ALGORITMA
DAN
STRUKTUR DATA I

Oleh: Agus Sujarwadi

FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS TEKNOLOGI YOGYAKARTA
2018

MODUL PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA I

Deskripsi Materi

Mata Kuliah Praktikum Algoritma dan Struktur Data I membahas tentang Konsep Dasar Struktur Data yang dimulai dari fungsi dan rekursi, array, pointer, struktur dan dilengkapi dengan aplikasi yang berkaitan meliputi struktur data dinamis, pengurutan dan pencarian data.

Pada dasarnya algoritma dapat diterapkan pada bahasa pemrograman apapun. Adapun bahasa pemrograman yang digunakan pada praktikum ini menggunakan Bahasa C++.

Petunjuk Pemakaian Modul

Untuk dapat memahami praktikum ini dengan baik mahasiswa disarankan membaca secara berurutan, karena modul ini disusun secara terstruktur diawali dengan tujuan instruksional dan kompetensi yang dihasilkan, pengantar teori, contoh kasus dan solusinya serta diakhiri dengan tugas mandiri yang berfungsi melatih mahasiswa untuk mencoba menyelesaikan suatu permasalahan sesuai dengan metode penyelesaian masing-masing.

Standar Kompetensi

Setelah mahasiswa mengambil matakuliah ini diharapkan mahasiswa memiliki kemampuan menganalisa dan mendefinisikan suatu masalah, mencari kemungkinan solusi-solusi yang dapat digunakan dan memilih solusi yang terbaik, menyusun algoritma program kemudian menyusunnya ke dalam bahasa pemrograman.

1.
FUNGSI & REKURSI

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui dasar-dasar fungsi dan rekursi.
- Mengetahui penulisan fungsi dan rekursi.
- Mengetahui penggunaan fungsi dan rekursi untuk aplikasi.

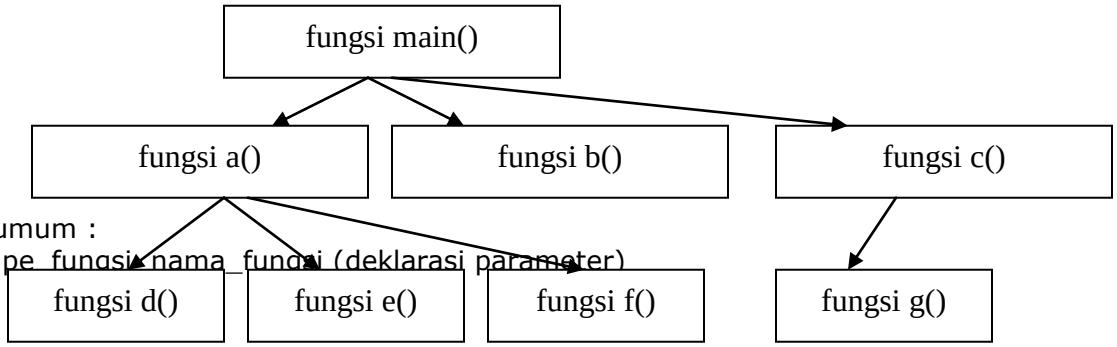
Indikator Keberhasilan :
- Mampu menjelaskan dasar-dasar fungsi dan rekursi.
- Mampu menuliskan contoh program menggunakan fungsi dan rekursi.
- Mampu menggunakan fungsi dan rekursi pada contoh aplikasi.

Dasar Teori

Fungsi adalah sekumpulan perintah operasi program yang dapat menerima argumen input dan dapat memberikan hasil output yang dapat berupa nilai ataupun sebuah hasil operasi. nama fungsi yang dibuat tidak boleh menyamai nama-nama fungsi yang sudah ”bulid in” dalam program c++. Fungsi merupakan sub program. Fungsi yang pasti ada di dalam program adalah fungsi main() seperti program-program yang sudah dibuat sebelumnya.
Contoh fungsi yang sering digunakan: clrsrc(); merupakan fungsi yang digunakan untuk membersihkan layar. fungsi clrscr ini terdapat dalam file header conio.h. sehingga untuk memanggil perintah clrscr() sebelumnya harus mengikutkan file conio.h dengan perintah #include <conio.h>. karena fungsi untuk membersihkan layar sudah dibuatkan oleh C dalam file conio.h, maka pemrogram tidak perlu susah-susah membuat sebuah fungsi sendiri. fungsi lain adalah printf(), puts(), gets() dan lain-lain.

adapun fungsi-fungsi yang tidak dibuatkan oleh C, maka pemrogram bisa membuat sendiri fungsi-fungsi sesuai dengan keinginan pemrogram

beberapat tujuan dibuatnya fungsi:
- agar perintah lebih terstruktur, sehingga mudah dibaca, mudah dalam debugging dan dokomentasi
- mengurangi banyaknya perintah akibat perintah yang sama di ulang-ulang
- efektifitas memori dll



Bentuk umum :
tipe_fungsi nama_fungsi (deklarasi parameter)

```
{
pernyataan;
pernyataan;
return var
}
```

cara pemanggilan fungsi

```
var=namafungsi(parameter);
```

Keterangan:
- tipe_fungsi merupakan tipe dari hasil kembalian yang diinginkan. adapun nilai yang akan dikembalikan digunakan perintah return var. jika tidak menginginkan kembalian, cukup diberikan perintar return; dan tipe_fungsi diisi dengan ”void”.
- perintah return berfungsi untuk mengembalikan nilai, keluar dari fungsi dan kembali ke program yang memanggil.
- deklarasi parameter merupakan parameter-parameter yang dilewatkan. jika tidak memerlukan parameter, maka cukup dikosongkan

```
tipe_fungsi  nama_fungsi ()
```

Contoh 1:

```
//Menggunakan fungsi

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

void tampilkanGaris()
{
printf("=========================================\n");
}

void main()
{
tampilkanGaris();
printf("Belajar Fungsi\n");
tampilkanGaris();
getch();
}
```

hasil:

```
==============================
Belajar Fungsi
==============================
```

Keterangan:
- Pada contoh1 di atas, fungsi tampilkanGaris berisi perintah untuk menampilkan tanda = berulang kali.
- karena fungsi tampilkanGaris tidak mengembalikan nilai, maka dibeikan tipe void dan perintah return tidak harus dituliskan

- fungsi tampilkanGaris dipanggil 2 kali, sehingga tampak hasil garis sebanyak 2 kali pula.

Contoh 2:

```
 //Penggunaan fungsi dan parameter
#include <stdio.h>
#include <conio.h>

void tampilkanGaris(int panjang)
{
int i;
for (i=1;i<=panjang;i++) printf("=");
printf("\n");
}

void main()
{
tampilkanGaris(20);
printf("Belajar Fungsi\n");
tampilkanGaris(30);
getch();
}
```

Hasil:

```
====================
Belajar Fungsi
==============================
```

Contoh 3a:

```
//penggunaan fungsi, parameter dan return
#include <stdio.h>
#include <conio.h>

void tambah1(int angka1,int angka2)
{
int hasil;
hasil=angka1+angka2;
printf("%d ditambah %d sama dengan %d\n", angka1, angka2, hasil);
}

int tambah2(int angka1,int angka2)
{
int hasil;
hasil=angka1+angka2;
return hasil;
}

void main()
{
int h;
```

```
tambah1(2,3);
h=tambah2(33,20);
printf("Hasil 33 ditambah 20 sama dengan %d\n",h);
getch();
}
```

hasil:

```
2 ditambah 3 sama dengan 5
Hasil 33 ditambah 20 sama dengan 53
```

Keterangan:

- Pada contoh 3 di atas, fungsi tambah1 tidak mengembalikan nilai, meskipun bisa menampilkan hasil, sehingga menggunakan tipe void
- fungsi tambah2 mengembalikan nilai bertipe integer yang dimasukkan dalam variabel ”hasil”. karena nilai yang dikembalikan bertipe integer, maka fungsi yang dipanggil pun harus pertipe integer, yaitu: **int** tambah2(...)
- untuk memanggil fungsi tambah1, cukup dengan perintah tambah1(2,3) di mana angka 2 bertipe integer dan angka 3 juga bertipe integer. angka 2 dan tidak bisa diganti-ganti sesuai dengan keinginan.
- untuk memanggil fungsi tambah2 menggunakan h=tambah2(33,20); yang mana variabel h harus bertipe integer, sesuai dengan tipe dari fungsi tambah2.

Contoh 3b:

```
//penggabungan fungsi dan pointer
#include <stdio.h>
#include <string.h>
#include <conio.h>

void bulan(int bl)
{
        static char *namaBulan[]=
        { "-", "Januari", "Feburari", "Maret",
          "April", "Mei", "Juni", "Juli", "Agustus", "September",
          "Oktober", "November", "Desember" };
 printf("%s", namaBulan[bl]);
}

void main()
{
printf("Bulan ke-1 adalah bulan ");bulan(1);
};
```

Hasil:

```
Bulan ke-1 adalah bulan Januari
```

Scope Variabel

Jika diperhatikan, dari contoh 3 di atas, terdapat 2 kali inisial variabel hasil, pada fungsi tambah1 dan pada fungsi tambah2.

Pada fungsi tambah1 terdapat inisial variabel int hasil;

Pada fungsi tambah2 juga terdapat inisial variabel int hasil;

Hal ini menunjukkan bahwa inisial variabel pada suatu fungsi tidak dikenali di fungsi yang lain. Variabel ini disebut dengan **variabel local**. Adapun variabel jenis lain adalah variabel public, dimana semua fungsi mengenali variabel tersebut. Variabel public ini biasanya diinisialisasi pada sebelum penulisan fungsi-fungsi. Ruang lingkup variabel ini disebut dengan *scope variabel*

.

Perhatikan contoh di bawah ini:

Contoh 4a:

```
//penggunaan fungsi, parameter dan return
#include <stdio.h>
#include <conio.h>

void tambah1(int angka1,int angka2)
{
int hasil;
hasil=angka1+angka2;
printf("%d ditambah %d sama dengan %d\n", angka1, angka2, hasil);
}

int tambah2(int angka1,int angka2)
{
int hasil;
hasil=angka1+angka2;
return hasil;
}

void main()
{
int h;
tambah1(2,3);
h=tambah2(33,20);
printf("Hasil 33 ditambah 20 sama dengan %d\n",h);
getch();
}
```

Variabel "hasil" masih bersifat local. Agar menjadi variabel yang bersifat public, maka program dapat diubah menjadi:

Contoh 4b:

```
//penggunaan fungsi, parameter dan return
#include <stdio.h>
#include <conio.h>

int hasil;

void tambah1(int angka1,int angka2)
{
hasil=angka1+angka2;
printf("%d ditambah %d sama dengan %d\n", angka1, angka2, hasil);
```

```
}

int tambah2(int angka1,int angka2)
{
hasil=angka1+angka2;
return hasil;
}

void main()
{
int h;
tambah1(2,3);
h=tambah2(33,20);
printf("Hasil 33 ditambah 20 sama dengan %d\n",h);
getch();
}
```

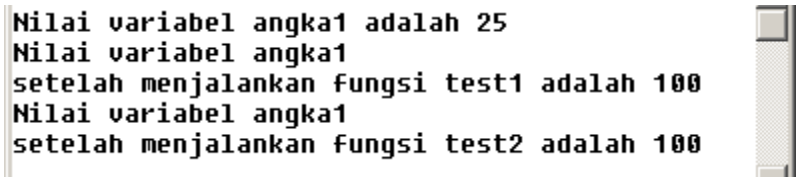
Variabel yang sudah diinisialisasi secara public, tidak perlu diinsialisasikan di dalam fungsi lagi. Sehingga inisial "int hasil" pada fungsi tambah1 dan tambah2 dihilangkan dan dipindahkan ke luar fungsi.

Contoh 5:

```
//penggunaan fungsi,variabel local dan variable public
#include <stdio.h>
#include <conio.h>
#include <math.h>

int angka1;

void test1()
{
angka1=100;
}

void test2()
{
int angka1;
angka1=200;
}

void main()
{
 clrscr();
 angka1=25;
 printf("\nNilai variabel angka1 adalah %d",angka1);
 test1();
 printf("\nNilai variabel angka1 ");
 printf("\nsetelah menjalankan fungsi test1 adalah %d",angka1);
 test2();
 printf("\nNilai variabel angka1 ");
 printf("\nsetelah menjalankan fungsi test2 adalah %d",angka1);
 getch();
```

}

Hasil:



Keterangan:

- terdapat 2 kali inisialisais variabel angka1 (int angka1), inisial variabel angka1 yang diletakkan di luar fungsi bersifat public, tetapi inisial variabel yang diletakkan pada fungsi test2 bersifat local. Karena variabel angka1 yang ada di fungsi test2 bersifat local, maka pengubahan nilai variabel angka1 yang ada fungsi tersebut tidak mengubah nilai yang ada pada variabel nilai1 yang bersifat public. Pada fungsi main, angka1 diberi nilai 25, sehingga saat ditampilkan, bernilai 25. setelah menjalankan fungsi test1, nilai angka1 berubah menjadi 100. hal ini dikarenakan pada fungsi test1 terdapat perintah mengubah angka1 (angka1=100), yang mana pada fungsi test1 tidak terdapat inisialisasi variabel angka1 yang menunjukkan bahwa angka1 yang diubah adalah angka1 yang bersifat public. Setelah menjalankan fungsi test2, angka1 tidak berubah. Padahal pada fungsi test2 terdapat pengubahan angka1 (angka1=200). Hal ini dikarenakan variabel angka1 yang diubah oleh test2 adalah variabel angka1 local.

Recursi (Recursion)

Rekursi merupakan salah satu cara untuk membagi masalah ke dalam sub masalah pada tipe yang sama. recursi ini sendiri di dalam pemrograman lebih sering dikatakan suatu fungsi yang di dalamnya terdapat perintah untuk menjalankan fungsi itu sendiri.

Kelebihan dan kekurangan Rekursi

Kelebihan utama dari rekursi, adalah kemudahan dalam memprogram. kkketika menggunakan rekursi, seorang programmer akan terfokus pada mencari solusi untuk masalah yang sedang dihadapi dan melupakan sesaat masalah secara keseluruhan dan kembali kepada permasalahan secara keseluruhan setelah masalah teratasi.

Di lain piha, rekursi memerlukan memori yang lebih besar, bahkan di sebagian besar bahasa pemrograman, rekursi menggunakan stack tersendiri untuk memasukkan kondisi dari semua rekursi yang sedang dipanggil, sehingga rekursi yang terlaku kompoleks terkadang menyebabkan *Stack Overflow* untuk mengatasi hal ini akan didiskusikan di dalam bab lain, yaitu bab loop dan struktur data stack.

Contoh penggunaan recursi

1. Menghitung Faktorial

Faktorial suatu n (dilambangkan dengan n!) merupakan hasil perkalian dari 1 sampai n., sebagai contoh: 5! merupakan hasil kali dari 1 X 2 X 3 X 4 X 5 =120

Salah satu cara penghitungan faktorial ini adalah dengan menggunakan metode rekursi. Jika diperhatikan, 5! sama dengan 5 X 4!, sehingga untuk menghitung nila fakorial n, sebelumnya perlu diketahui nilai faktorial dari n-1 atau (n-1)!. untuk menghitung (n-1)! harus diketahui pula (n-2)! dan seterusnya hingga mencarpai (n-1)<=0. yang mana faktorial 1 dan faktorial 0 menghasilkan nilai 1.

Kode Program:

```
int factorial(int n) {
    if (n <= 1)
        return 1;
    else
        return n * factorial(n - 1);
}
```

2. Contoh dalam Validasi

```
//fungsi recursi
#include <stdio.h>
#include <conio.h>

int validasi(int x)
{
if (x>=100)
{
scanf("%d",&x);
validasi(x);
}
else
return x;
}

void main()
{
 clrscr();
```

```
 int y=125;
 printf("\nMasukkan angka kurang dari 100 :");
 validasi(y);
 printf("\nTerima kasih");
 getch();
}
```

Hasil:

```
Masukkan angka kurang dari  100: 252
254
25

Terima kasih
```

Keterangan:

- Pada fungsi validasi di atas, terdapat perintah untuk memanggil diri sendiri, yaitu perintah validasi(x). perintah ini akan dijalankan jika nilai x yang sebelumnya kurang dari 100. jika nilai lebih dari 100, maka akan langsung dikembalikan dengan perintah return x.

Latihan

Ketiklah program berikut:

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

float akar(float r)
{
 float s;
 if (r>=0)
 {
        s=sqrt(r);
        return s;
 }
  else
  {
  printf("\nMaaf, Anda memasukkan data yang salah");
  return 0;
  }
 }

void main()
{
 float a,b;
 clrscr();
 printf("Masukkan bilangan >= 0 : "); scanf("%f",&a);
 b=akar(a);
```

```
 printf("\nAkar dari %8.2f adalah %8.2f",a,b);
 getch();
}
```

- Simpan dan jalankan

Latihan 2
CV TRUST merupakan sebuah lembaga yang bergerak dalam bidang IT. Untuk menghitung gaji karyawan menggunakan sistem perjam dan per golongan dengan ketentuan sebagai berikut:

Golongan	Gaji Perjam Normal
1	50000
2	40000
3	30000

Jika Lembur, maka gaji perjamnya adalah gaji perjam normal ditambah 5000.

Buatlah programnya jika diinginkan:
Input: nama, jumlah jam kerja normal,golongan, jumlah jam kerja lembur
Fungsi yang digunakan:
- fungsi hitunggaji dengan parameter:gol,jamnormal,jamlembur bertipe float

output: total gaji yang diterima

contoh kerangka program:

```
#.include …..

float hitunggaji(int gol, int jamlembur, int jamlembur)
{
        ….
}

void main()
{
        ….
}
```

contoh hasil yang diinginkan:

```
Data Karyawan
Nama        :Siti
Golongan    :1
Jam Normal  :20
Jam Lembur  :5


Total gaji yang diterima adalah 1275000.00
```

Latihan 7.c
Dari soal 7b,

- tambahkan input untuk jumlah karyawan
- gunakan perulangan untuk memasukkan data-data karyawan
- buatlah fungsi tersendiri untuk memasukkan data karyawan,
  contoh pemanggilan: inputKaryawan(x) dimana x adalan nomor karyawan
- buat pula fungsi tersendiri untuk menampilkan hasilnya
  contoh: tampilkanHasil;

Contoh kerangka program:

```
float hitunggaji(int gol, int jamlembur, int jamlembur)
{
        ….
}

void inputKaryawan(int x)
{
        …
…
}

void tampilkanHasil()
{
…
}

void main()
{
        ….
        For (x=1;x<=jlhkaryawan;x++) inputKaryawan(x);

        tampilkanHasil();

}
```

RESPONSI I

Nilai	**Materi**		**Nim**	
	Tanggal		**Nama**	
	Dosen		**Kelas**	

.....
.....
.....
.....
.....
.....
.....
.....

2.
ARRAY DAN POINTER

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui dasar-dasar array dan pointer.
- Mengetahui penulisan array dan pointer.
- Mengetahui penggunaan array dan pointer untuk aplikasi.

Indikator Keberhasilan :
- Mampu menjelaskan dasar-dasar array dan pointer.
- Mampu menuliskan contoh program menggunakan array dan pointer.
- Mampu menggunakan array dan pointer pada contoh aplikasi.

Dasar Teori

Array merupakan deretan variabel yang berjenis sama dan mempunyai nama sama. Pada bahasa C, array mempunyai lokasi yang bersebelahan. Alamat terkecil menunjuk ke elemen pertama dan alamat terbesar menunjuk ke alamat terakhir. Sebuah elemen pada array diakses melalui indeksnya.

Array 1 dimensi

Bentuk umum:

```
tipe_array nama_array[ukuran];                    //array 1 dimensi
```

cara pemberian nilai/assignmen

```
nama_array[index_array]=constant;
nama_array[index_array]=var;
```

Keterangan:

tipe_array : menyatakan tipe dasar array
ukuran : menyatakan banyaknya elemen pada array
index_array : nomor index dari array yang akan di isi

Contoh :

```
int nilai[5];
nilai[1]=90;
nilai[2]=80;
```

hal ini menyatakan array *nilai* mempunyai 5 elemen, yaitu :
nilai[1] , nilai[2] , nilai[3] , nilai[4] dan nilai [5]

Semakin besar ukuran array, semakin besar pula memori yang digunakan. Total memori yang digunakan adalah :

byte total = ukuran dari tipe dasar * banyaknya elemen

dalam contoh diatas, jumlah memori yang digunakan sebesar 2*5 byte =10 byte di mana 2 byte merupakan jumlah memori yang digunakan untuk menyimpan 1 buah variabel bertipe integer. meskipun jumlah elemen yang dideklarasikan 5, tetapi dalam prakteknya, pemrogram bisa menambah elemen secara otomatis jika nomor index yang bersangkutan di sebutkan, contoh: nilai[8]=75, padahal pada saat pendefinisian maksimal 5.

Array Multi Dimensi

Bentuk umum:

```
tipe_array nama_array[ukuran1][ukuran2];                //array 2 dimensi
tipe_array nama_array[ukuran1][ukuran2][ukuran3];       //array 3 dimensi
dst.
```

Contoh:

```
int matrix1[2][2];
char nama[10][20];
```

matrix1 mempunyai elemen array :

matrix1[1][1], matrix1[1][2], matrix1[2][1], dan matrix1[2][2]

sedangkan contoh cara pengisian data:

```
matrix1[1][1]=90;
nama[1]="Um"
```

jika diinginkan menampilkan, maka dapat digunakan printf atau untuk string gunatan gets

```
printf("nilai matrix1[1,1] adalah %d", matrix1[1][1]);
printf("nama anak ke 1 adalah "), gets(nama[1]);
```

Adapun jumlah memori yang digunakan sebesar

byte total = ukuran dari tipe dasar * banyaknya elemen1 * banyaknya elemen2

sehingga

matrix1[2][2] memerlukan memori sebesar 2*2*2 byte =8 byte.

contoh penggunaan Array:

```
int i,j;
 int m[2][2];
 clrscr();
 for(i=1; i<=2; i++)
 {
  for(j=1; j<=2; j++)
  {
   m[i][j] = i*j;
   printf("%3d", m[i][j]);
  }
```

```
 printf("\n");
}
```

Inisialisasi Array
Seperti halnya variabel biasa, variabel array juga dapat diberikan nilai awal/inisial.
Bentuk umum :

```
tipe nama_array [ukuran1, ukuran2, …, ukuran n] = {daftar nilai};
```

Daftar nilai berisi konstanta yang sesuai dengan tipe array. Setiap konstanta dipisahkan oleh tanda koma. Konstanta pertama akan mengisi elemen pertama, konstanta kedua untuk elemen kedua dan seterusnya.

contoh:

```
int nilai[5] = {90,85,75,99,100};
```

hal ini menunjukkan bahwa
nilai[1]=90, nilai[2]=80 dan seterusnya.

Pointer
Pointer adalah sebuah variabel yang berisi alamat memori komputer di mana suatu variabel diletakkan.

Perhatikan contoh penggunaan memori di bawah ini:

			varA	varB			
nilai variable	...	...	255	700	...	...	...
alamat memori			1456	1457			

- variabel varA di simpan di memori komputer pada alamant 1456 dan , variabel varB diletakkan di 1457
- varA berisi angka 255 dan varB berisi angka 700

Operator Pointer
Ada 2 operator pointer :
1. Operator &
   Bersifat unary (hanya memerlukan satu operan). Operator ini menghasilkan alamat dan operannya. Contoh : x = &a, jika kita mengacu pada gambar di atas, maka isi variabel x adalah 1004.

2. Operator *
   Operator * dan operator & bersifat saling komplemen. Operator * juga bersifat unary dan menghasilkan nilai yang berada pada sebuah alamat..

Contoh:

```
int varA, *alamat1, varB;
 varA=5;
 alamat1=&varA;
 varB=*alamat1;
```

```
printf("\n Isi variabel varA adalah %4d", varA);
printf("\n Isi variabel alamat1 adalah %4u", alamat1);
printf("\n Isi variabel varB adalah %4d", varB);
printf("\n Isi variabel alamat1 sekarang adalah %4u", alamat1++);
```

Jika dijalankan, maka hasilnya adalah:

```
Isi variabel varA adalah    5
Isi variabel alamat1 adalah 8808
Isi variabel varB adalah    5
Isi variabel alamat1 sekarang adalah 8810_
```

Keterangan:

- variabel alamat1 berisi alamat memory dari variabel varA.
- pada contoh di atas, 8808 merupakan tempat di mana data variabel a di simpan di memory komputer pada saat program dijalankan. angka tidak mutlak, bisa berubah setiap saat jika program ini di jalankan di lain waktu.
- variabel alamat1 harus bertipe sama dengan varA, yaitu integer.
- *x menghasilkan isi dari alamat yang dituju X, sehingah asil dari varB=5, karena alamat yang dituju alamat1 adalah variabel A.
- Alamat dapat dicetak dengan format **%u** sebab alamat adalah bilangan bulat tanpa tanda (unsigned integer)
- nilai increment alamat1 adalah sama dengan jumlah memory yang digunakan oleh variabel int (variabel yang dituju), yaitu 2, sehingga ++alamat1 menghasilkan nilai 8808+2=8810;

Contoh 2:

```
int nilai[5],*alamat1;
int hasil;
nilai[1]=90;
nilai[2]=80;
nilai[3]=75;

alamat1=&nilai[1];
hasil=*(alamat1+2);//sama dengan hasil=nilai[1+2]
printf("\n Isi variabel nilai[1] adalah %4d", nilai[1]);
printf("\n Isi variabel alamat1 adalah %4u", alamat1);
printf("\n Isi variabel nilai[3] adalah %4d", nilai[3]);
printf("\n Isi variabel Hasil adalah %4u", hasil);
```

Hasilnya:

```
Isi variabel nilai[1] adalah   90
Isi variabel alamat1 adalah 8798
Isi variabel nilai[3] adalah   75
Isi variabel Hasil adalah   75
```

Keterangan:

- hasil=*(alamat1+2); sama halnya dengan hasil= nilai[3], yaitu nilai[1+2]

Array dapat ditunjuk dengan index-nya dan dengan alamat. Penunjukan dengan alamat akan lebih cepat daripada penunjukan dengan index-nya.

Contoh 1:

```
/* menggunakan index array */
#include <stdio.h>
#include <ctype.h>
#include <conio.h>

void main()
{
 char kata[80];
 int i;

 printf("Masukkan kalimat(huruf besar):\n");gets(kata);
 printf("huruf kecilnya adalah : ");
 for(i=0; kata[i]; i++) printf("%c",tolower(kata[i]));
 getch();
}

/* menggunakan pointer */
#include <stdio.h>
#include <ctype.h>
#include <conio.h>

void main()
{
 char kata[80], *p;
 clrscr();
 printf("Masukkan kalimat(huruf besar):\n"); gets(kata);
 printf("huruf kecilnya adalah:\n");
 p=kata;
 while (*p) printf("%c",tolower(*p++));
 getch();
}
```

Hasil:

```
Masukkan kalimat(huruf besar):
SEJERNIH MATA AIR
huruf kecilnya adalah:
sejernih mata air_
```

Contoh 2:

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

 void main()
{
 char kata[80];
 char *p;
 int i;
```

```
 clrscr();
 printf("Masukkan 2 kalimat: "); gets(kata);
 for (i=0; kata[i] && kata[i] != '.'; i++);
 p=&kata[i+1];
 printf("Kalimat ke-2 adalah : \n");
 printf(p);
 getch();
}
```

Hasil:

```
Masukkan 2 kalimat: Ketentraman Hati.Kebahagiaan Haqiqi
Kalimat ke-2 adalah :
Kebahagiaan Haqiqi_
```

Array Pointer

Seperti halnya variabel lain, Array pointer bisa juga digunakan.
bentuk umum:

```
tipe *nama_variable_pointer[ukuran]
```

contoh

```
char *y[10]
```

array pointer ini sering digunakan dalam berbagai operasi string.

Contoh dalam aplikasi:

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

void main()
{
 char *jurusan[] = {
  "Jogja-Semarang",
  "Jogja-Solo",
  "Jogja-Surabaya",
  "Jogja-Malang"
 };
 int pilih;
 clrscr();
 printf("Masukkan Pilihan Anda(1-4):"); scanf("%d",&pilih);
 printf("\nAngkutan yang Anda pilih adalah  : \n");
 printf("%s",jurusan[pilih-1]);
 getch();
}
```

Hasil:

```
Masukkan Pilihan Anda(1-4):2

Angkutan yang Anda pilih adalah  :
Jogja-Solo
```

Contoh lainnya:

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

void main()
{
char *nama1 = "SEBENING EMBUN PAGI";
char *nama2 = "SEJERNIH MATA AIR";
char *nama3;
 clrscr();

 puts("Awal :");
 printf("Nama-1 : %s", nama1);
 printf("\nNama-2 : %s", nama2);

nama3=nama1;
nama1=nama2;
nama2=nama3;

 puts("\n\nSetelah ditukar :");
 printf("Nama-1 : %s", nama1);
 printf("\nNama-2 : %s", nama2);
}
```

Hasil:

```
Awal :
Nama-1 : SEBENING EMBUN PAGI
Nama-2 : SEJERNIH MATA AIR

Setelah ditukar :
Nama-1 : SEJERNIH MATA AIR
Nama-2 : SEBENING EMBUN PAGI
```

Pointer banyak dilibatkan dalam program C, misalnya untuk melewatkan string dari suatu fungsi ke fungsi yang lain. Penerapan pointer yang paling umum yaitu untuk menciptakan variabel dinamis yang memungkinkan untuk memakai memori bebas (memori yang belum dipakai) selama eksekusi program.

Agar suatu pointer menunjujk ke variabel lain, mula-mula pointer harus diisi dengan alamat dari variabel yang akan ditunjuk. Untuk menyatakan alamat dari suatu variabel, operator & (operator alamat, yang bersifat unary) bisa digunakan dengan cara menempatkan operator di depan nama variabel.

Jika suatu variabel sudah ditunjuk pointer, variabel tersebut dapat diakses melalui variabel itu sendiri (dikatakan sebagai pengaksesan langsung) ataupun melalui

pointer (dikatakan sebagai pengaksesan tidak langsung). Pengaksesan tak langsung dilakukan dengan menggunakan operator indirection berupa simbol * (bersifat unary).

Latihan

Buatlah sebuah program dengan menggunakan tipe struk untuk memasukkan data buku perpustakaan, jika diketahui field-field bukunya: judul,penerbit,pengarang,tahun terbit.

RESPONSI II

Nilai	**Materi**		**Nim**	
	Tanggal		**Nama**	
	Dosen		**Kelas**	

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

3.
STRUKTUR

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui *struct*.
- Mengetahui penulisan *struct*.
- Mengetahui penggunaan dan manfaat *struct* pada aplikasi.

Indikator Keberhasilan :
- Mampu menjelaskan tentang *struct*.
- Mampu menuliskan program menggunakan *struct*.
- Mampu menggunakan fungsi pada contoh *struct*.

Dasar Teori

Struktur adalah koleksi dari variabel yang dinyatakan dengan sebuah nama dengan sifat setiap variabel dapat memiliki tipe yang berlainan. Struktur biasa dipakai untuk mengelompokkan beberapa informasi yang berkaitan menjadi sebuah kesatuan. Struktur ini sering digunakan untum mencatat berkas data, seperti database mahasiswa, database penjualan dll.

bentuk umum:

```
typedef struct nama_tipe_struktur
{ tipe field-1;
  tipe field-2;
  tipe field-n;
} var_struk1, var_struk2,..., var_strukn;
```

atau

```
typedef struct nama_tipe_struktur
{ tipe field-1;
  tipe field-2;
  tipe field-n;
}

nama_tipe_struk var_struk1, var_struk2,..., var_strukn;
```

contoh :

```
typedef struct dataMahasiswa
{ char nama[30];
  char kelas[10];
  int nilai;
};

dataMahasiswa mahasiswa1;
```

atau

```
typedef struct dataMahasiswa
{ char nama[30];
```

```
  int nilai;
} mahasiswa1;
```

untuk mengisi atau mengambil suatu nilai field yang ada di dalam struk digunakan

```
var_struk1.nama_field
```

contoh:

```
mahasiswa1.nilai=90;
gets(mahasiswa1.nama);
```

Struk bertingkat

Dalam prakteknya, suatu struk bisa di definisikan ke dalam struk lain.

contoh:

```
struct tanggal
  { int tanggal;
        int bulan;
        int tahun;
  };

 struct dataMahasiswa
  {
       char nama[30];
       int nilai;
       struct tanggal tgl_lahir;
} mahasiwa1;
```

jika diinginkan mengisi data bulan, maka dapat dilakukan dengan cara:

```
mahasiswa1.tgl_lahir.bulan=9
```

sedangkan untuk mengambil data nama atau data nilai,

```
puts(mahasiswa1.nama);
n=mahasiswa1.nilai;
```

Contoh:

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

void main()
{
 struct tanggal
  { int tanggal;
        int bulan;
        int tahun;
  };

 struct dataMahasiswa
  {
       char nama[30];
```

```
        struct tanggal tgl_lahir;
};

struct dataMahasiswa dataku;
char tombol;
int i, jumlah = 0;

puts("DATA MAHASISWA : \n");
clrscr();
printf("Nama  : "); gets(dataku.nama);
printf("Tanggal Lahir (tgl-bulan-tahun) : ");
scanf("%d-%d-%d", &dataku.tgl_lahir.tanggal,
&dataku.tgl_lahir.bulan,
&dataku.tgl_lahir.tahun);

printf("\nData MAHASISWA\n");
printf("NAMA                TANGGAL LAHIR");
printf("\n%-21s %d-%d-%d", dataku.nama,  dataku.tgl_lahir.tanggal,
dataku.tgl_lahir.bulan,  dataku.tgl_lahir.tahun);
getch();
}
```

Hasil:

```
Nama  : Siti
Tanggal Lahir (tgl-bulan-tahun) : 12-12-1987

Data MAHASISWA
NAMA                    TANGGAL LAHIR
Siti                    12-12-1987
```

Array Struktur

Penggabungan struktur dengan array sering kali dilakukan.

Contoh :

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#define maks 20

void main()
{
 struct tanggal
  { int tanggal;
        int bulan;
        int tahun;
  };

 struct dataMahasiswa
  {
       char nama[30];
```

```
        struct tanggal tgl_lahir;
};

struct dataMahasiswa dataku[maks];
char tombol;
int i, jumlah = 0;

puts("DATA MAHASISWA : \n");
clrscr();
do
 {
printf("Nama  : "); gets(dataku[jumlah].nama);
printf("Tanggal Lahir (tgl-bulan-tahun) : ");
scanf("%d-%d-%d", &dataku[jumlah].tgl_lahir.tanggal,
&dataku[jumlah].tgl_lahir.bulan,
&dataku[jumlah].tgl_lahir.tahun);

fflush(stdin); //hapus sisa data dalam penampung keyboard
 jumlah ++;
 printf("\nMasih ada data lagi [y/t] ? ");
 tombol = toupper(getch());
 while ( !(tombol == 'T' || tombol == 'Y'))
                tombol = toupper(getch());
 printf("%c\n\n", tombol);
}

while (tombol== 'Y');
printf("\nData MAHASISWA\n");
printf("NAMA            TANGGAL LAHIR");
for(i=0; i<jumlah; i++)
printf("\n%-21s %d-%d-%d", dataku[i].nama,  dataku[i].tgl_lahir.tanggal,
dataku[i].tgl_lahir.bulan,  dataku[i].tgl_lahir.tahun);
getch();
}
```

Hasil :

```
Nama  : Aisyah
Tanggal Lahir (tgl-bulan-tahun) : 2-10-1985

Masih ada data lagi [y/t] ? Y

Nama  : Dian
Tanggal Lahir (tgl-bulan-tahun) : 18-8-1987

Masih ada data lagi [y/t] ? Y

Nama  : Artanti
Tanggal Lahir (tgl-bulan-tahun) : 20-10-1988

Masih ada data lagi [y/t] ? T


Data MAHASISWA
NAMA             TANGGAL LAHIR
Aisyah                 2-10-1985
Dian                   18-8-1987
Artanti                20-10-1988_
```

Latihan

Sebuah perusahaan memberikan aturan pembayan gaji sebagai berikut:

- Menghitung gaji pokok:
  Gol 1: gaji pokok = Rp 1 000 000,00
  Gol 2: gaji pokok = Rp 1 500 000,00
  Gol 3: gaji pokok = Rp 2 000 000,00
  Gol 3: gaji pokok = Rp 2 500 000,00
- Menghitung Tunjangan Istri
  Tunjangan istri hanya bagi laki-laki yang sudah menikah, yaitu 10% dari gaji pokok
- Menghitung Tunjangan Anak
  Tunjangan Anak sebesar 2% untuk setiap anak dan dibatasi maksimal 2 anak, sehingga anak ketiga dan selanjutnya tidak mendapat tunjangan
- Menghitung Tunjangan Beras
  Tunjangan beras dihitung sebanyak 5 Kg perkepala, sehingga jika dalam suatu keluarga terdapat ayah, ibu, dan 3 anak, maka tunjangan sebanyak 25 Kg

a. Dari data di atas, buatlah program yang dapat digunakan untuk mengisi data dan menaampilkan data, jika diinginkan hasil seperti berikut:

```
**********************************************************
PRAKTIKUM PENGGUNAAN STRUC
OLEH  (nama anda)
**********************************************************
Memasukkan data
**********************************************************
Nama : Aan
Golongan : 1
Jenis Kelamin [L/P]: L
Status Menikah [S/N]: N
Jumlah Anak : 5
**********************************************************
Terima kasih. Data Anda sudah dimasukan
Mau menambah data lagi [Y/T]? T
**********************************************************

**********************************************************
PRAKTIKUM PENGGUNAAN STRUC
OLEH  (nama anda)
**********************************************************
Melihat Data Karyawan
**********************************************************
Nama      Golongan JK  Status Jlh.Anak  Gaji Pokok  Gaji Bersih
**********************************************************
Iin         1          P  S             0           100000         1000000
Aan         1          L  N             5           100000         1140000
**********************************************************
Tekan sembarang tombol untuk mengakhiri
**********************************************************
```

b.Kembangkan program, sehingga dapat digunakan untuk memasukkan/menambah data, menghapus data, mencari data, dan memperbaiki data di atas, jika tampilan seperti berikut:

```
**********************************************
PRAKTIKUM PENGGUNAAN STRUC
OLEH  (nama anda)
**********************************************
PILIHAN MENU
**********************************************
1. Memasukkan Data
2. Menampilkan Data
3. Mencari Data
4. Memperbaiki Data
5. Menghapus Data
**********************************************
Pilihan Anda:1
**********************************************

Pilihan 1:
**********************************************
PRAKTIKUM PENGGUNAAN STRUC
OLEH  (nama anda)
**********************************************
Memasukkan data
**********************************************
Nama : Aan
Golongan : 1
Jenis Kelamin [L/P]: L
Status Menikah [S/N]: N
Jumlah Anak : 5
**********************************************
Terima kasih. Data Anda sudah dimasukan
Tekan sembarang tombol untuk kembali ke menu...
**********************************************
dst.
```

RESPONSI III

Nilai	**Materi**		**Nim**	
	Tanggal		**Nama**	
	Dosen		**Kelas**	

.....

.....

.....

.....

.....

4
Pengurutan (Sort)

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui dasar-dasar algoritma pengurutan.
- Mengetahui penggunaan masing-masing algoritma.
- Mengetahui efisiensi masing-masing algoritma.

Indikator Keberhasilan :
- Mampu menjelaskan dasar-dasar algoritma pengurutan.
- Mampu menuliskan program dengan menterjemahkan algoritma ke dalam bahasa pemrograman.
- Mampu membandingkan efisiensi algoritma yang satu dengan lainnya.

Dasar Teori

Secara bahasa, pengurutan merupakan suatu proses mengatur sekumpulan obyek menurut urutan atau susunanan terntentu baik berurutan naik maupun turun.
Pengurutan data secara menurun disebut dengan pengurutan Ascending

L[1]<=L[2]<=L[3]<=....<=L[N]

contoh urutan Naik (Ascending):
- data numeric: 1, 3, 4, 5, 9, 10
- data string: Aan, Budi, Dani, Dian, Siti
- data rekaman, urut berdasarkan field nomor: <001,Aan>,<002, Siti>,<003,Dani>
- data rekaman, urut berdasarkan field nama: <003,Aan>,<003, Dani>,<002,Siti>

Pengurutan data secara naik disebut dengan pengurutan Descending

L[1]>=L[2]>=L[3]>=....>=L[N]

contoh urutan Turun (Descending):
- data numeric: 10, 9, 5, 4, 3, 1
- data string: Siti, Dian, Dani, Budi, Aan
- data rekaman, urut berdasarkan field nomor: <003,Dani>,<002, Siti>,<001,Aan>

Tujuan dari pengurutan Data
Terdapat beberapa alasan mengapa data perlu diurutkan:
- mempercepat pencarian
- memudahkan menentukan nilai minimum dan maksimum

Pengurutan Internal dan Eksternal
Berdasarkan memori yang digunakan pada saat proses pengurutan, pengurutan dibagi menjadi:
- Pengurutan internal,

Adalah pengurutan terhadap sekumpulan data yang disimpan dalam memori utama komputer. umumnya struktur data yang dipakai adalah larik/array, sehingga pengurutan data internal disebut juga pengurutan array.

- Pengurutan Eksternal,
  Adalah pengurutan terhadap sekumpulan data yang disimpan dalam memori sekuder. Data yang disimpan biasanya berkapasitas besar, sehingga terdapat kendala apabila disimpan langsung di dalam array.biasanya penyimpanan data sekunder ini berupa file, sehingga pengurutan data secara eksternal disebut juga pengurutan arsip/file.

Algoritma Pengurutan
Terdapat berbagai jenis algoritma pengurutan, beberapa di antaranya adalah:

- Buble Sort
- Insertion Sort
- Selection Sort (Maximum/Minimum Sort)
- Shell Sort
- Quick Sort
- Merge Sort
- Radix Sort
- dll

Bubble Sort

Metode pengurutan gelembung diinspirasikan oleh gelembung sabun yang berada dipermukaan air. Karena berat jenis gelembung sabun lebih ringan daripada berat jenis air, maka gelembung sabun selalu terapung ke atas permukaan.

Buble sort merupakan algoritma pengurutan yang mudah dan sudah tidak asing lagi didengar.buble sort mengikuti algoritma pengurutan $O(n^2)$ yang membuat algoritma terlalu bertele-tele jika jumlah elemen yang akan diurutkan berjumlah besar. Jumlah langkah membandingkan nilai sebanyak n+n-1+n-2 …. +2 kali, artinya jika terdapat 5 komponen yang akan diurutkan, maka jumlah pembandingan dengan algortima buble sort maksimal sebanyak sebanyak 5+4+3+2 = 14 langkah.jika jumlah elemen sebanyak 9. maka jumlah maksimal pembandingan sebanyak 9+8+7+6+5+4+3+3+2 = 44

Jumlah maksimal langkah= (n*(n+1))/2-1

di mana n adalah jumlah elemen yang akan diurutkan

Algoritma Buble Sort

1. masukkan data yang akan diurutkan
2. putaran (j)=1, swap=false
3. bandingkan setiap data pada elemen (dt[i]) dengan data pada elemen setelahnya (dt[i+1]), jika data pada elemen lebih besar dari data pada elemen setelahnya, maka tukar data, swap=true.
4. lanjutkan untuk membandingkan data berikutnya dengan setelahya hingga akhir data-putaran (n-j).

5. jika tidak ada yang ditukar dari awal sampai batas atas (swap=false), proses dihentikan, hal ini **menunjjukkan bahwa data sudah urut.**
   jika tidak, kurangi batas atas, ba=ba-1dan ulangi dari langkah ketiga

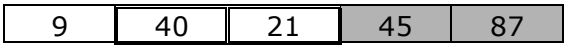
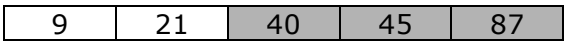
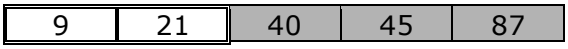
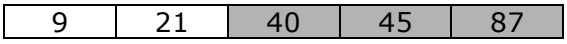
Perhatikan contoh berikut:

Contoh:.

Terdapat data 45, 40, 9, 87, 21

Proses pengurutan menggunakan Bubble sort:

45 40 9 87 21	Posisi awal n=5;j=1 swap=false
45 40 9 87 21	i=1,dt[i]=dt[1]=45, dt[i+1]=dt[2]=40 karena dt[1]>dt[2], maka tukar dt[1] dengan dt[2] dan swap=true
40 45 9 87 21	i=2,dt[i]=dt[2]=45, dt[i+1]=dt[3]=9 karena dt[2]>dt[3], maka tukar dan swap=true
40 9 45 87 21	i=3,dt[i]=dt[3]=45, dt[i+1]=dt[4]=87 karena dt[2]<dt[3], tidak terjadi pertukaran
40 9 45 87 21	i=4,dt[i]=dt[4]=87, dt[i+1]=dt[5]=21 karena dt[2]>dt[3], maka tukar dan swap=true
40 9 45 21 87	i=4 dan (n-j)=4, ini menujukan bahwa proses pertukaran putaran pertama sudah selesai. lanjutkan ke putaran berikutnya, j=j+1=2, i kembali ke 1, dan swap dikembalikan ke posisi false
40 9 45 21 87	i=1,dt[i]=dt[1]=40, dt[i+1]=dt[2]=9 karena dt[2]<dt[3], tidak terjadi pertukaran, swap=true
9 40 45 21 87	Dst, cara sama dengan di atas
9 40 45 21 87	Dst, cara sama dengan di atas
9 40 21 45 87	i=3 dan (n-j)=3 menunjukkan putaran sudah selesai, siap untuk menuju putaran berikutnya. selama perputaran pernah terjadi pertukaran (swap=true), maka j=j+1=3, i=1, swap=false, lanjutkan ke perputaran selanjutnya
9 40 21 45 87	

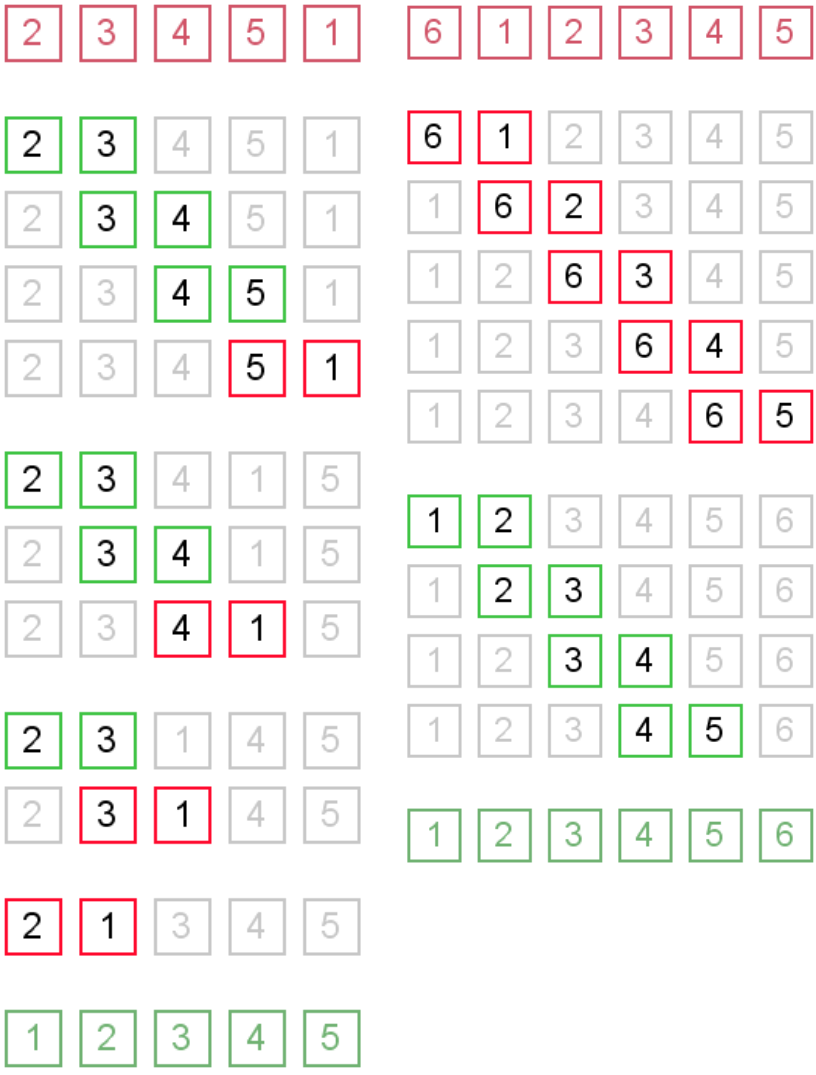
	
	Posisi akhir perputaran ketiga, selama perputaran ketiga pernah terjadi pertukaran, maka j=j+1=4, i=1, swap=false
	
	Posisi akhir Perputaran keempat. Selama perputaran keempat tidak ada pertukaran (swap masih dalam keadaan false), sehingga, proses dihentikan dan dinyatakan data sudah urut.

Analisa kekomplekan

Dari data di atas terlihat bahwa di setiap putaran, terdapat pengecekan apakah pernah terjadi pertukaran atau tidak yang ditandai dengan variable swap. Jika terjadi pertukaran, maka ubah status swap menjadi true. Di setiap akhir putaran ini, batas atas dikurangi 1. hal ini menunjjukan bahwa nilai terakhir putaran sudah tidak perlu lagi dibandingkan di putaran berikutnya.

Terkadang metode pengurutan bubble sort ini bias berjalan dengan cepat, terkadang pulab berjalan dengan lambat. Hal ini tergantung dari data yang data.

Misalkan terdapat data 2,3,4,5,1 akan mempunyai iterasi jauh lebih banyak dibanding dengan data 6,1,2,3,4,5 meskipun jumlah elemennya lebih sedikit.



Kode Program Buble Sort

Ada berbagai cara dalam mengimplementasikan pengurutan data bubble sort ke dalam program. Yang perlu diingat adalah perlunya pengecekan apakah terdapat pertukaran pada iterasi tertentu. Hal ini sangat digunakan untuk menentukan apakah akan menuju ke iterasi berikutnya atau dinyatakan data sudah urut.

```
void bubbleSort(int arr[], int n) {
    bool swapped = true;
    int j = 0;
    int tmp;
    while (swapped) {
        swapped = false;
        j++;
        for (int i = 0; i < n - j; i++) {
```

```
            if (arr[i] > arr[i + 1]) {
                tmp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = tmp;
                swapped = true;
            }
        }
    }
}
```

Selection Sort

Metode Pengurutan Selection sort merupakan salah satu dari metode algoritma pengurutan $O(n^2)$, yang mana besarnya elemen array menyebabkan proses pengurutan tidak efisien. Selection sort terkadang sangat penting digunakan dalam metode pengurutan secara mudah , bahan di dalam situasi tetentu metode pengurutan selection sort digunakan di dalam metode pengurutan yang lain karena kekomplekan data. Algoritma pengurutan selection sort sering juga disebut dengan maximum/minimum sort, dimana proses pengurutannya, pertama mencari nilai maksimum dan minimum dari deret yang ada yang kemudian digeser ke urutan awal.

Algoritma

Algoritma selection sort sangat mudah dipahami. bayangkan bahwa data dibagi dalam 2 bagian, bagian yang sudah diurutkan dan bagian yang belum. pada awal pengurutkan, bagian data yang urut masih kosong, sehingga bagian data yang belum urut adalah seluruh data. di setiap langkah, cari nilai minimal di dalam bagian yang belum diurutkan. pindahkan nilai minimal ini dari array yang belum diurutkan ke akhir dari array yang sudah diurutkan. ketika array yang belum diurutkan sudah kosong, maka proses dihentikan yang menunjukkan bahwa data sudah urut.

Algoritma ketika proses pengurutan, nilai yang paling kecil dari array ditukarkan ke elemen array yang pertama, dan anggap elemen pertama dari array yang belum urut ini sudah masuk dalam elemen dari array yang sudah diurutkan. hal ini dilakukan jika elemen pertama (dari array yang belum diurutkan) bukan merupakan nilai yang terkecil dari array yang belum diurutkan. jika elemen pertama merupakan nilai terkecil, maka tidak perlu penukaran dan langsung di anggap sebagai elemen data dari array yang sudah diurutkan.

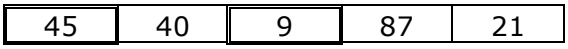
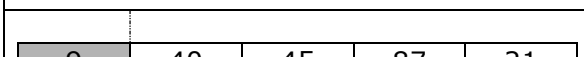
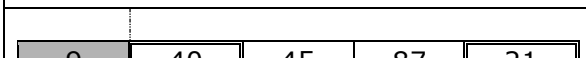
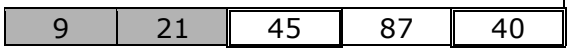
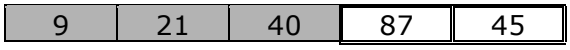
Contoh:

Terdapat data {45,40,9,87,21}

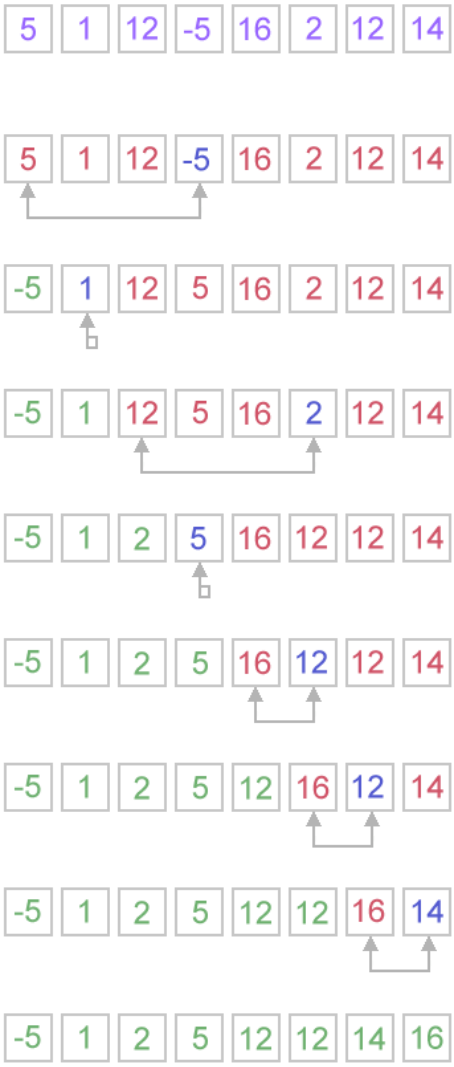
data ini akan diurutkan dengan metode pengurutan selection sort.

45	40	9	87	21

Terdapat 5 buah data/elemen, anggap semua data masuk dalam kelompok array yang belum diurutkan (ua).

	Dari data di samping tampak nilai yang terkecil adalah angka 9. dengankan elemen pertama array, angka 45. maka proses pertukaran dilakukan
	setelah proses pertukaran, maka elemen pertama dari ua, dianggap masuk ke dalam elemen sorted array (sa).
	lakukan pembandingan untuk mencari nilai terkecil dari ua. dari data disamping tampak nilai terkecildari ua adalah 21. lakukan pertukaran dengan elemen pertama dari ua. dan anggap masuk dalam elemen sa.
	sda
	
	
	hasil akhir

Contoh 2:. Sort {5, 1, 12, -5, 16, 2, 12, 14}



Complexity analysis

Selection sort stops, when unsorted part becomes empty. As we know, on every step number of unsorted elements decreased by one. Therefore, selection sort makes n steps (n is number of elements in array) of outer loop, before stop. Every step of outer loop requires finding minimum in unsorted part. Summing up, n + (n - 1) + (n - 2) + ... + 1, results in $O(n^2)$ number of comparisons. Number of swaps may vary from zero (in case of sorted array) to n - 1 (in case array was sorted in reversed order), which results in O(n) number of swaps. Overall algorithm complexity is $O(n^2)$.

Fact, that selection sort requires n - 1 number of swaps at most, makes it very efficient in situations, when write operation is significantly more expensive, than read operation.

Kode Program

```
void selectionSort(int arr[], int n) {
```

```
    int i, j, minIndex, tmp;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[minIndex])
                minIndex = j;
        if (minIndex != i) {
            tmp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = tmp;
        }
    }
}
```

Insertion Sort

Insertion sort termasuk dalam kategori algoritma pengurutan $O(n^2)$. tidak seperti algoritma pengurutan yang lain, insertion sort mempunyai algoritma yang praktis dan lebih pendek untuk array yang mempunyai elemen sedikit. untuk lebih mudahnya, biasanya orang pengututan ini seperit pengurutan kartu..

Algoritma

Insertion sort mirip dengan selection sort, yang mana Array diumpamakan dibagi dalam 2 bagian, yaitu bagian yang sudah diurutkan (sorted array) dan bagian yang belum diurutkan (unsorted Array). Awalnya bagian array yang sudah diurutkan berisi elemen pertama array, sedangkan bagian yang belum diurutkan berisi sisanya. di setiap langkah, algoritma memindah elemen pertama dari bagian array yang belum diurutkan untuk disisipkan ke tempat yang tepat dari array yang sudah diurutkan. pengurutan dihentikan jika bagian array yang bel

um diurutkan sudah kosong. Perhatikan sketsa berikut:

bagian array yang sudah urut			bagian array yang belum urut
≤ X	>X	X	...

menjadi

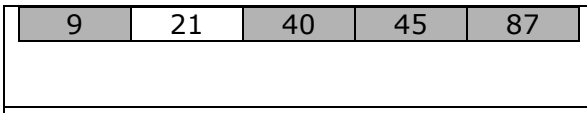
bagian array yang sudah urut			bagian array yang belum urut
≤ X	X	>X	...

Contoh:

Terdapat data {45,40,9,87,21}

data ini akan diurutkan dengan metode pengurutan insertion sort.

45 | 40 | 9 | 87 | 21	Posisi awal, anggap elemen pertama adalah bagian data yang sudah diurutkan (sorted array-sa), sedangkan bagian selanjutnya adalan bagian yang belum diurutkan. (unsorted array-ua), yaitu 40,9,87 dan 21
45 | 40 | 9 | 87 | 21	Mulai proses penyisipan. elemen pertama dari ua adalah 40. adalah angka 45, maka tukar posisi.
40 | 45 | 9 | 87 | 21	bandingkan angka 40 dengan angka sebelunya. jika data sebelunya lebih besar, maka tukar posisi. karena data sebelumnya.
40 | 45 | 9 | 87 | 21	perhatikan bahwa angka 40 sudah mencapat batas awal, artinya sudah tidak ada data lagi sebelumnya, maka penyisipan dihentikan dan dinyatakan sudah urut. perlu diperhatikan bahwa penyisipan akan berakhir jika data sebelumnya <= data yang akan disisipkan atau sudah berada di elemen pertama array (mencapai batas awal array).
40 | 45 | 9 | 87 | 21	langkah selanjutnya proses penyisipan untuk data berikutnya. data berikutnya adalah 9. data sebelumnya adalah lebih besar, yaitu angka 45, sehingga tukar posisi.
40 | 9 | 45 | 87 | 21	bandingkan kembali dengan data sebelumnya lebih besar, yaitu angka 40, maka tukar posisi
9 | 40 | 45 | 87 | 21	karena berada diawal array, maka proses penyisipan dianggap selesai dan dinyatakan urut. dan siap untuk menuju ke data berikutnya
9 | 40 | 45 | 87 | 21	langkah selanjutnya membandingkan angka 87 dengan data sebelunya. dalam hal ini data sebelumnya 45, 45<87, langsung dianggap sudah urut. dan persiapan untuk menuju data yang terakhir
9 | 40 | 45 | 87 | 21 9 | 40 | 45 | 21 | 87 9 | 40 | 21 | 45 | 87	data terakhir adalah 21, dengan cara yang sama dengan data sebelumnya, lakukan proses penyisipan.

	
	Hasil akhir setelah pengurutan dengan metode insertion sort.

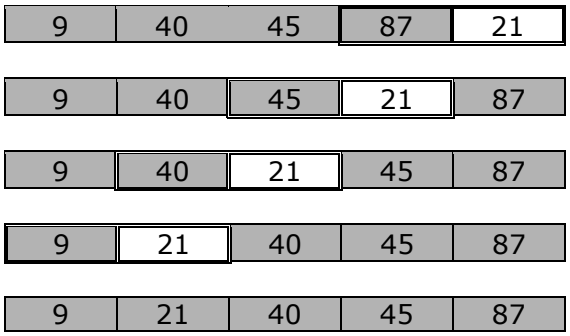
Cara penyisipan

Operasi utama dari algoritma insertion sort adalah penyisipan data (insertion), yaitu menyisipkan suatu nilai ke posisi yang tepat di bagian array yang sudah diurutkan (sorted array). ada beberapa jenis cara penyisipan ini, yaitu:

- logika pertukaran kebawah (shifting down)
- logika pergeseran tanpa pertukaran (Shifting instead of swapping)
- logika pencarian posisi dengan logika binary search.

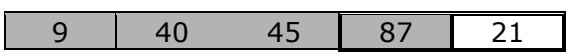
Logika pertukaran kebawah (shifting down)

Salah satu cara mudah penyisipan yaitu dengan cara menukar posisi dari posisi sekarang ke posisi sebelumnya hingga ditemukan tempat yang tepat. posisi yang tepat ini akan dicapai jika data sebelumnya lebih besar dibanding data yang akan disisipkan atau jika sudah mencapai di batas awal array.

Logika pergeseran tanpa pertukaran (Shifting instead of swapping)

Logika ini seperti halnya logika pertukaran kebawah, tetapi tanpa perlu melakukan pertukaran. jika sudah ditemukan tempat yang tepat, baru dilakukan proses penyisipan.

	Pergeseran dilakukan di akhir setelah ditemukan posisi yang tepat.

9	40	45	?	87
9	40	?	45	87
9	?	40	45	87
9	21	40	45	87

Logika pencarian posisi dengan logika binary search.

Untuk mencari posisi yang tepat dalam metode penyisipan ini adalah dengan metode binary search, sehingga metode penyisipan ini sering disebut dengan metode pengurutan **binary insertion sort**. Setelah posisi penyisipan ditemukan, baru dilakukan penyisipan. dengan menggunakan metode ini diharapkan pembandingan data lebih sedikit dilakukan dibanding metode penyisipan yang lain. Meskipun demikian, dalam prakteknya metode ini tidak terlalu penting dilakukan, mengingat metode pengurutan insertion sort hanya digunakan pada data yang jumlah elemennya sedikit.

Analisa kekomplekan

Secara umum, dalam beberapa kasus, kekomplekan Insertion sort sebesar $O(n^2)$. pengurutan Inserten sort pada data array berjumlah sedikit yang didalamnya sudah sebagian besar urut akan menunjukkan performan yang sangat baik, bahkan jumlah operasi hingga O(n). secara umum jumlah penulisan/perubahan sebesar $O(n^2)$, tetapi pergeseran angka sangat tergantung pada algoritma penyisipan yang digunakan. Jika menggunakan metode pertukaran, jumlah operasi mencapai $O(n^2)$, sedangkan jika menggunakan metode binary sejumlah O(n log n).

Kode Program

```
void insertionSort(int arr[], int length) {
    int i, j, tmp;
    for (i = 1; i < length; i++) {
        j = i;
        while (j > 0 && arr[j - 1] > arr[j]) {
            tmp = arr[j];
            arr[j] = arr[j - 1];
            arr[j - 1] = tmp;
            j--;
        }
    }
}
```

Quicksort

Quicksort merupakan algoritma cepat dalam pengurutan data yang sering digunakan tidak hanya dalam pendidikan saja. Secara normal kompleksitas operasi sebanyak O(n log n), sehingga quicksort dapat diaplikasikan dalam data baik yang berjumlah kecil maupun berjumlah besar.

Algoritma

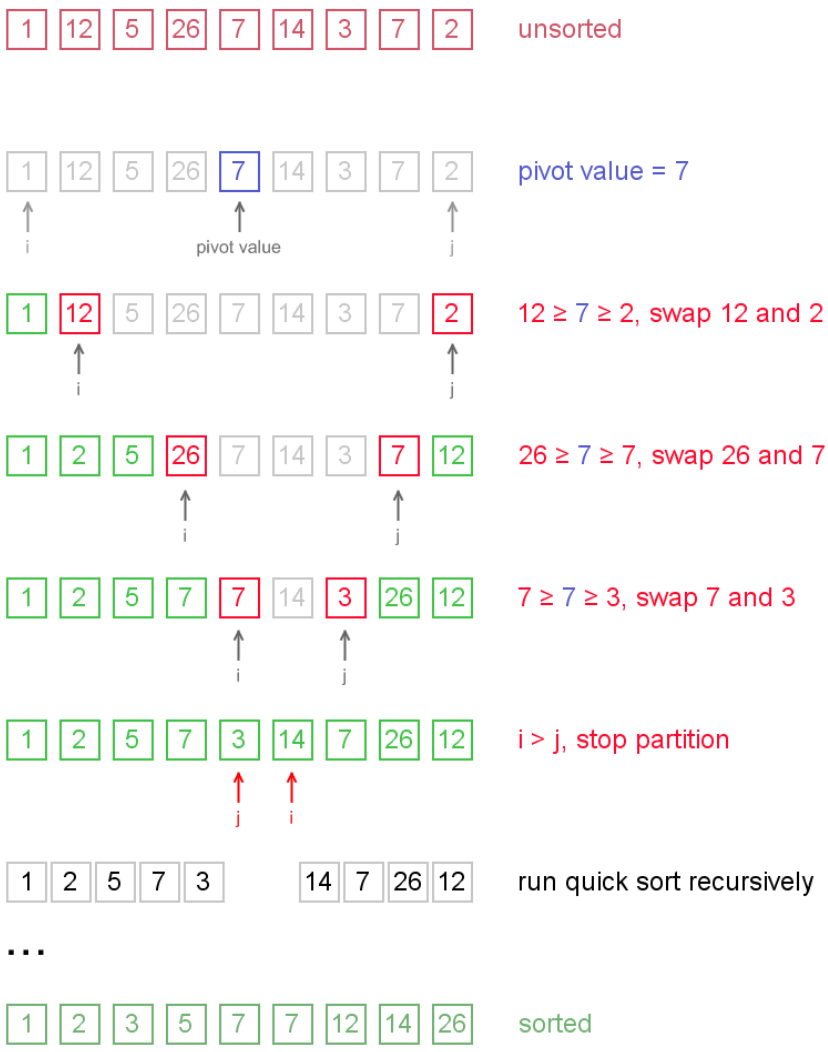
Strategi pisahkan dan menang digunakan dalam quicksort. Berikut algoritma quick sort:

1. **Pilih salah nilai sebagai pivot (pivot value).** nilai pivot bisa diambil dari nilai array yang berada di tengah-tengah, atau sembarang nilai yang ada pada range data yang akan diurutkan.
2. Pembagian. Atur ulang elemen-elemen, di mana data yang lebih kecil dari nilai pivot dipindahkan ke sebelah kiri dan data yang lebih besar dipindahkan ke sebelah kanan pivot. data yang sama dengan pivot tidak perlu dipindahkan kemanapun. perlu diperhatikan bahwa array belum tentu dibagi dalam jumlah elemen yang sama.
3. Anggap data yang dijadikan pivot termasuk dalam salah satu bagian (bagian kir atau kanan).
4. Dengan metode pengurutan yang sama, yaitu quicksort, urutkan array yang berada di sebelah kiri pivot. urutkan pula data yang ada di sebelah kanan pivot.

Algoritma pembagian secara detail

1. Ada 2 index, i dan j. di permulaan algoritma pembagian, i dianggap sebagia awal dari elemen array bagian pertama. dan j menunjukan akhir elemen dari array bagian kedua.
2. Secara berjalan, i bertambah kedepan sampai elemen yang di dalamnya lebih besar atau sama dengan elemen pivot ditemukan (dt[i]>=dt[pivot]). sedangkan index j berkurang sampai ditemukan elemen yang lebih kecil atau sama dengan data pivot (dt[j]<=dt[pivot]).
3. jika i ≤ j maka tukarkan dt[i] dengan dt[j].
4. lanjutkan langkah i dan j, yaitu i=i+1 dan j=j+1. jika **i >j** maka algoritma dihentikan. jika tidak, ulangi dari langkah 2.

Contoh: urutkan {1, 12, 5, 26, 7, 14, 3, 7, 2} menggunakan quicksort.



Perlu diperhatikan bahwa ilustrasi yang di tampilkan di atas hanya langkah rekursi pertama. untuk selanjutnya, {1, 2, 5, 7, 3} dan {14, 7, 26, 12} akan diurutkan secara recursi dengan metode yang sama.

Pada langkah pembagian ke dalam 2 bagian, setiap elemen a dari partisi kiri selalu lebih kecil atau sama dengan setiap elemen yang ada di partisi b. **a ≤ pivot ≤ b**. setelah recursi selesai, maka array akan urut secara keseluruhan.

Analisa kekomplekan

umumnya quicksort mempunyai kompleksitas O(n log n). dalam kasus terburuk, quicksort menjalankan operasi sebanyak $O(n^2)$ kali, tetapi dalam prakteknya, biasanya hal ini berjalan dengan baik, bahkan lebih baik dari O(n log n).

Kode Program

```
void quickSort(int arr[], int left, int right) {
     int i = left, j = right;
     int tmp;
     int pivot = arr[(left + right) / 2];

     /* partition */
     while (i <= j) {
          while (arr[i] < pivot) i++;
          while (arr[j] > pivot) j--;
          if (i <= j) {
               tmp = arr[i];
               arr[i] = arr[j];
               arr[j] = tmp;
               i++;
               j--;
          }
     };

     /* recursion */
     if (left < j) quickSort(arr, left, j);
     if (i < right) quickSort(arr, i, right);
}
```

Contoh hasil pada array {45,40,9,87,21}

```
      45      40       9      87      21
pivot = 9
Pada saat akan pertukaran, i = 0 j= 2

       9      40      45      87      21
Pada saat akan rekursi kiri, i = 1 right= 4
pivot = 45
Pada saat akan pertukaran, i = 2 j= 4

       9      40      21      87      45
Pada saat akan rekursi kiri, left = 1 j= 2
pivot = 40
Pada saat akan pertukaran, i = 1 j= 2

       9      21      40      87      45
Pada saat akan rekursi kiri, i = 3 right= 4
pivot = 87
Pada saat akan pertukaran, i = 3 j= 4

       9      21      40      45      87

       9      21      40      45      87
```

Mergesort

Metode pengurutan yang lain adalam mergesort. secara garis besar, pengurutan merge sort pertama-tama membagi array dalam 2 bagian. setiap bagian array ini diurutkan lagi secara recursif dengan mergesort. setelah kedua bagian diurutkan, kedua array digabungkan menjadi satu sambil diurutkan, dengan cara membandingkan setiap elemen array antara array a dan b, jika elemen a lebih kecil

dari elemen b, maka elemen a yang berhak lebih dulu untuk digabungkan, begitu pula selanjutnya. meskipun jumlah elemen bisa jadi sama antara a dan b, belum tentu elemen a dan elemen b bergabung bersamaan., sehingga sisa dari elemen (array a atau array b) di masukkan kemudian.

algortima merge sort secara recursif

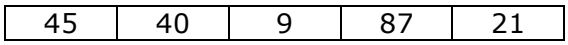
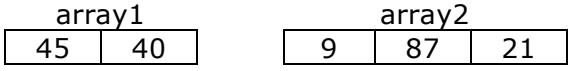
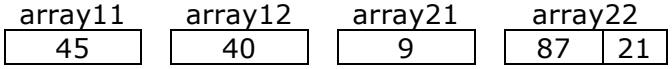
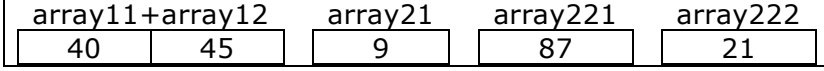
1. ketahui jumlah elemen array(n)
2. jika n>1
   - bagi array dalam 2 bagian, sehingga menjadi array1 (b) dan array2 (c)
   - dengan cara recursif, urutkan data array a dan array b dengan metode merge sort.
   - gabungkan array b dan array c dengan algoritma berjalan sehingga terdapat sebuah array baru d yang sudah urut. algoritma pengurutan dan penggabungan ini adalah:
     - i=0.j=0,k=0
     - selama i <p dan j<q
       jika a[i]=<b[j] maka
       - d[k]=a[i]
       - k++
       - i++

       jika a[i]>b[j] maka
       - d[k]=b[j]
       - k++
       - j++
     - selama penggabungan biasanya terdapat sisa data yang belum digabung. jika i=p berarti array a sudah digabungkan semua, sedangkan jika j=q berarti array b sudah digabungkan semua.
       - jika i=p,
         - selama j<q, c[k]=b[j], j++ dan k++
       - jika j=q
         - selama i<p, c[k]=b[i], j++ dan k++

Perhatikan contoh berikut:

Terdapat data 45, 40, 9, 87, 21

jumlah data (n)= 5

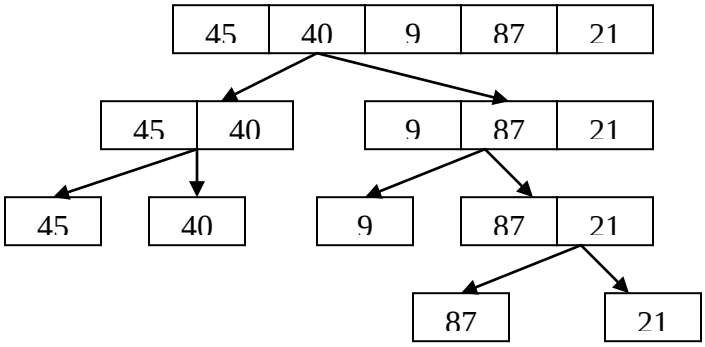
	data awal n=5
	a. array dibagi menjadi 2, jumlah elemen array a=int(n/2)=2 dan array b=n-2=3
	array a dibagi lagi menjadi 2, dan array b dibagi lagi menjadi 2
	array 11 dan array 12 digabungkan sambil

array1 40 | 45 array21 9 array221 87 array222 21	diurutkan, sehingga terbentuk array 11 sementara array 22 masih harus dibagi, karena jumlah elemen lebih dari 1
array1 40 | 45 array21 9 array221+array222 21 | 87 array1 40 | 45 array21 9 array22 21 | 87	gabungan array 11 dan array12 sementara dibiarkan menunggu. array 21 juga menunggu array221 dan array 222 digabungkan sambil diurutkan
array1 40 | 45 array21+array22 9 | 21 | 87 array1 40 | 45 array2 9 | 21 | 87	gabungan array 11 dan array12 sementara dibiarkan menunggu. array21 dan array 22 digabungkan sambil diurutkan
array1+array2 9 | 21 | 40 | 45 | 87 array1 9 | 21 | 40 | 45 | 87	gabunkan array 11 dan array12, sehingga tampak hasil akhir seperti di samping

jika n adalah jumlah perbandingan perlevel, maka jumlah level= log n dan total operasi= n log n

- log n levels

Ringkasan Pohon Recursi Merge Sort



RESPONSI IV

Nilai	**Materi**		**Nim**	
	Tanggal		**Nama**	
	Dosen		**Kelas**	

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

5
Pencarian (Search)

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui dasar-dasar pencarian.
- Mengetahui penggunaan algoritma pencarian.

Indikator Keberhasilan :
- Mampu menjelaskan dasar-dasar pencarian.
- Mampu menuliskan program dengan menggunakan pencarian.
- Mampu membuat program aplikasi menggunakan teori pencarian.

Dasar Teori

Pencarian/ search merupakan suatu proses di mana mencari data pada suatu deretan obyek.

Seperti halnya pengurutan/sort, algoritma pencarian juga bermacam macam, beberapa di antaranya adalah:

- Squential Search
- Sentinel Search
- Binary Search

Squential Search

Salah satu metode pencarian yang sering digunakan dan merupakan metode paling mudah adalah Squential Search, yaitu memandingkan satu-persatu antara data yang dicri dengan data yang ada pada array dari awal sampai akhir atau data yang dicari ditemukan. Metode pencarian squential ini tidak memandang apakah data sudah urut belum.

jika secara kebetulan data yang dicari berada di elemen-eleman awal dari array, maka proses pencarian cepat dilakukan. sebaliknya jika data yang dicari berada di akhir atau bahkan tidak ditemukan, maka proses pencarian menjadi lebih lama. jumlah langkah pembandingan jika data yang dicari tidak ditemukan adalah sejumlah elemen array (n).

Algoritma

1. masukkan array
2. ketahui jumlah elemen(n), mulai dari data pertama, i=1;
3. bandingkan array ke-i dengan data yang dicari. jika tidak sama, maka lanjutkan dengan menambahkan i=i+1
4. jika i>jumlah elemen array (n), maka pencarian dihentikan dan dianggap tidak ketemu dan selesai. jika i<=jumlah elemen array, maka ulangi dari langkah 3

Contoh:

Terdapat data sebagai berikut: {2,4,26,82,49}

Dari data ini akan dicari angka 4

Proses pencarian:

- Ketahui jumlah elemen (n), n=5, i=1
- Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[1]=2 sedangkan data yang dicari=4, sehingga tidak sama, maka lanjutkan pencarikan ke denga cara menambahkan i, i=i+1=2
- i=2, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[2]=4 sedangkan data yang dicari=4, sehingga data yang dicari ketemu. proses pencarian selesai.

Jika yang dicari adalah angka 50,

Proses pencarian:

- Ketahui jumlah elemen (n), n=5, i=1
- Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[1]=2 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarikan ke data berikutnya dengan cara menambahkan i, i=i+1=2
- i=2, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[2]=4 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarian ke data berikutnya dengan cara menambahkan i, i=i+1=3
- i=3, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[3]=26 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarian ke data berikutnya dengan cara menambahkan i, i=i+1=4
- i=4, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[4]=82 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarian ke data berikutnya dengan cara menambahkan i, i=i+1=5
- i=2, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[5]=49 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarian ke data berikutnya dengan cara menambahkan i, i=i+1=6
- i=6, sedangkan n=5, sehingga i>n, maka pencarian dihentikan dan dinyatakan bahwa data yang dicari tidak ditemukan. proses selesai.

Binary Search

Secara umum. Untuk mencari suatu nila di array yang tidak urut, diperlukan pencarian dengan cara membandingkan saut persatu.sampai nilai yang dicari ketemu . apabila nilai yang dicari tidak ditemukan di dalam array, maka secara otomatis semua elemen array sudah dibandingkan satu persatu.

Berbeda secara signifikan ketika array sudah urut sebelumnya. Pencarian dapat dilakukan jauh lebih cepat. Salah satu cara pencarian dengan cepat pada array yang sudah uruta adalah pencarian secara biner (binary search).bisa dibayangkan, jika terdapat sebuah array dengan elemen sebanyak 1000000 (satu juta) elemen , maka untuk mencari suatu jnilai dalam array tersebut cukup 20 langkah pencarian yang dihitung dari log2(1000000).

Algoritma

5. masukkan array
6. urutkan array
7. ketahui jumlah elemen array (n), batas bawah(bb), batas atas (ba), dan titik tengah (tt) yang dihitung dengan cara bb=1 ba= jumlah elemen array, dan tt= int((ba+bb)/2)
8. cari nilai dari elemen tengah-tengah (tt)
9. jika nilai elemen tengah-tengah sama dengan yang dicari, maka **algoritma dihentikan dan dinyatakan pencarian ketemu dan selesai**.
10. jika tidak sama, ada 2 kemungkinan:
    - jika nilai yang dicari < nilai elemen tengah, ini menunjukan bahwa nilai yang dicari jika ada pasti berada di bawah elemen tengah, sehingga pencarian dipersempit. ba=tt-1
    - jika nilai yang dicari > nilai elemen tengah, ini menunjukan bahwa nilai yang dicari jika ada pasti berada di setelah elemen tengah, sehingga pencarian dipersempit. bb=tt+1
11. Ada 3 kriteria
    - jika bb>ba, maka **pencarian dihentikan dan dinyatakan tidak ketamu dan selesai**,
    - jika bb<=ba, ulangi dari langkah 3

Contoh:

Terdapat data sebagai berikut:

int dt[9]={2,4,6,8,9,21,40,45,87}

Dari data ini akan dicari angka 4

Proses pencarian:

- Ketahui jumlah elemen (n),batas bawah (bb), batas atas(ba) dan titik tengah(tt)
  n=9, bb=1, ba=9, tt= int((1+9)/2)=5
- Bandingkan data yang ada pada tt dengan yang dicari
  dt[5]=9, sedangkan yang dicari 4, sehingga 4<9, alias dinyatakan belum ketemu. lanjutkan ke langkah berikutnya
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt].
  dalam hal ini, data yang dicari angka 4 dan data tengah dt[tt] adalah 9, sehingga:
  tidak mungkin data yang dicari berada di elemen kelima sampak akhir, sehingga batas atas(ba) diubah menjadi titik tengah (tt)-1, ba=5-1=4

- setelah dilakukan perubahan batas, tampak bahwa bb tidak berubah, yaitu bb=1, sedangkan ba=4, bb<ba, proses dilanjutkan untuk mencari nilai tengah yang baru (kembali ke langkah 3)
- tt=int((bb+ba)/2) = int((1+4) /2) = 2
- bandingkan antara data tengah (dt[tt]) dengan data yang dicari dt[tt]=dt[2]=4, sedangkan yang dicari=4, sehingga dt[tt]=data yang dicari. hal ini menunjukkan bahwa data yang dicari sudah ketemu. pencarian dihentikan dan dinyatakan selesai.

Jika yang dicari adalah angka 50,

Proses pencarian:

- Ketahui jumlah elemen (n),batas bawah (bb), batas atas(ba) dan titik tengah(tt)
  n=9, bb=1, ba=9, tt= int((1+9)/2)=5
- Bandingkan data yang ada pada tt dengan yang dicari dt[5]=9, sedangkan yang dicari 50, sehingga 50>9, alias dinyatakan belum ketemu. lanjutkan ke langkah berikutnya
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt].
  dalam hal ini, data yang dicari angka 50 dan data tengah dt[tt] adalah 9, sehingga:
  tidak mungkin data yang dicari berada di elemen pertama sampak tengah (5), sehingga batas bawah(bb) diubah menjadi titik tengah (tt)+1, bb=5+1=6
- setelah dilakukan perubahan batas, tampak bahwa ba tidak berubah, yaitu ba=9, sedangkan bb=6, bb<ba, proses dilanjutkan untuk mencari nilai tengah yang baru (kembali ke langkah 3)
- tt=int((bb+ba)/2) = int((6+9) /2) = 7
- bandingkan antara data tengah (dt[tt]) dengan data yang dicari dt[tt]=dt[7]=40, sedangkan yang dicari=50, sehingga dt[tt]<data yang dicari. pencarian dilanjutkan
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt].
  dalam hal ini, data yang dicari angka 50 dan data tengah dt[tt] adalah 40, sehingga:
  tidak mungkin data yang dicari berada di elemen pertama sampak tengah (5), sehingga batas bawah(bb) diubah menjadi titik tengah (tt)+1, bb=7+1=8
- setelah dilakukan perubahan batas, tampak bahwa ba tidak berubah, yaitu ba=9, sedangkan bb=8, bb<ba, proses dilanjutkan untuk mencari nilai tengah yang baru (kembali ke langkah 3)
- tt=int((bb+ba)/2) = int((8+9) /2) = 8
- bandingkan antara data tengah (dt[tt]) dengan data yang dicari dt[tt]=dt[8]=45, sedangkan yang dicari=50, sehingga dt[tt]<data yang dicari. pencarian dilanjutkan
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt].

dalam hal ini, data yang dicari angka 50 dan data tengah dt[tt] adalah 45, sehingga:
tidak mungkin data yang dicari berada di elemen pertama sampak tengah (), sehingga batas bawah(bb) diubah menjadi titik tengah (tt)+1, bb=8+1=9

- setelah dilakukan perubahan batas, tampak bahwa ba tidak berubah, yaitu ba=9, sedangkan bb=9, bb=ba, proses dilanjutkan untuk mencari nilai tengah yang baru (kembali ke langkah 3)
- tt=int((bb+ba)/2) = int((9+9) /2) = 9
- bandingkan antara data tengah (dt[tt]) dengan data yang dicari dt[tt]=dt[9]=87, sedangkan yang dicari=50, sehingga dt[tt]>data yang dicari. pencarian dilanjutkan
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt].
  dalam hal ini, data yang dicari angka 50 dan data tengah dt[tt] adalah 87, sehingga:
  sehingga batas bawah(bb) diubah menjadi titik tengah (tt)+1, bb=9+1=10
- setelah dilakukan perubahan batas, maka ba tidak berubah, yaitu 9, sedangkan bb=10, karena bb>ba, maka proses dihentikan dan dinyatakan tidak ketemu dan selesai.

Praktikum:

- Buatlah sebuah program untuk mempraktikan pencarian dengan metode squential search
- Buat pula program untuk mempraktekkanb metode Binary Search

RESPONSI V

Nilai	**Materi**		**Nim**	
	Tanggal		**Nama**	
	Dosen		**Kelas**	

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

5
Struktur Data Dinamis

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui dasar-dasar struktur data dinamis.
- Mengetahui penggunaan struktur data dinamis.

Indikator Keberhasilan : - Mampu menjelaskan dasar-dasar struktur data dinamis.
- Mampu menuliskan program dengan menggunakan struktur data dinamis.
- Mampu membuat program aplikasi menggunakan teori struktur data dinamis.

Dasar Teori

Konsep dasar struktur data dinamis adalah alokasi memori yang dilakukan secara dinamis. Pada konsep ini, terdapat suatu struktur yang disebut dengan struktur referensi diri (self-referential structure), mempunyai anggota pointer yang menunjuk ke struktur yang sama dengan dirinya sendiri.

Struktur data dinamis sederhana dapat dibagi menjadi empat jenis, yaitu :

1. Linked list
2. Stack
3. Queue
4. Binary tree

Definisi ini dapat dituliskan secara sederhana dengan struktur :

```
struct node {
      int info;
      struct node *nextPtr;
}
```

Struktur ini mempunyai dua anggota, yaitu bilangan bulat info dan pointer nextPtr. Variabel pointer nextPtr ini menunjuk ke struktur bertipe struct node, yang sama dengan deklarasi induknya.

Untuk membuat dan menangani struktur data dinamis dibutuhkan alokasi memori dinamis agar tersedia ruang bagi node-node yang ada. Fungsi malloc, free, dan operator sizeof banyak digunakan dalam alokasi memori dinamis ini.
Contoh :

```
      typedef struct node NODE;
      typedef NODE *NODEPTR;
      NODEPTR newPtr = malloc (sizeof (NODE));
      newPtr -> info = 15;
      newPtr -> nextPtr = NULL;
```

Fungsi free digunakan untuk menghapus memori yang telah digunakan untuk node yang ditunjuk oleh pointer tertentu. Jadi free (newPtr); akan menghapus memori tempat node yang ditunjuk oleh newPtr.

A. DEFINISI LINKED LIST

Linked list adalah suatu cara untuk menyimpan data dengan struktur sehingga dapat secara otomatis menciptakan suatu tempat baru untuk menyimpan data yang diperlukan. Program akan berisi suatu struct atau definisi kelas yang berisi variabel yang memegang informasi yang ada didalamnya, dan mempunyai suatu pointer yang menunjuk ke suatu struct sesuai dengan tipe datanya.

Struktur dinamis ini mempunyai beberapa keuntungan dibanding struktur array yang bersifat statis. Struktur ini lebih dinamis, karena banyaknya elemen dengan mudah ditambah atau dikurangi, berbeda dengan array yang ukurannya bersifat tetap.
Manipulasi setiap elemen seperti menyisipkan, menghapus, maupun menambah dapat dilakukan dengan lebih mudah.

Contoh :

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

typedef struct nod {
      int data;
      struct nod *next;
} NOD, *NODPTR;

void CiptaSenarai(NODPTR *s)
{
      *s = NULL;
```

```
}

NODPTR NodBaru(int m){
      NODPTR n;

      n = (NODPTR) malloc(sizeof(NOD));
      if(n != NULL) {
            n -> data = m;
            n -> next = NULL;
      }
      return n;
}

void SisipSenarai (NODPTR *s, NODPTR t, NODPTR p)
{
      if (p==NULL) {
            t -> next = *s;
            *s = t;
      }
      else {
            t -> next = p -> next;
            p ->next = t;
      }
}

void CetakSenarai (NODPTR s)
{
      NODPTR ps;
      for (ps = s; ps != NULL; ps = ps -> next)
            printf("%d -->", ps -> data);
      printf("NULL\n");
}

int main ()
{
      NODPTR pel;
      NODPTR n;

      CiptaSenarai(&pel);
      n = NodBaru(55);
      SisipSenarai(&pel, n, NULL);
      n= NodBaru(75);
      SisipSenarai(&pel, n, NULL);
      CetakSenarai(pel);
      return 0;
}
```

Bila program dijalankan maka :

75->55->NULL

B. TEKNIK DALAM LINKED LIST

Pengulangan Linked List

Secara umum pengulangan ini dikenal sebagai while loop. Kepala pointer (head pointer) dikopikan dalam variabel lokal current yang kemudian dilakukan perulangan dalam linked list. Hasil akhir dalam linked list dengan current!=NULL. Pointer lanjut dengan current=current->next.

Contoh :

```
// Return the number of nodes in a list (while-loop version)
int Length(struct node * head) {
      int count = 0;
      struct node* current = head;
      while (current != NULL) {
            count++;
            current=current->next;
      }
      return (count);
}
```

Mengubah Pointer Dengan Referensi Pointer

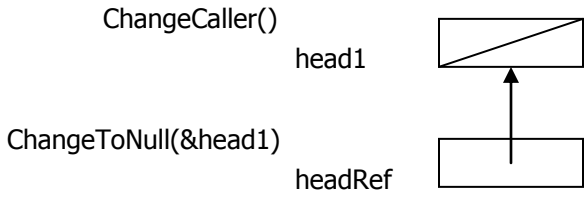
Banyak fungsi pada linked list perlu untuk merubah pointer kepala. Pointer ke pointer disebut dengan "reference pointer". Langkah utamanya dalam teknik ini adalah :

- Merancang fungsi untuk mengambil pointer ke pointer kepala. Untuk melewati pointer ke "value of interest" yang dibutuhkan untuk diubah. Untuk mengubah struct node*, lewati struct node**.
- Gunakan '&' dalam panggilan untuk menghitung dan melewati pointer ke value of interest.
- Gunakan '*' pada parameter dalam fungsi pemanggil untuk mengakses dan mengubah value of interest.

Contoh :

```
void ChangeToNull (struct node** headRef)
*headRef=NULL;
void ChangeCaller() {
      struct node* head1;
      struct node* head2;
      ChangeToNull (&head1);
      ChangeToNull (&head2);
}
```

Fungsi sederhana tersebut akan membuat pointer kepala ke NULL dengan menggunakan parameter reference. Gambar berikut menunjukkan bagaimana pointer headRef dalam ChangeToNull menunjuk ke head1 pada Change Caller.



Gambar 1. Pointer headRef dalam ChangeToNull menunjuk ke head1

Membuat Kepala Senarai Dengan Perintah Push()

Cara termudah untuk membuat sebuah senarai dengan menambah node pada "akhir kepala" adalah dengan push().

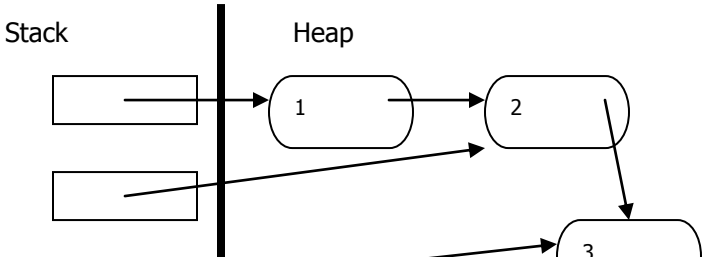
Contoh :

```
struct node* AddAtHead() {
      struct node* head = NULL;
      int i;
      for ( i=1; i<6; i++) {
            push (&head, i);
      }
      // head == { 5, 4, 3, 2, 1 };
      return (head);
}
```

Dalam membuat tambahan node pada akhir senarai (list) dengan menggunakan perintah Push(), jika terjadi kesalahan maka urutan akan terbalik.

Membuat Ekor Pada Akhir Senarai

Menambah ekor pada senarai akan melibatkan penempatan node terakhir, dan kemudian merubahnya .next field dari NULL untuk menunjuk node baru seperti variabel tail dalam contoh yaitu menambah node 3 ke akhir daftar {1,2}



head

tail

newNode

Gambar 2. Membuat ekor pada akhir senarai

Untuk memasukkan atau menghapus node di dalam senarai, pointer akan dibutuhkan ke node sebelum posisi, sehingga akan dapat mengubah .next field.

Agar dapat menambahkan node di akhir ekor pada data senarai {1, 2, 3, 4, 5}, dengan kesulitan node pertama pasti akan ditambah pada pointer kepala, tetapi semua node yang lain dimasukkan sesudah node terakhir dengan menggunakan pointer ekor. Untuk menghubungkan dua hal tersebut adalah dengan menggunakan dua fungsi yang berbeda pada setiap permasalahan.
Fungsi pertama adalah menambah node kepala, kemudian melakukan perulangan yang terpisah yang menggunakan pointer ekor untuk menambah semua node yang lain. Pointer ekor akan digunakan untuk menunjuk pada node terakhir, dan masing-masing node baru ditambah dengan tail->next.

Contoh :

```
struct node* BuildWithSpecialCase () {
      struct node* head = NULL;
      struct node* tail;
      int i;
      push (&head, 1);
      tail = head;
      for (i=2; i<6; i++) {
            push (&(tail->next), i);
            tail = tail->next;
      }
      return (head);
}
```

Membuat Referansi Lokal

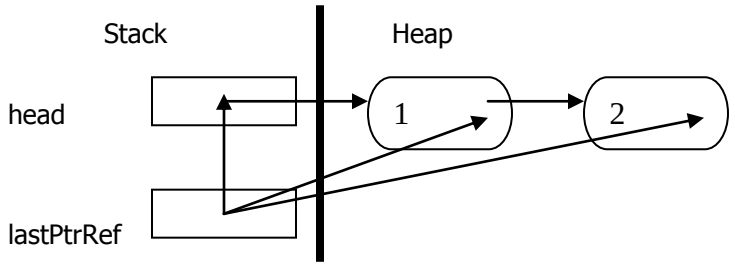
Dengan menggunakan "reference pointer" lokal, pointer akan selalu menunjuk ke pointer terakhir dalam senarai. Semua tambahan pada senarai dibuat dengan reference pointer. Reference pointer tidak menunjuk ke pointer kepala, tetapi menunjuk ke .next field didalam node terakhir dalam senarai.

Contoh :

```
struct node* BuildWithLocalRef() {
      struct node* head = NULL;
      struct node** lastPtrRef=&head;
      int i;

      for (i=2; i<6; i++) {
            Push (lastPtrRef, i);
            lastPtrRef=&((*lastPtrRef)->next);
      }
      return (head);
}
```

Cara kerja teknik ini adalah :

- Pada puncak putaran, lastPtrRef menunjuk pointer terakhir dalam senarai. Pada permulaannya menunjuk ke pointer kepala itu sendiri. Berikutnya menunjuk ke .next field di dalam node terakhir dalam senarai.
- Push(lastPtrRef); menambah node baru pada pointer akhir. Node baru menjadi node terakhir dalam senarai.
- lastPtrRef=&((*lastPtrRef)->next; lastPtrRef lanjut ke .next field dan .next field sekarang menjadi pointer terakhir dalam senarai.



Gambar 3. Membuat ekor pada akhir senarai

C. Operasi dalam Link List

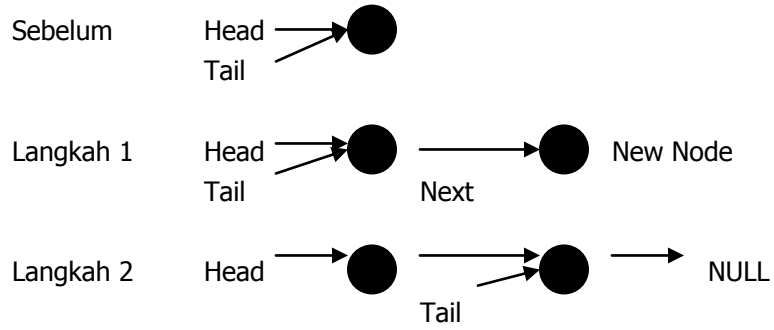
Menambah Node Baru

Untuk menambahkan sebuah node di dalam senarai, perlu didefinisikan sebuah kepala (head) dan ekor (tail). Pada awal senarai, posisi kepala dan ekor masih menjadi satu. Setelah ada tambahan node yang baru, maka posisinya berubah. Posisi kepala tetap pada awal senarai dan posisi ekor pada akhir senarai atau node yang baru. Bila ada tambahan node, maka posisi ekor akan bergeser sampai ke belakang dan akhir senarai ditunjukkan dengan NULL.

Contoh :

```
void SLList::AddANode()
{
      Tail->Next = new List;
      Tail=Tail->Next;
}
```

Ilustrasi penambahan Node adalah sebagai berikut :



Menghapus Node

Berikut ini adalah contoh program untuk menghapus node :

```
void SLList::DeleteANode(ListPtr corpse)
{
      ListPtr temp;
      if (corpse==Head) {
            temp=Head;
            Head=Head->Next;
            delete temp;
      }
      else if (corpse==Tail) {
            temp=Tail;
            Tail=Previous(Tail);
            Tail->Next=NULL;
            delete temp;
      }
      else {
            temp=Previous(corpse);
            temp->Next=corpse->Next;
            delete corpse;
      }
      CurrentPtr=Head;
}
```

Contoh latihan :

Buatlah program untuk memasukkan beberapa data dalam sebuah senarai (linked list), jika akan mengakhiri tekan n maka akan muncul semua node yang masuk ke dalam linked list tersebut.

Contoh program :

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

typedef struct node {
      int lkey;
      char name[10];
      struct node* next;
} TNODE;

TNODE *first, *last;

int LoadNode(TNODE *p);
void FreeNode(TNODE *p);
void PrintNode(TNODE *p);

void CreateList()
{
      TNODE *p;
      int n=sizeof(TNODE);

      first=last=0;
      for(;;)
      {
            if( (p=(TNODE*)malloc(n))==0 )
            {
                  printf("\nmemori tidak cukup");
                  break;
            }
            if(LoadNode(p)!=1)
            {
                  FreeNode(p);
                  break;
            }
```

```
            p->next=0;
            if (first==0)
                  first=last=p;
            else {
                  last->next=p;
                  last=p;
            }
      }
}

int LoadNode(TNODE *p)
{
      char opt;
      printf("\nMasukkan node baru?");
      opt=getche();
      opt=toupper(opt);
      if(opt!='N') {
            puts("\nMasukkan data untuk node:");
            printf("\nlkey:\t");
            if (scanf("%d",&(p->lkey))!=1) return 0;

            printf("\nname:\t");if (scanf("%s",p->name)!=1)
return 0;
            return 1;
      }
      else
            return -1;
}

void FreeNode(TNODE *p) {
      free(p);
}

void ViewAllList()
{
      TNODE *p;
      p=first;
      while(p) {
            PrintNode(p);
            p=p->next;
      }
}

TNODE* FindNode(int key)
{
```

```
      TNODE *p;
      p=first;
      while(p) {
            if(p->lkey == key) return p;
            p=p->next;
      }
      return 0;
}

void PrintNode(TNODE *p)
{
      if(p) printf("\n%d\t%s",p->lkey,p->name);
}

TNODE* InsertBeforeFirst()
{
      TNODE *p;
      int n=sizeof(TNODE);
      if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1))
      {
            if (first==0) {
                  p->next=0;
                  first=last=p;
            }
            else {
                  p->next=first;
                  first=p;
            }
            return p;
      }
      if(p==0)
            printf("\nMemori tidak cukup");
      else
            FreeNode(p);
      return 0;
}

TNODE* InsertBeforeKey(int key)
{
      TNODE *p, *q, *q1;
      int n=sizeof(TNODE);

      q1=0;
      q=first;
      while(q) {
```

```
            if(q->lkey == key) break;
            q1=q;
            q=q->next;
      }
      if(q==0) {
            printf("\nTidak ada node yang mempunyai kunci atau
senarai kosong");
            return 0;
      }
      if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1)) {
            if(q==first) {
                  p->next=first;
                  first=p;
            }
            else {
                  p->next=q;
                  q1->next=p;
            }
            return p;
      }
      if(p==0)
            printf("\nMemori tidak cukup");
      else
            FreeNode(p);
      return 0;
}
TNODE* InsertAfterKey(int key)
{
      TNODE *p, *q;
      int n=sizeof(TNODE);

      q=first;
      while(q) {
            if(q->lkey == key) break;
            q=q->next;
      }
      if(q==0) {
            printf("\nTidak ada node yang mempunyai kunci atau
senarai kosong");
            return 0;
      }
      if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1))
      {
            if(q==last) {
                  p->next=0;
```

```
                  last->next=p;
                  last=p;
            }
            else {
                  p->next=q->next;
                  q->next=p;
            }
            return p;
      }
      if(p==0)
            printf("\nMemori tidak cukup");
      else
            FreeNode(p);
      return 0;
}

TNODE* InsertAfterLast()
{
      TNODE *p;
      int n=sizeof(TNODE);
      if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1))
      {
            p->next=0;
            if (first==0)
                  first=last=p;
            else {
                  last->next=p;
                  last=p;
            }
            return p;
      }
      if(p==0)
            printf("\nMemori tidak cukup");
      else
            FreeNode(p);
      return 0;
}


void RemoveFirst()
{
      TNODE *p;

      if(first==0)
            return;
```

```
	if(first==last) {
		FreeNode(first);
		first=last=0;
		return;
	}
	p=first;
	first=first->next;
	FreeNode(p);
}

void RemoveLast()
{
	TNODE *p, *q;

	if(first==0)
		return;
	if(first==last) {
		FreeNode(first);
		first=last=0;
		return;
	}
	q=0;
	p=first;
	while(p!=last)	{
		q=p;
		p=p->next;
	}
	p=last;
	FreeNode(p);
	q->next=0;
	last=q;
}

void RemoveByKey(int key)
{
	TNODE *p, *q;

	if(first==0)
		return;
	q=0;
	p=first;
	while(p) {
		if(p->lkey == key) break;
		q=p;
		p=p->next;
```

```
	}
	if(!p) {
		printf("\nTidak ada node yang mempunyai kunci");
		return;
	}

	if(first==last) {
		FreeNode(first);
		first=last=0;
		return;
	}
	if(p==first) {
		first=first->next;
		FreeNode(p);
		return;
	}
	if(p==last) {
		q->next=0;
		last=q;
		FreeNode(p);
		return;
	}
	q->next=p->next;
	FreeNode(p);
}

void DeleteList()
{
	TNODE *p;

	p=first;
	while(p) {
		first=first->next;
		FreeNode(p);
		p=first;
	}
	last=0;
}

void main()
{
	CreateList();
	ViewAllList();
	InsertAfterLast();
	ViewAllList();
```

```
        RemoveFirst();
        ViewAllList();
        InsertAfterKey(1);
        ViewAllList();
        RemoveByKey(1);
        ViewAllList();
        DeleteList();
        ViewAllList();
}
```

RESPONSI VI

Nilai	**Materi**		**Nim**	
	Tanggal		**Nama**	
	Dosen		**Kelas**	

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....