



MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Kontrak Perkuliahan,
Penyampaian Silabus
Algoritma & Struktur Data dan
Struktur Pemilihan dengan IF

Fakultas

Ilmu Komputer

Program Studi

Sistem
Informasi

Tatap Muka

01

Kode MK

MK87042

Disusun Oleh

Tim Dosen

Abstract

Kompetensi

Memberikan gambaran umum mengenai struktur kendali proses penyeleksian kondisi/percabangan.

Mahasiswa mampu memahami dasar penggunaan Control Statement **if** untuk memberikan pilihan alternatif bagian algoritma yang dilaksanakan atau yang tidak dilaksanakan.

Kontrak Perkuliahan

Agar terciptanya Kegiatan Belajar Mengajar yang kondusif, maka ada beberapa ketentuan yang harus diketahui oleh para mahasiswa diantaranya :

1. Penilaian

Komponen penilaian pada dasarnya terdiri dari :

- Presensi/Kehadiran	: 10%
- Tugas Mandiri/Quiz	: 20%
- UTS	: 30%
- UAS	<u>: 40%</u>
Total	100%

2. Konversi Nilai

Min	Max	Abjad
80	100	A
74	79	B+
68	73	B
62	67	C+
56	61	C
45	55	D
0	44	E

3. Dispensasi Absensi Perkuliahan

- Koreksi nilai akhir yang didasari oleh revisi kehadiran mahasiswa tidak diperkenankan.
- Bagi mahasiswa yang tidak dapat masuk kuliah dikarenakan sakit, maka harus menyerahkan surat Keterangan Sakit dari Dokter dan diserahkan kepada dosen paling lambat 2 (dua) pekan setelah ketidakhadiran.
- Bagi mahasiswa yang tidak dapat masuk kuliah dikarenakan ijin/dispensasi, maka harus menyerahkan surat ijin/dispensasi dari orang tua/wali/organisasi tempatnya

- bernaung dan diserahkan kepada dosen paling lambat 2 (dua) pekan setelah ketidakhadiran.
- d) Segala bentuk ketidakhadiran tanpa menyertakan bukti fisik tidak akan dilayani.

Silabus

Pekan ke-	Materi
1	Kontrak Kuliah, Penyampaian Silabus, Struktur Pemilihan dengan IF
2	Operator dan && dalam Bahasa C, Proses konversi dari multi conditions menjadi nested if.
3	Struktur perulangan dengan for, do-while dan while
4	Pengaplikasian struktur perulangan untuk memecahkan soal-soal matematika dan fisika
5	Function dalam Bahasa C dan Fungsi rekursif
6	Fungsi rekursif (Lanjutan)
7	Array satu dimensi dan array dua dimensi
8	Ujian Tengah Semester
9	Pointer dan Linked-list
10	List traversal secara rekursif dan Penyisipan node di awal dan di tengah linked-list
11	ADT Stack, Implementasi stack dengan array dan Implementasi stack dengan linked-list
12	Implementasi stack dengan linked-list
13	ADT Queue, Implementasi queue dengan array linear dan sirkular dan Implementasi queue dengan linked-list
14	ADT Binary Tree
15	ADT Binary Tree

Control Statement If (Penggunaan Dasar)

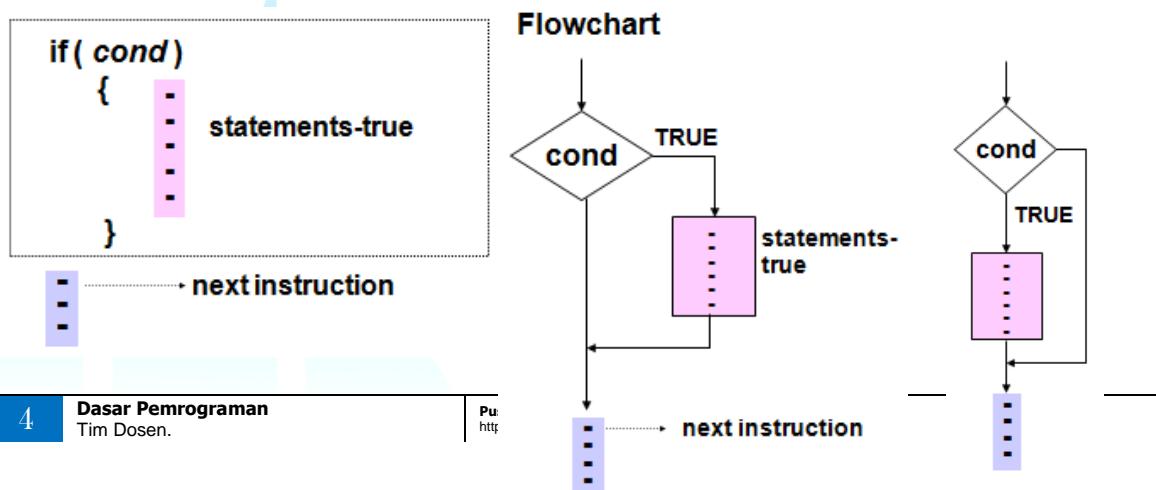
Dalam pembuatan algoritma atau program untuk memecahkan masalah adakalanya kita dihadapi beberapa kondisi atau keadaaan, kondisi tersebut biasanya menentukan hasil akhir dari proses yang akan didapat. Misalnya untuk menentukan kelulusan seorang siswa berdasarkan nilai. Jika nilai siswa lebih dari atau sama dengan 60 maka siswa tersebut dinyatakan lulus, sebaliknya jika nilai siswa tersebut di bawah 60 maka tidak lulus, dan seterusnya.

Kondisi merupakan hal yang menentukan didalam mengambil keputusan mengenai tindakan yang diambil. Didalam pemrograman kondisi dikenal dalam bentuk IF, CASE ataupun yang setara. Ada 2 (dua) bentuk umum dari penggunaan if yaitu if – then dan if – then – else.

1. Pernyataan If - then

Pada bentuk if - then, pernyataan hanya akan dijalankan kalau kondisi bernilai benar. Yang terpenting dari bagan alur diatas adalah kondisinya. Kodisilah yang menentukan apakah sebuah pernyataan program akan dieksekusi atau tidak

- Bentuk algoritma untuk pernyataan **if - then** adalah sebagai berikut :



Penggambaran FLOWCHART bebas

Cara-Kerja

Bila nilai cond

- TRUE, maka kerjakan semua instruksi yang ada dalam statements-true
Setelah selesai, lanjutkan ke next-instruction
- FALSE, maka langsung ‘meloncat’ mengerjakan isntruksi yang ada di next-instruction

2. Pernyataan If – then – else

Pernyataan **if – then – else** digunakan untuk menguji sebuah kondisi. Bila kondisi yang diuji terpenuhi, program akan menjalankan pernyataan-pernyataan tertentu; dan bila kondisi yang diuji salah, program akan menjalankan pernyatan-pernyataan yang lain.

- Bentuk umum pernyataan if – then – else dalam algoritma adalah sebagai berikut :



```

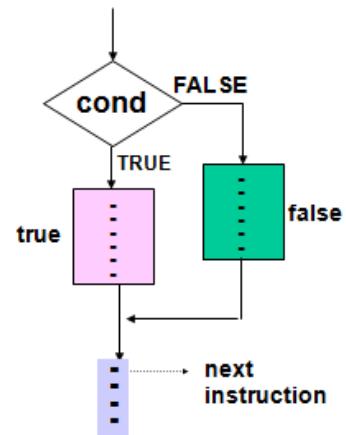
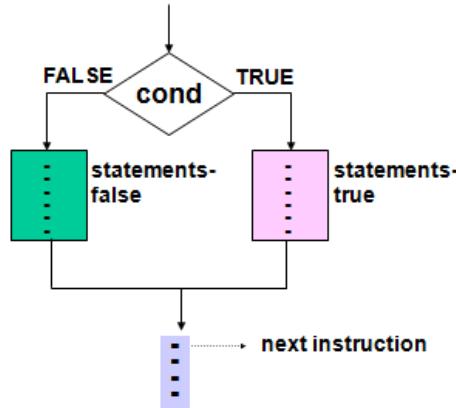
if( cond )
{
    statements-true
}

else
{
    statements-false
}

```

next instruction

Flowchart



Cara-Kerja

Bila nilai cond

- TRUE, maka kerjakan semua instruksi yang ada dalam statements-true
Setelah selesai, lanjutkan ke next-instruction

- FALSE, maka kerjakan semua instruksi yang ada dalam statements-false
Setelah selesai, lanjutkan ke next-instruction

Penulisan/Penggunaan IF Statement

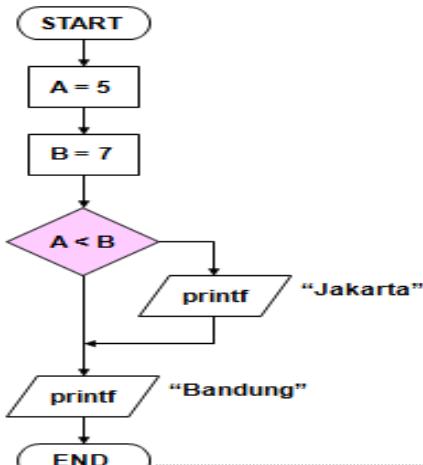
1. IF – THEN Statement

Contoh :

1

```
int A=5, B=7;
if(A<B)
{
    printf("Jakarta");
}
printf("\nBandung");
```

Tercetak : Jakarta
Bandung



2

```
int A=5, B=7;
if(A<B)
{
    printf("Jakarta");
}
printf("\nBandung");
```

Tercetak : Jakarta
Bandung

3

```
int A=5, B=7;
if(A<B)
{
    printf("Jakarta");
}
printf("\nBandung");
```

4

```
int A=5, B=7;
if(A<B)
    printf("Jakarta");
printf("\nBandung");
```

5

```
int A=5, B=7;
if(A<B)printf("Jakarta");
printf("\nBandung");
```

6

```
int A=5, B=7;
if(A<B)
    printf("Jakarta");
    printf("Bandung");
```

7

```
int A=5, B=7;
if(A<B)
printf("Jakarta");
printf("Bandung");
```

No. 6 dan 7 Cara menulis yang TIDAK BAIK
walaupun bagi komputer tidak ada bedanya dengan program nomor 1,2,3,dan4

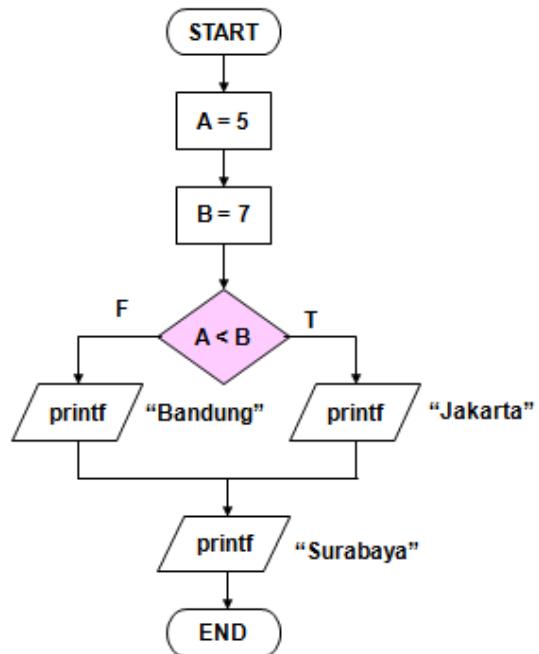
2. IF – THEN – ELSE Statement

Contoh :

1

```
int A=5, B=7;
if(A<B)
{
    printf("Jakarta");
}
else
{
    printf ("Bandung");
}
printf ("\nSurabaya");
```

Tercetak: Jakarta
Surabaya



2

```
int A=5, B=7;
if(A<B)
{
    printf ("Jakarta");
}
else
{
    printf ("Bandung");
}
printf ("\nSurabaya");
```

3

```
int A=5, B=7;
if(A<B)
{
    printf ("Jakarta");
}
else
{
    printf ("Bandung");
}
printf ("\nSurabaya");
```

4

```
int A=5, B=7;
if(A<B)
    printf("Jakarta");
else
    printf("Bandung");
printf("\nSurabaya");
```

5

```
int A=5, B=7;
if(A<B) { printf("Jakarta"); }
else { printf("Bandung"); }
printf("\nSurabaya");
```

6

```
int A=5, B=7;
if(A<B)
    printf("Jakarta");
else
    printf("Bandung");
    printf("\nSurabaya");
```

6

```
int A=5, B=7;
if(A<B)
    printf("Jakarta");
else
    printf("Bandung");
    printf("\nSurabaya");
```

No. 6 Cara menulis yang TIDAK BAIK
walaupun bagi komputer tidak ada bedanya dengan program nomor 1,2,3,dan4

Beberapa contoh penulisan / penggunaan if(cond) bentuk IF-THEN-ELSE

1

```
int A=5, B=7;
if(A<B)
{
    printf("Jakarta");
}
else
{
    printf("Bandung");
}
printf("\nSelesai");
```

Tercetak: Jakarta
Selesai

2

```
int A=5, B=7;
if(A<B)
{ printf("Jakarta");
}
else
{ printf("Bandung");
}
printf("\nSelesai");
```

Tercetak: Jakarta
Selesai

3

```
int A=5, B=7;
if(A<B)
{ printf("Jakarta");
}
else
{ printf("Bandung");
}
printf("\nSelesai");
```

Tercetak: Jakarta
Selesai

4

```
int A=5, B=7;
if(A<B){ printf("Jakarta");
}
else { printf("Bandung");
}
printf("\nSelesai");
```

Tercetak: Jakarta
Selesai

5

```
int A=5, B=7;
if(A<B)
    printf("Jakarta");
else
    printf("Bandung");
printf("\nSelesai");
```

Tercetak: Jakarta
Selesai

6

```
int A=5, B=7;
if(A<B)printf("Jakarta");
else printf("Bandung");
printf("\nSelesai");
```

Tercetak: Jakarta
Selesai

Catatan : Bila instruksi yang ada dalam kelompok statements-true atau dalam kelompok statements-false hanya terdiri dari **satu instruksi**, maka instruksi tersebut boleh tidak diapit oleh kurung buka “{“ dan kurung tutup “}” atau dengan perkataan lain : bila tidak menggunakan (atau tidak diapit oleh) kurung buka “{“ dan kurung tutup “}”, berarti kelompok statements-true atau false, hanya terdiri dari satu instruksi.

7.

```
int A=5, B=7;  
if(A<B) printf("Jakarta"); else printf("Bandung");  
printf("\n Selesai");
```

Tercetak: Jakarta
Selesai

8.

```
int A=5, B=7;  
if(A<B) printf("Jakarta"); else printf("Bandung"); printf("\n Selesai");
```

Tercetak: Jakarta
Selesai

9.

```
int A=5, B=7; if(A<B) printf("Jakarta"); else printf("Bandung"); printf("\n Selesai");
```

Tercetak: Jakarta
Selesai

10.

```
int A=5, B=7;  
if(A<B)  
    printf("Jakarta");  
else  
    printf("Bandung");  
printf("\n Selesai");
```

Tercetak: Jakarta
Selesai

Walaupun ditulis satu garis vertikal dengan
printf("Bandung");,
tapi bukan merupakan milik statements-else
karena milik statemts-else hanya satu instruksi
yaitu printf("Bandung");

11.

```
int A=5, B=7;  
if(A<B)  
    printf("Jakarta");  
    printf("Bogor");  
else  
    printf("Bandung");  
printf("\n Selesai");
```

error

Error,
karena printf("Jakarta"); tidak diapit oleh
tanda kurung {" dan "}, maka compiler
menganggap statements-true hanya terdiri
dari satu instruksi,
maka instruksi printf("Bogor"); dianggap
sebagai next instruktion
sehingga instruksi else tak punya hubungannya
dengan instruksi if diatasnya.
else yang berdiri sendiri, menyebabkan error

Tugas Mandiri

Soal-01.

Tulis program (penggalan program) dalam bahasa C untuk menyatakan algoritma yang digambarkan oleh flowchart Gambar-1 dan Gambar-2. :

Soal-02.

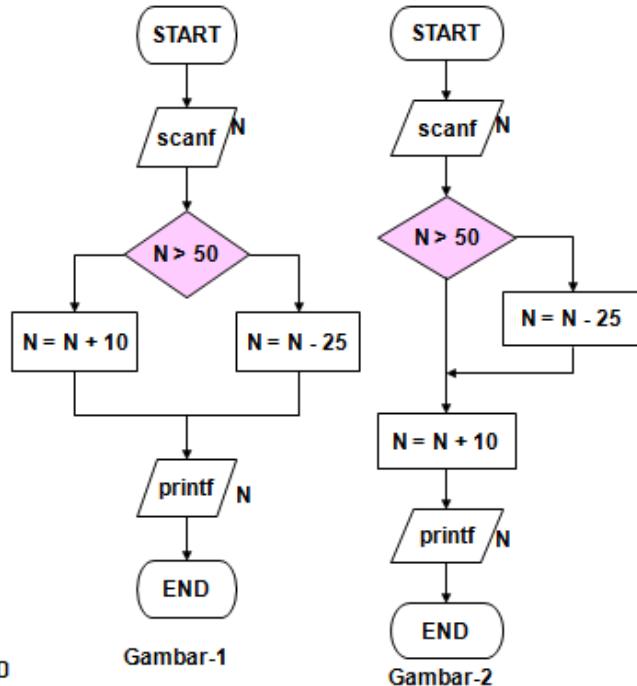
Untuk program yang algoritmanya digambarkan dengan Flowchart Gambar-1, apa yang tercetak bila untuk nilai N diinputkan nilai :

- a. 30
- b. 50
- c. 65

Soal-03.

Untuk program yang algoritmanya digambarkan dengan Flowchart Gambar-2 , apa yang tercetak bila untuk nilai N diinputkan nilai :

- a. 30
- b. 50
- c. 65



Soal-04

Susun program untuk menginput sebuah nilai integer (nilai ujian mahasiswa) kemudian cetak perkataan “LULUS”, bila nilai tersebut ≥ 60 atau cetak perkataan “GAGAL” bila nilai tersebut < 60 .

Soal-05

Susun program untuk menginput sebuah nilai integer, bilangan bulat positip lebih besar dari nol, kemudian cetak perkataan “EVEN”, bila bilangan tersebut merupakan bilangan GENAP, sebaliknya cetak perkataan “ODD” bila bilangan tersebut merupakan bilangan GANJIL.

ooooooooooooOOOoooooooooooo

Daftar Pustaka

1. Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001
2. Jogyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993
3. Moh. Sjukani, 2009, Algoritma & Struktur Data 1 dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.
4. Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004.





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Operator || dan && dalam
Bahasa C, Proses konversi
dari multi conditions menjadi
nested if.

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

02

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Memberikan gambaran umum
mengenai operator logika and or

Kompetensi

Mahasiswa mampu memahami dasar
penggunaan operator || dan && pada

pada struktur kendali proses penyeleksian kondisi/percabangan.

Control Statement Nested **if** untuk memberikan pilihan alternatif bagian algoritma yang dilaksanakan atau yang tidak dilaksanakan.

Operator Logika

Operator logika adalah operator yang digunakan untuk membandingkan dua buah nilai logika. Nilai logika adalah nilai benar atau salah. Jika sebelumnya pada operator relasi yang dibandingkan adalah nilai dari data apakah benar (1) atau salah (0), maka pada operator logika bisa dikatakan yang dibandingkan adalah logika hasil dari operator relasi. Operator-operator logika dan hasil perbandingan logika yang disediakan pada bahasa C disajikan pada table-tabel dibawah ini.

Tabel Operator Logika dalam Bahasa C++

Operator	Fungsi	Contoh	Hasil	Penjelasan
!	NOT	$!(5 > 4)$	0 (salah)	Fungsi NOT digunakan untuk membalik hasil logika.
&&	AND	$(3 \geq 3) \&\& (2 != (4/2))$	0 (salah)	Fungsi AND akan memberikan nilai benar (1) jika kedua nilai logika yang dibandingkan bernilai benar (1).
	OR	$(3 \geq 3) (2 != (4/2))$	1(benar)	Fungsi OR akan memberikan nilai benar (1) jika salah satu nilai logika yang dibandingkan bernilai benar (1).

Tabel Operator Logika dalam Bahasa C

A	B	!B	A && B	A B
1	1	0	1	1
1	0	1	0	1
0	1	0	0	1
0	0	1	0	0

Catatan:
1 → True / benar
0 → False/ salah

Listing Program Operator Logika

```
#include <iostream.h>

main()
{
    cout<< !(5 > 4);
    cout<< (3 >= 3) && (2 != (4/2));
    cout<< (3 >= 3) || (2 != (4/2));
```

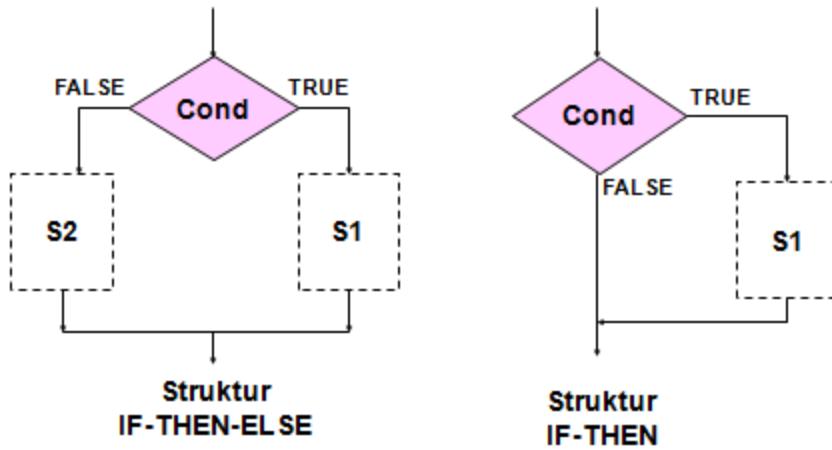
0
0
1

Control Statement IF (Nested IF)

1. Pernyataan Nested IF

Dalam kasus-kasus tertentu, sebuah konstruksi **if** dan **else** dapat berbentuk bersarang (nested). Bentuk bersarang merupakan contoh bentuk dari sebuah atau beberapa buah pernyataan if dan else yang terdapat didalam bentuk if dan else yang lainnya. Nested IF dan ELSE merupakan pernyataan yang lebih rumit dibandingkan dengan pernyataan IF dan ELSE sederhana.

Perhatikan kembali struktur IF-THEN-ELSE dan IF-THEN Statement seperti yang sudah diterangkan sebagai berikut :



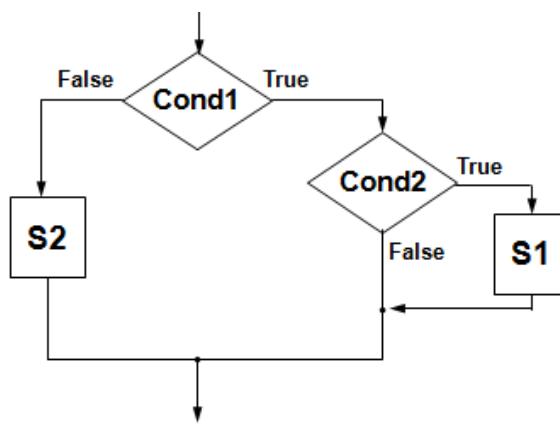
Dari ilustrasi struktur diatas, S adalah satu atau sekelompok statement. Didalam kelompok S mungkin terdapat statement IF sehingga terjadi IF secara berjenjang atau secara bersarang yang biasa disebut Nested If (nest = sarang).

Bentuk umum pernyataan IF bersarang (Nested IF) dalam algoritma dan flowchart

1).

```
if (cond1)
{ if (cond2)
  {
    -
    - S1
    -
  }
else {
  -
  - S2
  -
}
```

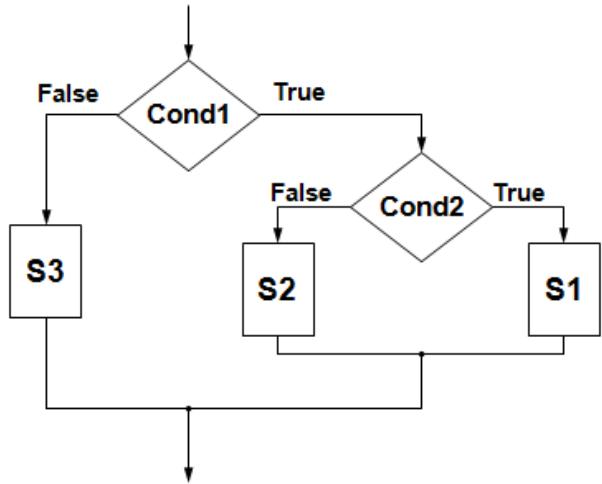
2014



```

2)
if (cond1)
{if (cond2)
 {
 - s1
 -
 }
else {
 - s2
 -
 }
}
else {
 - s3
 -
 }

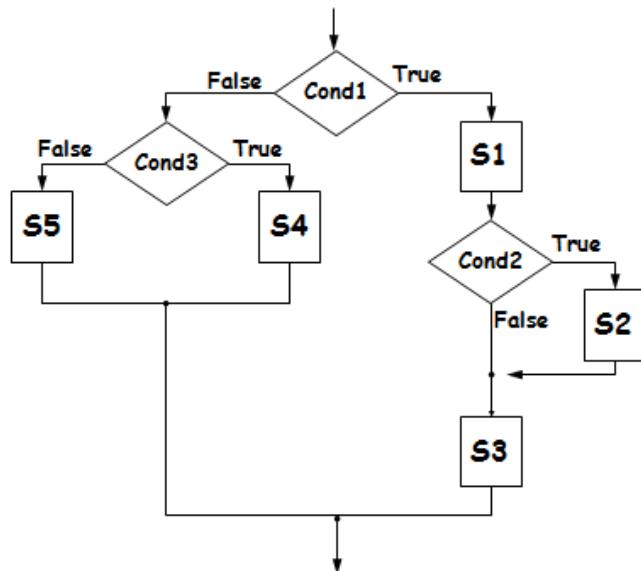
```



```

3)
if cond1
{
 - s1
 if cond2
 {
 - s2
 -
 }
 - s3
}
else
{
if cond3
{
 - s4
 -
}
else
{
 - s5
}
}

```

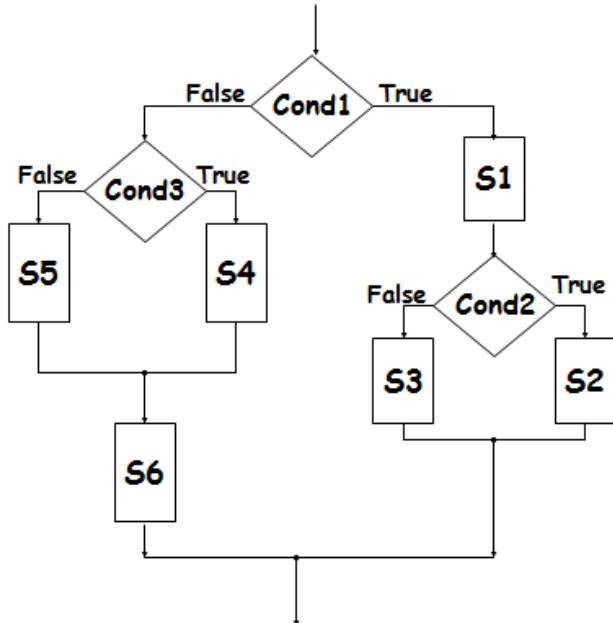


Perhatikan posisi letak 'titik' End if (akhir fungsi if) dalam flowchart. Posisi ini penting untuk menganalisa aliran terutama untuk nested IF yang komplek atau untuk proses pengulangan yang bersifat rekursif.

```

4) if (cond1)
  {
    - s1
    -
    if (cond2)
    {
      - s2
      -
    }
    else {
      - s3
      -
    }
  }
  else
  {
    if (cond3)
    {
      - s4
      -
    }
    else
    {
      - s5
      -
    }
    -
    - s6
  }
}

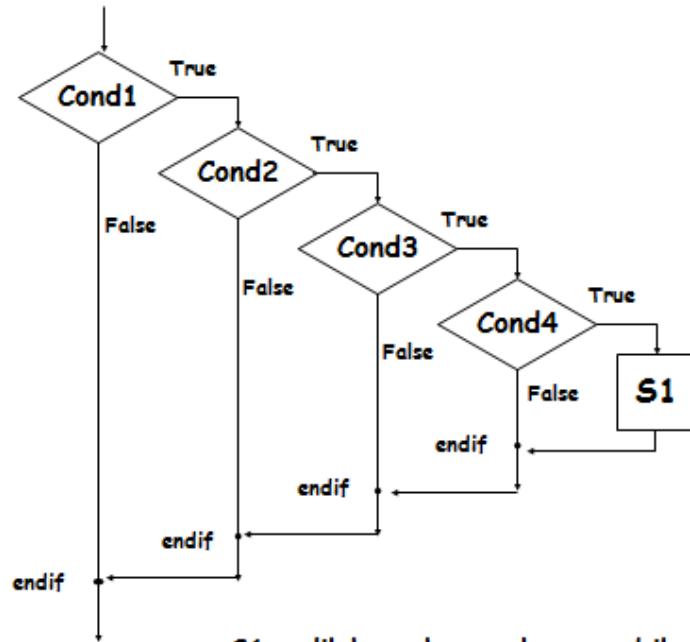
```



```

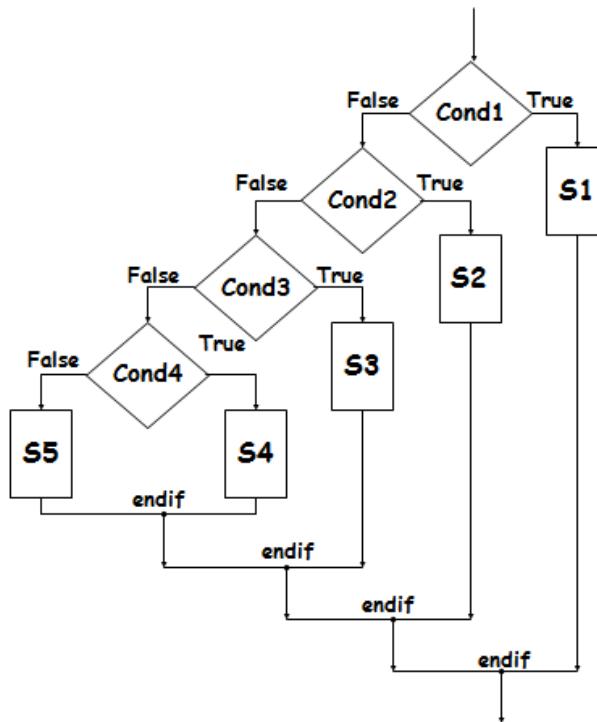
5) if(cond1)
  {
    if(cond2)
      {
        if(cond3)
          {
            if(cond4)
              {
                -
                - s1
                -
              }
            }
          }
    }
  }
}

```



S1 dilaksanakan, hanya bila keempat kondisi nilainya TRUE

```
6) if(cond1)
    {
        -
        - s1
        -
    }
else
    {if(cond2)
        {
            -
            - s2
            -
        }
    else
        {if(cond3)
            {
                -
                - s3
                -
            }
        else
            {if(cond3)
                {
                    -
                    - s4
                    -
                }
            else
                {
                    -
                    - s5
                    -
                }
            }
        }
    }
}
```



Contoh soal :

Buat program untuk menginput Jumlah buku yang dibeli. Kemudian cetak Harga yang harus dibayar, jika berlaku ketentuan sebagai berikut :

- Harga Satuan = Rp. 2.500,-
 - Harga buku = Harga Satuan * Jumlah buku
 - Potongan dihitung berdasarkan :
 - Jika Jumlah buku ≥ 50 , maka potongan 20% dari Harga buku
 - Jika Jumlah buku ≥ 35 , maka potongan 15% dari Harga buku
 - Jika Jumlah buku ≥ 20 , maka potongan 10% dari Harga buku, selain itu tidak mendapatkan potongan.
 - Harga yang harus dibayar = Harga buku – potongan.

Jawab :

```
#include <iostream.h>

void main(void)
{
    int Jumlah_Buku, Harsat=2500, Harga_buku=0, Pot=0;
    float harga;
    cout<<"Jumlah Buku : ";
    cin>>Jumlah_Buku;
    Harga_buku=Jumlah_Buku*Harsat;

    if (Jumlah_Buku >= 50)
    {
        Pot= 0.2 * Harga_buku;
    }
    else
        if (Jumlah_Buku >= 35)
        {
            Pot= 0.15 * Harga_buku;
        }
        else
            if (Jumlah_Buku >= 20)
            {
                Pot= 0.1 * Harga_buku;
            }
            else
            {
                Pot=0;
            }
    Harga=Harga_buku-Pot;
    cout<<"Total Harga :"<<Harga;
}
```

2.

Pernyataan **switch-case** digunakan untuk menyederhanakan struktur pemilihan dengan banyak kondisi.

➤ Bentuk umum dalam bahasa C++ :

```
switch (<ekspresi>)
{
    case <nilai_1>: statemen_1;
                      break;
    .....
    case <nilai_n> : statemen_n;
                      break;
    default:         statemen_default;
}
```

- <nilai> pada case hanya dapat diisi oleh nilai yang “*integral constants*”
- case dengan break: digunakan jika satu kondisi hanya mengerjakan satu kelompok statemen.
- case tanpa break: digunakan jika suatu case dapat mengerjakan satemen milik case yang ada di bawahnya.

Statemen **switch** akan menyeleksi kondisi yang diberikan dan kemudian membandingkan hasilnya dengan nilai-nilai yang berada dalam **case**. Pembandingan akan dimulai dari nilai1 sampai dengan nilai_n. Jika hasil dari kondisi sama dengan nilai tertentu, maka statemen-statemen yang ada di **case** nilai tersebut akan diproses sampai ditemui statemen **break** yang akan membawa proses keluar dari blok **switch**. Jika semua nilai yang dibandingkan tidak ada yang sama dengan kondisi, maka statemen yang akan diproses adalah statemen yang berada dalam **default**.

Statemen **break** digunakan untuk membuat proses keluar dari blok **switch** setelah selesai memproses sebuah **case**. Jika statemen **break** tidak digunakan, maka apabila proses sudah selesai memproses sebuah bagian dalam **case** proses akan masuk ke bagian **case** yang berikutnya.

Pernyataan **switch** digunakan untuk menyederhanakan konstruksi **if-else-if** yang terlalu banyak. Sebagai contoh penggalan program berikut:

```
if (x == 0) cout<<"X bernilai 0";
else if (x == 1) cout<<"X bernilai 1";
else if (x == 2) cout<<"X bernilai 2";
else if (x == 3) cout<<"X bernilai 3";
else cout<<"X tidak bernilai 0, 1, 2, ataupun 3";
```

Dapat diganti menjadi lebih ringkas dan mudah dibaca dengan program berikut:

```
switch(x)
{
    case 0 : cout<<"X bernilai 0"; break;
    case 1 : cout<<"X bernilai 1"; break;
    case 2 : cout<<"X bernilai 2"; break;
    case 3 : cout<<"X bernilai 3"; break;
    default: cout<<"X tidak bernilai 0, 1, 2, ataupun 3";
}
```

Jika dilihat pada contoh di atas bahwa meskipun hasil dari kedua program tersebut sama tetapi penulisan dengan case lebih mudah dibaca.

Contoh soal:

Tulislah algoritma dan program yang meminta masukan bilangan bulat dari pengguna. Jika pengguna memasukkan 1, program menampilkan 'Minggu'; jika pengguna memasukkan 2, program menampilkan 'Senin', dan seterusnya sampai dengan 'Sabtu'. Jika pengguna memasukkan nilai di luar jangkauan 1 sampai dengan 7, program menuliskan 'Hari tidak Valid'

Jawab:

Algoritma untuk menampilkan hari dapat dituliskan sebagai berikut:

1. Masukkan x.
2. Jika ($x = 1$) Tulis 'Minggu'. Selesai.
3. Jika ($x = 2$) Tulis 'Senin'. Selesai.
4. Jika ($x = 3$) Tulis 'Selasa'. Selesai.
5. Jika ($x = 4$) Tulis 'Rabu'. Selesai.
6. Jika ($x = 5$) Tulis 'Kamis'. Selesai.
7. Jika ($x = 6$) Tulis 'Jumat'. Selesai.
8. Jika ($x = 7$) Tulis 'Sabtu'. Selesai.
9. Tulis 'Hari tidak Valid'

Jika ditulis dengan menggunakan *Nested IF* sebagai berikut:

```
#include<iostream.h>
main()
{
    int X;
    cin>>X;
    if (X==1)
        cout<<"Minggu";
    else
        if (X==2)
            cout<<"Senin";
        else
            if (X==3)
                cout<<"Selasa";
            else
                if (X==4)
                    cout<<"Rabu";
                else
                    if (X==5)
                        cout<<"Kamis";
                    else
                        if (X==6)
                            cout<<"Jumat";
                        if (X==7)
                            cout<<"Sabtu";
                        else cout<<"Hari Tdk Valid";
    } } } } } }
```

Jika ditulis dengan menggunakan **switch-case** sebagai berikut :

```
#include <iostream.h>

void main(void)
{
    unsigned int x; /* deklarasi variabel x */

    cout<<"Masukkan bilangan bulat (1 - 7) : ";
    cin>>x;

    switch (x)
    {
        case 1: cout<<"Minggu"; break;
        case 2: cout<<"Senin"; break;
        case 3: cout<<"Selasa"; break;
        case 4: cout<<"Rabu"; break;
        case 5: cout<<"Kamis"; break;
        case 6: cout<<"Jumat"; break;
        case 7: cout<<"Sabtu"; break;
        default : cout<<"Hari tidak Valid";
    }
}
```

Masukkan bilangan bulat (1 – 7) : 5

Kamis

Masukkan bilangan bulat (1 – 7) : 8

Hari tidak Valid

Tugas Mandiri

1. Susun program untuk menginput tiga buah bilangan yang menyatakan nilai ujian tiga buah mata kuliah. Cetak perkataan “LULUS” bila nilai rata-rata ketiga buah bilangan tersebut lebih besar atau sama dengan 60 atau, walaupun nilai rata-rata lebih kecil dari 60, tapi bila ada salah satu mata kuliah yang nilainya lebih besar atau sama dengan 80 maka kategorinya juga dinyatakan lulus dan cetak perkataan “LULUS”. Bila syarat diatas tidak terpenuhi maka cetak perkataan “TIDAK LULUS”.
2. Susun program untuk menginput tiga buah bilangan yang menyatakan nilai ujian tiga buah mata kuliah. Cetak perkataan “LULUS” bila nilai rata-rata ketiga buah bilangan tersebut lebih besar atau sama dengan 60. Bila nilai rata-rata lebih kecil dari 60, maka cetak perkataan “TIDAK LULUS”.

ooooooooooooOOOoooooooooooo

Daftar Pustaka

5. Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001
6. Jogyianto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993
7. Moh. Sjukani, 2009, Algoritma & Struktur Data 1 dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.
8. Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004.





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Struktur perulangan dengan
for, do-while dan while

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

03

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Memberikan gambaran umum mengenai statemen perulangan yang dapat digunakan untuk membantu dalam memecahkan masalah.

Kompetensi

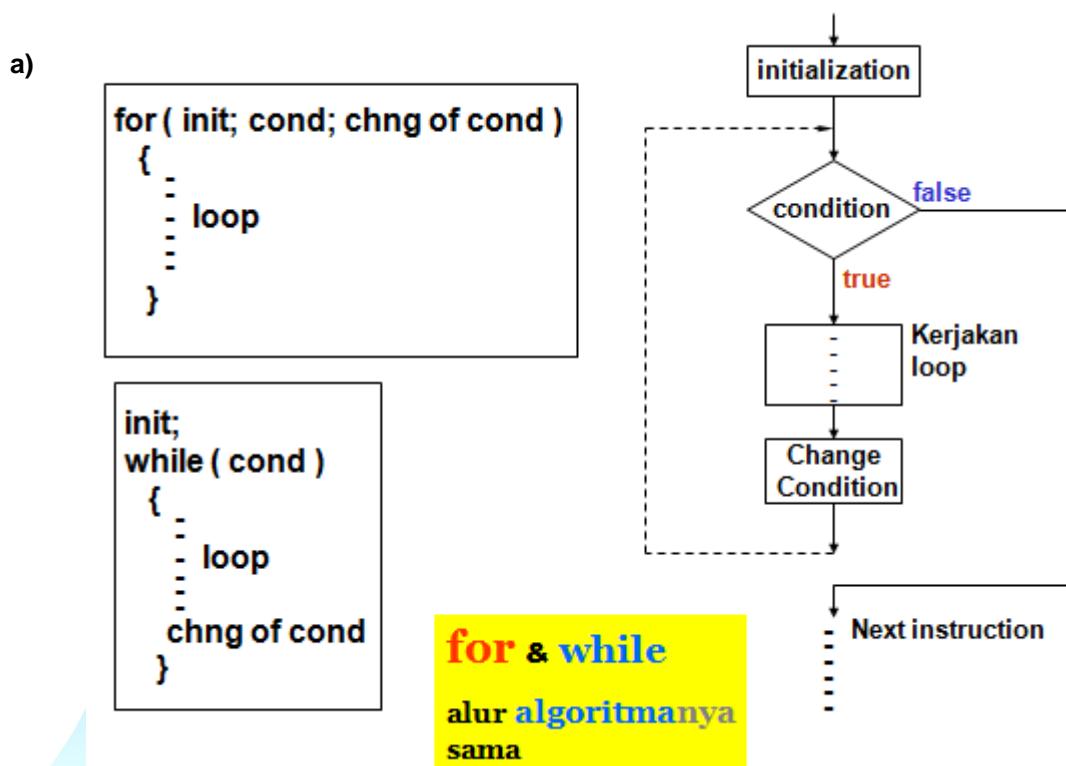
Mahasiswa mampu memahami dasar penggunaan statemen-statemen perulangan, dan dapat memilih statemen perulangan yang tepat untuk membantu memecahkan masalah.

Penggunaan Looping (Perulangan)

Pengulangan digunakan untuk menjalankan satu atau beberapa pernyataan sebanyak beberapa kali. Proses yang berulang adalah suatu urut-urutan pernyataan yang akan dieksekusi terus menerus selama kondisi yang disyaratkan terpenuhi. Pengulangan proses merupakan suatu kemampuan yang dimiliki oleh semua compiler bahasa pemrograman. Terdapat banyak jenis pengulangan proses, tetapi paling tidak akan dibahas dua buah bentuk jenis pengulangan :

1. **FOR.** Pengulangan dengan besarnya nilai integer sebagai kondisi (biasanya berbentuk for)
2. **WHILE.** Pengulangan dengan kondisi pernyataan boolean (biasanya berbentuk while).

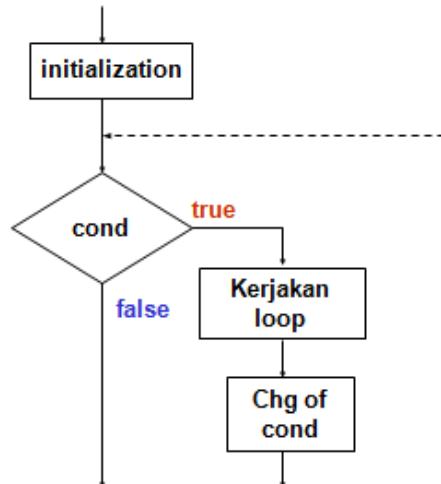
Bentuk umum penulisan dan flowchart dari perulangan adalah sebagai berikut :



b)

```
for( init; cond; chng of cond )  
{  
    :- loop  
}
```

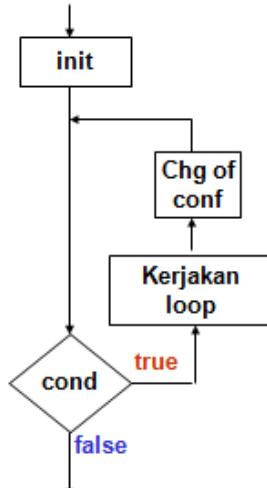
```
init;  
while ( cond )  
{  
    :- loop  
    chng of cond  
}
```



c)

```
for ( init; cond; chng of cond )  
{  
    :- loop  
}
```

```
init;  
while ( cond )  
{  
    :- loop  
    chng of cond  
}
```



cond = condition

Suatu pernyataan yang mengandung nilai BENAR (true) atau SALAH (False)

Chng of cond =

Change of condition

Suatu instruksi yang dapat mempengaruhi nilai condition. Pada proses yang normal, perubahan nilai disini suatu saat akan membuat nilai condition = false

1. Pernyataan for () dan while ()

Pernyataan dengan **for** biasanya digunakan untuk pernyataan yang sudah ditentukan jumlah pengulangannya.

Pernyataan **for** dengan kondisi numerik yang menaik (*increment*). Pada looping yang berbentuk seperti ini terdapat pernyataan *increment* atau penambahan yang berbentuk sebagai berikut : Kondisi = Kondisi + n, dimana n adalah bilangan yang bernilai positif, sebagai contoh pernyataan berikut: **j = j + 1;** atau **j++;**

Contoh 1 : Pernyataan for() dan while()

```
#include<iostream.h>
void main()
{int I;
 for(I=1; I<=5; I=I+1)
 {
    cout<<"Jakarta";
 }
 cout<<"Selesai";
}
```

```
#include<stdio.h>
void main()
{int I;
 I=1;
 while( I<=5 )
 {
    cout<<"Jakarta";
    I = I + 1;
 }
 cout<<"Selesai";
}
```

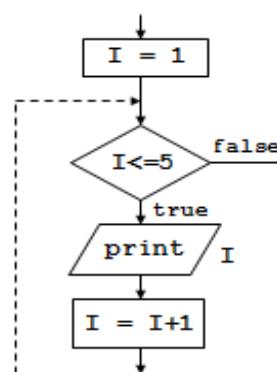
Bila program disamping dijalankan, (RUN) maka akan mencetak:

Jakarta
Jakarta
Jakarta
Jakarta
Jakarta
Selesai

Perhatikanlah didalam pengulangan dengan **for**, pernyataan yang berada didalam loop akan dieksekusi sebanyak 5x (lima kali) sampai kondisi loop tersebut tidak terpenuhi, baru mengerjakan perintah yang berada di bawahnya.

```
for()
#include<stdio.h>
void main()
{ int I;
 for(I=1; I<=5; I=I+1)
 {
    printf("\n %i", I );
 }
```

Tercetak :
1
2
3
4
5



Perkembangan nilai I

2014	nilai I	Kondisi I <= 5	Tercetak oleh printf I	Oleh I=I+1 nilai I menjadi:
	1	True	1	2
	2	True	2	3
	3	True	3	4

Contoh 2 :

<pre>for() { #include<stdio.h> void main() { int I, N; N = 8; for(I=1; I<=5; I=I+1) { printf("\n %i", N); N = N + 2; } }</pre>	<pre>while() { #include<stdio.h> void main() { int I, N; N = 8; I = 1; while(I<=5) { printf("\n %i", N); N = N + 2; I=I+1; } }</pre>
--	--

Perkembangan nilai I

N	nilai I	Kondisi I <= 5	Tercetak oleh printf N	Oleh N=N+2 nilai N menjadi:	Oleh I=I+1 nilai I menjadi:
8	1	True	8	10	2
10	2	True	10	12	3
12	3	True	12	14	4
14	4	True	14	16	5
16	5	True	16	18	6
18	6	False	Keluar dari loop		

Catatan:

- Nilai untuk perulangan dalam for tidak dibenarkan bertipe float.
- Di akhir penulisan statemen for tidak menggunakan tanda titik koma (;)
contoh: `for (int I = 1 ; I < = 5 ; I = I+1);` ← tidak dibenarkan
- Penggunaan tanda titik koma setelah statemen for tidak menimbulkan pesan kesalahan tetapi akan menghasilkan keluaran yang tidak sesuai karena statemen for tersebut tidak dianggap pernyataan perulangan, hanya akan dinyatakan sebagai pernyataan biasa (perhatikan listing program dibawah ini).

Listing Program contoh for dengan titik koma

```
#include <iostream.h>

main()
{
    for(int j = 1; j <= 4; j= j+1);
    {
        cout<<j<<endl;
    }
}
```

Menggunakan tanda titik koma

5

Hasil yang di dapat adalah 5, karena hanya nilai **j** akhir yaitu 5 yang tercetak, sedang saat **j** bernilai 1 hingga 4 tidak tercetak.

2. Penyataan do-while

Pernyataan pengulangan **do-while** hampir sama dengan pernyataan pengulangan **while**, dan biasanya digunakan bila jumlah pengulangan belum dapat ditentukan pada saat program ditulis.

Struktur **do-while** digunakan untuk melaksanakan (*do*) statemen atau blok statemen selama (*while*) kondisi yang diseleksi pada **while** tidak terpenuhi. Sintaks dari struktur ini adalah :

➤ Bentuk umum pernyataan **do-while** dalam bahasa C adalah sebagai berikut:

```
do {
    instruksi;
}
while(kondisi);
```

Catatan:

- Perbedaan pernyataan **do-while** dan **while** terletak pada pengecekan kondisi. Jika pada pernyataan **while**, kondisi dicek pada **awal** blok, pada pernyataan **do-while**, kondisi dicek pada **akhir** blok.
- struktur **do-while** paling tidak akan mengerjakan satu kali instruksi meskipun kondisi tidak terpenuhi karena pengecekan kondisi dilakukan di akhir blok.

- Contoh dari pernyataan **do-while** dalam bahasa C++ diperlihatkan pada program dibawah ini:

Listing Program Contoh do-while1

```
#include <iostream.h>

main()
{
    int j;
    j = 1;
    do {                                // akan dikerjakan statemen-statemen di bawah ini
        cout<<j;
        j = j + 1;
    }
    while(j < 10); /* Pengecekan kondisi, jika masih terpenuhi maka perulangan akan
                     dilanjutkan, jika tidak keluar dari blok perulangan */
}
```

1 2 3 4 5 6 7 8 9

Hasil yang didapat dari dua listing program tidak berbeda, meskipun menggunakan jenis perulangan yang berbeda, lalu pada saat kapan kita harus memilih statemen do-while? Untuk menjawab pertanyaan tersebut lihatlah listing program di bawah ini.

Listing Program Contoh do-while2

```
#include <iostream.h>

main()
{
    int j;
    j = 1;
    do {
        cout<<j;
        j = j + 1;
    } while(j > 10);
}
```

1

Perhatikan listing diatas, jika dilihat hasil yang didapat dari program tersebut hanya menampilkan nilai **j** awal yaitu 1. Jika dilihat kondisi yang membatasi adalah (**j > 10**) sehingga saat dilakukan pengecekan ternyata nilai **j** tidak memenuhi kondisi, perulangan selesai dan program keluar dari blok **while**, tetapi meskipun kondisi tidak terpenuhi pencetakan nilai **j** telah dikerjakan.

Jadi kita dapat menggunakan statemen **do-while** jika kita menginginkan proses perulangan dilakukan minimal satu kali.

3. Penyataan break dan continue

Pernyataan **break** akan menyebabkan proses keluar dari statemen berikut ini: **for**, **while**, **do-while**, **switch**.

Pernyataan **continue** menyebabkan proses mengabaikan instruksi selanjutnya dan melanjutkan proses iterasi berikutnya pada statemen **for**, **while**, **do-while**.

- Contoh dari penggunaan **break** dalam statemen **for** diperlihatkan pada program dibawah ini :

Listing Program Penggunaan **break** dalam **for**.

```
#include <iostream.h>

main()
{
    for(int v = 1; v <= 10; v++)
    {   if(v==5)
        break;
        cout<<v;
    }
}
```

Perintah ini akan menyebabkan proses menghentikan perulangan pada saat **v** bernilai sama dengan 5. Sehingga program hanya mencetak nilai 1 hingga 4.

- Contoh dari penggunaan **continue** dalam statemen **for** diperlihatkan pada program dibawah ini :

Listing Program 8.7 Penggunaan **break** dalam **for**.

```
#include <iostream.h>

main()
{
    for(int v=1; v <= 10; v++)
    {
        if(v == 5)
            continue;
        cout<<v;
    }
}
```

Perintah ini akan menyebabkan proses melewati proses perulangan pada saat **v** bernilai sama dengan 5, tetapi kemudian proses melanjutkan ke iterasi berikutnya. Sehingga program hanya mencetak nilai 1 hingga 4 dan nilai 6 hingga 10.

Tugas Mandiri

1. Susun program untuk mencetak 10 suku pertama deret berikut ini :
 - a. 1, 2, 3, 4, 5, ..., ...
 - b. 1, 3, 5, 7, 9, ..., ...
 - c. 1, 2, 4, 8, 16, ..., ...
 - d. 5, 8, 11, 14, 17, ..., ...
 - e. 5, 8, 13, 20, 29, ..., ...
2. Susun program untuk menginput 4 buah bilangan yang merupakan nilai ujian mahasiswa, kemudian cetak nilai terbesar yang didapat mahasiswa .
3. Seseorang menyimpan uang Rp. 1.000.000 di bank dengan bunga ber-bunga 2% perbulan. Jadi setelah satu bulan uangnya menjadi Rp. 1.020.000. Satu bulan berikutnya uang Rp. 1.020.000 ini mendapat bunga lagi 2%, yaitu Rp.20.400 sehingga setelah 2 bulan uangnya menjadi Rp. $1.020.000 + \text{Rp. } 20.400 = \text{Rp. } 1.040.400$. Demikian seterusnya (bunga bulan ini ditambahkan ke saldo uangnya dan mendapatkan bunga lagi pada bulan berikutnya) . Susun program untuk menghitung dan mencetak jumlah uangnya setelah 10 bulan.

Bu- lan ke-	Jumlah uang		
	Pada awal bulan ke-l	Bunga 2%	Pada akhir bulan ke-l
I	U	$B=0.02*U$	$U=U+B$
1	1.000.000	20.000	1.020.000
2	1.020.000	20.400	1.040.400
3	1.040.400	20.808	1.061.208
4	1.061.208	xxxxxx	xxxxxxxxxx
--	-----	-----	-----
--	-----	-----	-----
--	-----	-----	-----
10	xxxxxxxxxx	xxxxxx	xxxxxxxxxxxx

ini yang
dicetak

4. Ses kecepatan tetap 2 m/det. Susun program untuk mencetak berapa m yang dia tempuh setelah berjalan selama 100 detik.
5. Buat program seperti tampilan dibawah ini

2014	9	Dasar Pemrograman	Nilai x	Nilai Kuadrat
		Tim Dosen.	---	---
			1	1
			2	4
			3	9
		

Daftar Pustaka

9. Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001
10. Jogyianto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993
11. Moh. Sjukani, 2009, Algoritma & Struktur Data 1 dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.
12. Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004.



MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Pengaplikasian struktur perulangan untuk memecahkan soal-soal matematika dan fisika

Fakultas

Ilmu Komputer

Program Studi

Sistem
Informasi

Tatap Muka

04

Kode MK

MK87042

Disusun Oleh

Tim Dosen

Abstract

Kompetensi

Memberikan gambaran umum mengenai struktur kendali proses penyeleksian kondisi/percabangan.

Mahasiswa mampu mengaplikasikan dasar penggunaan Control Statement if untuk memberikan pilihan alternatif jawaban dalam memecahkan persoalan matematika dan fisika.

Contoh soal Penggunaan Perulangan

1. Susun program untuk menginput 100 buah bilangan integer (nilai ujian mahasiswa) dan mencetak totalnya

Terlihat dilayar :

Ilustrasi data

Daftar Nilai Mahasiswa			
No.	NIM	Nama	Nilai
1.	xxxx	xxxxxxx	xxx
	1	7	7
100	xxxx	xxxxxxx	xxx

Nilai ke-1 : xxx
Nilai ke-2 : xxx
Nilai ke-3 : xxx
--
--
--
--

Nilai ke-100 : xxx

Total = xxxxx

Yang diinput hanya nilai ujiannya saja

Nomor urut dimunculkan oleh program



```
#include <stdio.h>
void main()
{ int I,N,T;
T = 0;
for( I=1; I <=100; I++ )
{ printf("Nilai ke-%i : ", I );
scanf(" %i", &N);
T = T + N;
}
printf("\n Total = : %i", T );
}
```

2. Susun program untuk menginput 100 buah bilangan (nilai ujian 100 orang mahasiswa) dan mencetak jumlah mahasiswa yang lulus. Dinyatakan lulus bila nilai >=60.

```
#include <stdio.h>
void main()
{ int I, N, Jum;
Jum = 0;
for(I=1; I <=100; I++)
{ printf("Nilai ke-%i : ", I );
scanf(" %i", &N);
if(N >= 60 )
{ Jum++; }
}
printf("\n Jumlah yang Lulus = : %i", Jum);
```

3. Dalam “Daftar Nilai Mahasiswa” terdapat nilai sejumlah mahasiswa. Jumlah mahasiswa tidak diketahui.
Susun program untuk menginput nilai-nilai ujian tersebut (hanya nilai ujinya saja) Proses input berakhir bila diinput nilai 999 sebagai End of Data, kemudian cetak jumlah mahasiswa, dan nilai Rata-rata yang didapat oleh mahasiswa.

Ilustrasi data

Nomor terakhir tidak tetap

Daftar Nilai Mahasiswa			
No.	NIM	Nama	Nilai
1.	xxxx	xxxxxxx	xxx
xx.	xxxx	xxxxxxx	xxx 999

→ 999 sebagai End Of Data
Bila nilai ini diinput, berarti data telah habis, dan proses input selesai

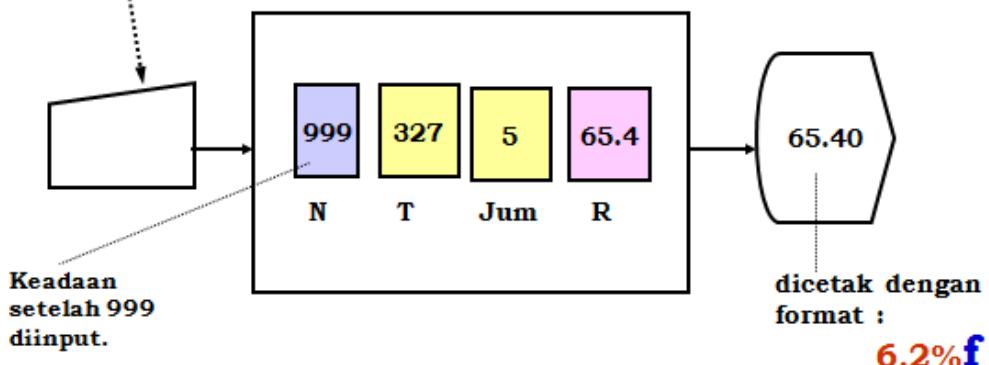
Ilustrasi data

60
50
70
65
77

Ada 5 buah data,
yang Totalnya =
 $60 + 50 + 70 + 65 + 77 = 327$
Nilai rata-rata = $327 / 5 = 65.4$

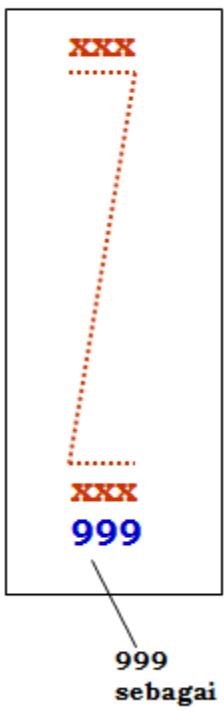
999 Hanya sebagai batas (end of data)
tidak ikut ditambah

Ada 6 kali input : 5 kali menginput data, dan
1 kali menginput end of data



JAWAB :

Ilustrasi input



Cara-1

```
#include <stdio.h>
void main()
{ int N, Jum;
  float T, R;
  T = 0; Jum = 0;
  scanf("%i", &N);
  while( N != 999)
  {
    T = T + N;
    Jum++;
    scanf("%i", &N);
  }
  R = T / Jum;
  printf("Jumlah Mahasiswa : %i", Jum );
  printf("\n");
  printf("Nilai Rata-rata : %6.2f",R );
}
```

yang umum digunakan

Input yang pertama dilakukan disini

Input selanjutnya berulang-ulang dilakukan disini sampai yang diinput = 999 baru keluar dari loop

Cara-2.

Untuk : **input, dan menghitung T, Jum dan R dibuat sebagai berikut :**

```
T = 0; Jum = 0;
```

```
N = 0;
while( N != 999)
{
  T = T + N;
  Jum++;
  scanf("%i", &N);
}
```

N sengaja diberi nilai awal = 0, agar pertama kali ($N \neq 999$) akan bernilai TRUE, sehingga paling tidak loop dikerjakan satu kali.

Pertama kali T ditambahkan dengan 0 (nol) sehingga tidak mempengaruhi nilai awal T.

```
R = T / Jum;
```

Cara-3. Sebagai tanda proses selesai, untuk keluar dari loop, digunakan variabel Flag = 1.

```
T = 0; Jum = 0;  
  
Flag = 0;  
while( Flag == 0 ) {  
    scanf("%i", &N);  
    if( N != 999 )  
        { T = T + N;  
        Jum++;  
        }  
    else  
        Flag = 1;  
}  
  
R = T / Jum;
```

Selama Flag == 0,
proses tetap
berlangsung

Bila yang
diinput = 999,
maka Flag
dibuat = 1 agar
condition pada
WHILE bernilai
False sehingga
keluar dari loop.

Cara-4. Sebagai tanda proses selesai,
untuk keluar dari loop, digunakan instruksi :
break.

```
T = 0; Jum =0;
```

```
while ( 1 )
{ scanf("%i", &N);
  if (N == 999)
    break;
  else
    { T = T + N;
      Jum++;
    }
}
```

```
R = T / Jum;
```

Perhatikan : while(1)

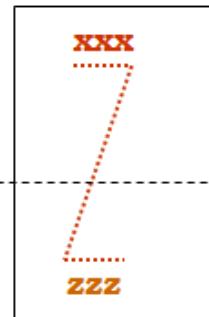
Dengan (1) sebagai
condition, maka
condition while() akan
selalu bernilai true,
sehingga selalu
mengerjakan loop.

Keluar dari loop hanya
oleh intruksi **break**;
yang terjadi hanya bila
nilai yang diinput =
999.

Lihat penggunaan **break**
pada contoh
sebelumnya

4. Dalam lembar dokumen tersedia banyak sekali data berupa bilangan-bilangan integer yang total nilainya dipastikan lebih dari 1000. Susun program untuk menginput hanya sebagian dari data tersebut, dan mencetak totalnya. Berapa buah bilangan yang diinput, atau berapa kali kita menginput tergantung total data yang telah diinput. Bila totalnya sudah melebihi dari 1000, maka berhenti menginput dan langsung mencetak total kemudian proses selesai.

Ilustrasi Data input



Menginput sampai disini
totalnya sudah > 1000,
proses input dan hitung
total selesai

Cara-1.

```
int N , T = 0;  
while( T <= 1000)  
{ scanf("%i", &N );  
    T = T + N;  
}  
printf("Total: %i", T);
```

Selama Total belum lebih besar dari 1000 proses input dan hitung total tetap berlangsung

Cara-2.

```
int N,T,Flag;  
T = 0; Flag = 0;  
while( Flag == 0)  
{ scanf("%i", &N );  
    T = T + N;  
    if(T > 1000)  
        { Flag = 1; }  
}  
printf("Total :%i", T);
```

Selama Flag sama dengan nol, proses input dan hitung total tetap berlangsung

Cara-3.

```
int N, T = 0;  
do  
{ scanf("%i", &N );  
    T = T + N;  
}  
while( T <= 1000);  
printf("Total: %i", T);  
}
```

Kerjakan input dan hitung Total selama Total belum lebih besar dari 1000

Cara-4.

```
int N, T = 0;  
while( 1 )  
{ scanf("%i", &N );  
    T = T + N;  
    if(T > 1000)  
        break;  
}  
printf("Total :%i", T);
```

Selama TRUE, laksanakan input dan hitung Total, Tapi bila Total sudah > 1000, maka keluar dari loop

5. Susun program untuk menginput 100 buah bilangan yang merupakan nilai ujian mahasiswa, kemudian mencetak nilai tertinggi yang didapat mahasiswa .

```

int N, I, MAX;
scanf("%i", &N);
MAX = N;
for( I=2; I <= 100; I++)
{
    scanf("%i", &N);
    if( N > MAX )
        MAX = N ;
}
printf("\n Terbesar : %i",MAX);

```

Untuk pertama kali, nilai MAX dibuat sama dengan data pertama yang dibaca.

Disini dilakukan kali input. (dari 2 sampai dengan 100).

Bila nilai yang baru dibaca lebih besar dari MAX, maka nilai MAX diganti dengan nilai yang baru.

- Susun program untuk menginput 100 buah bilangan yang merupakan nilai ujian mahasiswa, kemudian mencetak nilai tertinggi yang didapat mahasiswa serta mencetak jumlah mahasiswa yang mendapat nilai tertinggi tersebut.

```

scanf("%i", &N);
MAX = N;
Jum = 1;
for( I=2; I <= 100; I++ )
{ scanf("%i", &N);
  if(N > MAX)
    { MAX = N ;
      Jum = 1;
    }
  else if(N == MAX)
    Jum++;
}
printf("\n Terbesar : %i", MAX);
printf("\n Jum Mhs : %i", Jum);

```

Untuk pertama kali, nilai MAX dibuat sama dengan data pertama yang dibaca, dan Jum dibuat = 1 (satu)

Disini dilakukan 99 kali input. (dari 2 sampai dengan 100).

Bila nilai yang baru dibaca lebih besar dari MAX, maka nilai MAX diganti dengan nilai yang baru.

Perhatikan :
Setiap kali nilai MAX diganti, nilai Jum kembali menjadi 1 (satu) lagi.

Kalau N tidak lebih besar dari MAX, tapi sama dengan MAX, maka Jum ditambah 1

ooooooooooooOOoooooooooooo

Daftar Pustaka

13. Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001
14. Jogyianto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993
15. Moh. Sjukani, 2009, Algoritma & Struktur Data 1 dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.
16. Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004.



MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Fungsi Dalam Bahasa C

Fakultas	Program Studi	Tatap Muka	Kode MK	Disusun Oleh
Ilmu Komputer	Sistem Informasi	05	MK87042	Tim Dosen

Abstract

Mampu memahami konsep pemrograman modular dan mengetahui fungsi-fungsi standar yang disediakan oleh bahasa C++ sehingga mampu mengimplementasikannya dalam pembuatan program dengan

Kompetensi

Mahasiswa mampu memahami kosep modular programming (sub program) dalam pemrograman, mengetahui konsep fungsi dalam C++ dan penggunaannya dalam program, Mengetahui fungsi-fungsi standar yang telah disediakan oleh bahasa C++,

menggunakan bahasa C++ untuk permasalahan yang cukup kompleks.

Mampu membuat program yang memiliki fungsi-fungsi yang didefinisikan sendiri.

Konsep Dasar Sub Program

Salah satu metode perancangan program yang baik adalah menggunakan konsep modular atau sering disebut dengan pemrograman modular. Dengan metode ini suatu masalah dipecah menjadi beberapa masalah yang lebih kecil (ke dalam modul-modul). Dengan membagi masalah menjadi beberapa modul, maka masalah tersebut akan menjadi lebih sederhana, sehingga program dapat lebih mudah disusun dan dipahami. Dalam bahasa C++ modul direalisasi dengan fungsi, dalam bahasa Pascal sub program terbagi dua jenis yaitu prosedur dan fungsi.

Manfaat lain dari pemrograman modular adalah *software reusability*. Fungsi yang dibuat satu kali diharapkan dapat digunakan oleh program lain, sehingga penulisan program (proses coding) lebih efektif.

Keuntungan menggunakan modul:

1. Rancangan *top-down* dengan pendekatan *divide-and-conquer*, program besar dapat dibagi menjadi modul-modul yang lebih kecil;
2. Dapat dikerjakan oleh beberapa orang dengan koordinasi yang lebih mudah;
3. Mencari kesalahan relatif lebih mudah karena alur logika lebih jelas, dan kesalahan dapat dilokalisasi dalam satu modul;
4. Modifikasi dapat dilakukan tanpa mengganggu program secara keseluruhan;
5. Mempermudah dokumentasi.

Sifat-sifat modul yang baik adalah:

- *Fan-in* yang tinggi: makin sering suatu modul dipanggil oleh pengguna makin tinggi nilai *fan-in*.

- *Fan-out* yang rendah: makin sedikit (spesifik) tugas yang dilakukan oleh suatu modul, makin rendah nilai *fan-out*.
- *Self-contained*: kemampuan memenuhi kebutuhannya sendiri.

Kategori Fungsi

Suatu fungsi adalah suatu bagian dari program yang dimaksudkan untuk mengerjakan suatu tugas tertentu dan letaknya dipisahkan dari bagian program yang menggunakannya. Fungsi-fungsi merupakan elemen utama dari program bahasa C++. program dari bahasa C++ dibentuk dari kumpulan fungsi, mulai dari fungsi utama dengan nama `main()`, fungsi-fungsi pustaka (standar) dan fungsi-fungsi yang dibuat sendiri oleh pembuat program.

Fungsi pada bahasa C++ terbagi dua:

1. **Standard library function.** *Standard library function* adalah fungsi-fungsi yang telah disediakan compiler C, tinggal digunakan.Untuk menggunakan *Standard library function* maka *header file* tempat fungsi tersebut didefinisikan harus dipanggil. Misalnya instruksi-instruksi yang berhubungan dengan posisi *cursor* `gotoxy()`, `wherex()`, `wherey()`, dan pembersihan layar `clrscr()`, `creol()` adalah fungsi standar yang prototipe fungsinya didefinisikan di *header file* `conio.h`. untuk menggunakan `gotoxy()` misalnya maka harus disertakan *preprocessor directive* `# include <conio.h>` pada awal agar copiler tahu bagaimana melaksanakan `gotoxy()`. Yang disebut dengan prototipe fungsi adalah judul fungsi beserta deklarasi jumlah dan tipe data parameter formal (data apa yang harus diberikan kepada fungsi ini untuk diolah).
2. **Programmer-defined function.** *Programmer-defined function* adalah fungsi-fungsi yang dibuat pemrogram untuk digunakan alm program. Fungsi yang dibuat dapat digunakan dalam satu program saja atau dijadikan satu *library* yang berisi kumpulan fungsi-fungsi sejenis.

Standard Library Function

Bahasa C++ menyediakan cukup banyak perpustakaan fungsi. Fungsi – fungsi tersebut dikelompokkan dan deklarasi *function-prototype*-nya disimpan dalam beberapa header file (.h).

Beberapa *standard function* yang berhubungan dengan perhitungan matematika dapat dilihat pada tabel dibawah ini. Waktu penggunaan fungsi tersebut harus menggunakan *processor directive* #include <math.h>.



Tabel Fungsi Standar Matematika

Function	Deskripsi	Contoh
int abs (int x)	Absolute bilangan bulat	abs (-5) → 5
double sqrt (double x)	Akar positif pangkat dua	sqrt (900.0) → 30.0
double exp (double x)	e^x	exp(1.0) → 2.718282
double log (double x)	$e \log x$	log (2.718282) → 1.0
double log 10 (double x)	$^{10} \log x$	log 10 (100.0) → 2.0
double fabs (double x)	Abs bilangan pecahan	fabs(-5.6) → 5.6
double ceil (double x)	Membulatkan x ke bilangan bulat terkecil yang tidak kurang dari x	ceil(9.2) → 10.0 ceil(-9.8) → -9.0
double floor(double x)	Membulatkan x ke bilangan bulat terbesar yang tidak lebih dari x	floor(9.2) → 9.0 floor(-9.8) → -10.0
double pow (double x, double y)	x^y	pow(3,4) → 81
double fmod(double x, double y)	Bilangan pecah sisa bagi x/y	fmod (12.35,5) → 2.35
double sin (double x)	Sinus dari x dalam radian	sin(0.0) → 0.0
double cos (double x)	Cosinus dari x dalam radian	cos (0.0) → 1.0
double tan (double x)	Tangen dari x dalam radian	tan (0.0) → 0.0

Deklarasi dan Pendefinisian Fungsi

Selain fungsi standar, bahasa C++ juga memungkinkan sebuah program mempunyai fungsi-fungsi yang dibuat oleh pembuat program. Fungsi ini disebut *programmer-defined function*.

Dalam merancang fungsi harus ditentukan:

1. Data apa yang ditentukan fungsi sebagai masukan untuk diolah,
2. Informasi apa yang dihasilkan (dikembalikan) fungsi kepada pemanggil,

3. Algoritma yang digunakan untuk mengolah data masukan menjadi informasi keluaran, beserta variabel lokal.

Penulisan sebuah fungsi dibagi atas prototipe fungsi (pendeklarasian fungsi) dan pendefinisian fungsi.

- ✓ Pendeklarasian Fungsi (*function prototype*)

```
Tipe_nilai_balik nama_fungsi(daftar_argumen);
```

- *Function Prototype* adalah kepala (judul) fungsi. Deklarasi ini digunakan oleh compiler dan pemrogram. Oleh compiler digunakan untuk memeriksa apakah pemanggilan terhadap suatu fungsi sudah sesuai baik masukannya (jumlah dan tipe data) maupun keluarannya. Oleh pemrogram digunakan untuk mengetahui argumen (actual parameter) apa yang perlu dikirim kepada fungsi dan jenis data hasil proses.
- Pembuatan prototype biasanya dilakukan jika pendefinisian fungsi diletakkan atau dituliskan setelah penulisan program utama. Supaya kompiler mengetahui fungsi yang dipanggil dalam fungsi utama maka sebelumnya fungsi yang dipanggil tersebut harus dituliskan prototype sebelum penulisan fungsi utama dan penulisan definisi fungsi dapat dituliskan setelah fungsi utama.

- ✓ Pendefinisian Fungsi (*function definition*)

```
Tipe_nilai_balik nama_fungsi(daftar_argumen) {  
    deklarasi_variabel_lokal;  
    instruksi_1;  
    instruksi_2;  
    ...  
    instruksi_n;  
    return(value);
```

}

function definition mendefinisikan fungsi secara lengkap. **Tipe_nilai_balik** menyatakan data jenis apa yang akan dikembalikan oleh fungsi sebagai hasil proses. **nama_fungsi** harus mengikuti ketentuan penulisan identifier. **daftar_argumen** terdiri nol, satu atau beberapa parameter. Parameter (disebut juga formal parameter) merupakan data yang harus dikirim kepada fungsi untuk diolah. Instruksi **return** digunakan untuk mengembalikan hasil proses kepada pemanggil.

Contoh program fungsi dapat dilihat pada listing program dibawah ini

Listing Program Fungsi Absolut

```
#include <iostream.h>

double Absolut(double x);      // Prototype fungsi Absolut

void main(void)
{
    float nilai;

    cout<<"Masukkan bilangan sembarang : ";
    cin>>nilai;

    cout<<"\nNilai mutlak untuk bilangan tersebut : "<<Absolut(nilai);
}

double Absolut(double x)      // Pendefinisan fungsi Absolut
{
    if(x<0) x= -x;
    return x;
}
```

Hasil Program:

Masukkan bilangan sembarang : -32.23

Nilai mutlak untuk bilangan tersebut : 32.23

Masukkan bilangan sembarang : 25

Nilai mutlak untuk bilangan tersebut : 25.00

Hasil Balik Fungsi

Sebuah fungsi dapat tanpa instruksi `return` (tidak mengembalikan nilai hasil proses), mempunyai sebuah instruksi `return` atau memiliki beberapa instruksi `return` (lebih dari satu). Untuk fungsi yang tidak mengembalikan nilai maka `return_type`-nya dapat ditulis `void`.

Letak instruksi `return` disesuaikan dengan kebutuhan. Saat diproses instruksi `return` memindahkan kontrol proses dari fungsi yang sedang aktif, kembali kepada fungsi pemanggil. Jika setelah instruksi `return` ada instruksi lagi maka instruksi tersebut tidak akan dikerjakan.

Contoh sebuah fungsi yang memiliki dua instruksi `return` tanpa kondisi:

```
float hitung(float n1, float n2)
{
    float kurang, tambah;
    tambah = n1 + n2;
    kurang= n1-n2;
    return tambah;
    return kurang; // instruksi ini tidak akan dikerjakan
}
```

Contoh sebuah fungsi yang memiliki dua instruksi `return` tanpa kondisi:

```
float hitung(float n1, float n2)
{
    float kurang, tambah;
    tambah = n1 + n2;
    kurang= n1-n2;
```

```

    if (n1< n2) return tambah; //n1 + n2 jika n1 < n2
    return kurang;           // n2 - n2 jika n1 > n2
}

```

Sebuah fungsi dapat pula tidak mempunyai parameter dengan cara mengosongkan daftar parameter atau dapat pula dengan menuliskan statemen `void`. Untuk lebih jelasnya dapat dilihat pada contoh di bawah ini:

```

void footer()
{ cout<<"Program 2013"<<endl;
}

```

atau:

```

int header(void)
{ cout<<"Dasar Pemrograman"<<endl;
  cout<<"dengan Bahasa C++";
}

```

Tugas Mandiri

1. Ketiklah program yang mempunyai fungsi untuk operasi aritmatika: penjumlahan, pengurangan, perkalian, dan pembagian dua buah bilangan bulat.

```

#include <iostream.h>
#include <ctype.h>

float tambah(float x,float y){ return(x+y);}

float kurang(float x, float y){return(x-y);}

float kali(float x, float y){return(x*y);}

float bagi(float x, float y){
  if(y != 0) return(x/y);
}

```

```

    return 0;
}

void main(void)
{
    float bil1, bil2, hasil;
    char op, pil;
    do{
        cout<<"Masukkan Bilangan 1 : "; cin>>bil1;
        cout<<"Masukkan Bilangan 2 : "; cin>>bil2;

        cout<<"Masukkan Operator (+, -, *, /) : "; cin>>op;

        switch (op){
            case '+': hasil = tambah(bil1,bil2); break;
            case '-': hasil = kurang(bil1,bil2); break;
            case '*': hasil = kali(bil1,bil2); break;
            case '/': hasil = bagi(bil1,bil2); break;
            default: cout<<"\nMaaf operator tidak dikenal!"<<endl;
        }
        cout<<"Hasil = "<<hasil<<endl;
        cout<<"Anda ingin menghitung lagi(y/t)? "; cin>>pil;
    }
    while(toupper(pil)== 'Y');
}

```

Hasil Program:



```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)
Masukkan Bilangan 1 : 13
Masukkan Bilangan 2 : 5
Masukkan Operator(+, -, *, /) : /
Hasil = 2.60

Anda ingin menghitung lagi(y/t)? y

Masukkan Bilangan 1 : 12.45
Masukkan Bilangan 2 : 1.34
Masukkan Operator(+, -, *, /) : *
Hasil = 16.68

Anda ingin menghitung lagi(y/t)? t
```

Gambar Hasil Running Program Fungsi Aritmatika

2. Ketiklah program yang mempunyai 2 fungsi, yaitu fungsi pangkat dan fungsi akar, untuk mencari hasil pemangkatan dan akar dapat menggunakan fungsi matematika pow() dan sqrt().

```
#include <iostream.h>
#include <math.h>

double pangkat(float x, float y);
double akar(float bil);

void main(void)
{
    float x, y;
    cout<<"Masukkan x : "; cin>>x;
    cout<<"Masukkan y : "; cin>>y;
    cout<<x<<" "<<"Pangkat"<<y<<" = "<<pangkat(x,y)<<endl;
    cout<<y<<" "<<"Pangkat"<<x<<" = "<<pangkat(y,x)<<endl;
    cout<<endl;
    cout<<x<<" "<<"akar dua = "<<akar(x)<<endl;
    cout<<y<<" "<<"akar dua = "<<akar(y)<<endl;
}

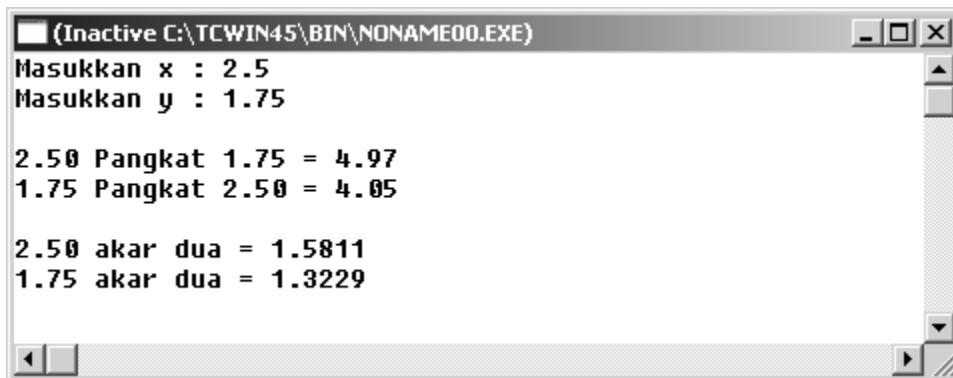
double pangkat(float x, float y)
```

```

{
    return pow(x,y);
}
double akar(float bil)
{
    return sqrt(bil);
}

```

Hasil Program:



Gambar 9.2 Hasil Running Program Fungsi Pangkat dan Akar

=====

Daftar Pustaka

17. Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001
18. Jogyianto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993
19. Moh. Sjukani, 2009, Algoritma & Struktur Data 1 dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.
20. Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004.



MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Fungsi Rekursif

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

06

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Mampu memahami konsep pemrograman modular dan mengetahui fungsi-fungsi standar yang disediakan oleh bahasa C++ sehingga mampu mengimplementasikannya dalam pembuatan program dengan

Kompetensi

Mahasiswa dapat membedakan ruang lingkup variabel dalam sebuah program dan fungsi, memahami tentang pengiriman parameter, baik pengiriman parameter secara nilai maupun pengiriman parameter secara alamat, Memahami konsep rekursi dan dapat

menggunakan bahasa C++ untuk permasalahan yang cukup kompleks.

mengimplementasikannya dalam pemrograman, khususnya dalam bahasa C++.

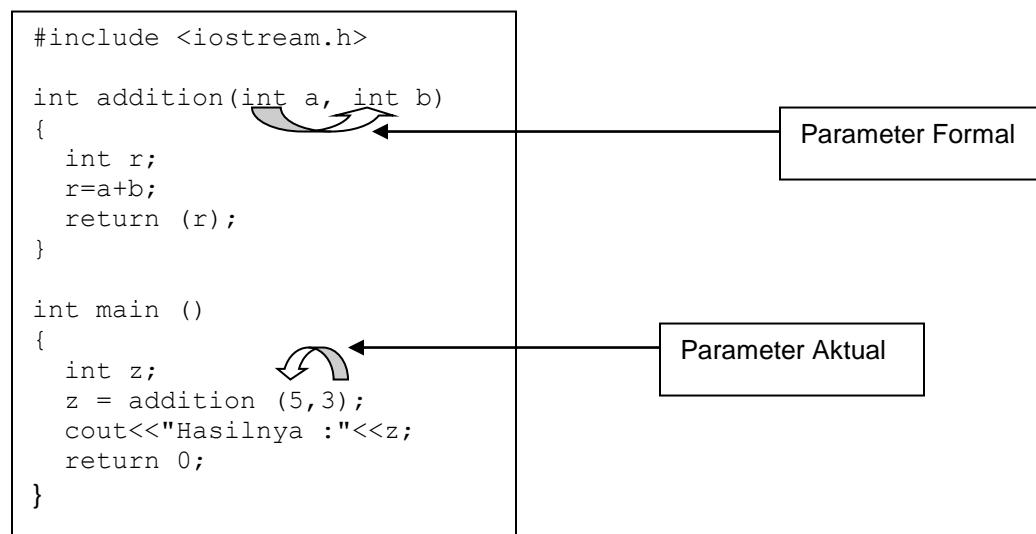
Pemanggilan Fungsi

Sebuah fungsi yang didefinisikan oleh pembuat program selain dapat dipanggil oleh fungsi main() juga dapat dipanggil oleh fungsi lain. Seperti kita ketahui bahwa bentuk umum sebuah fungsi adalah sebagai berikut:

Tipe_nilai_balik nama_fungsi(daftar_argumen) { instruksi; }

Daftar argumen berisikan parameter yang dituliskan pada saat pendefinisian fungsi disebut **parameter formal**, sedangkan daftar argumen berisikan parameter yang dituliskan pada saat fungsi dipanggil oleh fungsi lain **disebut parameter aktual**. Untuk dapat membedakan antara parameter formal dengan parameter aktual dapat kita perhatikan listing program di bawah ini:

Listing Program Pemanggilan Fungsi oleh main()



Hasil Program:

```
Hasilnya 8
```

Pemanggilan terhadap sebuah fungsi adalah suatu ekspresi. Pada contoh di atas fungsi `addition` dipanggil oleh fungsi `main()`. Pada saat pemanggilan tersebut terjadi pengiriman nilai dari parameter aktual kepada parameter formal.

```
int addition (int a, int b)
              ↑   ↑
z = addition ( 5 , 3 );
```

Variabel yang digunakan untuk menampung hasil pemanggilan suatu fungsi harus setipe dengan tipe_nilai_balik dari fungsi yang dipanggil. Berdasarkan program 10.1, kita lihat tipe data dari `z` harus sama dengan tipe data dari fungsi `addition`. Pada fungsi `addition` terdapat variabel `r` yang bertipe sama dengan tipe data fungsi yaitu `int`. Nilai yang dikembalikan dengan menggunakan instruksi `return` oleh fungsi `addition` adalah nilai dari `r`, kemudian nilai `r` tersebut diberikan kepada `z`.

```
return (r);
```

```
int addition (int a, int b)
              ↓
z = addition ( 5 , 3 );
```

Sebuah fungsi yang dideklarasikan oleh pembuat program dapat juga memanggil fungsi lain yang sebelumnya telah dideklarasikan atau dibuat sebelumnya. Penulian fungsi dalam program yang sama misalnya fungsi **A** dan fungsi **B**, dengan fungsi A akan memanggil fungsi B. Posisi fungsi B dapat dituliskan sebelum ataupun setelah fungsi A. Jika fungsi B dituliskan sesudah fungsi A maka *function prototype* dari fungsi B harus dituliskan, sebaliknya jika fungsi B dituliskan sebelum fungsi A maka *function prototype* dari fungsi B boleh tidak dituliskan. Untuk melihat perbedaan letak penulian fungsi yang dipanggil dapat melihat listing program dibawah ini.

Listing Program Fungsi Memanggil Fungsi yang Didefinisikan Sebelum Fungsi Utama

```
#include <iostream.h>

float kali(float x, float y)
{ return(x * y);}

float segitiga(float als, float tg)
{ return(kali(als, tg)/2); }           //fungsi kali dipanggil oleh fungsi
segitiga

float persegi(float p, float l)
{ return(kali(p,l)); }                //fungsi kali dipanggil oleh fungsi persegi

void cetakLuas(float nilai)
{ cout<<"Hasil = "<<nilai<<endl; }

void main(void)
{
    float a= 10.5, b=2.4;

    cetakLuas(segitiga(a, b)); } /* fungsi cetakLuas dipanggil dua kali,
parameter
    cetakLuas(persegi(a, b));      yang dikirim adalah nilai balik dari fungsi
                                    segitiga dan persegi */
```

```
}
```

Listing Program Fungsi Memanggil Fungsi yang Didefinisikan Sesudah Fungsi Utama

```
#include <iostream.h>

float segitiga(float als, float tg);
float persegi(float p, float l);

float kali(float x, float y)
{ return (x*y); }

void cetakLuas(float nilai)
{ cout<<"Hasil = "<<nilai<<endl; }

void main(void)
{ float a= 10.5, b=2.4;

    cetakLuas(segitiga(a, b));
    cetakLuas(persegi(a, b));
}

float segitiga(float als, float tg)
{ return(kali(als,tg)/2); }

float persegi(float p, float l)
{ return(kali(p,l)); }
```

Kedua program menghasilkan keluaran yang sama.

Hasil Program:

```
Hasil = 12.600
```

```
Hasil = 25.200
```

Pengiriman Parameter

Sebuah fungsi dapat dipanggil atau digunakan oleh fungsi lain. Komunikasi yang terjadi antara satu fungsi yang memanggil (*calling function*) dengan fungsi yang dipanggil (*called function*) dilakukan dengan cara saling bertukar data. Jika sebuah fungsi yang memanggil fungsi yang lain untuk melakukan suatu tugas tertentu , kepada fungsi yang dipanggil diberikan semua data yang dibutuhkan untuk melaksanakan tugasnya. Setelah fungsi yang dipanggil menerima data yang diberikan dan melaksanakan tugas tersebut, fungsi yang dipanggil akan mengembalikan hasil proses kepada fungsi yang memanggil.

Pertukaran atau pengiriman data tersebut melalui antar muka atau penghubung yang disebut parameter. Parameter dapat berupa *identifier* (variabel atau konstanta) dapat pula berupa nilai. Pengiriman parameter dalam pemrograman ada 2 jenis yaitu, pengiriman parameter secara nilai dan pengiriman parameter secara acuan/ alamat.

Pengiriman Parameter Secara Nilai (*call by value*)

Pada pemanggilan parameter secara nilai, pengiriman data dilakukan dengan cara mengirimkan nilai dari fungsi yang memanggil ke fungsi yang dipanggil. Dengan kata lain nilai dari parameter aktual diberikan kepada parameter formal. Fungsi yang menerima kiriman nilai akan menyimpannya di alamat terpisah dari nilai asli yang digunakan oleh program yang memanggil fungsi. Perubahan nilai di fungsi (parameter formal) tidak akan merubah nilai asli di bagian program yang memanggilnya.Pengiriman parameter secara nilai adalah pengiriman searah, yaitu dari bagian program yang memanggil fungsi

ke fungsi yang dipanggil. Pengiriman suatu nilai dapat dilakukan untuk suatu ungkapan, tidak hanya untuk sebuah variabel, elemen array atau konstanta saja.

```
int addition (int a, int b)
              ↑   ↑
z = addition ( 5 , 3 );
```

Berdasarkan contoh di atas, **addition** adalah nama fungsi untuk menjumlahkan dua bilangan yang memiliki dua parameter yaitu **a** dan **b** yang bertipe data integer. Nilai balik dari fungsi **addition** bertipe data integer. **z** adalah variabel yang berisi nilai hasil pemanggilan fungsi **addition**. Variabel **a** dan **b** adalah parameter formal, sedangkan 5 dan 3 adalah nilai yang dikirimkan oleh bagian yang memanggil fungsi (dalam hal ini adalah **z**), nilai 5 diberikan kepada **a** dan nilai 3 diberikan kepada **b**.

Untuk lebih memahami tentang pengiriman parameter secara nilai dapat dilihat pada listing program di bawah ini:

Listing Program Contoh Call by Value

```
#include <iostream.h>
#include <conio.h>

void tukar(int x, int y); /* pendeklarasian fungsi */

void main()
{
    int a,b;
    a = 20; b = 30;
    cout<<"Nilai sebelum pemanggilan fungsi"<<endl;
    cout<<"a = "<<a<<"  b = "<<b<<endl; // a dan b sebelum pemanggilan fungsi

    tukar(a,b); /* pemanggilan fungsi tukar() */

    cout<<"Nilai setelah pemanggilan fungsi"<<endl;
    cout<<"a = "<<a<<"  b = "<<b<<endl; // a dan b setelah pemanggilan fungsi
    getch();
}
```

Hasil Program:

```
Nilai sebelum pemanggilan fungsi  
a = 20    b = 30
```

```
Nilai di dalam fungsi tukar()  
x = 30    y = 20
```

Pengiriman Parameter Secara Acuan/ Alamat (*Call by Reference*)

Pada pemanggilan parameter secara Acuan, pengiriman data dilakukan dengan cara mengirimkan alamat-alamat dimana nilai dari parameter pada fungsi yang memanggil disimpan ke fungsi yang dipanggil, jadi pada pengiriman parameter secara acuan yang dikirimkan adalah alamat dimana sebuah nilai di simpan dan bukan nilainya. Fungsi yang menerima kiriman alamat ini akan menggunakan alamat yang sama untuk mendapatkan nilai datanya. Perubahan nilai di fungsi akan merubah nilai asli di bagian program yang memanggil fungsi. Pengiriman parameter secara acuan adalah pengiriman dua arah, yaitu dari fungsi pemanggil ke fungsi yang dipanggil dan juga sebaliknya. Pengiriman secara acuan tidak dapat dilakukan untuk suatu ungkapan. Contoh pengiriman parameter secara acuan:

```
void duplicate (int& a,int& b,int& c)  
{  
    a = 10;  
    b = 20;  
    c = 30;  
}  
  
duplicate ( x , y , z );
```

Pada contoh di atas yang menjadi parameter aktual adalah **x**, **y** dan **z**, sedangkan yang menjadi parameter formal adalah **a**, **b** dan **c**. Pada parameter formal variabel **a**, **b** dan **c** dideklarasikan sebagai variabel pointer (variabel yang menampung alamat) dengan menggunakan operator alamat **&** atau operator *****. Saat fungsi **duplicate** dipanggil, alamat dari **x**, **y** dan **z** dikirimkan kepada **a**, **b** dan **c**. Untuk lebih memahami tentang pengiriman parameter secara acuan dapat dilihat listing program 10.5 dan 10.6 di bawah ini:

Listing Program 10.5 Contoh Call by Reference_1

```
#include <iostream.h>

void duplicate (int& a, int& b, int& c)
{ a*=2;
  b*=2;
  c*=2;
}

int main ()
{ int x=1, y=3, z=7;
  cout<<"Sebelum pemanggilan fungsi duplicate"<<endl;
  cout<<"x= %d  y= %d  z= %d"<<endl;
  cout<<"Setelah pemanggilan fungsi duplicate"<<endl;
  duplicate (x, y, z);
  cout<<"x= %d  y= %d  z= %d";
  return 0;
}
```

Program di atas dapat ditulis seperti dibawah ini, kedua program akan menampilkan hasil keluaran yang sama.

Listing Program Contoh Call by Reference_2

```
#include <iostream.h>

void duplicate (int* a, int* b, int* c)
{
  *a*=2;
  *b*=2;
  *c*=2;
}

int main ()
{ int x=1, y=3, z=7;
  cout<<"Sebelum pemanggilan fungsi duplicate"<<endl;
  cout<<"x= %d  y= %d  z= %d"<<endl;
  cout<<"Setelah pemanggilan fungsi duplicate"<<endl;
  duplicate (&x, &y, &z);
  cout<<"x= %d  y= %d  z= %d"<<endl;
  return 0;
}
```

Hasil Program:

```
Sebelum pemanggilan fungsi duplicate  
x= 1    y= 3    z= 7
```

Ruang Lingkup Identifier dan Variabel

Berdasarkan jangkauannya atau ruang lingkupnya *identifier* terbagi dua, yaitu *global identifier* dan *local identifier*.

Global identifier adalah *identifier* yang dideklarasikan di luar fungsi dan ditempatkan di atas semua fungsi dalam suatu program. Jangkauan dari *global identifier* meliputi seluruh program. *Identifier* yang dideklarasikan secara global, dapat dideklarasikan kembali di dalam fungsi.

Local identifier adalah *identifier* yang dideklarasikan di dalam fungsi, termasuk daftar parameter. Jangkauan dari *local identifier* terbatas hanya pada fungsi atau sub program yang mendeklarasikan *identifier* itu sendiri.

Penggunaan *identifier global* memang terlihat lebih mudah karena satu *identifier* dapat dikenali oleh semua bagian dari program dan transfer data antar fungsi menjadi lebih mudah, tetapi ada beberapa kerugian jika menggunakan *identifier global*, yaitu:

- Jika program semakin besar, maka semakin besar pula kecenderungan terjadinya *error*.
- Sulit melacak kesalahan.
- Data tidak terjaga dengan baik, setiap fungsi dapat mengubah isi variabel tanpa sepenuhnya mengetahuan fungsi lainnya.

Nama variabel adalah *identifier* sehingga variabel dapat dideklarasikan sebagai variabel lokal atau variabel global. Bahasa C mengelompokkan variabel berdasarkan jangkauannya menjadi:

1. Variabel lokal
2. Variabel eksternal
3. Variabel statis, dan
4. Variabel register

Variabel Lokal

Variabel lokal adalah variabel yang dideklarasikan di dalam fungsi. Sifat-sifat variabel lokal adalah:

- Secara otomatis akan diciptakan ketika fungsi dipanggil dan akan lenyap ketika proses eksekusi terhadap fungsi berakhir.
- Hanya dikenal oleh fungsi tempat variabel dideklarasikan
- Tidak ada inisialisasi secara otomatis (saat variabel diciptakan nilainya random).
- Dideklarasikan dengan menambahkan kata “**auto**” (opsional).

Variabel lokal dapat dideklarasikan dengan memberikan kata kunci auto di depan tipe data. Variabel yang dideklarasikan di dalam fungsi bila tidak diberi kata kunci **auto** akan dianggap sebagai variabel lokal.

Variabel global (eksternal)

Variabel global (eksternal) adalah variabel yang dideklarasikan di luar fungsi. Sifat-sifat variabel global :

- Dikenal (dapat diakses) oleh semua fungsi.
- Jika tidak diberi nilai awal secara otomatis berisi nilai nol.
- Dideklarasikan dengan menambahkan kata “**extern**” (opsional).

Variabel global dapat juga diletakkan di antara fungsi sehingga menjadi semi global. Jangkauan variabel seperti ini adalah mulai dari tempat deklarasi sampai ujung program. Apabila variabel global mempunyai nama yang sama dengan variabel lokal maka yang dikenal pada lokasi setempat adalah variabel lokal.

Listing Program Variabel Global dan Variabel Lokal

```
#include <iostream.h>
#include <stdio.h>

void tampil(void);

int i = 25;           /* variabel global */

void main()
{ clrscr();
    cout<<"Nilai variabel i dalam fungsi main() adalah "<<i<<endl;
    tampil();
    i = i * 4;           /* nilai i yang dikali 4 adalah 25 (global) bukan 10 */
    cout<<"Nilai variabel i dalam fungsi main() sekarang adalah "<< i;
}

void tampil(void)
{ int i = 10;           /* variabel lokal */
    cout<<"Nilai variabel i dalam fungsi tampil() adalah "<<i<<endl;
}
```

Hasil Program:

```
Nilai variabel i dalam fungsi main() adalah 25
```

```
Nilai variabel i dalam fungsi tampil() adalah 10
```

Variabel Statis

Variabel statis adalah variabel yang nilainya tetap dan bisa berupa variabel lokal (internal) dan variabel global (eksternal). Sifat-sifat variabel statis :

- Jika bersifat internal (lokal), maka variabel hanya dikenal oleh fungsi tempat variabel dideklarasikan.
- Jika bersifat eksternal (global), maka variabel dapat dipergunakan oleh semua fungsi yang terletak pada program yang sama.

- Nilai variabel statis tidak akan hilang walau eksekusi terhadap fungsi telah berakhir.
- Inisialisasi hanya perlu dilakukan sekali saja, yaitu pada saat fungsi dipanggil pertama kali.
- Jika tidak diberi nilai awal secara otomatis berisi nilai nol.
- Dideklarasikan dengan menambahkan kata “static”.

Listing Program Variabel Statis

```
#include <iostream.h>

void nambah();

void main()
{ int stat=100;
    nambah();
    nambah();
    cout<<"Nilai stat dalam fungsi main()="<<stat<<endl;
}

void nambah()
{ static int stat=5;           /* variabel statis */
    stat++;
    cout<<"Nilai stat dalam fungsi nambah()="<<stat<<endl;
}
```

Hasil Running:

```
Nilai stat dalam fungsi nambah()= 6
Nilai stat dalam fungsi nambah()= 7
Nilai stat dalam fungsi nambah()= 100
```

Variabel Register

Variabel Register adalah variabel yang nilainya disimpan dalam register dan bukan dalam memori RAM.

Sifat-sifat variabel register :

- Hanya dapat diterapkan pada variabel lokal yang bertipe int dan char.
- Digunakan untuk mengendalikan proses perulangan (looping).
- Proses perulangan akan lebih cepat karena variabel register memiliki kecepatan yang lebih

- tinggi dibandingkan variabel biasa.
- Dideklarasikan dengan menambahkan kata “register”.

Listing Program Variabel Register

```
#include <iostream.h>

void main()
{
    register int x;          /* variabel register */
    int jumlah;

    for(x=1; x<=100; x++)
        jumlah = jumlah + x; cout<<"1+2+3+...+100 ="<<jumlah<<endl;
}
```

Hasil Running:

```
1+2+3+...+100 = 8260
```

Latihan:

1. Apakah kegunaan variabel statis? Berikan contoh pemakaianya.
2. Jika sebuah variabel tidak didefinisikan sebagai salah satu jenis variabel (lokal, global/eksternal, statis atau register) maka variabel tersebut diberi default apa?
3. Apa kelebihan dan kekurangan menggunakan variabel global?
4. Apa yang dimaksud dengan *local identifier* dan *global identifier*?
5. Apa yang dimaksud dengan *call by value*?
6. Apa yang dimaksud dengan *call by reference*?
7. Apa keuntungan menggunakan variabel register?

Tugas Mandiri:

1. Buatlah fungsi untuk menukar dua bilangan dengan menggunakan pengiriman parameter secara acuan.
 2. Tulislah sebuah fungsi yang dapat menguji apakah bilangan bulat kedua merupakan kelipatan dari bilangan bulat pertama
- =====

Daftar Pustaka

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Array (Larik)

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

07

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Memberikan gambaran umum mengenai konsep tipe data array dan mampu mengimplementasikan array

Kompetensi

Mahasiswa dapat memahami definisi array dan konsep array, mampu mendeklarasikan array dalam bahasa

untuk pemecahan masalah pemrograman yang lebih kompleks.

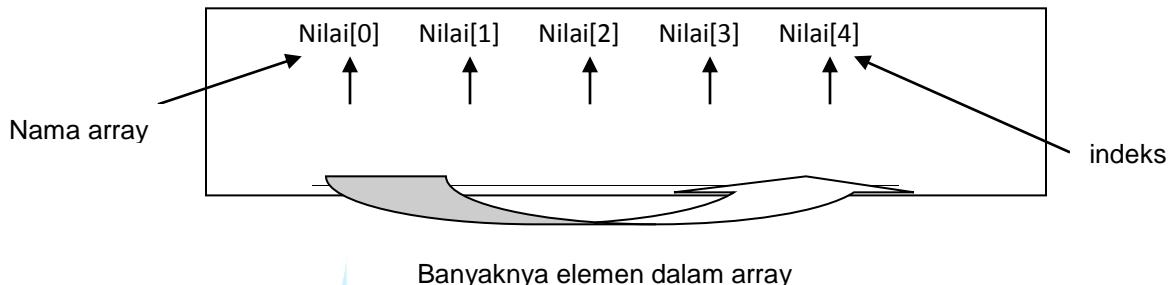
C, dan mampu membuat program dengan bantuan tipe data array.

Pengertian Array (Larik)

Array(larik) merupakan kumpulan dari nilai-nilai data yang bertipe sama dalam urutan tertentu yang menggunakan nama yang sama. Letak atau posisi dari elemen array ditunjukkan oleh suatu index. Dilihat dari dimensinya array dapat dibagi menjadi Array dimensi satu, array dimensi dua dan array multi-dimensi. Setiap elemen array dapat diakses melalui indeks. Dalam bahasa C indeks dimulai dari 0 (nol).

Itu berarti dengan menggunakan array jika kita ingin menyimpan lima nilai bertipe data *int* kita tidak perlu mendeklarasikan lima variabel yang berbeda dengan tipe data *int*, kita cukup mendeklarasikan satu variabel array dengan tipe data *int*.

Sebagai contoh, sebuah array yang dapat menampung lima bilangan bulat yang diberi nama Nilai maka dapat digambarkan sebagai berikut:



Deklarasi Array

Seperti halnya penggunaan variabel, array pun sebelum digunakan harus dideklarasikan terlebih dahulu. Pada bahasa C pendeklarasian array adalah sebagai berikut:

- ✓ Bentuk umum :

```
Tipe_array nama_array[ukuran];
```

- tipe_array : adalah tipe data yang dapat ditampung oleh array.
- nama_array : nama variabel array yang dideklarasikan.
- [ukuran] : banyaknya nilai yang dapat ditampung oleh array (banyaknya elemen array).

Contoh:

```
int x[5];  
  
float diameter[10];  
  
char kata[20];
```

- ✓ Inisialisai (pemberian nilai awal) sebuah array:

Pemberian nilai awal dari sebuah array dapat dilakukan bersamaan pada saat deklarasi ataupun setelah sebuah array dideklarasikan. Untuk inisialisasi bersamaan dengan pendeklarasian array dapat dilihat pada contoh di bawah ini:

```
int x[] = {10, 34, 22, 56, 23};
```

Dari contoh di atas dapat diberi penjelasan sebagai berikut:

- Dideklarasikan sebuah array yang diberi nama **x** yang mempunyai nilai-nilai elemen 10, 34, 22, 56 dan 23.
- Variabel **x** dideklarasikan dengan tidak menuliskan jumlah elemen yang dapat ditampung, hal ini diperkenankan dan biasanya array semacam ini disebut array yang tidak berukuran.
- Untuk array yang tidak berukuran kompiler C akan secara otomatis menentukan ukuran

tergantung dari jumlah nilai-nilai elemen yang diberikan saat deklarasi.

- Untuk contoh di atas, maka dapat dikatakan variabel x mempunyai lima elemen.

Catatan:

- Array tidak dapat dideklarasikan tanpa ukuran dan tanpa pemberian nilai awal, seperti berikut: `int x[];`; kompiler akan memberikan pesan kesalahan bahwa array tidak memiliki ukuran.
- Permasalahan sering muncul jika pembuat program belum dapat menentukan berapa jumlah elemen array yang akan digunakan dalam programnya. Hal ini dapat diatasi dengan menentukan jumlah maksimal elemen yang ditampung terlebih dahulu saat pendeklarasian dengan melihat batasan memori yang digunakan. Misalnya jumlah elemen array terlebih dahulu ditentukan 100, kemudian dalam program ada masukan yang menanyakan jumlah elemen array sesungguhnya yang dibutuhkan, misalnya pengguna program hanya menggunakan 10 elemen saja, maka sisa elemen yang telah ditentukan sebelumnya tidak akan digunakan meskipun sudah dipesan penempatannya dalam memori. Contoh program dapat dilihat pada listing program 12.1.

- ✓ Pengaksesan nilai dari sebuah array:

Elemen array dapat diakses dalam program. Pengaksesan yang dimaksud dapat berupa pemberian nilai maupun mengambil nilai yang disimpan dalam sebuah array. Bentuk umum pengaksesan array adalah sebagai berikut:

`nama_array[indeks]`

Contoh:

```
int athar[4];  
athar[0]=10;  
athar[1]= athar[0]/2;  
athar[2]= athar[1] * athar[0];
```

```
athar[3]=2;
```

Listing Program Contoh Array dengan ukuran

```
/* Program untuk menginput nilai mahasiswa ke dalam array satu dimensi */

#include <stdio.h>
#include <conio.h>
#include <iostream.h>

void main()
{
    int index, nilai[10];
    clrscr();

    cout<<"Input nilai 10 mahasiswa"<<endl; /* input nilai mahasiswa */
    for(index=0; index < 10; index++)
    {
        cout<<"Mahasiswa : "<<index+1;
        cin>>nilai[index];
    }

    /* tampilkan nilai mahasiswa */
    cout<<"Nilai mahasiswa yang telah diinput"<<endl;
    for(index=0; index < 10; index++)
    {
        cout<<nilai[index];
    }
    getch();
}
```

Hasil Running Program:

```
Input nilai 10 mahasiswa
Mahasiswa 1 : 73
Mahasiswa 2 : 43
Mahasiswa 3 : 56
Mahasiswa 4 : 93
Mahasiswa 5 : 38
Mahasiswa 6 : 45
Mahasiswa 7 : 78
Mahasiswa 8 : 84
Mahasiswa 9 : 76
Mahasiswa 10 : 58
Nilai mahasiswa yang telah diinput
73   43   56   93   38   45   78   84   76   58
```

Listing Program Jumlah Elemen Array Ditentukan Pengguna

```
/* Program untuk menginput nilai mahasiswa ke dalam array, jumlah mahasiswa ditentukan oleh pengguna*/

#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#define max 100

void main()
{
    int n, index, nilai[max];
    clrscr();

    cout<<"Input banyaknya mahasiswa: "; /*Input jumlah elemen array sesungguhnya */
    cin>>n;

    cout<<endl; /* input nilai mahasiswa */
    for(index=0; index < n; index++)
    {   cout<<"Mahasiswa : "<<index+1;
        cin>>nilai[index];
    }

    /* tampilkan nilai mahasiswa */
    cout<<"Nilai mahasiswa yang telah diinput"<<endl;
    for(index=0; index < n; index++)
    {   cout<<nilai[index];
    }
    getch();
}
```

Hasil Running Program:

```
Input banyaknya mahasiswa: 5
Mahasiswa 1 : 73
Mahasiswa 2 : 43
```

```
Mahasiswa 3 : 56  
Mahasiswa 4 : 93  
Mahasiswa 5 : 38  
Nilai mahasiswa yang telah diinput  
73 43 56 93 38
```

Array Multi Dimensi

Array multi dimensi dapat digambarkan sebagai “array di dalam array”. Bahasa C tidak membatasi jumlah dimensi array, yang membatasi hanyalah jumlah memori yang tersedia. Array multi dimensi yang banyak dipergunakan dalam pemrograman adalah array dua dimensi.

Array dua dimensi dapat diumpamakan sebagai sebuah tabel ataupun matrik yang memiliki baris dan kolom. Sebagai contoh dapat dilihat gambar di bawah ini:

The diagram shows a 2D array structure. On the left, the variable name "Ababil" is written vertically above a brace that spans three rows. To the right of the brace is a grid of 15 empty cells arranged in 3 rows and 5 columns. The columns are labeled at the top with blue numbers 0, 1, 2, 3, and 4. The rows are labeled on the left with blue numbers 0, 1, and 2.

	0	1	2	3	4
0					
1					
2					

Variabel Ababil merupakan array dua dimensi yang mempunyai elemen 3 baris, dimana masing-masing baris memiliki 5 kolom. Total elemen array adalah 15 elemen. Dalam bahasa C variabel array Ababil dapat dideklarsikan sebagai berikut:

```
int Ababil [3][5];
```

atau jika ingin mendeklarasikan dengan memberikan nilai awal untuk masing-masing elemen dapat dilakukan sebagai berikut:

```
int Ababil [3][5]= {7, 5, 6, 8, 3, 2, 3, 5, 6, 3, 1, 0, 4, 7, 5};
```

Supaya memperjelas pemberian nilai awal, penulisan deklarasi dapat juga dituliskan sebagai berikut:

```
int Ababil [3][5] = {  
    7, 5, 6, 8, 3,  
    2, 3, 5, 6, 3,  
    1, 0, 4, 7, 5  
};
```

Array dua dimensi Ababil ini dapat pula dideklarasikan secara tidak berukuran sebagai berikut:

```
int Ababil [] [5] = {7, 5, 6, 8, 3, 2, 3, 5, 6, 3, 1, 0, 4, 7, 5};
```

Catatan:

Dalam pendeklarasian array dua dimensi yang dapat tidak disebutkan jumlah elemennya adalah untuk dimensi yang pertama, sedang dimensi yang kedua harus ditentukan jumlah elemennya. Dengan mengetahui jumlah elemen dari dimensi kedua maka dapat menentukan jumlah elemen untuk masing-masing dimensi kedua. Berdasarkan contoh di atas maka dapat ditentukan 5 nilai pertama adalah untuk baris pertama, kemudian 5 nilai berikutnya adalah untuk baris kedua dan 5 nilai terakhir adalah untuk baris terakhir.

Contoh penggunaan array dua dimensi adalah program untuk menginput matrik dan menampilkan matrik tersebut. Program matrik disajikan dalam listing program dibawah ini.

Listing Program Input Matriks

```
/* Program menginput nilai(bilangan) ke dalam array  
dimensi dua dan menampilkannya */  
  
#include <stdio.h>  
#include <conio.h>  
#include <iostream.h>  
  
void main()
```

```

{
    int baris, kolom, matriks[2][3];

    // Input elemen array

    for(baris=0; baris<2; baris++)
    {   for(kolom=0; kolom<3; kolom++)

        {
            cout<<"matriks"<<baris+1<<kolom+1);
            cin>>matriks[baris][kolom];
        }
        cout<<endl;
    }

    // Tampilkan elemen Array

    cout<<"Isi array :"<<endl;
    for(baris=0; baris<2; baris++)
    {   for(kolom=0; kolom<3; kolom++)
        {   cout<<matriks[baris][kolom];
        }
        cout<<endl;
    }

    getch();
}

```

Hasil Running Program:

```

matriks[1][1]:12
matriks[1][2]:13
matriks[1][3]:21

matriks[2][1]:11
matriks[2][2]:7
matriks[2][3]:9

Isi array :
  12      13      21
  11      7       9

```

Array sebagai Parameter

Jika array dijadikan sebagai parameter dari suatu fungsi, maka untuk mengirimkan array tersebut harus menggunakan alamat array, yaitu pengiriman parameter secara alamat.

Sebagai contoh pada listing program dibawah ini, program meminta pengguna program memasukkan sejumlah bilangan bulat, kemudian bilangan bulat tersebut akan dikalikan dengan dua dan ditampilkan.

Pada program dibawah ini mempunyai modul `inputArray(arg[], pjng)` yang bertipe `void`, modul `inputArray` bertujuan untuk meminta pengguna memasukkan nilai dari elemen-elemen array, banyaknya elemen array tergantung dari nilai yang diterima oleh parameter `pjng`. Nilai-nilai tersebut akan diberi kepada array `arg`. `arg[]`, `pjng` dijadikan sebagai parameter aktual.

Modul `kaliArray(arg[], pjng)` bertujuan untuk mengalikan nilai elemen-elemen `arg` dengan dua. Untuk mengisikan nilai elemen-elemen array `arg` digunakan perulangan sebanyak jumlah elemen yang nilai banyaknya elemen didapat dari parameter `pjng`.

Modul `cetakArray(arg[], pjng)` bertujuan untuk mencetak nilai elemen-elemen `arg`. Untuk menampilkan nilai elemen-elemen array `arg` digunakan perulangan sebanyak jumlah elemen yang nilai banyaknya elemen didapat dari parameter `pjng`.

Dalam modul `main()`, dideklarasikan array `arr` yang mampu menampung maksimal 100 nilai, dan variabel `bnyk` yang bertipe integer. Pertama program meminta masukan dari pengguna berupa jumlah elemen array, nilai tersebut diberikan kepada variabel `bnyk`. Kemudian program memanggil fungsi `inputArray`, variabel yang dijadikan parameter aktual adalah `arr` dan `bnyk`. Pada pemanggilan fungsi dalam fungsi `main()` dengan mengirimkan array sebagai parameter ukuran dari array yang ditandai dengan tanda “[]” tidak perlu dituliskan cukup nama array-nya saja, seperti pemanggilan berikut: `inputArray(arr, bnyk)`.

Listing Program Array Sebagai Parameter

```
#include <stdio.h>
#include <conio.h>

void inputArray(int arr[], int bnyk)      /* fungsi Input elemen array */
{ for(int i=0; i< bnyk; i++){
    printf("Masukkan arr[%d]: ", i+1);
    scanf("%d", &arr[i]);
}
}
```

```

void kaliArray (int arg[], int pjng)      /* fungsi mengalikan elemen array
dengan 2 */
{ for (int n=0; n<pjng; n++)
    arg[n] *=2;
}

void cetakArray (int arg[], int pjng) /* fungsi cetak elemen array */
{ for (int n=0; n<pjng; n++)
    printf("%4.0d",arg[n]," ");
    printf("\n");
}

int main ()
{
    int bnyk, arr[100];
    printf("Program Array sebagai Argumen dalam Fungsi\n");
    printf("Masukkan jumlah elemen : ");
    scanf("%d", &bnyk);
    inputArray(arr, bnyk); /* Pengiriman array ke parameter dalam fungsi */
    kaliArray(arr, bnyk);
    cetakArray(arr, bnyk);
    return 0;
}

```

Hasil Running Program:

```

Program Array sebagai Argumen dalam Fungsi
Masukkan jumlah elemen : 5
Masukkan arr[1] : 12
Masukkan arr[2] : 23
Masukkan arr[3] : 5
Masukkan arr[4] : 33
Masukkan arr[5] : 65

```

```

24     46     10     33     65

```

Latihan:

1. Apakah array dapat digunakan sebagai *return type* sebuah fungsi?
2. sebuah array dideklarasikan dengan float xy[10][5]; berapa banyak memori yang digunakan array ini?
3. Bagaimana mengetahui jumlah dimensi array?

4. Apakah beda variabel array dengan variabel biasa?

Tugas Mandiri:

1. Buatlah algoritma untuk menghitung rata-rata dari beberapa nilai mahasiswa, banyaknya nilai yang akan dihitung berdasarkan masukan dari pengguna program.
2. Buatlah algoritma untuk menjumlahkan dua buah matriks yang terdiri fungsi untuk menginput matriks, fungsi untuk menjumlahkan matriks dan fungsi untuk mencetak matriks.

=====

Daftar Pustaka

Cheltenham Computer Training, C Programming, www.cctglobal.com, United Kingdom, 1997

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Pointer

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

08

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Memberikan gambaran umum mengenai konsep tipe data pointer dan mampu mengimplementasikan pointer

Kompetensi

Mahasiswa dapat memahami definisi pointer dan konsep pointer, mampu mendeklarasikan pointer dalam bahasa

untuk pemecahan masalah pemrograman yang lebih kompleks.

C, dan mampu membuat program dengan bantuan tipe data pointer.

Pengertian Pointer

Pointer adalah suatu variabel penunjuk yang menunjuk pada suatu alamat memori komputer tertentu. Pointer merupakan variabel level rendah yang dapat digunakan untuk menunjuk nilai integer, character, float, double, atau single, dan bahkan tipe-tipe data lain yang didukung oleh bahasa C. Variabel biasa, sifatnya statis dan sudah pasti, sedangkan pada pointer sifatnya dinamis dan dapat lebih fleksibel. Variabel pointer yang tidak menunjuk pada nilai apapun berarti memiliki nilai NULL, dan disebut sebagai dangling pointer karena nilainya tidak diinisialisasi dan tidak dapat diprediksi.

Pendeklarasian variabel pointer menggunakan tanda * sebelum nama variabelnya, sedangkan untuk menampilkan nilai yang ditunjuk oleh suatu variabel pointer, juga digunakan operator * (tanda asterisk). Jika diinginkan untuk menampilkan alamat tempat penyimpanan nilai yang ditunjuk oleh suatu variabel pointer, digunakan operator & (tanda ampersand).

Pada suatu tipe data array, variabel pointer hanya perlu menunjuk pada nama variabel arraynya saja tanpa perlu menggunakan tanda ampersand, atau menunjuk pada nama variabel array pada indeks yang ke nol nya. Untuk lebih jelasnya, silahkan lihat contoh berikut:

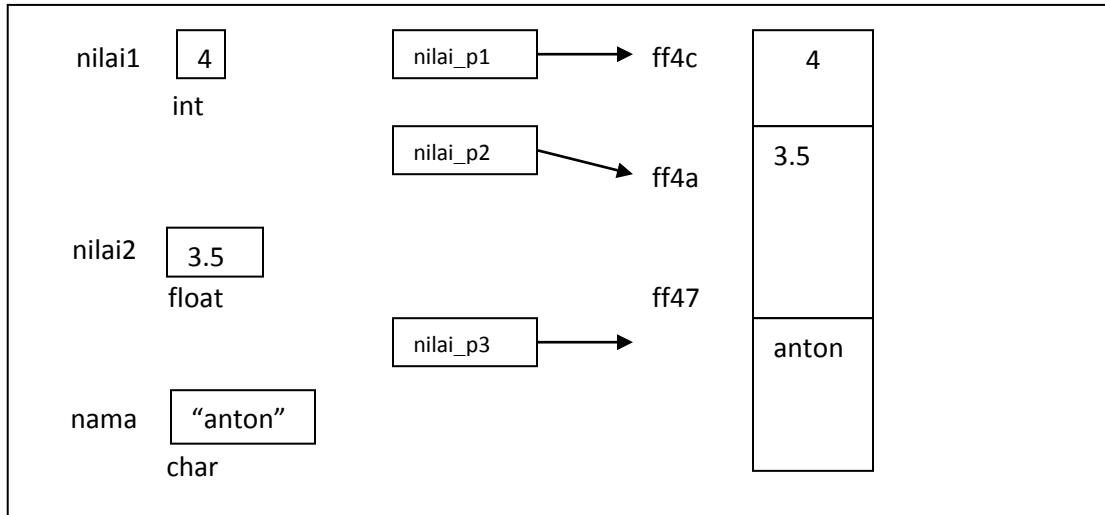
Pendeklarasian variabel biasa dan pointer:

```
//variabel biasa  
int nilai1 = 4;  
float nilai2 = 3.5;  
char nama[10] = "anton"; //array of char (string)
```

```
//variabel pointer
```

```
int *nilai_p1; //dangling pointer  
int *nilai_p2 = &nilai1; //menunjuk ke tipe data int  
char *nilai_p3 = nama; //menunjuk ke tipe data array of char  
char *nilai_p4 = &nama[0];
```

Ilustrasi Pointer:



Contoh program untuk dicoba:

```
#include <stdio.h>
#include <conio.h>

int main(){
    int nilai1 = 4;
    int nilai2 = 5;
    float nilai3 = 3.5;
    char nama[11] = "abcdefghijkl";
    int *nilai_p1 = &nilai1;
    int *nilai_p2 = &nilai2;
    char *nilai_p4 = nama;
    float *nilai_p3= &nilai3;

    printf("nilai 1 = %d, alamat1 = %p,
ukuran %d\n",*nilai_p1,&nilai_p1,sizeof(nilai1));
    printf("nilai 2 = %d, alamat2 = %p,
ukuran %d\n",*nilai_p2,&nilai_p2,sizeof(nilai2));
    printf("nilai 3 = %f, alamat3 = %p,
ukuran %d\n",*nilai_p3,&nilai_p3,sizeof(nilai3));
}
```

```

    printf("nilai 4 = %s, alamat4 = %p,
ukuran %d\n",nama,&nilai_p4,sizeof(nama));
    getch();
    return 0;
}

```

Operasi pointer:

1. Operasi Pemberian nilai
Contoh 1:

```

#include <stdio.h>
#include <conio.h>

int main(){
    float nilai,*p1,*p2;
    nilai = 14.54;
    printf("nilai = %2.2f, alamatnya %p\n",nilai,&nilai);
    p1 = &nilai;
    printf("nilai p1 = %2.2f, p1 menunjuk alamat %p\n",*p1,p1);
    //pada awalnya p2 masih dangling pointer
    printf("mula-mula nilai p2 = %2.2f, p2 menunjuk
alamat %p\n",*p2,p2);
    p2 = p1;      //operasi pemberian nilai, berarti alamat x2 sama
dengan x1
    printf("sekarang nilai p2 = %2.2f, p2 menunjuk
alamat %p\n",*p2,p2);
    getch();
}

```

Contoh 2:

```

#include <stdio.h>
#include <conio.h>

int main(){
    int *p,a=25,b;
    //p masih dangling
}

```

```

printf("nilai a = %d di alamat a = %p,\n",a,&a);
printf("nilai p di alamat = %p\n",p);
p = &a;
printf("nilai p = %d di alamat %p\n",*p,p);
//b diisi dengan nilai yang berasal dari nilai
//variabel a yang ditunjuk oleh pointer p
b = *p;
printf("nilai b = %d di alamat %p\n",b,&b);
getch();
}

```

Contoh 3:

```

#include <conio.h>
#include <stdio.h>

int main(){
    int a=25,b=12,t;
    int *p,*q;
    p = &a;
    q = &b;
    printf("nilai yang ditunjuk p = %d di alamat %p\n",*p,p);
    printf("nilai yang ditunjuk q = %d di alamat %p\n",*q,q);
    //Contoh kasus, penukaran nilai 2 variabel dengan pointer
    t = *p;
    *p = *q;
    *q = t;
    printf("nilai yang ditunjuk p sekarang = %d di
alamat %p\n",*p,p);
    printf("nilai yang ditunjuk q sekarang = %d di
alamat %p\n",*q,q);
    getch();
}

```

Contoh 4:

```

#include <stdio.h>
#include <conio.h>

```

```

int main(){
    int a,*p;
    p=&a;
    *p=25;
    printf("nilai a = %d",a);
}

```

2. Operasi Aritmatika

```

#include <stdio.h>
#include <conio.h>

int main(){
    int a,b=10,*p,*q;
    p=&a;
    *p=25;
    printf("nilai a = %d\n",a);
    printf("alamat p = %p\n",p);
    q=&b;
    printf("alamat q = %p\n",q);
    printf("nilai a + b = %d\n",(*p+*q));
    //posisi alamat p menjadi bergeser, nilai berubah
    p=p+1;
    printf("nilai p = %d, alamat = %p\n",*p,&p);
    q=q-1;
    printf("nilai q = %d, alamat = %p\n",*q,&q);
    getch();
}

```

Pointer pada array 1 dimensi

```

#include <stdio.h>
#include <conio.h>

int main(){
    char S[] = "anton";
    char *p;

```

```

//cara 1
//langsung menunjuk nama array.
p=S;
for(int i=0;i<5;i++) {
    printf("%c",*p);
    p++;
}
printf("\n");

//cara 2
p=&S[0];
for(int i=0;i<5;i++) {
    printf("%c",*p);
    p++;
}
printf("\n");

//Membalik kalimat
p--;
for(int i=0;i<5;i++) {
    printf("%c",*p);
    p--;
}
printf("\n");

getch();
}

```

Pointer pada Struct:

```

#include <stdio.h>
#include <conio.h>

typedef struct{
    int nim;
    int umur;
    float ipk;
} Mahasiswa;

```

```

Mahasiswa m;
Mahasiswa *p = &m;

int main(){
    //struct biasa
    m.nim=123;
    m.ipk=3.2;
    m.umur=23;
    printf("nim = %d\n",m.nim);
    printf("ipk = %f\n",m.ipk);
    printf("umur = %d\n",m.umur);

    //struct pointer
    p->ipk = 3.5;
    p->nim = 321;
    p->umur = 32;
    printf("nim = %d\n",p->nim);
    printf("ipk = %f\n",p->ipk);
    printf("umur = %d\n",p->umur);

    //mengacu pada variabel aslinya
    printf("nim = %d\n",m.nim);
    printf("ipk = %f\n",m.ipk);
    printf("umur = %d\n",m.umur);
    getch();
}

```

Pengembangan:

Buatlah sebuah program untuk mengecek apakah suatu kata palindrom atau bukan, tanpa memperhatikan spasi dan huruf besar/kecilnya. Program dibuat dengan menggunakan template struct sebagai berikut:

```

typedef struct{
    char elemen[30];
    int jml_kata;
} Kata;

```

```
Kata kata;  
Kata *p_kata=&kata;
```

Lanjutkanlah program berikut agar hasilnya sesuai dengan soal di atas:

```
#include <stdio.h>  
#include <conio.h>  
  
typedef struct{  
    char elemen[30];  
    int jml_kata;  
} Kata;  
  
Kata kata;  
Kata *p_kata=&kata;  
  
int main(){  
    char kalimat[30];  
    p_kata->jml_kata=0;  
    char *p = p_kata->elemen;  
    printf("Masukkan kata : ");gets(kalimat);  
    fflush(stdin);  
    printf("Kalimat : %s\n",kalimat);  
    for(int i=0;i<kalimat[i];i++){  
        *p=kalimat[i];  
        p_kata->jml_kata++;  
        p++;  
    }  
  
    p=p_kata->elemen;  
    //tampilkan kembali kalimat tersebut  
    for(int i=0;i<=p_kata->jml_kata;i++){  
        printf("%c ",*p);  
        p++;  
    }  
  
    //kembangkan....
```

```
    getch();  
}
```

Soal:

Buatlah program data KTP, dengan menggunakan pointer pada struct KTP sebagai berikut:

```
typedef struct  
{  
    int tgl;  
    int bln;  
    int th;  
}Tanggal;  
  
typedef struct  
{  
    char noID[5];  
    char nama[30];  
    char jenis_kelamin; // 'L' atau 'P'  
    Tanggal t;  
}KTP;  
  
typedef struct  
{  
    KTP ktp;  
    int jml;  
}Data_KTP;  
  
Data_KTP data_ktp;  
Data_KTP *p;
```

Buatlah fungsi untuk:

1. Menambah data
2. Mencari data berdasarkan tahun kelahiran tertentu
3. Menampilkan data berdasarkan L dan P
4. Mengedit data

Semua pengaksesan dilakukan dengan menggunakan pointer.

=====

Daftar Pustaka

Cheltenham Computer Training, C Programming, www.cctglobal.com, United Kingdom, 1997

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Linked List, Insert Awal dan Insert Tengah

Fakultas

Ilmu Komputer

Program Studi

Sistem
Informasi

Tatap Muka

09

Kode MK

MK87042

Disusun Oleh

Tim Dosen

Abstract

Memberikan gambaran umum mengenai konsep linked list dan mampu mengimplementasikan linked list untuk pemecahan masalah pemrograman yang lebih kompleks.

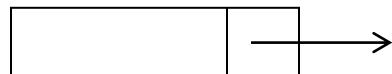
Kompetensi

Mahasiswa dapat memahami definisi dan konsep linked list, mampu mendeklarasikan linked list dalam bahasa C, dan mampu membuat program dengan bantuan linked list.

Pengertian Linked List

Linked list adalah rangkaian elemen yang saling berhubungan/berkaitan, dimana setiap elemen dihubungkan dengan elemen lainnya oleh sebuah pointer. Pointer adalah sel yang nilainya merupakan alamat sel yang lain, sel yang lain itu dapat berupa data atau berupa pointer juga. Jadi setiap elemen dalam linked list selalu berisi pointer.

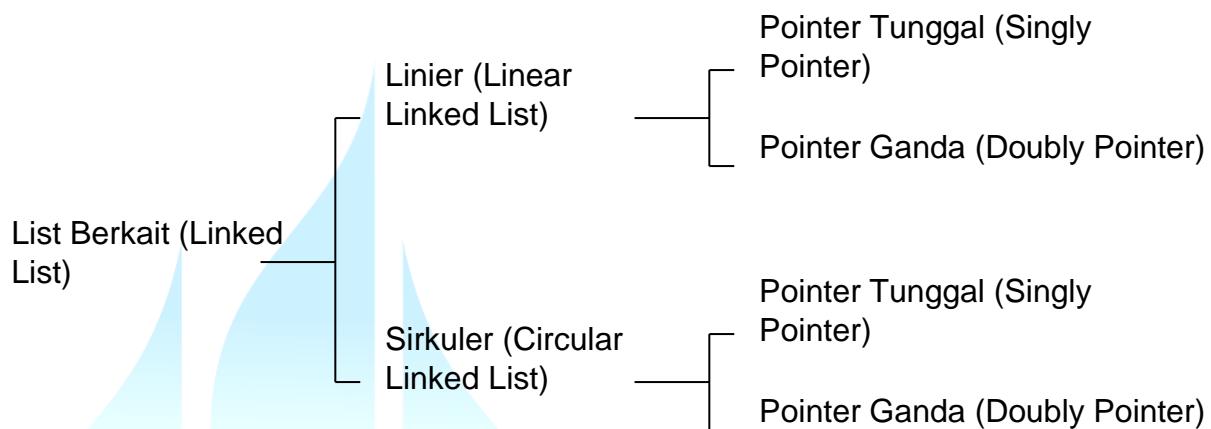
Informasi/Data Pointer

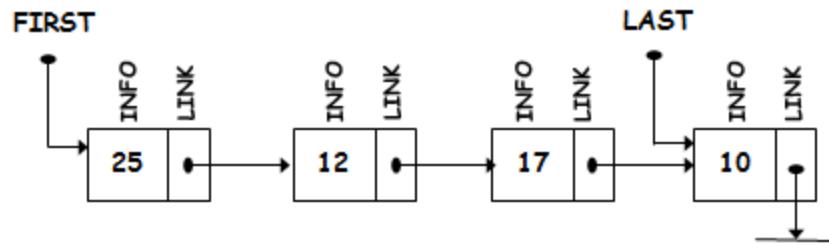


Gambar Elemen Dasar sebuah pointer

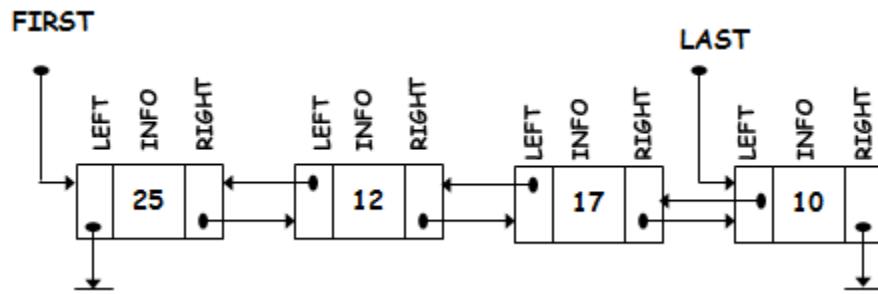
Pada umumnya suatu linked list memiliki sejumlah elemen, dimana salah satu elemen yang paling awal bersifat agak khusus dan tidak digunakan untuk menyimpan data. Elemen paling awal ini dikenal dengan nama elemen kepala atau Head. Berikut ini adalah contoh suatu list sederhana dengan sebuah elemen kepala dan sejumlah elemen biasa lainnya.

Jenis-jenis Linked List

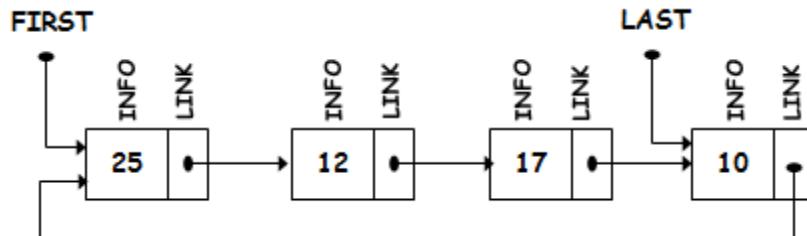




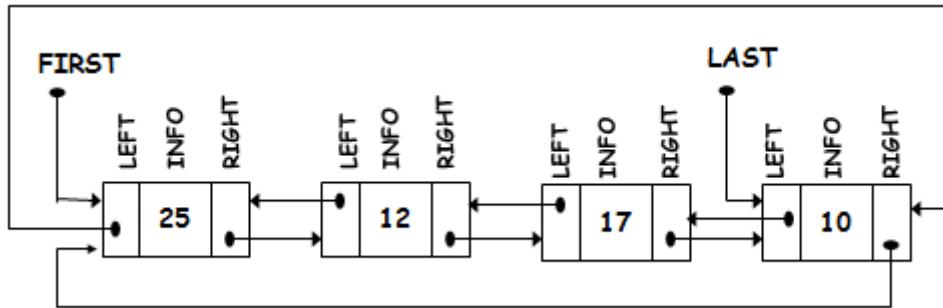
Gambar 1. Linear Singly Linked List



Gambar 2. Linear Doubly Linked List



Gambar 3. Circular Singly Linked List



Gambar 4. Circular Doubly Linked List

LINEAR SINGLY LINKED LIST (Linked List Lurus dengan Pointer Tunggal)

Satu elemen terdiri dari 2 bagian utama:

- Bagian/Field yang menyimpan data
- Bagian/Field yang menyimpan alamat record_next (selanjutnya)

Proses-proses yang ada pada Linked List

- Pembuatan Record Awal (inisialisasi)
- Insert Kanan (Insert Akhir)
- Insert Kiri (Insert Awal)
- Insert Tengah
- Delete Kanan
- Delete Kiri
- Delete Tengah

Ilustrasi sebuah Simpul (Vertex, atau Node, atau Record)

5.03



Simpul dengan 2 elemen atau field

Nama field: LINK

Tipe: pointer

Isi: akan diisi dengan
alamat record berikutnya

Nama field: INFO

Tipe: integer

Isi: akan diisi data

Tipe data tentunya disesuaikan
dengan keperluan. Disini
diambil saja contoh yang
sederhana dimana data yang
akan disimpan adalah bernilai
integer.

Struktur sebuah Simpul



Untuk 'memberitahukan komputer'
bahwa kita memerlukan suatu Simpul atau
record dengan tipe strukur demikian ini,
perlu ditulis intruksi-instruksi sebagai berikut :

```
typedef struct Node {  
    int INFO;  
    struct Node *LINK;  
};  
typedef struct Node Simpul;  
Simpul *P, *FIRST, *LAST;
```

Masih banyak cara penulisan lain untuk maksud yang sama.

Disini diambil suatu cara yang dianggap paling sederhana.

Disiapkan 3 buah variabel pointer, yaitu : P, FIRST dan LAST yang kesemuanya berkaitan dengan simpul atau record atau node yang bertipe Simpul

Disini Simpul :

Adalah nama tipe sebagaimana nama tipe yang telah disediakan oleh Bahasa C seperti : int, long int, float, dan sebagainya. Bila int adalah nama tipe suatu variable, maka Simpul disini adalah nama tipe dari suatu *structure* atau *record*, dimana structure tersebut terdiri dari dua *field*, yaitu :

INFO yang bertipe integer (int), dan
LINK yang bertipe pointer (pakai *) untuk structure tersebut.

1) Pembuatan Record/simpul Awal (inisialisasi)

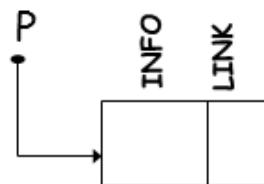
Instruksi untuk membuat sebuah record baru :

```
P = (Simpul *) malloc(sizeof(Simpul));
```

Terbentuk sebuah simpul yang alamatnya disimpan dalam pointer P atau disebut : "simpul yang ditunjuk oleh pointer P" yang dapat diilustrasikan sebagai berikut :

malloc :

Maksudnya : memory allocation yaitu mengalokasikan memory sebesar atau seukuran (sizeof) yang diperlukan untuk Simpul



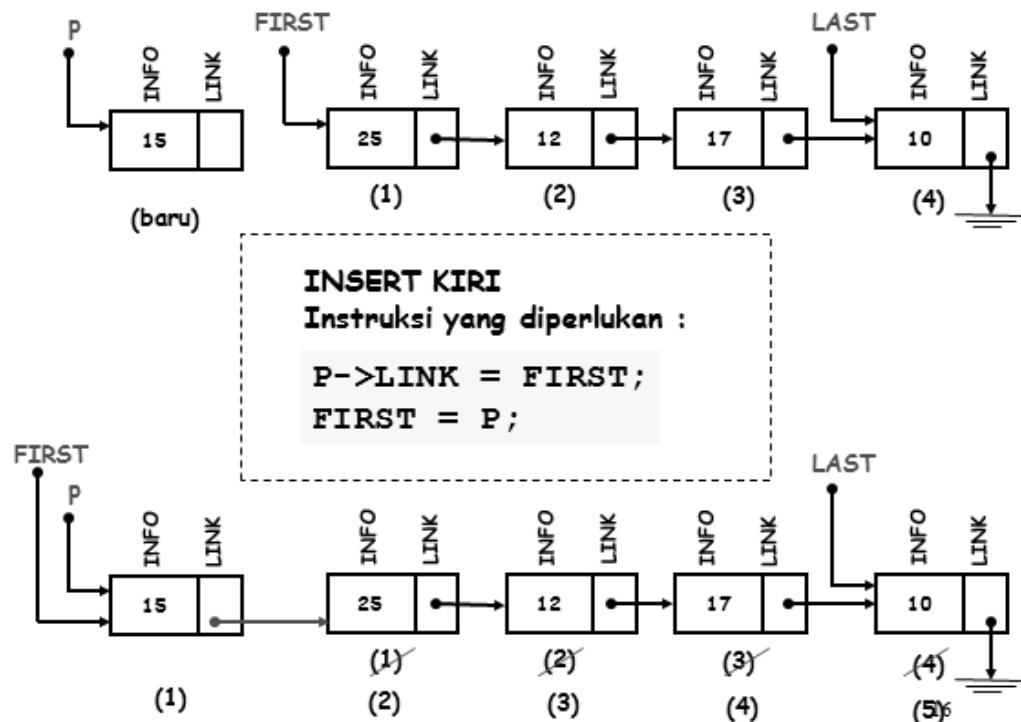
Contoh (sederhana) program selengkapnya untuk membuat sebuah record atau simpul yang alamat simpul tersebut dicatat dalam pointer P.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
typedef struct Node {
    int INFO;
    struct Node *LINK;
};

typedef struct Node Simpul;
Simpul *P, *FIRST, *LAST;
main()
{
    int X;
    scanf("%i", &X);
    P = (Simpul *)alloc(sizeof(Simpul));
    P->INFO = X;
    P->LINK = NULL;
}
```

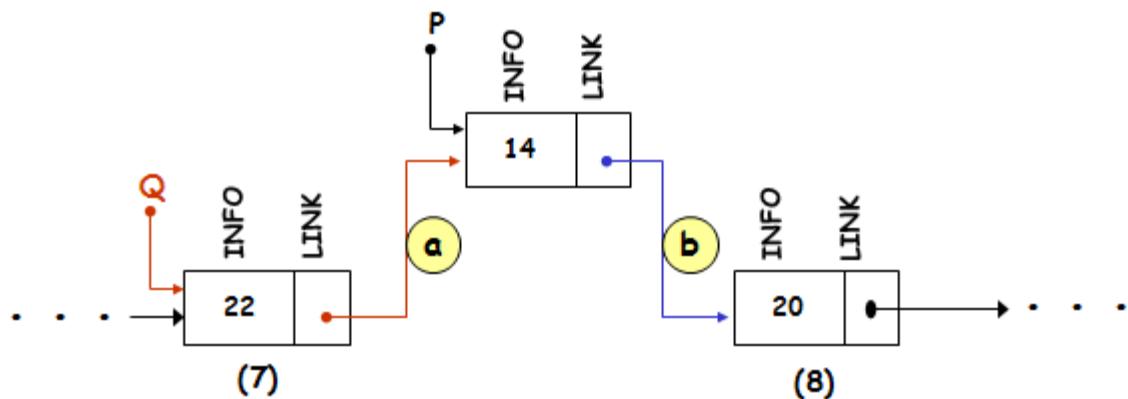
2) INSERT KIRI

Insert adalah menyisipkan record baru pada awal linked list.



3) INSERT TENGAH

Insert adalah menyisipkan record baru pada tengah linked list.



Algoritma untuk : INSERT TENGAH
Bila pointer Q sudah menunjuk simpul
Yang disebelah kiri :

Q->LINK = P;

P->LINK = Q->LINK;

Soal:

Ketiklah program dibawah ini :

```
1. #include <conio.h>
2. #include <iostream.h>
3. #include <stdio.h>
4.
5. typedef struct simpul tsimpul;
6. struct simpul
7. {
8.     int info;
9.     tsimpul *next;
10.};
11. tsimpul *awal = NULL, *akhir = NULL, *data, *hapus, *b1, *b2;
12.
13. void muncul()
14. {
15.     b1=awal;
16.     cout<<"\n      DATA ==> ";
17.     while(b1!=NULL)
18.     {
19.         cout<<b1->info;
20.         b1=b1->next;
21.         cout<<"  ";
22.     }
23. }
24.
25. void tambah_awal(int x)
26. {
27.     data=new simpul;
28.     data->info=x;
29.     data->next=NULL;
30.     if(awal==NULL)
31.     {
32.         awal=akhir=data;
```

```

33.    }
34.  else
35.  {
36.    data->next=awal;
37.    awal=data;
38.  }
39.}
40.
41.void tambah_akhir(int x)
42.{ 
43.  data=new simpul;
44.  data->info=x;
45.  data->next=NULL;
46.  if(awal==NULL)
47.  {
48.    awal=akhir=data;
49.  }
50. else
51. {
52.   akhir->next=data;
53.   akhir=data;
54. }
55.}
56.
57.void tambah_tengah(int in,int setelah)
58.{ 
59.  b1=awal;
60.  int ada=0;
61.  while (b1 != NULL)
62.  {
63.    if (b1->info == setelah)
64.    {
65.      ada++;
66.    }
67.    b1 = b1->next;
68.  }
69. if(ada==0)
70.   cout<<"\n-->Data Yang Di Masukkan Tidak Falid<--\n\n";
71. else
72. {
73.   b1=awal;
74.   b2=b1->next;
75.   while(b1->info!=setelah)
76.   {
77.     b1=b1->next;
78.     b2=b2->next;
79.   }
80.   data=new simpul;
81.   data->info=in;
82.   b1->next=data;
83.   if(b1==akhir)
84.   {
85.     akhir=data;
86.   }
87.   data->next=b2;

```

```

88. }
89. }
90.
91.void hapus_awal()
92.{
93.    if(awal==NULL)
94.    {
95.        cout<<"Maaf....List Kosong!!!\n";
96.    }
97.    else if (awal==akhir)
98.    {
99.        hapus=awal;
100.       awal=NULL;
101.       akhir=NULL;
102.    }
103.    else
104.    {
105.        hapus=awal;
106.        awal=awal->next;
107.    }
108. }
109.
110. void hapus_akhir()
111. {
112.    if(awal==NULL)
113.    {
114.        cout<<"Maaf....List Kosong!!!\n";
115.    }
116.    else if(awal==akhir)
117.    {
118.        hapus=awal;
119.        awal=NULL;
120.        akhir=NULL;
121.    }
122.    else
123.    {
124.        b1=awal;
125.        while(b1->next!=akhir)
126.        {
127.            b1=b1->next;
128.        }
129.        hapus=akhir;
130.        b1->next=NULL;
131.        akhir=b1;
132.    }
133. }
134.
135. void hapus_tengah(int x)
136. {
137.     tsimpul *b;
138.     int ada=0;
139.     b=awal;
140.     if(awal==NULL)
141.         cout<<"Maaf....List Kosong!!!\n";
142.     else if(b->info==x)

```

```

143.         hapus_awal();
144.     else if(akhir->info==x)
145.         hapus_akhir();
146.     else
147.     {
148.         while(b!=NULL)
149.         {
150.             if(b->info==x)
151.             {
152.                 ada++;
153.             }
154.             b=b->next;
155.         }
156.         if(ada==0)
157.             cout<<"\n-->Data Yang Di Masukkan Tidak Falid<--\n\n";
158.     else
159.     {
160.         b=awal;
161.         while(b->next->info!=x)
162.         {
163.             b=b->next;
164.         }
165.         b1=b->next;
166.         b2=b1->next;
167.         hapus=b1;
168.         b->next=b2;
169.     }
170. }
171. }
172.
173. void main()
174. {
175.     int pilih,n,nn;
176.     char lagi;
177.     do
178.     {
179.         clrscr();
180.         cout<<"=====\n";
181.         cout<<"||      PROGRAM SINGLE LINKED LIST      ||\n";
182.         cout<<"=====\n";
183.         muncul();
184.         cout<<"\n1. Tambah Awal";
185.         cout<<"\n2. Tambah Akhir";
186.         cout<<"\n3. Tambah Tengah";
187.         cout<<"\n4. Hapus Awal";
188.         cout<<"\n5. Hapus Akhir";
189.         cout<<"\n6. Hapus Tengah\n=====";
190.         cout<<"\nPilihan Anda : ";cin>>pilih;
191.         cout<<"=====\n";
192.         switch(pilih)
193.         {
194.             case 1:
195.                 cout<<"Masukkan Data    : ";cin>>n;
196.                 tambah_awal(n);
197.                 break;

```

```
198.         case 2:
199.             cout<<"Masukkan Data    : ";cin>>n;
200.             tambah_akhir(n);
201.             break;
202.         case 3:
203.             cout<<"Masukkan Data    : ";cin>>n;
204.             cout<<"Dimasukkan Setelah : ";cin>>nn;
205.             tambah_tengah(n,nn);
206.             break;
207.         case 4:
208.             hapus_awal();
209.             break;
210.         case 5:
211.             hapus_akhir();
212.             break;
213.         case 6:
214.             cout<<"data yg ingin di hapus : ";cin>>nn;
215.             hapus_tengah(nn);
216.             break;
217.         }
218.         cout<<"Lagi (y/t)? ";cin>>lagi;
219.     }
220.     while(lagi=='y'||lagi=='Y');
221. }
```

=====

Daftar Pustaka

Cheltenham Computer Training, C Programming, www.cctglobal.com, United Kingdom, 1997

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004

Moh. Sjukani, 2009, Struktur Data dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

**ADT (Abstack Data Type)
Stack, Implementasi Stack
dengan Array**

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

10

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Memberikan gambaran umum mengenai konsep Abstack Data Type pada Stack dan mampu mengimplementasikan Stack dengan

Kompetensi

Mahasiswa dapat memahami definisi Abstack Data Type pada Stack dan konsep Stack dengan Array, mampu mendeklarasikan Stack dalam bahasa

Array untuk pemecahan masalah pemrograman yang lebih kompleks.

C, dan mampu membuat program dengan bantuan Stack.

ADT (Abstrack Data Type) atau Tipe Data Bentukan

Bahasa C memiliki tipe data numerik dan karakter (seperti integer, float, char dan lain-lain). Bagaimana jika kita ingin membuat tipe data baru? ADT adalah tipe data yang dibuat oleh programmer sendiri yang memiliki suatu nama tertentu. ADT dapat berupa tipe data dasar namun diberi nama baru atau berupa kumpulan tipe data berbeda yang diberi nama baru. Untuk pembuatan ADT digunakan keyword `typedef`.

Contoh

```
#include <stdio.h>
#include <conio.h>

typedef int angka;
typedef float pecahan;
typedef char huruf;

void main(){
    clrscr();
    angka umur;
    pecahan pecah;
    huruf h;
    huruf nama[10];

    printf("masukkan umur anda : ");scanf("%d",&umur);
    printf("Umur anda adalah %d",umur);

    printf("\nmasukkan bilangan pecahan : ");scanf("%f",&pecah);
    printf("Bilangan pecahan %f",pecah);

    printf("\nmasukkan huruf : ");h=getche();
    printf("\nHuruf anda %c",h);

    printf("\nmasukkan nama : ");scanf("%s",nama);
    printf("Nama anda %s",nama);

    getch();
}
```

Hasil :

```
C:\TCWIN45\BIN\NONAME00.EXE
nasukkan umur anda : 4
Umur anda adalah 4
nasukkan bilangan pecahan : 2.5
Bilangan pecahan 2.500000
nasukkan huruf : a
Huruf anda a
nasukkan nama : anton
Nama anda anton
```

Struck

Struct adalah tipe data bentukan yang berisi kumpulan variabel-variabel yang bernaung dalam satu nama yang sama. Berbeda dengan array yang berisi kumpulan variabel yang bertipe data sama, struct dapat memiliki variabel-variabel yang bertipe data sama atau berbeda, bahkan bisa menyimpan variabel yang bertipe data array atau struct variabel-variabel yang menjadi anggota struct disebut dengan elemen struct.

Ilustrasi Strack

Struct bisa diumpamakan sebagai sebuah class, misalnya: Mahasiswa

Struct Mahasiswa memiliki property atau atribut atau variabel yang melekat padanya:

- NIM yaitu karakter sejumlah 8
- Nama yaitu karakter
- IPK yaitu bilangan pecahan

Struct hampir mirip dengan class pada Java, namun struct tidak memiliki method atau function.

Struct dapat digunakan dengan cara membuat variabel (analogikan dengan obyek pada Java)

Misalnya :

obyek anton bertipe struct Mahasiswa
obyek erick bertipe struct Mahasiswa

Pendeklarasian dan penggunaan Struct (1)

```
typedef struct Mahasiswa
{
    char NIM[8];
    char nama[50];
    float ipk;
};

//untuk menggunakan struct Mahasiswa dengan membuat variabel mhs dan mhs2 Mahasiswa
mhs,mhs2;

//untuk menggunakan struct Mahasiswa dengan membuat variabel array m; Mahasiswa m[100];
```

Pendeklarasian dan penggunaan Struct (2)

```
struct {
    char NIM[8];
    char nama[50];
    float ipk;
} mhs;
```

Berarti kita sudah mempunyai variabel **mhs** yang bertipe data struct seperti diatas.

Cara penggunaan struct dan pengaksesan elemen-elemennya

- Penggunaan struct dilakukan dengan membuat suatu variabel yang bertipe struct tersebut.
- Pengaksesan elemen struct dilakukan secara individual dengan menyebutkan nama variabel struct diikuti dengan operator titik (.)
- Misalnya dengan struct mahasiswa seperti contoh di atas, kita akan akses elemen-elemennya seperti contoh berikut:

Contoh 1 :

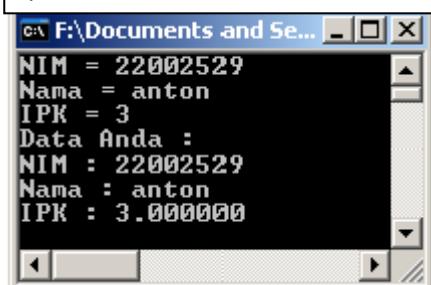
```
#include <stdio.h>
#include <conio.h>

//Pendeklarasian tipe data baru struct Mahasiswa
typedef struct Mahasiswa{
    char NIM[9];
    char nama[30];
    float ipk;
};

void main(){
//Buat variabel mhs bertipe data Mahasiswa
    Mahasiswa mhs;
    clrscr();

    printf("NIM = ");scanf("%s",mhs.NIM);
    printf("Nama = ");scanf("%s",mhs.nama);
    printf("IPK = ");scanf("%f",&mhs.ipk);
    printf("Data Anda : \n");
    printf("NIM : %s\n",mhs.NIM);
    printf("Nama : %s\n",mhs.nama);
    printf("IPK : %f\n",mhs.ipk);

    getch();
}
```



Contoh 2 :

```
#include <stdio.h>
#include <conio.h>
#define phi 3.14

//langsung dianggap variabel 'lingkaran'
struct {
    float jari2;
    float keliling;
    float luas;
} lingkaran;
```

```

//fungsi void untuk menghitung luas lingkaran

void luasLingkaran()
{
    //langsung menggunakan luas lingkaran asli
    lingkaran.luas = lingkaran.jari2 * lingkaran.jari2 * phi;
    printf("\nLuas lingkaran = %f",lingkaran.luas);
}

//fungsi yang mengembalikan nilai float untuk menghitung keliling
lingkaran
float kelLingkaran(float j)
{
    return 2*phi*lingkaran.jari2;
}

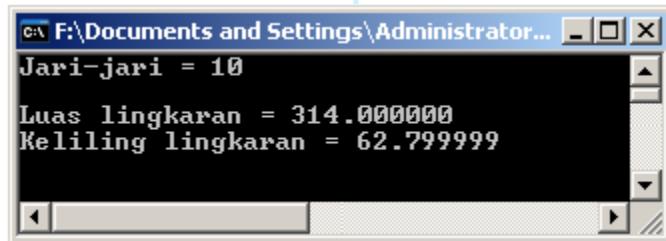
int main()
{
    clrscr();
    printf("Jari-jari = ");scanf("%f",&lingkaran.jari2);

    //panggil fungsi luasLingkaran
    luasLingkaran();

    //panggil fungsi keliling, nilai kembaliannya dikirim ke keliling
    lingkaran.keliling = kelLingkaran(lingkaran.jari2);

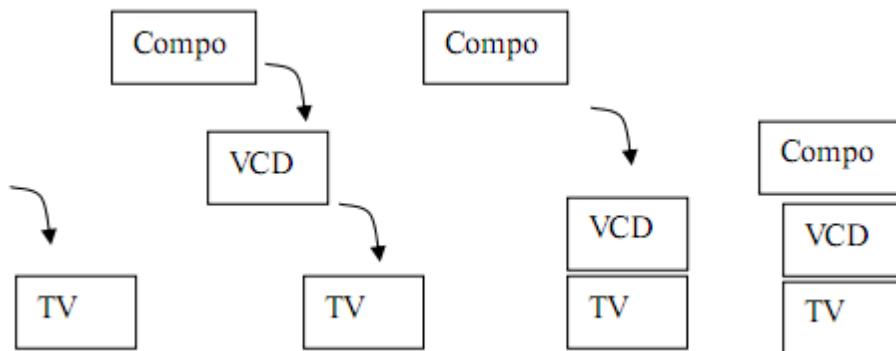
    //tampilkan keliling lingkaran asli
    printf("\nKeliling lingkaran = %f",lingkaran.keliling);
    getch();
}

```



Stack

Stack atau tumpukan adalah suatu struktur data yang penting dalam pemrograman. Bersifat LIFO (Last In First Out). Data yang terakhir masuk ke dalam stack akan menjadi data pertama yang dikeluarkan dari stack. Contohnya, karena kita menumpuk Compo di posisi terakhir, maka Compo akan menjadi elemen teratas dalam tumpukan. Sebaliknya, karena kita menumpuk Televisi pada saat pertama kali, maka elemen Televisi menjadi elemen terbawah dari tumpukan. Dan jika kita mengambil elemen dari tumpukan, maka secara otomatis akan terambil elemen teratas, yaitu Compo juga.



Operasi-operasi/fungsi Stack

- **Push** : digunakan untuk menambah item pada stack pada tumpukan paling atas
- **Pop** : digunakan untuk mengambil item pada stack pada tumpukan paling atas
- **Clear** : digunakan untuk mengosongkan stack
- **IsEmpty** : fungsi yang digunakan untuk mengecek apakah stack sudah kosong
- **IsFull** : fungsi yang digunakan untuk mengecek apakah stack sudah penuh

Stack with Array of Struct

- Definisikan Stack dengan menggunakan struct
- Definisikan MAX_STACK untuk maksimum isi stack
- Buatlah variabel array data sebagai implementasi stack secara nyata

- Deklarasikan operasi-operasi/function di atas dan buat implemetasinya

Deklarasi MAX_STACK

```
#define MAX_STACK 10 //hati-hati mulai dari 0 jadi 0-9
```

Deklarasi STACK dengan struct dan array data

```
typedef struct STACK{  
    int top;  
    char data[10][10]; //misalkan : data adalah array of string  
    //berjumlah 10 data, masing-masing string  
    //menampung maksimal 10 karakter  
};
```

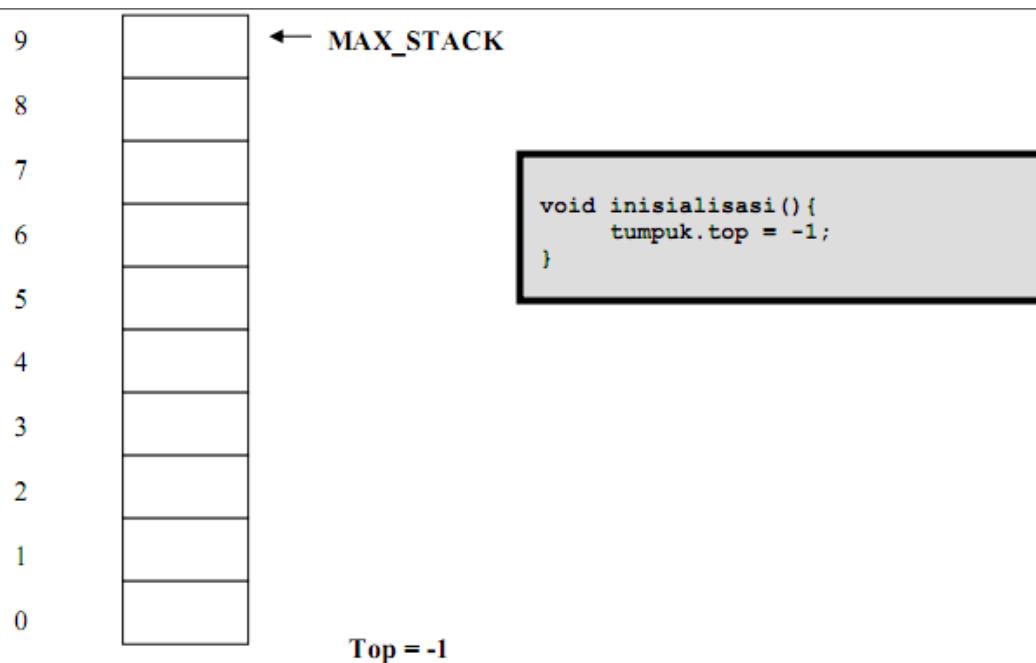
Deklarasi/buat variabel dari struct

```
STACK tumpuk;
```

Inisialisasi Stack

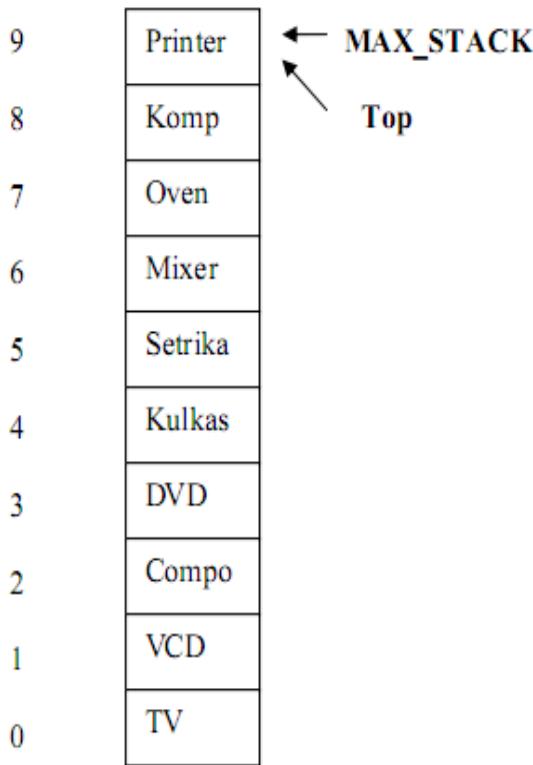
- Pada mulanya isi top dengan -1, karena array dalam C dimulai dari 0, yang berarti stack adalah KOSONG!
- Top adalah suatu variabel penanda dalam STACK yang menunjukkan elemen teratas Stack sekarang. Top Of Stack akan selalu bergerak hingga mencapai MAX of STACK sehingga menyebabkan stack PENUH!
- Ilustrasi stack pada saat inisialisasi:





Fungsi IsFull

- Untuk memeriksa apakah stack sudah penuh?
- Dengan cara memeriksa top of stack, jika sudah sama dengan MAX_STACK-1 maka full, jika belum (masih lebih kecil dari MAX_STACK-1) maka belum full
- Ilustrasi:



```
int IsFull(){
    if(tumpuk.top == MAX_STACK-1)
        return 1;
    else
        return 0;
}
```

Fungsi IsEmpty

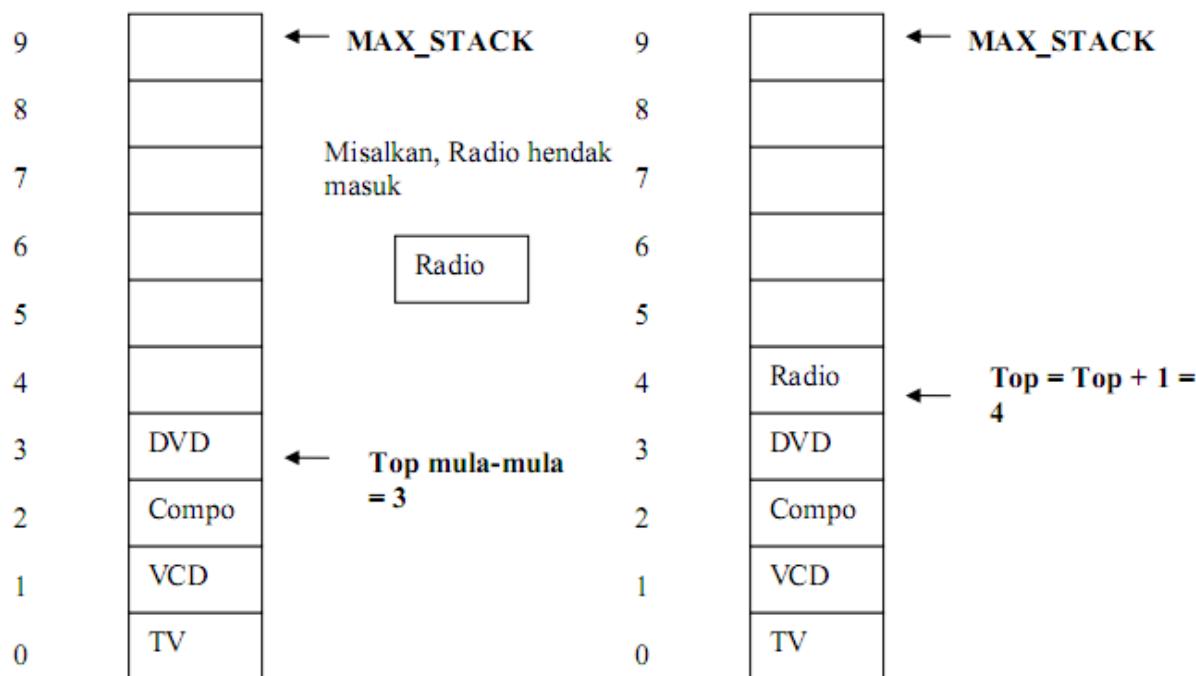
- Untuk memeriksa apakah stack masih kosong?
- Dengan cara memeriksa top of stack, jika masih -1 maka berarti stack masih kosong!
- Program:

```
int IsEmpty(){
    if(tumpuk.top == -1)
        return 1;
    else
        return 0;
}
```

Fungsi Push

- Untuk memasukkan elemen ke stack, selalu menjadi elemen teratas stack

- Tambah satu (increment) nilai top of stack terlebih dahulu setiap kali ada penambahan elemen stack, asalkan stack masih belum penuh, kemudian isikan nilai baru ke stack berdasarkan indeks top of stack setelah ditambah satu (diincrement)
- Ilustrasinya:

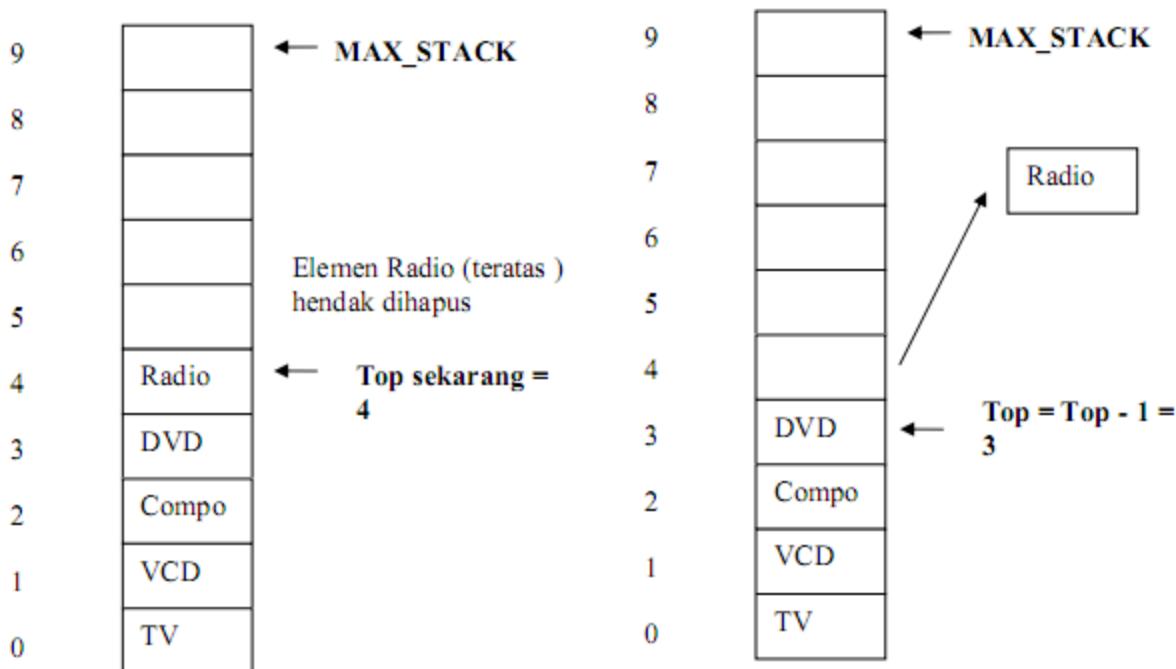


```
void Push(char d[10]){
    tumpuk.top++;
    strcpy(tumpuk.data[tumpuk.top],d);
}
```

Fungsi Pop

- Untuk mengambil elemen teratas dari stack.
- Ambil dahulu nilai elemen teratas stack dengan mengakses top of stack, tampilkan nilai yang akan diambil terlebih dahulu, baru didecrement nilai top of stack sehingga jumlah elemen stack berkurang.

Ilustrasi fungsi POP



```
void Pop(){
    printf("Data yang terambil = %s\n", tumpuk.data[tumpuk.top]);
    tumpuk.top--;
}
```

Program selengkapnya :

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX_STACK 10

typedef struct STACK {
    int top;
    char data[10][10];
};

STACK tumpuk;

void inisialisasi(){
    tumpuk.top = -1;
```

```
}
```

```
int IsFull(){
    if(tumpuk.top == MAX_STACK-1) return 1; else return 0;
}
```

```
int IsEmpty(){
    if(tumpuk.top == -1) return 1; else return 0;
}
```

```
void Push(char d[10]){
    tumpuk.top++;
    strcpy(tumpuk.data[tumpuk.top],d);
}
```

```
void Pop(){
    printf("Data yang terambil = %s\n",tumpuk.data[tumpuk.top]);
    tumpuk.top--;
}
```

```
void Clear(){
    tumpuk.top=-1;
}
```

```
void TampilStack(){
    for(int i=tumpuk.top;i>=0;i--) {
        printf("Data : %s\n",tumpuk.data[i]);
    }
}
```

```
int main(){
    int pil;
    inisialisasi();
    char dt[10];
    do{
        printf("1. push\n");
        printf("2. pop\n");
        printf("3. print\n");
        printf("4. clear\n");
        printf("5. exit\n");
        printf("Pilihan : ");scanf("%d",&pil);
        switch(pil){
            case 1: if(IsFull() != 1){
                printf("Data = ");scanf("%s",dt);
                Push(dt);
            } else printf("\nSudah penuh!\n");
            break;
            case 2: if(IsEmpty() != 1)
                Pop();
            else
                printf("\nMasih kosong!\n");
            break;
        }
    }
}
```

```
case 3: if(IsEmpty() != 1)
          TampilStack();
      else
          printf("\nMasih kosong!\n");
          break;
    case 4: Clear();
          printf("\nSudah kosong!\n");
          break;
    }
    getch();
}while(pil != 5);
getch();
}
```

=====

Daftar Pustaka

Cheltenham Computer Training, C Programming, www.cctglobal.com, United Kingdom, 1997

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004

Moh. Sjukani, 2009, Struktur Data dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

Implementasi Stack dengan Linked-List

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

11

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Memberikan gambaran umum
mengenai konsep Stack dengan Linked-
List dan mengimplementasikan Stack

Kompetensi

Mahasiswa dapat memahami definisi
Abstrack Data Type pada Stack dan
konsep Stack dengan Linked-List,

dengan Linked-List untuk pemecahan masalah pemrograman yang lebih kompleks.

mampu mendeklarasikan Stack dalam bahasa C, dan mampu membuat program dengan bantuan Stack.

Stack dengan Single Linked-List

Selain implementasi stack dengan array seperti telah dijelaskan sebelumnya, ada cara lain untuk mengimplementasi stack dalam C++, yakni dengan single linked list. Keunggulannya jika dibandingkan dengan array, tentu saja adalah penggunaan alokasi memori yang dinamis sehingga menghindari pemborosan memori. Misalnya saja pada stack dengan array disediakan tempat untuk stack berisi 150 elemen, sementara ketika dipakai oleh user stack hanya diisi 50 elemen, maka telah terjadi pemborosan memori untuk sisa 100 elemen, yang tak terpakai. Dengan penggunaan linked list maka tempat yang disediakan akan sesuai dengan banyaknya elemen yang mengisi stack. Oleh karena itu pula dalam stack dengan linked list tidak ada istilah full, sebab biasanya program tidak menentukan jumlah elemen stack yang mungkin ada (kecuali jika sudah dibatasi oleh pembuatnya). Namun demikian sebenarnya stack ini pun memiliki batas kapasitas, yakni dibatasi oleh jumlah memori yang tersedia.

Operasi-operasi untuk Stack dengan Linked List

IsEmpty

Fungsi memeriksa apakah stack yang adamasih kosong.

Push

Fungsi memasukkan elemen baru ke dalam stack. Push di sini mirip dengan insert dalam single linked list biasa.

Pop

Fungsi ini mengeluarkan elemen teratas dari stack.

Contoh

```
#include <stdio.h>
#include <conio.h>

typedef int angka;
typedef float pecahan;
typedef char huruf;

void main(){
    clrscr();
    angka umur;
    pecahan pecah;
    huruf h;
    huruf nama[10];

    printf("masukkan umur anda : ");scanf("%d",&umur);
    printf("Umur anda adalah %d",umur);

    printf("\nmasukkan bilangan pecahan : ");scanf("%f",&pecah);
    printf("Bilangan pecahan %f",pecah);

    printf("\nmasukkan huruf : ");h=getche();
    printf("\nHuruf anda %c",h);

    printf("\nmasukkan nama : ");scanf("%s",nama);
    printf("Nama anda %s",nama);

    getch();
```

Hasil :

```
C:\TCWIN45\BIN\NONAME00.EXE
nasukkan umur anda : 4
Umur anda adalah 4
nasukkan bilangan pecahan : 2.5
Bilangan pecahan 2.500000
nasukkan huruf : a
Huruf anda a
nasukkan nama : anton
Nama anda anton
```

Struck

Struct adalah tipe data bentukan yang berisi kumpulan variabel-variabel yang bernaung dalam satu nama yang sama. Berbeda dengan array yang berisi kumpulan variabel yang bertipe data sama, struct dapat memiliki variabel-variabel yang bertipe data sama atau berbeda, bahkan bisa menyimpan variabel yang bertipe data array atau struct variabel-variabel yang menjadi anggota struct disebut dengan elemen struct.

Ilustrasi Strack

Struct bisa diumpamakan sebagai sebuah class, misalnya: Mahasiswa

Struct Mahasiswa memiliki property atau atribut atau variabel yang melekat padanya:

- NIM yaitu karakter sejumlah 8
- Nama yaitu karakter
- IPK yaitu bilangan pecahan

Struct hampir mirip dengan class pada Java, namun struct tidak memiliki method atau function.

Struct dapat digunakan dengan cara membuat variabel (analogikan dengan

Pendeklarasian dan penggunaan Struct (1)

```
typedef struct Mahasiswa
{
    char NIM[8];
    char nama[50];
    float ipk;
};

//untuk menggunakan struct Mahasiswa dengan membuat variabel mhs dan mhs2 Mahasiswa
mhs,mhs2;

//untuk menggunakan struct Mahasiswa dengan membuat variabel array m; Mahasiswa m[100];
```

Pendeklarasian dan penggunaan Struct (2)

```
struct {
    char NIM[8];
    char nama[50];
    float ipk;
} mhs;
```

Berarti kita sudah mempunyai variabel **mhs** yang bertipe data struct seperti diatas.

Cara penggunaan struct dan pengaksesan elemen-elemennya

- Penggunaan struct dilakukan dengan membuat suatu variabel yang bertipe struct tersebut.

- Pengaksesan elemen struct dilakukan secara individual dengan menyebutkan nama variabel struct diikuti dengan operator titik (.)
- Misalnya dengan struct mahasiswa seperti contoh di atas, kita akan akses elemen-elemennya seperti contoh berikut:

Contoh 1 :

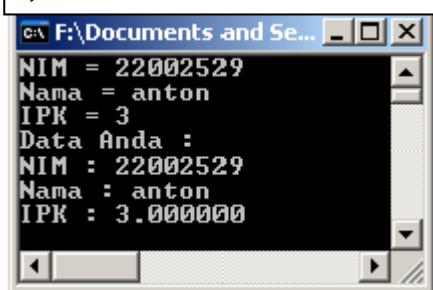
```
#include <stdio.h>
#include <conio.h>

//Pendeklarasian tipe data baru struct Mahasiswa
typedef struct Mahasiswa{
    char NIM[9];
    char nama[30];
    float ipk;
};

void main(){
//Buat variabel mhs bertipe data Mahasiswa
    Mahasiswa mhs;
    clrscr();

    printf("NIM = ");scanf("%s",mhs.NIM);
    printf("Nama = ");scanf("%s",mhs.nama);
    printf("IPK = ");scanf("%f",&mhs.ipk);
    printf("Data Anda : \n");
    printf("NIM : %s\n",mhs.NIM);
    printf("Nama : %s\n",mhs.nama);
    printf("IPK : %f\n",mhs.ipk);

    getch();
}
```



```
F:\Documents and Se...
NIM = 22002529
Nama = anton
IPK = 3
Data Anda :
NIM : 22002529
Nama : anton
IPK : 3.000000
```

Contoh 2 :

```
#include <stdio.h>
#include <conio.h>
#define phi 3.14

//langsung dianggap variabel 'lingkaran'
struct {
    float jari2;
    float keliling;
    float luas;
} lingkaran;

//fungsi void untuk menghitung luas lingkaran

void luasLingkaran()
{
    //langsung menggunakan luas lingkaran asli
    lingkaran.luas = lingkaran.jari2 * lingkaran.jari2 * phi;
    printf("\nLuas lingkaran = %f",lingkaran.luas);
}

//fungsi yang mengembalikan nilai float untuk menghitung keliling
//lingkaran
float kelLingkaran(float j)
{
    return 2*phi*lingkaran.jari2;
}

int main()
{
    clrscr();
    printf("Jari-jari = ");scanf("%f",&lingkaran.jari2);

    //panggil fungsi luasLingkaran
    luasLingkaran();

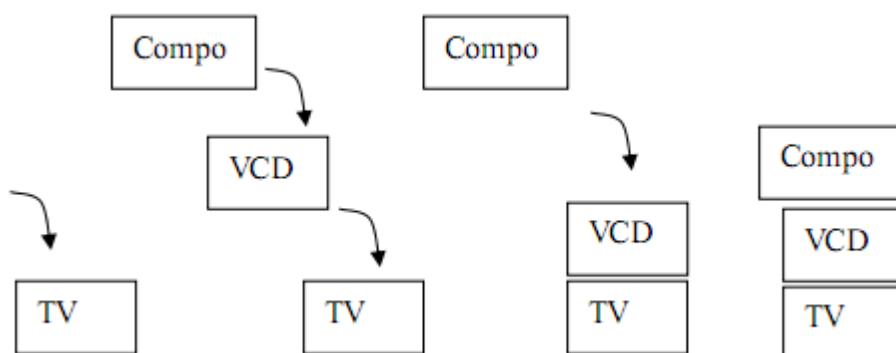
    //panggil fungsi keliling, nilai kembalinya dikirim ke keliling
    //lingkaran asli
    lingkaran.keliling = kelLingkaran(lingkaran.jari2);

    //tampilkan keliling lingkaran asli
    printf("\nKeliling lingkaran = %f",lingkaran.keliling);
    getch();
}
```

```
E:\Documents and Settings\Administrator... Jari-jari = 10 Luas lingkaran = 314.000000 Keliling lingkaran = 62.799999
```

Stack

Stack atau tumpukan adalah suatu struktur data yang penting dalam pemrograman. Bersifat LIFO (Last In First Out). Data yang terakhir masuk ke dalam stack akan menjadi data pertama yang dikeluarkan dari stack. Contohnya, karena kita menumpuk Compo di posisi terakhir, maka Compo akan menjadi elemen teratas dalam tumpukan. Sebaliknya, karena kita menumpuk Televisi pada saat pertama kali, maka elemen Televisi menjadi elemen terbawah dari tumpukan. Dan jika kita mengambil elemen dari tumpukan, maka secara otomatis akan terambil elemen teratas, yaitu Compo juga.



Operasi-operasi/fungsi Stack

- **Push** : digunakan untuk menambah item pada stack pada tumpukan paling atas
- **Pop** : digunakan untuk mengambil item pada stack pada tumpukan paling atas
- **Clear** : digunakan untuk mengosongkan stack

- **IsEmpty** : fungsi yang digunakan untuk mengecek apakah stack sudah kosong
- **IsFull** : fungsi yang digunakan untuk mengecek apakah stack sudah penuh

Stack with Array of Struct

- Definisikan Stack dengan menggunakan struct
- Definisikan MAX_STACK untuk maksimum isi stack
- Buatlah variabel array data sebagai implementasi stack secara nyata
- Deklarasikan operasi-operasi/function di atas dan buat implemetasinya

Deklarasi MAX_STACK

```
#define MAX_STACK 10 //hati-hati mulai dari 0 jadi 0-9
```

Deklarasi STACK dengan struct dan array data

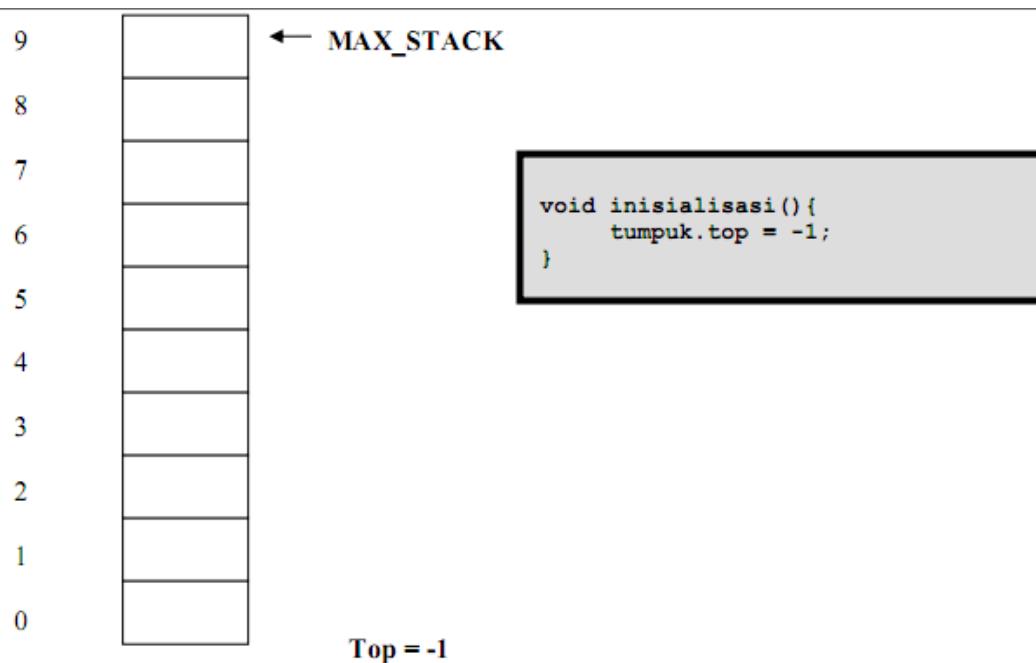
```
typedef struct STACK{
    int top;
    char data[10][10]; //misalkan : data adalah array of string
    //berjumlah 10 data, masing-masing string
    //menampung maksimal 10 karakter
};
```

Deklarasi/buat variabel dari struct

```
STACK tumpuk;
```

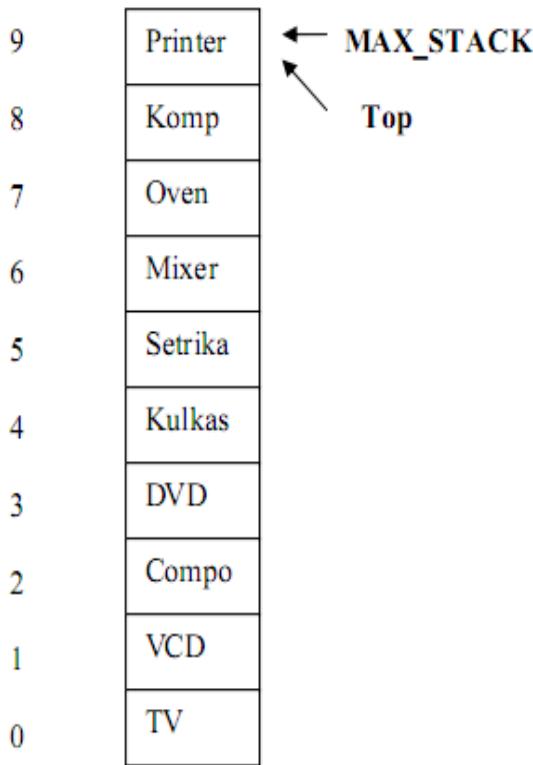
Inisialisasi Stack

- Pada mulanya isi top dengan -1, karena array dalam C dimulai dari 0, yang berarti stack adalah KOSONG!
- Top adalah suatu variabel penanda dalam STACK yang menunjukkan elemen teratas Stack sekarang. Top Of Stack akan selalu bergerak hingga mencapai MAX of STACK sehingga menyebabkan stack PENUH!
- Ilustrasi stack pada saat inisialisasi:



Fungsi IsFull

- Untuk memeriksa apakah stack sudah penuh?
- Dengan cara memeriksa top of stack, jika sudah sama dengan MAX_STACK-1 maka full, jika belum (masih lebih kecil dari MAX_STACK-1) maka belum full
- Ilustrasi:



```
int IsFull(){
    if(tumpuk.top == MAX_STACK-1)
        return 1;
    else
        return 0;
}
```

Fungsi IsEmpty

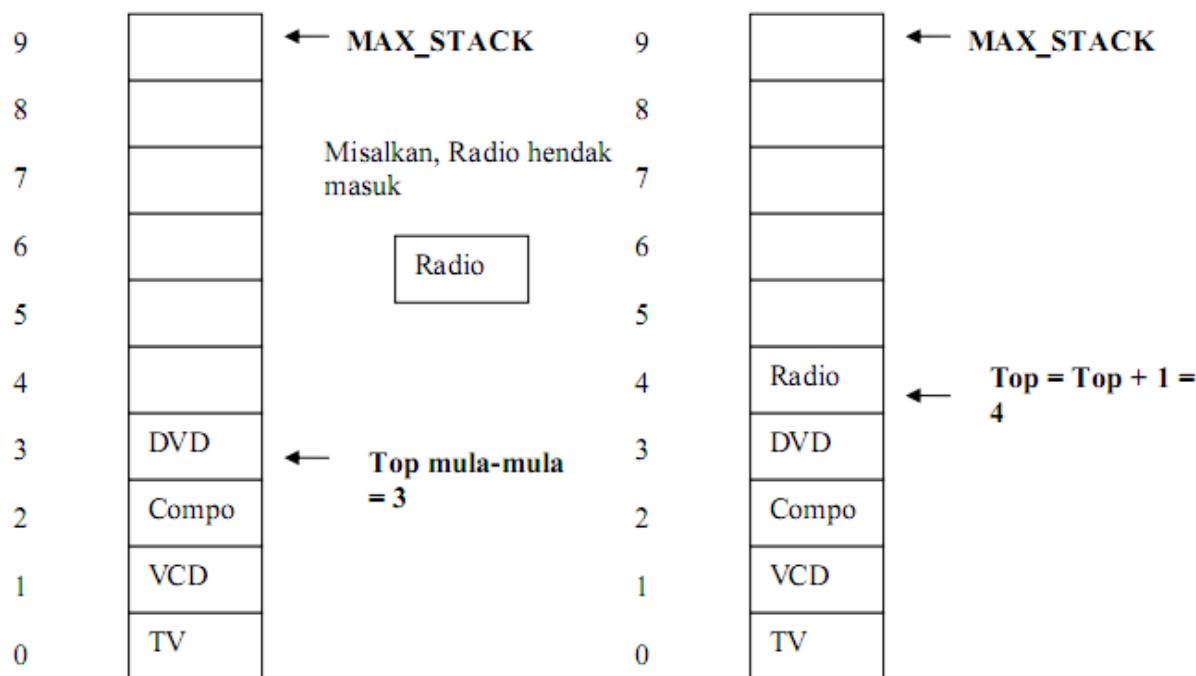
- Untuk memeriksa apakah stack masih kosong?
- Dengan cara memeriksa top of stack, jika masih -1 maka berarti stack masih kosong!
- Program:

```
int IsEmpty(){
    if(tumpuk.top == -1)
        return 1;
    else
        return 0;
}
```

Fungsi Push

- Untuk memasukkan elemen ke stack, selalu menjadi elemen teratas stack

- Tambah satu (increment) nilai top of stack terlebih dahulu setiap kali ada penambahan elemen stack, asalkan stack masih belum penuh, kemudian isikan nilai baru ke stack berdasarkan indeks top of stack setelah ditambah satu (diincrement)
- Ilustrasinya:

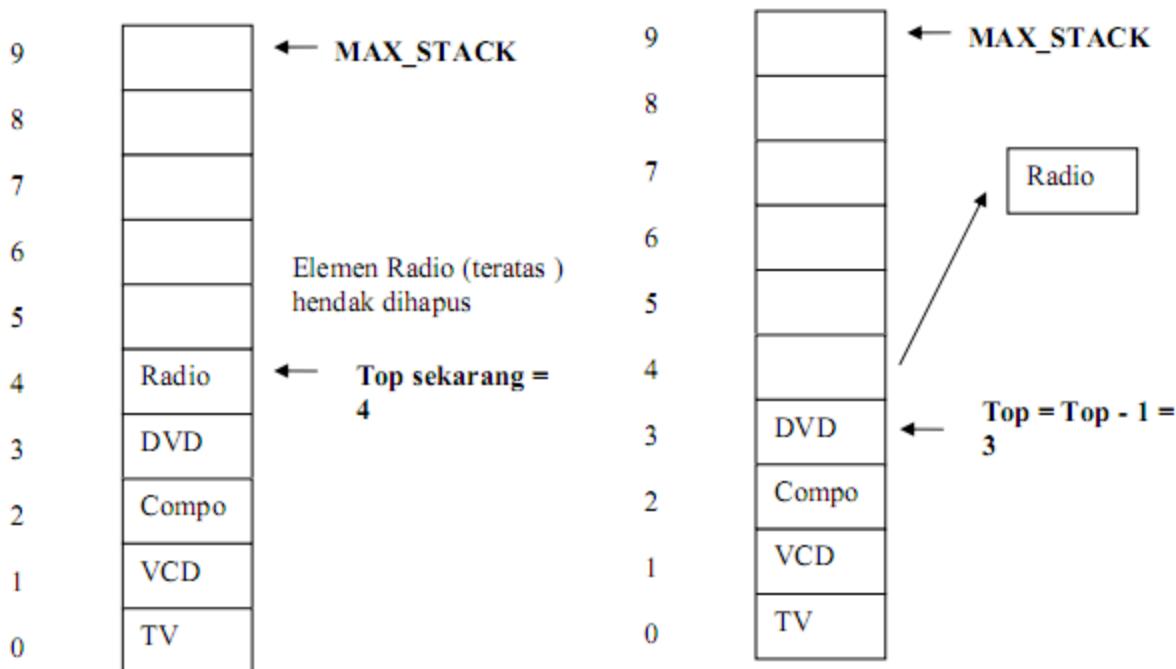


```
void Push(char d[10]){
    tumpuk.top++;
    strcpy(tumpuk.data[tumpuk.top],d);
}
```

Fungsi Pop

- Untuk mengambil elemen teratas dari stack.
- Ambil dahulu nilai elemen teratas stack dengan mengakses top of stack, tampilkan nilai yang akan diambil terlebih dahulu, baru didecrement nilai top of stack sehingga jumlah elemen stack berkurang.

Ilustrasi fungsi POP



```
void Pop(){
    printf("Data yang terambil = %s\n", tumpuk.data[tumpuk.top]);
    tumpuk.top--;
}
```

Program selengkapnya :

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX_STACK 10

typedef struct STACK {
    int top;
    char data[10][10];
};

STACK tumpuk;

void inisialisasi(){
    tumpuk.top = -1;
```

```
}
```

```
int IsFull(){
    if(tumpuk.top == MAX_STACK-1) return 1; else return 0;
}
```

```
int IsEmpty(){
    if(tumpuk.top == -1) return 1; else return 0;
}
```

```
void Push(char d[10]){
    tumpuk.top++;
    strcpy(tumpuk.data[tumpuk.top],d);
}
```

```
void Pop(){
    printf("Data yang terambil = %s\n",tumpuk.data[tumpuk.top]);
    tumpuk.top--;
}
```

```
void Clear(){
    tumpuk.top=-1;
}
```

```
void TampilStack(){
    for(int i=tumpuk.top;i>=0;i--) {
        printf("Data : %s\n",tumpuk.data[i]);
    }
}
```

```
int main(){
    int pil;
    inisialisasi();
    char dt[10];
    do{
        printf("1. push\n");
        printf("2. pop\n");
        printf("3. print\n");
        printf("4. clear\n");
        printf("5. exit\n");
        printf("Pilihan : ");scanf("%d",&pil);
        switch(pil){
            case 1: if(IsFull() != 1){
                printf("Data = ");scanf("%s",dt);
                Push(dt);
            } else printf("\nSudah penuh!\n");
            break;
            case 2: if(IsEmpty() != 1)
                Pop();
            else
                printf("\nMasih kosong!\n");
            break;
        }
    }
}
```

```
case 3: if(IsEmpty() != 1)
          TampilStack();
      else
          printf("\nMasih kosong!\n");
          break;
    case 4: Clear();
          printf("\nSudah kosong!\n");
          break;
    }
    getch();
}while(pil != 5);
getch();
}
```

=====

Daftar Pustaka

Cheltenham Computer Training, C Programming, www.cctglobal.com, United Kingdom, 1997

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004

Moh. Sjukani, 2009, Struktur Data dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

**ADT (Abstack Data Type)
Queue, Implementasi Queue
dengan Array Linear dan
Linked List**

Fakultas

Ilmu Komputer

Program Studi

Sistem
Informasi

Tatap Muka

12

Kode MK

MK87042

Disusun Oleh

Tim Dosen

Abstract

Kompetensi

Memberikan gambaran umum mengenai konsep Abstrack Data Type pada queue dan mampu mengimplementasikan queue dengan Array dan linked list untuk pemecahan masalah pemrograman yang lebih kompleks.

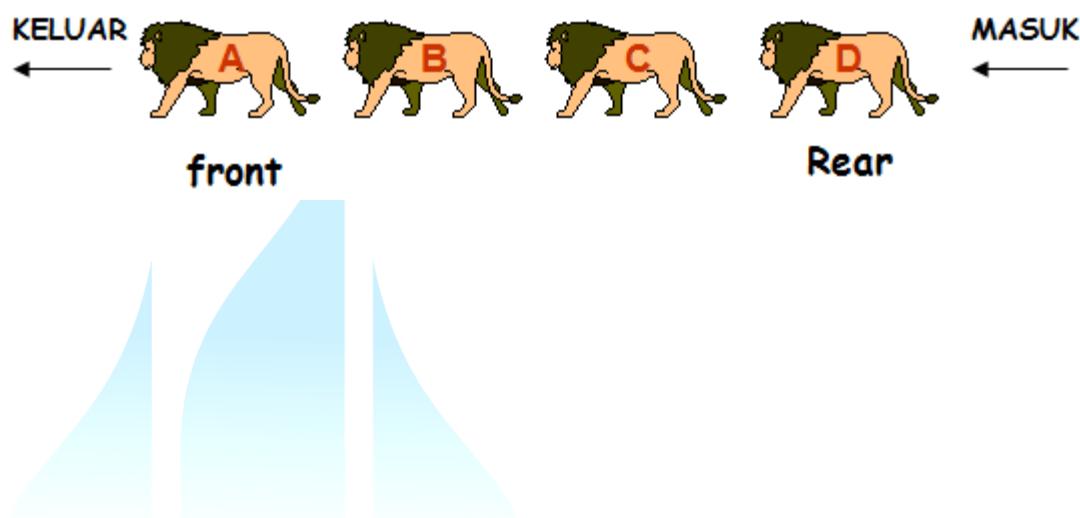
Mahasiswa dapat memahami definisi Abstrack Data Type pada queue dan konsep queue dengan Array dan linked list, mampu mendeklarasikan queue dalam bahasa C, dan mampu membuat program dengan bantuan queue.

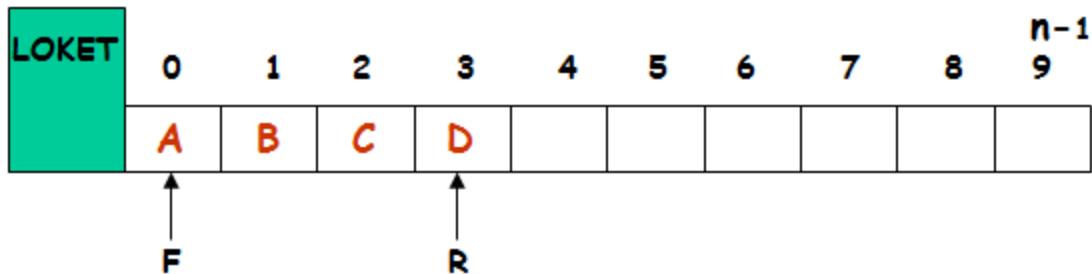
ADT Queue

Queue adalah sebuah antrian. queue merupakan salah satu contoh aplikasi dari pembuatan double linked list yang cukup sering kita temui dalam kehidupan sehari-hari, misalnya saat Anda mengantri di loket untuk membeli tiket. Istilah yang cukup sering dipakai seseorang masuk dalam sebuah antrian adalah **enqueue**. Dalam suatu antrian, yang dating terlebih dahulu akan dilayani lebih dahulu. Istilah yang sering dipakai bila seseorang keluar dari antrian adalah **dequeue**.

Menurut **Yedidyah L, Moshe J. A., and Aaron M. Tenenbaum** dalam bukunya yang berjudul **Data Structures Using C and C++**. Memberikan definisi mengenai queue yaitu “A Queue is an ordered collection of items into which new items may be inserted at one end (called the *rear* of the queue) and from which items may be deleted at one end (called the *front* of the queue)”.

Ilustrasi sebuah queue :





Walaupun berbeda implementasi, struktur data queue setidaknya harus memiliki operasi-operasi sebagai berikut :

EnQueue	Memasukkan data ke dalam antrian
DeQueue	Mengeluarkan data terdepan dari antrian
Clear	Menghapus seluruh antrian
IsEmpty	Memeriksa apakah antrian kosong
IsFull	Memeriksa apakah antrian penuh

Implementasi Queue dengan Linear Array

Linear array adalah suatu array yang dibuat seakan-akan merupakan suatu garis lurus dengan satu pintu masuk dan satu pintu keluar. Berikut ini diberikan deklarasi kelas Queue Linear sebagai implementasi dari Queue menggunakan linear array. Dalam prakteknya, anda dapat menggantinya sesuai dengan kebutuhan Anda. Data diakses dengan field data, sedangkan indeks item pertama dan terakhir disimpan dalam field Head dan Tail. Konstruktor akan menginisialisasikan nilai Head dan Tail dengan -1 untuk menunjukkan bahwa antrian masih kosong dan mengalokasikan data sebanyak MAX_QUEUE yang ditunjuk oleh Data. Destruktor akan mengosongkan antrian kembali dan mendealokasikan memori yang digunakan oleh antrian.

Operasi-Operasi Queue dengan Linear Array

IsEmpty

Fungsi IsEmpty berguna untuk mengecek apakah queue masih kosong atau sudah berisi data. hal ini dilakukan dengan mengecek apakah tail bernilai -1 atau tidak. Nilai -1 menandakan bahwa queue masih kosong.

IsFull

Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bisa menampung data dengan cara mengecek apakah nilai tail sudah sama dengan jumlah maksimal queue. Jika nilai keduanya sama, berarti queue sudah penuh.

EnQueue

Fungsi EnQueue berguna untuk memasukkan sebuah elemen dalam queue.

DeQueue

Fungsi DeQueue berguna untuk mengambil sebuah elemen dari queue. Operasi ini sering disebut juga serve. Hal ini dilakukan dengan cara memindahkan sejauh satu langkah ke posisi di depannya sehingga otomatis elemen yang paling depan akan tertimpa dengan elemen yang terletak di belakangnya.

Clear

Fungsi Clear berguna untuk menghapus semua elemen dalam queue dengan jalan mengeluarkan semua elemen tersebut satu per satu hingga queue kosong dengan memanfaatkan fungsi DEQueue.

Implementasi Queue dengan Double Linked List

Selain menggunakan array, queue juga dapat dibuat dengan linked list. Metode linked list yang digunakan adalah double linked list.

Operasi-operasi Queue dengan Double Linked List

IsEmpty

Fungsi IsEmpty berguna untuk mengecek apakah queue masih kosong atau sudah berisi data. Hal ini dilakukan dengan mengecek apakah head masih menunjukkan pada Null atau tidak. Jika benar berarti queue masih kosong.

IsFull

Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bisa menampung data dengan cara mengecek apakah Jumlah Queue sudah sama dengan MAX_QUEUE atau belum. Jika benar maka queue sudah penuh.

EnQueue

Fungsi EnQueue berguna untuk memasukkan sebuah elemen ke dalam queue (head dan tail mula-mula meunjukkan ke NULL).

DeQueue

Procedure DeQueue berguna untuk mengambil sebuah elemen dari queue. Hal ini dilakukan dengan cara menghapus satu simpul yang terletak paling depan (head).

Contoh Program :

1. Queue dengan Menggunakan Array

```

#include <iostream.h>
#include <stdlib.h>

#define MAX 10 //Ukuran Maksimum Queue

void insert (int queue[], int *rear, int nilai);
void del (int queue[], int *front, int rear, int *nilai);

void main()
{
    int queue[MAX];
    int front, rear;
    int n, nilai;

    front = rear = (-1);
    do
    {
        do
        {
            cout<<"Masukkan Nilai Elemen : ";
            cin>>nilai;
            insert (queue,&rear,nilai);

            cout<<endl;
            cout<<"Tekan 1 untuk Melanjutkan : ";
            cin>>n;
        } while (n == 1);

        cout<<endl;
        cout<<"Tekan 1 untuk Menghapus Sebuah Elemen : ";
        cin>>n;

        while (n == 1)
        {
            del(queue,&front,rear,&n);
            cout<<"Nilai telah Dihapus : "<<n<<endl;
            cout<<endl;
            cout<<"Tekan 1 untuk Menghapus Sebuah Elemen : ";
            cin>>n;
        }

        cout<<endl;
        cout<<"Tekan 1 untuk Melanjutkan : ";
        cin>>n;
    } while (n == 1);
}

```

```

void insert (int queue[], int *rear, int nilai)
{
    if (*rear < MAX-1)
    {
        *rear = *rear + 1;
        queue[*rear] = nilai;
    }
    else
    {
        cout<<"Queue Penuh, Insert Tidak Dapat Dilakukan"<<endl;
        exit(0);
    }
}

void del (int queue[], int *front, int rear, int *nilai)
{
    if (*front == rear)
    {
        cout<<"Queue Kosong, Delete Tidak Dapat Dilakukan"<<endl;
        exit(0);
    }

    *front = *front + 1;
    *nilai = queue[*front];
}

```

Output

```

Masukkan Nilai Elemen : 78
Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 85

Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 78

Tekan 1 untuk Melanjutkan : 2

Tekan 1 untuk Menghapus Sebuah Elemen : 1
Nilai telah Dihapus : 78

Tekan 1 untuk Menghapus Sebuah Elemen : 1
Nilai telah Dihapus : 85

Tekan 1 untuk Menghapus Sebuah Elemen : 2

Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 45

Tekan 1 untuk Melanjutkan : 2

Tekan 1 untuk Menghapus Sebuah Elemen : 2

```

2. Queue dengan menggunakan Linked-List

```
#include <iostream.h>
#include <stdlib.h>

#define Nil NULL

struct node
{
    int data;
    struct node *link;
};

void insert (struct node **front, struct node **rear, int nilai)
{
    struct node *temp;

    temp = (struct node *)malloc(sizeof(struct node));

    /*Buat Node Baru dengan Menggunakan Nilai Data sebagai
     parameter */

    if (temp == Nil)
    {
        cout<<"Error, Memori Penuh"<<endl;
        exit(0);
    }

    temp -> data = nilai;
    temp -> link = Nil;

    if (*rear == Nil)
    {
        *rear = temp;
        *front = *rear;
    }
}
```

```

else
{
    (*rear) -> link = temp;
    *rear = temp;
}
}

void del(struct node **front, struct node **rear, int *nilai)
{
    struct node *temp;

    if ((*front == *rear) && (*rear == Nil))
    {
        cout<<"Queue Kosong, Delete Tidak Dapat Dilakukan"<<endl;
        exit(0);
    }

    *nilai = (*front) -> data;
    temp = *front;
    *front = (*front) -> link;

    if (*rear == temp)
        *rear = (*rear) -> link;
    free(temp);
}

void main()
{
    struct node *front = Nil, *rear = Nil;
    int n,nilai;

    do
    {
        do
        {
            cout<<"Masukkan Nilai Elemen : ";
            cin>>nilai;
            cout<<endl;
            insert(&front,&rear,nilai);
            cout<<"Tekan 1 untuk Melanjutkan : ";
            cin>>n;
        } while (n == 1);

        cout<<endl;
        cout<<"Tekan 1 untuk Menghapus Elemen : ";
        cin>>n;
    }
}

```

```
while (n == 1)
{
    del(&front, &rear, &nilai);
    cout<<endl;
    cout<<"Nilai yang Dihapus : "<<nilai<<endl;
    cout<<"Tekan 1 untuk Menghapus Elemen : ";
    cin>>n;
}

cout<<endl;
cout<<"Tekan 1 untuk Melanjutkan : ";
cin>>n;
} while (n == 1);
}
```

Output :

```
Masukkan Nilai Elemen : 7

Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 12

Tekan 1 untuk Melanjutkan : 1
Masukkan Nilai Elemen : 35

Tekan 1 untuk Melanjutkan : 2

Tekan 1 untuk Menghapus Elemen : 1

Nilai yang Dihapus : 7
Tekan 1 untuk Menghapus Elemen : 1

Nilai yang Dihapus : 12
Tekan 1 untuk Menghapus Elemen : 2

Tekan 1 untuk Melanjutkan : 2
```

=====

Daftar Pustaka

Cheltenham Computer Training, C Programming, www.cctglobal.com, United Kingdom, 1997

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004

M. Fachrurrozi, Novi Yusliani, Modul Paktikum Struktur Data, Universitas Sriwijaya, 2006

Moh. Sjukani, 2009, Struktur Data dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.







MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

ADT (Abstack Data Type) Binary Tree

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

13

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Memberikan gambaran umum
mengenai konsep Abstack Data Type
pada Binary Tree dan mampu

Kompetensi

Mahasiswa dapat memahami definisi
Abstack Data Type pada Binary Tree
dan mampu mendeklarasikan Binary

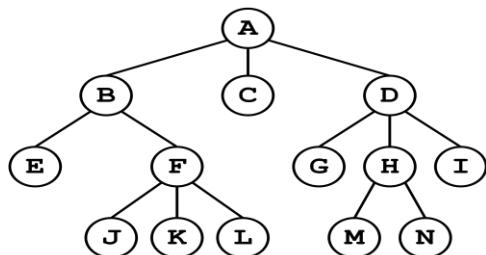
mengimplementasikan Binary Tree untuk pemecahan masalah pemrograman yang lebih kompleks.

Tree dalam bahasa C++, dan mampu membuat program dengan bantuan Binary Tree.

ADT Binary Tree

Definisi pohon adalah susunan dari satu atau lebih simpul (node) yang terdiri dari satu simpul khusus yang disebut akar (root) sedangkan sisanya membentuk subtree dan akar. Tree atau Pohon, termasuk struktur non linear, yang oleh beberapa buku literatur didefinisikan sebagai berikut : “Tree is a very important data object. Intuitively , a tree structure means that the data is organized so that items of information are related by branches”.

Ilustrasi sebuah Tree :



Untuk lebih jelasnya, di bawah akan diuraikan istilah-istilah umum dalam tree.

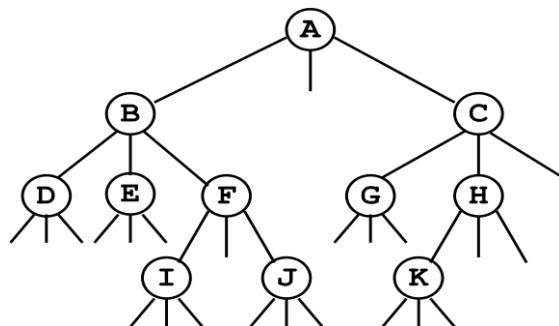
- Predecessor : Node yang berada di atas node tertentu
- Successor : Node yang berada dibawah node tertentu
- Ancestor : Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
- Descendant : Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
- Parent : Predecessor satu level di atas suatu node
- Child : Successor satu level di bawah suatu node
- Sibling : Node-node yang memiliki parent yang sama dengan suatu node

Subtree	: Bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
Size	: Banyaknya node dalam suatu tree
Height	: Banyaknya tingkatan / level dalam suatu tree
Root	: Satu-satunya node khusus dalam tree yang tak punya predecessor
Leaf	: Node-node dalam tree yang tak memiliki successor
Degree	: Banyaknya child yang dimiliki suatu node

Jenis-Jenis Tree

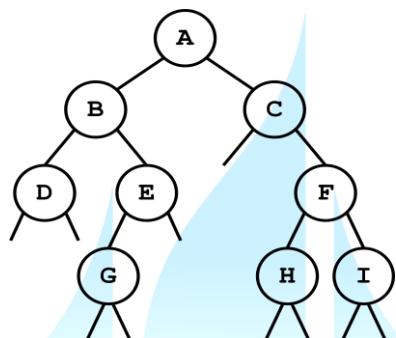
1. M-ary Tree

Adalah tree yang memiliki degree (derajat) mulai dari $M=1$ sampai $M=n$. Contoh sebuah pohon dengan degree $M=3$.



2. Binary Tree

Adalah Binary Tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Sesuai dengan definisi tersebut tiap node dalam binary tree hanya boleh memiliki paling banyak dua child.

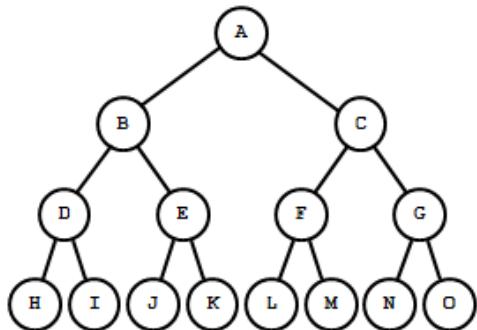


Implementasi Queue dengan Linear Array

Jenis-jenis Binary Tree

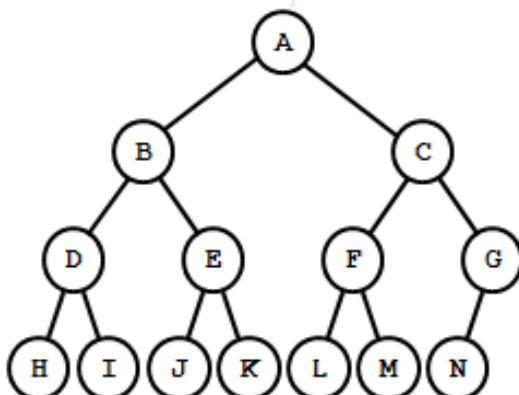
1. Complete Binary Tree

Adalah Binary Tree yang setiap node hanya boleh memiliki 2 child.



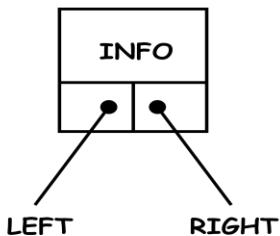
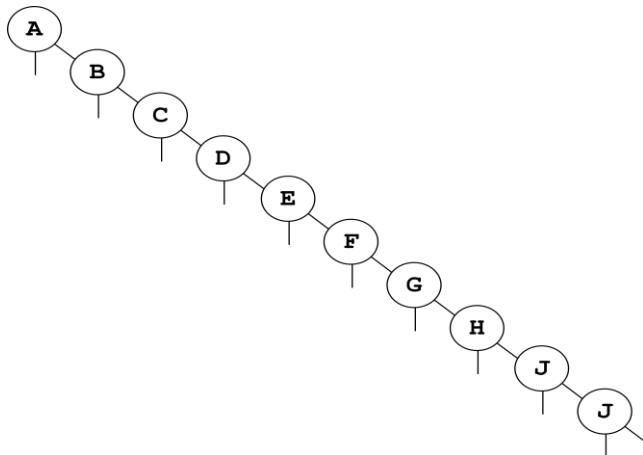
2. Almost Complete Binary Tree

Adalah Binary Tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Sesuai dengan definisi tersebut tiap node dalam binary tree hanya boleh memiliki paling banyak dua child.



3. Skewed Binary Tree

Skewed Binary Tree adalah Binary Tree yang semua nodenya (kecuali leaf) hanya memiliki satu child.



Strukturnya dapat dibuat dengan :

```
typedef struct Node {  
    struct Node *LEFT;  
    int INFO;  
    struct Node *RIGHT;  
};  
typedef struct Node Simpul;
```

atau

```
typedef struct Node {  
    int INFO;  
    struct Node *LEFT;  
    struct Node *RIGHT;  
};  
typedef struct Node Simpul;
```

```

typedef struct Node {
    struct Node *LEFT;
    int INFO;
    struct Node *RIGHT;
};

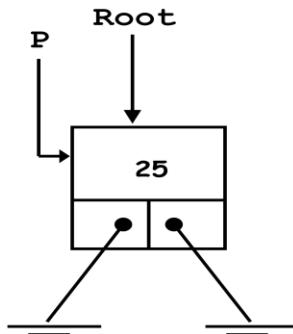
typedef struct Node Simpul;
Simpul *First, *Last, *P, *Q, *Root;

```

```

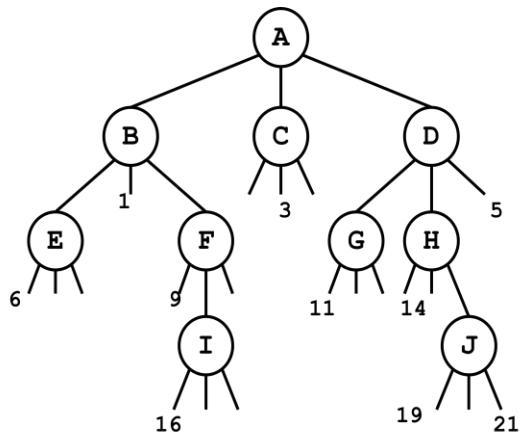
// membuat Simpul Akar
P = (Simpul * .....);
P->INFO = X; //misal X = 25
Root = P;
P->Left = NULL;
P->Right = NULL;

```



Link, Null-Link dan Bukan Null-Link

Contoh soal : Sebuah pohon M-ary dengan 10 buah simpul. Bila $M = 3$, maka Ditanya berapa jumlah Null-Link:

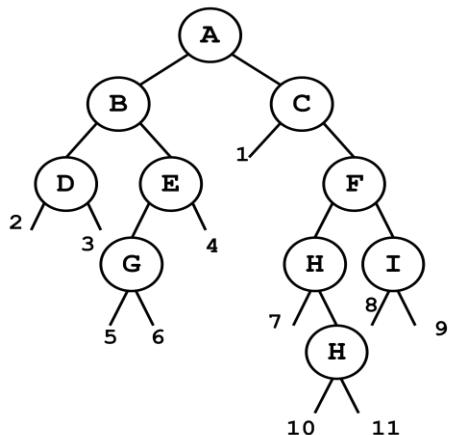


Jawab :

Pohon dengan $M = 3$ Jumlah simpul 10, jadi : $n = 10$

$$\begin{aligned}
 \text{Jumlah Null-Link} &= n * (M-1) + 1 \\
 &= 10 * (3-1) + 1 \\
 &= 10 * 2 + 1 \\
 &= 21
 \end{aligned}$$

Soal-2. Sebuah Pohon Biner dengan 10 buah simpul
Ditanya berapa jumlah Null-Link:



Jawab :

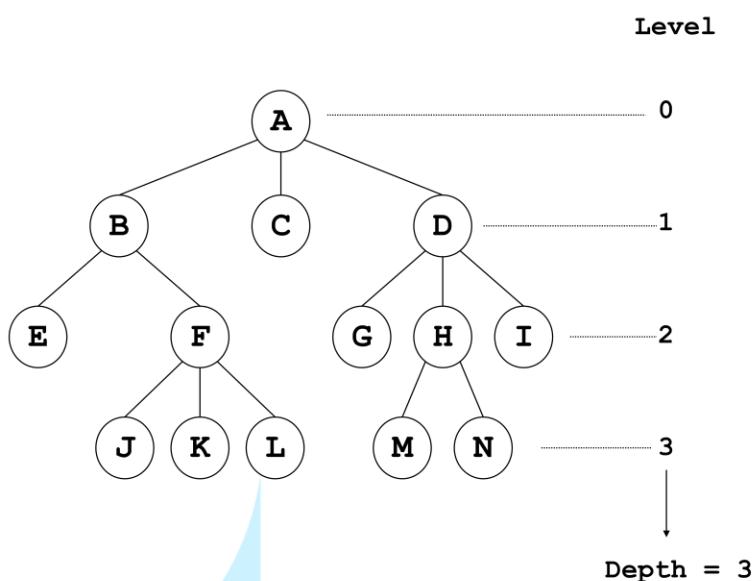
Pohon Biner, berarti $M = 2$
Jumlah simpul 10, jadi : $n = 10$

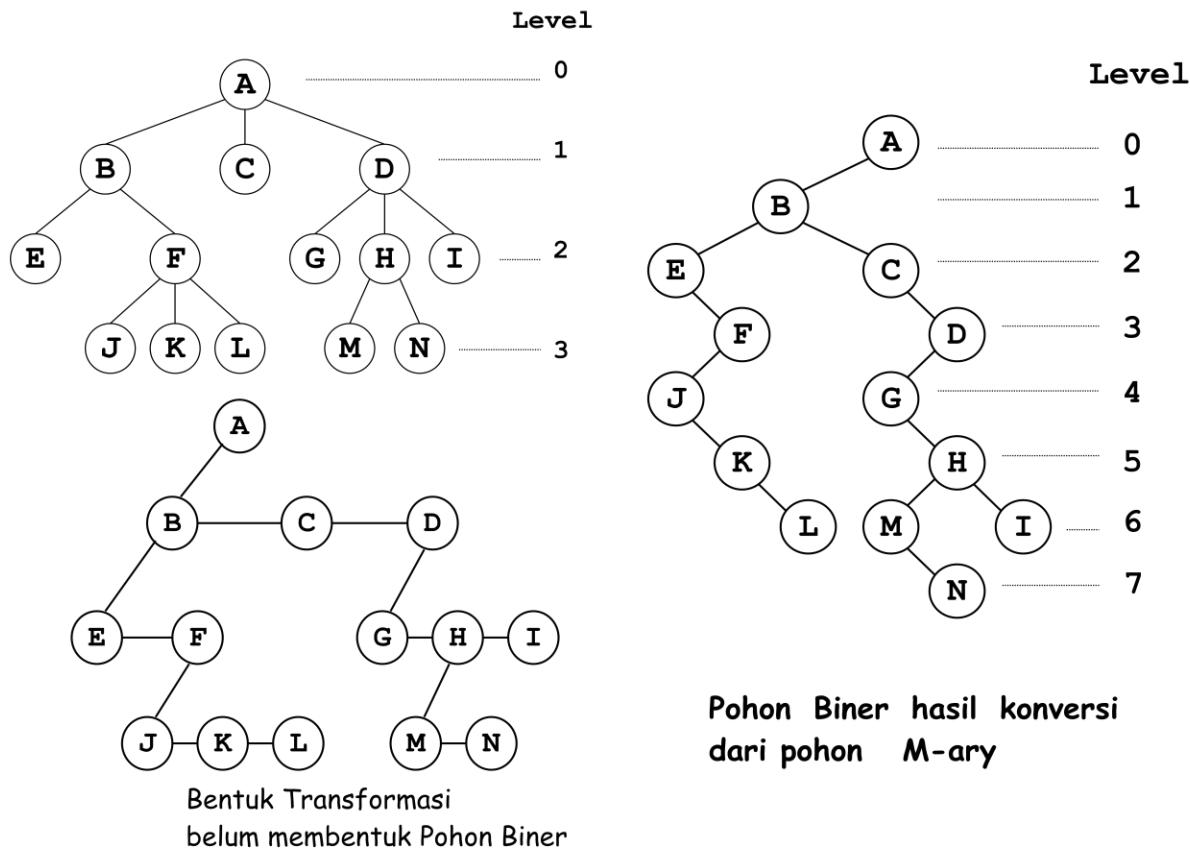
Jumlah Null-Link

$$\begin{aligned} &= n * (M - 1) + 1 \\ &= 10 * (2 - 1) + 1 \\ &= 10 * 1 + 1 \\ &= 11 \end{aligned}$$

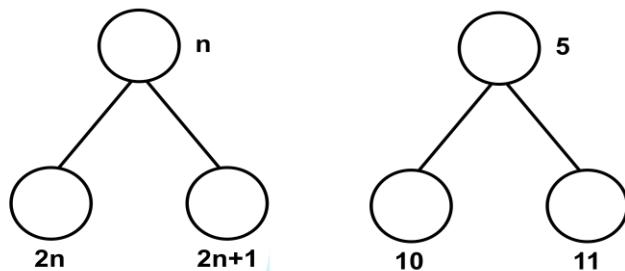
Konversi Pohon M-ary ke Pohon Biner

Soal : Konversikan pohon M-Ary berikut ini menjadi Pohon Biner

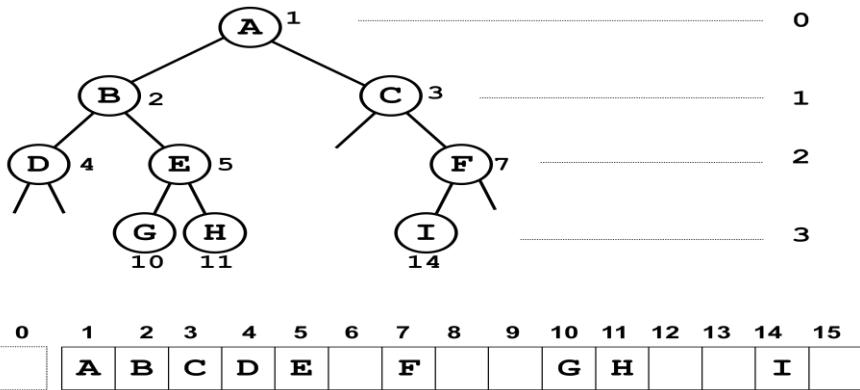




Penomoran Simpul Pohon Biner



Konversi Pohon Biner kedalam array 1 dimensi



Proses (Operasi) Pada Pohon Biner.

1. Inisialisasi
2. Pembuatan sebuah simpul.
3. Pembuatan simpul akar
4. Penambahan (insert) simpul kedalam sebuah pohon
5. Penghapusan (delete) simpul dari sebuah pohon
6. Pembacaan / Penelusuran pohon biner

1) Inisialisasi Struktur Simpul

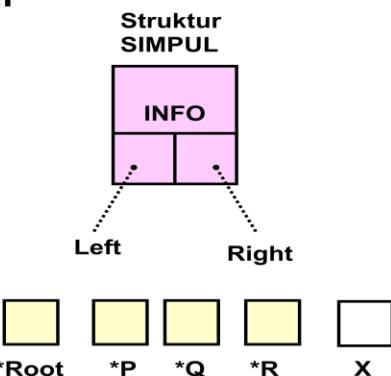
Mendeklarasikan struktur Simpul

```

struct Node
{
    struct Node *Left;
    char INFO;
    struct Node *Right;
};

typedef struct Node Simpul;
Simpul *Root, *P, *Q, *R;
char X;

```



Pointer Root digunakan khusus menunjuk simpul akar.

Pointer P digunakan khusus menunjuk simpul yang baru dibuat

Pointer Q, R digunakan sebagai pointer pembantu

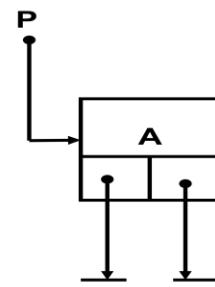
Pointer-pointer lain dapat ditambahkan bilamana diperlukan

Selain itu dideklarasikan juga sebuah variabel X
bertipe sama dengan tipe INFO yaitu tipe : char

2) Pembuatan Sebuah Simpul

Fungsi untuk Pembuatan Sebuah Simpul

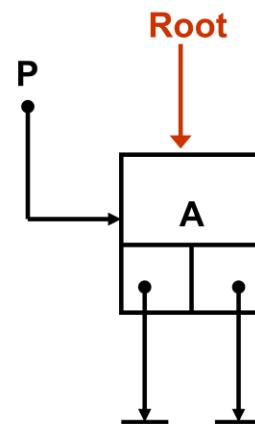
```
void BuatSimpul( char X)
{
    P = (Simpul*) malloc(sizeof(Simpul));
    if(P != NULL)
    {
        P->INFO = X;
        P->Left = NULL;
        P->Right = NULL;
    }
    else
    {
        printf("Memory Heap Full");
        exit(1);
    }
}
```



3) Pembuatan Simpul Akar

Fungsi untuk Menjadikan Sebuah Simpul Sebagai Simpul Akar

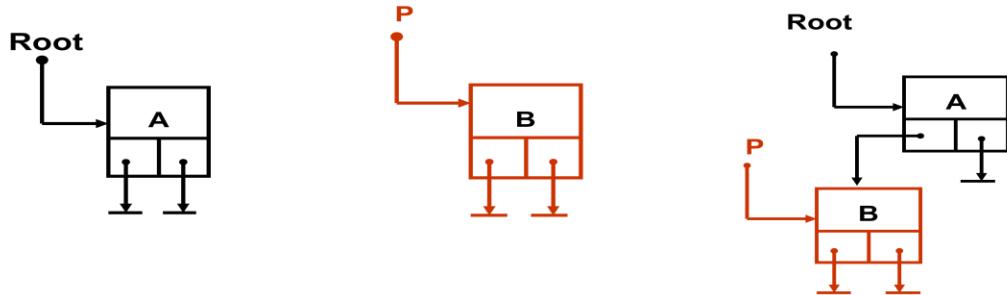
```
void BuatSimpulAkar( )
{
    if(Root == NULL)
    {
        if(P != NULL)
        {
            Root = P;
            Root->Left = NULL;
            Root->Right = NULL;
        }
        else
            printf("\n Simpul Belum Dibuat");
    }
    else
        printf("Pohon Sudah Ada");
}
```



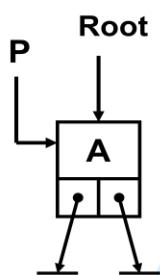
4) Penambahan (insert) simpul kedalam sebuah pohon

Menambahkan (Insert) Sebuah Simpul ke Pohon Yang Sudah Ada.

6.8.1 Insert urut nomor simpul atau insert level per level.



```
int main()
{ int i, j, Flag;
char X;
clrscr();
Inisialisasi();
X = getche();
BuatSimpul(X);
BuatSimpulAkar();
InsertSimpulUrutNomor();
BacaUrutNomor();
}
```



Daftar Pustaka

Cheltenham Computer Training, C Programming, www.cctglobal.com, United Kingdom, 1997

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Ellis Horowitz, Sartaj Sahni, Fundamental of Data Structure, Computer Science Press, 1983

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

M. Fachrurrozi, Novi Yusliani, Modul Paktikum Struktur Data, Universitas Sriwijaya, 20

Moh. Siukani. 2009. Struktur Data dengan C, C++ dan Java. Edisi 5. Mitra Wacana

Jakarta.





MODUL PERKULIAHAN

Algoritma & Struktur Data (Lab)

ADT (Abstack Data Type) Binary Tree (Lanjutan)

Fakultas
Ilmu Komputer

Program Studi
Sistem
Informasi

Tatap Muka

14

Kode MK
MK87042

Disusun Oleh
Tim Dosen

Abstract

Memberikan gambaran umum mengenai konsep Abstack Data Type pada Binary Tree dan mampu mengimplementasikan Binary Tree untuk pemecahan masalah pemrograman yang lebih kompleks.

Kompetensi

Mahasiswa dapat memahami definisi Abstack Data Type pada Binary Tree dan mampu mendeklarasikan Binary Tree dalam bahasa C++, dan mampu membuat program dengan bantuan Binary Tree.

Representasi Arithmetic Statement kedalam POHON BINER

1. Arithmetic Statement.

Paerhatikan Assignment Statement dalam bahasa C sebagai berikut :

$$X = A + B * C;$$

- A + B * C** adalah arithmetic statement atau arithmetic expression yang nilainya di-assign ke variabel **X**.
- A, B, dan C** disebut **operand**, yaitu yang dioperasikan, sedangkan **+** dan ***** disebut **operator**, yaitu yang mengoperasikan.

Dalam arithmetic statement, **operand** dilambangkan dengan Huruf atau Angka. Untuk memudahkan penulisan, huruf atau angka hanya digunakan **satu** karakter, dan untuk OPERATOR, digunakan lambang :

- (dan) kurung buka dan kurng tutup untuk pengelompokan nilai
^ untuk pangkat
* dan / untuk kali dan bagi
+ dan - untuk tambah dan kurang

Dalam Bahasa **C** , operator pangkat tidak dinyatakan dengan tanda ‘^’ seperti pada Bahasa BASIC, melainkan dinyatakan dengan fungsi pustaka **pow**. Contoh : 5 pangkat 2 dinyatakan dengan : **pow(5,2)**. Walaupun kita menggunakan Bahasa C , tapi untuk memudahkan pemahaman konversi, maka dalam buku ini tanda pangkat dinyatakan dengan karakter “^”, sehingga pow(5,2) dinyatakan dengan **5^2**

Dalam suatu arithmetic statement, masing-masing OPERATOR mempunyai **power** (kekuatan) dengan hirarkhi sebagai berikut:

^ lebih kuat dari ***** dan **/** .
***** dan **/** lebih kuat dari **+** dan **-** . Sedangkan ***** dan **/** sama kuatnya,
+ dan **-** sama kuatnya

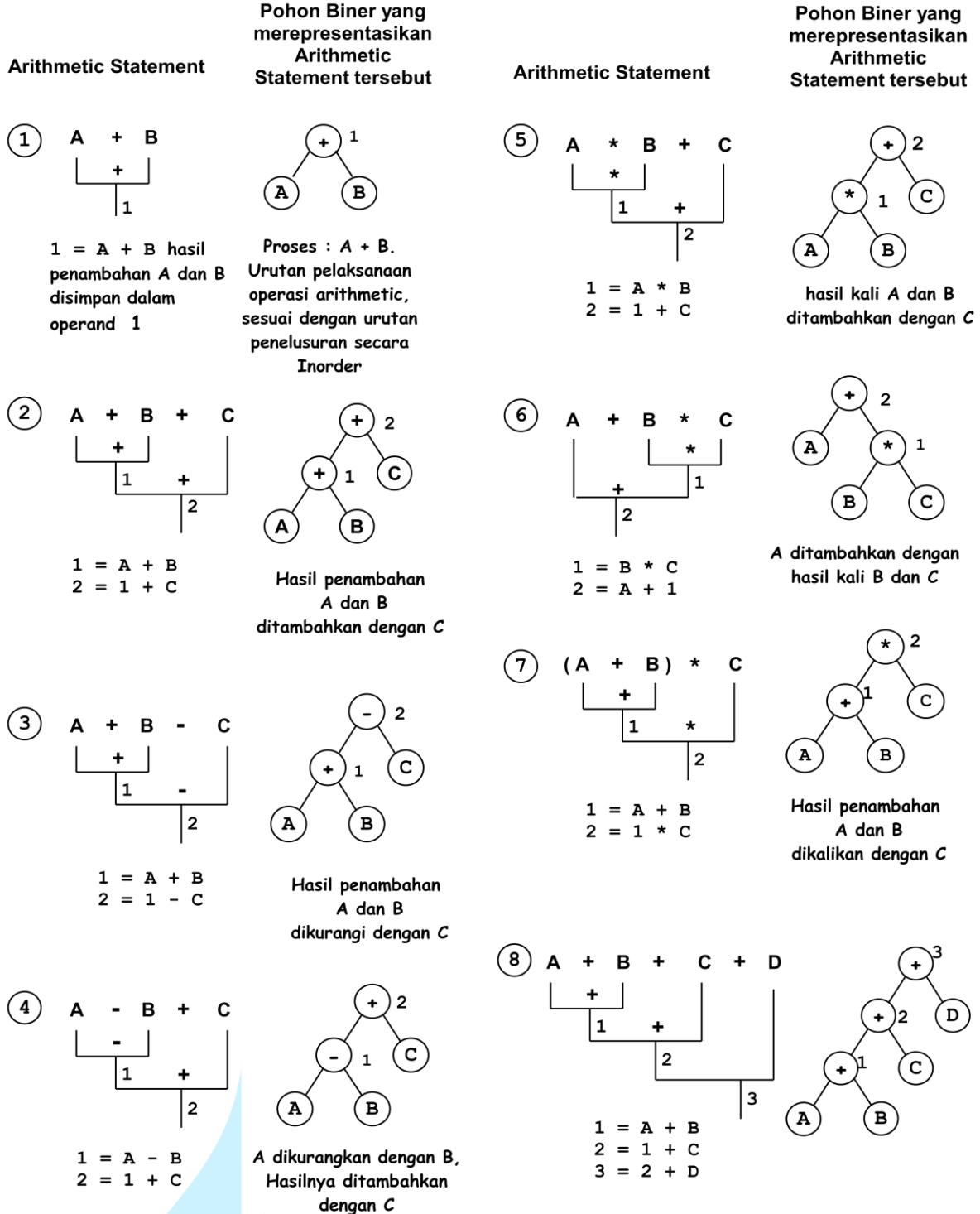
sedangkan kurung buka dan kurung tutup (), yang juga disebut operator, digunakan untuk menyatakan sebuah kelompok statement yang mempunyai sebuah nilai atau satu kesatuan nilai.

Perhatikan :

Dalam hirarkhi OPERATOR diatas, sebagai contoh disebutkan ‘kali’ (*) lebih kuat dari ‘tambah’ (+). Hanya disebutkan lebih kuat, **bukan** disebutkan bahwa ‘kali’ harus lebih dulu dikerjakan dari ‘tambah’.

2. **Contoh representasi arithmetic statement kedalam pohon biner.**





3. Penelusuran Pohon Biner Arithmetic

Penelusuran (traverse atau traversal) pohon biner, maksudnya membaca atau mengunjungi (visit) simpul-simpul pohon biner dengan urutan tertentu. Ada 3 (tiga) macam

penelusuran, yang bila ditambah dengan kebalikannya menjadi 6 (enam) macam penelusuran sebagai berikut :

1. Preorder (atau depth-first order)
2. Inorder (atau symetric order)
3. Postorder
4. Inverse Preorder
5. Inverse Inorder
6. Inverse Postorder

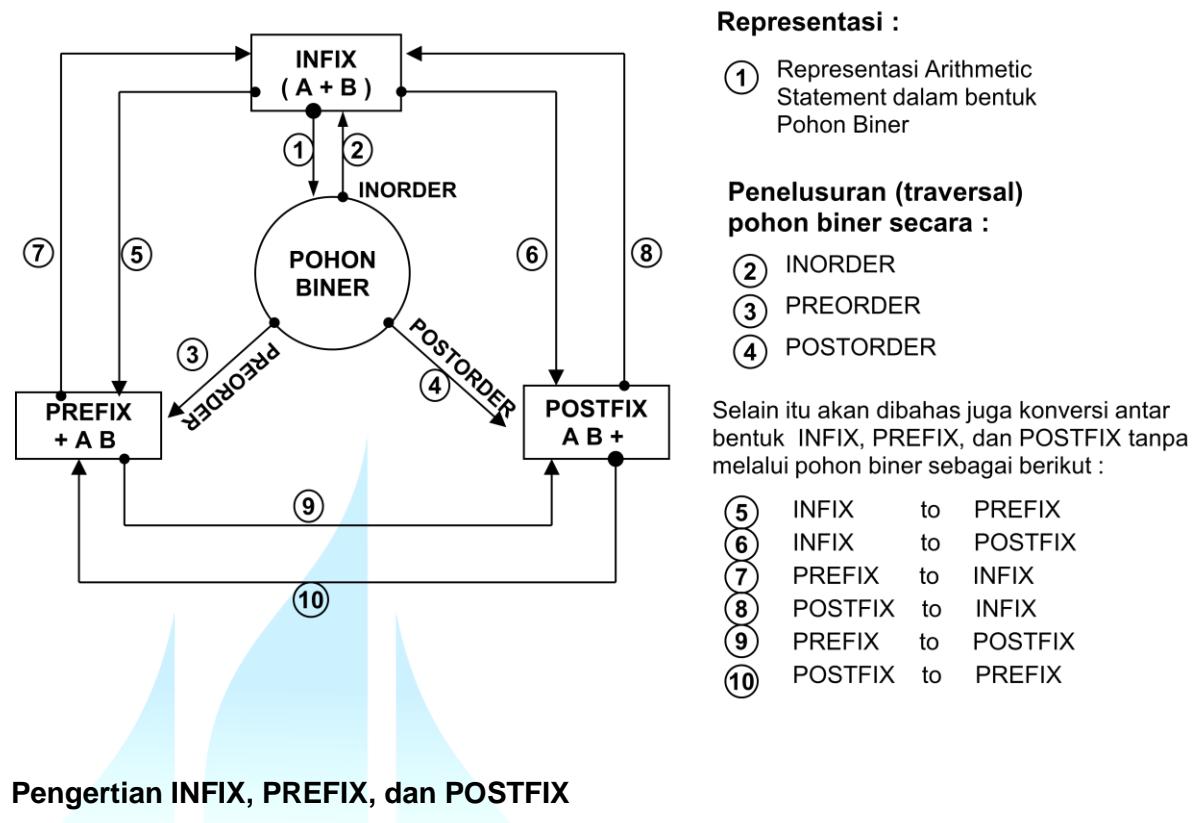
Pada modul ini akan dibahas 3 (tiga) penelusuran saja yaitu preorder, inorder dan postorder.

INORDER akan menghasilkan bentuk : **INFIX**

PREORDER akan menghasilkan bentuk : **PREFIX**

POSTORDER akan menghasilkan bentuk : **POSTFIX**

Yang secara bagan dapat digambarkan sebagai berikut :



Pengertian INFIX, PREFIX, dan POSTFIX

INFIX : Bentuk arithmetic statement yang kita kenal sehari-hari. IN artinya didalam, maksudnya operator ada didalam atau diantara dua buah operand

A + B → A dan B adalah operand
(yang dioperasikan)
+ adalah operator
(yang mengoperasikan)

A + B artinya A ditambahkan dengan B
Bila A=5 dan B=2
maka **A+B** hasilnya = 7

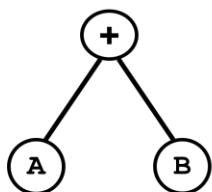
PREFIX : Arithmetic statement yang kita tulis dalam program, oleh compiler akan diubah kedalam bentuk PREFIX atau POSTFIX, tergantung sistem yang digunakan oleh komputer. PRE artinya sebelum maksudnya operatornya ditempatkan sebelum (atau ada didepan) kedua buah operand

contoh : **+ A B** **+AB** maksudnya A ditambahkan dengan B, hanya saja operatornya ada di depan
Bila A=5 dan B=2 maka **+AB** hasilnya = 7

POSTFIX : Biasanya compiler mengubah bentuk INFIX yang ditulis dalam program menjadi bentuk POSTFIX . POST artinya sesudah, maksudnya operatornya ada sesudah dua buah operand

contoh : **A B +**
AB+ maksudnya A ditambahkan dengan B, hanya saja operatornya ada di belakang.
Bila A=5 dan B=2 maka **AB+** hasilnya = 7

Kesimpulan : **INFIX** : A + B
PREFIX : + A B
POSTFIX : A B +



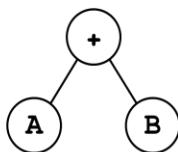
Pohon ini bila ditelusuri secara :
INORDER akan menghasilkan bentuk INFIX : A + B
PREORDER akan menghasilkan bentuk PREFIX : + A B
POSTORDER akan menghasilkan bentuk POSTFIX : A B +

Urutan penelusuran:

INORDER : SubPohonKiri, Akar, SubPohonKanan
 KIRI, AKAR, KANAN → A+B
PREORDER : AKAR, KIRI, KANAN → +AB
POSTORDER : KIRI, KANAN, AKAR → AB+

Contoh : Tuliskan hasil penelusuran secara : INORDER,
PREORDER, dan
POSTORDER
pohon-pohon biner berikut ini:

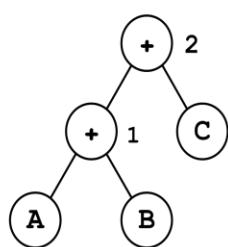
1



Hasil penelusuran

secara :
INORDER : $(A + B)$ atau $A + B$
PREORDER : $+ A B$
POSTORDER : $A B +$

2



Hasil penelusuran
secara :

INORDER : $A + B + C$
PREORDER : $+ + A B C$
POSTORDER : $A B + C +$

Keterangan : INORDER : Kiri, Akar, Kanan :

$$\begin{array}{c}
 (\text{_____}) \\
 \hline
 \text{kiri} & \text{akar} & \text{kanan}
 \end{array}
 \quad +
 \quad
 \begin{array}{c}
 (\text{C}) \\
 \hline
 \text{kanan}
 \end{array}$$

$$\begin{array}{c}
 (\text{A} \quad + \quad \text{B}) \\
 \hline
 \text{kiri} \quad \text{akar} \quad \text{kanan}
 \end{array}
 \quad +
 \quad
 \begin{array}{c}
 (\text{C}) \\
 \hline
 \text{kanan}
 \end{array}$$

Hasil akhir : $(A+B)+(C)$

Yang dapat disederhanakan menjadi : $A+B+C$

Catatan :

Hasil penelusuran secara INORDER akan menghasilkan bentuk INFIX, yaitu kembali ke bentuk arithmetic statement asalnya. Oleh karena itu, hasil **setiap langkah** proses penelusuran INORDER harus **selalu dalam tanda kurung**. Setelah hasil akhir, tanda kurung yang tidak diperlukan dapat dibuang, dengan pengertian bila tanda kurung tersebut dibuang, tidak akan merubah maksud arithmetic statement tersebut.

PREORDER : Akar, Kiri, Kanan :

$$\begin{array}{c}
 + \quad \text{_____} \quad \text{C} \\
 \hline
 \text{akar} \quad \text{kiri} \quad \text{kanan}
 \end{array}$$

$$\begin{array}{c}
 + \quad \text{A} \quad \text{B} \quad \text{C} \\
 \hline
 \text{akar} \quad \text{kiri} \quad \text{kanan}
 \end{array}$$

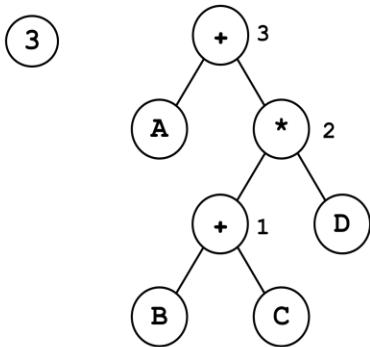
Hasil : $+ + A B C$

POSTORDER : Kiri, Kanan, Akar :

$$\begin{array}{c}
 \text{_____} \quad \text{C} \quad + \\
 \hline
 \text{kiri} \quad \text{kanan} \quad \text{akar}
 \end{array}$$

Hasil : $A B + C +$

$$\begin{array}{c}
 \text{A} \quad \text{B} \quad + \quad \text{C} \quad + \\
 \hline
 \text{kiri} \quad \text{kanan} \quad \text{akar} \quad \text{kanan} \quad \text{akar}
 \end{array}$$



Keterangan : INORDER : Kiri, Akar, Kanan :

Hasil : A + (B + C) * D

Hasil penelusuran

secara : INORDER : A + (B + C) * D
 PREORDER : + A * + B C D
 POSTORDER : A B C + D * +

$$\underline{A} \quad + \quad (\quad \text{_____} \quad)$$

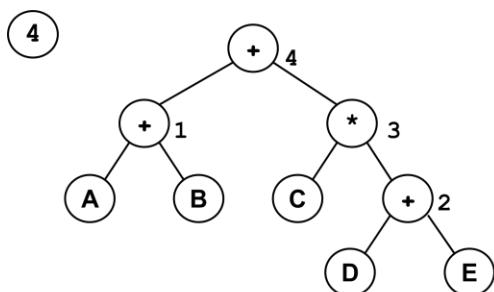
kiri akar kanan

A + (() * **D**)
 kiri akar kanan

$$A + \left(\frac{B + C}{D} \right) \text{ kiri}$$

Hasil akhir : A + ((B + C) * D)

Yang dapat disederhanakan menjadi : $A + (B + C) * D$



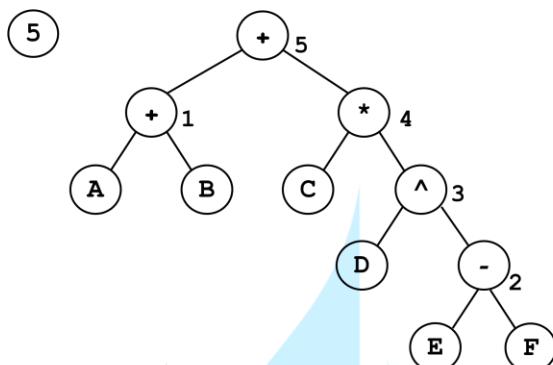
Hasil penelusuran

secara :

INORDER : $(A + B) + (C * (D + E))$
atau : $A + B + C * (D + E)$

PREORDER : + + A B * C + D E

POSTORDER: A B + C D E + * +



Hasil penelusuran

secara :

INORDER : $(A + B) + (C * (D \wedge (E - F)))$
atau : $A + B + C * D \wedge (E - F)$

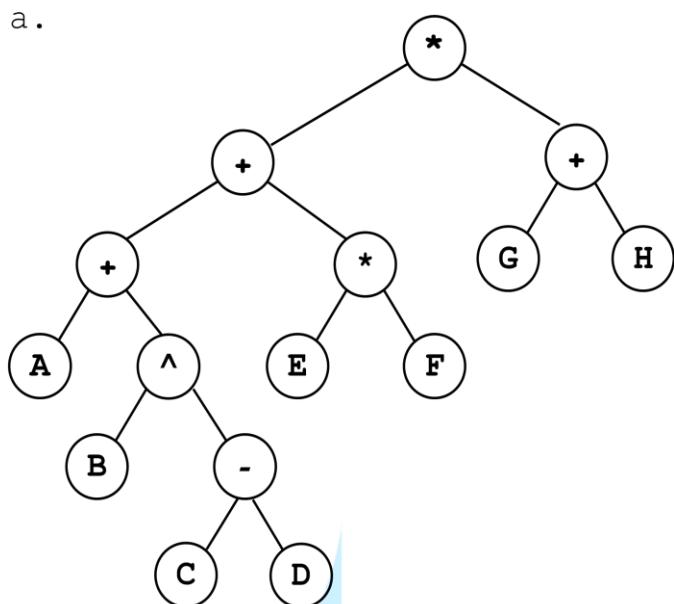
PREORDER : + + A B * C ^ D - E F

POSTORDER: A B + C D E F - ^ * +

Latihan Soal

1. Gambarkan pohon biner yang merupakan representasi arithmetic statement berikut ini.
 - a. $A * (B + C)$
 - b. $A + B + C + D * E$
 - c. $A - B * (C + D) * E$
 - d. $(A - B) * C + D * E$
 - e. $A * B + C + D * E$
2. Tuliskan arithmetic statement yang pohn binernya digambarkan sebagai berikut :

a.



=====

Daftar Pustaka

Cheltenham Computer Training, C Programming, www.cctglobal.com, United Kingdom, 1997

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Ellis Horowitz, Sartaj Sahni, Fundamental of Data Structure, Computer Science Press, 1983

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

M. Fachrurrozi, Novi Yusliani, Modul Paktikum Struktur Data, Universitas Sriwijaya, 2006

Moh. Sjukani, 2009, Struktur Data dengan C, C++ dan Java, Edisi 5, Mitra Wacana Media, Jakarta.

