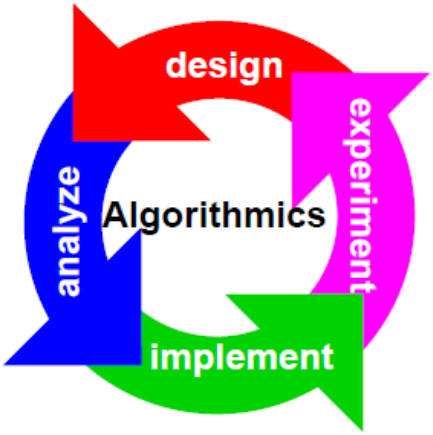
Algoritma dan Struktur Data II

Oleh : Tono Hartono, S.Si.,MT

(Digunakan di lingkungan sendiri, sebagai buku ajar mata kuliah Algoritma dan Struktur Data II)



Fakultas Teknik dan Ilmu Komputer

Program Studi Sistem Informasi

Universitas Komputer Indonesia

1. Pendahuluan

1.1 Algoritma dan Struktur Data II merupakan mata kuliah yang membahas tentang algoritma Array dan Struct, Antrian, Stack dan penerapannya dalam ekspresi matematik, Pencarian data baik yang sudah terurut maupun yang belum terurut, Pengurutan data baik secara ascending dan descending, linked list baik single dan double, binary tree dan penggunaannya dalam ekspresi matematik.

1.2 Kontrak Perkuliahan

a. Perkuliahan diselenggarakan 14 kali pertemuan (3 SKS)
b. Wajib kehadiran Mahasiswa 80% (-3 kali tidak masuk)
c. Materi perkuliahan akan diberikan salinannya kepada Mahasiswa
d. Batas keterlambatan 15 menit setelah perkuliahan dimulai
e. Mahasiswa diperbolehkan berkonsultasi dengan dosen; mengenai materi perkuliahan secara personal atau kelompok di luar jam perkuliahan (tatap muka; via email; kuliah online)
f. Mengikuti tata tertib perkuliahan
g. Tidak diperbolehkan menggunakan perangkat komunikasi selama perkuliahan (setting silent/vibrate)
h. Bersikap sopan dan tidak mengganggu keberlangsungan perkuliahan
i. Tersedia waktu Shalat bagi yang beragama Islam.

Silabus :

Pertemuan 1 : Array dan Struct

1. Definisi Array
2. Deklarasi Array
3. Penggunaan Array
4. Algoritma Array dalam Bahasa Pemrograman
5. Definisi Struct
6. Deklarasi Struct
7. Penggunaan Struct
8. Algoritma Struct dalam Bahasa Pemrograman

9. Penggunaan Struct bersamaan dengan Array
10. Algoritma kombinasi penggunaan Struct dan Array dalam Bahasa Pemrograman

Pertemuan 2 : Stack

1. Definisi Stack
2. Contoh kasus stack dalam sehari-hari
3. Algoritma Stack
4. Algoritma Stack dalam Bahasa Pemrograman

Pertemuan 3 : Stack II

1. Stack dan penerapannya dalam penyelesaian ekspresi matematik
2. Mengubah ekspresi matematik ke dalam bentuk postfix dengan metode stack
3. Menyelesaikan ekspresi matematik dalan bentuk postfix

Pertemuan 4 : Queue

1. Definisi Queue
2. Contoh kasus queue dalam sehari-hari
3. Algoritma queue
4. Algoritma queue dalam Bahasa Pemrograman

Pertemuan 5 : Sorting

1. Definisi Sorting
2. Jenis-jenis Sorting
3. Algoritma Sorting
4. Algoritma Sorting dalam Bahasa Pemrograman

Pertemuan 6 : Searching

1. Definisi Searching
2. Jenis-jenis Searching
3. Algoritma Searching pada data tidak terurut
4. Algoritma Searcging pada data terurut
5. Algoritma Searching dalam Bahasa Pemrograman

Pertemuan 7 : *Review*

Pertemuan 8 : UJIAN TENGAH SEMESTER

Pertemuan 9 : Pembahasan UTS

Pertemuan 10 : Rekursif

1. Definisi Rekursif
2. Contoh kasus penggunaan Rekursif
3. Algoritma Rekursif
4. Algoritma Rekursif dalam Bahasa Pemrograman

Pertemuan 11 : Single Linked List

1. Definisi Linked List
2. Definisi Single Linked List
3. Algoritma Single Linked List: Sisip depan, tengah dan belakang
4. Algoritma Single Linked List: Hapus depan, tengah dan belakang
5. Algoritma Single Linked List dalam Bahasa Pemrograman

Pertemuan 12 : Double Linked List

1. Definisi Double Linked List
2. Algoritma Double Linked List: Sisip depan, tengah, dan belakang
3. Algoritma Single Linked List: Hapus depan, tengah, dan belakang
4. Algoritma Double Linked List dalam Bahasa Pemrograman

Pertemuan 13 : Binary Tree : Penelusuran

1. Definisi Binary Tree
2. Penelusuran Binary Tree: Preorder
3. Penelusuran Binary Tree: Inorder
4. Penelusuran Binary Tree: Postorder

Pertemuan 14 : Binary Tree : Penciptaan

1. Level operator matematik
2. Penciptaan binary tree dari ekspresi matematik
3. Penelusuran hasil penciptaan binary tree secara preorder dan postorder

Pertemuan 15 : *Review*

Pertemuan 16 : UJIAN AKHIR SEMESTER

Pertemuan 1 : Array dan Struct

1.1 Definisi Array

Array adalah kumpulan dari nilai-nilai data bertipe sama dalam urutan tertentu yang menggunakan sebuah nama yang sama Nilai-nilai data

disuatu array disebut dengan elemen-elemen array Letak urutan dari elemen-elemen array di tunjukkan oleh suatu subscript atau indek.

1.2 Deklarasi Array

– Suatu array berdimensi satu dideklarasikan dalam bentuk umum berupa :

Tipe_data nama_var[ukuran];

– Tipe_data : untuk menyatakan tipe dari elemen arrat, misalnya int, char, float.

Nama_var: nama variable array

Ukuran : untuk menyatakan jumlah maksimal elemen array.

Contoh pendeklarasikan array :

Float nilai [5];

Menyatakan bahwa variable nilai bertipe array of float dan memiliki 5 elemen bertipe float.

1.3 Penggunaan Array

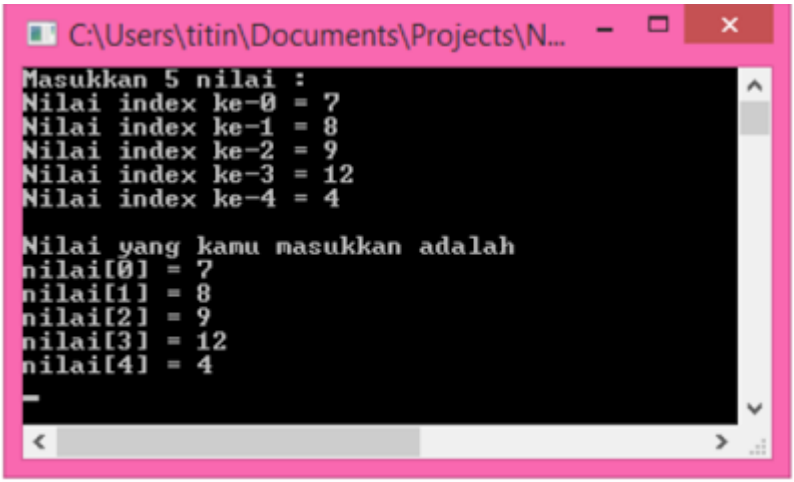
```
#include <iostream>
#include <conio.h>

using namespace std;

main()
{
    int nilai[5]; //deklarasi array
    int i; //deklarasi variabel untuk perulangan

    //script untuk menginputkan nilai ke dalam array
    cout<<"Masukkan 5 nilai : \n";
    for(i=0;i<5;i++){
        cout<<"Nilai index ke-"<<i<<" = ";
        cin>>nilai[i];
    }

    //script untuk menampilkan nilai yang sudah diinputkan
    cout<<"\nNilai yang kamu masukkan adalah \n";
    for(i=0;i<5;i++){
        cout<<"nilai["<<i<<"] = "<<nilai[i]<<endl;
    }
    getch();
}
```



1.4 Algoritma Array dalam Pemrograman

Array Satu Dimensi

Array satu dimensi adalah array yang hanya berisi satu dimensi saja.

a. Mendeklarasikan varibel Array

Pendeklarasian variabel array dilakukan dengan bentuk:

tipe namaVariabelArray[];

atau

tipe [] namaVariabelArray;

Contoh :

int [] umur;

Sampai disini kita hanya mendeklarasikan variabel bertipe array.

Artinya,

jumlah elemen dan nilai-nilai dari setiap elemennya masih di set dengan

nilai null.

Untuk mendeklarasikan array sekaligus memberikan nilai dapat dilakukan dengan cara sebagai berikut :

```
double []grades = {100, 90, 80, 75};

String days[] = { “Mon”, “Tue”, “Wed”, “Thu”, “Fri”, “Sat”,“Sun”};
```

b. Mengakses Elemen Array

Elemen array dapat diakses melalui bentuk:

namaVariabelArray[index].

Contoh :

```
class cobaArray {
   public static void main(String [] args){
   int [] jumlahHari;
   jumlahHari = new int[4];
   jumlahHari[0] = 31;
   jumlahHari[1] = 28;
   jumlahHari[2] = 31;
   jumlahHari[3] = 30;

   System.out.println("Bulan Maret memiliki " + jumlahHari[2] + " hari.");
```

```
    }
  }
```

2. Array Multi dimensi / Dua Dimensi

Array multi-dimensi merupakan array dari array. Artinya, terdapat sebuah array yang setiap elemennya juga bertipe array.

Cara mendeklarasikan Array Multi Dimensi :

```
tipe namaVariabelArray [][];

atau

tipe  [][] namaVariabelArray;
```

Contoh :

```
int[][] twoD = new int[512][128];


char[][][] threeD = new char[8][16][24];


String[][] dogs = {{ "terry", "brown" },
                   { "Kristin", "white" },
                   { "toby", "gray"},
                   { "fido", "black"}
                   };
```

1.5 Definisi Struct

Definisi **Struktur (*struct*)** sendiri adalah kumpulan dari variabel yang dinyatakan dengan sebuah nama , dengan sifat setiap variabel dapat memiliki tipe yang berlainan.

1.6 Deklarasi Struct

```
struct nama_struct {

tipe_data_1 nama_var_1;

tipe_data_2 nama_var_2;

tipe_data_3 nama_var_3;

……

};
```

Yang kedua adalah dengan deklarasi menggunakan typedef.

```
typedef struct {
tipe_data_1 nama_var_1;
tipe_data_n nama_var_n;
} nama_struct;
```

Kemudian untuk mendeklarasikan sebuah variable dengan tipe data struct yang telah dibuat sebelumnya adalah :

```
struct tipe_struct nama_variabel;
```

Jika pendeklarasian struct sebelumnya menggunakan typedef, maka untuk mendeklarasikan sebuah variable dengan tipe data struct adalah :

```
tipe_struct nama_variabel;
```

Dan untuk mengakses sebuah struct adalah dengan menggunakan operator titik (.)

```
nama_var_struct . nama_var_elemen;
```

NESTED STRUCT

Di dalam sebuah struct dapat dimungkinkan terdapat sebuah struct lagi. Jadi hal ini dapat diartikan struct di dalam struct. Hampir mirip nested loop, yaitu for di dalam for.

Contoh :

```
struct tanggal {
int hari;
int bulan;
int tahun;
};
struct karyawan {
char NIP [10];
char nama [20];
struct tanggal tgl_masuk;
float gaji;
};
```

STRUCT OF ARRAY

Sebuah struct yang di dalamnya tedapat variable dengan tipe data array.

Contoh :

```
struct data {
char nama[20];
```

```
char alamat[100];

};
```

ARRAY OF STRUCT

Sebuah array yang setiap data elemennya bertipe struct. Umumnya dipakai untuk menyimpan object data yang terstruktur, misal: data mahasiswa, karyawan, buku, barang, dsb.

Contoh :

```
typedef struct {

char npm [10];

char nama [20];

char alamat [30];

unsigned angkatan;

float ipk;

} mahasiswa ;

mahasiswa data[100];

// deklarasi var, menyiapkan 100 data dengan tipe data mahasiswa (struct yang telah dibuat sebelumnya).
```

1.7 Penggunaan Struct

- Struct adalah tipe data bentukan yang berisi kumpulan variabel-variabel
  yang bernaung dalam satu nama yang sama dan memiliki kaitan satu
  sama lain.

- Berbeda dengan array hanya berupa kumpulan variabel yang bertipe data
  sama, struct bisa memiliki variabel-variabel yang bertipe data sama atau
  berbeda, bahkan bisa menyimpan variabel yang bertipe data array atau
  struct itu sendiri.
- Variabel-variabel yang menjadi anggota struct disebut dengan elemen
  struct.
  Bentuk umum dari struct:

```
typedef struct{
  tipe_data <nama_var>;
  tipe_data <nama_var>;
  .... }
```

Struct bisa diumpamakan sebagai sebuah obyek, misalnya: obyek Mahasiswa. Struct Mahasiswa memiliki property atau atribut atau variabel yang melekat padanya:

- NIM misal karakter sejumlah 8
- Nama yaitu karakter
- IPK yaitu bilangan pecahan
  Struct tidak memiliki operasi (method) atau function. Struct dapat digunakan dengan cara membuat variabel yang bertipe struct tersebut.
  Misalnya :

- variabel anton bertipe struct Mahasiswa
- variabel erick bertipe struct Mahasiswa
  Dengan demikian variabel anton dan erick memiliki NIM, Nama, dan IPK masing-masing.

Ada dua cara untuk mendeklarasikan struct pada C yaitu:

Menggunakan keyword typedef:

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

```
//untuk menggunakan struct Mahasiswa dengan membuat variabel mhs dan mhs2
Mahasiswa mhs,mhs2;
//untuk menggunakan struct Mahasiswa dengan membuat variabel array m;
Mahasiswa m[100];
```

Menggunakan keyword struct:

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

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

Cara penggunaan struct dan pengaksesan elemen-elemennya:

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

Contoh Program 1 (Penggunaan Struct Sederhana) :

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

```
#include <conio.h>



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

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

     printf("NIM = ");scanf("%s",mhs.NIM);
     printf("Nama = ");scanf("%s",mhs.nama);
     printf("IPK = ");scanf("%f",&mhs.ipk);

     printf("Data Anda : \n");
     printf("NIM : %s\n",mhs.NIM);
     printf("Nama : %s\n",mhs.nama);
     printf("IPK : %f\n",mhs.ipk);
     getch();
}
```

Contoh Program 2 (Penggunaan Struct dalam Struct) :

```
#include <stdio.h>

#include <conio.h>
```

```
typedef struct Date{
	int dd;
	int mm;
	int yyyy;
};

typedef struct Time{
	int h;
	int m;
	int s;
};

typedef struct Login{
	int ID;
	Date tglLogin;
	Time waktuLogin;
};

int main(){
    Login user1;

    printf("USER 1\n");
    printf("ID : ");scanf("%d",&user1.ID);
    printf("Tanggal Login\n");
    printf("Hari : ");scanf("%d",&user1.tglLogin.dd);
    printf("Bulan : ");scanf("%d",&user1.tglLogin.mm);
    printf("Tahun : ");scanf("%d",&user1.tglLogin.yyyy);
    printf("Waktu Login\n");
    printf("Jam : ");scanf("%d",&user1.waktuLogin.h);
    printf("Menit : ");scanf("%d",&user1.waktuLogin.m);
    printf("Detik : ");scanf("%d",&user1.waktuLogin.s);
    printf("Terimakasih\n");
    printf("Data Anda :\n");
    printf("ID : %d\n",user1.ID);
```

```
    printf("Date : %d - %d -
%d\n",user1.tglLogin.dd,user1.tglLogin.mm,user1.tglLogin.yyyy)
;
    printf("ID
:%d:%d:%d\n",user1.waktuLogin.h,user1.waktuLogin.m,user1.w
aktuLogin.s);
    getch();
}
```

Contoh 3 (Penggunaan Array Of Struct)

```
#include <stdio.h>

#include <conio.h>



typedef struct Date{
        int dd;
        int mm;
        int yyyy;
       };

typedef struct Time{
        int h;
        int m;
        int s;
       };

typedef struct Login{
```

```
        int ID;
        Date tglLogin;
        Time waktuLogin;
        };

int main(){
    Login user[3];

    //3 user
    for(int i=0;i<3;i++){
        printf("\nUSER ke-%d\n",i+1);
        printf("ID : ");scanf("%d",&user[i].ID);
        printf("Tanggal Login\n");
        printf("Hari : ");scanf("%d",&user[i].tglLogin.dd);
        printf("Bulan : ");scanf("%d",&user[i].tglLogin.mm);
        printf("Tahun : ");scanf("%d",&user[i].tglLogin.yyyy);
        printf("Waktu Login\n");
        printf("Jam : ");scanf("%d",&user[i].waktuLogin.h);
        printf("Menit : ");scanf("%d",&user[i].waktuLogin.m);
        printf("Detik : ");scanf("%d",&user[i].waktuLogin.s);
        printf("Terimakasih Atas Pengisiannya\n");
        printf("\nData User ke-%d:\n",i+1);
        printf("Login ID : %d\n",user[i].ID);
        printf("Login Date : %d - %d -
%d\n",user[i].tglLogin.dd,user[i].tglLogin.mm,user[i].tglLogin.yy
yy);
        printf("Login Time :
%d:%d:%d\n",user[i].waktuLogin.h,user[i].waktuLogin.m,user[i].
waktuLogin.s);
     }
     getch();
}
```

1.8 Penggunaan Struct bersamaan dengan Array

```
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
#include<iomanip.h>
hadiah(char bonus[30],double total)
 {
if(total>=200000)
{strcpy(bonus,"1 Kaos Keren");}
else if(total>=150000)
{strcpy(bonus,"1 Topi Cantik");}
else
{strcpy(bonus,"Tidak Dapat");}
 }
main()
{
struct
{
char jenis_ctk[30],jenis_kertas[30];
int harga;
}offset[10];
int a,n,jumlah_ctk[10];
double ubay[10],ukem[10];
int total[10];
char bonus[10][30];
cout<<" \"Percetakan Amazing\""<<endl;
cout<<" Jl.Bagus Kita No.15, Jawa Tenggara"<<endl;
cout<<" No Telp.(099) 012113"<<endl;
cout<<" ========================================"<<endl;
cout<<"Input Banyak Data yang akan di Input :";cin>>n;
cout<<endl;
for(a=1;a<=n;a++)
{
cout<<"Data Ke-"<<a<<endl;
cout<<"Input Jenis Cetakan :";gets(offset[a].jenis_ctk);
cout<<"Input Jenis Kertas :";cin>>offset[a].jenis_kertas;
```

```
cout<<"Input Harga @lembar :";cin>>offset[a].harga;
cout<<"Input Banyak Cetak :";cin>>jumlah_ctk[a];
cout<<endl;
}
clrscr();
cout<<" \"Percetakan Amazing\""<<endl;
cout<<" Jl.Bagus Kita No.15, Jawa Tenggara"<<endl;
cout<<" No Telp.(099) 012113"<<endl;
cout<<" ========================================"<<endl;
for(a=1;a<=n;a++)
{
cout<<"Data Ke-"<<a<<endl;
cout<<"Jenis Cetakan :"<<offset[a].jenis_ctk<<endl;
cout<<"Jenis Kertas :"<<offset[a].jenis_kertas<<endl;
cout<<"Harga @Lembar :"<<offset[a].harga<<endl;
cout<<"Banyak Cetak :"<<jumlah_ctk[a]<<endl;
total[a]=offset[a].harga*jumlah_ctk[a];
cout<<"Total Bayar :"<<total[a]<<endl;
hadiah(bonus[a],total[a]);
cout<<"Bonus :"<<bonus[a]<<endl;
cout<<"-----------------------------------------------------"<<endl;
cout<<"Uang Bayar :Rp.";cin>>ubay[a];
ukem[a]=ubay[a]-total[a];
cout<<"Uang Kembali :Rp."<<ukem[a]<<endl;
cout<<"-----------------------------------------------------"<<endl;
cout<<endl;
}
getch();
}
```

1.9 Algoritma Kombinasi Struct dan Array

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

struct{
char nama[20]; int usia; char status[20]; char agama[20];//Variabel dalam
```

```
structure
}pdk[0];//Deklarasi Structure dan Variabelnya
int main()
{int x,p,i; char j;
printf("PROGRAM DATA PENDUDUK DESA MAKMUR 2014\n\n");
printf("Input Jumlah Penduduk Yang Akan Didata ! \xB2: ");
scanf("%d",&p);

for(x=1;x<=p;x++)
{printf("\n\nDATA PENDUDUK NOMOR URUT Ke-%d",x);
printf("\n\tInput Nama   : ");scanf("%s",pdk[x].nama);
printf("\tInput Usia   : ");scanf("%d",&pdk[x].usia);
printf("\tInput Status : ");scanf("%s",pdk[x].status);
printf("\tInput Agama  : ");scanf("%s",pdk[x].agama);
}
printf("\nInput Data Penduduk Sebanyak %d Orang Selesai\n",p);
printf("\nIngin Menampilkan Data Yang Sudah Diinput ?\
\n1.Tampilkan keseluruhan\n2.Tampilkan Per Penduduk\n3.Tidak Perlu\n\n
:");
scanf("%d",&i);

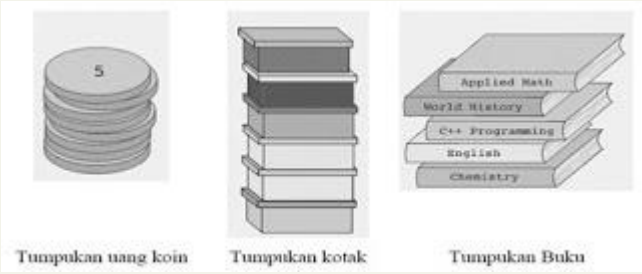
if(i==1)
{system("cls");
printf("DATA PENDUDUK DESA MAKMUR 2014\n\n");
for(x=1;x<=p;x++)
{printf("\n\nPenduduk %d",x);
printf("\nNama    :%s ",pdk[x].nama);
printf("\nUsia    :%d ",pdk[x].usia);
printf("\nStatus  :%s ",pdk[x].status);
printf("\nAgama   :%s ",pdk[x].agama);
}}
else if(i==2)
{system("cls");
printf("Input Nomor Penduduk Yang Datanya Ingin Ditampilkan ! : ");
scanf("%d",&x);
printf("\n\nPenduduk %d",x);
printf("\nNama    :%s ",pdk[x].nama);
```

```
printf("\nUsia    :%d ",pdk[x].usia);
printf("\nStatus  :%s ",pdk[x].status);
printf("\nAgama   :%s ",pdk[x].agama);
}
printf("\n\nTerimakasih Telah Menggunakan Program Ini !");
getche();
return 0;
}
```

Pertemuan 2 : Stack

2.1 Definisi Stack

Stack adalah suatu urutan yang elemennya dapat diambil dan ditambah hanya pada posisi akhir(top)saja.contoh dalam kehidupan sehari-hari adalah tumpukan piring disebuah restoran yang tumpukannya ditambah pada bagian paling atas dan jika mengambilnya pun dari bagian paling atas pula.



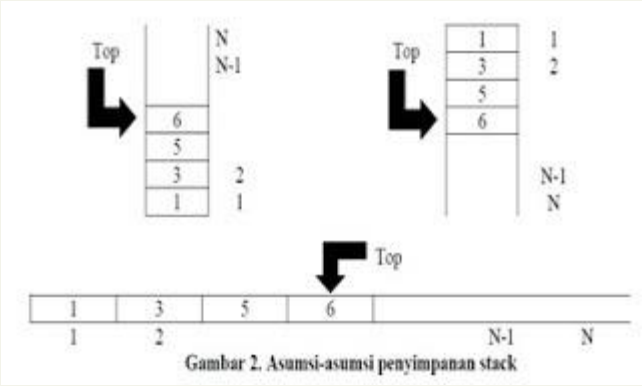
Bersifat **LIFO (Last In First Out)**. Benda yang terakhir masuk ke dalam stack akan menjadi benda pertama yang dikeluarkan dari stack.

Macam-Macam Stack

Stack ada 2 operasi paling besar yang dapat dilakukan,yaitu:

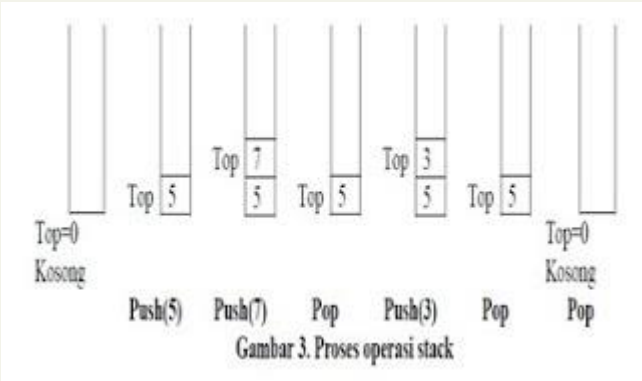
1. Operasi push yaitu operasi menambahkan elemen pada urutan terakhir(paling atas).
2. Operasi Pop yaitu mengambil sebuah elemen data pada urutan terakhir dan menghapus elemen tersebut dari stack

Sebagai contoh,misalkan ada data sebagai berikut:
1 3 5 6,maka data tersebut dapat tersimpan dalam bentuk sebagai berikut:



Gambar 2. Asumsi-asumsi penyimpanan stack

Contoh lain adalah ada sekumpulan perintah stack yaitu push(5),push(7),pop,push(3),pop.jika dijalankan,maka akan terjadi adalah:



Gambar 3. Proses operasi stack

Proses Operasi Stack

Selain operasi dasar stack,ada lagi operasi lain dapat terjadi dalam stack yaitu:

1. Proes Deklarasi yaitu proses pendeklarasian stack.
2. Proses Isempty yaitu proses pemeriksaan apakah stack dalam keadaan kosong.
3. Proses Isfull yaitu proses pemeriksaan apakah stack telah penuh.
4. Proses inisialisasi yaitu proses pembuangan stack kosong,biasanya dengan pemberian nilai untuk top.
5. Proses clear yaitu proses pengosongan stack.
6. Proses Print yaitu proses menampilkan semua elemen-elemen stack

Operasi-operasi Stack

1. Pendeklarasian stack

Proses pembuatan struktur stack dalam memori.

Top: yang berisi posisi data terakhir

Elemen: yang berisi data yang ada dalam stack bagian ini lah yang berbentuk array.

Maks_elemen yaitu variabel yang menunjuk maksimal banyaknya elemen dalam stack.

Dalam bahasa C, pendeklarasiannya adalah :

```
#define maks 100

//pendeklarasian struktur stack
struct tstack{
              int top;
              int maks_elemen;
              int elemen[maks];
            };

//pendeklarasian stack
tstack stack;
```

2. Proses Inisialisasi

Inisialisasi stack adalah proses pembuatan suatu stack kosong.




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

3. Operasi IsEmpty

Operasi yang digunakan untuk memeriksa apakah stack dalam keadaan kosong. Operasi ini dilakukan dengan memeriksa field top.jika top bernilai 0 atau top bernilai -1,maka berarti stack dalam keadaan empty.

Implementasi dalam bahasa C adalah :

```
int isempty(tstack stack)
{
    if (stack.top==-1)
       return 1;
    else
       return 0;
}
```

Cara penggunaannya adalah :

```
//Penggunaan isempty dalam statement if
if( isempty(stack) ) ...
```

4. Operasi IsFull

Untuk memeriksa keadaan stack apakah sudah penuh/belum.

Langkah-langkah untuk operasi ini:

a. Operasi IsFull pada stack yang menggunakan array.

b. Operasi ini akan memberikan nilai true(1) jika field top sama dengan field maks_elemen -1.

Implementasinya dalam bahasa C adalah :

```
int isfull(tstack stack)
{
    if (stack.top==(stack.maks_elemen-1))
       return 1;
    else
      return 0;
}
```

Cara pemanggilannya adalah :

```
//penggunaan isfull dalam statement if
if( !isfull(stack) ) … // jika stack tidak penuh
```

5. Operasi Push

Operasi push adalah operasi dasar dari stack yang berguna untuk menambah suatu elemen data baru pada stack dan di simpan pada posisi top yang akan mengakibatkan posisi top akan berubah.

Langkah-langkah operasi ini:

v Periksa apakah stack penuh(isfull).jika bernilai false/0 (tidak penuh)maka proses push dilaksanakan dan jika pemeriksaan ini bernilai true/1,maka proses push digagalkan.

v Proses push-nya sendiri adalah dengan menambahkan field top dengan 1,kemudian elemen pada posisi top di isi dengan elemen data baru.

Implementasinya dalam bahasa C adalah :

```
void push(tstack *stack, int baru)
{
   if(!isfull(*stack))
   {
      stack->top++;
      stack->elemen[stack->top]=baru;
   }
   else
   {
      printf("Stack Full. Push Gagal.\n");
   }
}
```

Cara penggunaannya adalah :

```
push(&stack,5);// push 5 ke dalam stack
```

6. Operasi Pop

Operasi Pop adalah salah satu operasi paling besar dari stack yang 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. 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 disimpan dalam variabel baru dengan nama “data”.

Implementasi operasi ini dalam bahasa C adalah :

```
int pop(tstack *stack)
{
      int data;
      if(!isempty(*stack))
      {
        data=stack->elemen[stack->top];
        stack->top--;
        return data;
      }
      else
       return 0;

}
```

Cara pemanggilannya adalah :

```
int data;
data=pop(&stack);
```

7. Operasi Print

Untuk menampilkan semua elemen-elemen stack. Dengan cara looping semua nilai array secara terbalik, karena kita harus mengaksesdari indeks array tertinggi terlebih dahulu baru ke indeks yang kecil.

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

2.2 Contoh Kasus Stack dalam sehari-hari

Stack adalah merupakan sebuah kumpulan benda dimana hanya benda yang baru dimasukkan yang dapat diakses atau dilihat. Stack juga merupakan perintah pengumpulan data secara linier yang disebut “last in, first out” (LIFO).
Misalnya:
• Setumpuk koran, dimana koran yang paling terakhir ditambahkan dan ditaruh di atas tumpukan yang dapat dilihat.
• Tumpukan kotak rokok, koin, buku, dll

2.3 Algoritma Stack

a. Pendeklarasian Stack

Suatu stack memiliki beberapa bagian yaitu:

1. Top : yang berisi posisi data terakhir.

2. Elemen : yang berisi data yang ada dalam stack bagian inilah yang berbentuk array.

3. Maks_elemen yaitu variabel yang menunjuk maksimal banyaknya elemen dalam stack.

Dalam bahasa C, pendeklarasiannya adalah :

```
#define maks 100

//pendeklarasian struktur stack
struct tstack{
            int top;
            int maks_elemen;
            int elemen[maks];
          };

//pendeklarasian stack
tstack stack;
```

b. Inisialisasi

Inisialisasi Stack adalah proses pembuatan suatu stack kosong. Adapun langkah-langkah proses inisialisasi stack yang menggunakan array. Dengan mengisi nilai field top dengan 0. Jika elemen pertama diawali dengan nomor 1, kalau elemen pertamanya array dimulai dengan 0, maka top diisi dengan -1.

Implementasinya dalam bahasa C adalah :

```
void inisialisasi(tstack *stack)
{
    stack->top=-1;//karena dalam C array dimulai dgn 0
    stack->maks_elemen=maks;
}
```

Cara pemanggilannya adalah

```
inisialisasi(&stack);
```

c. Operasi IsEmpty

Operasi yang digunakan untuk memeriksa apakah stack dalam keadaan kosong. Operasi ini dilakukan dengan memeriksa field top, jikatop bernilai 0 atau top bernilai -1, maka berarti dalam keadaan empty.

Implementasi dalam bahasa C adalah :

```
int isempty(tstack stack)
{
    if (stack.top==-1)
       return 1;
    else
       return 0;
}
```

Cara penggunaannya adalah :

```
//Penggunaan isempty dalam statement if
if( isempty(stack) ) ...
```

d. Operasi IsFull

Operasi ini untuk memeriksa keadaan stack apakah sudah penuh/belum. Operasi ini memberikan nilai true(1) jika field top sama dengan field maks_elemen.

Implementasinya dalam bahasa C adalah :

```
int isfull(tstack stack)
{
    if (stack.top==(stack.maks_elemen-1))
       return 1;
    else
      return 0;
}
```

Cara pemanggilannya adalah :

```
//penggunaan isfull dalam statement if
if( !isfull(stack) ) … // jika stack tidak penuh
```

e. Operasi Push

Operasi push adalah operasi dasar dari stack yang berguna untuk menambahkan suatu elemen data baru pada stack dan di simpan pada posisi top yang akan mengakibatkan posisi top akan berubah.

Langkah-langkah operasi ini:

1. Periksa apakah stack penuh(IsFull). Jika berniali false/0(tidak penuh) maka proses push dilakukan dan jika pemeriksaan ini bernilai true/1, maka proses push digagalkan.

2. Proses Push-nya sendiri adalah dengan menambahkan field topdengan 1, kemudian elemen pada posisi top di isi dengan elemen data baru.

Implementasinya dalam bahasa C adalah :

```
void push(tstack *stack, int baru)
{
   if(!isfull(*stack))
   {
      stack->top++;
      stack->elemen[stack->top]=baru;
   }
   else
   {
      printf("Stack Full. Push Gagal.\n");
   }
}
```

Cara penggunaannya adalah :

```
push(&stack,5);// push 5 ke dalam stack
```

f. Operasi Pop

Operasi Pop adalah salah satu operasi paling besar dari stack yang brguna 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.

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 disimpan dalam variabel baru dengan nama "data".

Implementasi operasi ini dalam bahasa C adalah :

```
int pop(tstack *stack)
{
      int data;
      if(!isempty(*stack))
      {
        data=stack->elemen[stack->top];
        stack->top--;
        return data;
      }
      else
       return 0;

}
```

Cara pemanggilannya adalah :

```
int data;
data=pop(&stack);
```

2.4 Algoritma Stack dalam Bahasa Pemrograman

```
#include <stdio.h>
#include <conio.h>
#define max 5

typedef struct {
   int top;
   int data[max+1];
}stack;

stack tumpukan;
void createEmpty();
int IsEmpty();
int IsFull();
void push(int x);
void pop();

main(){
   int lagi;
   int input;
   int pilih;
   createEmpty();
   pilih = 0;
   while (pilih != 5){
      puts("_____");
      printf("            MENU UTAMA\n");
      //Printf dan Puts memiliki Fungsi yang sama
      puts("_____");
      puts("1: Cek kondisi Stack");
      puts("2: Tambah data (Pust)");
      puts("3: Keluarkan isi stack (Pop)");
      puts("4: Kosongkan stack");
      puts("5: Keluar");
      printf("Pilihan Anda: ");
      scanf("%d",&pilih);
      switch(pilih){
      case 1:
         if (IsEmpty() == 1)
            puts("Stack masih kosong");
```

```
            else if ((IsEmpty() == 0) && (IsFull() == 0))
               puts("Stack sudah terisi (belum penuh)");
            else
               puts("Stack sudah penuh");
         break;
         case 2:
            if (IsFull() == 1)
               puts("Stack Penuh.");
            else
            {
               printf("Masukkan data: ");
               scanf("%d",&input);
               push(input);
               printf("Top baru: %d\n",tumpukan.data[tumpukan.top]);
               printf("IsFull: %d\n",IsFull());
               printf("IsEmpty: %d\n",IsEmpty());
            }
         break;
         case 3:
                                    printf("Top yang dikeluarkan bernilai: %d
\n",tumpukan.data[tumpukan.top]);
            pop();
         break;
         case 4:
            while (IsEmpty() == 0)
            {
                                     printf("Top yang dikeluarkan bernilai: %d
\n",tumpukan.data[tumpukan.top]);
               pop();
            }
            puts("Stack sudah kosong");
         break;
         case 5:
            puts("---Terimakasih || MARKIJAR.Com---");
         break;
         }
      }
}
```

```
void createEmpty(){
   tumpukan.top = 0;
}

void push(int x){
   tumpukan.top = tumpukan.top + 1;
   tumpukan.data[tumpukan.top] = x;
}

void pop(){
   tumpukan.top = tumpukan.top - 1;
}

int IsEmpty(){
   if (tumpukan.top == 0)
      return 1;
   else
      return 0;
}

int IsFull(){
   if (tumpukan.top == max)
      return 1;
   else
      return 0;
}
```

Pertemuan 3 : Stack II

3.1 Stack dan penerapannya dalam penyelesaian ekspresi matematik

Prefix, infix, dan postfix adalah suatu cara penulisan ungkapan-ungkapan yang rumit, misalnya pemakaian tanda kurung dalam operasi matematika.

Prefix adalah metode penulisan dengan meletakkan *operator di depan operand* dan tanpa menuliskan tanda kurung.

Contoh pemakaian prefix adalah +AB, – +ABC, * + AB – CD.

Infix adalah cara penulisan ungkapan dengan meletakkan *operator di antara dua operand*dalam hal ini pemakaian tanda kurung sangat menentukan hasil operasi.

Contoh pemakaian infix adalah A+B, A+B-C, (A+B)*(C-D).

3.2 Mengubah ekspresi matematik ke dalam bentuk postfix dengan metode stack

Postfix adalah metode penulisan dengan menuliskan *operator setelah operand* dan tanpa menuliskan tanda kurung.

Contoh penulisan sufix adalah AB + , AB + C – , AB + CD -*.

Salah satu contoh proses pengubahan infix menjadi postfix dari karakter:

(A + B) / ((C – D) * E ^ F)

Karakter dibaca	***Isi Stack***	***Karakter tercetak***	***Postfix yang terbentuk***
(	(		
A	(	A	A
+	(+		
B		B	A B
)		+	A B +
/	/		
(	/ (		
(	/ ((		
C	/ ((	C	A B + C
–	/ ((–		
D	/ ((–	D	A B + C D
)	/ (	–	A B + C D –
*	/ (*		
E	/ (*	E	A B + C D – E
^	/ (* ^		
F	/ (* ^	F	A B + C D – E F
)	/ (*	^	A B + C D – E F ^
	/ (	*	A B + C D – E F ^ *
	/		
		/	A B + C D – E F ^ * /

3.3 Menyelesaikan ekspresi matematik dalan bentuk postfix

Postfix yaitu notasi yang terbentuk atas operator dengan operand, dimana operator berada dibelakang operand. Notasi ini hanya dikenal oleh processor dan dipahami dalam ALU.

Contoh : A + B * C (**Infix**)

maka notasi **postfix**nya adalah **ABC*+**

Pemecahannya :

A + B * C

diketahaui ada 3 operand yaitu : A, B, C, dan 2 operator yaitu : +, *. Proses dimulai dengan melihat dari hirarkhi operator. Contoh diatas operator yang tertinggi adalah * kemudian +.

Tanda * diapit oleh dua operand yaitu B dan C yaitu B * C
, **postfixnya** dengan menggabungkan operand B dan C menjadi BC lalu memindahkan operator ke belakang operand C, **sehingga fungsi B * C, notasi postfixnya menjadi BC*. Sehingga hasil sementara dari notasi postfix adalah**

A + BC*

selanjutnya mencari postfix untuk operator yang berikutnya, yaitu +, cara yang dilakukan sama seperti di atas, operator +, diapit oleh 2 operand, yaitu A dan BC*, gabungkan operand tersebut, sehingga menjadi ABC*, lalu pindahkan operator + ke belakang operand ABC*, **sehingga hasil akhir menjadi**

ABC*+

Contoh yang lain:

1. A + B – C * D

2 3 1 ——> hirarkhi level

A + B – CD* ——> 1

AB+ – CD* ——> 2

AB+CD*- ——> 3

2. A * B ^ C – D

2 1 3 ——> hirarkhi

A * BC^ – D ——> 1

ABC^* – D ——> 2

ABC^*D- ——> 3

3. A + (B – C) * D

3 1 2 ——> hirarkhi

A + BC- * D ——> 1 (karena diapit tanda paranthesis atau kurung buka/tutup,( ))

A + BC-D* ——> 2

A BC-D*+ ——> 3

Pertemuan 4 : Queue

4.1 Definisi Queue

Queue pada **Struktur Data** atau antrian adalah sekumpulan data yang mana penambahan elemen hanya bisa dilakukan pada suatu ujung disebut dengan sisibelakang(rear), dan penghapusan(pengambilan elemen) dilakukan lewat ujung lain (disebut dengan sisi depan atau front).

4.2 Contoh Kasus Queue dalam Kehidupan Sehari-hari

Queue atau antrian banyak kita jumpai dalam kehidupan sehari-hari, ex: antrian Mobil diloket Tol, Antrian mahasiswa Mendaftar, dll.

Contoh lain dalam bidang komputer adalah pemakaian sistem komputer berbagi waktu(time-sharing computer system) dimana ada sejumlah pemakai yang akan menggunakan sistem tersebut secara serempak.

4.3 Algoritma Queue

- tambah(menambah item pada belakang antrian)
- hapus (menghapus elemen depan dari antrian)
- kosong(mendeteksi apakah pada antrian mengandung elemen atau tidak)

4.4 Algoritma Queue dalam Bahasa Pemrograman

1. **Create()**

Untuk menciptakan dan menginisialisasi Queue

Dengan cara membuat Head dan Tail = -1

DEKLARASI QUEUE

```
#define MAX 8
typedef   struct{
                    int data[MAX];
                    int head;
                    int tail;
                } Queue;

Queue antrian;
```



Head = -1

Antrian pertama kali

Tail = -1

```
void Create(){
     antrian.head=antrian.tail=-1;
}
```

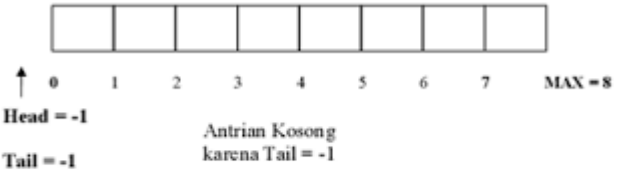
2. **IsEmpty()**

Untuk memeriksa apakah Antrian sudah penuh atau belum

Dengan cara memeriksa nilai Tail, jika Tail = -1 maka empty

Kita tidak memeriksa Head, karena Head adalah tanda untuk kepala antrian (elemen pertama dalam antrian) yang tidak akan berubah-ubah

Pergerakan pada Antrian terjadi dengan penambahan elemen Antrian kebelakang, yaitu menggunakan nilai Tail.

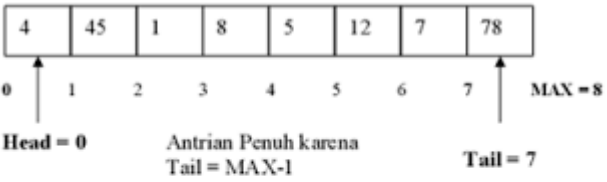



```
int IsEmpty(){
     if(antrian.tail==-1)
          return 1;
     else
          return 0;
}
```

3. **IsFull**

Untuk mengecek apakah Antrian sudah penuh atau belum

Dengan cara mengecek nilai Tail, jika Tail >= MAX-1 (karena MAX-1 adalah batas elemen array pada C) berarti sudah penuh

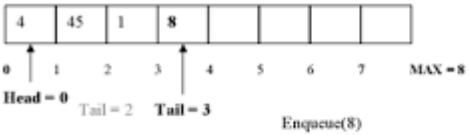



```
int IsFull(){
     if(antrian.tail==MAX-1) return 1;
     else return 0;
}
```

4. **Enqueue**

Untuk menambahkan elemen ke dalam Antrian, penambahan elemen selalu ditambahkan di elemen paling belakang

Penambahan elemen selalu menggerakan variabel Tail dengan cara increment counter Tail terlebih dahulu



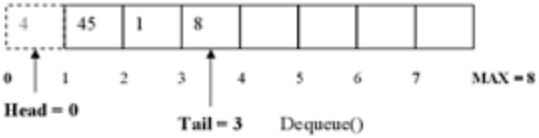

```
void Enqueue(int data){
     if(IsEmpty()==1){
          antrian.head=antrian.tail=0;
          antrian.data[antrian.tail]=data;
          printf("%d masuk!",antrian.data[antrian.tail]);
     } else
     if(IsFull()==0){
          antrian.tail++;
          antrian.data[antrian.tail]=data;
          printf("%d masuk!",antrian.data[antrian.tail]);
     }
}
```

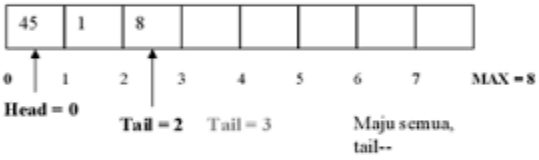
5. **Dequeue()**

Digunakan untuk menghapus elemen terdepan/pertama (head) dari Antrian

Dengan cara menggeser semua elemen antrian kedepan dan mengurangi Tail dgn 1

Penggeseran dilakukan dengan menggunakan looping.



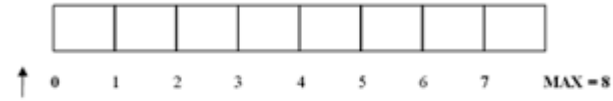



```
int Dequeue(){
     int i;
     int e = antrian.data[antrian.head];
     for(i=antrian.head;i<=antrian.tail-1;i++){
          antrian.data[i] = antrian.data[i+1];
     }
     antrian.tail--;
     return e;
}
```

6. **Clear()**

Untuk menghapus elemen-elemen Antrian dengan cara membuat Tail dan Head = -1

Penghapusan elemen-elemen Antrian sebenarnya tidak menghapus arraynya, namun hanya mengeset indeks pengaksesan-nya ke nilai -1 sehingga elemen-elemen Antrian tidak lagi terbaca




```
void Clear(){
     antrian.head=antrian.tail=-1;
     printf("data clear");
}
```

7. **Tampil()**

Untuk menampilkan nilai-nilai elemen Antrian

Menggunakan looping dari head s/d tail

```
void Tampil(){
     if(IsEmpty()==0){
          for(int i=antrian.head;i<=antrian.tail;i++){
               printf("%d ",antrian.data[i]);
          }
     }else printf("data kosong!\n");
}
```

Pertemuan 5 : Sorting

5.1 Definisi Sorting

Pengurutan *(Sorting)* merupakan proses pengurutan sekumpulan data dalam suatu urutan tertentu.

Sorting dipakai untuk:

1.Membantu proses pencarian (searching)

2.Menyelesaikan masalah-masalah kompleks seperti penjadwalan (scheduling), pengolahan basis data, riset operasi, dsb.

5.2 Jenis-jenis Sorting

1. Bubble Sort

Memindahkan element sekarang dengan elemen berikutnya , jika elemen sekarang itu lebih besar / lebih kecil dari elemen berikutnya maka di tukar (berpindah posisi).

Proses Bubble Sort

Ascending

- Data yang paling awal dibandingkan dengan data berikutnya jika ternyata lebih besar maka tukar.
- Data yang paling akhir dibandingkan dengan data sebelumya jika ternyata lebih kecil maka tukar.

Descending

- Data yang paling awal dibandingkan dengan data berikutnya jika ternyata lebih kecil maka tukar.

- Data yang paling akhir dibandingkan dengan data sebelumya jika ternyata lebih besar maka tukar.

2. Exchange Sort

Exchange sort itu sangat mirip dengan buble sort. Bahkan banyak yang mengatakan bahwa exchange sort sama dengan buble sort. Bedanya jika bubble sort proses pertukarannya harus sistematis, dari awal atau dari belakang. Sedangkan exchange sort proses pertukaran hanya akan dilakukan jika diperlukan saja.

Proses Exchange Sort

Ascending

- Jika terdapat elemen sebelumnya lebih besar dari elemen berikutnya maka tukar.
- Jika terdapat elemen sesudahnya lebih kecil dari elemen sebelumnya maka tukar.

Descending

- Jika terdapat elemen sebelumnya lebih kecil dari elemen berikutnya maka tukar.
- Jika terdapat elemen sesudahnya lebih besar dari elemen sebelumnya maka tukar.

3. Selection Sort

Memindahkan elemen dengan cara membandingkan elemen sekarang dengan elemen yang berikutnya sampai dengan elemen terakhir . Jika ditemukan elemen lain yang lebih kecil / lebih besar

dari elemen sekarang maka dicatat posisinya dan kemudian ditukar dan begitu seterusnya.

Proses Selection Sort

1. Ascending

❖ Elemen yang paling besar diletakkan di akhir.

❖ Elemen yang paling kecil diletakkan di awal.

2. Descending

❖ Elemen yang paling kecil diletakkan di akhir.

❖ Elemen yang paling besar diletakkan di awal.

Contoh Selection Sort

Ascending (Urutan Naik)

4. Insertion Sort

Pengurutan yang dilakukan dengan cara membandingkan dengan data ke-2 sampai data terakhir. Jika ditemukan data yang lebih kecil atau lebih besar maka data tersebut disisipkan kedepan sesuai posisi yang seharusnya.

Contoh Insertion Sort

Ascending (Urutan Naik)

5. Shell Sort

Merupakan proses pengurutan data yang sebelumnya acak menjadi data yang terurut dengan cara menentukan jarak antar elemen yang akan dibandingkan.

Contoh Shell Sort

Ascending (Urutan Naik)

6. Merge Sort

Merupakan proses pengurutan data yang menggunakan merging dua buah vector. Pada proses merge sort, data dibuat sepasang-sepasang, yang terdiri dari dua elemen. Jika N ganjil, maka ada satu vector yang terdiri dari 1 elemen. Lalu kemudian data tersebut di merging sampai terurut.

Contoh Merge Sort

Ascending (Urutan Naik)

7. Quick Sort

Merupakan proses penyusunan elemen yang membandingkan suatu elemen (pivot) denan elemen yang lain, dan menyusunnya sedemikian rupa sehingga elemen –elemen lain yang lebih kecil dari

pivot terletak disebelah kiri pivot. Dan elemen yang lebih besar dari pivot terletak disebelah kanan pivot.

Dengan demikian akan terbentuk dua sublist, yang terletak disebelah kanan dan kiri pivot. Lalu pada sublist kiri dan kanan itu kita anggap sebuah list baru, dan kita kerjakan proses yang sama seperti yang sebelumnya. Demikian seterusnya sampai tidak terdapat sublist lagi.

5.3 Algoritma Sorting

- Membandingkan data ke-i dengan data ke-(i+1) (tepat bersebelahan). Jika tidak sesuai maka tukar (data ke-i = data ke-(i+1) dan data ke-(i+1) = data ke-i). Apa maksudnya tidak sesuai? Jika kita menginginkan algoritme menghasilkan data dengan urutan ascending (A-Z) kondisi tidak sesuai adalah data ke-i > data ke-i+1, dan sebaliknya untuk urutan descending (A-Z).
- Membandingkan data ke-(i+1) dengan data ke-(i+2). Kita melakukan pembandingan ini sampai data terakhir. Contoh: 1 dgn 2; 2 dgn 3; 3 dgn 4; 4 dgn 5 … ; n-1 dgn n.
- Selesai satu iterasi, adalah jika kita sudah selesai membandingkan antara (n-1) dgn n. Setelah selesai satu iterasi kita lanjutkan lagi iterasi berikutnya sesuai dengan aturan ke-1. mulai dari data ke-1 dgn data ke-2, dan seterusnya.

- Proses akan berhenti jika tidak ada pertukaran dalam satu iterasi

5.4 Algoritma Sorting dalam Bahasa Pemrograman

- Insertion Sort

```
public class insertion
{
public static void main(String args[])
{
int a[]={8,5,9,6,3,4,2,1,7,5};
for(int i=0;i<a.length;i++)
{
int min=a[i];
int j=i;
while((j>0)&&(min<a[j-1]))
{
a[j]=a[j-1];
j--;
}
a[j]=min;
}
// mencetak hasil
for(int h=0;h<a.length;h++)
{
System.out.println(a[h]+",");
}
}
}
```

- Selection Sort

```
a[]={8,5,9,6,3,4,2,1,7,5};
int i=0;
while(i<a.length-1){
int tmp=i;
for(int j=i+1;j<a.length;j++){
if(a[j]<a[tmp])
tmp=j;
}
int hlp=a[i];
```

```
a[i]=a[tmp];
a[tmp]=hlp;
i++;
}
// mencetak hasil
for(int h=0;h<a.length;h++){
System.out.println(a[h]+",");
}
}
}
```

- Bubble Sort

```
public class bubble
{
public static void main(String args[])
{
int a[]={8,5,9,6,3,4,2,1,7,5};
for(int i=1;i<a.length;i++)
{
for(int h=0;h<a.length-1;h++)
{
if(a[h+1]<a[h])
{
int tmp=a[h+1];
a[h+1]=a[h];
a[h]=tmp;
}
}
}
// mencetak hasil
for(int h=0;h<a.length;h++)
{
System.out.println(a[h]+",");
}
}
```

Pertemuan 6 : Searching

6.1 Definisi Searching

Pencarian (Searching) merupakan proses yang fundamental dalam pemrograman, guna menemukan data (nilai) tertentu di dalam sekumpulan data yang bertipe sama. Fungsi pencarian itu sendiri adalah untuk memvalidasi (mencocokkan) data.

Jenis-jenis Searching

- Metode Pencarian Beruntun
  Konsep yang digunakan dalam metode ini adalah membandingkan data-data yang ada dalam kumpulan tersebut, mulai dari elemen pertama sampai elemen ditemukan, atau sampai elemen terakhir.
- Metode Pencarian Bagi Dua (Binary Search)
  Metode ini diterapkan pada sekumpulan data yang sudah terurut (menaik atau menurun). Metode ini lebih cepat dibandingkan metode pencarian beruntun. Data yang sudah terurut menjadi syarat mutlak untuk menggunakan metode ini.
  Konsep dasar metode ini adalah membagi 2 jumlah elemennya, dan menentukan apakah data yang berada pada elemen paling tengah bernilai sama, lebih dari atau kurang dari nilai data yang akan dicari. Jika bernilai sama, maka langsung data yang dicari ditemukan. Jika data di elemen terurut naik, maka jika data yang berada di tengah kurang dari data yang dicari, maka pencarian selanjutnya berkisar di elemen tengah ke kanan, dan begitu seterusnya sampai ketemu atau tidak sama sekali. Dan sebaliknya untuk nilai data yang berada di tengah lebih dari data yang dicari, maka pencarian selanjutnya berkisar di elemen tengah ke kiri, dan begitu seterusnya sampai ketemu atau tidak sama sekali. Dan demikian sebaliknya untuk data yang terurut menurun. Dalam hal ini tentukan indeks paling awal dan indeks paling akhir, untuk membagi 2 elemen tersebut.
  Indeks awal = i, dimana nilai i, pada awalnya bernilai 0;

Indeks akhir = j, dimana nilai j, pada awalnya bernilai sama dengan jumlah elemen.

Algoritma Searching pada data tidak terurut

6.4 Algoritma Searcging pada data terurut

Pencarian Sekuensial (sequential searching) atau pencarian berurutan sering disebut pencarian linear merupakan metode pencarian yang paling sederhana. Pencarian beruntun adalah proses membandingkan setiap elemen larik satu per satu secara beruntun, mulai dari elemen pertama sampai elemen yang dicari ditemukan atau seluruh elemen sudah diperiksa. Pencarian beruntun terbadi dua:

- Pencarian beruntun pada larik tidak terurut;
- Pencarian beruntun pada larik terurut.

Algoritma

Pencarian berurutan menggunakan prinsip sebagai berikut :

- data yang ada dibandingkan satu per satu secara berurutan dengan yang dicari sampai data tersebut ditemukan atau tidak ditemukan.
- Pada dasarnya, pencarian ini hanya melakukan pengulangan dari 1 sampai dengan jumlah data.
- Pada setiap pengulangan, dibandingkan data ke-i dengan yang dicari.
- Apabila sama, berarti data telah ditemukan. Sebaliknya apabila sampai akhir pengulangan tidak ada data yang sama, berarti data tidak ada.

Kelemahan pada kasus yang paling buruk, untuk N elemen data harus dilakukan pencarian sebanyak N kali pula. Algoritma pencarian berurutan dapat dituliskan sebagai berikut :

(1) i ← 0

(2) ketemu ← false

(3) Selama (tidak ketemu) dan (i <= N) kerjakan baris 4

(4) Jika (Data[i] = x) maka ketemu ← true, jika tidak i ← i + 1

(5) Jika (ketemu) maka i adalah indeks dari data yang dicari, jika data tidak ditemukan

Contoh

```
#include <stdio.h>
#include <conio.h>
void main(){
int data[8] = {8,10,6,-2,11,7,1,100};
 int cari;
   int flag=0;
  printf("masukkan data yang ingin dicari = ");scanf("%d",&cari);
   for(int i=0;i<8;i++){
              if(data[i] == cari) flag=1;
   }
   if(flag==1) printf("Data ada!\n");
else printf("Data tidak ada!\n");
getch();
return 1;
}
```

Dari program diatas, terlihat bahwa dilakukan perulangan untuk mengakses semua elemen array data satu persatu berdasarkan indeksnya.

Program menggunakan sebuah variabel flag yang berguna untuk menadai ada atau tidaknya data yang dicari dalam array data. Hanya bernilai 0 atau 1.

Flag pertama kali diinisialiasasi dengan nilai 0.

Jika ditemukan, maka flag akan diset menjadi 1, jika tidak ada maka flag akan tetap bernilai 0.

Semua elemen array data akan dibandingkan satu persatu dengan data yang dicari dan diinputkan oleh user.

Algoritma Searching dalam Bahasa Pemrograman

Sequential Search

```
#include <stdio.h>
#include <stdlib.h>
main(){
    int tabInt[10]={24,17,18,15,22,26, 13,21, 16,
28};
    int ketemu;
    int cariData;
    int i;
    printf("Masukkan data yang dicari = ");
    scanf("%d",&cariData);
    i=0;
    ketemu=0;
    while (i<10 && ketemu!=1){
        if (tabInt[i]==cariData){
        ketemu=1;
        }else{
            i++;
        }
    }
    if(ketemu==1){
        printf("Data %d terdapat pada kumpulan
data\n",cariData );
    }else{
```

```
        printf("Data %d tidak terdapat pada kumpulan
data\n",cariData );
    }
}
```

Binary Search

```
#include <stdio.h>
#include <stdlib.h>
    int main(){
    int tabInt[10] = {12,23,29,34,56,60,67,78,84,99};
    int i,j,k;
    int cariData,ketemu;
    printf("Masukkan data yang dicari = ");
    scanf("%d",&cariData);
    i = 0; j = 9;ketemu = 0;
        while((ketemu == 0) && (i<=j)){
            k = (int)(i + j) / 2;
            if(tabInt[k] == cariData){
            ketemu = 1;
        }
        else{
            if(tabInt[k] > cariData){
                j = k - 1;
            }
            else{
                i = k + 1;
            }
        }
    }
    if(ketemu==1){
        printf("Data %d terdapat pada kumpulan
data\n",cariData );
    }else{
        printf("Data %d tidak terdapat pada kumpulan
data\n",cariData );
    }
}
```

Pertemuan 10 : Rekursif

10.1 Definisi Rekursif

Prosedur dan fungsi merupakan sub program yang sangat bermanfaat dalam pemrograman, terutama untuk program atau proyek yang besar. Manfaat penggunaan sub program antara lain adalah :

Prosedur dan fungsi merupakan sub program yang sangat bermanfaat dalam pemrograman, terutama untuk program atau proyek yang besar. Manfaat penggunaan sub program antara lain adalah :

- meningkatkan readibility, yaitu mempermudah pembacaan program
- meningkatkan modularity, yaitu memecah sesuatu yang besar menjadi modul-modul atau bagian-bagian yang lebih kecil sesuai dengan fungsinya, sehingga mempermudah pengecekan, testing dan lokalisasi kesalahan.
- meningkatkan reusability, yaitu suatu sub program dapat dipakai berulang kali dengan hanya memanggil sub program tersebut tanpa menuliskan perintah-perintah yang semestinya diulang-ulang.

Sub Program Rekursif adalah sub program yang memanggil dirinya sendiri selama kondisi pemanggilan dipenuhi.adalah Dengan melihat sifat sub program rekursif di atas maka sub program rekursif harus memiliki :

- kondisi yang menyebabkan pemanggilan dirinya berhenti (disebut kondisi khusus atau special condition)
- pemanggilan diri sub program (yaitu bila kondisi khusus tidak dipenuhi)

Secara umum bentuk dari sub program rekursif memiliki statemen kondisional :

- if kondisi khusus tak dipenuhi
  then panggil diri-sendiri dengan parameter yang sesuai
  else lakukan instruksi yang akan dieksekusi bila kondisi khusus dipenuhi
- Sub program rekursif umumnya dipakai untuk permasalahan yang memiliki langkah penyelesaian yang terpola atau langkah-langkah yang teratur. Bila kita memiliki suatu permasalahan dan kita mengetahui algoritma penyelesaiannya, kadang-kadang sub program rekursif menjadi pilihan kita bila memang memungkinkan untuk dipergunakan. Secara algoritmis (dari segi algoritma, yaitu bila kita mempertimbangkan penggunaan memori, waktu eksekusi sub program) sub program rekursif sering bersifat tidak efisien .

Dengan demikian sub program rekursif umumnya memiliki efisiensi dalam penulisan perintah, tetapi kadang tidak efisien secara algoritmis. Meskipun demikian banyak pula permasalahan-permasalahan yang lebih sesuai diselesaikan dengan cara rekursif (misalnya dalam pencarian / searching, yang akan dibahas pada pertemuan-pertemuan yang akan datang).

10.2 Contoh kasus penggunaan Rekursif

Kode berikut memperlihatkan contoh fungsi rekursif, untuk menghitung hasil kali dari dua bilangan:

```
def kali(a, b):
    return a if b == 1 else a + kali(a, b - 1)
```

Bagaimana cara kerja fungsi rekursif ini? Sederhananya, selama nilai b bukan 1, fungsi akan terus memanggil perintah a + kali(a, b - 1), yang tiap tahapnya memanggil dirinya sendiri sambil mengurangi nilai b. Mari kita coba panggil fungsi kali dan uraikan langkah pemanggilannya:

```
kali(2, 4)
  -> 2 + kali(2, 3)
  -> 2 + (2 + kali(2, 2))
  -> 2 + (2 + (2 + kali(2, 1)))
  -> 2 + (2 + (2 + 2))
  -> 2 + (2 + 4)
  -> 2 + 6
  -> 8
```

Perhatikan bahwa sebelum melakukan penambahan program melakukan pemanggilan fungsi rekursif terlebih dahulu sampai fungsi rekursif mengembalikan nilai pasti (\(2\)). Setelah menghilangkan semua pemanggilan fungsi, penambahan baru dilakukan, mulai dari nilai kembalian dari fungsi yang paling terakhir. Mari kita lihat contoh fungsi rekursif lainnya, yang digunakan untuk melakukan perhitungan faktorial:

```
def faktorial(n):
    return n if n == 1 else n * faktorial(n - 1)
```

Fungsi faktorial memiliki cara kerja yang sama dengan fungsi kali. Mari kita panggil dan lihat langkah pemanggilannya:

```
faktorial(5)
  -> 5 * faktorial(4)
```

```
-> 5 * (4 * faktorial(3))
-> 5 * (4 * (3 * faktorial(2)))
-> 5 * (4 * (3 * (2 * faktorial(1))))
-> 5 * (4 * (3 * (2 * 1)))
-> 5 * (4 * (3 * 2))
-> 5 * (4 * 6)
-> 5 * 24
-> 120
```

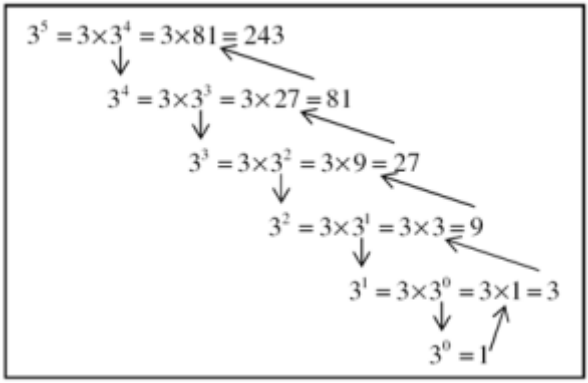
Dengan melihat kemiripan cara kerja serta kode dari fungsi faktorial dan kali, kita dapat melihat bagaimana fungsi rekursif memiliki dua ciri khas:

1. Fungsi rekursif selalu memiliki kondisi yang menyatakan kapan fungsi tersebut berhenti. Kondisi ini harus dapat dibuktikan akan tercapai, karena jika tidak tercapai maka kita tidak dapat membuktikan bahwa fungsi akan berhenti, yang berarti algoritma kita tidak benar.
2. Fungsi rekursif selalu memanggil dirinya sendiri sambil mengurangi atau memecahkan data masukan setiap panggilannya. Hal ini penting diingat, karena tujuan utama dari rekursif ialah memecahkan masalah dengan mengurangi masalah tersebut menjadi masalah-masalah kecil.

Setiap fungsi rekursif yang ada harus memenuhi kedua persyaratan di atas untuk memastikan fungsi rekursif dapat berhenti dan memberikan hasil. Kebenaran dari nilai yang dihasilkan tentu saja memerlukan pembuktian dengan cara tersendiri. Tetapi sebelum masuk ke analisa dan pembuktian fungsi rekursif, mari kita lihat kegunaan dan contoh-contoh fungsi rekursif lainnya lagi.

10.3 Algoritma Rekursif

Untuk memahami proses rekursif yang terjadi dalam sebuah fungsi rekursif, perhatikan contoh sederhana di bawah ini. Contoh di bawah ini menyajikan satu fungsi untuk menghitung harga pangkat suatu nilai bilangan bulat misalnya 3^5, berdasarkan hubungan rekurens seperti dijelaskan diatas, maka proses rekursif akan tampak pada gambar berikut ini:



Gambar: Ilustrasi Penghitungan pangkat secara rekursif

Dari definisi tersebut, statemen pertama menunjukkan nilai yang utama dari fungsi, dan statemen kedua menunjukan perulangan penurunan dari n yaitu n-1.

Selain fungsi, prosedur juga dapat dilakukan operasi rekursif. Sebagai contoh penggunaan proses rekursif pada prosedur adalah prosedur pencarian biner (binary search). Dalam beberapa hal rekursif kurang efisien dibandng proses iterasi. Dalam artian pemecahan secara rekursif dan secara iterasi mempunyai keuntungan dan kekurangan yang bisa saling diperbandingkan. Adalah cukup sulit untuk menentukan mana yang paling sederhana, paling jelas, paling efisien dan paling mudah dibanding yang lain. Bisa ditambahkan, pemilihan secara iteratif maupun rekursif boleh dikatakan merupakan kesenangan seorang programmer sesuai dengan keinginannya.

Algoritma

```
Judul : Algoritma Pangkat
secara rekursif

Kamus :
     X,Y : Integer
Algoritma:
(fungsi pangkat secara
rekursif)
FUNCTION
Pangkat(X:Integer,Y:Integer)
Integer
   IF Y = 0 THEN
      Pangkat ← 1
   ELSE
      Pangkat←X * Pangkat(X,Y-
1)
   ENDIF
END FUNCTION
(Bagian Pemanggil)
   INPUT X, Y
   OUTPUT Pangkat(X,Y)
```

10.4 Algoritma Rekursif dalam Bahasa Pemrograman

```
#include <iostream>
using namespace std;
void cetak(int n)
{
if(n<=4)
cetak(n+1);
cout<<n<<endl;
}

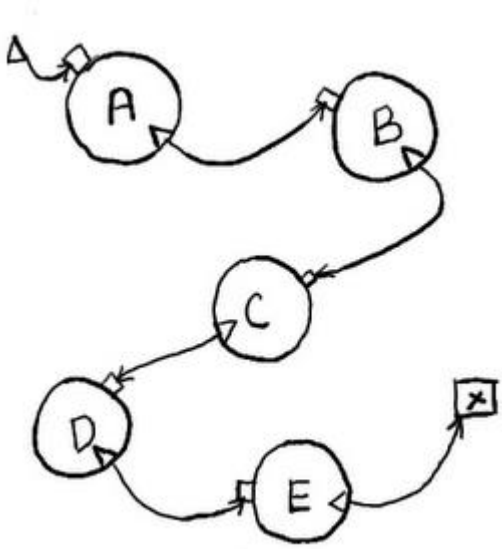

int main(int argc, char *argv[])
{
cout<<“Hasil dengan cara menggunakan rekursif: “;
cetak(1);
system(“PAUSE”);
return EXIT_SUCCESS;
}
```

Pertemuan 11 : Single Linked List

Single Linked List adalah salah satu bentuk struktur data, berisi kumpulan data
(node) yang tersusun secara sekuensial, saling sambungmenyambung, dinamis dan terbatas.
- Linked List sering disebut juga Senarai Berantai
- Linked List saling terhubung dengan bantuan variabel pointer
- Masing-masing data dalam Linked List disebut dengan node (simpul) yang
menempati alokasi memori secara dinamis dan biasanya berupa struct yang terdiri dari beberapa field.

Single Linked List adalah sebuah LINKED LIST yang menggunakan sebuah variabel pointer saja untuk menyimpan banyak data dengan metode LINKED LIST, suatu daftar isi yang saling berhubungan.

Ilustrasi single LINKED LIST:



Linked List

Pada gambar di atas, data terletak pada sebuah lokasi dalam sebuah memory, tempat yang disediakan memory untuk menyimpan data disebut node ? simpul, setiap node memiliki pointer (penunjuk) yang menunjuk ke node berikutnya sehingga terbentuk suatu untaian yang disebut single LINKED LIST. Bila dalam single LINKED LIST pointer hanya dapat bergerak ke satu arah saja, maju / mundur, kanan / kiri, sehingga pencarian datanya juga hanya satu arah saja.

Ada 2 Tipe Single Linked List yaitu

- Single Linked List Circular
- Single Linked List Non Circular

1. Single Linked List Circular

Single Linked List Circular adalah Single Linked List yang pointer nextnya menunjuk pada dirinya sendiri. Jika Single Linked List tersebut terdiri dari beberapa node,
maka pointer next pada node terakhir akan menunjuk ke node terdepannya.

Pengertian:
Single : artinya field pointer-nya hanya satu buah saja dan satu arah.
Circular : artinya pointer next-nya akan menunjuk pada dirinya sendiri sehingga berputar

Ilustrasi Single Linked List Circular
- Setiap node pada linked list mempunyai field yang berisi pointer ke node
  berikutnya, dan juga memiliki field yang berisi data.

- Pada akhir linked list, node terakhir akan menunjuk ke node terdepan sehingga linked list tersebut berputar. Node terakhir akan menunjuk lagi
ke head.

PEMBUATAN SINGLE LINKED LIST CIRCULAR
Deklarasi node
Dibuat dari struct berikut ini:

```
typedef struct TNode{
int data;
TNode *next;
};
```

Penjelasan:
- Pembuatan struct bernama TNode yang berisi 2 field, yaitu field data bertipe integer dan field next yang bertipe pointer dari TNode
- Setelah pembuatan struct, buat variabel haed yang bertipe pointer dari TNode yang berguna sebagai kepala linked list.

Pembentukan node baru
Digunakan keyword new yang berarti mempersiapkan sebuah node baru berserta alokasi memorinya.

```
TNode *baru;
baru = new TNode;
baru->data = databaru;
baru->next = baru;
```

SINGLE LINKED LIST CIRCULAR MENGGUNAKAN HEAD
- Dibutuhkan satu buah variabel pointer: head
- Head akan selalu menunjuk pada node pertama

Deklarasi Pointer Penunjuk Kepala Single Linked List
Manipulasi linked list tidak bisa dilakukan langsung ke node yang dituju,
melainkan harus melalui node pertama dalam linked list. Deklarasinya sebagai berikut:

```
TNode *head;
```

Fungsi Inisialisasi Single LinkedList

```
void init(){
head = NULL;
}
```

Function untuk mengetahui kosong tidaknya Single LinkedList

```
int isEmpty(){
if(head == NULL) return 1;
else return 0;
}
```

PENAMBAHAN DATA
Penambahan data di depan
Penambahan node baru akan dikaitan di node paling depan, namun pada saat
pertama kali (data masih kosong), maka penambahan data dilakukan pada
head nya.
Pada prinsipnya adalah mengkaitkan data baru dengan head, kemudian head
akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan. Untuk menghubungkan node terakhir dengan node
terdepan dibutuhkan pointer bantu.

```
void insertDepan(int databaru){
```

```
TNode *baru,*bantu;
baru = new TNode;
baru->data = databaru;
baru->next = baru;

if(isEmpty()==1){
head=baru;
head->next=head;
}
else {
bantu = head;
while(bantu->next!=head){
bantu=bantu->next;
}
baru->next = head;
head = baru;
bantu->next = head;
}
cout<<"Data masuk\n";
}
```

Penambahan data di belakang Penambahan data dilakukan di belakang, namun pada saat pertama kali data langsung ditunjuk pada head-nya. Penambahan di belakang lebih sulit karena kita membutuhkan pointer bantu untuk mengetahui data terbelakang, kemudian dikaitkan dengan data baru. Untuk mengetahui data terbelakang perlu digunakan perulangan.

```
void insertBelakang (int databaru)
{
TNode *baru,*bantu;
baru = new TNode;
baru->data = databaru;
baru->next = baru;
if(isEmpty()==1){
```

```
head=baru;
head->next=head;
}
else {
bantu = head;
while(bantu->next != head){
bantu=bantu->next;
}
bantu->next = baru;
baru->next = head;
}
cout<<"Data masuk\n";
}
```

“Bagaimana dengan penambahan di tengah?”

MENAMPILKAN DATA
Function untuk menampilkan isi single linked list

```
void tampil(){ TNode *b;
b = head;
if(isEmpty()==0)
{
do
{
cout<data<<" ";
b=b->next;
}
while(b!=head);
cout<<<"Masih kosong\n";
}
```

- Function di atas digunakan untuk menampilkan semua isi list, di mana linked list ditelusuri satu-persatu dari awal node sampai akhir node. Penelusuran ini dilakukan dengan menggunakan suatu variabel node bantu, karena pada prinsipnya variabel node head yang menjadi tanda

awal list tidak boleh berubah/berganti posisi.
- Penelusuran dilakukan terus sampai node terakhir ditemukan menunjuk ke head lagi. Jika belum sama dengan head, maka node bantu akan berpindah ke node selanjutnya dan membaca isi datanya dengan menggunakan field next sehingga dapat saling berkait.
- Jika head masih NULL berarti data masih kosong!

PENGHAPUSAN DATA
Function untuk menghapus data terdepan

```
void hapusDepan ()
{ TNode *hapus,*bantu;
if (isEmpty()==0)
{
int d;
hapus = head; d = head->data;
if(head->next != head){
bantu = head;
while(bantu->next!=head){
bantu=bantu->next;
}
head = head->next;
delete hapus;
bantu->next = head;
}else{
head=NULL;
}
cout<<<" terhapus\n";
}
else cout<<"Masih kosong\n";
}
```

- Function di atas akan menghapus data teratas (pertama) yang ditunjuk oleh head pada linked list
- Penghapusan node tidak boleh dilakukan jika keadaan node sedang

ditunjuk oleh pointer, maka harus ditampung dahulu pada variabel hapus dan barulah kemudian menghapus variabel hapus dengan menggunakan perintah delete.
- Sebelum data terdepan dihapus, head harus ditunjukkan ke data sesudahnya terlebih dahulu sehingga data setelah head lama akan menjadi head baru (data terdepan yang baru).
- Jika head masih NULL maka berarti data masih kosong!

Penghapusan data di belakang:

```
void hapusBelakang()
{ TNode *hapus,*bantu;
if (isEmpty()==0)
{
int d;
hapus = head;
if(head->next == head){
head = NULL;
}
else
{
bantu = head;
while(bantu->next->next != head){
bantu = bantu->next;
}
hapus = bantu->next;
d = bantu->data;
bantu->next = head;
delete hapus;
}
cout<<<" terhapus\n";
}
else
cout<<"Masih kosong\n";
}
```

- Membutuhkan pointer bantu dan hapus.
- Pointer hapus digunakan untuk menunjuk node yang akan dihapus, dan pointer bantu digunakan untuk menunjuk node sebelum node yang dihapus.
- Pointer bantu akan digunakan untuk menunjuk ke nilai NULL.
- Pointer bantu akan selalu bergerak bersama dengan pointer hapus tapi letak pointer bantu harus selalu dibelakang pointer hapus.

Function untuk menghapus semua elemen Linked List

```
void clear(){ TNode *bantu,*hapus;
bantu = head;
while(bantu->next!=head){
hapus = bantu;
bantu = bantu->next;
delete hapus;
}
head = NULL;
}
```

SINGLE LINKED LIST MENGGUNAKAN HEAD DAN TAIL
- Dibutuhkan dua buah variabel pointer: head dan tail
- Head akan selalu menunjuk pada node pertama, sedangkan tail akan selalu menunjuk pada node terakhir.

Inisialisasi LinkedList

```
TNode *head, *tail;
```

Fungsi Inisialisasi LinkedList

```
void init(){
head = NULL;
tail = NULL;
```

```
}
```

Function untuk mengetahui kosong tidaknya LinkedList

```
int isEmpty(){
if(tail == NULL) return 1;
else return 0;
}
```

PENAMBAHAN DATA
Pengkaitan node baru ke linked list di depan
Penambahan data baru di depan akan selalu menjadi head.

```
void insertDepan(int databaru){
TNode *baru;
baru = new TNode;
baru->data = databaru;
baru->next = baru;
if(isEmpty()==1){
head=baru;
tail=baru;
head->next=head;
tail->next=tail;
}
else {
baru->next = head;
head = baru;
tail->next = head;
}
cout<<"Data masuk\n";
}
```

Penambahan Data di belakang Pada penambahan data di belakang, data akan selalu dikaitkan dengan tail, karena tail terletak di node paling belakang. Setelah dikaitkan dengan node baru, maka node baru tersebut

akan menjadi tail.

```
void tambahBelakang(int databaru){ TNode *baru;
baru = new TNode;
baru->data = databaru;
baru->next = baru;
if(isEmpty()==1){
head=baru;
tail=baru;
head->next=head;
tail->next=tail;
}
else
{
tail->next = baru;
tail = baru;
tail->next = head;
}
cout<<"Data masuk\n";
}
```

Kelebihan dari Single Linked List dengan Head & Tail adalah pada penambahan data di belakang, hanya dibutuhkan tail yang mengikat node baru saja tanpa harus menggunakan perulangan pointer bantu.

Function untuk menampilkan isi linked list:

```
void tampil(){ TNode *b;
b = head; if(isEmpty()==0)
{
do
{ cout<data<<" ";
b=b->next;
}
while(b!=tail->next);
```

```
cout<<<"Masih kosong\n";
}
```

Pada prinsipnya sama dengan function tampil sebelumnya.

Function untuk menghapus data di depan

```
void hapusDepan(){ TNode *hapus;
if (isEmpty()==0){ int d;
hapus = head;
d = head->data;
if(head != tail){
hapus = head;
head = head->next;
tail->next = head;
delete hapus;
}else{
head=NULL;
tail=NULL;
}
cout<<<" terhapus\n";
}
else cout<<"Masih kosong\n";
}
```

- Function di atas akan menghapus data terdepan (pertama) yang ditunjuk oleh head pada linked list
- Penghapusan node tidak boleh dilakukan jika keadaan node sedang ditunjuk oleh pointer, maka harus dilakukan penunjukkan terlebih dahulu dengan variabel hapus pada head, kemudian dilakukan pergeseran ke node berikutnya sehingga data setelah head menjadi head baru, kemudian menghapus variabel hapus dengan menggunakan perintah delete.
- Jika tail masih NULL maka berarti data masih kosong!

Function untuk menghapus data di belakang:

Dengan menggunakan Single Linked List ber-Head dan Tail, pengahapusan data di belakang akan mudah dilakukan, tidak seperti pada Single Linked List hanya ber-Head saja.

```
void hapusBelakang(){ TNode *hapus,*bantu;
if (isEmpty()==0){ int d;
if(head == tail){ d = tail->data;
head = NULL;
tail = NULL;
}
else
{
bantu = head;
while(bantu->next != tail){
bantu = bantu->next;
}
hapus = tail;
tail = bantu;
d = hapus->data;
tail->next = head;
delete hapus;
}
cout<<<" terhapus\n";
}
else cout<<"Masih kosong\n";
}
```

- Function di atas akan menghapus data terbelakang (terakhir) yang ditunjuk oleh tail pada linked list
- Penghapusan node tidak boleh dilakukan jika keadaan node sedang ditunjuk oleh pointer, maka harus dilakukan penunjukkan terlebih dahulu dengan variabel hapus pada tail, kemudian dibutuhkan pointer bantu untuk membantu pergeseran dari head ke node berikutnya sampai

sebelum tail, sehingga tail dapat ditunjukkan ke bantu tersebut, dan bantu tersebut akan menjadi tail yang baru. Setelah itu hapus variabel hapus dengan menggunakan perintah delete.
- Jika tail masih NULL maka berarti data masih kosong!

Function untuk menghapus semua elemen LinkedList

```
void clear(){ TNode *bantu,*hapus;
if(isEmpty() == 0){ bantu = head;
while(bantu->next!=head){
hapus = bantu;
bantu = bantu->next;
delete hapus;
}
head = NULL;
tail = NULL;
}
}
```

- Menggunakan pointer bantu yang digunakan untuk bergerak sepanjang list, dan menggunakan pointer hapus yang digunakan untuk menunjuk node-node yang akan dihapus.
- Pada saat pointer hapus menunjuk pada node yang akan dihapus, pointer
bantu akan bergerak ke node selanjutnya, dan kemudian pointer hapus akan didelete.

2. Single Linked List Non Circular

Pengertian:
Single : artinya field pointer-nya hanya satu buah saja dan satu arah.
Linked List : artinya node-node tersebut saling terhubung satu sama lain.
Ilustrasi Linked List
- Setiap node pada linked list mempunyai field yang berisi pointer ke

node
berikutnya, dan juga memiliki field yang berisi data.
- Pada akhir linked list, node terakhir akan menunjuk ke NULL yang akan
digunakan sebagai kondisi berhenti pada saat pembacaan isi linked list

Contoh program single linked list non circular tambah list di depan :

```
# include<stdio.h>

# include<stdlib.h>

# include<conio.h>

# include<iostream.h>

# include<ctype.h>

# include<string.h>


struct simpul

{

        int angka;

        struct simpul*berikut;

} ;


struct simpul *awal=NULL;

int bil;


void tambah_list_didepan(int info);
```

```
void isi_list();
void tampil_list();
void hapus_list();

void main ()

{

        clrscr();
        isi_list();
        clrscr();
        tampil_list();
        hapus_list();
        getch();
}
void tambah_list_didepan(int info)
{
        struct simpul *baru;
        baru=(struct simpul *)malloc(sizeof(struct simpul));
        baru->angka=info;

        baru->berikut=awal;
```

```
        awal=baru;
}

void isi_list()
{
        char jawab;
        do
        {
        clrscr();
        cout<<"\ninput bilangan :";
        cin>>bil;
        tambah_list_didepan(bil);
        cout<<"\ntambah data Y/T :"  ;
        cin>>jawab;
        }
        while (toupper(jawab)!='T');

}
void tampil_list()
{
```

```
        struct simpul* baca;
        int i;
        baca=awal;
        i=1;

        while(baca!=NULL)

        {
                cout<<"\nbilangan ke-"<<i<<"yang dibaca :"<<baca->angka;
                i++;
                baca=baca->berikut;
        }

}

void hapus_list()

{
        struct simpul*hapus;
        hapus=awal;
        while(hapus!=NULL)
        {
```

```
                awal=hapus->berikut;
                free(hapus);
                hapus=awal;
        }
}
```

Contoh program single linked list non circular tambah di belakang :

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

struct siswa
{
 char nrp[8],nama[20];
 int umur;
 float ipk;
};

struct simpul
{
```

```
 char nrp[8],nama[20];
 int umur;
 float ipk;
 struct simpul *berikut;
};

struct simpul *awal = NULL, *akhir = NULL;
struct siswa mhs;

void tambah_list_dibelakang(struct siswa info);
void isi_list();
void tampil_list();
void hapus_list();

void main()
{
 clrscr();
 isi_list();
 clrscr();
 tampil_list();
 hapus_list();
 getch();
```

```
}

void tambah_list_dibelakang(struct siswa info)
{
 struct simpul *baru;

 baru = (struct simpul *) malloc(sizeof(struct simpul));
 strcpy (baru -> nrp,info.nrp);
 strcpy (baru -> nama,info.nama);
 baru -> umur = info.umur;
 baru -> ipk  = info.ipk;

 if (awal == NULL)
 {
  awal = baru;
 }
 else
 {
  akhir -> berikut = baru;
 }
 akhir = baru;
 akhir -> berikut = NULL;
```

```
}

void isi_list()
{
 char jawab;

 do
 {
  clrscr();
  cout<<"NRP   : ";cin>>mhs.nrp;
  cout<<"Nama  : ";cin>>mhs.nama;
  cout<<"Umur  : ";cin>>mhs.umur;
  cout<<"IPK   : ";cin>>mhs.ipk;
  tambah_list_dibelakang(mhs);
  cout<<"\nTambah Data [Y/T]? : ";
  cin>>jawab;

 }
 while (toupper(jawab) != 'T');
}

void tampil_list()
```

```
{
 struct simpul *baca;
 int i;

 baca = awal;
 i = 1;

 while (baca != NULL)
 {
  cout<<"Data Ke : "<<i<<endl;
  cout<<"NRP     : "<<baca -> nrp<<endl;
  cout<<"Nama    : "<<baca -> nama<<endl;
  cout<<"Umur    : "<<baca -> umur<<endl;
  cout<<"IPK     : "<<baca -> ipk<<endl;
  cout<<endl;
  i++;
  baca = baca -> berikut;
 }
}

void hapus_list()
{
```

```
 struct simpul *hapus;

 hapus = awal;
 while (hapus != NULL)
 {
  awal = hapus -> berikut;
  free (hapus);
  hapus = awal;
 }
}
```

Contoh program single linked list non circular tambah list ditengah :

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

struct siswa
{
 char nama[20];
 int umur;
```

```
};

struct simpul
{
 char nama[20];
 int umur;

 struct simpul *berikut;
};

struct simpul *awal=NULL,*akhir=NULL;
struct siswa mhs;

void tambah_list_belakang(struct siswa info);
void isi_list();
void sisip_list(struct simpul *first,struct siswa info,char posisi[20]);
void sisip_isi_list();
void tampil_list();
void hapus_list();

void main()
```

```
{
 clrscr();
 isi_list();

 clrscr();
 tampil_list();
 sisip_isi_list();
 tampil_list();
 hapus_list();
 getch();
}

void tambah_list_dibelakang(struct siswa info)
{
 struct simpul *baru;

 baru=(struct simpul*)malloc(sizeof(struct simpul));
 strcpy(baru->nama,info.nama);
 baru->umur=info.umur;

 if (awal==NULL)
  {
```

```
   awal=baru;
  }
 else
  {
   akhir->berikut=baru;
  }
 akhir=baru;

 akhir ->berikut=NULL;
}

void isi_list()
{
 char jawab;

 do
 {
  clrscr();
  cout<<"Nama: ";gets(mhs.nama);
  cout<<"Umur: ";cin>>mhs.umur;

  tambah_list_dibelakang(mhs);
```

```
  cout<<"\nTambah Data [Y/T]: ";
  cin>>jawab;
 }
 while (toupper(jawab)!='T');
}
void sisip_list(struct simpul *first,struct siswa info,char posisi[20])
{
struct simpul *bantu,*baru;

baru = new simpul;
strcpy(baru->nama,info.nama);
baru->umur=info.umur;
bantu=first;

do
{
if (strcmp(posisi,bantu->nama)!=0)  {bantu=bantu->berikut;}
}
while (bantu!=NULL && strcmp(posisi,bantu->nama)!=0);

baru->berikut=bantu->berikut;
```

```
bantu->berikut=baru;
}


void sisip_isi_list()
{
char cari[20];
struct siswa ganti;

cout<<"\nInput Data Baru :";
cout<<"\nNama :";gets(ganti.nama);
cout<<"\nUmur :";cin>>ganti.umur;
cout<<"\n";
cout<<"\nData Akan Disisipkan Setelah Nama: ";
gets(cari);
sisip_list(awal,ganti,cari);
}

void tampil_list()
{
 struct simpul *baca;
 int i;
```

```
 baca=awal;
 i=1;

 while(baca!=NULL)
 {

  cout<<"\nNama : "<<baca->nama;
  cout<<"\nUmur : "<<baca->umur;
  cout<<"\n";
  i++;
  baca=baca->berikut;
 }
}

void hapus_list()
{
 struct simpul *hapus;

 hapus=awal;
 while(hapus!=NULL)
 {
```

```
  awal=hapus->berikut;
  free(hapus);
  hapus=awal;
 }
}
```

Contoh program single linked list non circular untuk menghapus semua list ataupun salah satu dengan menggunakan menu:

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

struct nasabah
{
 char norek[12];
 char nama[25];
 char alamat[50];
 double saldo;
};

struct simpul
```

```
{
 char norek[12];
 char nama[25];
 char alamat[50];
 double saldo;
 struct simpul *berikut;
};

struct simpul *awal=NULL, *akhir=NULL;
struct nasabah bank;

void tambah_list_dibelakang(struct nasabah info);
void isi_list();
void sisip_list(struct simpul *first,struct nasabah info,char posisi[20]);
void sisip_isi_list();
void tampil_list();
void hapus_simpul(char info);
void hapus_data();
void hapus_list();
void menu();

void main()
```

```
{
 clrscr();
 menu();
 getch();
}
void menu()
{
 int pil;
 clrscr();
 cout<<"Programmer : Shandy Johan_6311171"<<endl<<endl;
 cout<<"Pilih Transaksi  : "<<endl;
 cout<<"          1. Isi List"<<endl;
 cout<<"          2. Sisip List"<<endl;
 cout<<"          3. Tampil List"<<endl;
 cout<<"          4. Hapus Salah Satu"<<endl;
 cout<<"          5. Hapus Semua List"<<endl;
 cout<<"          6. Exit"<<endl<<endl;
 cout<<"Tentukan Pilihan : ";
 cin>>pil;
 switch (pil)
 {
  case 1:
```

```
      isi_list();
      break;
  case 2:
      sisip_isi_list();
      break;
  case 3:
      tampil_list();
      break;
  case 4:
      hapus_data();
      break;
  case 5:
      hapus_list();
      break;
  case 6:
      clrscr();
      gotoxy(21,13);cout<<"Terima Kasih Telah Menggunakan Program
Ini";
      break;
 }
}

void isi_list()
```

```
{
 char jawab;

 do
 {
  clrscr();
  cout<<"No Rekening : ";
  cin>>bank.norek;
  cout<<"Nama        : ";
  cin>>bank.nama;
  cout<<"Alamat      : ";
  cin>>bank.alamat;
  cout<<"Saldo Awal  : ";
  cin>>bank.saldo;
  tambah_list_dibelakang(bank);
  cout<<"\n\nTambah Data [Y/T] : ";
  cin>>jawab;
 }
 while (toupper(jawab)!='T');
 menu();
}
```

```
void tambah_list_dibelakang(struct nasabah info)
{
 struct simpul *baru;

 baru = (struct simpul *) malloc (sizeof(struct simpul));
 strcpy (baru -> norek,info.norek);
 strcpy (baru -> nama,info.nama);
 strcpy (baru -> alamat,info.alamat);
 baru -> saldo = info.saldo;

 if (awal == NULL)
 {
  awal = baru;
 }
  else
  {
   akhir -> berikut = baru;
  }
 akhir = baru;
 akhir -> berikut = NULL;
}
```

```
void tampil_list()
{
 clrscr();
 struct simpul *baca;
 int i;

 baca = awal;
 i = 1;

 while (baca != NULL)
 {
  cout<<"Data Ke : "<<i<<endl;
  cout<<"No Rekening  : "<<baca -> norek<<endl;
  cout<<"Nama         : "<<baca -> nama<<endl;
  cout<<"Alamat       : "<<baca -> alamat<<endl;
  cout<<"Saldo Awal   : "<<baca -> saldo<<endl;
  cout<<endl;
  i++;
  baca = baca -> berikut;
 }
 getch();
 menu();
```

```
}

void hapus_list()
{
 struct simpul *hapus;
 hapus = awal;
 while (hapus != NULL)
 {
  awal = hapus -> berikut;
  free(hapus);
  hapus = awal;
 }
 menu();
}


void sisip_list(struct simpul *first, struct nasabah info, char posisi[20])
{
 struct simpul *bantu,*baru;

 baru = new simpul;
 strcpy (baru -> norek,info.norek);
```

```
 strcpy (baru -> nama,info.nama);
 strcpy (baru -> alamat,info.alamat);
 baru-> saldo = info.saldo;
 bantu = first;

 do
 {
  if (strcmp(posisi,bantu->norek)!=0)
  {
   bantu = bantu->berikut;
  }
 }
 while (bantu !=NULL & strcmp(posisi,bantu->norek)!=0);

 baru->berikut = bantu -> berikut;
 bantu -> berikut = baru;
}

void sisip_isi_list()
{
 char cari[20];
 struct nasabah ganti;
```

```
  clrscr();
  cout<<"Input Data Baru"<<endl;
  cout<<"No Rekening : ";
  cin>>ganti.norek;
  cout<<"Nama        : ";
  cin>>ganti.nama;
  cout<<"Alamat      : ";
  cin>>ganti.alamat;
  cout<<"Saldo Awal  : ";
  cin>>ganti.saldo;
  cout<<endl;
  cout<<"Data Akan Disisipkan Setelah No Rekening : ";
  gets(cari);
  sisip_list(awal,ganti,cari);
  menu();
}

void hapus_simpul(char info[20])
{
 struct simpul *bantu,*hapus;

 if (awal==NULL) { cout<<"\n Data List Kosong ";}
```

```
else
{
 if (strcmp(awal->norek,info)==0)
 {
  hapus=awal;
  awal=hapus->berikut;
  free(hapus);
 }
  else
  {
   bantu=awal;
   while ((strcmp(info,bantu->berikut->norek)!=0) && (bantu->berikut!=NULL))
   {
    bantu=bantu->berikut;
   }
   hapus=bantu->berikut;
   if (hapus!=NULL)
   {
    if (hapus!=akhir) { bantu->berikut=hapus->berikut;}
    else
    {
     akhir=bantu;
```

```
      akhir->berikut=NULL;
     }
     free(hapus);
    }
   }
  }
}


void hapus_data()
{
 char cari[20];
 char jawab;

 clrscr();

 cout<<"\nAda data yang akan dihapus Y/T :";cin>>jawab;
 if (toupper(jawab)=='Y')
 {
  cout<<"\nNo.Rekening yang akan dihapus :";
  gets(cari);
  hapus_simpul(cari);
```

```
 menu();
 }
}
```

Pertemuan 12 : Double Linked List

- **Senarai berantai ganda (*doubly linked-list*) atau senarai dua arah (*two-way list*), setiap simpulnya mempunyai 2 buah pointer, dengan pointer pertama menunjuk ke simpul sebelumnya dan pointer kedua menunjuk simpul sesudahnya.**
- **Jika senarainya mempunyai simpul kepala, dinamakan senarai berantai ganda berkepala (*headed doubly linked-list*).**



disebut ‘simpul ganda’ karena simpul ini mempunyai dua buah penunjuk, yaitu ‘kiri’ dan ‘kanan’. Artinya simpul ini dapat menunjuk ke dua buah simpul sekaligus yang serupa dengan dirinya baik yang terletak disebelah kirinya maupun yang di sebelah kanannya.

Deklarasi

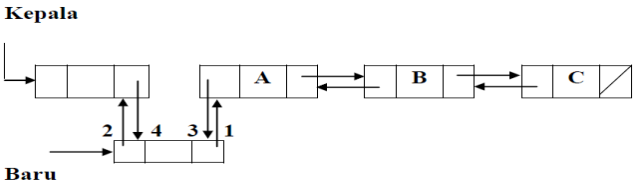
```
Type Simpul = ^Data;
        Data = record
                Info : char;
                kiri, kanan : Simpul
        End;
```

```
Var Kepala : Simpul;
```

Inisialisasi

```
Procedure INISIALISASI (var Kepala : Simpul);
Begin
        New(Kepala);
        With Kepala^ do
          begin
                Kiri := nil;
                Kanan := nil
          end
End;
```

Penambahan simpul sesudah simpul kepala




```
Baru^.Kanan  :=  Kepala^.Kanan;
Baru^.Kiri  :=  Kepala;
Kepala^.Kanan^.Kiri  :=  Baru;
Kepala^.Kanan  :=  Baru;
```

Penambahan simpul baru di tengah

```
Baru^.Kanan := Bantu^.Kanan; Baru^.Kiri := Bantu;

Bantu^.Kanan^.Kiri := Baru; Bantu^.Kanan := Baru;
```

Penambahan simpul baru di akhir

Untuk penambahan simpul baru yang ditempatkan di akhir senarai maka pointer Bantu kita tempatkan di simpul akhir, sehingga penambahannya adalah:

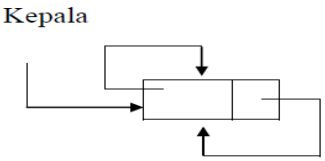
```
Bantu^.Kanan := Baru;

Baru^.Kiri := Bantu;
```

Linked List ganda berputar dan berkepala

- Pada senarai berantai ganda, pointer Kiri dari simpul pertama dan pointer Kanan dari simpul terakhir masih bernilai nil.
- Bila pointer Kiri kita arahkan ke simpul terakhir dan pointer Kanan dari simpul terakhir kita arahkan ke simpul kepala, senarai yang demikian dinamakan senarai berantai ganda berputar dan berkepala.

Inisialisasi



Prosedur inisialisasi simpul kepala tersebut sebagai berikut:

```
Procedure INISIALISASI_SBGBB (var Kepala  :  Simpul);
Begin
      New(Kepala);
      With  Kepala^  do
          begin
              Kiri := Kepala;
              Kanan := Kepala
          end
End;
```

Contoh

```
Procedure TAMBAH_SIMPUL (var Kepala : Simpul; Elemen : char);
Var Baru, Bantu : Simpul;
Begin
New(Baru);
Baru^.Info := Elemen;
If Elemen > Kepala^.Kiri^.Info then
        Begin
            Baru^.Kiri := Kepala^.Kiri;
            Baru^.Kanan := Kepala;
            Kepala^.Kiri^.Kanan := Baru;
            Kepala^.Kiri := Baru
        End;
Else
        begin
            Bantu := Kepala;
            While Bantu^.Kanan^.Info < Elemen do
                Bantu := Bantu^.Kanan;
            Baru^.Kanan := Bantu^.Kanan;
            Bantu^.Kanan^.Kiri := Baru;
            Bantu^.Kanan := Baru;
            Baru^.Kiri := Bantu
```

```
        end
End;
```

Pertemuan 13 : Binary Tree : Penelusuran

Binary Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkanhubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya (disebut subtree).

Dalam tree terdapat jenis-jenis tree yang memiliki sifat khusus, diantaranya adalah binary tree.

Binary tree adalah suatu tree dengan syarat bahawa tiap node (simpul) hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Tiap node dalam binary treee boleh memiliki paling banyak dua child (anak simpul), secara khusus anaknya dinamakan kiri dan kanan.

Binary Tree merupakan himpunan vertex-vertex yang terdiri dari 2 subtree (dengan disjoint) yaitu subtree kiri dan subtree kanan. Setiap vertex dalam binary tree mempunyai derajat keluar max = 2.

Sebuah pohon biner adalah grafik asiklis yang terhubung dimana setiap tingkatan dari susut tidak lebih dari 3. Ini dapat ditunjukkan bahwa dalam pohon biner manapun, terdapat persis dua atau lebih simpul dengan tingkat satu daripada yang terdapat dengan tingkat tiga, tetapi bisa terdapat angka apa saja dari simpul dengan tingkat dua. Sebuah pohon biner berakar merupakan sebuah grafik yang mempunyai satu dari sudutnya dengan tingkat tidak lebih dari dua sebagai akar.

Dengan akar yang dipilih, setiap sudut akan memiliki ayah khusus, dan diatas dua anak bagaimanapun juga, sejauh ini terdapat keterbatasan informasi untuk membedakan antara anak kiri atau kanan. Jika kita membuang keperluan yang tak terkoneksi, membolehkan bermacam koneksi dalam komponen di grafik, kita memanggil struktur sebuah hutan.

Sebuah jalan lain untuk mendefinisikan pohon biner melalui definisi rekursif pada grafik langsung. Sebuah pohon biner dapat berarti :

- Sebuah sudut tunggal.

- Sebuah graf yang dibentuk dengan mengambil dua pohon biner, menambahkan sebuah sudut, dan menambahkan sebuah panah langsung dari sudut yang baru ke akar dai setiap pohon biner.

Pohon biner dapat dikontruksi dari bahasa pemrogaraman primitif dalam berbagai cara. Dalam bahasa yang menggunakan records dan referensi. Pohon biner secara khas dikontruksi dengan mengambil sebuah struktur simpul pohon yang memuat beberapa data dan referensi ke anak kiri dan anak kanan.

Kadang-kadang itu juga memuat sebuah referensi ke ayahnya yang khas. Jika sebuah simpul mempunyai kurang dari dua anak, beberapa penunjuk anak diaatur kedalam nilai nol khusus atau kesebuah simpul sentinel.

Pohon biner dapat juga disimpan sebagai struktur data implisit dalam array, dan jika pohon tersebut merupakan sebuah pohon biner lengkap, metode ini tidak boros tempat. Dalam penyusunan yang rapat ini, jika sebuah simpul memiliki indeks i, anaknya dapat ditemukan pada indeks ke-2i+1 dan 2i+2, meskipun ayahnya (jika ada) ditemukan pada indeks lantai ((i-1)/2) (asumsikan akarnya memiliki indeks kosong). Metode ini

menguntungkan dari banyak penyimpanan yang rapat dan memiliki referensi lokal yang lebih baik, teristimewa selama sebuah preordeer traversal.

Istilah-istilah dalam pohon

1. Predesesor

Node yang berada diatas node tertentu.

(contoh : B predesesor dari E dan F)

2. Succesor

Node yang berada dibawah node tertentu.

(contoh : E dan F merupakan succesor dari B)

3. Ancestor

Seluruh node yang terletak sebelum node tertentu dan

terletak pada jalur yang sama.

(contoh : A dan B merupakan ancestor dari F)

Istilah – istilah Dalam Pohon

4. Descendant

Seluruh node yang terletak sesudah node tertentu

dan terletak pada jalur yang sama.

(contoh : F dan B merupakan ancestor dari A)

5. Parent

Predesesor satu level diatas satu node

(contoh : B merupakan parent dari F)

6. Child

Succesor satu level dibawah satu node

(contoh : F merupakan child dari B)

7. Sibling

Node yang memiliki parent yang sama dengan satu node (contoh : E dan F adalah sibling)

8. Subtree

Bagian dari tree yang berupa suatu node beserta descendant-nya (contoh : Subtree B, E, F dan Subtree D, G, H)

9. Size

Banyaknya node dalam suatu tree (contoh : gambar tree diatas memiliki size = 8)

10. Height

Banyaknya tingkat/level dalam suatu tree (contoh : gambar tree diatas memiliki height = 3)

11. Root (Akar)

Node khusus dalam tree yang tidak memiliki predesesor (Contoh : A)

12. Leaf (Daun)

Node-node dalam tree yang tidak memiliki daun (contoh : Node E,F,C,G,H)

13. Degree (Derajat)

Banyaknya child yang dimiliki oleh suatu node

(contoh : Node A memiliki derajat 3, node B memiliki derajat 2)

· Istilah pada pohon Binar

a. Pohon Biner Penuh (Full Binary Tree)

Semua simpul (kecuali daun) memiliki 2 anak dan tiap cabang memiliki panjang ruas yang sama.

b. Pohon Biner Lengkap (Complete Binary Tree)

Hampir sama dengan Pohon BinerPenuh, semua simpul (kecualidaun) memiliki 2 anak tetapi tiap cabang memiliki panjang ruas berbeda.

c. Pohon Biner Similer

Dua pohon yang memiliki struktur yang sama tetapi informasinya berbeda.

d. Pohon Biner Ekivalent

Dua pohon yang memiliki struktur dan informasi yangsama.

e. Pohon Biner Miring (Skewed Tree)

Dua pohon yang semua simpulnya mempunyai satu anak / turunan kecuali daun.

.

Sifat utama Pohon Berakar

1. Jika Pohon mempunyai Simpul sebanyak n, maka banyaknya ruas atau edge adalah (n-1).

2. Mempunyai Simpul Khusus yang disebut Root, jika Simpul tersebut memiliki derajat keluar >= 0, dan derajat masuk = 0.

3. Mempunyai Simpul yang disebut sebagai Daun / Leaf, jika Simpul tersebut berderajat keluar = 0, dan berderajat masuk = 1.

4. Setiap Simpul mempunyai Tingkatan / Level yang dimulai dari Root yang Levelnya = 1 sampai dengan Level ke - n pada daun paling bawah. Simpul yang mempunyai Level sama disebut Bersaudara atau Brother atau Stribling.

5. Pohon mempunyai Ketinggian atau Kedalaman atau Height, yang merupakan Level tertinggi

6. Pohon mempunyai Weight atau Berat atau Bobot, yang banyaknya daun (leaf) pada Pohon.

7. Banyaknya Simpul Maksimum sampai Level N adalah :

$$2 (N) - 1$$

8. Banyaknya Simpul untuk setiap Level I adalah :

$$\sum_{I=1}^{N} 2 (I - 1)$$

Kunjungan pada pohon Biner

Kunjungan pohon biner terbagi menjadi 3 bentuk binary tree :

1. Kunjungan secara preorder (Depth First Order), mempunyai urutan :

a. Cetak isi simpul yang dikunjungi (simpul akar),

b. Kunjungi cabang kiri,

c. Kunjungi cabang kanan .

2. Kunjungan secara inorder (symetric order), mempunyai urutan :

a. Kunjungi cabang kiri,

b. Cetak isi simpul yang dikunjungi (simpul akar),

c. Kunjungi cabang kanan .

3. Kunjungan secara postorder, mempunyai urutan :

a. Kunjungi cabang kiri,

b. Kunjungi cabang kanan,

c. Cetak isi simpul yang dikunjungi (simpul akar).

· Aplikasi pohon Biner

Notasi Prefix, Infix dan Postfix

Pada bagian ini akan dibahas tentang bagaimana menyusun sebuah Pohon Binar yang apabila dikunjungisecara PreOrder akan menghasilkan Notasi Prefix,kunjungan secara InOrder menghasilkan Notasi Infix, dankunjungan PostOrder menghasilkan Notasi Postfix.

Contoh Program Tree dalam C++

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct Node{
int data;
Node *kiri;
Node *kanan;
};
int count;
void tambah(Node **root, int databaru)
{
if((*root) == NULL){
Node *baru;
```

```
baru = new Node;
baru->data = databaru;
baru->kiri = NULL;
baru->kanan = NULL;
(*root) = baru;
(*root)->kiri = NULL;
(*root)->kanan = NULL;
printf("Data telah Dimasukkan");
}
else if(databaru < (*root)->data)
tambah(&(*root)->kiri,databaru);
else if(databaru > (*root)->data)
tambah(&(*root)->kanan,databaru);
else if(databaru == (*root)->data)
printf("Data sudah ada!!");
}
void preOrder(Node *root){
if(root != NULL){
printf("%d " ,root->data);
preOrder(root->kiri);
preOrder(root->kanan);
}
```

```
}
void inOrder(Node *root){
if(root != NULL){
inOrder(root->kiri);
printf("%d ",root->data);
inOrder(root->kanan);
}
}
void postOrder(Node *root){
if(root != NULL){
postOrder(root->kiri);
postOrder(root->kanan);
printf("%d ",root->data);
}
}
void search(Node **root, int cari)
{
if((*root) == NULL){
printf("Maaf,Data tidak ditemukan!");
}
else if(cari < (*root)->data)
search(&(*root)->kiri,cari);
```

```
else if(cari > (*root)->data)
search(&(*root)->kanan,cari);
else if(cari == (*root)->data)
printf("Data ditemukan!!!");
}
void hapus(Node **root, int del)
{
if((*root) == NULL){
printf("Data tidak ada!!");
}
else if(del < (*root)->data)
hapus(&(*root)->kiri,del);
else if(del > (*root)->data)
hapus(&(*root)->kanan,del);
else if(del == (*root)->data)
{
(*root)=NULL;
printf("Data telah Terhapus");
}
}
int main(){
int pil,cari,del;
```

```
Node *pohon;
pohon = NULL;
do{
int data;
system("cls");
printf("       PROGRAM TREE LANJUTAN    \n");
printf("================================\n");
printf("   1. Masukkan Data            \n");
printf("   2. Transverse               \n");
printf("   3. Cari                     \n");
printf("   4. Hapus                    \n");
printf("   5. Clear Data               \n");
printf("   6. Keluar                   \n");
printf("================================\n");
printf("Masukkan Pilihan Anda : ");
scanf("%d",&pil);
switch(pil){
case 1:
printf("Masukkan data baru : ");
scanf("%d", &data);
tambah(&pohon,data);
break;
```

```
case 2:
printf("\nPreOrder : ");
if(pohon!=NULL) preOrder(pohon);
else printf("Data masih kosong");
printf("\ninOrder : ");
if(pohon!=NULL) inOrder(pohon);
else printf("Data masih kosong");
printf("\npostOrder : ");
if(pohon!=NULL) postOrder(pohon);
else printf("Data masih kosong");
break;
case 3:
printf("Cari data : ");
scanf("%d", &cari);
search(&pohon,cari);
break;
case 4:
printf("Hapus data : ");
scanf("%d", &del);
hapus(&pohon,del);
break;
case 5:
```

```
pohon = NULL;
printf("Semua data telah terhapus");
break;
case 6:
return 0;
default:
printf("Maaf, pilihan Anda Salah");
}
getch();
}while(pil!=7);
}
```

Pertemuan 14 : Binary Tree : Penciptaan

14.1 Level Operator Matematika

1.Operator Aritmatika

Operator Matematika yaitu operator yang digunakan untuk operasi matematis terhadap suatu nilai data. Ada beberapa operator aritmatika yang dapat digunakan dalam pemrograman antara lain :

a.Pangkat (Eksponen)
Pemangkatan atau eksponen banyak digunakan dalam operasi komputer, eksponen mempunyai operator carret(^) dan sintaksnya adalah sebagai berikut :

Eksponen = bilanngan1^bilangan2

Contoh :

Eksponen = 5^2 hasilnya = 25
Eksponen = 3^2 hasilnya = 9
Eksponen = 2^-2 hasilnya = 0,25

b.Perkalian
Perkalian digunakan untuk mengalikan dua buah bilangan atau lebih, simbolnya (*) dan sintaksnya dalam pemrograman adalah :

Perkalian = bilangan1 * bilangan2

Contoh :
Perkalian = 2 * 2 hasilnya = 4
Perkalian = 5 * 5 hasilnya = 25

c.Pembagian
Pembagian digunakan untuk melakukan operasi pembagian terhadap bilangan, simbolnya (/) dan sintaksnya :

Pembagian = bilangan1 / bilangan2

Contoh :
Pembagian = 4 / 2 hasilnya = 2
Pembagian = 5 / 2 hasilnya = 2,5

d.Pembagian Integer
Pembagian Integer pada dasarnya sama dengan pembagian biasa, tetapi hasil yang diperoleh adalah bilangan integer atau bilangan bulat. Simbolnya (\) dan sintksnya :

PembagianInt = bilangan1 \ bilangan2

Contoh :
PembagianInt = 4 \ 2 hasilnya = 2
PembagianInt = 5 \ 2 hasilnya = 2

e.Sisa Bagi (Modulus)
Sisa bagi (Mod) digunakan untuk mendapatkan sisa dari pembagian dua buah bilangan, sintaksnya :

Sisa = bilangan1 Mod bilangan2

Contoh :
Sisa = 4 Mod 2 hasilnya = 0
Sisa = 5 Mod 2 hasilnya = 1

f.Penjumlahan
Penjumlahan (+) digunakan untuk melakukan operasi penjumlahan pada bilangan, sintaksnya :

Penjumlahan = bilangan1 + bilangan2

Contoh :
Penjumlahan = 4 + 2 hasilnya = 6
Penjumlahan = 5 + 5 hasilnya = 10

g.Pengurangan
Pengurangan (-) digunakan untuk melakukan operasi pengurangan terhadap bilangan, sintaksnya :

Pengurangan = bilangan1 – bilangan2

Contoh :
Pengurangan = 4 – 2 hasilnya = 2
Pengurangan = 5 – 2 hasilnya = 3

h.Penggabungan String (Concat)
Penggabungan string, simbolnya (&) digunakan untuk menggabungkan dua buah string. sintaksnya :

String = string1 & string2

Contoh :
String = harry & potter hasilnya = harrypotter
String = 5 & 6 hasilnya = 56

14.2 Penciptaan Binary Tree dari Ekspresi Matematik

Penelusuran pada *binary tree* ada tiga jenis:

- Preorder
- Inorder
- Postorder

DAFTAR PUSTAKA

1. Moh. Sjukani. Struktur Data (Algoritma dan Struktur Data 2) dengan C, C++ Edisi 5. Mitra Wacana Media. 2005
2. Budi Raharjo. Mudah belajar Bahasa C. Informatika.