --- | --- | --- |
| **No** | **Representasi Table** | **Representasi Pointer** |
| 1 | Jumlah Queue terbatas | Jumlah Queue tak-terbatas |
| 2 | Tidak ada management memori | ada management memori |

## Macam-macam Bentuk Queue

### Alternatif 1

Tabel dengan hanya representasi TAIL adalah indeks elemen terakhir, HEAD selalu diset sama dengan 1 jika Queue tidak kosong. Jika Queue kosong, maka HEAD=0.

Ilustrasi Queue tidak kosong, dengan 5 elemen :



Ilustrasi Queue kosong:



Algoritma paling sederhana untuk penambahan elemen jika masih ada tempatadalah dengan “memajukan” TAIL. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1. Algoritma paling sederhana dan “naif” untuk penghapusan elemen jika Queue tidak kosong: ambil nilai elemen HEAD, geser semua elemen mulai dari HEAD+1 s/d TAIL (jika ada), kemudian TAIL “mundur”. Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI. Algoritma ini mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi tidak efisien.

### Alternatif 2

Tabel dengan representasi HEAD dan TAIL, HEAD “bergerak” ketika sebuahelemen dihapus jika Queue tidak kosong. Jika Queue kosong, maka HEAD=0. Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan pertama HEAD“sedang berada di posisi awal:



Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan pertama HEAD tidakberada di posisi awal. Hal ini terjadi akibat algoritma penghapusan yang dilakukan (lihat keterangan).



Ilustrasi Queue kosong:



Algoritma untuk penambahan elemen sama dengan alternatif I: jika masih adatempat adalah dengan “memajukan” TAIL. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1. Algoritmanya sama dengan alternatif I.

Algoritma untuk penghapusan elemen jika Queue tidak kosong: ambil nilai elemen HEAD, kemudian HEAD “maju”. Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI. Algoritma ini TIDAK mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi efisien. Namun suatu saat terjadi keadaan Queue penuh tetapi “semu sebagai berikut :



Jika keadaan ini terjadi, haruslah dilakukan aksi menggeser elemen untukmenciptakan ruangan kosong. Pergeseran hanya dilakukan jika dan hanya jika TAIL sudah mencapai IndexMax.

### Alternatif 3

Tabel dengan representasi HEAD dan TAIL yang “berputar” mengelilingi indekstabel dari awal sampai akhir, kemudian kembali ke awal. Jika Queue kosong, maka HEAD=0. Representasi ini memungkinkan tidak perlu lagi ada pergeseran yang harus dilakukan seperti pada alternatif II pada saat penambahan elemen. Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD “sedang” berada di posisi awal:



Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD tidak berada diposisi awal, tetapi masih “lebih kecil” atau “sebelum” TAIL. Hal ini terjadi akibat algoritma penghapusan/penambahan yang dilakukan (lihat keterangan).



Ilustrasi Queue tidak kosong, dengan 5 elemen, HEAD tidak berada di posisiawal, tetapi “lebih besar” atau “sesudah” TAIL. Hal ini terjadi akibat algoritma penghapusan/Penambahan yang dilakukan (lihat keterangan)



Algoritma untuk penambahan elemen: jika masih ada tempat adalah dengan“memajukan” TAIL. Tapi jika TAIL sudah mencapai IdxMax, maka suksesor dari IdxMax adalah 1 sehingga TAIL yang baru adalah 1. Jika TAIL belum mencapai IdxMax, maka algoritma penambahan elemen sama dengan alternatif II. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1.

Algoritma untuk penghapusan elemen jika Queue tidak kosong: ambil nilai elemen HEAD, kemudian HEAD “maju”. Penentuan suatu suksesor dari indeks yang diubah/”maju” dibuat Seperti pada algoritma penambahan elemen: jika HEAD mencapai IdxMAx, maka suksesor dari HEAD adalah 1. Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI. Algoritma ini efisien karena tidak perlu pergeseran, dan seringkali strategi pemakaian tabel semacam ini disebut sebagai “circular buffer”, dimana tabel penyimpan elemen dianggap sebagai “buffer”.

Salah satu variasi dari representasi pada alternatif III adalah : menggantikan representasi TAIL dengan COUNT, yaitu banyaknya elemen Queue. Dengan representasi ini, banyaknya elemen diketahui secara eksplisit, tetapi untuk melakukan penambahan elemen harus dilakukan kalkulasi TAIL. Buatlah sebagai latihan.

## Primitif-Primitif dalam Queue

Berikut ini contoh program queue.h dalam ADT queue menggunakan representasi table.

|  |
| --- |
| #ifndef STACK\_H\_INCLUDED  #define STACK\_H\_INCLUDED  #include <iostream>  #include <conio.h>  #include <stdlib.h>  Using namespace std;  typedef struct {  char id[20];  char nama[20];  } infotype;  typedef struct {  infotype info[3];  int IMax;  int head;  int tail;  } queue;  /\* prototype \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah Stack penuh \*\*\*\*\*\*\*\*\*/  int isFull(queue Q);  /\* mengembalikan nilai 0 jika queue penuh \*/  /\*\*\*\*\*\*\*\*\* pengecekan apakah Stack kosong \*\*\*\*\*\*\*\*\*/  int isEmpty(queue Q);  /\* mengembalikan nilai 0 jika queue kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan Stack \*\*\*\*\*\*\*\*\*/  void createQueue(queue \*Q);  /\* I.S. sembarang \*/  /\* F.S. terbentuk queue dengan head=-1 dan tail=-1 \*/  /\*\*\*\*\*\*\*\*\* penambahan elemen pada Stack \*/  void enQueue(queue \*Q, infotype X);  /\* I.S. queue mungkin kosong \*/  /\* F.S. menambahkan elemen pada queue dengan nilai X \*/  /\*\*\*\*\*\*\*\*\* penghapusan elemen pada list \*/  void deQueue(queue \*Q);  /\* I.S. queue tidak kosong \*/  /\* F.S. head=head+1 \*/  /\*\*\*\*\*\*\*\*\* proses semua elemen list \*\*\*\*\*\*\*\*\*/  void viewQueue(queue Q);  /\* I.S. queue mungkin kosong \*/  /\* F.S. jika queue tidak kosong menampilkan semua info yang ada pada queue \*/  #endif // STACK\_H\_INCLUDED |

Gambar ‑ arrayQueue.h

## Latihan

1. Buatlah program priority queue di sebuah rumah sakit menggunakan model priority beberapa queue.

# TREE

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat :

1. Praktikan mampu mengaplikasikan struktur data tree dalam sebuah kasus pemrograman.
2. Praktikan bisa lebih mahir dalam menggunakan struktur data tree.

## Binary Search Tree

Binary search tree adalah Binary tree yang terurut dengan ketentuan:

1. Semua LEFTCHILD harus lebih kecil dari parentnya.
2. Semua RIGHTCHILD harus lebih besar dari parentnya dan leftchildnya.

#### AVL Tree

Adalah binary search tree yang mempunyai ketentuan bahwa maximum perbedaan height antara subtree kiri dan subtree kanan adalah 1.

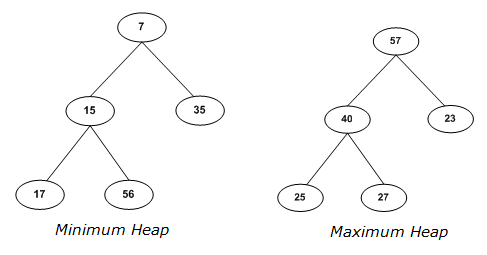
#### Heap Tree

Adalah tree yang memenuhi persamaan berikut: R[i] < r[2i] and R[i] < r[2i+1]

Heap juga disebut Complete Binary Tree, karena jika suatu node mempunyai child, maka jumlah childnya harus selalu dua.

Minimum Heap: jika parentnya selalu lebih kecil daripada kedua childrennya.

Maximum Heap : jika parentnya selalu lebih besar daripada kedua childrennya.

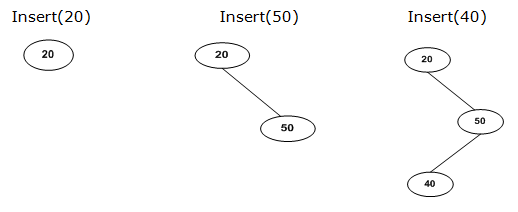


Gambar ‑ Heap Tree

## Operasi-Operasi dalam Binary Search Tree

#### Insert

1. Jika node yang akan diiinsert lebih kecil, maka diinsert pada Left Subtree
2. Jika lebih besar, maka diinsert pada Right Subtree.



Gambar ‑ Binary Search Tree Insert

#### Update

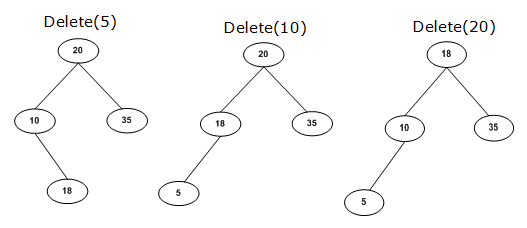
Jika setelah diupdate posisi/lokasi node yang bersangkutan tidak sesuai dengan ketentuan, maka harus dilakukan dengan proses REGENERASI agar tetap memenuhi kriteria Binary Search Tree.

#### Delete

1. LEAF, tidak perlu dilakukan modifikasi.
2. Node dengan 1 Child, maka child langsung menggantikan posisi Parent.
3. Node dengan 2 Child:
   * + Left Subtree, yang diambil adalah node yang paling kiri(nilai terbesar).
     + Right Subtree, yang diambil adalah node yang paling kanan(nilai terkecil).

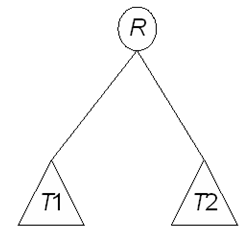


Gambar ‑ Binary Search Tree sebelum di-Delete



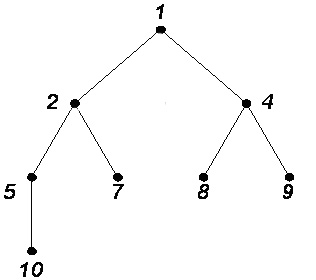
Gambar ‑ Binary Search Tree setelah di-Delete

## Traversal pada Binary Tree



Gambar ‑ Traversal pada Binary Tree 1

1. Pre-order : R, T1, T2
   * + kunjungi R
     + kunjungi T1 secara pre-order
     + kunjungi T2 secara pre-order
2. In-order : T1 , R, T2
   * + kunjungiT1 secara in-order
     + kunjungi R
     + kunjungi T2 secara in-order
3. Post-order : T1, T2 , R
   * + kunjungiT1 secara pre-order
     + kunjungi T2 secara pre-order
     + kunjungi R



Gambar ‑ traversal pada Binary Tree 2

Sebagai contoh apabila kita mempunyai tree dengan representasi seperti di atas ini maka proses traversal masing-masing akan menghasilkan ouput:

1. Pre-order : 1-2-5-10-7-4-8-9
2. In-order : 10-5-2-7-1-8-4-9
3. Post-order : 10-5-7-2-8-9-4-1

Berikut ini ADT untuk tree dengan menggunakan representasi list linier:

|  |
| --- |
| #ifndef tree\_H  #define tree\_H  #include "boolean.h"  #define Nil NULL  #define info(p) p->info  #define right(p) p->right  #define left(p) p->left  typedef int infotype;  typedef struct BinTree \*address;  typedef struct BinTree {  infotype info;  address left;  address right;  }elmtree;  typedef struct {  address root;  }Tree;  //fungsi primitif pohon biner  /\*\*\*\*\*\*\*\*\* pengecekan apakah tree kosong \*\*\*\*\*\*\*\*\*/  boolean EmptyTree(Tree T);  /\* mengembalikan nilai true jika list kosong \*/  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void CreateTree(Tree \*T);  /\* I.S. sembarang \*/  /\* F.S. terbentuk list kosong \*/  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotype X);  /\* mengirimkan address dari alokasi sebuah elemen \*/  /\* jika aloksi berhasil maka nilai address tidak Nil dan jika gagal nilai address Nil \*/  void Dealokasi(address P);  /\* I.S. P terdefinisi \*/  /\* F.S. memori yang digunakan P dikembalikan ke sistem \*/  /\*Konstruktor\*/  address CreateElemen(infotype X,address L,address R)  /\*menghasilkan sebuah elemen tree dengan info X daN elemen kiri L dan elemen kanan R\*/  /\*mencari elemen tree tertentu\*/  address findElmBinTree(Tree T, infotype X);  /\* mencari apakah ada elemen tree dengan info(P) = X\*/  /\* jika ada mengembalikan address elemen tsb, dan Nil jika sebaliknya\*/  address findLeftBinTree(Tree T, infotype X);  /\* mencari apakah ada elemen sebelah kanan tree dengan info(P) = X\*/  /\* jika ada mengembalikan address elemen tsb, dan Nil jika sebaliknya\*/  address findRightBinTree(Tree T, infotype X);  /\* mencari apakah ada elemen sebelah kiri tree dengan info(P) = X\*/  /\* jika ada mengembalikan address elemen tsb, dan Nil jika sebaliknya\*/  /\*menginsertkan elemen tree\*/  void InsertBinTree(Tree T,address p);  /\*IS:Tree bisa saja kosong\*/  /\*FS:memasukan p ke dalam tree terurut sesuai konsep binary tree\*/  /\*Menghapus elemen tree tertentu\*/  void DelBinTree(Tree T,address p);  /\*IS:Tree tidak kosong\*/  /\*FS:menghapus p ke dari tree\*/  /\*Selektor\*/  infotype Akar(Tree T);  /\*mengembalikan nilai dari akar\*/  void Preorder(Tree \*T);  /\*IS :Tree tidak kosong\*/  /\*FS : Menampilkan tree secara preorder\*/  void Inorder(Tree \*T);  /\*IS :Tree tidak kosong\*/  /\*FS : Menampilkan tree secara inorderr\*/  void Postorder(Tree \*T);  /\*IS :Tree tidak kosong\*/  /\*FS : Menampilkan tree secara post\*/  #endif |

Program ‑ tree.h

## Latihan

1. Dari contoh ADT Tree diatas buatlah file \*.cpp – nya! Langkah pertama salin terlebih dahulu file \*.h-nya yang isinya sama dengan contoh diatas. Kemudian buat file \*.cpp – nya dengan nama sesuai dengan file \*.h-nya.
2. Setelah selesai nomor 1, kemudian implementasikan ADT yang telah dibuat dengan membuat sebuah program yang bisa meminta masukan infotype , mahasiswa (nama, nim, nilai), dimana sebelah kiri nilainya selalu lebih kecil dari sebelah kanan (jika nilai sama, ke sebelah kanan). Menghapus dan menampilkan semua data tree. Untuk menampilkan data pada tree gunakan fungsi/prosedur seara rekursif dengan pre-order, in-order, dan post-order.

# GRAPH TAK BERARAH

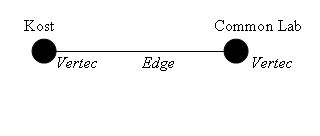
## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat :

1. Memahami konsep graph tak berarah
2. Mengaplikasikan graph tak berarah dengan menggunakan pointer.

## Pengertian Graph

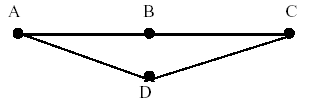
Graph merupakan himpunan tidak kosong dari node (vertec) dan garis penghubung (edge). Contoh sederhana tentang graph, yaitu antara Tempat Kost Anda dengan Common Lab. Tempat Kost Anda dan Common Lab merupakan node (vertec). Jalan yang menghubungkan tempat Kost dan Common Lab merupakan garis penghubung antara keduanya (Edge).



Gambar ‑ Graph Kost dan Common Lab

## Pengertian Graph tak Berarah

Merupakan graph dimana tiap node memiliki edge yang dihubungkan ke node lain tanpa arah.



Gambar ‑ Graph Tidak Berarah (Undirected Graph)

Selain arah, beban atau nilai sering ditambahkan pada edge . Misalnya nilai yang merepresentasikan panjang, biaya transportasi, dan lain-lain. Hal mendasar lain yang perlu diketahui adalah suatu node A dikatakan bertetangga dengan node B jika antara node A dan node B dihubungkan langsung dengan sebuah edge.

Misalnya:

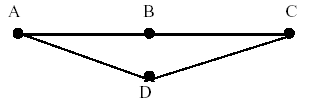
Dari gambar contoh graph diatas dapat disimpulkan bahwa: A bertetangga dengan B, B bertetangga dengan C, A tidak bertetangga dengan C, B tidak bertetangga dengan D.

Masalah ketetanggaan suatu node dengan node yang lain harus benar-benar diperhatikan dalam implementasi pada program. Ketetanggaan dapat diartikan sebagai keterhubungan antar node yang nantinya informasi ini dibutuhkan untuk melakukan beberapa proses seperti : mencari lintasan terpendek dari suatu node ke node yang lain, pengubahan graph menjadi tree (untuk perancangan jaringan) dan lain-lain.

Tentu anda sudah tidak asing dengan algoritma Djikstra, Kruskal, Prim dsb. Karena waktu praktikum terbatas, kita tidak membahas algoritma tersebut. Di sini anda hanya akan mencoba untuk mengimplementasikan graph dalam program.

## Representasi Graph

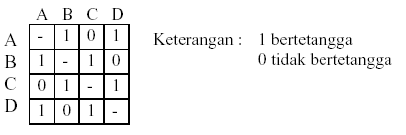
Dari definisi graph dapat kita simpulkan bahwa graph dapat direpresentasikan dengan Matrik Ketetanggaan (Adjacency Matrices), yaitu matrik yang menyatakan keterhubungan antar node dalam graph. Implementasi matrik ketetanggaan dalam bahasa pemrograman dapat berupa : Array 2 Dimensi dan Multi Linked List. Graph dapat direpresentasikan dengan matrik n x n, dimana n merupakan jumlah node dalam graph tersebut. Perhatikan graph tidak berarah berikut :



Gambar ‑ Graph

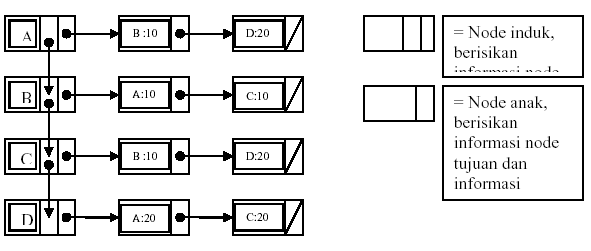
Graph tersebut dapat direpresentasikan dengan 2 cara:

### Array 2 Dimensi



Gambar ‑ Representasi Graph Array 2 Dimensi

### Multi Linked List



Gambar . Representasi Graph Multi Linked List

Dalam praktikum ini untuk merepresentasikan graph akan menggunakan multi list. Karena sifat list yang dinamis, sehinga data yang bisa ditangani bersifat dinamis. Contoh ada sebuah graph yang menggambarkan jarak antar kota:

.

Gambar ‑ Graph Jarak Antar kota

Representasi struktur data graph pada array:

|  |
| --- |
| #ifndef GRAPH\_H\_INCLUDED  #define GRAPH\_H\_INCLUDED  #include <iostream>  #include <stdlib.h>  #include <conio.h>  typedef char infoGraph;  typedef struct tElmNode \*adrNode;  typedef struct tElmEdge \*adrEdge;  typedef struct tElmNode {  infoGraph Info;  int Visited;  adrEdge firstEdge;  adrNode Next;  } ElmNode;  typedef struct tElmEdge {  adrNode Node;  adrEdge Next;  } ElmEdge;  typedef struct {  adrNode First;  } Graph;  #endif // GRAPH\_H\_INCLUDED |

Program ‑ Deklarasi Struktur Data Graph dengan Representasi Array 2 Dimensi

Karena representasinya menggunakan multilist maka primitive-primitif graph sama dengan primitif -primitif pada multilist. Jadi untuk membuat ADT graph bisa memanfaatkan ADT yang sudah dibuat pada multilist.

## Metode-Metode Penulusuran Graph

### Breadth First Search (BFS)

Cara kerja algoritma ini adalah dengan mengunjungi root, kemudian node yang terhubung dengan root disimpan ke dalam container queue. Node yang pertama kali masuk ke dalam container maka yang pertama dikunjungi, node lainnya yang terhubung pada node yang dikunjungi masuk ke dalam queue, dan seterusnya. BFS menelusuri dengan menggunakan Queue, sehingga memiliki kunjungan FIFO.

Perhatikan graph berikut :

|  |  |
| --- | --- |
| Gambar ‑ Graph Breadth First Search (BFS) | Tabel ‑ Container Queue BFS |

Urutannya : A B D C E H FG I

### Depth First Search (DFS)

Cara kerja algoritma ini adalah dengan mengunjungi root, kemudian node yang terhubung dengan root disimpan ke dalam container stack. Node yang terakhir masuk ke dalam container maka yang pertama dikunjungi, node lainnya yang terhubung pada node yang dikunjungi masuk ke dalam stack, dan seterusnya. DFS menelusuri dengan menggunakan Stack, sehingga memiliki kunjungan LIFO.

Perhatikan graph berikut :

|  |  |
| --- | --- |
| Gambar ‑ Graph Depth First Search (DFS) | Tabel ‑ Container Stack DFS |

Urutannya : A B D H E F C G

## Latihan

Buatlah implementasi dari Graph tidak berarah dan primitif yang harus ada:

|  |
| --- |
| // infotype bertipe Char  void CreateGraph (Graph \*G);  /\* prosedur untuk membuat Graph \*/  // I.S. : Graph belum tersedia  // F.S. : Graph telah dibuat  void InsertNode (Graph \*G, infotype X);  /\* prosedur untuk menambahkan node ke dalam Graph \*/  // I.S. : Graph telah dibuat  // F.S. : Node baru diinsertkan dengan mengisi info dengan X  // Tidak boleh ada node yang memiliki info yang sama  void DeleteNode (Graph \*G, infotype X);  /\* prosedur untuk menghapus node pada Graph \*/  // I.S. : Graph telah dibuat, Node dengan info berisi X ada  // F.S. : Node dengan info X terdelete  // Tidak ada Node lain yang memiliki tetangga Node tersebut  void ConnectNode (adrNode N1, adrNode N2);  /\* prosedur untuk menyambungkan Node 1 dengan Node 2 \*/  // I.S. : Graph telah dibuat, Node 1 dan Node 2 ada  // F.S. : Node 1 dan Node 2 bertetangga  void DisconnectNode (adrNode N1, adrNode N2);  /\* prosedur untuk memutuskan Node 1 dengan Node 2 \*/  // I.S. : Graph telah dibuat, Node 1 dan Node 2 ada dan bertetangga  // F.S. : Node 1 dan Node 2 tidak bertetangga  void PrintInfoGraph (Graph G);  /\* prosedur untuk menampilkan semua informasi ketetanggaan pada Graph tersebut \*/  // I.S. : Graph telah dibuat  // F.S. : menampilkan semua informasi ketetanggan masing-masing Node  void PrintDFS (Graph G, adrNode N);  /\* prosedur untuk menampilkan hasil DFS pada Graph dimulai dari Node N \*/  // I.S. : Graph telah dibuat, Node N ada  // F.S. : menampilkan hasil DFS yang dimulai dari Node N  void PrintBFS (Graph G, adrNode N);  /\* prosedur untuk menampilkan hasil BFS pada Graph dimulai dari Node N \*/  // I.S. : Graph telah dibuat, Node N ada  // F.S. : menampilkan hasil BFS yang dimulai dari Node N  // Hint: Gunakan Queue |

Program ‑ graph.h

# 

# GRAPH BERARAH

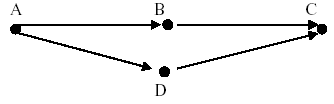
## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat :

1. Memahami konsep graph berarah
2. Mengaplikasikan graph berarah dengan menggunakan pointer.

## Pengertian

Graph berarah merupakan graph dimana setiap node memiliki edge yang berarah kemana node tersebut dihubungkan.



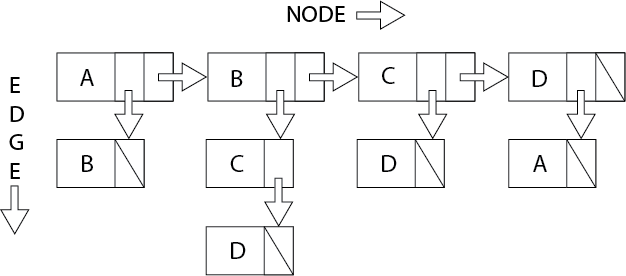
Gambar ‑ Graph Berarah (Directed Graph)

## Representasi Graph

### Multi Linked List

Pada praktikum ini untuk merepresentasikan graph akan menggunakan multi-linked-list. Karena sifat list yang dinamis sehingga data yang bisa ditangani bersifat dinamis pula. Pada dasarnya representasi graph berarah sama dengan graph tak-berarah. Perbedaannya yaitu apabila graph tak-berarah terdapat node A terhubung dengan node B, maka node B akan terhubung juga dengan node A. Pada Graph berarah node A terhubung dengan node B, namun belum tentu node B terhubung dengan node A.

Graph berarah bisa di-representasikan dalam multilist sebagai berikut,



Dari multilist di atas apabila digambarkan dalam bentuk graph menjadi :

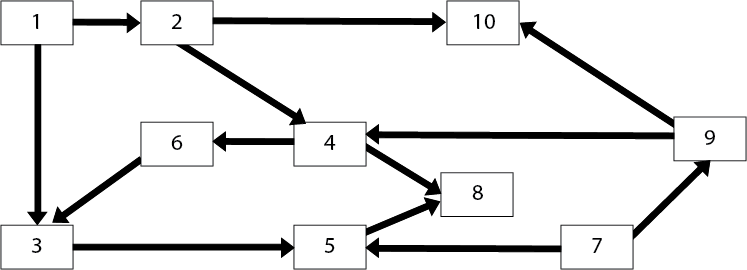
## Topological Sort

### Pengertian

Diberikan urutan partial dari elemen suatu himpunan, dikehendaki agar elemen yang terurut partial tersebut mempunyai keterurutan linier. Contoh dari keterurutan partial banyak dijumpai dalam kehidupan sehari-hari, misalnya:

1. Dalam suatu kurikulum, suatu mata pelajaran mempunyai prerequisit mata pelajaran lain. Urutan linier adalah urutan untuk seluruh mata pelajaran dalam kurikulum.
2. Dalam suatu proyek, suatu pekerjaan harus dikerjakan lebih dulu dari pekerjaan lain (misalnya membuat fondasi harus sebelum dinding, membuat dinding harus sebelum pintu. Namun pintu dapat dikerjakan bersamaan dengan jendela.
3. Dalam sebuah program Pascal, pemanggilan prosedur harus sedemikian rupa sehingga peletakan prosedur pada teks program harus seuai dengan urutan(partial) pemanggilan.
4. Dalam pembuatan tabel pada basis data, tabel yang di-refer oleh tabel lain harus dideklarasikan terlebih dulu. Jika suatu aplikasi terdiri dari banyak tabel, maka urutan pembuatan tabel harus sesuai dengan definisinya.

Jika X < Y adalah simbol untuk X “sebelum” Y, dan keterurutan partial digambarkan sebagai graph, maka graph yang terbentuk sebagai berikut :



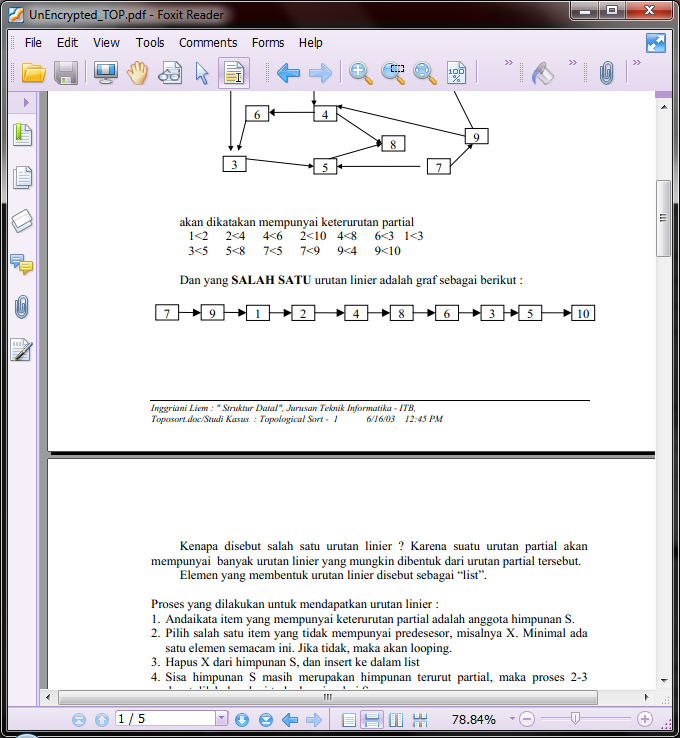
Gambar ‑ Graph Berarah 2

Akan dikatakan mempunyai keterurutan partial ;

1<2 2<4 4<6 2<10 4<8 6<3 1<3

3<5 5<8 7<5 7<9 9<4 9<10

Dan **SALAH SATU** urutan linier adalah graph diatas sebagai berikut :



Kenapa disebut salah satu urutan linier? Karena suatu urutan partial akan mempunyai banyak urutan linier yang mungkin dibentuk dari urutan partial tersebut. Elemen yang membentuk urutan linier disebut sebagai “list”.

Proses yang dilakukan untuk mendapatkan urutan linier :

1. Misal item yang mempunyai keterurutan partial adalah anggota himpunan S.
2. Pilih salah satu item yang tidak mempunyai predesesor, misalnya X. Minimal ada satu elemen semacam ini. Jika tidak, maka akan looping.
3. Hapus X dari himpunan S, dan insert ke dalam list.
4. Sisa himpunan S masih merupakan himpunan terurut partial, maka proses 2 - 3 dapat dilakukan lagi selama himpunan S tidak kosong.
5. Lakukan sampai himpunan S menjadi kosong, dan list mempunyai elemen dengan keterurutan linier

**Solusi I :**

Untuk melakukan hal ini, perlu ditentukan suatu representasi internal. Operasi yang penting adalah memilih elemen tanpa predesesor (yaitu jumlah predesesor elemen sama dengan nol). Maka setiap elemen mempunyai 3 karakteristik : identifikasi, list suksesornya, dan banyaknya predesesor. Karena jumlah elemen bervariasi, maka representasi yang paling cocok adalah list berkait dengan representasi dinamis (pointer). List dari suksesor direpresentasikan pula secara berkait.

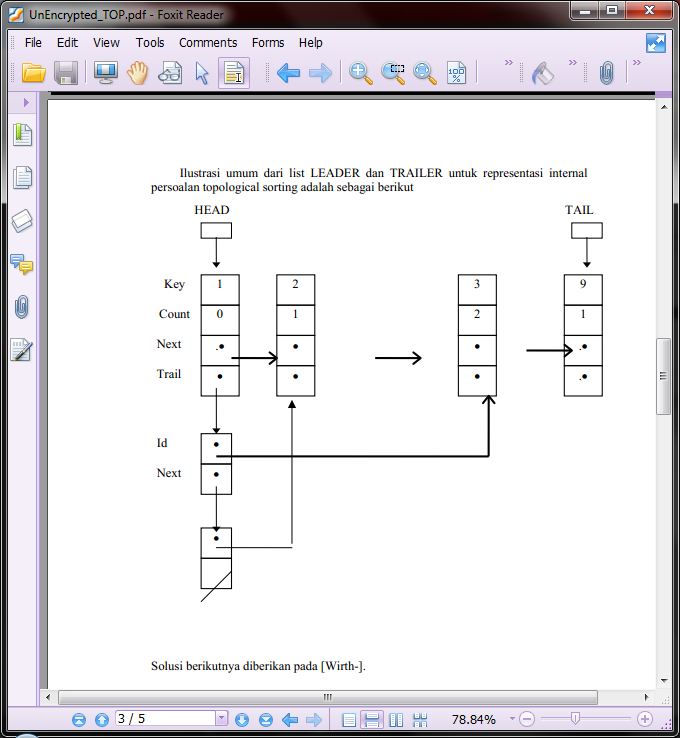
Representasi yang dipilih untuk persoalan ini adalah multilist sebagai berikut :

1. List yang digambarkan horisontal adalah list dari banyaknya predesesor setiapitem, disebut list “Leader”, yang direpresentasikan sebagai list yang dicatat alamat elemen pertama dan terakhir (Head-Tail) serta elemen terurut menurut key. List ini dibentuk dari pembacaan data. Untuk setiap data keterurutan partial X<Y : Jika X dan/atau Y belum ada pada list leader, insert pada Tail dengan metode search dengan sentinel
2. List yang digambarkan vertical adalah list yang merupakan indirect-adressing ke setiap predesesor, disebut sebagai “Trailer”. Untuk setiap elemen listLeader X, list dari suksesornya disimpan sebagai elemen list Trailer yang setiap elemennya berisi alamat dari suksesor. Penyisipan data suatu suksesor (X<Y), dengan diketahui X, maka akan dilakukan dengan InsertFirst alamat Y sebagai elemen list Trailer dengan key X

**Algoritma secara keseluruhan terdiri dari dua pass :**

1. Bentuk list leader dan Trailer dari data keterurutan partial : baca pasangan nilai(X<Y). Temukan alamat X dan Y (jika belum ada, sisipkan), kemudian dengan mengetahui alamat X dan Y pada list Leader, InsertFirst alamat Y sebagai trailer X
2. Lakukan topological sort dengan melakukan search list Leader dengan jumlah predesesor=0, kemudian insert sebagai elemen list linier hasil pengurutan.

Ilustrasi umum dari list LEADER dan TRAILER untuk representasi internal persoalan topological sorting adalah sebagai berikut :



Gambar ‑ Ilustrasi Topological Sort

**Solusi II : pendekatan “fungsional” dengan list linier sederhana.**

Pada solusi ini, proses untuk mendapatkan urutan linier diterjemahkan secara fungsional, dengan representasi sederhana. Graph partial dinyatakan sebagai list linier dengan representasi fisik First-Last dengan dummy seperti representasi pada Solusi I dengan elemen yang terdiri dari <Precc,Succ>.

Contoh : sebuah elemen bernilai <1,2> artinya 1 adalah predesesor dari 2.

**Langkah-Langkah :**

1. Fase input : Bentuk list linier yang merepresentasikan graph seperti pada solusi I.
2. Fase output : Ulangi langkah berikut sampai list “habis”, artinya semua elemen list selesai ditulis sesuai dengan urutan total.

* P adalah elemen pertama (First(L)
* Search pada sisa list, apakah X=Precc(P) mempunyai predesesor.
  + Jika ya, maka elemen ini harus dipertahankan sampai saatnya dapat dihapus dari list untuk dioutputkan:
    - Delete P, tapi jangan didealokasi
    - Insert P sebagai Last(L) yang baru
* Jika tidak mempunyai predesesor, maka X siap untuk dioutputkan, tetapi Y masih harus dipertanyakan. Maka langkah yang harus dilakukan :
  + Outputkan X
  + Search apakah Y masih ada pada sisa list, baik sebagai Precc maupun sebagai Succ
    - Jika ya, maka Y akan dioutputkan nanti. Hapus elemen pertama yang sedang diproses dari list.
    - Jika tidak muncul sama sekali, berarti Y tidak mempunyai predesesor. Maka Outputkan Y, baru hapus elemen pertama dari list

### Representasi Topological Sort

Representasi graph untuk topological sort sama dengan graph berarah pada umumnya.

## Latihan

1. Buatlah implementasi dari ADT berikut :

|  |
| --- |
| #ifndef GRAPH\_H\_INCLUDED  #define GRAPH\_H\_INCLUDED  #include <iostream>  #include <stdlib.h>  #include <conio.h>  using namespace std;  typedef char infoGraph;  typedef struct tElmNode \*adrNode;  typedef struct tElmEdge \*adrEdge;  typedef struct tElmNode {  infoGraph Info;  int Visited; //Successor  int Pred; //Predecessor  adrEdge firstEdge; //Mengarah ke first edge pada list  adrNode Next; //Next node pada list  } ElmNode;  typedef struct tElmEdge {  adrNode Node; //Mengarah ke node yang dituju  adrEdge Next; //Next edge pada list  } ElmEdge;  typedef struct {  adrNode First; //Mengarah ke node pertama pada list  } Graph;  adrNode AllocateNode (infoGraph X);  adrEdge AllocateEdge (adrNode N);  void CreateGraph (Graph \*G);  void InsertNode (Graph \*G, infoGraph X);  void DeleteNode (Graph \*G, infoGraph X);  void ConnectNode (adrNode N1, adrNode N2);  void DisconnectNode (adrNode N1, adrNode N2);  adrNode FindNode (Graph G, infoGraph X);  adrEdge FindEdge (adrNode N, adrNode NFind);  void PrintInfoGraph (Graph G);  void PrintTopologicalSort (Graph G);0  #endif // GRAPH\_H\_INCLUDED |

Program ‑ topological.h

# 

# Responsi Tugas Besar 1

Kegiatan Responsi dilakukan diluar kegiatan praktikum dan dilaksanakan pada modul 6 keatas.  
  
Target pencapaian responsi Tugas Besar 1

* + - 1. Mahasiswa sudah memiliki kelompok masing-masing yang terdiri dari maksimal 3 orang per kelompok.
      2. Setiap Kelompok sudah memiliki tema dan judul tugas besar masing-masing.
      3. Setiap Kelompok sudah membuat Proposal pengerjaan tugas besar yang menjelaskan latar belakang permasalahan, penjelasan system yang dibangun dan pembagian tugas masing-masing per orang pada setiap kelompok.
      4. Progress pencapaian tugas besar per kelompok pada responsi 1 adalah sebesar 25% hingga 50%.

# 

# Responsi Tugas Besar 2

Kegiatan Responsi dilakukan diluar kegiatan praktikum dan dilaksanakan pada modul 8 keatas.  
  
Target pencapaian responsi Tugas Besar 2

1. System yang dibangun sudah dapat berjalan sesuai dengan target yang dibuat oleh setiap kelompok.
2. Setiap kelompok membuat poster promosi mengenai system yang dibuat.
3. Setiap kelompok menyiapkan slide presentasi untuk presentasi tugas besar.
4. Setiap kelompok mem burn keseluruhan artefak pengerjaan tubes ke dalam CD maupun DVD.
5. Progress pencapaian tugas besar per kelompok pada responsi 2 adalah sebesar 75% hingga 100%.

# 

# Presentasi Tugas Besar

Presentasi Tugas Besar merupakan ujian terakhir yang harus dilalui mahasiswa untuk dapat menyelesaikan tugas pada mata praktikum ini.

Beberapa persyaratan yang harus dipenuhi oleh mahasiswa agar dapat mengikuti kegiatan presentasi ini antara lain:

Mahasiswa telah mengikuti kegiatan responsi tugas besar 1 dan responsi tugas besar 2.

Kehadiran kegiatan praktikum minimal 75%.

Mahasiswa telah mengumpulkan keseluruhan artefak tugas besar.

Mematuhi aturan presentasi yang telah disepakati dengan dosen mata praktikum masing-masing.

# 

# DAFTAR PUSTAKA

|  |  |
| --- | --- |
| [1] | Anonim. 2010 .*Modul Algoritma Struktur Data*. Departemen Teknik Informatika.Institut Teknologi Telkom, Bandung. |
| [2] | Anonim. 2011 .*Modul Algoritma Struktur Data*. Departemen Teknik Informatika.Institut Teknologi Telkom, Bandung. |
| [3] | Anonim. 2012 .*Modul Algoritma Struktur Data*. Fakultas Informatika.Institut Teknologi Telkom, Bandung. |
| [4] | Kernighan, Brian W., Ritchie, Dennis M. 1988.*C Programming Language. Second Ed*. Prentice Hall. |
| [5] | Liem, Inggriani. 2003. *Diktat Kuliah IF2181 Struktur Data*. Institut Teknologi Bandung, Bandung. |
| [6] | Standish, Thomas A. 1995.*Data structures, Algorithms, & Software Principles in C*. Addison Wesley Publishing Company. |
| [7] | Wirth, Niklaus. 1996.*Algorithm + data structure = program*. Prentice Hall. |
|  |  |
|  |  |