



C & Data Structure

Training Module

October 2010

For the latest information, please see bluejack.binus.ac.id



Information in this document, including URL and other Internet Web site references, is subject to change without notice. This document supports a preliminary release of software that may be changed substantially prior to final commercial release, and is the proprietary information of Binus University.

This document is for informational purposes only. BINUS UNIVERSITY MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

The entire risk of the use or the results from the use of this document remains with the user. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Binus University.

Binus University may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Binus University, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2010 Binus University. All rights reserved.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

TABLE OF CONTENT

TABLE OF CONTENT	2
OVERVIEW	4
OBJECTIVE	5
SYSTEM REQUIREMENT	6
Algoritma & Pemrograman	7
Module 00 - Pendahuluan	7
Algoritma & Pemrograman	13
Module 01 - I/O Syntax and Variabel	13
Algoritma & Pemrograman	19
Module 02 - Aritmathic Operation	19
Algoritma & Pemrograman	22
Module 03 - Selection	22
Algoritma & Pemrograman	26
Module 04 - Looping	26
Algoritma & Pemrograman	29
Module 05 - Array	29
Algoritma & Pemrograman	32
Module 06 - Function	32
Algoritma & Pemrograman	35
Module 07 Built In Function	35
Algoritma & Pemrograman	41
Module 08 - Rekursif	41
Algoritma & Pemrograman	43
Module 09 - File Operation	43
Algoritma & Pemrograman	47
Module 10 - Sorting	47
Algoritma & Pemrograman	50
Module 11 - String	50
Data Structure	54
Module 12 - Struct	54
Data Structure	57
Module 13 - Linked List	57
a. Penempatan Node Baru	62
b. Menambah Node Pada Posisi Tertentu	62

Data Structure.....	66
Module 14 – Double Linked List.....	66
Data Structure.....	70
Module 15 - Queue	70
Data Structure.....	72
Module 16 – Priority Queue.....	72
Data Structure.....	74
Module 17 – Array Stack	74
Data Structure.....	79
Module 18 – Linked List Stack.....	79
Data Structure.....	83
Module 19 – Prefix, Infix, Postfix	83
Metode Infix Ke Postfix.....	84
Metode Infix Ke Prefix	85
Data Structure.....	93
Module 20 - Binary Tree	93
Data Structure.....	96
Module 21 – Binary Search Tree	96
Data Structure.....	101
Module 22 – AVL Tree	101
Data Structure.....	111
Module 23 – Heap Tree	111

OVERVIEW

Modul ini membahas tentang bahasa c, dimulai dari dasar sampai dengan materi struktur data. Materi yang diajarkan meliputi :

- "IDE dan I/O instruction,I/O Syntax and Variabel"
- Arithmetic Operation
- Selection Controlling Structure
- Loop Controlling Structure
- Array
- Function
- Built-in Function
- Recursive
- File Operation
- Sorting
- String
- Struct
- Linked List
- Double Linked List
- Queue
- Priority queue
- Array Stack
- Linked List Stack
- Prefix, Infix, Postfix
- Tree
- Binary Search Tree

Pada modul ini semua bahan diambil dari bahan materi yang akan diajarkan dalam Training tahap I, disertai dengan teori dan contoh program agar pembaca dapat mencoba sendiri code yang ada. Pada Modul ini pembahasan sudah diurutkan dari materi yang paling dasar sehingga pembaca akan semakin mudah mempelajari materi-materi yang ada.

OBJECTIVE

Dibuatnya modul ini ditujukan agar pembaca dapat belajar terlebih dahulu sebelum mengikuti training. Juga sebagai salah satu panduan dalam membuat soal-soal latihan selama training dan mempermudah trainee untuk mengerti dan memahami tiap-tiap materi yang akan diajarkan.

Dengan adanya modul ini diharapkan pula pembaca dapat mencoba code yang ada sehingga dapat lebih memahami maksud dan konsep pemrograman. Diharapkan Pembaca dapat mencoba menulis ulang program yang ada dan melihat hasilnya untuk dapat lebih mengerti.

SYSTEM REQUIREMENT

Software utama yang digunakan adalah :

➔ Microsoft Visual C++ 2008 express edition.

- **Sistem Requirement**

- 1.1 Supported Architectures**


- x86
 - x64 (WOW)

- 1.2. Supported Operating Systems**

- Windows XP Service Pack 2 or above
 - Windows Server 2003 Service Pack 1 or above
 - Windows Server 2003 R2 or above
 - Windows Vista
 - Windows Server 2008

- 1.3. Hardware Requirements**

- Minimum: 1.6 GHz CPU, 192 MB RAM, 1024x768 display, 5400 RPM hard disk
 - Recommended: 2.2 GHz or higher CPU, 384 MB or more RAM, 1280x1024 display, 7200 RPM or higher hard disk
 - On Windows Vista: 2.4 GHz CPU, 768 MB RAM
 - 1.3 GB of available disk space for the full installation

Algoritma & Pemrograman	
Module 00 - Pendahuluan	
Last Update 4-10-2010	Revision 00

1. Module Description

Pada bagian ini akan dijelaskan secara singkat pendahuluan tentang bahasa C dan sedikit konsep dalam pembuatan program berbahasa c.

2. Learning Outcomes

- Mengerti konsep dasar pemrograman bahasa c.
- Mengetahui dan mengerti istilah-istilah dasar pemrograman.

3. Material

a. Pembuka

Bahasa C / C++ merupakan salah satu bahasa pemrograman komputer dimana perintah-perintahnya terdiri dari beberapa modul (biasanya disebut fungsi). Bahasa pemrograman ini sudah menyediakan cukup banyak modul yang disimpan di *Standard Library* namun user pun dapat membuat fungsi atau modul sendiri. Oleh karena itu, dalam pemrograman bahasa C / C ++ ada dua hal yang penting, yaitu belajar bahasa C / C ++ itu sendiri dan belajar menggunakan modul dalam bahasa C / C ++.

b. Tahapan melakukan execute sebuah program

Pada bahasa C terdapat 6 tahap sebelum sebuah program dapat dijalankan yaitu :

→ Edit

Kita mengetik program menggunakan *text editor* kemudian di simpan dengan extention `'.c'` untuk bahasa C, dan extention `'.cpp'` untuk bahasa C++.

→ Preprocess

Di dalam sistem bahasa C / C++, *preprocessor* dijalankan secara otomatis sebelum proses *compile*. Tahap *preprocessor* menjalankan perintah yang dikenal dengan *preprocessor directives* yang menunjukkan adanya sesuatu yang harus dilakukan dahulu sebelum *compile*.

→ Compile

Menterjemahkan program C / C++ menjadi bahasa mesin yang biasa disebut *object code*.

→ Link

Program C / C++ biasanya mempunyai referensi ke fungsi yang di definisikan di tempat lain, misalnya di *Standard Library*. Oleh karena itu, *Object Code* yang dibuat biasanya mempunyai *holes* karena ada bagian-bagian yang hilang ini. Tahap *link* ini

adalah untuk melengkapi bagian-bagian yang hilang ini supaya menjadi image program yang lengkap.

→ Load

Tahap *load* menempatkan image program di dalam memory untuk kemudian dijalankan.

→ Execute

Menjalankan program yang telah *diload* ke dalam memory.

Tahapan *Preprocess*, *Compile*, *Link* berjalan bersamaan dengan menggunakan satu perintah. Biasa disebut *compile time*.

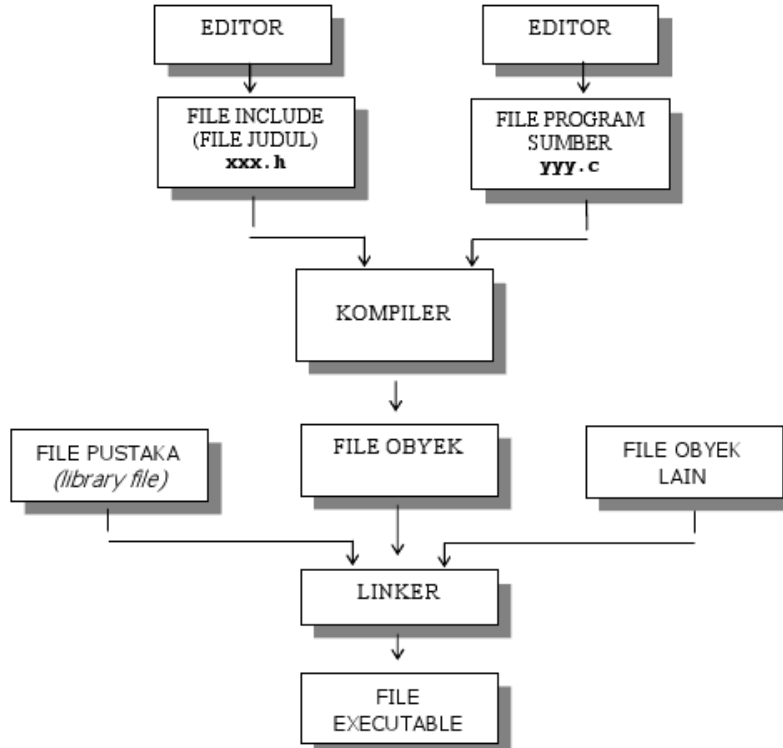
Tahapan *Load*, *Execute* berjalan bersamaan dengan menggunakan satu perintah. Biasa disebut *run time*.

Oleh karena itu, ada dua tipe *error* yang biasanya terjadi yaitu *compile-time error* dan *run-time error*.

c. Preprocess, Compile dan Link

- ✓ Dalam system UNIX dijalankan dengan perintah `cc {namafile}.c` dan menghasilkan image program dengan nama `a.out`.
- ✓ Dalam system LINUX dijalankan dengan perintah `gcc -o {namaimagefile} {namafile}.c` menghasilkan image program dengan nama `namaimagefile`.

Pemrosesan dari bentuk source code menjadi program yang executable dapat ditunjukkan oleh gambar sebagai berikut :



d. **Load, Execute**

Load dan Execute akan dilakukan dengan cara menjalankan file image.

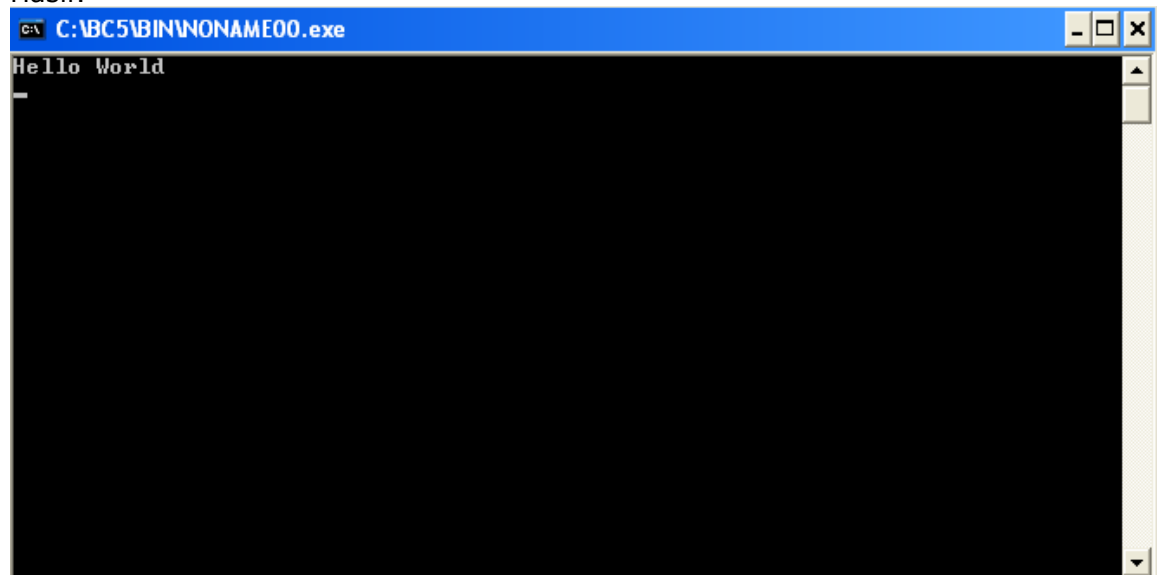
Contoh :

Kalian tidak perlu pusing bila tidak mengerti pembahasan masalah diatas, yang penting kalian bisa membuat program dengan menggunakan bahasa C, karena pemrograman bahasa C sekarang ini sudah menggunakan *Integrated Development Environment (IDE)* seperti Borland C++ dan Microsoft Visual C++. Sehingga kita tidak lagi bisa melihat tahap-tahap tersebut dengan jelas.

Contoh program:

```
=====
1  /* Bagian comment, digunakan untuk dokumentasi internal
2     bagian ini akan dilewati */
3  #include <stdio.h>
4
5  int main()
6  {
7      printf( "Hello World\n" );
8      /*getchar() digunakan agar console tidak langsung tertutup*/
9      getchar();
10     return 0;
11 }
=====
```

Hasil:



Penjelasan code:

Baris 1 dan 2 adalah *comment* yang akan diabaikan oleh *compiler*.

Baris 3 adalah bagian *Preprocessor* yang dijalankan sebelum *compile*.

Baris 5 adalah bagian yang harus ada di dalam setiap program C / C++. Program C / C++ selalu dijalankan dari fungsi/modul *main*.

Baris 6 dan 10 adalah bagian yang menunjukkan bagian dari *block statement*. Dalam hal ini bagian dari modul *main*.

Baris 7 adalah statement yang ada di dalam *Standard Library* yang menyebabkan computer untuk mencetak ke *stdout*.

Statement adalah bagian yang akan dijalankan oleh program. Satu statement diakhiri oleh titik-koma (;), disebut juga sebagai *statement terminator*.

Biasanya satu statement terdiri dari satu syntax seperti baris 7 diatas. Tapi bisa juga terdiri dari beberapa syntax.

Penting untuk diingat, statement-statement yang ada di dalam *Standard Library* bukan bagian dari bahasa pemrograman C / C++. Oleh karena itu *compiler* tidak dapat menemukan kesalahan dalam statement-statement tersebut. Yang tahu kalau ada kesalahan adalah *linker*.

Dalam IDE Borland C / C++ , kesalahan statement-statement yang ada di dalam *Standard Library* tidak dianggap sebagai *error* melainkan sebagai *warning* dengan pesan *warning: "called to function with no prototype"*. Ini disebabkan karena *linker* mencari di *Standard Library* dan tidak menemukan statement yang kita tulis di dalam program.

e. Algoritma dan Pseudocode

Algoritma adalah urutan langkah-langkah untuk menyelesaikan suatu masalah. Sesuai definisi, maka urutan langkah-langkah yang harus dilakukan adalah sangat penting. Tapi lebih dari itu, yang paling penting adalah masalahnya selesai. Sedangkan Pseudocode adalah kode-kode yang bisa kita mengerti yang nantinya akan kita olah dan kita ubah ke suatu bahasa pemrograman.

Contoh pseudocode untuk meminta inputan nama dan mencetaknya :

1. Baca nama
2. Cetak nama

Dalam bahasa C kita mengubahnya menjadi
char nama[100];
scanf("%s", nama); fflush(stdin);
printf("nama = %s", nama);

f. Identifier

Identifier adalah suatu pengenal atau pengidentifikasi yang dideklarasikan agar computer dapat mengenalnya. Identifier ini dapat berupa nama variable, konstanta, fungsi, class , struct ,dll. Panjang sebuah *identifier* tidak dibatasi tetapi hanya 32 karakter pertama yang akan dikenal.

Ada beberapa aturan dalam membuat sebuah identifier :

1. Tidak boleh diawali angka
2. Tidak boleh mengandung symbol.
3. Tidak boleh mengandung spasi
4. Tidak boleh ada dua identifier yang memiliki nama sama.

Identifier yang benar :

X
AbAb
_luas

Identifier yang salah :

3X (diawali angka)
Luas!! (terdapat simbol)
Lebar segitiga (terdapat spasi)

g. Operasi dan Operator

Suatu data/variabel yang sejenis dapat dilakukan suatu pengolahan data, seperti sebuah penambahan atau pengurangan, sebagai contoh : variabel dengan data integer dapat ditambahkan dengan variabel lain yang bertipe integer juga. Berdasarkan jenis operasi, maka operator dalam bahasa C dapat dikelompokkan menjadi :

1. Operator matematika.

Operator ini digunakan untuk pengolahan aritmatika seperti pengurangan, penjumlahan, perkalian, pembagian, sisa bagi dua bilangan bulat, penambahan nilai variabel dengan satu ataupun pengurangan nilai variabel dengan satu.

Tabel Operator Aritmatika.

Simbol	Fungsi	Contoh
+	Penjumlahan	$x + 3$
-	Pengurangan	saldoAwal - saldoAkhir
*	Perkalian	$3.14 * \text{jari_jari}$
/	Pembagian	berat / 100
%	Modulo	$k \% 7$
++	Increment	a++ atau ++a
--	Decrement	b++ atau ++b

2. Operator relasi.

Operator ini membandingkan dua buah nilai sejenis. Perbandingannya dapat berupa variabel ataupun constanta.

Simbol	Fungsi	Contoh
==	Sama Dengan	umur == 23
!=	Tidak Sama Dengan	input != 2
<	Lebih Kecil Dari	score < 70
>	Lebih Besar Dari	score > 70
<=	Lebih Kecil atau Sama Dengan	score <= 70
>=	Lebih Besar atau Sama Dengan	score >= 70

3. Operator logika.

Operator logika merupakan suatu operator yang berhubungan dengan logika matematika, seperti negasi(not), konjungsi(and), dan disjungsi(or).


Tabel dibawah ini merupakan penjelasan dari operator logika.

Simbol	Fungsi	Contoh
!	Not	!valid
&&	And	(score >= 75) && (score < 85)
	Or	(input < 0) (input > 100)

4. Operator bitwise.

Dalam pemrograman komputer, sebuah operasi bitwise beroperasi pada satu atau dua bit pola atau angka biner pada tingkat masing-masing bit. Pada kebanyakan komputer atau mesin hitung, operasi bitwise sedikit lebih cepat dibandingkan dengan penjumlahan atau pengurangan biasa.

Si mb ol	Fungsi	Con toh	Penjelasan
&	AND	X & Y	AND bitwise dari X dan Y
	OR	X Y	Or bitwise dari X dan Y
^	XOR	X ^ Y	Bernilai 1 jika bit-bit X dan Y berbeda
~	Comple ment 1	~X	Mengubah bit 1 menjadi 0 dan sebaliknya
>>	Shift right	X >> 3	X digeser kekanan sebanyak 3 posisi dari bit
<<	Shift left	Y << 1	y digeser kekiri sebanyak 1 posisi dari bit

Algoritma & Pemrograman	
Module 01 - I/O Syntax and Variabel	
Last Update 06-10-2010	Revision 00

1. Module Description

Pada modul ini trainee akan dijelaskan tentang input output, bagaimana mencetak karakter pada layar dan bagaimana meinput suatu masukan untuk diolah di komputer.

2. Learning Outcomes

- Mengetik program yang ada kesalahan sintaks dan I/O
- Meng-compile, run, dan debug program
- Mengenal tipe data dan penyimpanan data ke dalam variabel
- Membuat program dengan Sintaks I/O

3. Material

a. Struktur Dasar pembuatan program

Untuk dapat memahami bagaimana suatu program ditulis, maka struktur dari program harus dimengerti terlebih dahulu, atau sebagai pedoman penulis program (programmer) bagaimana seharusnya program tersebut ditulis.

```
main()
{
    statemen_1;
    statemen_2;
    .....
    statemen_n;
}
```

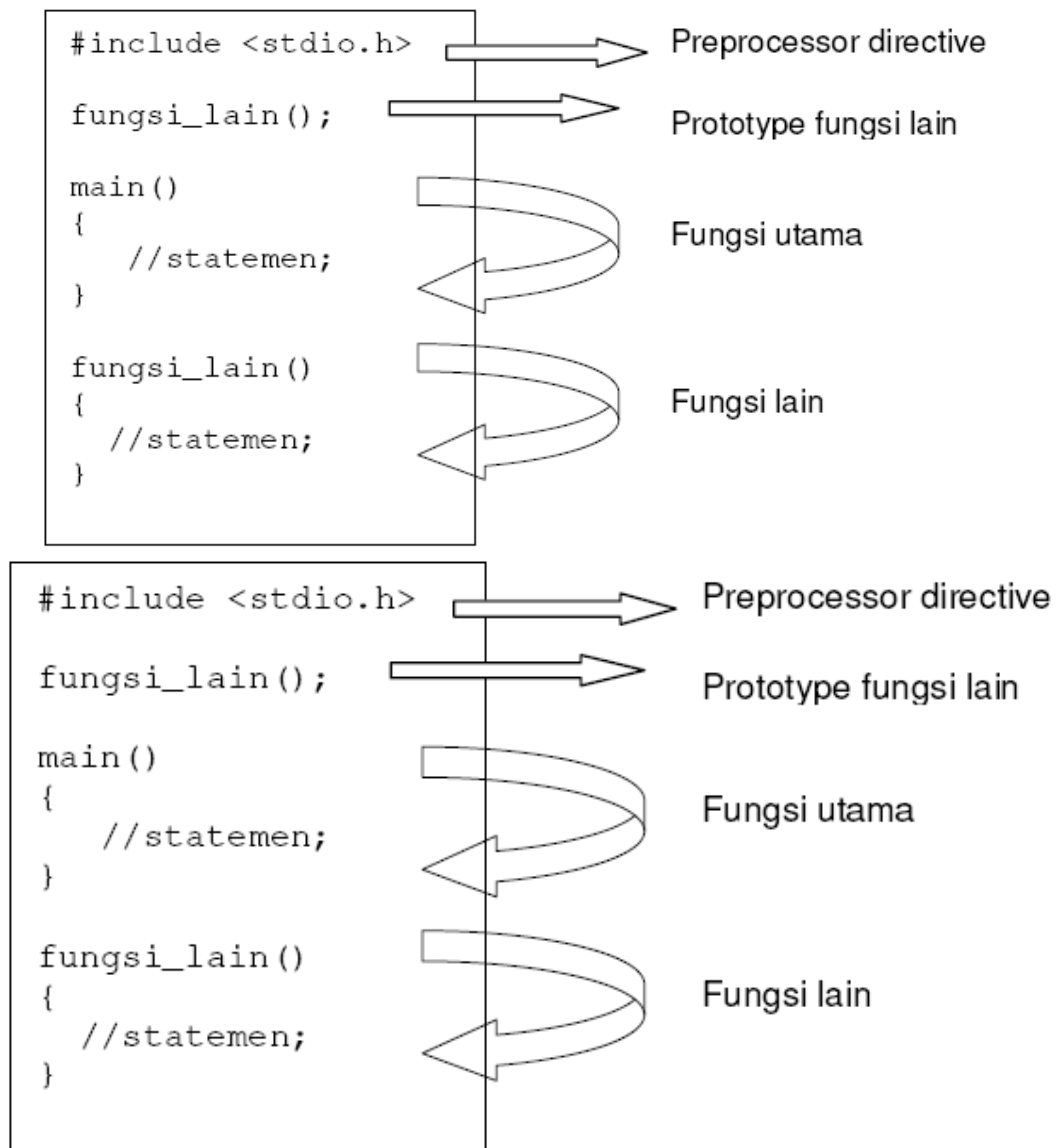
Fungsi Utama

```
fungsi_lain()
{
    statemen_statemen;
}
```

*Fungsi-fungsi lain
yang ditulis oleh pemrogram
komputer*

Struktur dari program C dapat dilihat sebagai kumpulan dari sebuah atau lebih fungsifungsi. Fungsi pertama yang harus ada di program C yang sudah ditentukan namanya, yaitu fungsi main(). Artinya program C minimal memiliki satu fungsi (fungsi main()). Fungsi-fungsi lain selain fungsi utama bisa dituliskan setelah atau sebelum fungsi utama dengan deskripsi prototype fungsi pada bagian awal

program. Bisa juga dituliskan pada file lain yang apabila kita ingin memakai atau memanggil fungsi dalam file lain tersebut, kita harus menuliskan header file-nya, dengan preprocessor directive `#include`. File ini disebut file pustaka (library file).



Keterangan :

- Dimulai dari tanda `{` hingga tanda `}` disebut tubuh fungsi / blok.
- Tanda `()` digunakan untuk mengapit argumen fungsi, yaitu nilai yang dilewatkan ke fungsi.
- Pada fungsi `main()` tidak ada argumen yang diberikan, maka tidak ada entri di dalam `()`.
- Kata `void` menyatakan bahwa fungsi ini tidak memiliki nilai balik.
- Tanda `{` menyatakan awal eksekusi program dan tanda `}` menyatakan akhir eksekusi program.
- Didalam tanda `{ }` bisa tergantung sejumlah unit yang disebut pernyataan (statemen).

Umumnya pernyataan berupa instruksi untuk :

- a. Memerintah komputer melakukan proses menampilkan string ke layar.
- b. Menghitung operasi matematika.
- c. Membaca data dari keyboard.
- d. dll.

Bahasa C dikatakan sebagai bahasa pemrograman terstruktur, karena strukturnya menggunakan fungsi-fungsi sebagai program bagian (subroutine). Fungsi-fungsi selain fungsi utama merupakan program-program bagian. Fungsi-fungsi ini dapat ditulis setelah fungsi utama atau diletakkan di file pustaka (library). Jika fungsi-fungsi diletakkan di file pustaka dan akan dipakai disuatu program, maka nama file judul (header file) harus dilibatkan didalam program yang menggunakannya dengan preprocessor directive #include.

b. Input / Output

Di dalam bahasa C / C++, untuk melakukan input/output, kita bisa menggunakan fungsi-fungsi yang sudah ada di standard library, yaitu di file header "stdio.h". Fungsi-fungsi tersebut antara lain :

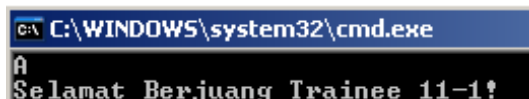
putchar : untuk menulis satu karakter
 printf : untuk menulis sesuai dengan format
 getchar : untuk membaca input satu karakter
 scanf : untuk membaca sesuai dengan format
 gets : untuk membaca string
 puts : untuk menulis string

Berikut contoh pemakaian fungsi-fungsi tersebut :

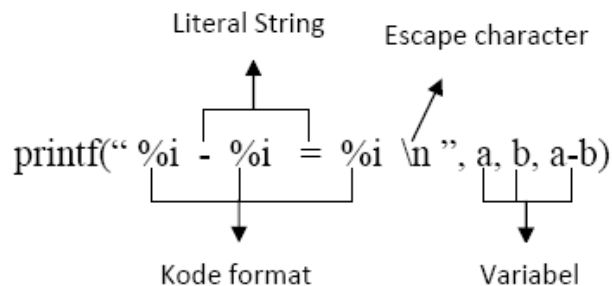
1. putchar dan printf

```
putchar('A');|
printf("\nSelamat Berjuang Trainee 11-1!\n");
```

Potongan program tersebut akan menghasilkan :



Contoh penggunaan printf yang lain :



a. Kode format.

Kode format menunjukkan format dari variabel yang akan ditampilkan nilainya.

b. Literal string.

Literal string adalah suatu konstanta string yang akan ditampilkan sesuai dengan apa yang dituliskan.

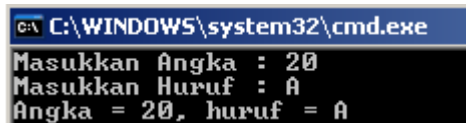
c. Escape character.

Karakter escape merupakan suatu konstanta karakter yang diawali dengan tanda back slash (\). Karakter escape "\n" ang digunakan fungsi printf() digunakan untuk menggeser posisi kursor turun satu baris kembali ke kolom pertama.

2. Getchar dan Scanf

```
int angka;
char huruf;
printf("Masukkan Angka : ");
scanf("%d", &angka);fflush(stdin);
printf("Masukkan Huruf : ");
huruf = getchar();
fflush(stdin);
printf("Angka = %i, huruf = %c\n" , angka, huruf);
```

Potongan program tersebut akan menghasilkan :



```
C:\WINDOWS\system32\cmd.exe
Masukkan Angka : 20
Masukkan Huruf : A
Angka = 20, huruf = A
```

d. Tipe Data dan Variabel

1. Tipe Data

Tipe data sederhana

Tipe data ini terdiri dari 5 macam, yaitu:

1. int untuk bilangan bulat
2. char untuk karakter
3. float untuk bilangan pecahan
4. double , seperti float, yaitu untuk bilangan pecahan juga, namun jangkauannya lebih besar
5. void (kosong).

Tipe data kompleks

Tipe data kompleks terdiri dari :

1. pointer
2. array
3. struct
4. union
5. class.

Pemodifikasi tipe data(*modifier*)

Modifier ini berguna untuk mengubah jangkauan dari tipe data sederhana

Modifier terdiri dari :

1. signed
2. unsigned
3. short
4. long.

Bentuk umum pendefinisian sebuah variabel :

```
tipe data namavariabel1, namavariabel2;
```

Contoh :

```
int x,y,z;          /*x,y,z bertipe bilangan bulat*/
float angka;        /*angka adalah bilangan pecahan*/
char huruf;         /*huruf adalah variabel bertipe char */
char kal[20];
/*kal adalah kumpulan data tipe char sebanyak 20 buah (array of char /
string)*/
```

Pada saat pendeklarasian anda dapat langsung memberi nilai awal :

Contoh :

```
int n = 10;         /*variabel n diberi nilai awal 10*/
char x = 'A';        /*x diberi nilai awal huruf 'A'*/
char y = 65;         /*y diberi nilai awal kode ascii 65 (kode ascii untuk
huruf 'A')*/
char z[21] = "Bluejackets All Star";
//z diberi nilai awal berupa kalimat "Bluejackets All Star".
```

Catatan : Operator untuk assignment dalam C/ C++ adalah "=" lain dengan Pascal yang menggunakan ":=".

Jika suatu variabel didefinisikan di dalam fungsi (misalnya fungsi *main ()*), maka ia bersifat lokal terhadap fungsi *main ()*, sedangkan jika ia dideklarasikan di luar fungsi maka ia bersifat global terhadap fungsi-fungsi lain di bawahnya.

Memasukkan Nilai ke dalam Variabel menggunakan fungsi Input

Jika anda membaca diktat ini dari awal, anda tentu menyadari adanya tulisan "%d", "%i", dll. Tulisan ini sebenarnya adalah suatu format input/output yang sudah dibuat sebagai sebuah standart untuk setiap variabel, berikut daftar format-format tersebut :

1	Int	%d, %i	Untuk bilangan bulat
2	Float	%f	Untuk bilangan pecahan
3	Double	%lf	Untuk bilangan pecahan
4	Char	%c	Untuk sebuah karakter
5	String	%s	Untuk kumpulan karakter

Format tersebut dapat digunakan untuk format output maupun input.

Contoh :

Untuk menginput sebuah bilangan bulat(int) dan character :

```

int angka;
char huruf;
printf("Masukkan Angka : ");
scanf("%d", &angka);fflush(stdin);
printf("Masukkan Huruf : ");|
scanf("%c", &huruf)
fflush(stdin);
printf("Angka = %i, huruf = %c\n" , angka, huruf);
Keterangan

```

- untuk menginput sebuah angka kita menggunakan fungsi scanf dan menggunakan 2 parameter, parameter pertama berisi format tipe data dan parameter kedua berisi nama variabel diawali tanda "&". Awal tanda & berarti menunjuk ke alamat dari nama variabel yang akan dibahas lebih lengkap pada sesi-sesi berikutnya. Tipe data int, float, double, char jika menggunakan fungsi scanf harus diawali dengan lambang "&".
- Untuk mencetak angka kita bisa menggunakan "%i". Jika anda bertanya kapan kita menggunakan %i dan %d, keduanya adalah sama. Keduanya merupakan format untuk bilangan bulat berbasis desimal.
- Cobalah ganti %i dengan %x atau %o dan lihat hasilnya. Cari tahu lebih lanjut tentang %x dan %o melalui buku atau google.


2. Konsep Memory

Setiap Variabel yang dideklarasikan tentu saja memerlukan memori untuk menampung nilai yang di *assign*. Dalam bahasa C / C++, program akan memesan suatu tempat di alamat tertentu sebesar yang diperlukan oleh variable di memory computer. Jadi, Setiap variable di bahasa C / C++ mempunyai nama, tipe dan nilai, dan menunjuk ke suatu tempat di memory computer.

Misalnya deklarasi variable

```
int x;
```

dijalankan oleh program maka program memesan tempat di memory untuk menampung nilai yang mempunyai tipe integer dan kita mengaksesnya dengan menggunakan nama 'x'.

Algoritma & Pemrograman	
Module 02 - Arithmetic Operation	
Last Update 06-10-2010	Revision 00

1. Module Description

Modul ini menjelaskan tentang operasi – operasi aritmatika.

2. Learning Outcomes

- Membuat program operasi aritmatika

3. Material

a. Operasi Aritmatika

Operator ini digunakan untuk pengolahan aritmatika seperti pengurangan, penjumlahan, perkalian, pembagian, sisa bagi dua bilangan bulat, penambahan nilai variabel dengan satu ataupun pengurangan nilai variabel dengan satu.

Tabel Operator Aritmatika.

Simbol	Fungsi	Contoh
+	Penjumlahan	$x + 3$
-	Pengurangan	saldoAwal - saldoAkhir
*	Perkalian	$3.14 * \text{jari_jari}$
/	Pembagian	berat / 100
%	Modulo	$k \% 7$
++	Increment	a++ atau ++a
--	Decrement	b++ atau ++b

Penggunaan aritmatika dalam pemrograman khususnya dengan bahasa c sangat mudah. Anda hanya perlu menyiapkan variabel dan bisa langsung melakukan operasi aritmatika.

Contoh penggunaan Aritmatika :

```

int angka1 = 20, angka2 = 3;
//contoh program aritmatika
//penambahan
printf("%d + %d = %d\n", angka1, angka2, angka1 + angka2);
/*untuk operasi aritmatika anda bisa menyimpan
nilainya terlebih dahulu*/
int hasil = angka1- angka2;
printf("%d - %d = %d\n", angka1, angka2, hasil);
hasil = angka1 * angka2;
printf("%d * %d = %d\n", angka1, angka2, hasil);
hasil = angka1 % angka2;
printf("%d %% %d = %d\n", angka1, angka2, hasil);
/*untuk pembagian 20/3 akan menghasilkan angka koma
yang tidak bisa ditampung sempurna oleh tipe data int
agar hasil tersimpan sempurna kita harus merubah tipe
data variabel tersebut dengan cara type cast*/
float pembagian = (float)angka1/angka2;
/*cukup salah satu dari pembagi atau penyebut yang
dirubah tipe datanya*/
printf("%d / %d = %f \n", angka1, angka2, pembagian);

```

Program tersebut akan menghasilkan :

```

20 + 3 = 23
20 - 3 = 17
20 * 3 = 60
20 % 3 = 2
20 / 3 = 6.666667

```

Untuk increment dan decrement berfungsi untuk menambahkan nilai satu(1) untuk variable yang di increment atau decrement

b. Increment and decrement (Continued..)

Increment dan decrement dibagi menjadi 2 macam, post dan pre. Post berarti penambahan/pengurangan dilakukan diakhir, Pre berarti penambahan/pengurangan dilakukan di awal.

Contoh :

- ➔ Post
a++, b--
- ➔ Pre
--a, --b

Saat program yang mengandung drecement atau increment dijalankan perbedaan antara post dan pre adalah sebagai berikut.

➔ Pada Post, nilai dari variabel akan dibaca terlebih dahulu sebelum dia ditambahkan dengan 1.

○ Contoh :

```
int angka = 5;
printf("%d", angka++);
```

Berapakah hasil yang tercetak?? Kebanyakan orang yang baru belajar akan menjawab 6. Namun jawaban yang benar adalah 5. Tidak percaya? Try it.

Akan tetapi jika anda melakukan sedikit penambahan :

```
int angka = 5;
printf("%d\n", angka++);
printf("%d", angka);
```

hasil dari program tersebut adalah

5

6

Jadi kesimpulannya Post akan dijalankan setelah data dari variabel dibaca.

➔ Untuk Pre, adalah kebalikan dari Post, nilai akan ditambah terlebih dahulu sebelum data dari variabel dibaca

○ Contoh

```
int angka = 5;
printf("%d", ++angka);
```


Hasil potongan program tersebut adalah

6

Untuk menguji apakah anda benar-benar mengerti konsep pre dan post coba hitunglah hasil perhitungan dari soal berikut

```
angka = 5
++angka + angka ++ + angka++ + angka ++ + ++angka
```

Jika hasil perhitungan anda adalah 35 maka anda sudah cukup mengerti konsep pre dan post

Algoritma & Pemrograman	
Module 03 - Selection	
Last Update 07 October 2010	Revision 01

1. Module Description

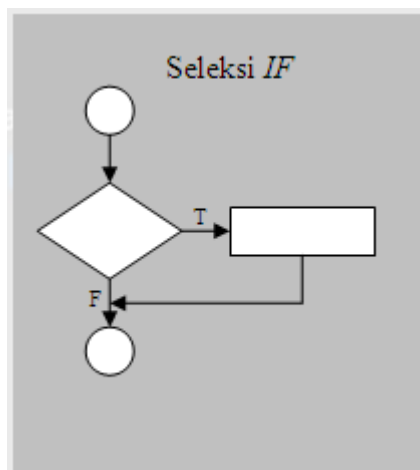
Pada bagian ini, akan dijelaskan cara menggunakan seleksi dalam bahasa C. Berikut dengan alur dan cara kerjanya.

2. Learning Outcomes

- Setelah mempelajari materi ini, diharapkan dapat mengerti dan menggunakan seleksi dalam bahasa C.
- Membuat program yang menggunakan struktur kendali perulangan.

3. Material

a. Selection using "if"



Seleksi *IF* adalah struktur kontrol yang menjalankan statement setelah memeriksa suatu kondisi terlebih dahulu. Jika kondisi tersebut bernilai benar, maka statement dijalankan, jika kondisi tersebut tidak benar, maka statement tidak dijalankan.

Jika suatu kondisi bernilai tidak benar, maka bahasa C menganggap kondisi tersebut bernilai nol. Jika suatu kondisi tersebut bernilai benar, maka bahasa C menganggap kondisi tersebut bernilai tidak 0 (nol).

Bentuk umumnya adalah

```
if( <<ekspresi>> ) <<statement>>;
```

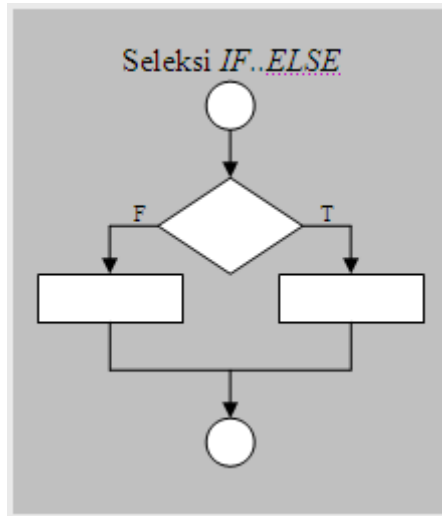
struktur kontrol *IF* ini memeriksa apakah ekspresi yang diberikan bernilai 0 (nol) atau tidak. Jika ekspresi tidak bernilai 0 (nol), maka statement dijalankan.

Contoh:

```
if( nilai < 55 ) printf( "Anda tidak lulus\n" );
```

```
if( a - 5 ) printf( "Ternyata a dikurang 5 tidak sama dengan 0\n" );
```

b. Selection using "if..else"



Seleksi *IF...ELSE* adalah struktur kontrol yang menjalankan statement setelah memeriksa suatu kondisi. Jika kondisi tersebut benar, maka statement dijalankan, jika kondisi tersebut tidak benar, maka statement lain dijalankan.

Bentuk umumnya adalah

```
if( <<ekspresi>> ) <<statement1>>;  
else <<statement2>>;
```

struktur kontrol *IF...ELSE* ini memeriksa apakah ekspresi yang diberikan bernilai 0 (nol) atau tidak. Jika ekspresi tidak bernilai 0 (nol), maka *statement1* dijalankan. Jika ekspresi bernilai 0 (nol), maka *statement2* dijalankan.

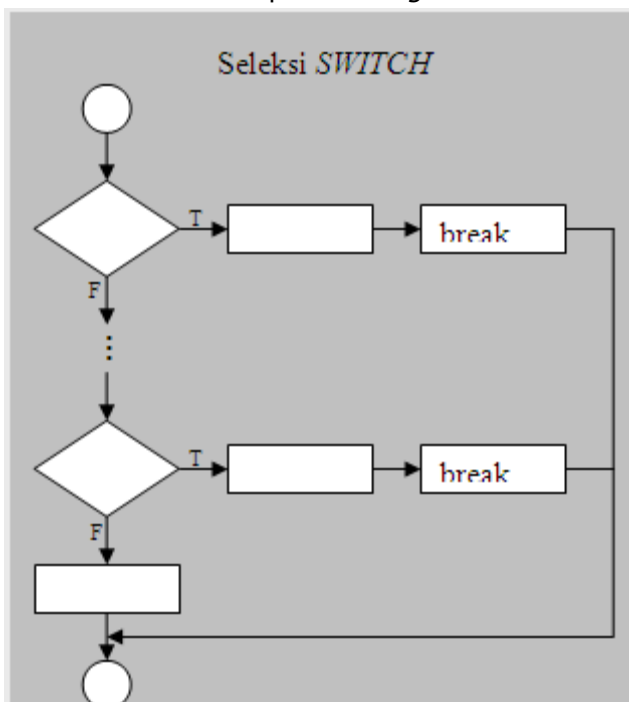
Contoh:

```
if( nilai < 55 ) printf( "Anda tidak lulus\n" );  
else printf( "Anda lulus\n" );
```

```
if( a - 5 ) printf( "ternyata a dikurang 5 tidak sama dengan 0\n" );  
else printf( "ternyata a dikurang 5 sama dengan 0\n" );
```

c. Selection using "switch..case"

Seleksi *IF* merupakan *single-selection* dan seleksi *IF..ELSE* merupakan *double-selection*. Sedangkan seleksi *SWITCH* merupakan *multiple-selection*. Seleksi *SWITCH* terdiri dari beberapa label *CASE* dan sebuah label *DEFAULT* yang optional.



Bentuk umumnya adalah:

```
switch( <<ekspresi>> )  
{  
    case <<nilai ekspresi>> :  
        <<statement>>;  
    case <<nilai ekspresi>> :  
        <<statement>>;  
    ...  
    default : <<statement>>;  
}
```

Struktur kontrol *SWITCH..CASE* memeriksa ekspresi dan kemudian langsung menjalankan statement mulai dari label *CASE* dengan nilai

ekspresi yang sesuai. Jika label *CASE* tidak mempunyai nilai ekspresi yang sesuai, maka statement akan dijalankan mulai dari label *DEFAULT*.

Hal yang penting dalam struktur kontrol *SWITCH..CASE* adalah statement dijalankan mulai dari label *CASE* dengan nilai ekspresi yang sesuai. Jika setelah statement tersebut masih ada statement lagi, maka statement-statement berikutnya akan terus dijalankan sampai akhir dari sintaks *SWITCH..CASE*.

Contoh:

```
switch ( a )
{
    case 1: printf("nilai a adalah 1\n") ;
    case 2: printf("nilai a adalah 2\n") ;
    case 3: printf("nilai a adalah 3\n") ;
    default: printf("nilai a bukan 1 ataupun 2 ataupun 3\n") ;
}
```

Pada contoh di atas, jika nilai variabel *a* adalah 1 maka yang akan statement akan mulai dijalankan dari case pertama, yaitu mencetak **nilai a adalah 1**. Kemudian statement-statement berikutnya juga dijalankan sampai akhir dari struktur kontrol *SWITCH..CASE*. Dari contoh di atas maka yang akan tercetak di layar adalah:

```
nilai a adalah 1
nilai a adalah 2
nilai a adalah 3
nilai a bukan 1 ataupun 2 ataupun 3
```

Jika nilai variabel *a* adalah 3 maka yang yang tercetak di layar adalah:

```
nilai a adalah 3
nilai a bukan 1 ataupun 2 ataupun 3
```

Posisi label *CASE* tidak membuat program tersebut jadi error. Posisi label *CASE* hanya menentukan awal pelaksanaan statement.

Tentunya hasil ini tidak sesuai dengan apa yang kita mau, oleh karena itu hal penting berikutnya untuk kita ingat adalah *break* yang digunakan untuk menuju ke kurung kurawal tutup.

Contoh:


```
switch ( a )
{
    case 1: printf("nilai a adalah 1\n") ; break;
    case 2: printf("nilai a adalah 2\n") ; break;
    case 3: printf("nilai a adalah 3\n") ; break;
    default: printf("nilai a bukan 1 ataupun 2 ataupun 3\n") ;
}
```

Pada contoh di atas, jika nilai variabel *a* adalah 1, maka yang tercetak di layar adalah:

```
nilai a adalah 1
```

Karena ketika struktur kontrol *SWITCH..CASE* mulai dijalankan dan memeriksa nilai *a*, statement dijalankan mulai dari label *CASE* dengan nilai yang sesuai. Setelah

statement dijalankan, kemudian statement berikutnya dijalankan. Tapi statement berikutnya adalah *break*. Dan *break* berarti berarti menuju ke kurung kurawal tutup yang berarti menuju ke akhir dari sintaks.

Algoritma & Pemrograman	
Module 04 - Looping	
Last Update 05 October 2010	Revision 00

1. Module Description

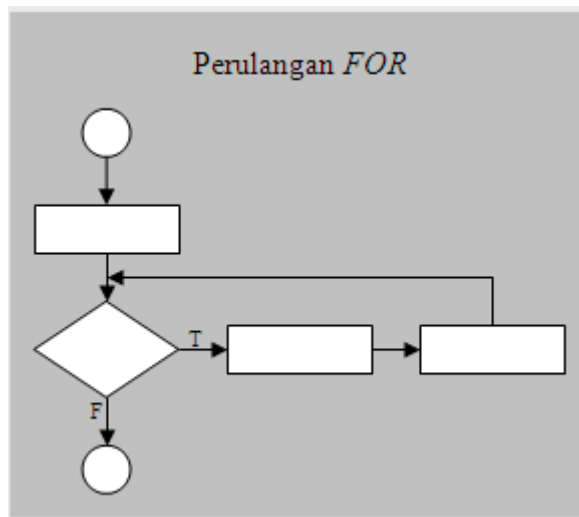
Pada bagian ini, akan dijelaskan cara menggunakan perulangan dalam bahasa C.

2. Learning Outcomes

- Setelah mempelajari materi ini, diharapkan dapat mengerti dan menggunakan perulangan dalam bahasa C.
- Membuat program yang menggunakan struktur kendali perulangan.

3. Material

a. Looping using “for”



Perulangan *FOR* menggunakan semua detail yang dibutuhkan dalam sebuah *counter-controlled repetition*.

Bentuk umumnya adalah

```
for(          <<ekspresi1>>;
    <<ekspresi2>>; <<ekspresi3>> )
<<statement>>;
```

Struktur kontrol *FOR* ini melakukan perulangan sampai ekspresi2 bernilai 0 (nol).

Struktur kontrol *FOR* menjalankan langkah-langkah :

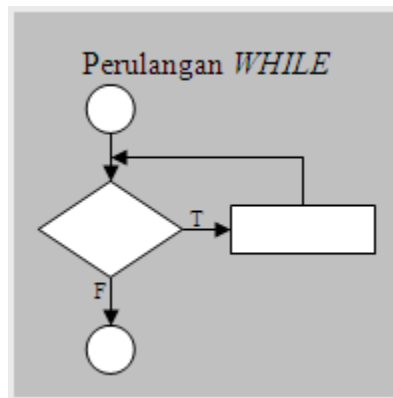
1. <<ekspresi1>> dijalankan
2. Nilai <<ekspresi2>> diperiksa apakah 0 (nol) atau tidak. Jika 0 (nol), maka <<statement>> tidak dijalankan. Jika tidak 0 (nol), maka <<statement>> dijalankan.
3. Setelah <<statement>> dijalankan, <<ekspresi3>> dijalankan. Kemudian kembali lagi ke langkah 2.

Contoh dalam bahasa C:

```
for( i = 10; i > 5; i = i - 1 );
```

Bandingkan dengan yang diberikan pada contoh *WHILE*. Pada contoh ini, struktur kontrol *FOR* tetap menjalankan statement. Walaupun statement merupakan statement yang kosong. Ingat bahwa tanda titik-koma (;) merupakan sebuah *statement-terminator*. Bisa dikatakan bahwa pada contoh ini, struktur kontrol *FOR* tidak menjalankan statement apa-apa tapi tetap mengikuti langkah-langkah yang di atas sampai <<ekspresi2>> bernilai 0 (nol).

b. Looping using "while"



Bentuk umumnya adalah

```
while( <<ekspresi>> ) <<statement>>;
```

struktur kontrol *WHILE* ini akan memeriksa ekspresi yang diberikan bernilai 0 (nol) atau tidak. Jika ekspresi tidak bernilai 0 (nol), maka statement dijalankan. Setelah menjalankan statement, ekspresi diperiksa kembali apakah bernilai 0 (nol) atau tidak. Demikian seterusnya sampai ekspresi bernilai 0 (nol).

Jika ekspresi tidak mungkin bernilai 0 (nol), maka situasi ini disebut *infinite loop* atau *looping forever*.

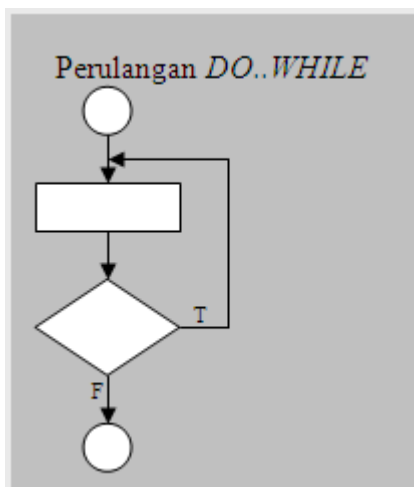
Contoh dalam bahasa C:

```
int i = 10;
int j = 5;
while( i > 5 ) i = i - 1;

while( a ) scanf( "%d", &a );

while(j>0)
{
    printf("Nilai j sekarang adalah : %d",j);
    j--;
}
```

c. Looping using "do..while"



Bentuk umumnya adalah:

```
do <<statement>>; while( <<ekspresi>> );
```


Struktur kontrol *DO..WHILE* menjalankan statement kemudian memeriksa ekspresi yang diberikan. Jika ekspresi tidak bernilai 0 (nol), maka statement dijalankan lagi. Setelah statement dijalankan, ekspresi diperiksa kembali apakah bernilai 0 (nol) atau tidak. Demikian seterusnya sampai ekspresi bernilai 0.

Jika ekspresi tidak mungkin bernilai 0 (nol), maka situasi ini disebut *infinite loop* atau *looping forever*.

Contoh dalam bahasa C:

```
int i = 10;

do
{
    printf("Nilai i sekarang adalah : %d",i);
    i--;
}while(i>20);
```

Algoritma & Pemrograman	
Module 05 - Array	
Last Update 7 Oktober 2010	Revision 01

1. Module Description

Modul ini menjelaskan tentang array, bagaimana cara kerja dan konsep penggunaan array.

2. Learning Outcomes

- Membuat program dalam bentuk function-function dengan pengiriman parameter called by value & by reference
- Local variabel dan global variabel

3. Material

a. Latar Belakang Array

- Sejauh ini cuplikan program yang kita pelajari masih sangat terbatas, karena statement assignment hanya berupa pemberian satu nilai pada satu variabel
- Padahal sering kali kita perlu untuk meng-assign ataupun memanipulasi banyak nilai ke sekelompok variabel
- Sebagai contoh jika anda membuat program yang membaca 6000 nilai percobaan untuk dihitung nilai rata-rata-nya, akan sangat lucu jika anda menghitungnya dengan cara berikut: $\text{rata_rata} = (\text{x1} + \text{x2} + \text{x3} + \text{x4} + \text{x5} + \dots + \text{dst}) / 6000$.
- Ekspresi matematis berikut akan lebih baik : $\bar{x} = \frac{1}{n} \sum_{k=1}^n x_k$

b. Pendahuluan

i. Deskripsi Array

Array Kumpulan nilai dengan tipe data yang sama yang menggunakan nama sama.. Jika dilihat dari penempatan elemen sebuah array di dalam memori, maka elemen pertama dari array tersebut (indeks yang ke-0) ditempatkan di alamat memori yang lebih rendah dibandingkan dengan elemen kedua, ketiga, dan seterusnya. Elemen-elemennya secara khusus dapat diakses dengan menggunakan indeks. Dalam C / C++ tidak ada tipe data *string* seperti pada Pascal tetapi yang ada adalah *array of char*.

ii. Deklarasi Array

- Tipe data elemen array
- Nama array
- Jumlah elemen array

iii. Prinsip kerja Array

- Jika sebuah array y memiliki n elemen, maka:
- **Elemen pertama adalah : $y[0]$**
- **Elemen terakhir adalah : $y[n-1]$**

iv. Array dapat dibedakan menjadi :

- Array berdimensi satu (1D), berpadanan dg vektor di Matematika
- Array berdimensi dua (2D), berpadanan dg matriks di Matematika
- Array berdimensi banyak

c. Alokasi Memory Array

- Array, seperti halnya variabel biasa ataupun fungsi harus dideklarasikan terlebih dahulu.
- Array dapat dideklarasikan secara global dengan mendeklarasikannya diluar fungsi main
- Hati-hati jika anda mendeklarasikannya secara (didalam sebuah fungsi atau main), karena, variabel lokal akan dibentuk dalam stack ketika fungsi tersebut dibentuk, dan akan dihapus ketika fungsi tersebut dihancurkan (ketika fungsi tersebut selesai dieksekusi).
- Mungkin hal ini tidak bermasalah bagi variabel biasa, namun array umumnya akan menuntut jumlah memory yang sangat besar (array 'float mydata[5000]' akan membutuhkan memory sebesar 20000 bytes), sementara stack hanya memiliki kapasitas sekitar 2000 - 4000 bytes
- Oleh karena itu untuk array yang besar harus dideklarasikan secara global, atau mempergunakan static statement (static float mydata[5000];) jika anda hendak mendeklarasikannya secara local.

d. Array Satu Dimensi

- Bentuk umum :

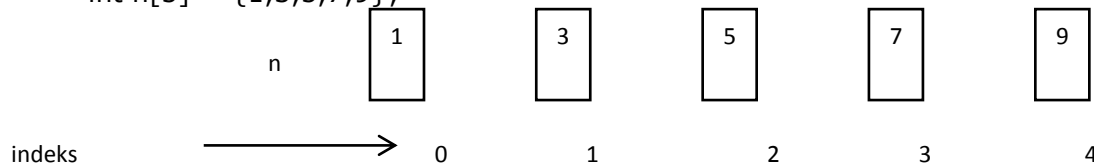
```
var_arr [size]; /*memiliki indeks dari 0 sampai (size-1) dengan  
elemen          sebanyak (size) buah*/
```

Contoh :

```
int angka[10]={10,9,8,7,6,5,4,3,2,1}
```

ini berarti nilai dari angka[0] adalah 10, nilai dari angka [5] adalah 5, dan seterusnya.
- Contoh lain array integer satu dimensi bernama n yg memiliki 5 elemen,
 $n[0] = 1, n[1] = 3, n[2] = 5, n[3] = 7, n[4] = 9$ dideklarasikan sbb:

```
int n[5] = {1,3,5,7,9};
```



e. Array Dua Dimensi


- Bentuk umum :

```
type var_arr[size1][size2];
```

f. Array multi dimensi

- **Bentuk umum :**

```
Type var_arr[size1][size2]...[sizen];
```


Algoritma & Pemrograman	
Module 06 - Function	
Last Update 07-10-2010	Revision 00

1. Module Description

Fungsi merupakan salah satu cara yang dapat mempermudah user dalam membuat program berskala menengah keatas.

2. Learning Outcomes

- Membuat program dalam bentuk function-function dengan pengiriman parameter called by value & by reference
- Local variabel dan global variabel

3. Material

a. Deskripsi

Program C sebagaimana dikatakan adalah kumpulan dari fungsi-fungsi/modul-modul. Tahap-tahap di C direpresentasikan dalam fungsi yang mengembalikan nilai yang diabaikan (*void*). Sebuah fungsi memiliki argumen yang disebut dengan parameter. Parameter ini digunakan untuk passing data antar fungsi yang satu dengan fungsi lainnya, tentu saja jika data yang digunakan adalah data global maka sebuah fungsi tidak perlu menggunakan parameter, karena data global dikenal oleh semua fungsi-fungsi yang ada di bawahnya.

b. Contoh

Contoh :

```

int angka;
void Input (void) {
}

void main (void) {
}

void Input (int *a) {
    cin >> a;
    cin >> angka;
}

void main(void) {
    Input ( );
    cout << angka;
    cout << angka;
}

int angka;
Input (&angka);

```

Pada contoh sebelah kiri kita tidak perlu mentransfer variabel *angka* ke fungsi input karena variabel *angka* juga dikenal di fungsi *Input ()*, sedangkan contoh sebelah kanan kita perlu mentransfer variabel *angka* ke fungsi *Input ()* agar dapat dikenal oleh si fungsi *Input ()*, dikarenakan variabel tersebut adalah variabel yang

lokal di fungsi *main* (). Kita juga bisa mentransfer data dalam bentuk array maupun struct.

Catatan : parameter adalah sebuah variabel, ekspresi, maupun fungsi, atau beberapa variabel, ekspresi, maupun fungsi yang dipisahkan oleh tanda ",", yang dituliskan di antara tanda "(" dan tanda ")" pada suatu fungsi. Parameter yang terdapat pada definisi suatu fungsi disebut parameter formal, sedangkan parameter saat fungsi tersebut dipanggil disebut parameter aktual.

Dalam C dikenal dua macam passing parameter, yaitu :

By value : hanya mengirimkan nilai pada parameter aktual tanpa mengubah isi dari data yang terdapat pada parameter aktual.

By pointer : dengan mengirimkan address / alamat dari suatu variabel, yang dituliskan pada parameter aktual dengan tanda "&" dan diterima dengan pointer (tanda "**") pada parameter formal. Dengan passing parameter by reference kita dapat mengubah nilai data yang ditransfer melalui parameter aktual.

c. Passing :

- Value

```
void Cetak (int x,int y,char *kalimat)
{
    gotoxy(x,y);
    puts(kalimat);
}
```

- Pointer

```
//parameternya berupa alamat dari variabel alamat
void tukar(int *a, int *b)
{
    int temp;
    temp = *a;//temp menampung isi dari alamat a
    *a = *b; // alamat a menampung isi dari alamat b
    *b = temp; // alamat b menampung isi dari variable temp
}
```

- Reference

```
// variable dari luar fungsi akan dipanggil sebagai a dan b di dalam fungsi swap
void Swap (int &a,int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void main (void)
{
    int x = 8,y = 9;
    Swap (x,y);
}
```

Pada contoh passing parameter by value di atas, terdapat parameter formal dengan tanda pointer di depannya, yaitu "char *kalimat", hal ini merupakan pengecualian untuk data yang tipenya array, dalam hal ini array of char / string,

untuk tipe data semacam ini passing parameter pastilah by pointer, karena nama suatu variabel array sudah berupa alamat pada memori tanpa menggunakan tanda "&", jadi untuk mentransfer data berupa array, data yang dikirimkan pada parameter aktual adalah nama variabelnya saja tanpa tanda "&" di depannya, sama seperti passing parameter by value, tetapi diterima sebagai pointer.

Pada C terdapat kebiasaan menulis function sesudah function *main()* , tetapi dengan catatan hal ini bisa dilakukan dengan menambahkan deklarasi fungsi / prototype fungsi di atas fungsi *main ()*.

Contoh :

```
# include <stdio.h>

void swap (int *,int *);          //Deklarasi / prototype fungsi

void main (void)
{
    int a = 1, b = 2;
    swap (&a,&b);
    //memberi alamat dari variabel a dan b pada parameter
}


void swap (int *a,int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp ;
}
```

Catatan : Dengan menggunakan prototype fungsi, dalam membuat fungsi-fungsi dalam C, kita tidak perlu mengatur letak dari definisi fungsi-fungsi tersebut sehingga memudahkan kita dalam menggunakan fungsi-fungsi selain fungsi main () yang memanggil fungsi-fungsi buatan lainnya.

Contoh :

Tanpa prototype program di bawah akan terdapat error. Dengan prototype program di bawah tidak akan error.

<pre>void A (void) { } void A (void) { } void B (void) { } void main (void) {</pre>	<pre>void A (void); B (); void B (void); void B (void) { clrscr (); } void main (void) { clrscr (); A (); }</pre>
--	--

Algoritma & Pemrograman	
Module 07 Built In Function	
Last Update 07-10-2010	Revision 03

1. Module Description

Penggunaan built-in function pada bahasa pemrograman C.

2. Learning Outcomes

- Membuat program yang menggunakan built-in function.
- Dapat menggunakan built-in function dan diaplikasikan ke dalam program yang dibuat.

3. Material

Built-in function adalah suatu fungsi yang sudah disediakan oleh suatu bahasa pemrograman. Fungsi tersebut dipanggil dengan meng-include header file dari fungsi tersebut (berekstensi .h).

Beberapa contoh built-in function pada bahasa C:

a. **printf(const char *_Format, ...)**

Fungsi ini dipanggil dengan meng-include header **stdio.h**. Fungsi ini digunakan untuk mencetak huruf / kata ke dalam console (output).

```
#include <stdio.h>

void main() {
    printf("Hello World");
    getchar();
}
```

Hasil output dari codingan di atas:

```
Hello World
```

b. **strlen(const char *_Str)**

Fungsi ini dipanggil dengan meng-include header **string.h**. Fungsi ini digunakan untuk menghitung panjang dari suatu kata. Fungsi ini menerima parameter string.

```

#include <stdio.h>
#include <string.h>

void main() {
    char word[] = "This is a string";
    int len;

    len = strlen(word);
    printf("Length of word is: %d", len);
    getchar();
}

```

Hasil output dari codingan di atas:

```
Length of kata is: 16
```

c. **strcpy(char *_Dest, const char *_Source)**

Fungsi ini dipanggil dengan meng-include header **string.h**. Fungsi ini digunakan untuk meng-copy kata. Parameter pertama adalah variable tujuan, sedangkan parameter kedua adalah string yang ingin di-copy.

```

#include <stdio.h>
#include <string.h>

void main() {
    char word1[] = "This is a string";
    char word2[30];

    strcpy(word2, word1);
    printf("Value of word2 is: %s", word2);
    getchar();
}

```

Hasil output dari codingan di atas:

```
Value of word2 is: This is a string
```

d. **rand()**

Fungsi ini dipanggil dengan meng-include header **stdlib.h** dan **time.h** (untuk fungsi **srand()**). Fungsi ini digunakan untuk me-random angka.

```

#include <stdio.h>
#include <stdlib.h> //untuk fungsi rand()
#include <time.h> //untuk fungsi time

void main() {
    int number;
    srand(time(NULL)); //mencegah random yang sama
    number = rand()%10; //random antara 0-9

    printf("Number: %d", number);
    getchar();
}

```

Hasil output dari codingan di atas:

Number: 7

e. pow(double _X, double _Y)

Fungsi ini dipanggil dengan meng-include header **math.h**. Fungsi ini digunakan untuk melakukan perhitungan pangkat. Parameter pertama (n^m) adalah angkanya, sedangkan parameter ke dua (n^m) adalah basisnya.

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

void main() {
    double hasil;

    hasil = pow((double)2, (double)3);
    printf("2 to the power of 3 is %lf", hasil);
    getchar();
}
```

Hasil output dari codingan di atas:

2 to the power of 3 is 8.000000

f. atoi(const char *_Str)

Fungsi ini dipanggil dengan meng-include header **stdlib.h**. Fungsi ini digunakan untuk meng-convert dari string ke angka. Apabila string tidak dapat di-convert, atoi akan mengembalikan nilai 0.

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

void main() {
    int number;
    char word[20];

    printf("Input a number: ");
    scanf("%s", word);
    fflush(stdin);

    number = atoi(word);
    printf("Number : %d", number);
    getchar();
}
```

Hasil output dari codingan di atas:

Input a number: 6
Number : 6

g. itoa(int _Val, char *_DstBuf, int _Radix)

Fungsi ini dipanggil dengan meng-include header **stdlib.h**. Fungsi ini merupakan kebalikan dari fungsi **atoi()**, dimana fungsi ini akan meng-convert dari angka menjadi string dengan beberapa format (biner, desimal, hexa). Apabila kita menginginkan hasil dalam bentuk desimal, nilai **_Radix** harus diisi 10.

```

#include <stdio.h>
#include <stdlib.h>

void main(){
    int number;
    char word[20];

    printf("Input a number: ");
    scanf("%d", &number);
    fflush(stdin);

    itoa(number, word, 10);
    printf("Word: %s", word);
    getchar();
}

```

Hasil output dari codingan di atas:

```

Input a number: 30
Word: 30

```

h. **sprintf(char *_Dest, const char *_Format, ...)**

Fungsi ini dipanggil dengan meng-include header **stdio.h**. Penggunaan fungsi ini mirip dengan fungsi **printf()**, hanya saja, **sprintf()** akan mencetak ke dalam string.

```

#include <stdio.h>

void main(){
    int number = 10;
    char word[20];
    sprintf(word, "Number: %d", 10);
    printf("Word -> %s", word);
    getchar();
}

```

Hasil output dari codingan di atas:

```

Word -> Number: 10

```

i. **isdigit(int _C)**

Fungsi ini dipanggil dengan meng-include header **ctype.h**. Fungsi ini digunakan untuk mengecek angka atau bukan. Apabila bukan angka, fungsi **isdigit()** akan mengembalikan nilai 0.

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char word[20];
    int i, flag;
    printf("Student ID: ");
    scanf("%s", word);
    fflush(stdin);

    flag = 1;
    for (i=0; i<strlen(word); i++){
        if (isdigit(word[i]) == 0) flag = 0;
    }
    if (flag == 1){
        printf("Valid Student ID");
    }else{
        printf("Invalid student ID");
    }

    getchar();
}

```

Hasil output dari codingan di atas:

```

Student ID: 1401010101
Valid Student ID

```

j. isalpha(int _C)

Fungsi ini dipanggil dengan meng-include header **ctype.h**. Penggunaan fungsi ini mirip dengan fungsi **isdigit()**. Hanya saja, fungsi ini akan mengecek alfabet atau bukan. Apabila bukan alfabet, fungsi **isalpha()** akan mengembalikan nilai 0.


```
#include <stdio.h>
#include <string.h>
#include <ctype.h>


void main() {
    char word[20];
    int i, flag;
    printf("Insert word [alphabet only]: ");
    scanf("%[^\n]", word);
    fflush(stdin);

    flag = 1;
    for (i=0; i<strlen(word); i++){
        if (isalpha(word[i]) == 0) flag = 0;
    }
    if (flag == 1){
        printf("Valid Word");
    }else{
        printf("Invalid Word");
    }

    getchar();
}
```

Hasil output dari codingan di atas:

```
Insert word [alphabet only]: quick
Valid Word
```

Algoritma & Pemrograman	
Module 08 - Rekursif	
Last Update 6-10-2010	Software Laboratory Center Assistant Research Enrichment Revision 01

1. Module Description

Modul ini akan menjelaskan tentang bagaimana membuat sebuah program rekursif dan juga pola berjalannya program rekursif.

2. Learning Outcomes

- Trainee dapat memahami konsep rekursif
- Trainee dapat membuat program dengan fungsi rekursif

3. Material

a. Rekursif

Rekursif adalah alat/cara untuk memecahkan masalah dalam suatu fungsi atau procedure yang memanggil dirinya sendiri.

Contoh kasus dimana suatu masalah dapat diselesaikan dengan rekursif adalah faktorial. Fungsi factorial dari bilangan bulat positif n didefinisikan sebagai berikut:

$$n! = n \cdot (n-1)! \text{ , jika } n > 1$$

$$n! = 1 \text{ , jika } n = 0, 1$$

contoh :

$$3! = 3 \cdot 2!$$

$$3! = 3 \cdot 2 \cdot 1!$$

$$3! = 3 \cdot 2 \cdot 1$$

$$3! = 6$$

Kita dapat menuliskan fungsi faktorial seperti di bawah ini :

```
int faktorial(int n){
    if(n==0 || n==1)
        return 1; //baris 3
    else
        return n*faktorial(n-1); //baris 5
}
```

Pada baris 3 dari fungsi diatas, nilai n dicek sama dengan 0 atau 1, jika ya, maka fungsi mengembalikan nilai 1 (baris 3), jika tidak, fungsi mengembalikan nilai $n * \text{faktorial}(n-1)$ (baris 5) disinilah letak proses rekursif itu, perhatikan fungsi faktorial ini memanggil dirinya sendiri tetapi dengan parameter $(n-1)$

Contoh lain kasus kedua yang bisa diselesaikan dengan rekursif adalah bilangan fibonacci. Fungsi lain yang dapat diubah ke bentuk rekursif adalah perhitungan Fibonacci. Bilangan Fibonacci dapat didefinisikan sebagai berikut:

$$f_n = f_{n-1} + f_{n-2} \text{ untuk } n > 2$$

$$f_1 = 1$$


$$f_2 = 1$$

Berikut ini adalah barisan bilangan Fibonacci mulai dari $n=1$

1 1 2 3 5 8 13 21 34

Kita dapat menuliskan fungsi fibonacci seperti di bawah ini :

```
int fibonacci(int n){  
    if(n==1 || n==2)  
        return 1;  
    else  
        return fibonacci(n-1)+fibonacci(n-2);  
}
```

Algoritma & Pemrograman	
Module 09 - File Operation	
Last Update 07-10-2010	Software Laboratory Center Assistant Research Enrichment Revision 01

1. Module Description

Implementasi baca tulis file dalam pembuatan program

2. Learning Outcomes

- Membaca data dari file.
- Menulis data ke file.

3. Material

a. Baca dan Tulis File

File teks adalah suatu file yang pola penyimpanan datanya dalam bentuk karakter. Sehingga jika suatu variabel bertipe int dengan isi 10000, maka akan disimpan dalam bentuk karakter 10000 (5 karakter). File biner adalah suatu file yang pola penyimpanannya adalah dengan menuliskan data secara biner. Sebagai contoh, jika data yang disimpan ke file tersebut bertipe int dengan isi variabel 10000, maka akan disimpan dengan isi : 2710h. Tipe jenis ini sangat cocok untuk penyimpanan data dalam bentuk struct. Pembacaan file biner dilakukan perblok.

Perbedaan file Text dan file Biner:

File Text:

- Penyimpanan data dalam bentuk karakter.
- Lebih lambat diakses
- Mode operasi untuk membuat file adalah "w" (write)
- Mode operasi untuk membaca file adalah "r" (read)

File Biner:

- Penyimpanan data dengan menuliskan data secara biner.
- Lebih cepat diakses
- Mode operasi untuk membuat file adalah "wb" (write binary)
- Mode operasi untuk membaca file adalah "rb" (read binary)

Fungsi-fungsi pada operasi file:

- `fopen()`; : merupakan salah satu operasi file yang berfungsi untuk mengaktifkan sebuah file. hal ini dilakukan agar file tersebut dapat diakses.
Contoh : `FILE *fopen(char *namafile, char *mode);`
- `fclose()`; : merupakan salah satu operasi file yang berfungsi untuk menutup suatu file, hal ini dilakukan karena adanya keterbatasan jumlah file yang dapat diakses secara serentak. file yang tidak lagi digunakan sebaiknya ditutup.

Contoh : `int fclose(FILE *pf);`

- `fscanf();` : merupakan suatu operasi file yang berguna untuk membaca kembali sebuah data bilangan yang telah disimpan dalam sebuah file dengan keadaan diformat.
Contoh : `fscanf(ptr_file, "string kontrol", daftar argumen);`
- `fprintf();` : berfungsi untuk menyimpan sebuah data bilangan dalam sebuah file dalam keadaan diformat.
Contoh : `fprintf(ptr_file, "string kontrol", daftar argumen);`
- `fgets();` : merupakan suatu operasi file yang berfungsi untuk membaca string dari file sampai ditemukannya baris baru (`'\n'`) atau setelah `n-1` karakter, dengan `n` adalah panjang maksimal string yang dibaca perwaktu-baca.
Contoh : `char *fgets(char *str, int n, FILE *ptr_file);`
- `fputs();` : merupakan salah satu operasi file yang berfungsi untuk menyimpan data bertipe string kedalam file.
Contoh : `int fputs(char *str, FILE *ptr_file);`
- `fgetch();` : merupakan suatu operasi file yang berfungsi untuk membaca suatu karakter dari sebuah file.
- `fputc();` : merupakan operasi file yang berguna untuk menyimpan sebuah karakter kedalam file.
- `fread();` : merupakan suatu fungsi yang memungkinkan untuk membaca data file dalam bentuk kesatuan blok (sejumlah byte), misalnya untuk membaca sebuah data bertipe float.
Contoh : `int fread(void *buffer, int n, FILE *ptr_file);`
- `fwrite();` : merupakan suatu fungsi yang memungkinkan untuk menyimpan data file dalam bentuk kesatuan blok (sejumlah byte), misalnya untuk menyimpan sebuah data bertipe float.
Contoh : `int fwrite(void *buffer, int jum_byte, int n, FILE *ptr_file);`
- `fseek();` : merupakan suatu operasi file yang berfungsi untuk menempatkan penunjuk file ke suatu lokasi dalam file berdasarkan offset dan posisi. Dapat juga digunakan untuk membaca data secara acak dan memungkinkan juga untuk melakukan pengubahan data secara acak.
Contoh : `int fseek(FILE *ptr_file, long int offset, int posisi);`
- `feof();` : merupakan suatu operasi file yang berfungsi untuk mendeteksi akhir dari suatu file. Keluaran `feof()` berupa nilai null (`""`) jika operasi pembacaan yang terakhir (misalnya `getw()`) membaca tanda akhir file.
Contoh : `int feof(FILE *ptr_file);`

Jenis-jenis operasi file:

- `r/rt` = Menyatakan file yang akan hanya dibaca, dalam hal ini file yang akan dibaca haruslah sudah tersimpan pada disk.
- `w/wt` = Menyatakan bahwa file baru diciptakan. Selanjutnya operasi yang akan dilakukan terhadap file adalah operasi perekam data. Seandainya file tersebut sudah ada pada disk, isinya akan terhapus.
- `a/at` = Untuk membuka file yang sudah ada pada disk dan operasi yang akan dilakukan adalah operasi penambahan data pada file. Data baru akan ditempatkan dibagian belakan dari file. Seandainya file belum ada, secara otomatis file akan diciptakan terlebih dahulu.
- `r+/rt+` = Untuk membuka file yang sudah ada, dan operasi yang dilakukan berupa pembacaan serta penulisan. Data yang terakhir diisikan akan dihapus dan diganti data yang baru.
- `w+/wt+` = untuk membuka file dengan tujuan untuk pembacaan atau penulisan. Jika file sudah ada, isinya akan dihapus.

- a+/a+ = untuk membuka file yang ada dan akan dilakukan penambahan data baik berupa perekaman maupun pembacaan.
- rb = Merupakan file yang akan hanya dibaca, namun pembacaan dilakukan dalam keadaan binary mode, dalam hal ini file yang akan dibaca haruslah sudah ada didalam disk. Pembacaan file dilakukan per byte/ per record.
- wb = Menyatakan bahwa file baru diciptakan, penciptaan file akan dilakukan dalam keadaan binary mode. Selanjutnya operasi yang akan dilakukan terhadap file adalah operasi perekam data. Seandainya file tersebut sudah ada pada disk, isinya akan terhapus.
- ab = Untuk membuka file yang sudah ada pada disk dalam keadaan binary mode dan operasi yang akan dilakukan adalah operasi penambahan data pada file. Data baru akan ditempatkan dibagian belakan dari file. Seandainya file belum ada, secara otomatis file akan diciptakan terlebih dahulu.
- rb+ = Untuk membuka file yang sudah ada dalam keadaan binary mode, dan operasi yang dilakukan berupa pembacaan serta penulisan dalam keadaan binary mode. Data yang terakhir diisikan akan dihapus dan diganti data yang baru.
- wb+ = Untuk membuka file dengan tujuan untuk pembacaan atau penulisan dalam keadaan binary mode. Jika file sudah ada maka isinya akan terhapus.
- ab+ = Untuk membuka file dalam keadaan binary mode, dengan operasi yang dapat dilakukan berupa perekaman maupun pembacaan dalam keadaan binary mode. Jika file sudah ada isinya akan terhapus.

Contoh Source Code Program

Contoh program tulis file:

```
void main(){
    FILE *tulis;
    char write[100];
    tulis = fopen("file.txt", "w");

    if(tulis != NULL){
        do{
            printf("Masukkan String [quit untuk keluar] : ");
            scanf("%[^\\n]", &write);
            fflush(stdin);
            if(strcmp(write, "quit") != 0)
                fprintf(tulis, "%s\\n", write);
            else
                break;
        }while(1);
        fclose(tulis);
        printf("\\nFile selesai dibuat");
    }
    else{
        printf("\\nFile gagal dibuat");
    }
    getchar();
}
```

Contoh program baca file:


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

```

void main(){
    FILE *baca;
    char read[100];
    baca = fopen("file.txt","r");

    if(baca != NULL){
        while(fscanf(baca,"%s",read)!=EOF)
            printf("%s\n",read);
        fclose(baca);
        printf("\nFile selesai dibaca");
    }
    else{
        printf("\nFile gagal dibaca");
    }
    getchar();
}

```

Algoritma & Pemrograman	
Module 10 - Sorting	
Last Update 07-10-2010	Revision 02

1. Module Description

Modul ini menjelaskan tentang sorting, Jenis-jenis Sorting dan bagaimana cara kerja sorting beserta dengan contoh code-nya.

2. Learning Outcomes

- Trainee memahami tentang algoritma sorting
- Trainee dapat melakukan pembuatan dan implementasi sorting.
-

3. Material

b. Sorting

Didalam ilmu komputer dan matematika, sebuah algoritma pengurutan adalah algoritma yang menempatkan unsur-unsur dari sebuah daftar dalam suatu tatanan. Salah satu tujuan sorting adalah mempercepat pencarian data (searching, retrieving).

Pengurutan data dilakukan berdasarkan nilai kunci(key). Misalnya sejumlah data (record) mahasiswa BINUS University yang masing-masing terdiri atas NIM, Nama, dan ipk, maka kunci pengurutan dapat berupa NIM, ataupun IPK bila ingin didapatkan susunan data mulai dari IPK tertinggi sampai dengan IPK terendah ataupun sebaliknya.

Berdasarkan perbandingan nilai data, maka sorting dapat dilakukan secara menaik (*ascending*, atau *nondesceding order*) dan menurun(*descending*, atau *nonncreasing order*).

Terdapat Beberapa Jenis Sorting, seperti :

- **Insertion Sort**

Algoritma insertion sort pada dasarnya memilah data yang akan diurutkan menjadi dua bagian, yang belum diurutkan (meja pertama) dan yang sudah diurutkan (meja kedua). Elemen pertama diambil dari bagian array yang belum diurutkan dan kemudian diletakkan sesuai posisinya pada bagian lain dari array yang telah diurutkan. Langkah ini dilakukan secara berulang hingga tidak ada lagi elemen yang tersisa pada bagian array yang belum diurutkan.

- **Selection Sort**

Ide utama dari algoritma selection sort adalah memilih elemen dengan nilai paling rendah dan menukar elemen yang terpilih dengan elemen ke-i. Nilai dari i dimulai dari 1 ke n, dimana n adalah jumlah total elemen dikurangi 1.

- **Merge Sort**

Merge sort menggunakan pola divide and conquer yang mengimplementasikan konsep rekursi untuk menyelesaikan permasalahan. Permasalahan utama kemudian dipecah menjadi sub-masalah, kemudian solusi dari sub-masalah akan membimbing menuju solusi permasalahan utama.

- **Quicksort**

Quicksort ditemukan oleh C.A.R Hoare. Seperti pada merge sort, algoritma ini juga berdasar pada pola divide-and-conquer.

- **Bubble Sort**

algoritma penyortiran yang dimulai dan berakhir pada sebuah daftar dengan n elemen dan memindahkan seluruhnya, menguji nilai setiap pasangan item yang berdekatan dan menukarkannya jika mereka tidak berada dalam urutan yang tepat.

Untuk lebih lengkapnya, dapat di baca di

<http://poss.ipb.ac.id/files/JENI-Intro2-Bab06-Algoritma%20Sorting.pdf>

Berikut ini adalah contoh codingan sorting, dengan algoritma bubble sort :

```
#include<stdio.h>

//fungsi untuk menukar nilai dari 2 variabel
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// fungsi untuk bubble sort
void bubbleSort(int numbers[], int n)
{
    int i, j;
    for(i=0;i<n;i++)
        for(j=0;j<n-(i+1);j++)
            if(numbers[j] > numbers[j+1])
// pada if di atas, jika > ditukar dengan <, sorting menjadi descending
                swap(&numbers[j], &numbers[j+1]);
}

int main()
{
    const int N = 9; // jumlah data dalam array
    int angka[N] = {8,7,9,1,6,5,4,3,2}; // angka sebelum di sort


    bubbleSort(angka, N); // dilakukan bubble sort pada array angka

    // mencetak isi dari array angka
    for(int k=0;k<N;k++)
        printf("%d ", angka[k]);

    getchar();
    return 1;
}
```

Output program di atas adalah :

```
1 2 3 4 5 6 7 8 9
```

Algoritma & Pemrograman	
Module 11 - String	
Last Update 07-10-2010	Revision 01

1. Module Description

Modul ini menjelaskan tentang string, bagaimana mendklarasi sebuah string dan meberikan value terhadap-nya.

2. Learning Outcomes

- Trainee memahami tentang string
- Trainee dapat melakukan pembuatan dan implementasi string.

3. Material

a. Deklarasi String

String adalah variable yang digunakan untuk menampung kalimat. pada bahasa C, String di deklarasikan dengan membuat array of char.

```
char kalimat[100];
```

Jadi variabel kalimat akan dapat menampung 99 karakter, bukan 100. Hal ini dikarenakan pada C, karakter terakhir pada sebuah string harus diakhiri oleh **NULL**.

NULL ini sama saja dengan angka 0 atau karakter '\0'. Karakter **NULL** ini menandai akhir dari String. Jadi misalkan :

```
char kalimat[6] = "Hallo";
```

Maka variabel kalimat akan disimpan dalam memori, dengan format :

0	1	2	3	4	5
H	A	L	L	O	\0

b. Input Output String

Kita juga dapat melakukan input output pada String. Untuk melakukan input output, jangan lupa untuk terlebih dahulu meng-include "**stdio.h**". Untuk melakukan output string, kita dapat menggunakan fungsi-fungsi, seperti :

- **printf**

printf sudah di bahas di module input ouput, jadi pada printf kita perlu memberi format dari variable yang ingin kita cetak. Untuk mencetak String ke layar console, kita menggunakan format **"%s"**.

```
char kal[100] = "Hallo Dunia";  
printf("%s", kal);
```

- **puts**

Pada fungsi puts, kita hanya cukup memberi variabel yang ingin kita cetak, dan pada akhir string akan otomatis diberikan new line(\n). Tapi fungsi puts ini kurang fleksibel, karena tidak memiliki format-format yang dapat kita atur. Jadi jika kita ingin mencetak dengan format-format tertentu, gunakan printf.

```
char kal[100] = "Hallo Dunia";  
puts(kal); // mencetak "Hallo Dunia",diakhiri \n
```

Sedangkan contoh fungsi-fungsi untuk input string adalah sebagai berikut :

- **scanf**

scanf juga sudah di bahas di module input output. Untuk input string kita menggunakan format **"%s"**.

```
char kal[100] = "";  
printf("Masukkan string : ");  
scanf("%s", kal);  
fflush(stdin);  
printf("Kata yang anda input : ");  
puts(kal);
```

coding di atas, akan meminta kita menginput sebuah string, inputan kita akan di tampung di variable **"kal"**. Kemudian isi dari variabel **"kal"** akan di cetak ke layar.

Tapi coding di atas masih memiliki kekurangan. Seandainya kita menginput string yang memiliki lebih dari 1 kata, maka hanya kata pertama yang akan diambil. Jadi misalkan kita menginput **"Selamat pagi dunia"**, variabel **"kal"** hanya akan berisi **"Selamat"**.

Untuk mengambil lebih dari 1 kata, kita perlu mengubah statement scanf yang di atas menjadi :

```
scanf("%[^\n]", kal);
```

"%[^\n]" artinya kita ingin mengambil inputan, sampai ditemukan karakter enter(\n).

- **gets**

Dengan fungsi gets, coding input string akan lebih sederhana. Contoh gets :

```
char kal[100] = "";
printf("Masukkan string : ");
gets(kal);
printf("Kata yang anda input : ");
puts(kal);
```

gets secara otomatis dapat mengambil inputan yang terdiri dari lebih dari 1 kata.

- c. **string.h**


pada header ini terdapat berbagai fungsi yang dapat memudahkan kita dalam melakukan manipulasi terhadap string. Berikut ini contoh beberapa fungsi yang ada di dalam string.h :

misalkan ada 2 variable

```
char a[100] = "Makan";
char b[100] = "Nasi";
```

Fungsi	Nilai a	Nilai b	Keterangan
strcat(a, b)	MakanNasi	Nasi	Untuk menggabungkan string
Strncat(a,b,2)	MakanNa	Nasi	Untuk menggabungkan string, tapi dapat di beri jumlah karakter yang ingin diambil.
strcpy(a, b)	Nasi	Nasi	Meng-copy string
Strncpy(a, b,3)	Nasan	Nasi	Meng-copy string, tapi kita dapat menentukan jumlah karakter yang ingin di copy.
strcmp(a, b)	Makan	Nasi	Membandingkan string dengan hasil <ul style="list-style-type: none"> • a < b, maka akan mengembalikan nilai < 0 • jika a == b, maka akan

			<p>mengembalikan nilai 0</p> <ul style="list-style-type: none"> • jika $a > b$, maka akan mengembalikan nilai > 0
strlen(a)	Makan	Nasi	Akan mengembalikan panjang karakter dari sebuah string.

Data Structure	
Module 12 - Struct	
Last Update 05-10-2010	Revision 01

1. Module Description

Pada bagian ini akan dijelaskan secara singkat definisi tentang struct dan sedikit cara pemakaian tentang Struct

2. Learning Outcomes

- Deklarasi Struct
- Penggunaan Struct

3. Material

a. Pembuka

Struct dalam C / record dalam Pascal, merupakan kumpulan field-field atau variabel-variabel yang saling berhubungan. Misalnya record mahasiswa bisa mengandung variabel nim, nama, alamat, jurusan, dll. Pendeklarasian struct dalam C / C++ menggunakan keyword *struct*. Dengan keyword *struct* ini kita seolah-olah membuat tipe data baru sesuai keinginan kita.

b. Bentuk umum suatu struct dan definisi variabel struct

```
struct <nama struct / tipe variabel buatan> {
    tipe    var1;
    tipe    var2[size];
    tipe    *var3;
    struct  <nama struct / tipe variabel buatan> *var4;
} var_struct;
atau
struct <nama struct / tipe variabel buatan>  var_struct;
```

Contoh :

```
struct myrec {
    char nim[11];
    char nama[21];
    char jurusan[3];
} dat_mhs;
```

Pada contoh di atas variabel "dat_mhs" juga bisa didefinisikan secara eksplisit dengan cara seperti ini :

```
struct myrec dat_mhs;
```

Untuk pengaksesan variable-variabel dalam struct myrec kita dapat menggunakan tanda titik (.).

Contoh :

```
gets (dat_mhs.nim);  
printf ("%s", dat_mhs.nim);
```

Catatan : Perlu anda ketahui juga bahwa variabel dari suatu struct bisa bertipe pointer, yang nantinya digunakan untuk pengalokasian memori yang dinamis, yang disebut dengan **linked list**.

c. Array of Struct

Jika sebelumnya kita telah mengetahui adanya array of character (string), maka struct pun dapat dibuat array. Cara pendefinisian sama, hanya pada saat mendefinisikan variabel struct perhatikan perbedaan antara variabel record (variabel yang ada dalam struct) dengan variabel struct, untuk variabel struct yang ingin dijadikan array kita tinggal menambahkan indeks seperti halnya mendefinisikan suatu array.

Contoh :

```
struct {  
    int hari;  
    int bulan;  
    int tahun;  
} tanggal[360];
```

Maka terdapat tempat di memory untuk 360 buah record tanggal. Cara pengaksesan variabel fieldnya pun sama, hanya tinggal menambahkan indeks. Contoh untuk menginput hari dari record pertama, maka kita bisa menuliskan perintah :

```
cin >> tanggal[0].hari;
```

Perlu anda ketahui juga dalam definisi struct di atas tidak terdapat <nama structnya / tipe data buatan> hal ini diperbolehkan jika variabel struct untuk struct tersebut hanya didefinisikan sekali saja, maksudnya tidak didefinisikan lagi di fungsi manapun. Dan memori yang dipakai untuk satu record adalah berukuran 3 * 2 byte yaitu sebesar 6 byte, karena satu record memuat 3 buah variabel bertipe int yang besarnya 2 byte.

d. Class

Dalam bahasa C++ struct sudah tidak di gunakan karena hanya beberapa bahasa pemrograman saja yang mengenal struct. Dalam bahasa C++ menggunakan class. Konsep class akan lebih dijelaskan pada mata training PBO dan JAVA. Berikut ini penggunaan class :

```
class <nama class > {  
    access modifier [private|protected|public]:  
    tipe var1;  
    tipe var2[size];  
    tipe *var3;  
    fungsi namaFungsi;
```



```
        fungsi namaFungsi1;  
};
```

Contoh :


```
struct contoh {  
    private :  
        char nim[11];  
        char nama[21];  
        char jurusan[3];  
    public :  
        void getNIM()  
{  
        Return nim;  
}  
} dat_mhs;
```

Pada contoh di atas variabel "dat_mhs" juga bisa didefinisikan secara eksplisit dengan cara seperti ini :

```
contoh dat_mhs;
```

Untuk pengaksesan variable-variabel dalam struct myrec kita dapat menggunakan tanda titik (.).

Catatan : Class tidak jauh beda dengan struct bedanya kalau class defaultnya berupa private atau tidak bisa di panggil di mana saja sedangkan struct defaultnya berupa public sehingga bisa di panggil dimana saja.

Data Structure	
Module 13 – Linked List	
Last Update 09-10-2010	Revision 02

1. Module Description

Pada bagian ini akan dijelaskan secara singkat definisi tentang single linked list dan sedikit cara pemakaian tentang single linked list

2. Learning Outcomes

- Single Linked List
- pointer of struct
- malloc
- free
- operator ->
- Push di belakang tail
- Pop head

3. Material

a. Linked List

Linked list adalah sejumlah struct/class yang saling terkait hingga membentuk suatu node/simpul berantai. Linked list sering digambarkan seperti gerbong kereta api yang saling terkait di mana masing-masing gerbong bisa memuat barang dengan kapasitas tertentu.

b. Implementasi

Tidak seperti array yang memesan memori secara statis, dengan Linked List kita bisa mengalokasi memori sesuai kebutuhan kita.

Contoh coding linked list :

```

#include <stdio.h>
#include <stdlib.h>

struct data
{
    int angka;
    struct data *next;
} *head, *tail, *curr;

void push(int angka)
{
    curr = (struct data*)malloc(sizeof(struct data));
    curr->angka = angka;
    if(head==NULL)
        head = tail = curr;
    else{
        tail->next = curr;
        tail = curr;
    }
    tail->next = NULL;
}

void pop(void)
{
    curr = head;
    head = head->next;
    free(curr);
}

```

```

void popall(void)
{
    while(head!=NULL)
        pop();
}

void cetak(void)
{
    curr = head;
    while(curr!=NULL)
    {
        printf("%d ",curr->angka);
        curr = curr->next;
    }
}

void main()
{
    int angka=0;
    char tekan;
    do{
        cetak();
        tekan = getchar();
        switch(tekan)
        {
            case '+' :   angka++;
                        push(angka);
                        break;
            case '-' :   if(head!=NULL)
                        pop();
                        break;
        }
    } while(tekan!='*');
    popall();
}

```

Keterangan :

- **head** adalah pointer yang digunakan untuk menunjuk node awal.
- **tail** adalah pointer yang digunakan untuk menunjuk node akhir.
- **curr** adalah pointer untuk menunjukkan node aktif, atau sebagai pointer pembantu saat hendak menambah atau menghapus data.

note :

kita dapat mendeklarasikan pointer tambahan seperti : *temp, *node sebagai pointer pembantu sesuai kebutuhan kita

- **malloc** adalah sintaks untuk mengalokasikan memori (memesan tempat pada memori)
- **free** adalah sintaks untuk membebaskan memori yang sebelumnya dialokasikan

Sekarang mari kita lihat coding di atas secara garis besarnya :

1. `#include <stdlib.h>`
Untuk pemakaian sintaks **malloc** kita membutuhkan header **stdlib.h**
2. Untuk linked list kita membutuhkan variabel pointer

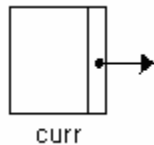
```

struct data
{
    int angka;
    struct data *next; //pointer untuk menunjuk ke node selanjutnya
} *head, *curr, *tail; //pointer untuk menunjuk ke node awal, akhir dan aktif

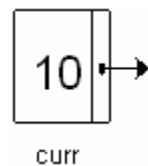
```
3. Push
 ➤ Jika kita memanggil fungsi **push** pertama kali maka yang terjadi adalah :

push(10)

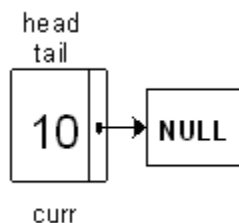
curr = (struct data *) malloc(sizeof(struct data));
 memesan tempat di memori sebesar ukuran struct data



curr->angka = angka;



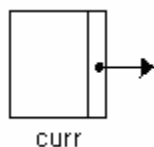
if(head==NULL)
head = tail = curr;
tail->next = NULL;



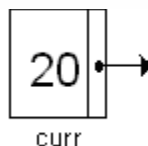
➤ Jika kita memanggil fungsi **push** untuk ke 2,3,4.... maka yang terjadi adalah :

push(20)

- curr = (struct data *) malloc(sizeof(struct data));



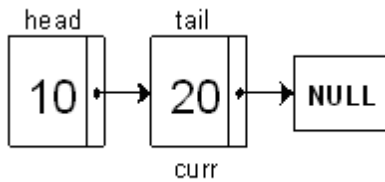
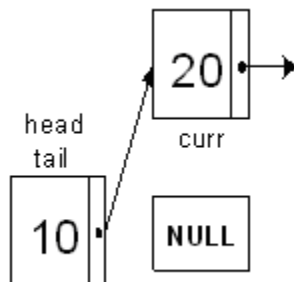
angka = angka;



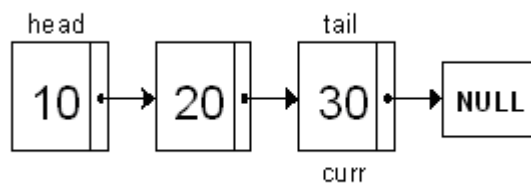
```

- else
{
    tail->next = curr;
    tail = curr;
}
tail->next = NULL;

```

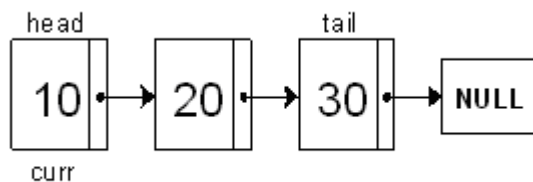


push(30)

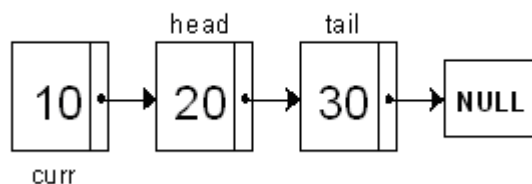


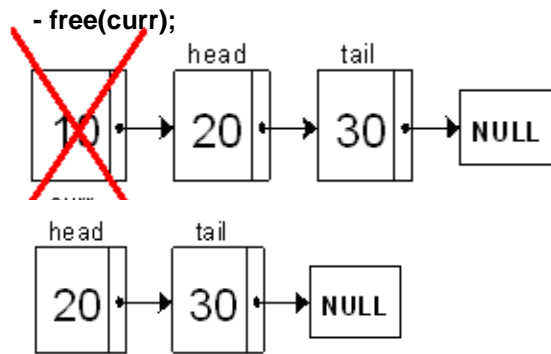
dst.. [setiap data yang dipush akan selalu diletakkan setelah tail]

4. Pop
- **curr = head;**



- **head = head->next;**





a. Penempatan Node Baru

Ditinjau dari tempat node baru ditempatkan, ada 2 teknik dasar, yaitu :

1. Push setelah **tail** (push belakang).

Pada contoh coding sebelumnya digunakan push belakang.

2. Push sebelum **head** (push depan).

Coding push depan :

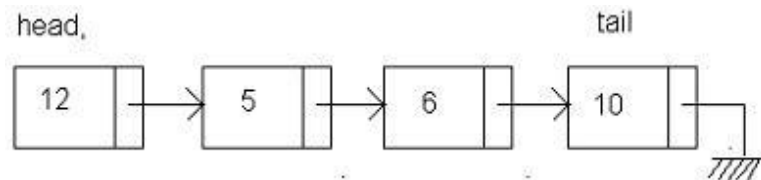
```

void push(int angka)
{
    curr = (struct data *) malloc(sizeof(struct data));
    curr->angka = angka;
    curr->next = head;
    if (head==NULL)
        head = tail = curr;
    else
    {
        curr->next = head;
        head = curr;
    }
}

```

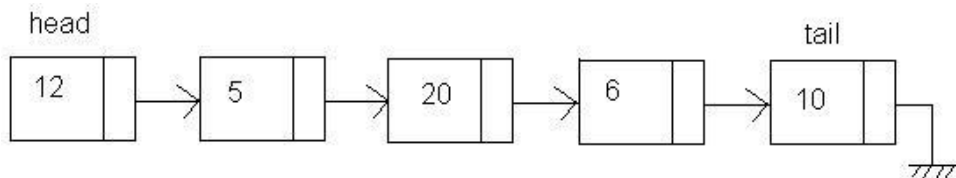
b. Menambah Node Pada Posisi Tertentu

Dengan linked list yang telah ada



kita hendak menambah node (antara nilai 5 dan 6, kita tambahkan node baru dengan nilai 20),

sehingga gambar linked list menjadi :



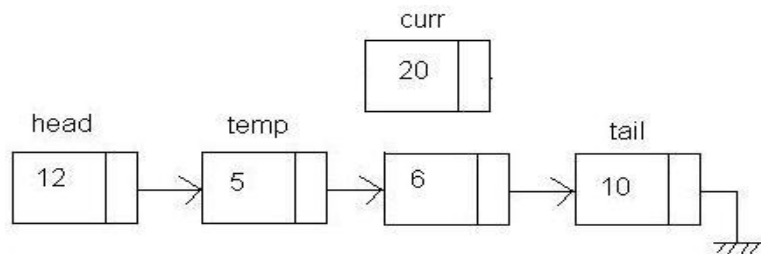
Pertama- tama, kita pesan node baru, isi node baru tersebut dengan nilai 20, maka sintaksnya :

```
curr = (struct data *) malloc (sizeof(struct data));
curr -> angka = 20;
```

Setelah itu, kita gunakan suatu pointer pembantu, yaitu temp. Gunakan temp untuk menunjuk ke node yang bernilai 5. Jangan gunakan curr !! karena curr sedang menunjuk ke node baru yang hendak kita insert !

```
temp = head;
while( temp->angka != 5 )
{
    temp = temp -> next;
}
```

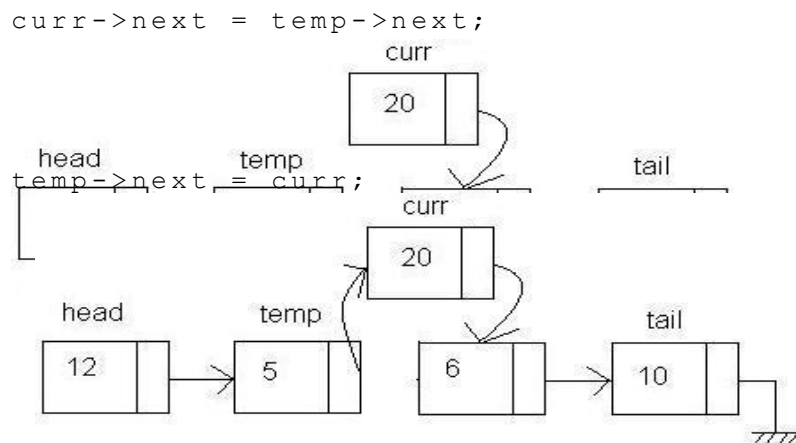
Sehingga gambaran Linked List menjadi :



Setelah itu kita sambung node curr, yaitu dengan cara :

- curr yang ke next kita tunjuk ke temp yang ke next
- temp yang ke next kita tunjuk ke curr

Gambaran Linked Listnya :



Keseluruhan sintaksnya menjadi :

```
curr = (struct data *) malloc(sizeof(struct data));
curr->angka = 20;
temp = head;
while( temp->angka != 5 )
{
    temp = temp->next;
}
```

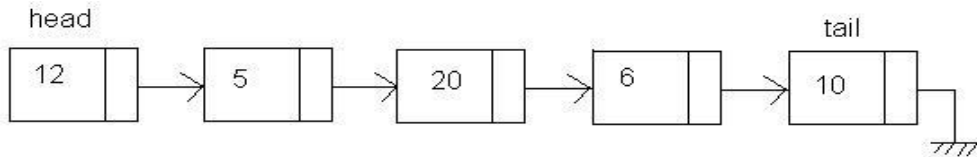


```
curr->next = temp->next;
temp->next = curr;
```

Intinya adalah sebelum dan sesudah disisipkan harus disambungkan sehingga rantai dari linked list tidak terputus

c. Menghapus Node Pada Posisi Tertentu

Pada Susunan linked link list :



Misalkan kita hendak menghapus node dengan nilai 20 !!

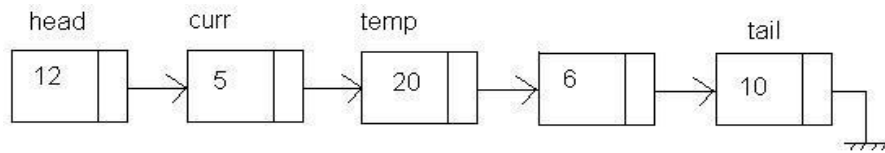
pertama - tama kita gunakan suatu pointer, curr untuk menunjuk ke node sebelum node yang hendak dihapus (pada linked - list di atas yaitu, node yang bernilai 5)

```
curr = head ;
while (curr->next->angka != 20)
{
    curr = curr->next;
}
```

Gunakan suatu pointer lagi untuk menunjuk ke node yang hendak dihapus

```
temp = curr->next;
```

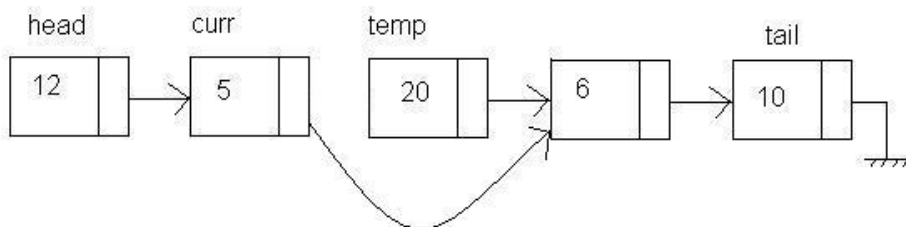
Sehingga gambaran Linked List menjadi :



Karena node yang ditunjuk pada pointer temp hendak kita hapus, maka node berikutnya setelah curr harusnya node dengan nilai 6, sehingga :

```
curr->next = temp->next ;
```

Sehingga gambaran Linked List menjadi :

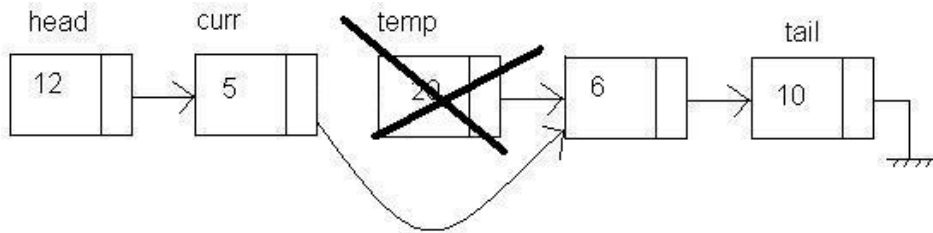


Setelah itu,

baru kita hapus node temp !

```
free (temp);
```


Sehingga gambaran Linked List menjadi :



Pada penghapusan node, perhatikan posisi node yang akan dihapus!

- Jika posisi di head, maka pindahkan head ke node selanjutnya
- Jika posisi di tail, maka pindahkan tail ke node sebelumnya

Hal ini dilakukan agar kita tidak kehilangan head dan tail yang berfungsi sebagai penunjuk data awal dan akhir node.

Data Structure	
Module 14 – Double Linked List	
Last Update 09-10-2010	Revision 03

1. Module Description

Modul Ini menjelaskan tentang struktur dan konsep Double Linked List

2. Learning Outcomes

- Trainee memahami tentang Struktur Data yang menggunakan Double Linked List
- Trainee dapat membuat fungsi push depan
- Trainee dapat membuat pop depan dan pop belakang

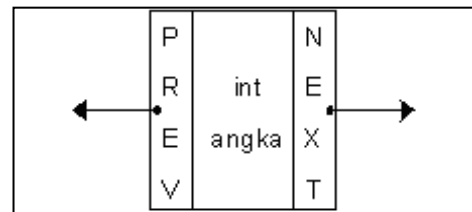
3. Material

a. Pendahuluan

Double Linked adalah linked list yang mempunyai dua pointer penghubung untuk menghubungkan tiap-tiap nodenya.

b. Deklarasi Struct

```
struct data{
    int angka;
    struct data *next;
    struct data *prev;
}*head, *curr, *tail;
```



Keterangan :

- **head** adalah pointer yang digunakan untuk menunjuk node awal.
- **tail** adalah pointer yang digunakan untuk menunjuk node akhir.
- **curr** adalah pointer untuk menunjukkan node aktif, atau sebagai pointer pembantu saat hendak menambah atau menghapus data.
- **next** adalah pointer untuk menunjuk node selanjutnya
- **prev** adalah pointer untuk menunjuk node sebelumnya

c. Push

Fungsi push yang akan dibahas disini adalah push di depan head pada Double Linked List :

```

void push(int angka)
{
    curr = (struct data *)malloc(sizeof(struct data));
    curr->angka = angka;
    curr->next = curr->prev = NULL;

    if(head == NULL) //data masih kosong
    {
        head = tail = curr;
    }
    else
    {
        head->prev = curr;
        curr->next = head;
        head = curr;
    }
}

```

Gambaran push pada Double Linked List :

d. Pop

Jika node yang akan dihapus adalah tail / data akhir, maka :

```

void pop_tail()
{
    if(head == NULL) //jika data masih kosong
    {
        printf("Data masih kosong..!");
    }
    else if(head == tail) //jika cuma ada satu data
    {
        free(head);
        head = tail = NULL;
    }
    else //jika data lebih dari satu
    {
        curr = tail;
        tail = tail->prev;
        free(curr);
        tail->next = NULL;
    }
}

```

Jika node yang akan dihapus adalah head / data awal, maka :

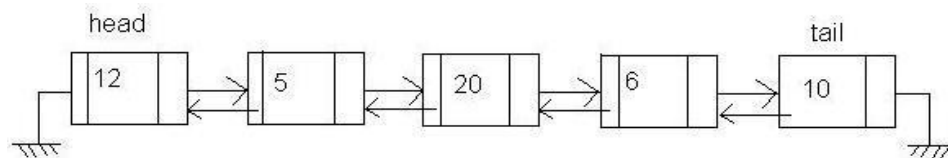
```

void pop_head()
{
    if(head == NULL) //jika data masih kosong
    {
        printf("Data masih kosong..!");
    }
    else if(head == tail) //jika data cuma satu
    {
        free(head);
        head = tail = NULL;
    }
    else //jika data lebih dari satu
    {
        curr = head;
        head = head->next;
        free(curr);
        head->prev = NULL;
    }
}

```

e. Menghapus Node pada Posisi Tertentu

Pada Susunan linked list :



Misalkan kita hendak menghapus node dengan nilai 20.

Pertama - tama kita gunakan suatu pointer, curr untuk menunjuk ke node sebelum node yang hendak dihapus (pada linked - list di atas yaitu node yang bernilai 5)

```

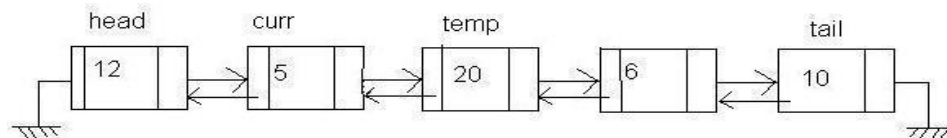
curr = head;
while (curr->next->angka != 20)
{
    curr = curr->next;
}

```

Gunakan suatu pointer lagi untuk menunjuk ke node yang hendak dihapus

```
temp = curr->next;
```

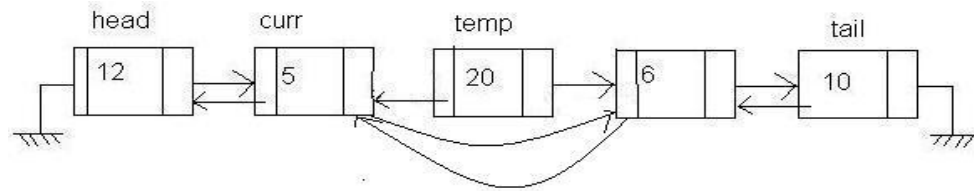
Sehingga gambaran linked list menjadi :



Karena node yang ditunjuk pada pointer temp hendak kita hapus, maka node berikutnya setelah curr harusnya node dengan nilai 6, sehingga :

```
curr->next = temp->next;  
temp->next->prev = curr;
```

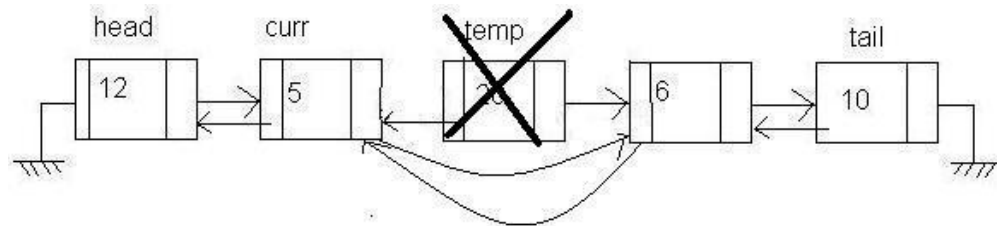
Sehingga gambaran linked list menjadi :




Setelah itu, baru kita hapus node temp !

```
free (temp);
```

Sehingga gambaran linked list menjadi :



Data Structure	
Module 15 - Queue	
Last Update 09-10-2010	Revision 01

1. Module Description

Modul ini berisi tentang Queue disertai dengan contoh code yang dapat dicoba.

2. Learning Outcomes

- Implementasi Queue dengan menggunakan Linked List

3. Material

a. Pendahuluan

Seperti kita ketahui, Queue berarti antrian dimana kita akan menggunakan konsep FIFO (First In First Out). Maka setiap node yang masuk pertama maka akan keluar juga pertama kali. Contoh yang paling populer untuk membayangkan sebuah queue adalah antrian pada kasir sebuah bank. Ketika seorang pelanggan datang, akan menuju ke belakang dari antrian. Setelah pelanggan dilayani, antrian yang berada di depan akan maju. Pada saat menempatkan elemen pada ujung (tail) dari queue disebut dengan enqueue, pada saat memindahkan elemen dari kepala (head) sebuah queue disebut dengan dequeue.

b. Pembuatan Fungsi Enqueue pada Queue

Proses enqueue adalah proses untuk penambahan di posisi belakang. Penambahan ini dilakukan jika kondisi queue tidak penuh. Jika keadaan masih kosong, maka field depan dan belakang bernilai 1 tetapi jika sudah mempunyai elemen maka yang nilai belakang harus bertambah 1. Kemudian data baru disimpan di array pada posisi belakang.

```

void enqueue(int angka)
{
    curr = (struct data *)malloc(sizeof(struct data));
    curr->angka = angka;
    curr->next = curr->prev = NULL;

    if(head == NULL)//data masih kosong
    {
        head = tail = curr;
    }
    else
    {
        tail->next = curr;
        curr->prev = tail;
        tail = curr;
    }
}

```


c. Pembuatan Fungsi Dequeue pada Queue

Operasi dequeue adalah proses pengambilan elemen queue. Tentunya elemen yang diambil selalu dari elemen pertama (1). Setelah elemen pertama diambil, maka akan diperlukan proses pergeseran elemen data setelah elemen data yang diambil (dari posisi ke-2 sampai posisi paling belakang), dan kemudian posisi belakang akan dikurangi 1 karena ada data yang diambil.

```

void dequeue()
{
    if(head == NULL) //jika data masih kosong
    {
        printf("Data masih kosong..!");
    }
    else if(head == tail) //jika data cuma satu
    {
        free(head);
        head = tail = NULL;
    }
    else //jika data lebih dari satu
    {
        curr = head;
        head = head->next;
        free(curr);
        head->prev = NULL;
    }
}

```


Data Structure	
Module 16 – Priority Queue	
Last Update 07-10-2010	Revision 01

1. Module Description

Modul ini berisi tentang Priority Queue disertai dengan contoh code yang dapat dicoba.

2. Learning Outcomes

- Implementasi Sorting dengan menggunakan Priority queue

3. Material

a. Pendahuluan

Priority queue adalah pengembangan lebih lanjut dari konsep queue yang sebelumnya kita pelajari. Dalam pembuatan priority queue kita akan menambahkan sebuah variable di dalam struct yang bertindak sebagai prioritas. Prioritas inilah yang selanjutnya akan digunakan untuk mengatur apakah node yang akan dipush harus didahulukan atau tidak. Contoh: Ketika kita mengantri di rumah sakit, pastilah orang yang sakitnya lebih parah akan didahulukan dalam antrian. Dalam kasus ini yang menjadi prioritas dari sebuah antrian di rumah sakit adalah tingkat keparahan pasien.

b. Pembuatan struct dalam Priority Queue

Di dalam struct yang kita buat kita akan menambahkan sebuah variable yang bertindak sebagai prioritas. Kita akan menggunakan konsep double linked list untuk membuat priority queue ini.

Contoh:

```

struct data
{
    char nama[100];
    int priority; //variable untuk menampung prioritas
    struct data *next, *prev;
}*head, *tail, *curr;

```

c. Pembuatan push dalam Priority Queue

Perbedaan push priority queue dengan queue biasa adalah pada saat data kedua dan seterusnya dipush. Disini akan terdapat beberapa validasi diantaranya:
(prioritas yang lebih kecil akan didahulukan)

1. Jika prioritas data yang akan dipush lebih kecil dari prioritas data pada head maka kita akan push data ke depan head(push depan).

2. Jika prioritas data yang akan dipush lebih besar atau sama dengan data pada tail maka kita akan push data ke belakang tail(push belakang).
3. Selain itu maka data akan di push ke tengah antrian dengan syarat data dengan prioritas yang sama akan ditaruh ke antrian paling belakang dari antrian dengan prioritas yang sama.

Contoh:

```
void push(char nama[], int priority)
{
    //alokasi memory
    curr = (struct data *)malloc(sizeof(struct data));
    //input data
    strcpy(curr->nama, nama);
    curr->priority = priority;
    curr->next = curr->prev = NULL;

    if(!head) //validasi data pertama kali dipush
        head = tail = curr;
    else
    {
        if(curr->priority < head->priority) //push depan
        {
            curr->next = head;
            head->prev = curr;
            head = curr;
        }
        else if(curr->priority >= tail->priority) //push
belakang
        {
            curr->prev = tail;
            tail->next = curr;
            tail = curr;
        }
        else //push tengah
        {
            struct data *temp;
            temp = head;


            while(temp->priority <= curr-
>priority) //looping untuk mencari posisis antrian
                temp = temp->next;

            curr->prev = temp->prev;
            temp->prev->next = curr;
            curr->next = temp;
            temp->prev = curr;
        }
    }
}
```

Contoh pemanggilan fungsi push di void main

```
push("William", 2);
push("Andy", 3);
push("Nico", 1);
push("Ali", 2);
```

Urutan hasil antrian yang benar adalah:
Nico, William, Ali, Andy

Data Structure	
Module 17 – Array Stack	
Last Update 07-10-2010	Revision 01

1. Module Description

Modul ini akan menjelaskan tentang Array Stack dan implementasinya.

2. Learning Outcomes

- Implementasi Stack dengan menggunakan array of struct

3. Material

a. Array of Struct

Jika sebelumnya kita telah mengetahui adanya array of character (string), maka struct pun dapat dibuat array. Cara pendefinisian sama, hanya pada saat mendefinisikan variabel struct perhatikan perbedaan antara variabel record (variabel yang ada dalam struct) dengan variabel struct, untuk variabel struct yang ingin dijadikan array kita tinggal menambahkan indeks seperti halnya mendefinisikan suatu array.

Contoh :

```
struct {
    int hari;
    int bulan;
    int tahun;
} tanggal[360];
```

b. Pembuatan Stack dengan Array of Struct

Untuk pembuatan stack dengan array of struct ikuti langkah-langkah berikut:

1. Definisikan Stack dengan menggunakan suatu struct
2. Definiskan konstanta MAX_STACK untuk menyimpan maksimum isi stack
3. Elemen struct Stack adalah array data dan top untuk menandakan posisi data teratas
4. Buatlah variabel tumpuk sebagai implementasi dari struct Stack
5. Deklarasikan operasi-operasi/function di atas dan buat implementasinya
6. Contoh deklarasi MAX_STACK

```
#define MAX_STACK 10
```

7. Contoh deklarasi STACK dengan struct dan array data

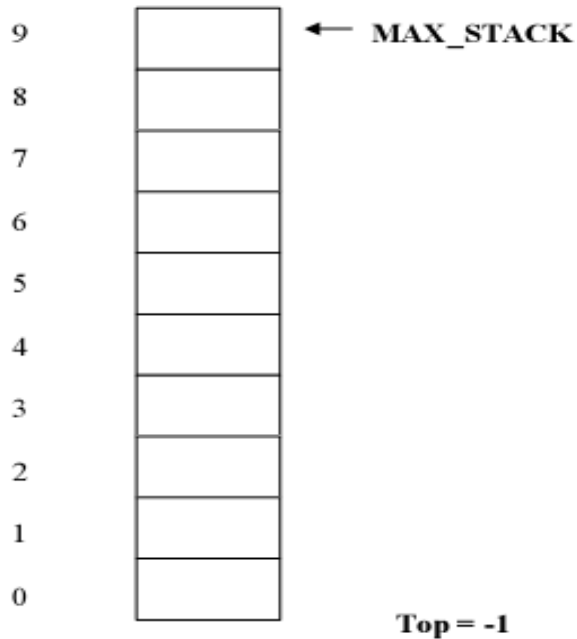
```
typedef struct STACK{
    int top;
    int data[10];
};
```

8. Deklarasi/buat variabel dari struct

```
STACK tumpuk;
```

c. Inisialisasi Stack

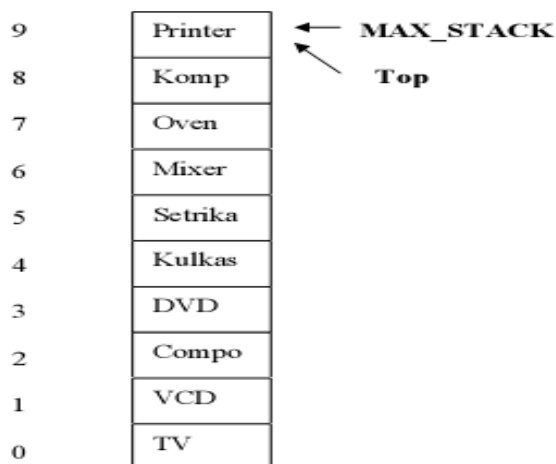
1. Pada mulanya isi top dengan -1, karena array dalam bahasa C dimulai dari 0, yang berarti bahwa data stack adalah KOSONG!
2. Top adalah suatu variabel penanda dalam Stack yang menunjukkan elemen teratas data Stack sekarang. Top Of Stack akan selalu bergerak hingga mencapai MAX of STACK yang menyebabkan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



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

d. Fungsi IsFull

3. Untuk memeriksa apakah stack sudah penuh?
4. 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



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

e. Fungsi IsEmpty

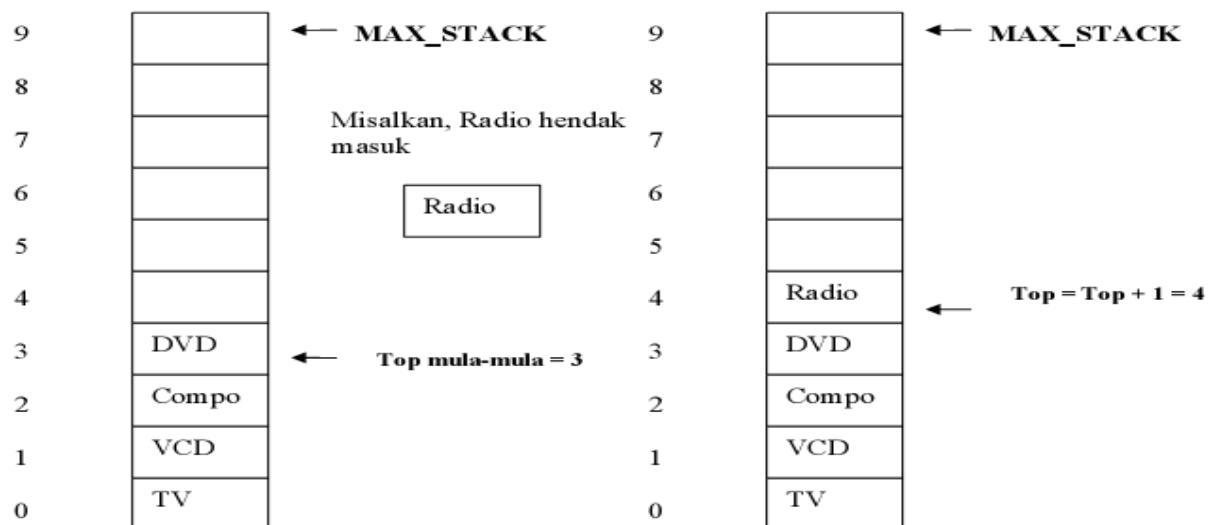
- Untuk memeriksa apakah data Stack masih kosong?

- Dengan cara memeriksa **top of stack**, jika masih -1 maka berarti data Stack masih kosong!

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

f. Fungsi Push

- Untuk memasukkan elemen ke data Stack. Data yang diinputkan **selalu** menjadi elemen teratas Stack (yang ditunjuk oleh ToS)
- Jika **data belum penuh**,
- Tambah satu (increment) nilai **top of stack** lebih dahulu setiap kali ada penambahan ke dalam array data Stack.
- Isikan data baru ke stack berdasarkan indeks top of stack yang telah di-increment sebelumnya.
- Jika tidak, outputkan "Penuh"

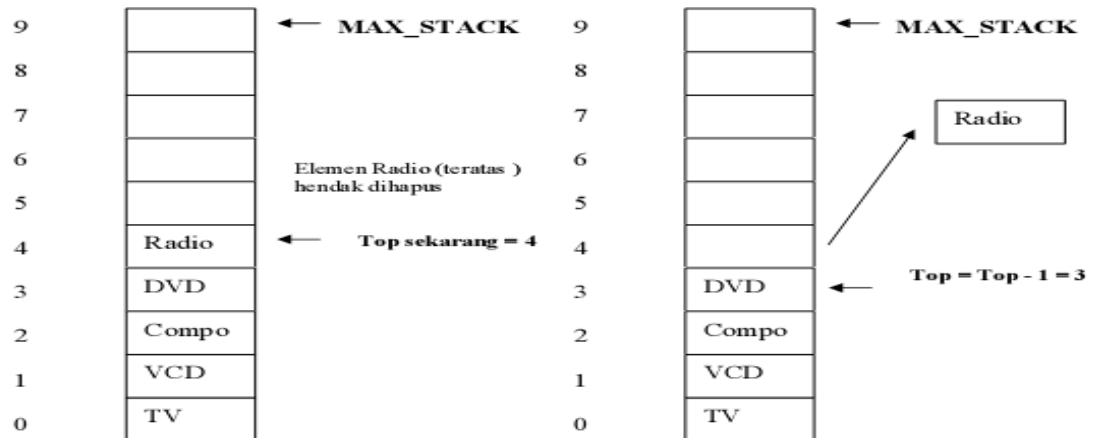


```
void Push(char d[10]){
    tumpuk.top++;
    strcpy(tumpuk.data[tumpuk.top], d);
}
```

g. Fungsi Pop

- Untuk mengambil data Stack yang terletak paling atas (data yang ditunjuk oleh TOS).

- **Tampilkan terlebih dahulu** nilai elemen teratas stack dengan mengakses indeksny sesuai dengan top of stacknya, baru dilakukan di-decrement nilai top of stacknya sehingga jumlah elemen stack berkurang.

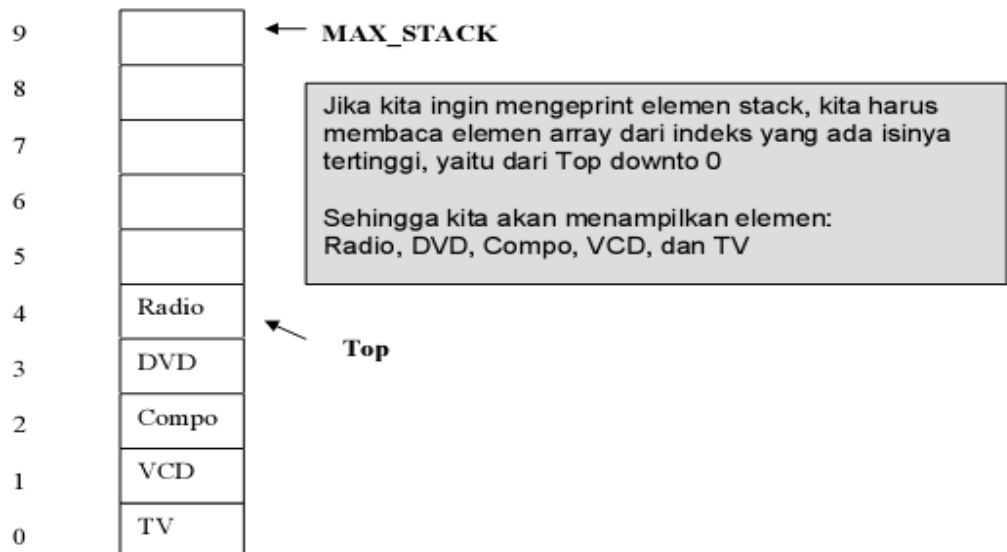


Programnya:

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

h. Fungsi Print


- Untuk menampilkan semua elemen-elemen data Stack
- Dengan cara me-loop semua nilai array secara **terbalik**, karena kita harus mengakses dari indeks array tertinggi terlebih dahulu baru ke indeks yang lebih kecil!



i. Fungsi Peek

- Digunakan untuk melihat top of stack

```
int peek() {  
    return tumpuk.data[tumpuk.top];  
}
```

Data Structure	
Module 18 – Linked List Stack	
Last Update 07-10-2010	Revision 01

1. Module Description

Modul ini akan menjelaskan tentang Linked List Stack, yang meliputi pengertian, deklarasi dan operasi-operasi yang bisa dilakukan.

2. Learning Outcomes

- Trainee memahami tentang Struktur data Linked List Stack
- Trainee dapat melakukan pembuatan dan implementasi Linked List Stack.

3. Material

Stack

Pengertian Stack :

- Stack atau tumpukan adalah suatu struktur data yang penting dalam pemrograman
- Bersifat LIFO (Last In First Out)
- Benda yang terakhir masuk ke dalam stack akan menjadi benda 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.

Selain implementasi stack dengan array seperti telah dijelaskan sebelumnya, stack dapat diimplementasikan dengan single linked list. Keunggulannya dibandingkan array adalah penggunaan alokasi memori yang dinamis sehingga menghindari pemborosan memori.

Misalnya 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.

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.

Aturan penyisipan dan penghapusan elemen tertentu dalam penggunaan **stack** :

- Penyisipan selalu dilakukan “di atas” / TOP dalam sebuah tumpukan.
- Penghapusan selalu dilakukan “di atas” / TOP dalam sebuah tumpukan.

Karena aturan penyisipan dan penghapusan semacam itu, TOP adalah satu-satunya alamat tempat terjadinya operasi. Elemen yang ditambahkan paling akhir akan menjadi elemen pertama yang akan dihapus. Dikatakan bahwa elemen stack akan tersusun secara LIFO (Last In First Out).

Ada 6 jenis operasi atau fungsi pada stack, yaitu:

- Create, digunakan untuk membuat stack baru.
- Push, digunakan untuk menambahkan elemen pada urutan terakhir.
- Pop, digunakan untuk mengambil elemen stack pada tumpukan paling atas.
- Clear, digunakan untuk mengosongkan stack.
- Print, digunakan untuk menampilkan semua elemen-elemen stack.
- IsEmpty, digunakan untuk mengecek apakah stack dalam keadaan kosong.
- IsFull, digunakan untuk memeriksa apakah stack sudah penuh..

Pendeklarasian Stack

Proses pendeklarasian stack adalah proses pembuatan struktur stack dalam memori. stack dapat direpresentasikan menggunakan array maka suatu stack memiliki beberapa bagian yaitu:

- Top yang menunjuk posisi data terakhir
- Elemen yang berisi data yang ada dalam stack. Bagian inilah yang berbentuk array.
- Maks_elemen yaitu variabel yang menunjuk maksimal banyaknya elemen dalam stack.

Operasi stack secara lengkapnya :

a. Operasi Push

Operasi push adalah operasi dasar dari stack. Operasi ini berguna untuk menambah suatu elemen data baru pada stack dan disimpan pada posisi top yang akan mengakibatkan posisi top akan berubah. Langkah operasi ini adalah:

- Periksa apakah stack penuh (isfull). Jika bernilai false/0 (tidak penuh) maka proses push dilaksanakan dan jika pemeriksaan ini bernilai true/1 (stack penuh), maka proses push digagalkan.
- Proses push-nya sendiri adalah dengan menambah field top dengan 1, kemudian elemen pada posisi top diisi dengan elemen data baru.

b. Operasi Pop

Operasi pop adalah salah satu operasi paling dasar dari stack. Operasi ini berguna untuk mengambil elemen terakhir (top) dan kemudian menghapus elemen tersebut sehingga posisi top akan berpindah. Operasi ini biasanya dibuat dalam bentuk function yang me-return-kan nilai sesuai data yang ada di top. Langkah operasi pop pada stack yang menggunakan array adalah terlebih dahulu memeriksa apakah stack sedang keadaan kosong, jika tidak kosong maka data diambil pada posisi yang ditunjuk oleh posisi top, kemudian simpan dalam variable baru dengan nama data, kemudian posisi top -1, kemudian nilai pada variable data di-return-kan ke function.

c. Operasi IsEmpty

Operasi ini digunakan untuk memeriksa apakah stack dalam keadaan kosong. Operasi ini penting dilakukan dalam proses pop. Ketika suatu stack dalam keadaan kosong, maka proses pop tidak bisa dilakukan. Operasi ini dilakukan hanya dengan memeriksa field top. Jika top bernilai 0 (untuk elemen yang dimulai dengan index 1) atau top bernilai -1 (untuk elemen yang dimulai dengan index 0), maka berarti stack dalam keadaan empty (kosong) yang akan me-return-kan true (1) dan jika tidak berarti stack mempunyai isi dan me-return-kan nilai false (0).

d. Operasi IsFull

Operasi ini berguna untuk memeriksa keadaan stack apakah sudah penuh atau belum. Operasi ini akan menghasilkan nilai true (1) jika stack telah penuh dan akan menghasilkan nilai false (0) jika stack masih bisa ditambah. Operasi ini akan memberikan nilai true (1) jika field top sama dengan field maks_elemen (untuk array yang elemennya dimulai dari posisi 1) atau top sama dengan maks_elemen-1 (untuk array yang elemennya dimulai dari posisi 0).

Contoh potongan coding fungsi pushDepan(), popBelakang(), dan popAll() di **stack dengan linked-list**:

```
struct data
{
    char name[51];
    char item[51];
    int price;
    struct data *next;
}*head, *tail, *curr;

void pushDepan(char name[], char item[], int price)
{
    curr = (struct data*)malloc(sizeof(struct data));

    strcpy(curr->name,name);
    strcpy(curr->item,item);
    curr->price = price;

    if(head == NULL)
    {
        head = tail = curr;
    }
    else
    {
        tail->next = curr;
        tail = curr;
    }
    tail->next = NULL;
}
```

```


void popBelakang()
{
    curr = head;
    if(head != tail)
    {
        while(curr->next != tail)
        {
            curr = curr->next;
        }
        printf("\n\n\n --- %s Has Been Served ---\n",curr->name);
        tail = curr;
        free(curr->next);
        tail->next = NULL;
    }
    else
    {
        printf("\n\n\n --- %s Has Been Served ---\n",curr->name);
        free(tail);
        head = tail = NULL;
    }
}

```

```

void popAll()
{
    while(head!=NULL)
    {
        curr=head;
        head=head->next;
        free(curr);
    }
}

```

Data Structure	
Module 19 – Prefix, Infix, Postfix	
Last Update 07-10-2010	Revision 01

1. Module Description

Prefix, infix, postfix merupakan notasi yang melibatkan operator dan operand.

2. Learning Outcomes

- Trainee memahami tentang Struktur data Prefix, Infix, Postfix
- Trainee dapat melakukan pembuatan dan implementasi Prefix, Infix, Postfix.

3. Material

a. PREFIX, INFIX, POSTFIX

Suatu perhitungan aritmatika biasanya berhubungan dengan operand dan operator. Operand merupakan suatu karakter atau elemen yang nilainya dioperasikan dengan bantuan suatu operator untuk menghasilkan suatu solusi. Misalkan jika diberikan suatu ekspresi aritmatika $2*3$, maka elemen 'dua' dan elemen 'tiga' merupakan operand dari ekspresi tersebut dan elemen '*' merupakan operator perkalian atas dua operand yang menghasilkan suatu solusi. Suatu ekspresi aritmatika dapat dibedakan dalam tiga bentuk notasi perhitungan yaitu :

1. Notasi prefix, jika operator ditempatkan sebelum dua operand
2. Notasi infix, jika operator ditempatkan diantara dua operand
3. Notasi postfix, jika operator ditempatkan setelah dua operand

Dalam penggunaannya di kehidupan sehari-hari, notasi infix merupakan notasi aritmatika yang paling banyak digunakan untuk mengekspresikan suatu perhitungan aritmatik dibanding dengan dua notasi yang lain, akan tetapi notasi postfix merupakan notasi yang digunakan oleh mesin kompilasi pada komputer dengan maksud untuk mempermudah proses pengkodean, sehingga mesin kompilasi membutuhkan stack untuk proses translasi ekspresi tersebut. Berdasarkan teori yang diterangkan tersebut di atas, proses konversi infix menjadi notasi postfix dalam implementasinya membutuhkan stack pada proses konversinya, adapun proses tersebut memiliki 4 (empat) aturan yang digunakan, yaitu :

1. Jika ditemukan simbol kurung buka "(", maka operasi push pada stack akan digunakan untuk menyimpan simbol tersebut ke dalam stack.
2. Jika ditemukan simbol kurung buka ")", operasi pop digunakan untuk mengeluarkan operator-operator yang berada di dalam stack.
3. Jika terdapat simbol operator, maka operasi yang dilakukan pada stack terbagi atas:
 - a. Jika TOP(S) dari stack tersebut kosong atau berisi simbol "(" maka operasi push akan digunakan untuk memasukkan operator tersebut pada posisi di TOP(S).
 - b. Jika operator yang berada dipuncak stack merupakan elemen yang memiliki tingkat yang sama atau lebih tinggi maka operasi pop digunakan untuk mengeluarkan operator tersebut diikuti operasi push untuk menyimpan operator hasil scanning untai.

- c. Jika operator yang berada di puncak stack memiliki tingkat yang lebih rendah dari operator yang discan, maka operator baru akan langsung dimasukan ke dalam stack dengan operasi push. Adapun tingkatan operator yang dilacak menurut urutan tingkat adalah:
 pangkat (^) atau akar -> Tinggi
 kali (*) atau bagi (/) -> Menengah
 tambah (+) atau kurang (-) -> Rendah

Notasi Prefix

Seorang ahli matematika “Jan Lukasiewicz” mengembangkan satu cara penulisan ungkapan numeris yang selanjutnya disebut “Notasi Polish” atau “Notasi Prefix” yang artinya: operator ditulis sebelum kedua operand yang akan disajikan. Contoh notasi prefix dari notasi infix:

Infix	Prefix
$A + B$	$+ A B$
$A + B - C$	$- + A B C$
$(A + B) * (C - D)$	$* + A B - C D$

Notasi Infix

Salah satu pemanfaatan tumpukan adalah untuk menulis ungkapan menggunakan notasi tertentu. Dalam penulisan ungkapan khususnya ungkapan numeris, kita selalu menggunakan tanda kurung untuk mengelompokkan bagian mana yang harus dikerjakan terlebih dahulu. Contoh :

1. $(A + B) * (C - D)$ Suku $(A + B)$ akan dikerjakan lebih dahulu, kemudian suku $(C - D)$ dan terakhir mengalikan hasil yang diperoleh dari 2 suku ini.
2. $A + B * C - D$ Maka $B * C$ akan dikerjakan lebih dahulu diikuti yang lain. Dalam hal ini pemakaian tanda kurung akan sangat mempengaruhi hasil akhir. Cara penulisan ungkapan sering disebut dengan “Notasi Infix” artinya operator ditulis diantara 2 operand.

Notasi Postfix

Notasi lain yang merupakan kebalikan notasi prefix adalah “Notasi Postfix” atau lebih dikenal dengan Notasi Polish Terbalik (Reverse Polish Notation atau RPN). Dalam hal ini operator ditulis sesudah operand. Sama halnya dengan notasi prefix disini juga diperlukan tanda kurung pengelompokan.

Metode Infix Ke Postfix

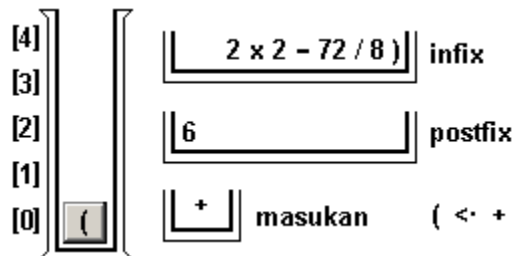
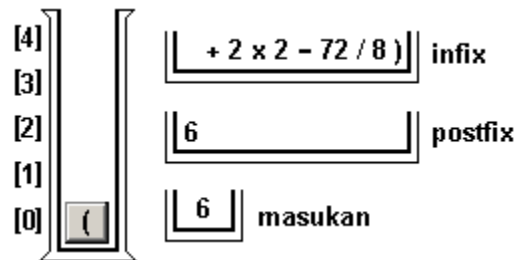
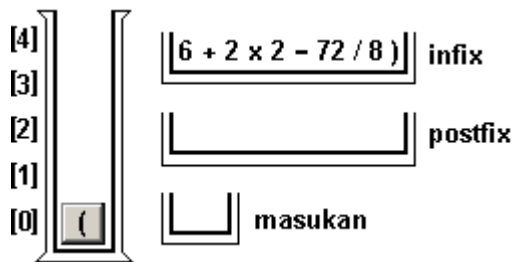
1. push operator '(' ke **stack operator**
2. berikan operator ')' di akhir **operasi**
3. pop nilai (bisa operator atau operand) dari depan **operasi**
4. jika operand, langsung tambahkan ke belakang **output**
5. jika operator, bandingkan dengan nilai **stack operator** terakhir:
 - a. jika $[\text{operator operasi}] > [\text{nilai stack operator terakhir}]$
push $[\text{operator operasi}]$ ke dalam **stack operator**
 - b. jika $[\text{operator operasi}] \leq [\text{nilai stack operator terakhir}]$
pop $[\text{nilai stack operator terakhir}]$ dan tampilkan ke output
ulangi point 5 hingga terjadi point 5a
6. ulangi point 3

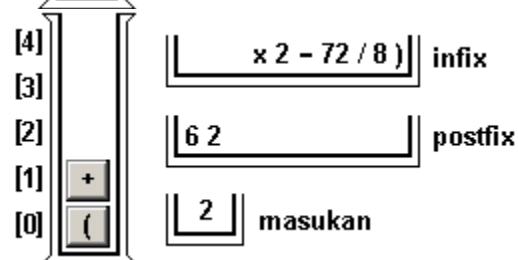
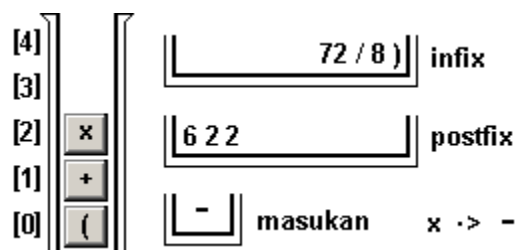
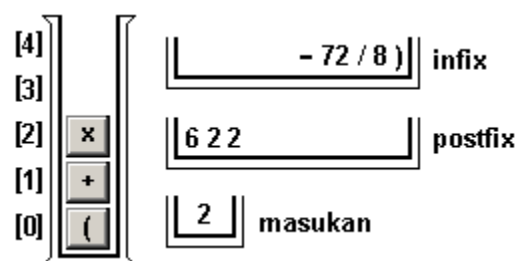
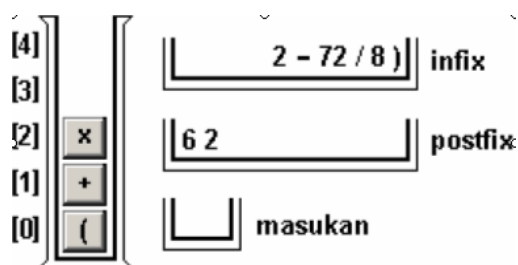
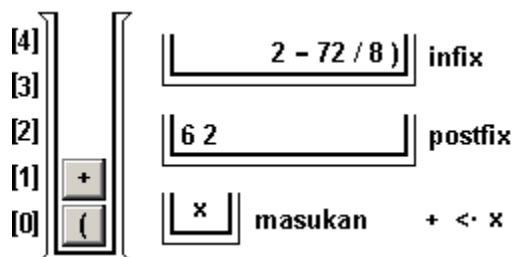
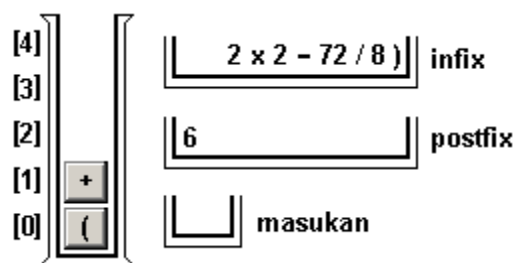
Metode Infix Ke Prefix

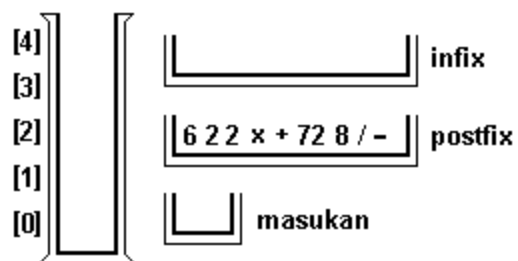
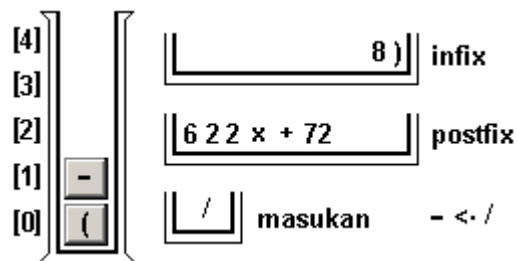
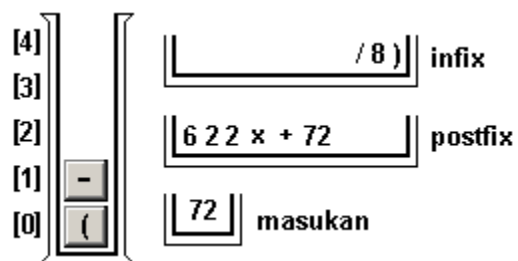
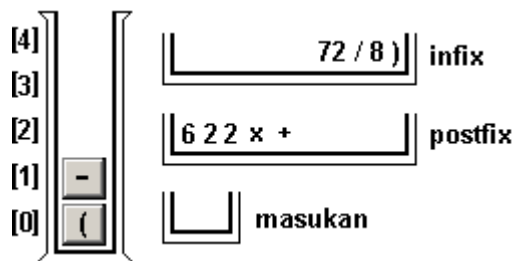
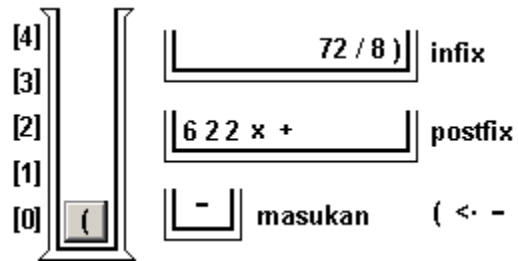
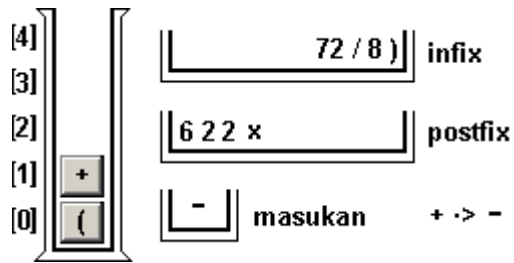
1. push operator ')' ke **stack operator**
2. berikan operator '(' di awal **operasi**
3. pop nilai (bisa operator atau operand) dari belakang **operasi**
4. jika operand, langsung tambahkan ke depan **output**
5. jika operator, bandingkan dengan nilai **stack operator** terakhir:
 - a. jika [operator operasi] \geq [nilai stack operator terakhir]
push [operator operasi] ke dalam **stack operator**
 - b. jika [operator operasi] $<$ [nilai stack operator terakhir]
pop [nilai stack operator terakhir] dan tampilkan ke output
ulangi point 5 hingga terjadi point 5a
6. ulangi point 3

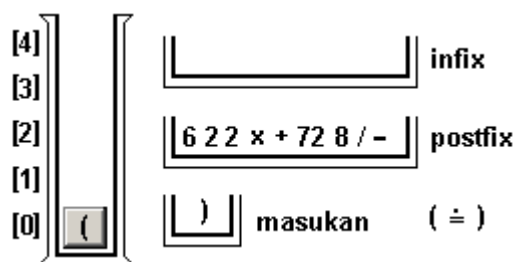
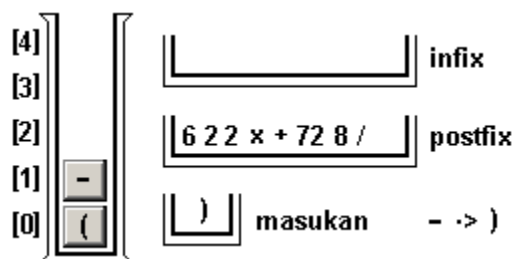
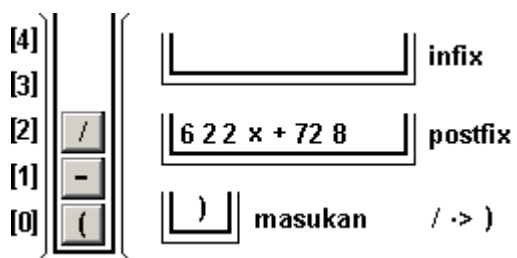
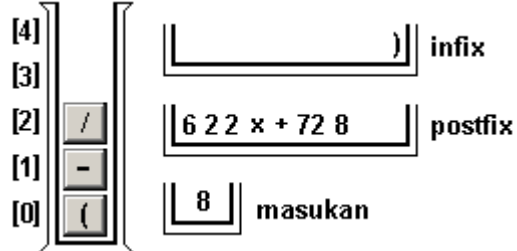
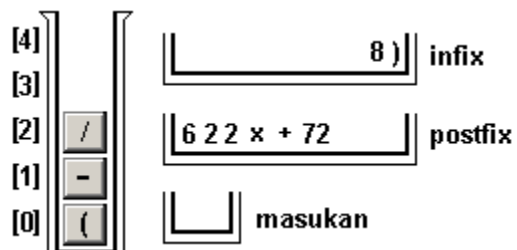
Simulasi Konversi dan Evaluasi

Contoh konversi infix ke postfix :









Hasil Akhir :

$$\begin{array}{c}
 6 + 2 \times 2 - 72 / 8 \\
 \downarrow \\
 6 \ 2 \ 2 \ x \ + \ 72 \ 8 \ / \ -
 \end{array}$$

Contoh Potongan coding untuk **Infix ke Prefix** dan **Infix ke Postfix** :

```
void infixToPrefix(char not[])
{
    int i, j;
    int value;
    index1 = 0;
    index2 = 0;

    printf("\n\n\n");

    for(i=strlen(not)-1;i>=0;i--)
    {
        if(not[i]>='0' && not[i]<='9')
        {
            value =(int)not[i]-'0' ;
            push1((int)value,1);
        }
        else
        {
            if(not[i]=='+' || not[i]=='-')
            {
                for(j=index2-1;j>=0;j--)
                {
                    if(temp2[j]=='*' || temp2[j]=='/')
                    {
                        push1((int)temp2[j],0);
                        index2--;
                    }
                }
                push2(not[i]);
            }
            else
            {
                push2(not[i]);
            }
        }
    }
}
```

```

    }
}

for(i=index2-1;i>=0;i--)
{
    push1(temp2[i],0);
    index2--;
}

printf("\n Prefix  :");
for(i=index1-1;i>=0;i--)
{
    if(info[i]==1)
    {
        printf(" %d", (int)temp1[i]);
    }
    else
    {
        printf(" %c",temp1[i]);
    }
}
}

```

```

void infixToPostfix(char not[])
{
    int i, j;
    int value;
    index1 = 0;
    index2 = 0;

    printf("\n\n\n");

    for(i=0;i<strlen(not);i++)
    {
        if(not[i]>='0' && not[i]<='9')
        {
            value = (int)not[i]-'0';
            push1((int)value,1);
        }
        else
        {
            if(not[i]=='+' || not[i]=='-')
            {
                for(j=index2-1;j>=0;j--)
                {
                    push1((int)temp2[j],0);
                    index2--;
                }
                push2(not[i]);
            }
            else
            {
                for(j=index2-1;j>=0;j--)
                {
                    if(temp2[j]=='*' || temp2[j]=='/')
                    {

```


```

        push1((int)temp2[j],0);
        index2--;
    }
    push2(not[i]);
}
}
}

for(i=index2-1;i>=0;i--)
{
    push1(temp2[i],0);
    index2--;
}

printf("\n Postfix :");
for(i=0;i<index1;i++)
{
    if(info[i]==1)
    {
        printf(" %d",(int)temp1[i]);
    }
    else
    {
        printf(" %c",temp1[i]);
    }
}
}

```

Data Structure	
Module 20 - Binary Tree	
Last Update 07-10-2010	Revision 02

1. Module Description

Modul ini menjelaskan tentang Binary Tree, konsep Binary Tree dan implementasinya dalam Pemrograman.

2. Learning Outcomes

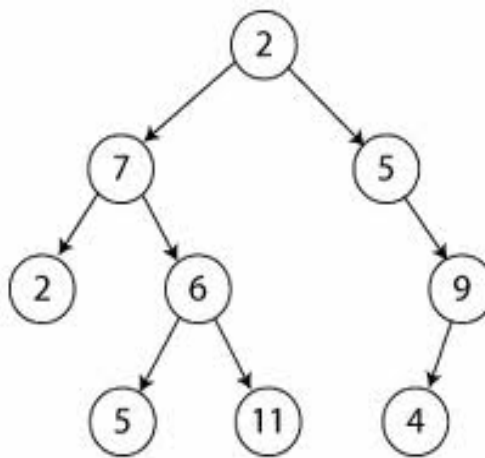
- Trainee memahami tentang Struktur data Binary Tree
- Trainee dapat melakukan pembuatan dan implementasi Binary Tree.

3. Material

a. Binary Tree

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hierarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul atau node dengan elemen khusus yang disebut **root**. Node lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lain (**subtree**).

Salah satu jenis Tree yang biasa dipakai dalam pemrograman adalah **Binary Tree**. Binary Tree adalah struktur tree (pohon) yang memiliki maksimal **2** (dua) node anak. Secara khusus anaknya dinamakan **left** (kiri) dan **right** (kanan). Di bawah ini adalah contoh sebuah Binary Tree sederhana dengan lebar 9 dan tinggi 3, dengan sebuah **root** yang memiliki nilai 2.



Gambar 9.1 Binary Tree

Dalam Binary Tree terdapat aturan - aturan yang perlu diikuti yaitu :

1. Deklarasi Tree

Karena tree tersusun oleh node-node, maka yang perlu kita deklarasikan adalah komponen

2. Inisialisasi Tree

Untuk pertama kali, saat kita akan membuat sebuah pohon biner, asumsi awal adalah pohon itu belum bertumbuh, belum memiliki node sama sekali, sehingga masih kosong. Oleh karena itu perlu kita tambahkan kode berikut pada baris awal fungsi Main:

Kita mendeklarasikan sebuah pointer yang akan menunjuk ke akar pohon yang kita buat, dengan nama `*root`. Pointer ini ditujukan untuk menunjuk struktur bertipe `Node`, yang telah dibuat pada bagian 1. Karena pohon tersebut sama sekali belum memiliki node, maka pointer `*root` ditunjukkan ke `NULL`.

3. Menambahkan Node Pada Tree

Karena pohon yang kita buat merupakan sebuah Binary Tree, maka untuk menambahkan sebuah node, secara otomatis penambahan tersebut mengikuti aturan penambahan node pada Binary Tree:

- Jika pohon kosong, maka node baru ditempatkan sebagai akar pohon.
- Jika pohon tidak kosong, maka dimulai dari node akar, user memilih menempatkan di kiri atau kanan pohon.

Contoh coding Binary Tree menggunakan double pointer (penempatan setiap node baru pada program ini berdasarkan input user disebelah kiri atau kanan) :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct data
{
    int number, level;
    struct data *left, *right; //pointer untuk menunjuk child
}*root;
```

```

void push(struct data **node, int number, int level)
{
    char direction[10];

    if(level<4) //cek apakah level tree masih lebih kecil dari 4
    {
        if((*node)==NULL) //cek jika bertemu node kosong
        {
            (*node)=(struct data*)malloc(sizeof(struct data));
            (*node)->number=number;
            (*node)->left=(*node)->right=NULL;
            printf("Success add data");
        }
        else //jika node yang ditempati tidak kosong
        {
            do
            {
                printf("Select the direction [left/right] : ");
                //minta input user di mana node akan ditempatkan
                gets(direction);
            }while(!strcmpi(direction,"left") &&!strcmpi(direction,"right"));

            if(strcmpi(direction,"left")
                push(&(*node)->left,number,level+1);
            else
                push(&(*node)->right,number,level+1);
        }
    }
    else //jika level tree sudah mencapai 4
        printf("Level is full");


    getchar();
}

```

Cara pemanggilan function di void main (untuk push angka 33) :

```
push(&root,33,0);
```

Angka 0 di parameter ketiga berarti level di root adalah 0.

Data Structure	
Module 21 – Binary Search Tree	
Last Update 07-10-2010	Revision 01

1. Module Description

Modul ini merupakan materi lanjutan dari Binary Tree, yaitu Binary Search Tree, sesuai dengan namanya BST digunakan untuk mencari suatu data.

2. Learning Outcomes

- Trainee memahami tentang Struktur data Binary Search Tree
- Trainee dapat melakukan pembuatan dan implementasi Binary Search Tree.

3. Material

a. Pendahuluan

Binary Search Tree adalah Binary Tree dimana nilai yang lebih kecil dari node parent akan ditempatkan sebagai node child kiri dan nilai yang lebih besar dari node parent akan ditempatkan sebagai node child kanan dari node parent.

b. Menambahkan Node pada Tree

Karena pohon yang kita buat merupakan sebuah Binary Search Tree, maka untuk menambahkan sebuah node, secara otomatis penambahan tersebut mengikuti aturan penambahan node pada pohon biner:

1. Jika **pohon kosong**, maka node baru ditempatkan sebagai akar pohon.
2. Jika **pohon tidak kosong**, maka dimulai dari node akar, dilakukan proses pengecekan berikut:
 - Jika nilai node baru **lebih kecil** dari nilai node yang sedang dicek, maka lihat ke kiri node tersebut. Jika kiri node tersebut kosong (belum memiliki kiri), maka node baru menjadi kiri node yang sedang dicek. Seandainya kiri node sudah terisi, lakukan kembali pengecekan a dan b terhadap node kiri tersebut. Pengecekan ini dilakukan seterusnya hingga node baru dapat ditempatkan.
 - Jika nilai node baru **lebih besar** dari nilai node yang sedang dicek, maka lihat ke kanan node tersebut. Jika kanan node tersebut kosong (belum memiliki kanan), maka node baru menjadi kanan node yang sedang dicek. Seandainya kanan node sudah terisi, lakukan kembali pengecekan a dan b terhadap node kanan tersebut. Pengecekan ini dilakukan seterusnya hingga node baru dapat ditempatkan.

Fungsi untuk mempush nilai ke dalam tree ada dua macam antara lain dengan menggunakan single pointer dan double pointer.

1. BST dengan Single Pointer

```
void push(struct data *temp, int number, int level)
{
    if(level<4) //cek apakah level tree masih lebih kecil dari 4
    {
        curr = (struct data*)malloc(sizeof(struct data));
        curr->number=number;
        curr->level=level;
        curr->left=curr->right=NULL;

        if(root==NULL) //cek jika bertemu node kosong
            root=curr;
        else if(number<temp->number) //cek jika angka lebih kecil
        {
            if(temp->left==NULL)
                temp->left=curr;
            else
                push(temp->left,number,level+1);
        }
        else if(number>temp->number) //cek jika angka lebih besar
        {
            if(temp->right==NULL)
                temp->right=curr;
            else
                push(temp->right,number,level+1);
        }
        else
            free(curr); //melepaskan memori
    }
    else //jika level tree sudah mencapai 4
        printf("Level is full");

    getchar();
}
```

Cara pemanggilan function di void main (untuk push angka 4) :

```
push(root,4,0);
```

Angka 0 di parameter ketiga berarti level di root adalah 0.

2. BST dengan Double Pointer

```
void push(struct data **node, int number, int level)
{
    if ((*node) == NULL) //cek jika bertemu node kosong
    {
        (*node) = (struct data *) malloc(sizeof(struct data));
        (*node) -> number = number;
        (*node) -> level = level;
        (*node) -> left = (*node) -> right = NULL;
    }
    else
    {
        if (level > 4) //cek apakah level tree masih lebih kecil dari 4
        {
            printf("Level is full");
            getchar();
        }
        else if ((*node) -> number == number) //cek jika angka sudah ada
        {
            printf("Number already exist");
            getchar();
        }
        else if ((*node) -> number > number) //cek jika angka lebih besar
        {
            push(&(*node) -> left, number, level + 1);
        }
        else if ((*node) -> number < number) //cek jika angka lebih kecil
        {
            push(&(*node) -> right, number, level + 1);
        }
    }
}
```

Cara pemanggilan function di void main (untuk push angka 4) :

```
push(&root, 4, 0);
```

Angka 0 di parameter ketiga berarti level di root adalah 0.

c. Operasi Transversal (Penjelajahan Tree)

Untuk membaca dan menampilkan seluruh node yang terdapat pada pohon biner, terdapat 3 macam cara, atau yang biasa disebut operasi transversal (penjelajahan tree). Semua kunjungan diawali dengan mengunjungi akar pohon. Karena proses kunjungan ini memerlukan perulangan proses yang sama namun untuk depth (kedalaman) yang berbeda, maka ketiganya diimplementasikan dengan fungsi rekursif.

1. **Kunjungan Pre-Order** (Root – Left – Right)

Kunjungan pre-order dilakukan mulai dari akar pohon, dengan urutan:

- Cetak isi (data) node yang sedang dikunjungi
- Kunjungi kiri node tersebut
- Jika kiri bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kiri tersebut.
- Jika kiri kosong (NULL), lanjut ke langkah ketiga.
- Kunjungi kanan node tersebut
- Jika kanan bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kanan tersebut.
- Jika kanan kosong (NULL), proses untuk node ini selesai, tuntaskan proses yang sama untuk node yang dikunjungi sebelumnya.

2. **Kunjungan In-Order** (Left – Root – Right)

Kunjungan in-order dilakukan mulai dari akar pohon, dengan urutan:

- Kunjungi kiri node tersebut
- Jika kiri bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kiri tersebut.
- Jika kiri kosong (NULL), lanjut ke langkah kedua.
- Cetak isi (data) node yang sedang dikunjungi
- Kunjungi kanan node tersebut
- Jika kanan bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kanan tersebut.
- Jika kanan kosong (NULL), proses untuk node ini selesai, tuntaskan proses yang sama untuk node yang dikunjungi sebelumnya.

3. **Kunjungan Post-Order** (Left – Right – Root)

Kunjungan post-order dilakukan mulai dari akar pohon, dengan urutan:

- Kunjungi kiri node tersebut
- Jika kiri bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kiri tersebut.
- Jika kiri kosong (NULL), lanjut ke langkah kedua.
- Kunjungi kanan node tersebut
- Jika kanan bukan kosong (tidak NULL) mulai lagi dari langkah pertama, terapkan untuk kanan tersebut.
- Jika kanan kosong (NULL), proses untuk node ini selesai, tuntaskan proses yang sama untuk node yang dikunjungi sebelumnya.
- Cetak isi (data) node yang sedang dikunjungi

Pemakaian di sintaksnya menggunakan fungsi rekursif sebagai berikut :

```
void inorder(struct data *node)
{
    if (node != NULL) {
        inorder (node->left);
        printf ("%d ", node->angka);
        inorder (node->right);
    }
}


void preorder(struct data *node)
{
    if (node != NULL) {
        printf ("%d ", node->angka);
        preorder (node->left);
        preorder (node->right);
    }
}

void postorder(struct data *node)
{
    if (node != NULL) {
        postorder (node->left);
        postorder (node->right);
        printf ("%d ", node->angka);
    }
}
```

d. Penghapusan Tree

Penghapusan tree dapat dilakukan secara rekursif menggunakan postorder.

```
void popall(struct data **node)
{
    if ((*node) != NULL) {
        popall (&(*node)->left);
        popall (&(*node)->right);
        free ((*node));
    }
}
```

Data Structure	
Module 22 – AVL Tree	
Last Update 08-04-2011	Revision 00

1. Module Description

Pada bagian ini akan dijelaskan secara singkat mengenai apa dan bagaimana cara membuat AVL Tree

2. Learning Outcomes

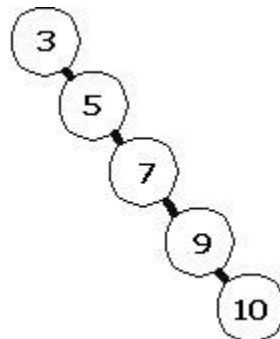
Trainee dapat menerapkan konsep penggunaan AVL Tree pada kasus nyata.

3. Material

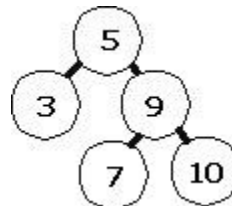
AVL Tree adalah pengembangan dari Binary Search Tree, dengan beberapa kelebihan yaitu dalam proses pencarian data akan lebih cepat dan data yang dimasukkan akan menjadi seimbang pada bagian kiri dan kanan.

Contoh:

Asumsikan data yang diinput di Binary Search Tree adalah 3, 5, 7, 9, 10 maka binary tree akan berbentuk seperti ini:



Namun saat kita ingin mencari node (misalkan yang bernilai 10), untuk mendapatkan nilai tersebut, maka kita harus melewati node bernilai 3, 5, 7, dan 9, hingga mendapatkan nilai 10. Hal ini tentu tidak efektif, oleh karena itu lebih baik menggunakan AVL Tree. Dengan AVL Tree Node yang tadi akan berubah menjadi seperti berikut:



AVL Tree mempunyai suatu teknik yang disebut rotasi, teknik rotasi ini digunakan saat memasukkan maupun menghapus data lebih dikenal dengan istilah **Left Rotation** dan **Right Rotation**.

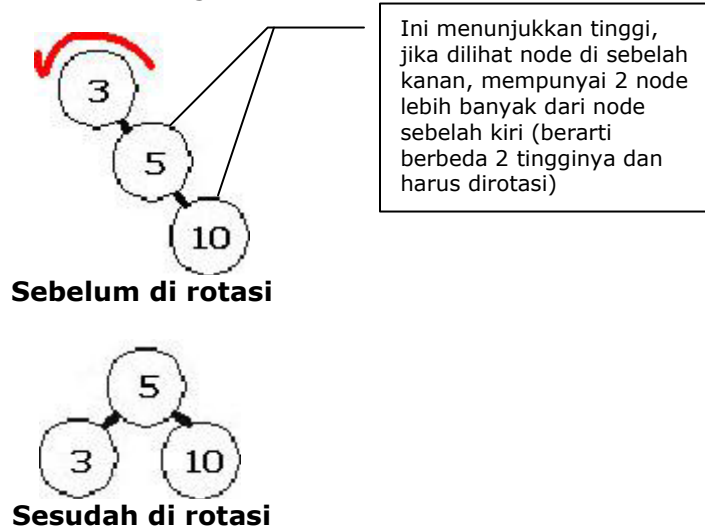
Left Rotation digunakan saat tinggi dari tree yang terdapat di sebelah kiri berbeda 2 tingkat dengan tree yang terdapat di sebelah kanan.

Sedangkan **Right Rotation** digunakan saat tinggi dari tree yang di sebelah kanan berberda 2 tingkat dengan tree yang terdapat di sebelah kiri.

Dalam beberapa kasus dapat dilakukan **Double Rotation**.

Contoh:

Node dengan nilai 3, 5, dan 10 ditambahkan sehingga tree tidak seimbang. Left Rotation dilakukan untuk membuat jumlah node seimbang.



Sekarang kita buat struct sederhana dengan keterangan sebagai berikut:

Nama Variable	Type Data	Penjelasan
value	int	Nilai dari node yang ada
height	int	Node (berfungsi juga menunjukkan tinggi)
left	struct data *	Nantinya akan menunjukkan subtree yang sebelah kiri
right	struct data *	Nantinya akan menunjukkan subtree yang sebelah kanan
parent	struct data *	Node yang berperan sebagai parent

Setiap node di AVL tree menyimpan suatu nilai dan mempunyai **subtree kanan** maupun **subtree kiri**. Node yang paling atas dikenal dengan istilah **root**, sedangkan node yang tidak mempunyai subtree sama sekali dikenal dengan istilah **leaf**

Hasil pembuatan struct dari data di atas adalah

```
struct data{
    int value, height;
    struct data *parent, *left, *right;
} *root = NULL;
```

Langkah berikutnya setelah kita membuat struct adalah membuat suatu method untuk mengembalikan nilai terbesar, contohnya adalah

```
int maxValue(int val1, int val2){
    if (val1 > val2) return val1;
    else return val2;
}
```

Kemudian kita membuat dua method lagi, yaitu **"clear"** (untuk membersihkan layar) dan **"space"** (untuk mencetak spasi)

```
void clear(){
    int i;
    for (i=0; i<25; i++) printf("\n");
}

void space (int n){
    int i;
    for (i=0; i<n; i++) printf (" ");
}
```

Kita juga membuat satu fungsi **"getHeight"** untuk mendapatkan tinggi dari node yang ada (akan mengembalikan **node height** atau **0** (jika node bernilai NULL))

```
int getHeight (struct data* curr){
    if (curr == NULL) return 0;
    return curr->height;
}
```

Kemudian kita buat fungsi **"swapParent"** yang berguna untuk menukar parent dari 2 nodes (parent dan child), child yang ditukar dapat **left child** atau **right child**.

```
//c = child, p = parent
void swapParent (struct data *c, struct data *p){
    c->parent = p->parent;
    if(p == root) root = c;
    else if (p->parent->left == p) p->parent->left = c;
    else if (p->parent->right == p) p->parent->right = c;
    p->parent = c;
}
```

Kita buat fungsi lainnya untuk melakukan rotasi ke **kanan** dan rotasi ke **kiri** dengan nama **"rotateRight"** dan **"rotateLeft"**

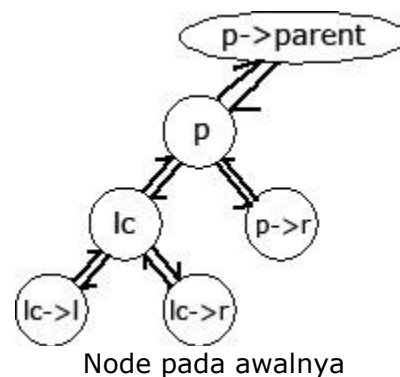

```

void rotateRight (struct data *lc, struct data *p){
    swapParent(lc, p);
    p->left = lc->right;
    if (p->left!=NULL) p->left->parent = p;
    lc->right = p;
    p->height = maxValue(getHeight(p->left), getHeight(p->right)) + 1;
    lc->height = maxValue(getHeight(lc->left), getHeight(lc->right)) + 1;
}

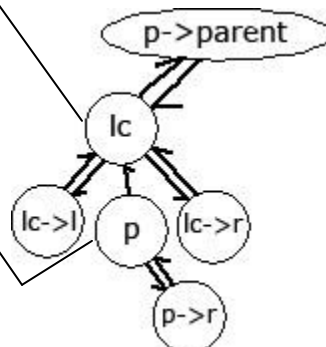
void rotateLeft (struct data *rc, struct data *p){
    swapParent(rc, p);
    p->right = rc->left;
    if (p->right!=NULL) p->right->parent = p;
    rc->left = p;
    p->height = maxValue(getHeight(p->left), getHeight(p->right)) + 1;
    rc->height = maxValue(getHeight(rc->left), getHeight(rc->right)) + 1;
}

```

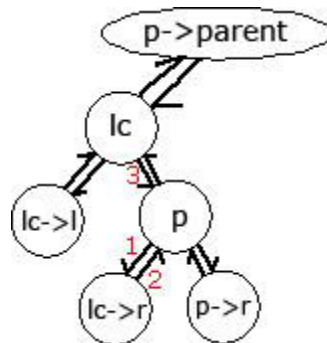
Ilustrasinya adalah sebagai berikut:



Perhatikan bahwa **lc** dan **p** sudah ditukar namun ini menyebabkan sementara **lc** mempunyai 3 kaki, ini akan diperbaiki di langkah berikutnya



p dan **lc** ditukar (menggunakan fungsi **swapParent**)



Kemudian **lc->r** dipindahkan menjadi left subtree milik **p**

Fungsi berikutnya yang tidak kalah penting adalah fungsi untuk menambahkan node baru dan mengisi nilainya, kita buat saja sebuah fungsi baru dengan nama **"newNode"**

```
struct data* newNode (int number, struct data *parent){
    //alokasi memori baru
    struct data *node = (struct data*) malloc(sizeof (struct data));

    node->value = number;
    node->height = 1;
    node->parent = parent;
    node->left = node->right = NULL;
    printf("\nberhasil insert node baru...");
    return node;
}
```

Setelah kita menambahkan node baru, hal yang harus dilakukan selanjutnya adalah menambahkan data ke dalam node baru tersebut. Hal ini kita lakukan dengan membuat fungsi baru, yang kita sebut **"insert"**.

Fungsi ini serupa dengan fungsi insert dalam binary search tree, tetapi ditambahkan fungsi untuk mengecek, apakah data yang dimasukkan adalah data yang paling awal, sehingga data tersebut akan otomatis menjadi **root**. Namun jika data yang dimasukkan bukan data awal, maka data tersebut akan **ditambahkan**, dan dicek apakah data tersebut harus dimasukkan di sebelah kanan atau kiri, dan kemudian dilakukan tindakan **penyeimbangan**, jika node tidak seimbang.

Contohnya dapat dilihat pada potongan program di bawah ini. Dalam fungsi tersebut, akan dipanggil 2 prosedur yaitu untuk **menambahkan** node baru ke dalam tree dan untuk **menyeimbangkan** tree.

```

void insert(int number){
    if(root == NULL) root = newNode (number, root);
    else{
        //tambahkan node baru ke dalam tree

        //seimbangkan tree

    }
}

```

Berikut adalah **prosedur menambahkan** node baru ke dalam tree

```

int insert = 0; //inisialisasi 0=false
struct data *curr = root;
while(!insert){
    //jika lebih kecil dari nilai yang ditunjuk, insert node ke kiri
    if(number < curr->value) {
        if(curr->left == NULL){
            curr->left = newNode(number, curr);
            insert = 1;//insert dirubah menjadi 1/bernilai true
        }
        curr = curr->left;//curr dipindahkan ke kiri
    }
    //jika lebih besar dari nilai yang ditunjuk, insert node ke kanan
    else if (number > curr->value){
        if (curr->right == NULL){
            curr->right = newNode (number, curr);
            insert = 1;//insert dirubah menjadi 1/ bernilai true
        }
        curr = curr->right;//curr dipindahkan ke kanan
    }
    //jika ada node dengan nilai yang sama
    else{
        printf("Sudah terdapat data dengan nilai yang sama\n");
        return;
    }
}
}

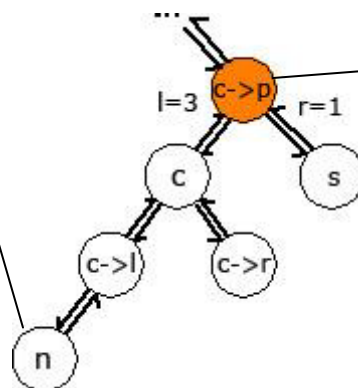
```

Berikut adalah **prosedur menyeimbangkan tree**

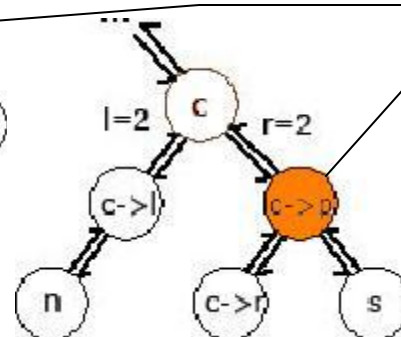
```
while (curr!=root){
    if(curr->parent->left == curr){
        int l = curr->height;
        int r = getHeight(curr->parent->right);
        curr->parent->height = maxValue(l,r)+1;

        if(l-r >1){
            if(number < curr->value) rotateRight(curr, curr->parent);
            else {
                rotateLeft(curr->right, curr);
                rotateRight(curr->parent, curr->parent->parent);
            }
            return;
        }
    }
    else if (curr->parent->right == curr){
        int l = getHeight(curr->parent->left);
        int r = curr->height;
        curr->parent->height = maxValue(l,r)+1;
        if(r-l >1){
            if(number > curr->value) rotateLeft (curr, curr->parent);
            else {
                rotateRight(curr->left, curr);
                rotateLeft(curr->parent, curr->parent->parent);
            }
            return;
        }
    }
    curr = curr->parent;
}
```

Asumsikan seperti ini, jika **n** kita input sebagai data baru, maka akan terjadi ketidakseimbangan tree



a. Saat menambahkan data



b. Saat sudah dirotasi (right)

Yang berwarna orange, karena merupakan parent dan merupakan pusat yang tidak seimbang, ini yang akan dirotasi

Buat satu fungsi untuk menampilkan struktur tree yang sudah dibuat, kita beri nama **"view"** misalnya

```
void view(struct data *curr, int level){
    space ((level -1)*2);
    if(curr==NULL){printf ("'--NULL\n"); return;}
    else printf("'--%d\n", curr->value);
    view (curr->left, level+1);
    view (curr->right, level+1);
}
```

Untuk menghapus node kita buat satu fungsi misalnya kita beri nama **"removeNode"**

```
void removeNode(int number, struct data *curr)
{
    int flag = 0;
    while(curr != NULL){
        if(number < curr->value){
            curr = curr->left;
        }
        else if(number > curr->value){
            curr = curr->right;
        }
        else{
            flag = 1;
            break;
        }
    }
    if(flag == 0 && number > 0){
        printf("node tidak ditemukan...");
        return;
    }

    //jika terdapat data di sebelah kanan atau kiri subtree
    //sebelum di remove ada langkah penukaran terlebih dahulu
    if(curr->right !=NULL && curr->left !=NULL)
        removeChange (curr,curr->parent,number);

    //selain itu, langsung saja node tersebut di remove
    else
        removeDirectly(curr,curr->parent,number);
}
```

Berikut detail isi dari fungsi **“removeChange”**

```
void removeChange(struct data *l, struct data *p,int number)
{
    struct data *ptr, *save, *suc, *psuc;
    ptr=l->right;
    save=l;
    while(ptr->left!=NULL) {
        save=ptr;
        ptr=ptr->left;
    }

    suc=ptr;
    psuc=save;
    removeDirectly(suc,psuc,number);
    if(p!=NULL)
        if(l==p->left)
            p->left=suc;
        else
            p->right=suc;
    else root=l;

    suc->left=l->left;
    suc->right=l->right;

    return;
}
```

Berikut detail isi dari fungsi **“removeDirectly”**

```
void removeDirectly(struct data *s, struct data *p, int number)
{
    struct data *child;
    if(s->left==NULL && s->right==NULL)
        child=NULL;
    else if(s->left!=NULL)
        child=s->left;
    else
        child=s->right;
    if(p!=NULL)
        if(s==p->left)
            p->left=child;
        else
            p->right=child;
    else
        root=child;
}
```

Untuk menghapus semua tree yang ada kita buat satu fungsi lagi kita beri nama misalnya **“popall”**


```
void popall(struct data *curr){
    if (curr!=NULL){
        popall(curr->left);
        popall(curr->right);
        free(curr);
    }
}
```

Setelah semua langkah selesai dikerjakan, untuk penerapan, kita hanya perlu memanggil fungsi saja

```
//untuk insert
insert([angka yang ingin ditambahkan]);
contoh: insert(12);

//untuk menampilkan
view(root, [level]);
contoh: view(root, 1);

//untuk menghapus
removeNode([angka yang ingin dihapus], root);
contoh: removeNode(12, root);
```

Data Structure	
Module 23 – Heap Tree	
Last Update 02-04-2011	Revision 00

1. Module Description

Pada bagian ini akan dijelaskan secara singkat mengenai apa dan bagaimana cara membuat Heap Tree

2. Learning Outcomes

Trainee dapat menerapkan konsep penggunaan Heap Tree pada kasus nyata.

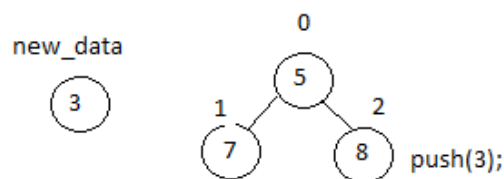
3. Material

Heap merupakan struktur data berbasis tree yang membuat pengurutan tree dalam kunci tertentu, contohnya elemen terkecil dari tree akan menjadi **root** (biasa disebut **min-heap**), atau element terbesar dari suatu tree akan menjadi **root** (biasa disebut **max-heap**).

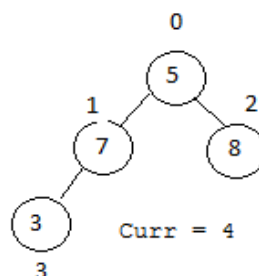
Bisa dikatakan heap merupakan pengembangan dari **priority queue**, selain itu heap juga merupakan binary tree yang dapat menggunakan array sebagai media penyimpanannya kelebihanannya yaitu kita dapat mengurangi beban memory tapi tetap mempertahankan pencarian tree yang sederhana.

Konsepnya (dalam hal ini **min-heap** yang dijadikan contoh),

Pada awalnya terdapat data 5, 7, 8 (gambar **23.1**), kemudian dilakukan push data baru yaitu 3 hasil di dalam tree akan seperti gambar **23.2**.

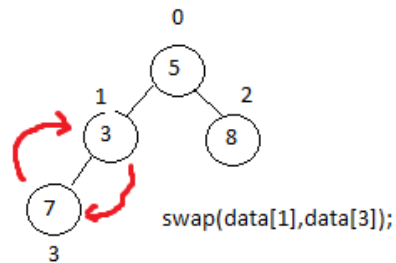


Gambar 23.1

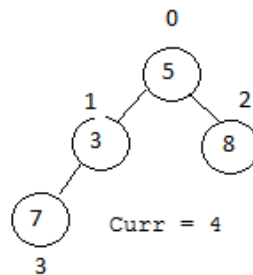


Gambar 23.2

Kemudian setelah data tersebut dimasukkan, perhatikan bahwa ada variable **curr**, yang digunakan untuk menentukan index data selanjutnya. Pada contoh, node baru yang diinput adalah node dengan nilai 3 dan karena menggunakan konsep **min-heap** (root merupakan yang paling kecil, oleh karena itu node dengan nilai 3 ini ditukar (swap) dengan nilai yang lebih besar pada node di atasnya), secara konsep proses dan hasilnya akan seperti gambar **23.3** dan **23.4**.

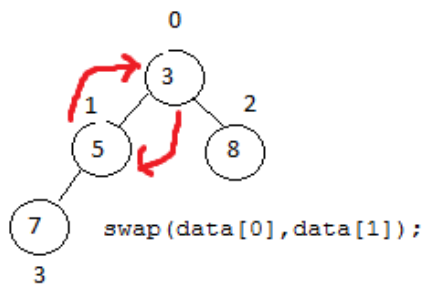


Gambar 23.3

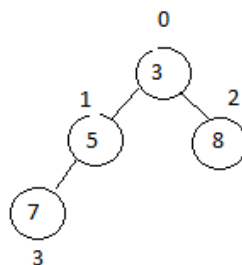


Gambar 23.4

Langkah tersebut masih belum selesai, karena dalam hal ini root bukanlah nilai terkecil dari semua node yaitu 3, oleh karena itu diperlukan satu kali proses swap lagi untuk menukar antara node yang bernilai 3 dan node yang bernilai 5 (root merupakan nilai terkecil), hasilnya akan seperti gambar **23.5** dan **23.6**.



Gambar 23.5



Gambar 23.6

Sebelum memulai, kita membuat dahulu dua buah variabel, yang berfungsi sebagai penampung dari data yang akan kita input (**DATA array**) dan sebuah variabel lagi yang berfungsi untuk menampung perhitungan index keberapa (**INDEX**) seperti berikut ini.

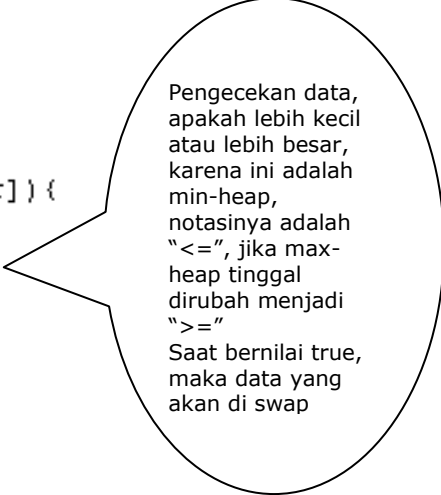
```
int DATA[100];  
int INDEX = 1;
```

Kemudian kita membuat sebuah fungsi untuk menukar array, kita dapat beri nama fungsi ini **"swap"**:

```
void swap(int *val1, int *val2){  
    int temp = *val1;  
    *val1 = *val2;  
    *val2 = temp;  
}
```

Langkah yang tidak kalah pentingnya adalah membuat fungsi untuk menambahkan node baru, karena disini kita membuat menggunakan **array**, maka fungsi untuk menambahkan data adalah sebagai berikut:

```
void push (int newData){  
    int curr = INDEX;  
    DATA[curr] = newData;  
    while (curr != 1 && DATA[curr/2] > DATA[curr]){  
        int parent = curr/2;  
        if (DATA[parent] <= DATA [curr]) break;  
        swap (&DATA[curr], &DATA[parent]);  
        curr = parent;  
    }  
    INDEX++;  
    printf("berhasil insert node baru...");  
    getchar();  
}
```



Pengecekan data, apakah lebih kecil atau lebih besar, karena ini adalah min-heap, notasinya adalah "<=", jika max-heap tinggal dirubah menjadi ">=" Saat bernilai true, maka data yang akan di swap

Setelah dilakukan penambahan data dan terjadi penukaran jika ada data yang lebih kecil, maka langkah berikutnya adalah membuat satu fungsi yang dapat digunakan untuk **menghapus** data yang sudah dimasukkan, misalnya saja kita beri nama **"pop"**

```

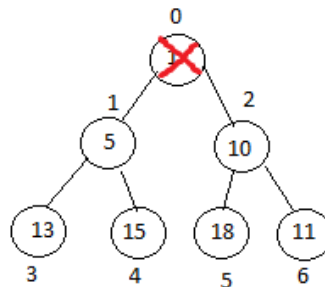
void pop(){
    if (INDEX==1){
        printf("data kosong");
        getchar();
    }
    else {
        int curr = 1, LEFT, RIGHT;
        INDEX--;
        swap(&DATA[1], &DATA[INDEX]);
        while (curr * 2 < INDEX){
            LEFT = curr * 2;
            RIGHT = curr * 2 + 1;
            if (DATA[LEFT] < DATA[curr]){
                if (RIGHT < INDEX && DATA[RIGHT] < DATA[LEFT]){
                    swap (&DATA[curr], &DATA[RIGHT]);
                    curr = RIGHT;
                }
                else {
                    swap (&DATA[curr], &DATA[LEFT]);
                    curr = LEFT;
                }
            }
            else if (RIGHT < INDEX && DATA[RIGHT] < DATA[curr]){
                swap (&DATA[curr], &DATA[RIGHT]);
                curr = RIGHT;
            }
            else break;
        }
        printf ("berhasil menghapus");
        getchar();
    }
}

```

Fungsi **"pop"** ini digunakan untuk menghapus node yang ada, dan node yang dihapus adalah selalu node paling atas **root**, kenapa? Karena biasanya heap digunakan untuk priority atau sorting, jadi otomatis data yang dihapus adalah **data yang terkecil/terbesar** terlebih dahulu.

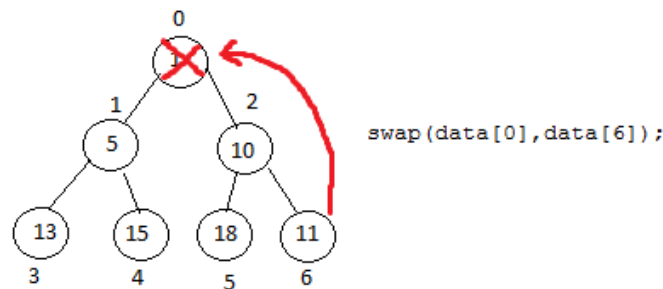
Penjelasan secara detail mengenai potongan program diatas adalah sebagai berikut:

Saat kita ingin menghapus node, maka **root** yang akan dihapus, seperti gambar **23.7**



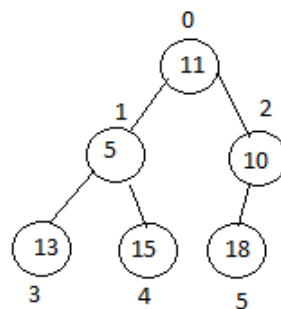
Gambar 23.7

Kemudian kita melakukan langkah **penukaran**, dimana data di index terakhir akan menjadi root menggantikan root yang dihapus sebelumnya, seperti gambar **23.8**



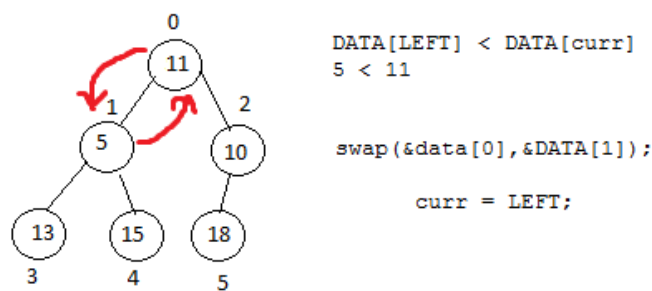
Gambar 23.8

Saat sudah ditukar, bentuk tree akan menjadi seperti gambar **23.9**



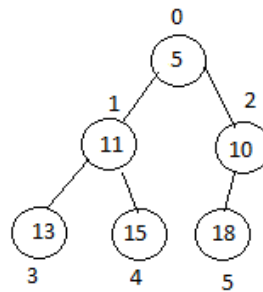
Gambar 23.9

Kemudian kita akan melakukan normalisasi terhadap tree dengan langkah – langkah seperti gambar **23.10** dan **23.11**



Gambar 23.10

Swap datanya sehingga menjadi



Gambar 23.11

Selain fungsi **"push"** untuk **menambahkan** dan fungsi **"pop"** untuk **menghapus**, kita bisa juga **mencetak** semua node yang ada di tree, misalkan saja kita buat satu fungsi dengan nama **"print"** yang isi dari fungsinya adalah sebagai berikut:

```
void print(){
    if(INDEX==1){
        printf("data kosong");
        getchar();
    }
    else{
        for(int i=1;i<INDEX;i++){
            printf("\nIndex : %d , Data : %d ", i-1, DATA[i]);
        }
        getchar();
    }
}
```

Setelah semua langkah selesai dilakukan, untuk penerapan kita hanya perlu memanggil fungsinya saja.

```
//untuk insert
push([angka yang ingin ditambahkan]);
contoh: push(12);

//untuk menampilkan
print();

//untuk menghapus (root)
pop();
```