

BAB 1

Struct, Array, dan Pointer

Tujuan :

1. Mahasiswa memahami apakah yang dimaksud dengan struktur data.
2. Mahasiswa memahami apakah yang dimaksud dengan algoritma.
3. Mengingat kembali array, struktur, pointer dalam bahasa C.

1.1 Pengenalan Struktur Data

Struktur data adalah sebuah skema organisasi, seperti struktur dan array, yang diterapkan pada data sehingga data dapat diinterpretasikan dan sehingga operasioperasi spesifik dapat dilaksanakan pada data tersebut

1.2 Pengenalan Algoritma

Algoritma adalah barisan langkah-langkah perhitungan dasar yang mengubah masukan (dari beberapa fungsi matematika) menjadi keluaran. Contoh :

Perkalian

Input : integer positif a, b

Output : $a \times b$

Algoritma perkalian :

Contoh kasus : $a = 365, b = 24$

$$\begin{aligned}\text{Metode 1 : } 365 * 24 &= 365 + (365 * 23) \\ &= 730 + (365 * 22) \\ &\quad \dots \\ &= 8760 + (365 * 0) \\ &= 8760\end{aligned}$$

$$\begin{array}{r} \text{Metode 2 : } \begin{array}{r} 365 \\ 24 \\ \hline 1460 \\ 730 \\ \hline 8760 \end{array} \end{array}$$

Manakah algoritma yang lebih baik ?

1.3 Array

Array adalah organisasi kumpulan data homogen yang ukuran atau jumlah elemen maksimumnya telah diketahui dari awal. Array umumnya disimpan di memori komputer secara kontigu (berurutan). Deklarasi dari array adalah sebagai berikut:

int A[5]; artinya variabel A adalah kumpulan data sebanyak 5 bilangan bertipe integer.

Operasi terhadap elemen di array dilakukan dengan pengaksesan langsung. Nilai di masing-masing posisi elemen dapat diambil dan nilai dapat disimpan tanpa melewati posisi-posisi lain.

Terdapat dua tipe operasi, yaitu:

1. Operasi terhadap satu elemen/posisi dari array
2. Operasi terhadap array sebagai keseluruhan

Dua operasi paling dasar terhadap satu elemen/posisi adalah

1. Penyimpanan nilai elemen ke posisi tertentu di array
2. Pengambilan nilai elemen dari posisi tertentu di array

1.3.1 Penyimpanan dan Pengambilan Nilai

Biasanya bahasa pemrograman menyediakan sintaks tertentu untuk penyimpanan dan pengambilan nilai elemen pada posisi tertentu di array.

Contoh:

A[10] = 78, berarti penyimpanan nilai 78 ke posisi ke-10 dari array A
C = A[10], berarti pengambilan nilai elemen posisi ke-10 dari array A

1.3.2 Keunggulan dan Kelemahan Array

Keunggulan array adalah sebagai berikut:

1. Array sangat cocok untuk pengaksesan acak. Sembarang elemen di array dapat diacu secara langsung tanpa melalui elemen-elemen lain.
2. Jika berada di suatu lokasi elemen, maka sangat mudah menelusuri ke elemen-elemen tetangga, baik elemen pendahulu atau elemen penerus
3. Jika elemen-elemen array adalah nilai-nilai independen dan seluruhnya harus terjaga, maka penggunaan penyimpanannya sangat efisien

Kelemahan array. Array mempunyai fleksibilitas rendah, karena array mempunyai batasan sebagai berikut:

1. Array harus bertipe homogen. Kita tidak dapat mempunyai array dimana satu elemen adalah karakter, elemen lain bilangan, dan elemen lain adalah tipe-tipe lain
2. Kebanyakan bahasa pemrograman mengimplementasikan array statik yang sulit diubah ukurannya di waktu eksekusi. Bila penambahan dan pengurangan terjadi terus-menerus, maka representasi statis
 - Tidak efisien dalam penggunaan memori
 - Menyiaikan banyak waktu komputasi
 - Pada suatu aplikasi, representasi statis tidak dimungkinkan

1.4 Pointer

Misalnya kita ingin membuat beberapa penunjuk ke blok penyimpan yang berisi integer. Deklarasi pada C adalah:

```
int *IntegerPointer;
```

Tanda asterik (*) yang berada sebelum nama variable IntegerPointer menandakan ‘pointer pada suatu int’. Jadi deklarasi diatas berarti ‘definisikan sebuah tipe yang terdiri dari pointer bertipe integer yang bernama IntegerPointer’. Apabila didepannya ditambahkan typedef sebagai berikut

```
typedef int *IntegerPointer;
```

Berarti IntegerPointer merupakan suatu tipe pointer berbentuk integer.

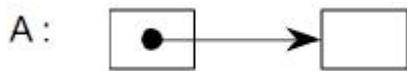
Apabila akan mendeklarasikan dua variable A dan B sebagai penunjuk ke bilangan integer :

```
IntegerPointer A, B;
```

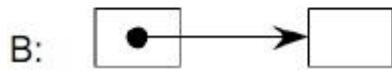
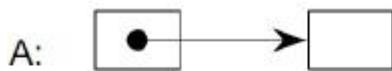
Berarti kompiler C akan berisi nilai dari variable A dan B yang ‘menunjuk ke integer’.

Untuk membuat beberapa penunjuk ke beberapa penyimpanan integer yang kosong dan untuk membuat A dan B menunjuk tempat tersebut, digunakan prosedur dinamis untuk alokasi penyimpanan yang disebut malloc

```
A = (IntegerPointer *) malloc (sizeof(int));
```

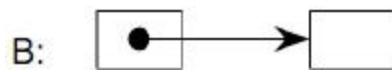


```
B = (int *) malloc (sizeof(int));
```



Misalnya kita akan menyimpan integer 5 pada blok penyimpanan yang ditunjuk pointer pada variable A. Untuk menyimpan angka 5 pada blok penyimpanan integer itu melalui pointer A, digunakan pernyataan :

```
*A = 5;
```



Linked list adalah salah satu struktur data yang paling fundamental. Linked list terdiri dari sejumlah kelompok elemen (*linked*) dengan urutan tertentu. Linked list sangat berguna untuk memelihara sekelompok data, semacam array, tetapi linked list lebih menguntungkan dalam beberapa kasus. Linked list lebih efisien dalam proses penyisipan (*insertion*) dan penghapusan (*deletion*). Linked list juga menggunakan pengalokasian penyimpanan secara dinamis, dimana penyimpanan dialokasikan pada saat waktu berjalan (*runtime*).

1.5 Struktur

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 satuan kesatuan.

Contoh sebuah struktur adalah informasi data tanggal, yang berisi: tanggal, bulan dan tahun.

1.5.1 Mendeklarasikan Struktur

Contoh pendefinisian tipe struktur adalah sebagai berikut:

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

yang mendefinisikan tipe struktur bernama data_tanggal, yang terdiri dari tiga buah elemen (field) berupa : tanggal, bulan dan tahun. Pendefnisian dan pendeklarasian struktur dapat juga ditulis sebagai berikut:

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

Bentuk umum dalam mendefinisikan dan mendeklarasikan struktur adalah sebagai berikut

```
struct nama_tipe_struktur {  
    tipe field1;  
    tipe field2;  
    ...  
    ...  
    tipe fieldn;  
} variabel_struktur1, ... , variabel_strukturM;
```

Masing-masing tipe dari elemen struktur dapat berlainan. Adapun variabel_struktur1 sampai dengan variabel_strukturM menyatakan bahwa variabel struktur yang dideklarasikan bisa lebih dari satu. Jika ada lebih dari satu variabel, antara variabel struktur dipisahkan dengan tanda koma.

1.5.2 Mengakses Elemen Struktur

Elemen dari struktur dapat diakses dengan menggunakan bentuk

```
variabel_struktur.nama_field
```

Antara variabel_struktur dan nama_field dipisahkan dengan operator titik (disebut operator anggota struktur). Contoh berikut merupakan instruksi untuk mengisikan data pada field tanggal

```
tgl_lahir.tanggal = 30;
```

1.6 Kesimpulan

1. Struktur data adalah sebuah skema organisasi yang diterapkan pada data sehingga data dapat diinterpretasikan dan sehingga operasi-operasi spesifik dapat dilaksanakan pada data tersebut
2. Apabila kita membuat program dengan data yang sudah kita ketahui batasnya, maka kita bisa menggunakan array (tipe data statis), namun apabila data kita belum kita ketahui batasnya, kita bisa menggunakan pointer (tipe data dinamis)
3. Untuk sekumpulan data dengan tipe data yang berlainan, namun merupakan satu-kesatuan, kita dapat menggunakan struktur untuk merepresentasikannya

2. LATIHAN PRAKTIKUM

Percobaan 1: Penggunaan array pada bilangan fibonacci

```
#include <stdio.h>
#define MAX 10

int fibo[MAX];

void main()
{
    int i;

    fibo[1] = 1;
    fibo[2] = 1;

    for (i=3;i<=MAX;i++)
        fibo[i]=fibo[i-2]+fibo[i-1];

    printf("%d Bilangan Fibonaci Pertama adalah : \n",MAX);
    for (i=1;i<MAX;i++)
        printf("%d-",fibo[i]);
}
```

Percobaan 2: Program mengubah isi variabel melalui pointer

```
#include <stdio.h>

main()
{
    int y, x = 87;
    int *px;

    px = &x;
    y = *px;

    printf("Alamat x      = %p\n", &x);
    printf("Isi px        = %p\n", px);
    printf("Isi x         = %d\n", x);
    printf("Nilai yang ditunjuk oleh px = %d\n", *px);
    printf("Nilai y         = %d\n", y);
}
```

Percobaan 3: Program mengakses dan mengubah isi suatu variabel pointer

```
#include <stdio.h>

main()
{
    float d = 54.5f, *pd;

    printf("Isi d mula-mula = %g\n", d);

    pd = &d;
    *pd += 10;

    printf("Isi d sekarang  = %g\n", d);
}
```

Percobaan 4: Penggunaan pointer untuk bilangan fibonacci

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int *fibo;

void main()
{
    int i;

    fibo = (int *) malloc(MAX * sizeof(int));

    *(fibo + 1) = 1;
    *(fibo + 2) = 1;

    for (i=3;i<=MAX;i++)
        *(fibo + i)= (*(fibo + i - 2) + *(fibo + i - 1));

    printf("%d Bilangan Fibonaci Pertama adalah : \n",MAX);
    for (i=1;i<MAX;i++)
        printf("%d-",*(fibo+i));
}
```

Percobaan 5: Penggunaan struktur pada konversi koordinat polar ke koordinat cetersian

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

struct polar {
    double r;
    double alpha;
};

struct kartesian {
    double x;
    double y;
};
```

```
void main()
{
    struct polar pl;
    struct kartesian k1;

    printf("Masukkan nilai r untuk koordinat polar : ");
    scanf("%lf",&pl.r);

    printf("Masukkan nilai alpha untuk koordinat polar : ");
    scanf("%lf",&pl.alpha);

    k1.x = pl.r * cos(pl.alpha);
    k1.y = pl.r * sin(pl.alpha);

    printf(
        "Nilai koordinat kartesian untuk koordinat polar r= %2.2lf alpha=
        %2.2lf adalah:\n",pl.r,pl.alpha);
    printf("x = %2.2lf  y = %2.2lf",k1.x,k1.y);
}
```

Percobaan 6: Program struktur dalam array

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

struct dtnilai
{
    char nrp[10];
    char nama[20];
    double nilai;
};

struct dtnilai data[10];
int j=0;

void tambah_data()
{
    char jawab[2];
    while(1)
    {
        fflush(stdin);
        printf("NRP :");scanf("%s",&data[j].nrp);
        printf("Nama :");scanf("%s",&data[j].nama);
        printf("Nilai Test :");scanf("%lf",&data[j].nilai);

        printf("Ada data lagi(y/t):"); scanf("%s",&jawab);
        if(jawab[0] == 'y' || jawab[0] == 'Y') continue;
        else break;
    }
}
```

```
if((strcmp(jawab,"Y")==0) || (strcmp(jawab,"y")==0))
{
    j++;continue;
}
else if ((strcmp(jawab,"T")==0) || (strcmp(jawab,"t")==0))
    break;
}

void tampil()
{
    int i;
    printf("Data Mahasiswa yang telah diinputkan :\n");
    printf("NRP\tNama\tNilai\n");

    for (i=0;i<=j;i++)
    {
        printf("%s\t%s\t%6.2f\n",data[i].nrp,data[i].nama, data[i].nilai);
    }
}

void main()
{
    tambah_data();
    tampil();
}
```

3. TUGAS RUMAH

Tugas Rumah 1:

Aritmatika polinom

Masalah aritmatika polinom adalah membuat sekumpulan subrutin manipulasi terhadap polinom simbolis (symbolic Polynomial).

Misalnya:

$$\begin{aligned}P1 &= 6x^8 + 8x^7 + 5x^5 + x^3 + 15 \\P2 &= 3x^9 + 4x^7 + 3x^4 + 2x^3 + 2x^2 + 10 \\P3 &= x^2 + 5\end{aligned}$$

Terdapat empat operasi aritmatika polinom dasar antara lain:

Penambahan:

$$P1 + P2 = 3x^9 + 6x^8 + 12x^7 + 5x^5 + 3x^4 + 3x^3 + 2x^2 + 25$$

Pengurangan:

$$P1 - P2 = -3x^9 + 6x^8 + 4x^7 + 5x^5 - 3x^4 - x^3 - 2x^2 + 5$$

Perkalian:

$$P1 * P3 = 6x^{10} + 8x^9 + 5x^7 + x^5 + 15x^2 + 30x^8 + 40x^7 + 25x^5 + 5x^3 + 75 = \\6x^{10} + 8x^9 + 30x^8 + 45x^7 + 26x^5 + 5x^3 + 15x^2 + 75$$

Turunan:

$$P2' = 27x^8 + 28x^6 + 12x^3 + 6x^2 + 4x$$

Representasikan bilangan polinom dengan array dan buatlah prosedur-prosedur yang melakukan kelima operasi aritmatika di atas.

Tugas Rumah 2:
Bilangan kompleks

Bilangan kompleks berbentuk $a + bi$, dimana a dan b adalah bilangan nyata dan $i^2 = -1$. Terdapat empat operasi aritmatika dasar untuk bilangan kompleks, yaitu:

Penambahan:

$$(a+bi) + (c+di) = (a+c) + (b+d)i$$

Pengurangan:

$$(a+bi) - (c+di) = (a-c) + (b-d)i$$

Perkalian:

$$(a+bi) * (c+di) = (ac-bd) + (ad+bc)i$$

Pembagian:

$$(a+bi) / (c+di) = [(ac+bd) / (a^2+b^2)] + [(bc-ad) / (c^2+d^2)]i$$

Tulis program yang membaca dua bilangan kompleks dan simbol operasi yang perlu dilakukan, kemudian lakukan operasi yang diminta. Gunakan struktur untuk merepresentasikan bilangan kompleks dan gunakan prosedur untuk implementasi tiap operasi.

BAB 2

SORTING (PENGURUTAN)

1. Tujuan

Setelah mempelajari modul ini, mahasiswa diharapkan:

- a. Mampu menjelaskan mengenai algoritma Sorting
- b. Mampu membuat dan mendeklarasikan struktural algoritma Sorting
- c. Mampu menerapkan dan mengimplementasikan algoritma Sorting

2. Dasar Teori

Pengurutan data dalam struktur data sangat penting terutama untuk data yang bertipe data numerik ataupun karakter. Pengurutan dapat dilakukan secara ascending (urut naik) dan descending (urut turun). Pengurutan (Sorting) adalah proses pengurutan data yang sebelumnya disusun secara acak sehingga tersusun secara teratur menurut aturan tertentu.

Contoh:

Data Acak	3 9 6 21 10 1 13
Ascending	1 3 6 9 10 13 21
Descending	21 13 10 9 6 3 1

Deklarasi Array Sorting

Mendeklarasikan array secara global:

```
int data[100];  
int n; //untuk jumlah data
```

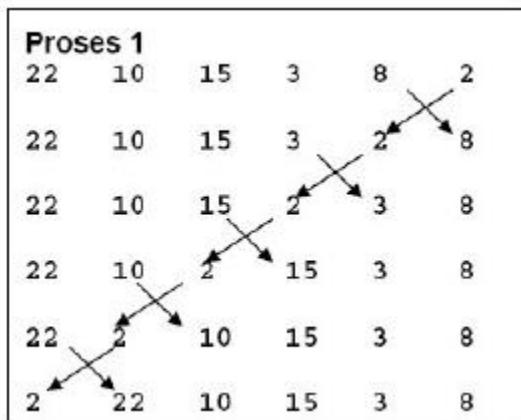
Fungsi Tukar 2 Buah Data:

```
void tukar (int a, int b) {  
    int tmp;  
    tmp = data [a] ;  
    data [a] = data [b] ;  
    data [b] = tmp ;
```

BUBBLE SORT

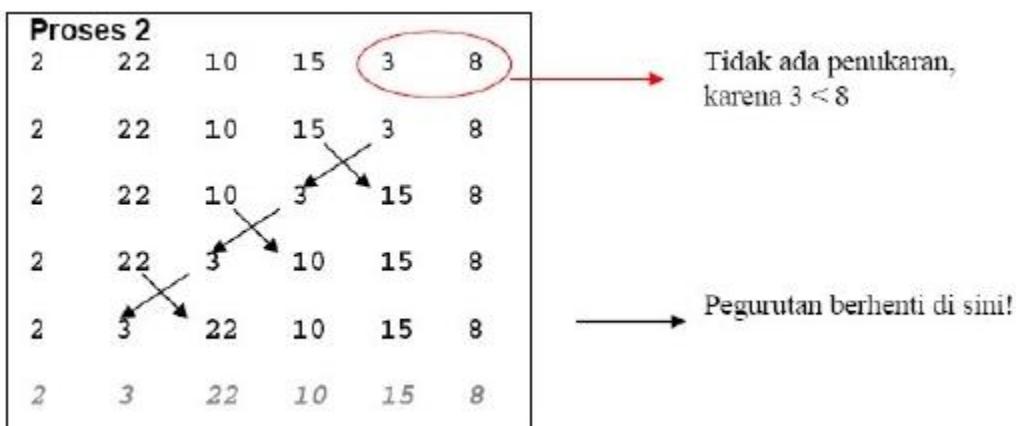
Merupakan metode sorting termudah, diberi nama “Bubble” karena proses pengurutan secara berangsur-angsur bergerak/berpindah ke posisinya yang tepat, seperti gelembung yang keluar dari sebuah gelas bersoda. *Bubble Sort* mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya. Jika elemen sekarang lebih besar dari elemen berikutnya maka kedua elemen tersebut ditukar, jika pengurutan ascending. Jika elemen sekarang lebih kecil dari elemen berikutnya, maka kedua elemen tersebut ditukar, jika pengurutan descending. Algoritma ini seolah-olah

menggeser satu per satu elemen dari kanan ke kiri atau kiri ke kanan, tergantung jenis pengurutannya. Ketika satu proses telah selesai, maka bubble sort akan mengulangi proses, demikian seterusnya. Kapan berhentinya? Bubble sort berhenti jika seluruh array telah diperiksa dan tidak ada pertukaran lagi yang bisa dilakukan, serta tercapai perurutan yang telah diinginkan.



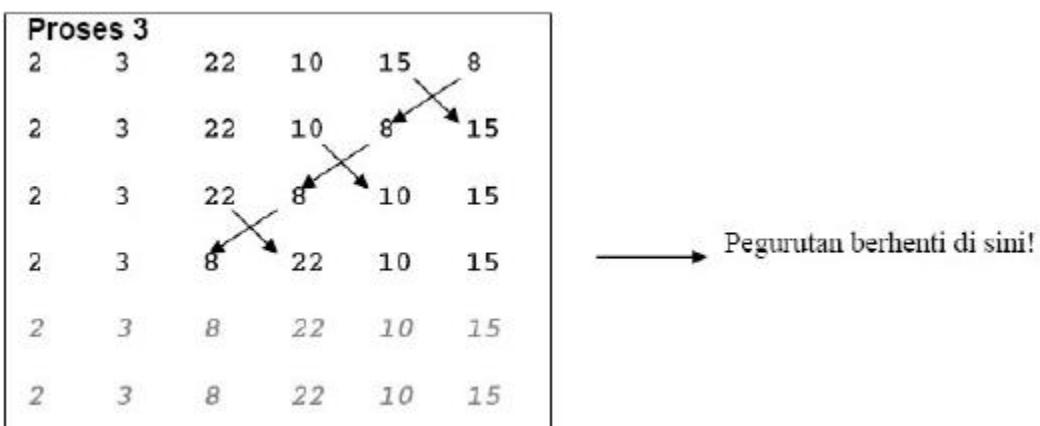
Gambar 1. Proses ke-1 Algoritma Bubble Sort

Pada gambar di atas, pengecekan dimulai dari data yang paling akhir, kemudian dibandingkan dengan data di depannya, jika data di depannya lebih besar maka akan ditukar.



Gambar 2. Proses ke-2 Algoritma Bubble Sorting

Pada proses ke-2, pengecekan dilakukan sampai dengan data ke-2 karena data pertama pasting sudah paling kecil.



Gambar 3. Proses ke-3 Algoritma Bubble Sorting

Proses 4	2	3	8	22	10	15	
	2	3	8	22	10	15	
	2	3	8	10	22	15	
	2	3	8	10	22	15	
	2	3	8	10	22	15	
	2	3	8	10	22	15	

Tidak ada penukaran, karena
10 < 15

Pengurutan berhenti di sini!

Gambar 4. Proses ke-4 Algoritma Bubble Sorting

Proses 5	2	3	8	10	22	15	
	2	3	8	10	15	22	
	2	3	8	10	15	22	
	2	3	8	10	15	22	
	2	3	8	10	15	22	
	2	3	8	10	15	22	

Pengurutan berhenti di sini!

Gambar 5. Proses ke-5 Algoritma Bubble Sorting

Sintaks program fungsi Bubble Sort

```
void bubble sort () {
    for (int i = 1; i<n; i++) {
        for (int j = n-1; j>=1; j--) {
            if (data [j] < data [j-1])
                tukar (j, j-1); // ascending
        }
    }
}
```

Dengan prosedur di atas, data terurut naik (ascending), untuk urut turun (descending) silahkan ubah bagian:

```
if (data [j] < data [j-1] )
```

Menjadi:

```

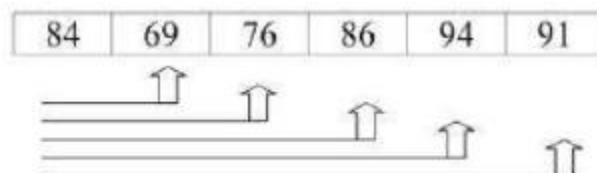
if (data [j] > data [j-1])
    tukar (j, j-1);

```

Algoritma Bubble Sorting mudah dalam sintaks, tetapi lebih hemat dibandingkan dengan algoritma sorting yang lain.

EXCHANGE SORT

Sangat mirip dengan Bubble Sort, dan banyak yang mengatakan Bubble Sort sama dengan Exchange Sort. Perbedaan ada dalam hal bagaimana membandingkan antar elemen-elemennya. Exchange sort membandingkan suatu elemen dengan elemen-elemen lainnya dalam array tersebut, dan melakukan pertukaran elemen jika perlu. Jadi ada elemen yang selalu menjadi elemen pusat (pivot). Sedangkan Bubble Sort akan membandingkan elemen pertama/terakhir dengan elemen sebelumnya/sesudahnya, kemudian elemen sebelum/sesudahnya itu akan menjadi pusat (pivot) untuk dibandingkan dengan elemen sebelumnya/sesudahnya lagi, begitu seterusnya.



Proses 1

Pivot (Pusat)					
84	69	76	86	94	91
84	69	76	86	94	91
84	69	76	86	94	91
86	69	76	84	94	91
94	69	76	84	86	91
94	69	76	84	86	91

Proses 2

Pivot (Pusat)					
94	69	76	84	86	91
94	76	69	84	86	91
94	84	69	76	86	91
94	86	69	76	84	91
94	91	69	76	84	86

Proses 3

94	91	69	76	84	86
94	91	76	69	84	86
94	91	84	69	76	86
94	91	86	69	76	84

Proses 4

94	91	86	69	76	84
94	91	86	76	69	84
94	91	86	84	69	76

Proses 5

94	91	86	84	69	76
94	91	86	84	76	69

Sintaks program fungsi Exchange Sorting

```
void exchange sort () {
{
    for (int i=0; i<n-1; i++) {
        for (int j=(i+1); j<n; j++) {
            if (data [i] < data[j])
                tukar(i,j); //descending
        }
    }
}
```

SELECTION SORT

Merupakan kombinasi antara sorting dan searching. Untuk setiap proses, akan dicari elemen-elemen yang belum diurutkan yang memiliki nilai terkecil atau terbesar akan dipertukarkan ke posisi yang tepat di dalam array. Misalnya untuk putaran pertama, akan dicari data dengan nilai terkecil dan data ini akan ditempatkan di indeks terkecil (data[0]), pada putaran kedua akan dicari data kedua terkecil, dan akan ditempatkan di indeks kedua (data[1]). Selama proses, pembandingan dan pengubahan hanya dilakukan pada indeks pembandingan saja, pertukaran data secara fisik terjadi pada akhir proses.

Proses 1

0	1	2	3	4	5
32	75	69	58	21	40

Pembanding Posisi

32 < 75	0
32 < 69	0
32 < 58	0
32 > 21 (tukar idx)	4
21 < 40	4

Tukar data ke-0 (32) dengan data ke-4 (21)

0	1	2	3	4	5
21	75	69	58	32	40

Proses 2

0	1	2	3	4	5
21	75	69	58	32	40

Pembanding Posisi

75 > 69 (tukar idx)	2
69 > 58 (tukar idx)	3
58 > 32 (tukar idx)	4
32 < 40	4

Tukar data ke-1 (75) dengan data ke-4 (32)

0	1	2	3	4	5
21	32	69	58	75	40

Proses 3

0	1	2	3	4	5
21	32	69	58	75	40

Pembanding Posisi

69 > 58 (tukar idx)	3
58 < 75	3
58 > 40	5

Tukar data ke-2 (69) dengan data ke-5 (40)

0	1	2	3	4	5
21	32	40	58	75	69

Proses 4

0	1	2	3	4	5
21	32	40	58	75	69

Pembanding Posisi

58 < 75	3
58 < 69	3

Tukar data ke-3 (58) dengan data ke-3 (58)

0	1	2	3	4	5
21	32	40	58	75	69

Proses 5

0	1	2	3	4	5
21	32	40	58	75	69

Pembanding Posisi

75 > 69	5
---------	---

Tukar data ke-4 (75) dengan data ke-5 (69)

0	1	2	3	4	5
21	32	40	58	69	75

Gambar 7. Proses Algoritma Selection Sorting

Sintaks program fungsi Selection Sorting

```
void selection_sort () {  
    for (int i=0; i<n-1; i++) {  
        pos = i;  
        for (int j=i+1; j<n; j++) {  
            if (data[j] < data[pos])  
                pos = j; //ascending  
        }  
        If (pos != i) tukar(pos, i);  
    }  
}
```

INSERT SORT

Mirip dengan cara orang mengurutkan kartu, selembar demi selembar kartu diambil dan disisipkan (insert) ke tempat yang seharusnya. Pengurutan dimulai dari data ke-2 sampai dengan data terakhir, jika ditemukan data yang lebih kecil, maka akan ditempatkan (di-insert) diposisi yang seharusnya. Pada penyisipan elemen, maka elemen-elemen lain akan bergeser ke belakang.

Proses 1

0	1	2	3	4	5
22	10	15	3	8	2

Temp	Cek	Geser
10	Temp<22?	Data ke-0 ke posisi 1

Temp menempati posisi ke -0

0	1	2	3	4	5
10	22	15	3	8	2

Proses 2

0	1	2	3	4	5
10	22	15	3	8	2

Temp	Cek	Geser
15	Temp<22	Data ke-1 ke posisi 2
15	Temp>10	-

Temp menempati posisi ke-1

0	1	2	3	4	5
10	15	22	3	8	2

Proses 3

0	1	2	3	4	5
10	15	22	3	8	2

Temp	Cek	Geser
3	Temp<22	Data ke-2 ke posisi 3
3	Temp<15	Data ke-1 ke posisi 2
3	Temp<10	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
3	10	15	22	8	2

Proses 4

0	1	2	3	4	5
3	10	15	22	8	2

Temp	Cek	Geser
8	Temp<22	Data ke-3 ke posisi 4
8	Temp<15	Data ke-2 ke posisi 3
8	Temp<10	Data ke-1 ke posisi 2
8	Temp>3	-

Temp menempati posisi ke-1

0	1	2	3	4	5
3	8	10	15	22	2

Proses 5

0	1	2	3	4	5
3	8	10	15	22	2

Temp	Cek	Geser
2	Temp<22	Data ke-4 ke posisi 5
2	Temp<15	Data ke-3 ke posisi 4
2	Temp<10	Data ke-2 ke posisi 3
2	Temp<8	Data ke-1 ke posisi 2
2	Temp<3	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
2	3	8	10	15	22

Gambar 9. Proses Algoritma Insertion Sorting

Sintaks program fungsi Insertion Sort

```
void insertion_sort () {  
    int temp;  
    for(int i=1; i<n; i++) {  
        temp = data[i];  
        j = i-1;  
        while (data[j]>temp && j>=0) {  
            data[j+1] = data[j];  
            j--;  
        }  
        Data[j+1] = temp;  
    }  
}
```

Program lengkapnya: (**PERCOBAAN LATIHAN**)

```
#include <stdio.h>  
#include <conio.h>  
int data[10], data2[10];  
int n;  
void tukar (int a, int b) {  
    int t;  
    t = data[b];  
    data[b] = data[a];  
    data[a] = t;  
}  
void bubble_sort () {  
    for (int i=1; i<n; i++) {  
        for (int j=n-1; j>=i; j--) {  
            if (data[j] < data[j-1])  
                tukar(j, j-1);  
        }  
    }  
    printf ("bubble sort selesai!\n");  
}  
void exchange_sort () {  
    for (int i=0; i<n-1; i++) {  
        for (int j = (i+1); j<n; j++) {  
            if (data [i] > data[j])  
                tukar (i, j);  
        }  
    }  
    printf ("exchange sort selesai!\n");  
}  
void selection_sort () {  
    int pos, i, j;  
    for (i=0; i<n-1; i++) {  
        pos = i;  
        for (j = i+1; j<n; j++) {  
            if (data [j] < data[pos])  
                pos = j;
```

```
        }
        if (pos != i) tukar (pos, i);
    }
    printf ("selection sort selesai!\n");
}
void insertion_sort () {
    int temp, i, j;
    for (i=1; i<n; i++) {
        temp = data[i];
        j = i-1;
        while (data[j] > temp && j>=0) {
            data[j+1] = data[j];
            j--;
        }
        data[j+1] = temp;
    }
    printf("insertion sort selesai!\n");
}
void Input () {
    printf ("Masukkan jumlah data = ");
    scanf ("%d", &n);
    for(int i=0; i<n; i++) {
        printf ("Masukkan data ke-%d = ", (i+1));
        scanf ("%d", &data[i]);
        data2[i] = data[i];
    }
}
void AcakLagi () {
    for (int i=0; i<n; i++) {
        data[i] = data2[i];
    }
    printf ("Data sudah teracak!\n");
}
void Tampil () {
    printf ("Data : ");
    for (int i=0; i<n; i++) {
        print ("%d ", data[i]);
    }
    printf ("\n");
}
void main() {
    clrscr();
    int pil;
    do {
        clrscr ();
        printf ("1. Input Data\n");
        printf("2. Bubble Sort\n");
        printf("3. Exchange Sort\n");
        printf("4. Selection Sort\n");
        printf("5. Tampilkan Data\n");
        printf("6. Acak\n");
        printf("7. Exit");
        printf("Pilihan = "); scanf("%d", &pil);
        switch(pil) {
        case 1: Input(); break;
        case 2: bubble_sort(); break;
        case 3: exchange_sort(); break;
        case 4: selection_sort(); break;
        case 5: Tampil(); break;
    }
}
```

```
        case 6: AcakLagi(); break;
    }
    getch();
}
while (pil!=7);
}
```

3. Latihan Praktikum

Latihan 1

Berikan algoritma dan penjelasan dari syntaks program di bawah ini!

```
/* Bubble Sorting */
#include <iostream>
#include <iomanip>

using namespace std;
int main()
{
    int NumList [10]={12,29,56,4,31,9,17,19,48,3};
    int temp;
    cout<<"Data Angka Sebelum diurutkan : \n";
    for (int d=0; d<10; d++)
    {
        cout<<setw(3)<<NumList[d];
    }
    cout<<"\n\n";
    for (int a=0; a<10; a++)
        for (int b=0; b<10; b++)
            if (NumList [b]>= NumList [b+1])
            {
                temp = NumList[b];
                NumList[b]= NumList [b+1];

                NumList[b+1]= temp;
            }
    cout<<"Data setelah diurutkan:\n";
    for (int c=0; c<10; c++)
        cout<<setw(3)<<NumList[c]<<"  ";
    cout<<endl;
}
```

Latihan 2

Berikan algoritma dan penjelasan dari syntaks program di bawah ini!

```
/*Selection Sorting */
#include <iostream>
#include <iomanip>
using namespace std;

void SelectionSort (int Array[], const int Size)
{
    int i,j,small, temp;
    for (i=0;i<Size;i++)
    {
        small=i;
        for (j=0;j<Size;j++)
        {
            if (Array[j]>Array[small]) //Pembanding
            {

                small = j;
                temp = Array[i];
                Array[i] = Array[small];
                Array[small]=temp;
            }
        }
    }
}

int main()
{
    int NumList [10]={12,29,56,4,31,9,17,19,48,3};
    int temp;
    cout<<"Data sebelum diurutkan: \n";
    for(int d=0; d<10; d++)
    {
        cout<<setw(3)<<NumList[d];
    }
    cout<<"\n\n";
    SelectionSort(NumList,10);

    cout<<"Data setelah diurutkan\n";
    for (int a=0; a<10; a++)
        cout<<setw(3)<<NumList[a]<<endl<<endl;
}
```

Latihan 3

Berikan algoritma dan penjelasan dari syntaks program di bawah ini!

```
/* Shell Shorting */
#include <iostream>
using namespace std;

int main()
{
    int array[5]; //An Array of integers
    int length=5; //Length of the array
    int i,j,d;
    int tmp,flag;

    //some input
    for(i=0;i<length;i++)
    {
        cout<<"enter a number : ";
        cin>>array[i];
    }

    //Algorithm
    d=length;
    flag=1;

    flag=1;
    while(flag || (d>1))
    {
        flag=0;
        d=(d+1)/2;
        for(i=0;i<(length-d);i++)
        {
            if(array[i+d]>array[i])
            {
                tmp=array[i+d];
                array[i+d]=array[i];
                array[i]=tmp;
                flag=1;
            }
        }
    }
    for(i=0;i<5;i++)
    {
        cout<<array[i]<<endl;
    }
}
```

Latihan 4

Berikan algoritma dan penjelasan dari syntaks program di bawah ini!

```
/*Quick Sorting*/
#include <iostream>
#include <iomanip>
using namespace std;

void quickSort (int[],int);
void q_sort (int[],int,int);

int main()
{
    int NumList[10]={12,29,56,4,31,9,17,19,48,3};
    int temp;
    cout<<"Data sebelum diurutkan:\n";
    for (int d=0;d<10;d++)
    {
        cout<<setw(3)<<NumList[d];
    }
    cout<<"\n\n";
    quickSort (NumList,10);
    cout<<"Data setelah diurutkan:\n";
    for (int iiii=0; iiii<10; iiii++)

        cout<<setw(3)<<NumList[iiii]<<endl<<endl;
}
void quickSort (int numbers[],int array_size)
{
    q_sort(numbers,0,array_size-1);
}
void q_sort (int numbers[], int left, int right)
{
    int pivot, l_hold, r_hold;
    l_hold=left;
    r_hold=right;
    pivot=numbers[left];

    while (left<right)
    {
        while ((numbers[right]>=pivot) && (left<right))
            right--;
        if (left!=right)
        {
            numbers[left]=numbers[right];
            left++;
        }

        while ((numbers[left]<=pivot)&& (left<right))
            left++;
        if (left!=right)
        {
            numbers[right]=numbers[left];
            right--;
        }
    }
    numbers[left]=pivot;
    pivot=left;
    left=l_hold;
    right=r_hold;
    if(left<pivot)
        q_sort(numbers,left,pivot-1);
    if (right>pivot)
        q_sort(numbers,pivot+1,right);
}
```

Latihan 5

Berikan algoritma dan penjelasan dari syntaks program di bawah ini!

```
/*Radix Sorting*/
#include <iostream>
#include <stdlib.h>
#include <string.h>
using namespace std;

void radix (int byte, long N, long *source, long *dest)
{
    long count[256];
    long index[256];
    int i;
    memset (count, 0, sizeof(count));
    for (i=0; i<N; i++) count[((source[i])>>(byte*8))&0xff]++;
    
    index[0]=0;
    for (i=1;i<256; i++) index [i]=index[i-1]+count[i-1];
    for (i=0; i<N; i++) dest[index[(source[i]>>(byte*8))&0xff]++]=source[i];
}
void radixsort (long *source, long*temp, long N)
{
    radix (0,N,source,temp);
    radix (1,N,temp,source);
    radix (2,N,source,temp);
    radix (3,N,temp,source);
}

void make_random (long *data, long N)
{
    for (int i=0; i<N; i++) data[i]=rand() | (rand()<<16);
}

long data[100];
long temp[100];
int main (void)
{
    make_random(data,100);
    radixsort (data, temp, 100);
    for (int i=0; i<100; i++) cout<<data[i]<<'\n';
}
```

4. Tugas Rumah

Buatlah sebuah program untuk mengurutkan sepasang data yang dimasukkan berdasarkan abjad/huruf.

Contoh

(Input) :

Masukan huruf ke-1 : w

Masukan angka ke-1 : 7

Masukan huruf ke-2 : d

Masukan angka ke-2 :2

Masukan huruf ke-3 : b

Masukan angka ke-3 :0

Masukan huruf ke-4 : a

Masukan angka ke-4 :8

Masukan huruf ke-5 : z

Masukan angka ke-5 :4

(Output)

Data Sebelum Diurutkan :

w	d	b	a	z
7	2	0	8	4

Urutan Berdasarkan Huruf :

a	b	d	w	z
8	0	2	7	4

Keterangan: Data yang dihasilkan setelah diurutkan TETAP BERPASANG-PASANGAN.

... SELAMAT MENGERJAKAN ...

BAB 4 SEARCHING

A. TUJUAN

1. Mahasiswa dapat melakukan perancangan aplikasi menggunakan struktur *Searching* (Pencarian)
2. Mahasiswa mampu melakukan analisis pada algoritma *Searching* yang dibuat
3. Mahasiswa mampu mengimplementasikan algoritma *Searching* pada sebuah aplikasi secara tepat dan efisien
4. Mahasiswa mampu menjelaskan mengenai algoritma *Searching*.
5. Mahasiswa mampu membuat dan mendeklarasikan struktur algoritma *Searching*.
6. Mahasiswa mampu menerapkan dan mengimplementasikan algoritma *Searching*.

B. ALOKASI WAKTU

4js (4x50 menit)

C. PETUNJUK

1. Awali setiap aktivitas dengan do'a yang khusuk, semoga berkah dan mendapat kemudahan.
2. Lakukan praktikum dengan cara;
 - a) Mengamati tujuan, dasar teori, dan latihan-latihan praktikum dengan baik dan benar.
 - b) Mencoba mengerjakan tugas-tugas sesuai dengan perintah.
 - c) Membuat hasil laporan praktikum

D. DASAR TEORI

1. Linked List

Secara umum search dapat diartikan mencari data dengan cara menelusuri tempat penyimpanan data tersebut. Tempat penyimpanan data dalam memory dapat berupa array atau dapat juga dalam bentuk Linked List.

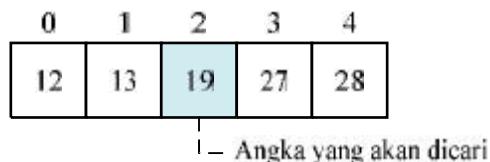
Pencarian dapat dilakukan terhadap data yang secara keseluruhan berada dalam memory komputer ataupun terhadap data yang berada dalam penyimpanan eksternal (*hard disk*).

1.1 Sequential Search

Teknik pencarian data dari array yang paling mudah adalah dengan cara sequential search, dimana data dalam array dibaca 1 demi satu, diurutkan dari index terkecil ke index terbesar, maupun sebaliknya.

Array

int A[5] = { 12, 13, 19, 27, 28 }



Gambar 1.1 Data Pencarian Sekuensial

Misalkan, dari data diatas angka yang akan dicari adalah angka 19 dalam array A, maka proses yang akan terjadi pada proses pencarian adalah sebagai berikut.

- pencarian dimulai pada index ke-0 yaitu angka 12, kemudian dicocokan dengan angka yang akan dicari, jika tidak sama makapencarian akan dilanjutkan ke index selanjutnya.
- Pada index ke-1, yaitu angka 13, juga bukan angka yang dicari, maka pencarian juga akan dilanjutkan pada index selanjutnya.
- Pada index ke-2, yaitu angka 19, ternyata angka 19 merupakan angka yang dicari. Pencarian angka telah ditemukan, maka pencarian akan dihentikan dan keluar dari looping pencarian.

1.2 Binary Search

Metode pencarian yang kedua adalah binary search, pada metode pencarian ini, data harus diurutkan terlebih dahulu. Pada metode pencarian ini, data dibagi menjadi dua bagian (secara logika), untuk setiap tahap pencarian.

Algoritma binary search :

- a) Data diambil dari posisi 1 sampai posisi akhir N
- b) Kemudian cari posisi data tengah dengan rumus: $(\text{posisi awal} + \text{posisi akhir}) / 2$
- c) Kemudian data yang dicari dibandingkan dengan data yang di tengah, apakah sama atau lebih kecil, atau lebih besar?
- d) Jika lebih besar, maka proses pencarian dicari dengan posisi awal adalah posisi tengah+1
- e) Jika lebih kecil, maka proses pencarian dicari dengan posisi akhir adalah posisi tengah-1
- f) Jika data sama, berarti ketemu.

1.3 Fibonacci Search

Fibonacci Search adalah pencarian sebuah elemen dalam array satu dimensi dengan menggunakan angka fibonacci sebagai titik-titik (indeks) elemen array yang isinya dibandingkan dengan nilai yang dicari. Sama halnya dengan Binary Search, Fibonacci Search juga mengharuskan data yang sudah terurut baik menaik (*ascending*) maupun menurun (*descending*).

1.4 Interpolation Search

Interpolation Search adalah pencarian sebuah elemen dalam array satu dimensi dengan metode interpolasi atau perkiraan secara interpolasi, dimana data harus diurutkan terlebih dahulu.

- a) Jika $\text{data}[\text{posisi}] > \text{data yg dicari}$, $\text{high} = \text{pos} - 1$
- b) Jika $\text{data}[\text{posisi}] < \text{data yg dicari}$, $\text{low} = \text{pos} + 1$

E. LATIHAN

1. Percobaan Program Sequential Search: mencoba, membuat, menampilkan sebuah program *Searching*.

Source Code :

```

1 #include <stdio.h>
2
3 int cari (int data[], int n, int k)
4 {
5     int posisi, i, ketemu;
6
7     if(n <= 0)
8         posisi = -1;
9     else
10    {
11         ketemu = 0;
12         i = 1;
13         while((i <= n-1) && (!ketemu))
14             if(data[i] == k)
15             {
16                 posisi = i;
17                 ketemu = 1;
18             }
19             else
20                 i++;
21         if(!ketemu)
22             posisi = -1;
23     }
24     return posisi;
25 }
```

```
26
27 int main ()
28 {
29     int data[5] = {12, 13, 19, 27, 28};
30     int dicari = 19;
31
32     printf("\tMetode Sequentian Search\n\n");
33     printf("Data: 12, 13, 19, 27, 28\n\n");
34     printf("Posisi %d berada pada index ke-: %d \n ", dicari, cari(data, 5, dicari));
35
36     return 0;
37
38 }
```

Tampilan :

```
Metode Sequentian Search
Data: 12, 13, 19, 27, 28
Posisi 19 berada pada index ke-: 2
```

2. Percobaan Program *Binary Search*: mencoba, membuat, menampilkan sebuah program *Binary Searching*.

Source Code :

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int poskar (char st[], char m)
5 {
6     int i, posisi, panjang;
7
8     i = 0;
9     posisi = -1;
10    panjang = strlen(st);
11    while((i < panjang-1) && posisi == -1)
12    {
13        if(st[i] == m)
14            posisi = i;
15            i++;
16    }
17    return posisi;
18 }
```

```

19
20 int main()
21 {
22     printf("\t Metode Binary Search\n\n");
23
24     char kalimat[] = "Teknik Informatika";
25     char dicari = 'm';
26
27     printf("\nPosisi %c dalam string %s berada pada index ke- [%d] ", dicari, kalimat, poskar(kalimat, dicari));
28
29     dicari = 'n';
30     printf("\nPosisi %c dalam string %s berada pada index ke- [%d]", dicari, kalimat, poskar(kalimat, dicari));
31
32     return 0;
33 }

```

Tampilan :

```

Metode Binary Search

Posisi m dalam string Teknik Informatika berada pada index ke- [12]
Posisi n dalam string Teknik Informatika berada pada index ke- [3]

```

3. Percobaan Program *Fibonacci Search*: mencoba, membuat, menampilkan sebuah program *Fibonacci Searching*.

Source Code :

```

binarysearch.cpp  [*] FibonacciSearch.cpp
1 #include <stdio.h>
2 #include <conio.h>
3 #include<iostream>
4 using namespace std;
5
6 int main()
7 {
8     int i, j, F0, F1, Fibo, n, m, N, Flag, x;
9     int FK, FK1, FK2, FK3, s, p, q, t;
10    int A[10] = {8, 15, 21, 28, 31, 37, 39, 46, 48, 50};
11    int FIBO[8];
12    cout<<"\tMetode Fibonacci Search\n\n";
13    cout<<"Data : ";
14    for(x=0; x<10; x++)
15    {
16        cout<<A[x]<<" ";
17        n = 9;
18        F0 = 1; F1 = 1; Fibo = 1;
19        j =1;
20    }
21
22    while (Fibo<= n+1)
23    {
24        FIBO[j] = Fibo;
25        Fibo = F0+F1; F0=F1; F1=Fibo;
26        j++;
27    }
28
29    s = j - 1;
30    FK = FIBO[s];
31    FK1 = FIBO[s-1]; i = FK1;
32    FK2 = FIBO[s-2]; p = FK2;
33    FK3 = FIBO[s-3]; q = FK3;
34

```

```

35     m = (n+1) - FK;
36     printf("\n\nMasukan data yang ingin dicari : ");
37     scanf("%d", &N);
38
39     if(N > A[i]) i = i+m;
40         Flag = 0;
41         while(i != 0 && Flag ==0)
42         {
43             if(N == A[i]) Flag = 1;
44
45             else if(N < A[i])
46             {
47                 if(q == 0) i = 0;
48                 else
49                 {
50                     i = i - q;
51                     t = p;
52                     p = q;
53                     q = t - q;
54                 }
55             }
56             else
57             {
58                 if(p == 1)
59                     i = 0;
60                 else
61                 {
62                     i = i + q;
63                     p = p - q;
64                     q = q - p;
65                 }
66             }
67         }
68         if(Flag == 1)
69             printf("\nData ditemukan");
70         else
71             printf("\nData tidak ditemukan");
72     }

```

Tampilan :

```

Metode Fibonacci Search
Data : 8 15 21 28 31 37 39 46 48 50
Masukan data yang ingin dicari : 40
Data tidak ditemukan

```

```

Metode Fibonacci Search
Data : 8 15 21 28 31 37 39 46 48 50
Masukan data yang ingin dicari : 50
Data ditemukan

```

4. Percobaan Program *Interpolation Search*: mencoba, membuat, menampilkan sebuah program *Interpolation Searching*.

Source Code :

```

1 #include <iostream>
2 #include <math.h>
3 using namespace std;

4
5 int main()
6 {
7     int data[10] = {9, 17, 24, 25, 36, 49, 52, 54, 59, 60};
8     int low,high,cari,posisi;
9     float posisi1;
10    int N = 10,tanda=0;
11    low=0,high=N-1;
12
13    cout<<"\tMetode Interpolation Search\n\n";
14    cout<<"Data : 9, 17, 24, 25, 36, 49, 52, 54, 59, 60\n";
15    cout<<"\nMasukan data yang dicari : ";
16    cin>>cari;
17    do
18    {
19        posisi1 = (cari-data[low])/((data[high]-data[low])*(high-low)+low);
20        posisi = floor(posisi1); //pembulatan ke bawah
21        if(data[posisi]==cari) {
22            tanda =1;
23            break;
24        }
25
26        if(data[posisi]>cari)
27            high=posisi-1;
28        else if(data[posisi]<cari)
29            low=posisi+1;
30    }
31
32    while (cari>=data[low]&&cari<=data[high]);
33    {
34        if(tanda==1)
35            cout<<"\nData ditemukan\n";
36        else
37            cout<<"\nData tidak ada\n";
38    }
39 }
```

Tampilan :

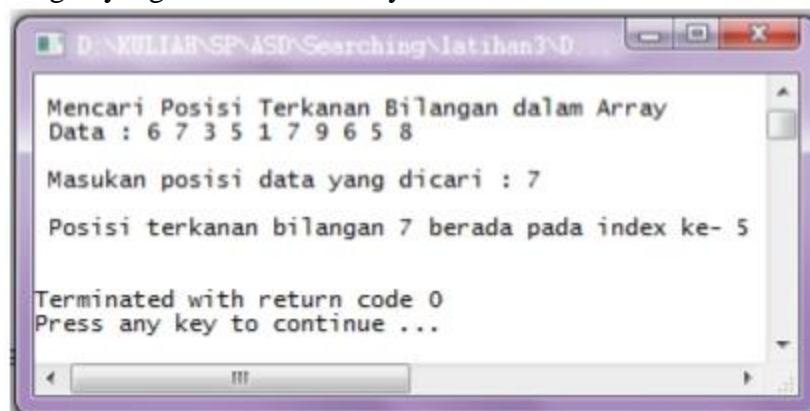
```

Metode Interpolation Search
Data : 9, 17, 24, 25, 36, 49, 52, 54, 59, 60
Masukan data yang dicari : 25
Data ditemukan
```

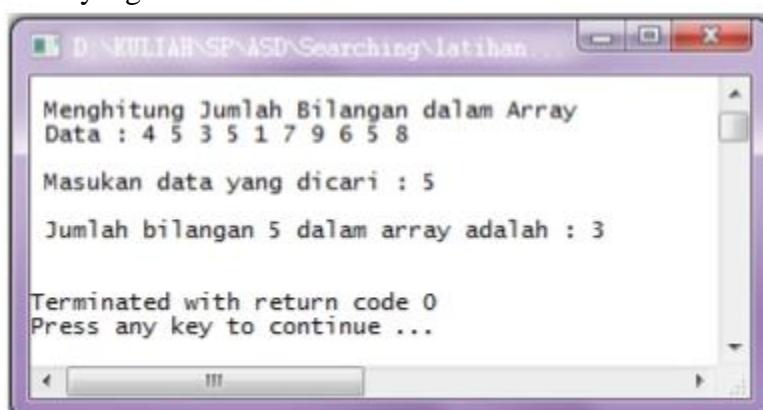
```
Metode Interpolation Search  
Data : 9, 17, 24, 25, 36, 49, 52, 54, 59, 60  
Masukan data yang dicari : 17  
Data ditemukan
```

F. TUGAS PRAKTIKUM

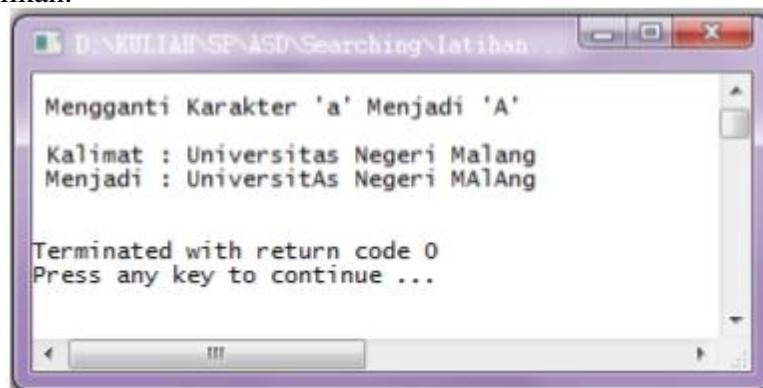
- Buatlah sebuah program untuk melakukan pencarian data berupa posisi terkanan dari suatu bilangan yang dicari dalam array satu dimensi.



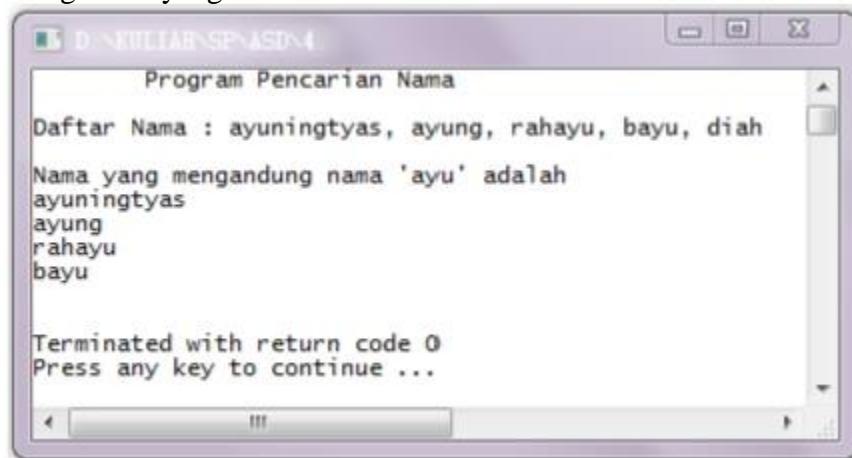
- Buatlah sebuah program untuk menghitung jumlah suatu bilangan dalam sebuah array satu dimensi yang berisi n buah elemen.



- Buatlah sebuah program yang dapat melakukan pencarian karakter dalam kalimat string, kemudian hasil pencarian karakter tersebut diubah menjadi karakter lain dan ditampilkan.



4. Buatlah sebuah program untuk melakukan pencarian nama dari beberapa daftar nama dalam array yang disediakan, kemudian tampilkan nama-nama yang didalamnya mengandung nama yang dicari



G. TUGAS RUMAH

Ada sebuah Supermarket memiliki sebuah gudang, Supermaket tersebut menginginkan untuk membuat sebuah aplikasi yang dapat mendata data-data barang yang ada di dalam gudang tersebut yang terdiri dari.

1. Kode Barang
2. Nama Barang
3. Harga
4. Stok (Jumlah Barang)

Buatlah aplikasi yang dapat memasukkan informasi barang-barang yang ada di gudang tersebut. Selain dapat menginputkan data, Aplikasi juga mempunya kemampuan untuk

1. Menambahkan/Mengurangi STOK barang yang sudah ada
2. Merubah Harga
3. Dan Menghapus Data

(Gunakan Prinsip Searching untuk melakukan ketiga perintah diatas)

MODUL IV

STACK

A. TUJUAN

1. Memahami terminologi yang terkait dengan struktur data stack.
2. Memahami operasi-operasi yang ada dalam stack.
3. Dapat mengidentifikasi permasalahan-permasalahan pemrograman yang harus diselesaikan dengan menggunakan stack, sekaligus menyelesaiakannya.

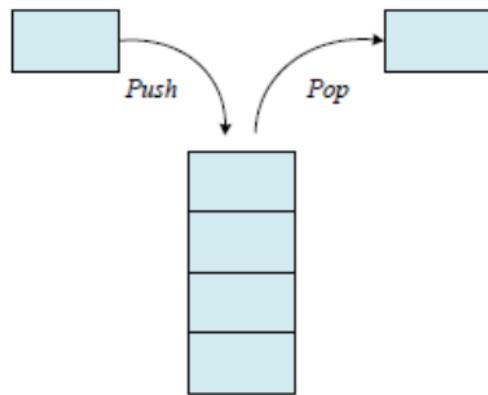
B. PETUNJUK

1. Awali setiap aktivitas anda dengan doa, agar anda lancar dalam belajar.
2. Pahami tujuan, dasar teori, dan latihan-latihan praktikum dengan baik.
3. Kerjakan tugas-tugas praktikum dengan baik, jujur, dan sabar.
4. Tanyakan kepada asisten apabila ada hal-hal yang kurang jelas.

C. DASAR TEORI

1. Pengertian Stack

Stack merupakan sebuah kumpulan data yang diletakkan di atas data lainnya, seperti sebuah tumpukan. Dengan demikian, stack merupakan salah satu struktur data yang menerapkan prinsip LIFO (Last In First Out). Dimana elemen yang terakhir disimpan dalam stack, menjadi elemen yang pertama diambil. Untuk meletakkan sebuah elemen pada bagian atas dari stack, maka dilakukan operasi push. Sedangkan untuk memindahkan sebuah elemen dari tempat atas tersebut dalam sebuah stack, maka dilakukan operasi pop.



Gambar 4.1 Ilustrasi sebuah stack

2. Operasi Dasar Pada Stack

- **Create**

Merupakan operator yang berfungsi untuk membuat sebuah stack kosong.

```
struct STACK {  
    int top;  
    float data[5];  
};  
float dta;  
struct STACK stackbaru;
```

- **IsEmpty**

Merupakan operator yang berfungsi untuk menentukan apakah suatu stack merupakan stack kosong. Tanda bahwa sebuah stack kosong adalah Top bernilai kurang dari nol (-1).

```
bool isempty() {  
    if (stackbaru.top==1) return true;  
    else return false;  
}
```

- **IsFull**

Merupakan operator yang digunakan untuk memeriksa apakah stack yang ada sudah penuh. Stack akan penuh jika puncak stack terletak tepat dibawah jumlah maksimum yang dapat ditampung stack (Top = MAX_STACK-1).

```
bool isfull() {  
    if (stackbaru.top==maxstack) return  
true;  
    else return false;  
}
```

- **Push**

Merupakan operator yang berfungsi untuk menambahkan satu elemen ke dalam stack dan tidak dapat dilakukan jika stack dalam keadaan penuh.

```
void push(float dta) {  
    if (isfull()==false) {  
        puts("stack penuh");  
    } else {  
        stackbaru.top++;  
        stackbaru.data[top]=dta;  
    }  
}
```

- **Pop**

Merupakan operator yang berfungsi untuk mengeluarkan satu elemen teratas dari dalam stack dengan syarat stack tidak dalam kondisi kosong.

```
void pop() {  
    if (isempty()==false) {  
        cout<<"data kosong";  
    } else {  
        cout<<"data yang terambil : "  
        <<stackbaru.data[top]<<endl;  
        stackbaru.top--;  
    }  
}
```

- **Clear**

Fungsi yang digunakan untuk mengosongkan stack dengan cara mengeset Top dengan -
1. Jika Top bernilai kurang dari nol maka stack dianggap kosong.

```
void clear () {  
    top=-1  
}
```

- **Retrieve**

fungsi yang digunakan untuk melihat nilai yang berada pada posisi tumpukan teratas.

```
void print() {  
    for (int i=0; i<=top; i++) {  
        cout<<stackbaru.data[i]<<"  
        ";  
    }  
}
```

3. Pointer Sebagai Penunjuk Stack

Selain menggunakan indeks, untuk menunjuk sebuah Top atau posisi teratas dari stack, dapat juga digunakan pointer sebagai berikut.

▪ Membuat Stack Dengan Pointer

```
int S[10], *Top, *BatasAtas  
Top = &S[-1];  
BatasAtas = &S[10];
```

▪ Proses Push

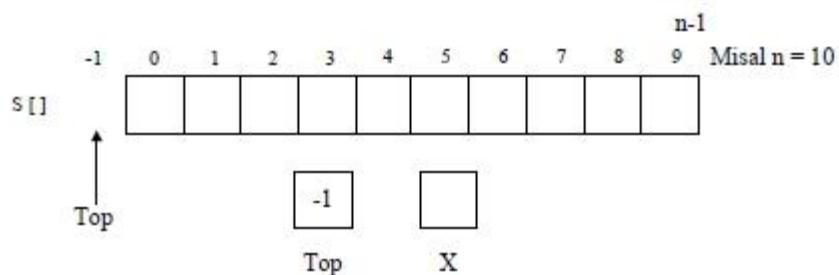
```
if (Top < BatasAtas)  
    Top++;  
    *Top = X;  
else  
    printf("Stack Penuh");
```

▪ Proses Pop

```
if (Top > &A[-1])  
    X= *Top;  
    Top --;  
else  
    printf("Stack Kosong");
```

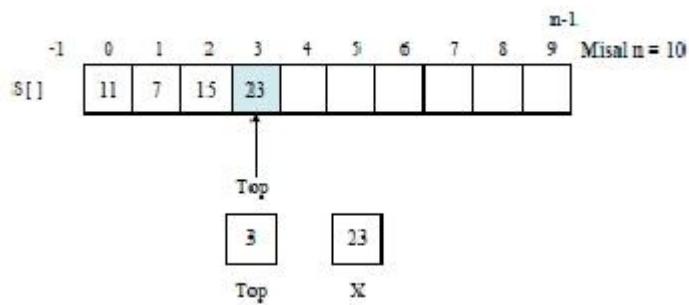
4. Representasi Proses Stack

Stack adalah salah satu dari contoh struktur data yang terdiri dari satu collection, yang juga menerapkan prinsip LIFO. Bila stack tersebut menggunakan array satu dimensi, maka stack tersebut dapat diilustrasikan sebagai berikut :



Gambar 4.2 Ilustrasi sebuah stack 1 Dimensi menggunakan indeks array

Pada gambar 4.2 diatas diilustrasikan ada sebuah indeks array yang masih kosong pada awal pembuatan stack dimana $n[10]$, variabel Top berada pada -1 yang menunjukkan indeks masih dalam keadaan kosong.



Gambar 4.3 Ilustrasi Stack ketika sudah diisi data

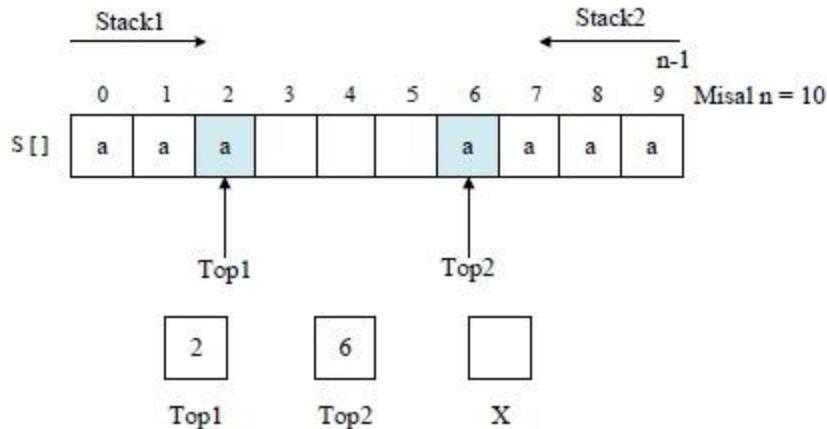
Pada gambar 4.3 adalah keadaan ketika stack sudah diisi data. Pada kondisi ini data pertama yang diinputkan adalah $S[0]=11$, data kedua $S[1]=7$, data ketiga $S[2]=15$, data keempat $S[3]=23$. Kondisi Top sudah berubah menjadi data yang terakhir diinputkan (data keempat) sehingga $\text{Top}[3] = X=23$.

Variabel Top digunakan sebagai indeks untuk menunjuk nomor elemen array yang berisi nilai stack yang berada paling kanan atau Top, yang ditunjukan dengan angka 3. Variabel X bertipe integer digunakan sebagai perantara, dimana data yang akan disimpan kedalam stack harus berasal dari X. Demikian juga data yang baru diambil dari dalam stack harus diterima terlebih dahulu oleh variabel X, kemudian baru diberikan ke variabel lain untuk diolah.

Oleh karena itu, jika ada instruksi PUSH, maka data baru (yang diambil dari isi variabel X) akan disimpan dalam elemen $S[4]$ sehingga indeks Top harus diarahkan ke posisi no.4. Artinya, Top maju terlebih dahulu satu langkah ke $S[4]$, kemudian baru mengisi nilai pada $S[4]$. Sedangkan jika ada instruksi Pop, maka yang akan diambil adalah isi dari $S[3]$ dan datanya akan disimpan terlebih dahulu dalam variabel X, kemudian indeks Top menjadi mundur satu langkah sehingga akan menunjuk $S[2]$.

5. Double Stack

Double Stack atau Stack Ganda adalah dua stack yang berada dalam satu array. Satu array digunakan untuk dua stack dimana dasar Stack1 berada pada sisi indeks yang terkecil dan dasar Stack2 berada pada sisi indeks yang terbesar. Sama halnya dengan Single Stack, Double Stack juga menerapkan prinsip LIFO (Last in First Out).



Gambar 4.4 Ilustrasi double stack

Pada gambar 4.4 diatas adalah ilustrasi indeks array pada double stack. Padastack 1 kondisi data pertama yang diinputkan adalah $S[0]$, data kedua $S[1]$, data ketiga $S[2]$ dan $\text{Top1}=\text{data input terakhir } S[2]$. Sedangkan pada stack 2 data pertama adalah $S[9]$, data kedua $S[8]$, data ketiga $S[7]$, data keempat $S[6]$ dan $\text{Top2}=\text{data input terakhir } S[6]$.

6. Operasi Dasar Pada Double Stack

▪ Inisialisasi

Proses awal adalah proses menyiapkan indeks penunjuk stack untuk pertama kali. Pada tahap ini ditetapkan $\text{Top1}=-1$ (sama seperti single stack) dan $\text{Top2}=\text{banyak jumlah data}$.

```
void AWAL (void)
{
    Top1 = -1;
    Top2 = n;
}
```

- **IsEmpty**

Sama dengan single stack, yaitu proses pengecekan stack dalam kondisi kosong

```
if(top1== -1) return true;      if(top2==n) return true;
```

- **Is Full**

Sama dengan single stack, yaitu proses pengecekan stack dalam kondisi kosong

```
int full(void){  
    if(top1+1>=top2){  
        return true;  
    }  
}
```

- **Push (Stack1 dan Stack2)**

Proses mengisi data pada stack1 maupun stack2

```
void PUSH1 (void)  
{  
    Top1 = Top1 + 1;  
    S[Top1] = X;  
}
```

```
void PUSH2 (void)  
{  
    Top2 = Top2 - 1;  
    S[Top2] = X;  
}
```

- **Pop (Stack1 dan Stack2)**

Proses mengambil data pada stack1 maupun stack2

```
void POP1 (void)  
{  
    X = S[Top1];  
    Top1 = Top1 - 1;  
}
```

```
void POP2 (void)  
{  
    X = S[Top2];  
    Top2 = Top2 + 1;  
}
```

D. LATIHAN

1. Program Single Stack

```
1 #include <iostream>
2 #include <stdio.h>
3 #include <conio.h>
4 #include <windows.h>
5 #define maxstack 4
6
7 using namespace std;
8 struct STACK { //Membuat jenis data abstrak 'STACK'
9     int top;
10    float data[4];
11 };
12    float dta;
13    struct STACK stackbaru;
14
15 void inisialisasi()
16 {
17     stackbaru.top=-1;
18 }
19 bool isfull()//mengecek apakah stack kondisi penuh?
20 {
21     if(stackbaru.top ==maxstack) return true;
22     else return false;
23 }
24 bool isempty()//mengecek apakah stack kondisi kosong?
25 {
26     if(stackbaru.top== -1) return true;
27     else return false;
28 }
29
30 void push(float dta)//mengisi stack(menyimpan data di stack)
31 {
32     if(isfull()==true)
33     {
34         puts("Maaf,stack penuh");
35         getch();
36     }
37     else
38     {
39         stackbaru.top++;
40         stackbaru.data[stackbaru.top]=dta;
41     }
42 }
43
44 void pop()//mengambil isi satck
45 {
46     if(isempty()==true)
47     {
48         cout<<"Data telah kosong!";
49         getch();
50     }
51     else
52     {
```

```
53     cout<<"data yang terambil adalah data ke : "<<stackbaru.data[stackbaru.top]<<endl;
54     stackbaru.top--;
55     getch();
56 }
57
58 void print()//men cetak stack
59 {
60     for(int i=0; i<=stackbaru.top; i++)
61     {cout<<stackbaru.data[i]<<"    ";}
62 }
63
64 void clear()//mengosongkan stack
65 {
66     stackbaru.top = -1;
67 }
68
69
70 int main()
71 {
72     inisialisasi();
73     char menu;
74     char uulang;
75     do{
76         system("cls");
77         printf("\t ----- STACK ----- \n");
78         printf("\t |          STACK          | \n");
79         printf("\t |-----| \n");
80         printf("\t \n\337 --> Menu STACK :\n\n");
81         puts("1. push stack");
82         puts("2. pop stack");
83         puts("3. cetak");
84         puts("4. bersihkan stack");
85         puts("5. exit");
86         cout<<"Menu pilihan anda : ";
87         cin>>menu;
88         if(menu =='2')
89         {
90
91
92
93         pop();
94         uulang ='y';
95     }
96     else if(menu =='1')
97     {
98         cout<<"data yang akan disimpan di stack: ";
99         cin>>dta;
100        push(dta);
101        uulang='y';
102    }
103
104    else if(menu =='3')
105    {
106        print();
107        cout<<"\nUlang?(y/t)";
108        cin>>uulang;
109    }
110    else if(menu== '4')
111    {
112        clear();
113        cout<<"\nUlang?(y/t)";
114        cin>>uulang;
115    }
116    else if(menu =='5')
117    {
118        exit(0);
119    }
120 }while(uulang =='Y' || uulang == 'y') ;
121
122
123 }
```

2. Program Double Stack

```
1  #include <stdio.h>
2  #include <conio.h>
3  #include <stdlib.h>
4  #define MAX 10
5  #define true 1
6  #define false 0
7  char stack[MAX];
8  int top1, top2;
9
10 void init(void);
11 void push(char data, int nomorstack);
12 char pop(int nomorstack);
13 void clear(int nomorstack);
14 int full(void);
15 int empty(int nomorstack);
16 void baca();
17
18 main(){
19     char data;
20     int pilih, nomorstack;
21     init();
22     do{
23         system("cls");
24         printf("Contoh program double stack");
25         printf("\n1. Push");
26         printf("\n2. Pop");
27         printf("\n3. Clear");
28         printf("\n4. Cetak Data");
29         printf("\n5. Selesai");
30         printf("\nPilihan anda : \n");
31         scanf("%i",&pilih);
32         switch(pilih){
33
34             case 1: printf("Push\n");
35             printf("Masukkan datanya :\n"); scanf("%s",&data);
36             printf("Mau dimasukkan ke stack berapa ? 1 atau 2 ? :\n");
37             scanf("%i",&nomorstack);
38             push(data, nomorstack);
39
40             break;
41
42             case 2: printf("Pop\n");
43             printf("Masukkan nomor stack\n");
44             scanf("%i",&nomorstack);
45             data=pop(nomorstack);
46             printf("\nData yang dikeluarkan adalah %s", data);
47             break;
48
49             case 3: printf("Clear\n");
50             printf("Nomor Stack yang akan dikosongkan :\n");
51             scanf("%s",&nomorstack);
52             clear(nomorstack);
53             break;
54
55             case 4: printf("Cetak Data :\n\n");
56             baca();
57             break;
58         }
59     }while(pilih!=5);
60 }
```

```
57     case 5: printf("Exit");
58     break;
59     default: printf("Pilihan yang anda masukkan tidak ada");
60     break;
61   }
62 }
63
64 }while(pilih!=5);
65 getch();
66 }
67
68 //init
69 void init(){
70   top1=0;
71   top2=MAX+1;
72 }
73
74 //PUSH
75 void push(char data, int nomorstack){
76   if(full()!=true){
77     switch(nomorstack){
78
79       case 1: top1++;
80       stack[top1]=data+1;
81       break;
82
83       case 2: top2--;
84       stack[top2]=data-1;
85       break;
86       default: printf("\nNomor stack salah");
87       break;
88     }
89   }
90   else
91     printf("\nStack penuh");
92   getch();
93 }
94
95 //POP
96 char pop(int nomorstack){
97   char data;
98   if(empty(nomorstack)!=true){
99     switch(nomorstack){
100
101       case 1: data=stack[top1];
102       top1--;
103       return data;
104       break;
105
106       case 2: data=stack[top2];
107       top2++;
108       return data;
109       break;
110       default: printf("\nNomor stack salah");
111       break;
112     }
113   }
114   else printf("\nStack masih kosong");
115   getch();
116 }
```

```
116    return 0;
117 }
118 //cek full
119
120 int full(void){
121     if ((top1==0) && (top2==MAX+1)){
122         return true;
123     }
124     else return false;
125 }
126
127 //cek empty
128 int empty(int nomorstack){
129     switch(nomorstack){
130     case 1: if (top1==0) return true;
131     else return false;
132     break;
133
134     case 2: if (top2==MAX+1) return true;
135     else return false;
136     break;
137     default: printf("nomor stack salah");
138     break;
139     }
140 }
141
142 //clearing
143 void clear(int nomorstack){
144     switch(nomorstack){
145     case 1: top1=1;
146     break;
147
148     case 2: top2=MAX;
149     break;
150     default: printf("Nomor stack salah");
151     break;
152     }
153 }
154
155 void baca(){
156     int i;
157     printf("Cetak isi stack pertama :\n");
158     for(i=1; i<top1; i++){
159         printf(" %c ",stack[i]);
160         printf("\n");
161     }
162     printf("Cetak Isi stack kedua :\n");
163     for(i=MAX; i>=top2; i--){
164         printf(" %c ",stack[i]);
165         printf("\n");
166     }
167
168     getch();
169 }
```

3. Lengkapi sintaks program berikut ini agar bisa berjalan menjadi program pembalik kata dengan stack

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<iostream>
4  #include<string.h>
5  #include<stdlib.h>
6  #define MAX 75
7  #define true 1
8  #define false 0
9  char stack[MAX];
10 int top;
11 void init(void);
12 int full(void);
13 int empty(void);
14 char pop(void);
15 void clear(void);
16 void push(char info);
17 main()
18 {
19     char pilih;
20     char kal[75];
21     int k,p;
22     printf(" --- PROGRAM MEMBALIK KATA DENGAN STACK ---\n\n");
23     init();
24     do
25     {
26         printf(" ======\n");
27         printf(" || MENU PILIHAN ||\n");
28         printf(" ======\n");
29         printf(" || [1] Masukan Kata ||\n");
30         printf(" || [2] Balik Kata ||\n");
31         printf(" || [3] Selesai ||\n");
32         printf(" ======\n\n");
33         printf(" Pilihan : ");
34         scanf("%s", &pilih);
35         system("cls");
36         switch(pilih)
37         {
38             case '1': printf("\n Masukkan kata : ");
39                         ..... ;
40                         scanf("%s", &kal);
41                         p=strlen(kal);
42                         for(.....)
43                             push(kal[k]);
44                         printf("\n Kata yang masuk : ");
45                         for(.....)
46                             printf("%c ",stack[k]);
47                         printf("\n");
48                         break;
49
50             case '2': if(empty() !=true)
51                         {
52                             printf("\n Kata setelah dibalik : ");
53                             .....
54                             .....
55                         }
56             else printf("\n Stack kosong !!\n");
57             break;
58         }
59     }
60 }
```

```
58
59         case '3': break;
60     default : printf("\n Pilihan anda salah !!\n");
61 }
62 printf("\n");
63 while(pilih != '3');
64 }
65 void init(void)
66 {
67     top=0;
68 }
69 void push(char info )
70 {
71     if(full() !=true)
72     {
73         top++;
74         stack[top]=info;
75     }
76     else
77         printf("\n Stack overflow... \n");
78     }
79 char pop(void)
80 {
81     char info;
82     if(empty() !=true)
83     {
84         info=stack[top];
85         top--;
86         return(info);
87     }
88     else
89         printf("\n Stack underflow... \n");
90     }
91 int full(void)
92 {
93     if(top==MAX)
94         return(true);
95     else
96         return(false);
97 }
98 int empty(void)
99 {
100     if(top==0)
101         return(true);
102     else
103         return(false);
104 }
```

E. TUGAS RUMAH

1. Buatlah sebuah program dengan menggunakan konsep stack yang berfungsi untuk menyimpan data nilai mahasiswa yang terdiri dari id, nama, dan nilai mahasiswa. Tambahkan beberapa fasilitas seperti **sorting**(pengurutan data), **multi push** (menambahkan data lebih dari satu), dan **multi pop** (mengambil data lebih dari satu sekaligus).

MODUL V

QUEUE

A. Tujuan Pembelajaran

Mahasiswa mampu menjelaskan pengertian queue dan dequeue

- Mahasiswa mampu menjelaskan dan menunjukkan cara pembuatan queue, operasi push dan pop pada array
- Mahasiswa mampu menjelaskan dan menunjukkan program dengan ADT (Abstract Data Type) queue dan dequeue dengan array

B. Dasar Teori

Queue atau antrian adalah suatu kumpulan data yang penambahan elemennya hanya bisa dilakukan pada suatu ujung (disebut dengan sisi belakang atau rear), dan penghapusan atau pengambilan elemen dilakukan lewat ujung yang lain (disebut dengan sisi depan atau front).

Kalau tumpukan dikenal dengan menggunakan prinsip LIFO (Last In First Out), maka pada antrian prinsip yang digunakan adalah FIFO (First In First Out).

Implementasi Antrian dengan Array

Untuk memahami penggunaan antrian dalam array, kita membutuhkan deklarasi antrian, misalnya:

```
#define MAXN 6
typedef enum {NOT_OK, OK} Tboolean;
typedef struct {
    Titem array[MAXN];
    int first;
    int last;
    int number_of_items;
} Tqueue;
```

Dengan deklarasi di atas, elemen antrian dinyatakan dalam tipe integer yang semuanya terdapat dalam struktur. Variabel first menunjukkan posisi elemen pertama dalam *array*, dan variable last menunjukkan posisi elemen terakhir dalam *array*.

Algoritma dari penggalan program di atas adalah:

1. Tentukan elemen yang akan dimasukkan ke dalam antrian (dalam hal ini adalah 6 elemen)
2. Deklarasikan struktur untuk menampung elemen pada antrian
3. Selesai

Untuk menambah elemen baru dan mengambil elemen dari antrian dalam antrian, diperlukan deklarasi berikut ini:


```

void initialize_queue (Tqueue *Pqueue) {
    Pqueue->first = 0;
    Pqueue->last= -1;
    Pqueue->number_of_items = 0;
}
Tboolean enqueue (Tqueue *Pqueue, Titem item) {
    if (Pqueue->number_of_items>=MAXN)
        return(NOT_OK);
    else {
        Pqueue->last++;
        if (Pqueue->last>MAXN-1)
            Pqueue->last=0;
        Pqueue->array[Pqueue->last]=item;
        Pqueue->number_of_items++;
        return (OK);
    }
}

Tboolean dequeue (Tqueue *Pqueue, Titem *Pitem){
    if (Pqueue->number_of_items == 0)
        return (NOT_OK);
    else {
        *Pitem = Pqueue->array[Pqueue->first++];
        if (Pqueue->first>MAXN-1)
            Pqueue->first=0;
        Pqueue->number_of_items--;
        return(OK);
    }
}

```

Implementasi Antrian dengan Pointer

Untuk mengimplementasikan antrian dengan menggunakan pointer, perhatikan algoritma berikut ini:

1. Tentukan struktur untuk menampung node yang akan dimasukkan pada antrian. Deklarasi struktur pada penggalan program berikut ini:

```

struct queueNode {
    char data;
    struct queueNode *nextPtr;
};
typedef struct queueNode QUEUENODE;
typedef QUEUENODE *QUEUENODEPTR;
QUEUENODEPTR headPtr=NULL, tailPtr=NULL;

```

2. Deklarasikan penambahan elemen baru pada antrian, di mana letaknya adalah paling belakang. Deklarasi penambahan elemen baru tersebut dapat dilihat pada penggalan program berikut ini:

```

void enqueue (QUEUENODEPTR *headPtr, QUEUENODEPTR *tailPtr,
             char value) {
    QUEUENODEPTR newPtr=malloc (sizeof(QUEUENODE));
    if(newPtr!=NULL){
        newPtr->data=value;
        newPtr->nextPtr=NULL;
        if(isEmpty(*headPtr))
            *headPtr=newPtr;
        else
            (*tailPtr)->nextPtr=newPtr;
        *tailPtr=newPtr;
    }
}

```

3. Lakukan pengecekan terhadap antrian, apakah antrian dalam kosong atau tidak. Kalau kondisi antrian kosong, maka elemen bisa dihapus. Penggalan program berikut ini akan menunjukkan kondisi tersebut.

```
int isEmpty(QUEUENODEPTR headPtr) {
    return headPtr==NULL;
}
```

C. Latihan

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define max 10

using namespace std;

int antrian[1000000]; //array untuk menampung data antrian
int head = 0 , tail = 0 , cur = 0;
//head -> data antrian paling depan
//tail -> data antrian paling belakang
//cur -> jumlah data pada antrian

bool full();
bool empty();
int push(int data);
int pop();
int print();

int main(){
    int pilihan,data;
    do{
        system("cls");
        printf("\n-----MENU-----\n");
        printf("1. PUSH Data Ke Antrian \n");
        printf("2. POP Data Dari Antrian \n");
        printf("3. Print Data Di Antrian \n");
        printf("4. Exit \n\n");
        printf("Tentukan Pilihanmu : ");
        scanf("%d",&pilihan);
        switch(pilihan){
            case 1 :
                if(!full()){
                    printf("---PUSH DATA---\n");
                    printf("Masukkan Data : ");
                    scanf("%d",&data);
                    push(data);
                }
                else{
                    printf("Antrian Penuh !!!\n");
                    getch();
                }
            break;
        }
    }while(pilihan!=4);
}
```

```
case 2:
    if(!empty()){ //sama kayak if(empty() == false/0) kalo -> if(empty()) sama kayak if(empty() == true/1)
        printf("---POP DATA---\n");
        printf("Data yang dihapus : %d\n", pop()); //fungsi pop() return data yang di hapus, kemudian di print
        getch();
    }
    else{
        printf("ANTRIAN KOSONG !!!\n");
        getch();
    }
    break;
case 3:
    printf("DATA ANTRIAN : \n");
    print();
    break;
case 4:
    printf("Keluar. . .\n");
    break;
default :
    printf("PILIHAN TIDAK ADA !!!\n");
    getch();
    break;
}
}while(pilihan != 4);
getch();
}

bool full(){
    return (cur >= max) ? true : false; //cek apakah cur >= max ?, kalo benar return true kalo salah return false
}

bool empty(){
    return (cur == 0) ? true : false;
}

int push (int data){
    antrian[tail] = data;
    tail++;
    cur++;
}

int pop(){
    int data = antrian[head];
    head++;
    cur--;
    return data;
    getch();
}

int print(){
    for (int i= head ; i< tail ; i++){
        printf("%d ",antrian[i] );
    }
    getch();
}
```

D. Tugas Praktikum



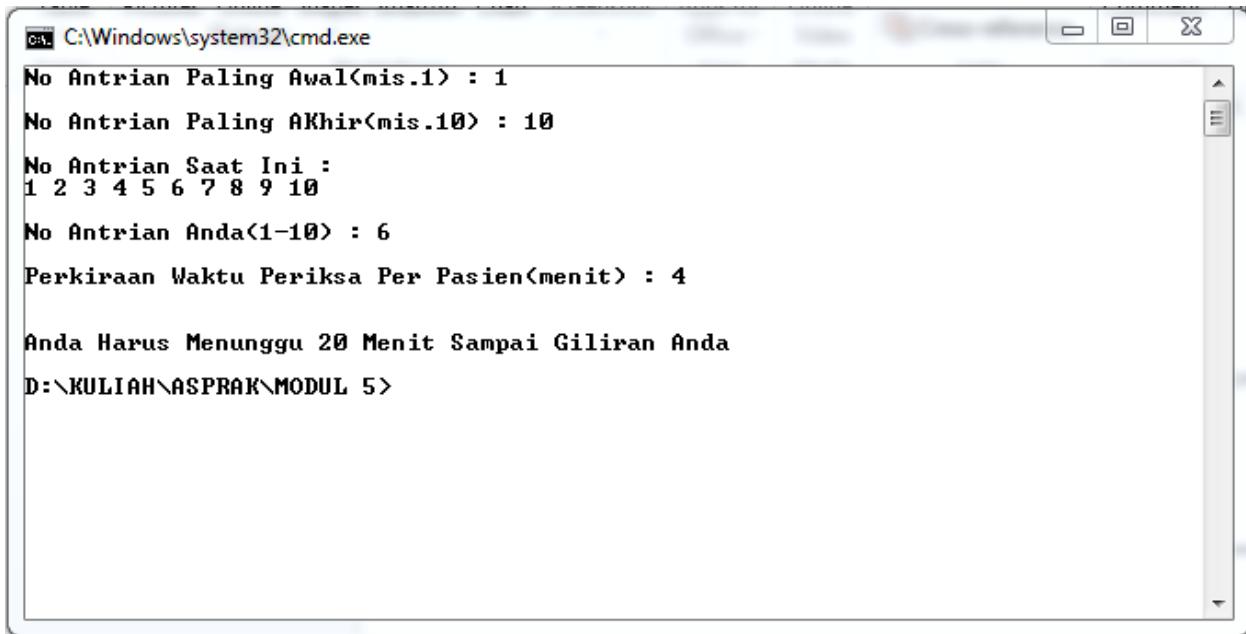
Sebuah bank membutuhkan program untuk melakukan antrian data , buatlah program tersebut dengan metode queue.

Syarat:

- Menggunakan array atau linked list
- Ada 2 menu berbeda untuk teler dan nasabah

E. Tugas

Buatlah sebuah program yang dapat menghitung waktu tunggu pasien pada saat mengantri untuk berobat. Gunakan algoritma queue
Minimal program dapat melakukan hal berikut :



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text output:

```
No Antrian Paling Awal<mis.1> : 1
No Antrian Paling Akhir<mis.10> : 10
No Antrian Saat Ini :
1 2 3 4 5 6 7 8 9 10
No Antrian Anda<1-10> : 6
Perkiraan Waktu Periksa Per Pasien<menit> : 4

Anda Harus Menunggu 20 Menit Sampai Giliran Anda
D:\KULIAH\ASPRAK\MODUL 5>
```

MODUL 6

SINGLE & DOUBLE LINKED LIST

1. Tujuan Instruksional Umum

- a. Mahasiswa dapat melakukan perancangan aplikasi menggunakan struktur Linked List (Senarai Berkait)
- b. Mahasiswa mampu melakukan analisis pada algoritma *Single & Double Linked List* yang dibuat
- c. Mahasiswa mampu mengimplementasikan algoritma *Single & Double Linked List* pada sebuah aplikasi secara tepat dan efisien

2. Tujuan Instruksional Khusus

- a. Mahasiswa dapat menjelaskan mengenai *Single & Double Linked List*
- b. Mahasiswa dapat membuat dan mendeklarasikan Abstraksi Tipe Data *Single & Double Linked List*
- c. Mahasiswa mampu menerapkan operasi *Single Linked List Non Circular & Single Linked List Circular*
- d. Mahasiswa mampu menerapkan *Double Linked List Non Circular & Double Linked List Circular*

Pengertian Linked List

Salah satu bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis dan terbatas adalah senarai berkait (*linked list*). Suatu senarai berkait (*linked list*) adalah suatu simpul (*node*) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. Suatu simpul dapat berbentuk suatu struktur atau *class*. Simpul harus mempunyai satu atau lebih elemen struktur atau *class* yang berisi data. Secara teori, *linked list* adalah sejumlah node yang dihubungkan secara linier dengan bantuan *pointer*. Senarai berkait lebih efisien di dalam melaksanakan penyisipan-penyisipan dan penghapusan-penghapusan. Senarai berkait juga menggunakan alokasi penyimpanan secara dinamis, yang merupakan penyimpanan yang dialokasikan pada *runtime*. Karena di dalam banyak aplikasi, ukuran dari data itu tidak diketahui pada saat kompile, hal ini bisa merupakan suatu atribut yang baik juga. Setiap *node* akan berbentuk *struct* dan memiliki satu buah *field* bertipe *struct* yang sama, yang berfungsi sebagai *pointer*. Dalam menghubungkan setiap node, kita dapat menggunakan cara *first-create-first-access* ataupun *first-create-last-access*. Yang berbeda dengan deklarasi *struct* sebelumnya adalah satu *field* bernama *next*, yang bertipe *struct tnode*. Hal ini sekilas dapat membingungkan. Namun, satu hal yang jelas, variabel *next* ini akan menghubungkan kita dengan *node* di sebelah kita, yang juga bertipe *struct tnode*. Hal inilah yang menyebabkan *next* harus bertipe *struct tnode*.

Bentuk Umum :

```
typedef struct telnmtlist
{
    infotype info;
    address next;
} telnmtlist;
```

infotype → sebuah tipe terdefinisi yang menyimpan informasi sebuah elemen list
next → address dari elemen berikutnya (suksesor) .

Jika L adalah list, dan P adalah address, maka alamat elemen pertama list L dapat diacu dengan notasi : `first (L)`

Sebelum digunakan harus dideklarasikan terlebih dahulu :

```
#define First(L) (L)
```

Elemen yang diacu oleh P dapat dikonsultasi informasinya dengan notasi :

```
info(P) deklarasi #define info(P) P->info  
next(P) deklarasi #define next(P) P->next
```

Beberapa Definisi :

1. List l adalah list kosong, jika `First(L) = Nil`
2. Elemen terakhir dikenali, dengan salah satu cara adalah karena `Next(Last) = Nil`
Nil adalah pengganti Null, perubahan ini dituliskan dengan `#define Nil Null`

Operasi-operasi Linked List

➤ **Insert**

Istilah Insert berarti menambahkan sebuah simpul baru ke dalam suatu linked list.

➤ **IsEmpty**

Fungsi ini menentukan apakah linked list kosong atau tidak.

➤ **Find First**

Fungsi ini mencari elemen pertama dari linked list.

➤ **Find Next**

Fungsi ini mencari elemen sesudah elemen yang ditunjuk now.

➤ **Retrieve**

Fungsi ini mengambil elemen yang ditunjuk oleh now. Elemen tersebut lalu dikembalikan oleh fungsi.

➤ **Update**

Fungsi ini mengubah elemen yang ditunjuk oleh now dengan isi dari sesuatu.

➤ **Delete Now**

Fungsi ini menghapus elemen yang ditunjuk oleh now. Jika yang dihapus adalah elemen pertama dari linked list (head), head akan berpindah ke elemen berikutnya.

➤ **Delete Head**

Fungsi ini menghapus elemen yang ditunjuk head. Head berpindah ke elemen sesudahnya.

➤ **Clear**

Fungsi ini menghapus linked list yang sudah ada. Fungsi ini wajib dilakukan bila anda ingin mengakhiri program yang menggunakan linked list. Jika anda melakukannya, data-data yang dialokasikan ke memori pada program sebelumnya akan tetap tertinggal di dalam memori.

a. **Single Linked List (Senarai berkait tunggal)**

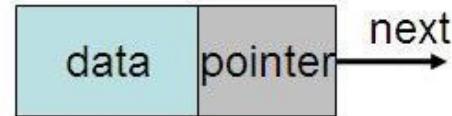
Single linked list adalah apabila hanya ada satu pointer yang menghubungkan setiap node (satu arah “next”).

a.1. Single Linked List non Circular

Pembuatan struct bernama tnode berisi 2 field, yaitu field data bertipe integer dan field next yang bertipe pointer dari tnode.

Deklarasi node dengan struct:

```
struct tnode
{
    int data;
    struct tnode *next;
}
```

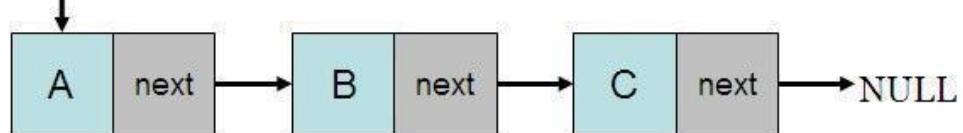


Gambar 1. Sebuah node pada Single Linked List

Asumsikan kita memiliki sejumlah *node* yang selalu menoleh ke sebelah dalam arah yang sama. Atau, sebagai alat bantu, kita bisa mengasumsikan beberapa orang yang bermain kereta api. Yang belakang akan memegang yang depan, dan semuanya menghadap arah yang sama. Setiap pemain adalah *node*. Dan tangan pemain yang digunakan untuk memegang bahu pemain depan adalah variabel *next*. Sampai di sini, kita baru saja mendeklarasikan tipe data dasar sebuah *node*. Selanjutnya, kita akan mendeklarasikan beberapa variabel *pointer* bertipe *struct tnode*. Beberapa variabel tersebut akan kita gunakan sebagai awal dari *linked list*, *node* aktif dalam *linked list*, dan *node* sementara yang akan digunakan dalam pembuatan *node* di *linked list*. Berikan nilai awal *NULL* kepada mereka. Deklarasi node untuk beberapa keperluan, seperti berikut ini:

```
struct tnode *head=NULL, *curr=NULL, *node=NULL;
```

Dengan demikian, sampai saat ini, telah dimiliki tiga *node*. Satu sebagai kepala (*head*), satu sebagai *node* aktif dalam *linked list* (*curr*) dan satu lagi *node* sementara (*node*). Untuk mempermudah pengingatan, ingatlah gambar anak panah yang mengarah ke kanan. Head akan berada pada pangkal anak panah, dan node-node berikutnya akan berbaris ke arah bagian anak panah yang tajam.



Gambar 2. Single Linked List

Apabila diperhatikan, setiap node memiliki petunjuk untuk node sebelahnya. Node terakhir akan diberikan nilai *NULL*. Dengan demikian, setiap node kebagian jatah.

```
int i;
for (i=0; i<5; i++)
{
```

```

node = (struct tnode *)
malloc (sizeof(struct tnode));
node -> data = i;
if (head == NULL)
{
head = node;
curr = node;
}
else
{
curr -> next = node;
curr = node;
}
curr -> next = NULL;
    
```

Berikut adalah penjelasan kode-kode pembuatan singly linked list tersebut. Pertama-tama, akan dibuat perulangan dari 0 sampai 4, yang dimaksudkan untuk membuat lima buah node yang masing-masing field data nya berisikan nilai dari 0 sampai 4. Pembuatan node dilakukan dengan fungsi malloc().

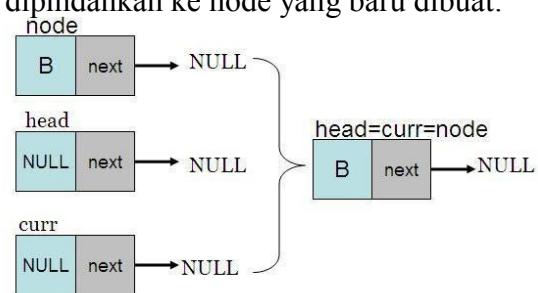
```

for (i=0; i<5; i++)
{
node = (struct tnode *)
malloc (sizeof(struct tnode));
node -> data = i;
...
... }
    
```

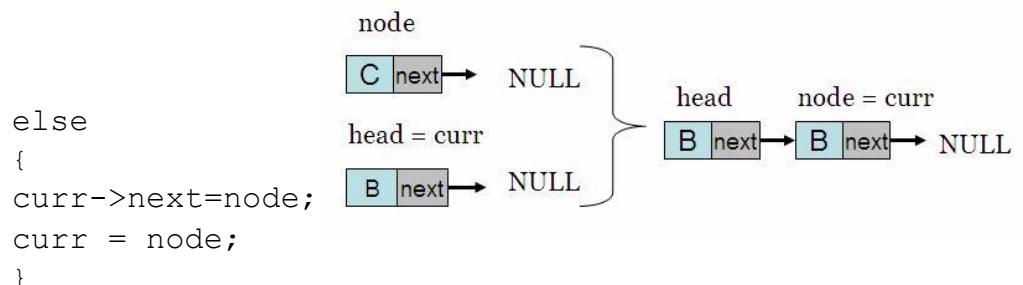
Setelah itu, perlu dibuat node dan penghubung. Pertama-tama, akan diuji apakah head bernilai NULL. Kondisi head bernilai NULL hanya terjadi apabila belum dimiliki satu node pun. Dengan demikian, node tersebut akan dijadikan sebagai head. Node aktif (curr), juga kita dapat dari node tersebut. Kalau head tidak bernilai NULL alias telah dimiliki satu atau lebih node, yang pertama dilakukan adalah menghubungkan pointer next dari node aktif (curr) ke node yang baru saja dibuat. Dengan demikian, baru saja dibuat penghubung antara rantai lama dengan mata rantai baru. Atau, dalam permainan kereta api, pemain paling depan dalam barisan lama akan menempelkan tangannya pada bahu pemain yang baru bergabung. Node aktif (curr) kemudian dipindahkan ke node yang baru dibuat.

```

if (head == NULL)
{
head = node;
curr = node;
}
    
```



Gambar 3. Membuat elemen pertama SLL



Gambar 4. Penambahan elemen dibelakang SLL

a.2. Single Linked List Circular

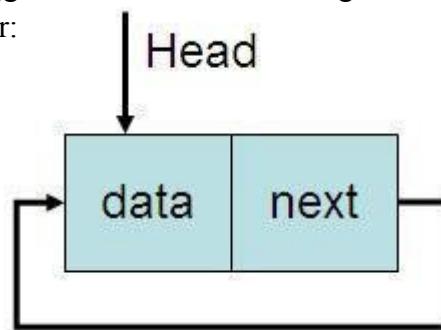
Hampir sama dengan single linked list non circular, bahwa dibutuhkan sebuah kait untuk menghubungkan node-node data yang ada, dimana pada node terakhir atau tail yang semula menunjukkan NULL diganti dengan menunjuk ke kepala atau head. Dimana inisialisasi senarai berkait tunggal sirkular menggunakan struc adalah sebagai berikut:

Deklarasi Single Linked List Circular:

```

Struct tnode
{
    int data;
    tnode *next;
};

void main()
{
    head = new tnode;
    head->next = head;
}
    
```



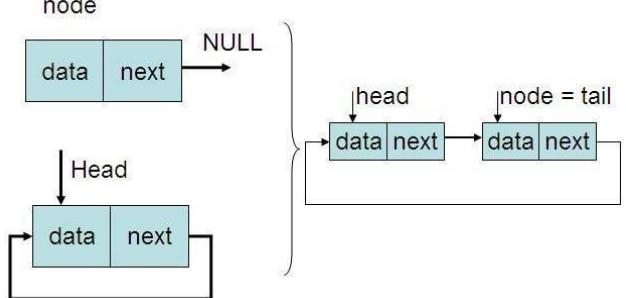
Gambar 5. Single Linked List circular

Menambah node dan membuat tail dari single linked list circular

Deklarasi penambahan node baru:

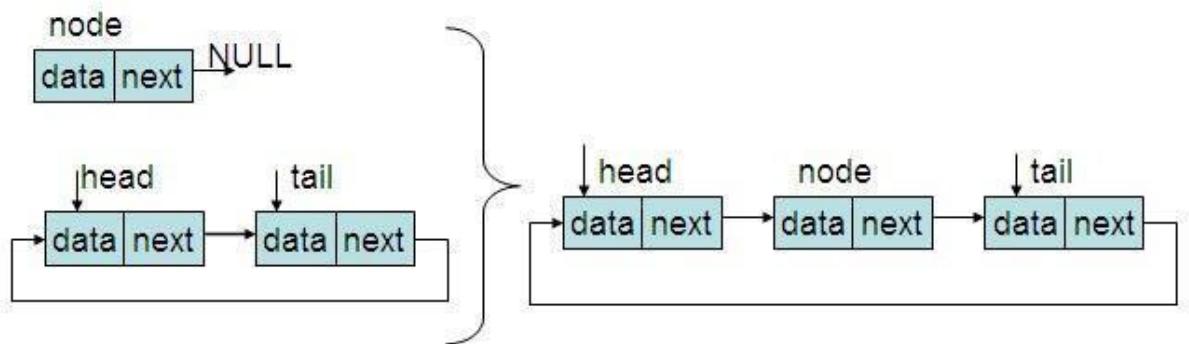
```

Void main()
{
    node = new tnode;
    tail = new tnode;
    node->next = head->next;
    head->next = node;
    tail = node;
}
    
```



Gambar 6. Penambahan Node baru

Menyisipkan Node baru



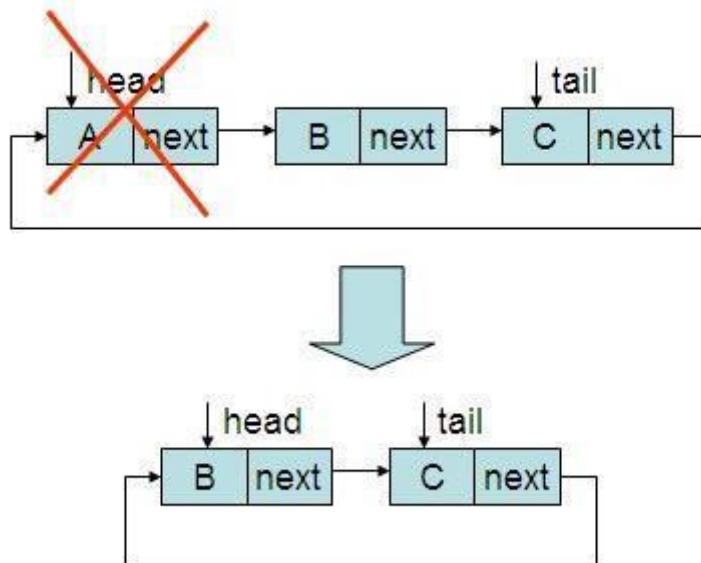
Gambar 7. Menyisipkan Node Baru

Deklarasi menyisipkan node baru menggunakan sintak berikut:

```
Void main()
{
    node = new tnode;

    node->next = head->next;
    head->next = node;
}
```

Menghapus Node dari Single Linked List Circular



Gambar 8. Menghapus node dari SLLC

Deklarasi menghapus node dari single linked list circular, menggunakan sintaks berikut:

```
Void main()
{
    hapus = new tnode;
    if( head != tail)
    {
        hapus = head;
```

```

head = head->next;
tail->next = head;
delete hapus;
} else
{
head = NULL;
tail = NULL;
}
}

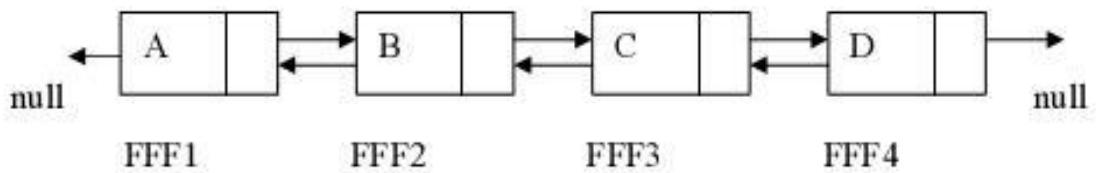
```

b. Double Linked List (Senarai berkait ganda)

Double Linked List adalah elemen-elemen yang dihubungkan dengan dua pointer dalam satu elemen dan list dapat melintas baik di depan atau belakang. Elemen double linked list terdiri dari tiga bagian :

1. Bagian data informasi
2. Pointer next yang menunjuk ke elemen berikutnya
3. Pointer prev yang menunjuk ke elemen sebelumnya

b.1. Double Linked List non Circular



Gambar 9. Ilustrasi Double Link List Non Circular

Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya dan ke node sebelumnya. Untuk pembentukan node baru, mulanya pointer next dan prev akan menunjuk ke nilai NULL. Selanjutnya, pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node selanjutnya pada list.

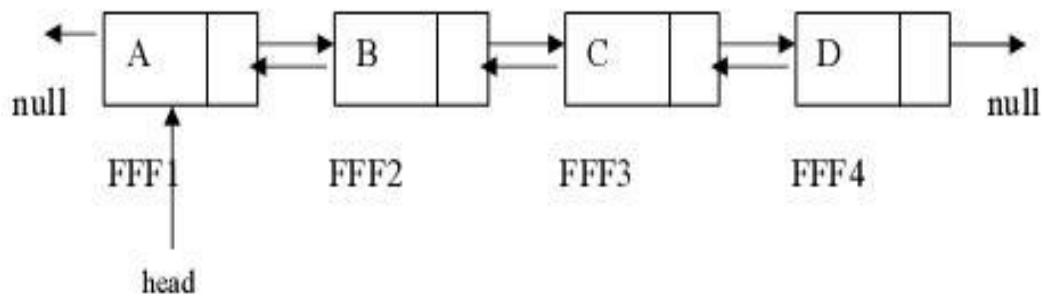
Deklarasi node dibuat dari struct berikut ini :

```

typedef struct TNode{
    int data;
    Tnode *next;
    Tnode *prev;
};

```

Double Linked List Non Circular dengan HEAD membutuhkan satu buah variabel pointer, yaitu: head. Head akan selalu menunjuk pada node pertama.



Gambar 10. Double Linked List Non Circular dengan HEAD

Deklarasi Pointer Penunjuk Kepala Double 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 Linked List non Circular
void init(){
    head = NULL;
}
```



Function untuk mengetahui kosong tidaknya DLLNC

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

Penambahan data di depan

Penambahan node baru akan dikaitkan di node paling depan, namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan pada head-nya. Pada prinsipnya adalah mengaitkan 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.

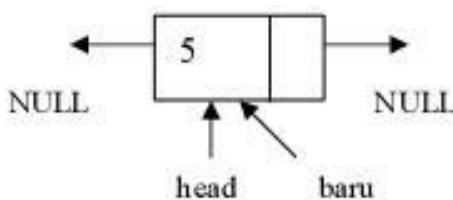
Fungsi Menambah Di Depan

```
void insertDepan(int databaru){
    TNode *baru;
    baru = new TNode;
    baru->data = databaru;
    baru->next = NULL;
```

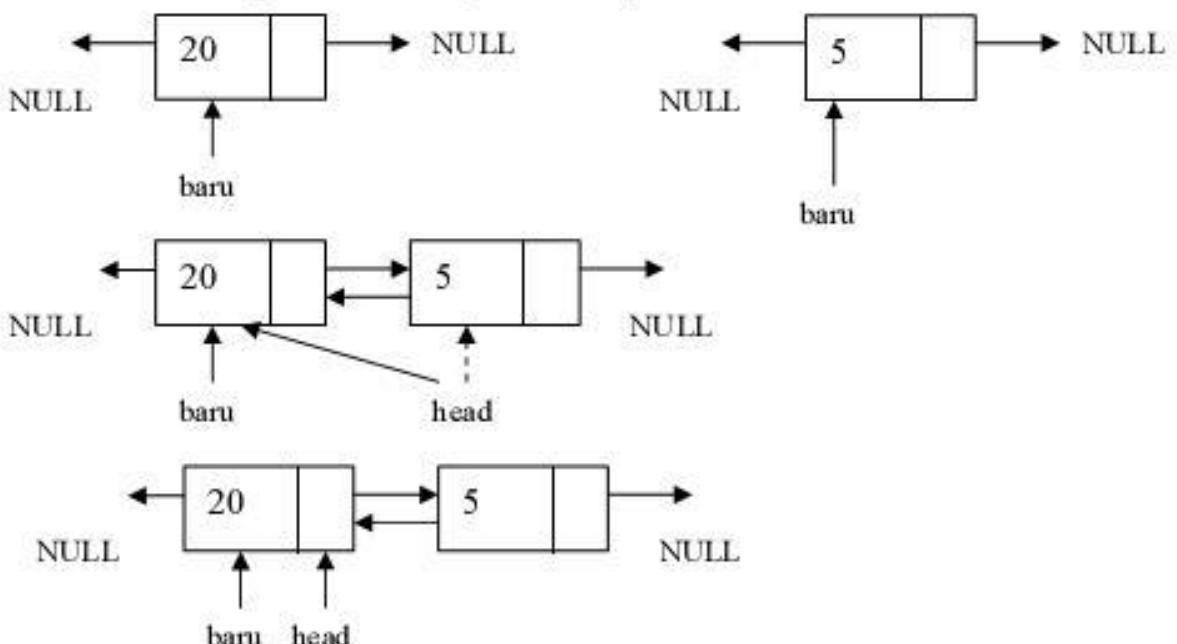
```

baru->prev = NULL;
if(isEmpty() == 1){
head=baru;
head->next = NULL;
head->prev = NULL;
}
else {
baru->next = head;
head->prev = baru;
head = baru;
}
cout<<"Data masuk\n";
}

```



3. Datang data baru, misalnya 20



Gambar11. Ilustrasi Penambahan Node Di Depan

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.

Fungsi penambahan Node ke belakang :

```

Void insertBelakang (int databaru){
    Tnode *baru, *bantu;
    Baru = new Tnode;
    baru->data = databaru;
    baru->next = NULL;
    baru->prev = NULL;
    if(isEmpty() == 1){
        head=baru;
        head->next = NULL;
        head->prev = NULL;
    }
    else{
        bantu=head;
        while(bantu->next!=NULL){
            bantu=bantu->next
        }
        bantu->next = baru;
        baru->prev = bantu;
    }
    Cout<<"Data masuk\n";
}

```

Ilustrasi Penambahan Node di Belakang

Ilustrasi:

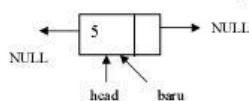
1. List masih kosong (`head=NULL`)

NULL

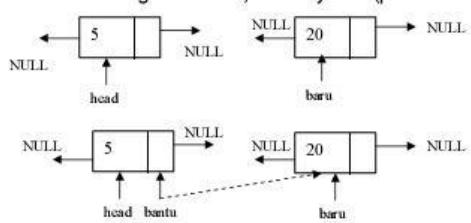
↑

head

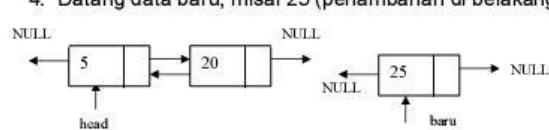
2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20 (penambahan di belakang)



4. Datang data baru, misal 25 (penambahan di belakang)



Function untuk menampilkan isi DLLNC

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

Function untuk menghapus data di depan:

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

Fungsi untuk menghapus node terbelakang

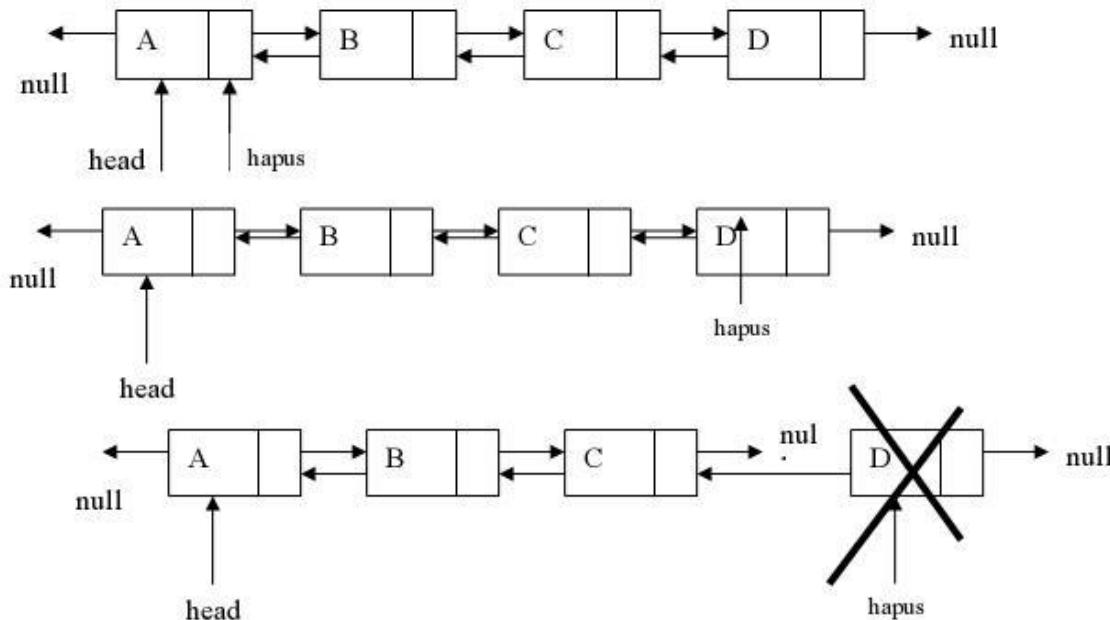
```
void hapusBelakang(){
    TNode *hapus;
    int d;
    if (isEmpty() == 0){
        if(head->next != NULL){
            hapus = head;
            while(hapus->next != NULL){
                hapus = hapus->next;
            }
            d = hapus->data;
            hapus->prev->next = NULL;
```

```

        delete hapus;
    } else {
        d = head->data;
        head = NULL;
        cout<<d<<" terhapus\n";
    } else cout<<"Masih kosong\n";
}

```

Ilustrasi Penghapusan Node



Menghapus Node Double Linked List

Tidak diperlukan pointer bantu yang mengikuti pointer hapus yang berguna untuk menunjuk ke NULL. Karena pointer hapus sudah bisa menunjuk ke pointer sebelumnya dengan menggunakan elemen prev ke node sebelumnya, yang akan diset agar menunjuk ke NULL setelah penghapusan dilakukan.

Fungsi menghapus semua elemen

```

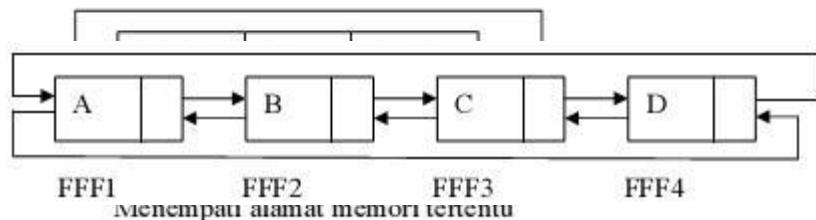
void clear(){
    TNode *bantu,*hapus;
    bantu = head;
    while(bantu!=NULL){
        hapus = bantu;
        bantu = bantu->next;
        delete hapus;
    }
    head = NULL;
}

```

b.2. Double Linked List Circular

Double Linked List Circular (DLLC) adalah linked list dengan menggunakan pointer, dimana setiap node memiliki 3 field, yaitu 1 field

pointer yang menunjuk pointer berikutnya (next), 1 field menunjuk pointer sebelumnya (prev), serta sebuah field yang berisi data untuk node tersebut dengan pointer next dan pre-nya menunjuk ke dirinya sendiri secara circular.



Ilustrasi DLLC

Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya dan ke node sebelumnya. Untuk pembentukan node baru, mulanya pointer next dan prev akan menunjuk ke dirinya sendiri. Jika sudah lebih dari satu node, maka pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node sesudahnya.

Deklarasi dan node baru DLLC

Deklarasi node

Dibuat dari struct berikut ini:

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

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;  
baru->prev = baru;
```

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 mengaitkan 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;  
    baru->prev = baru;  
    if(isEmpty() == 1){  
        head=baru;  
        head->next = head;  
        head->prev = head;  
    }  
    else {  
        bantu = head->prev;  
        baru->next = head;  
        head->prev = baru;  
        head = baru;  
        head->prev = bantu;  
        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.

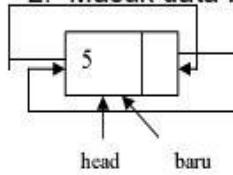
```
void insertBelakang (int databaru){  
    Tnode *baru, *bantu;  
    baru = new Tnode;  
    baru->data = databaru;  
    baru->next = baru;  
    baru->prev = baru;  
    if(isEmpty() == 1){  
        head=baru;  
        head->next = head;  
        head->prev = head;  
    }  
    else{  
        bantu=head->prev;  
        bantu->next = baru;  
        baru->prev = bantu;  
        baru->next = head;  
        head->prev = baru;  
    }  
    Cout<<"Data masuk\n";  
}
```

Ilustrasi:

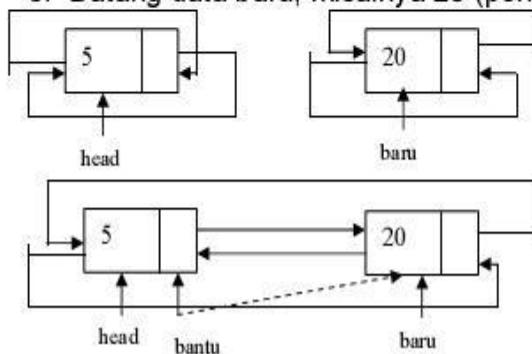
1. List masih kosong (`head=NULL`)



2. Masuk data baru, misalnya 5

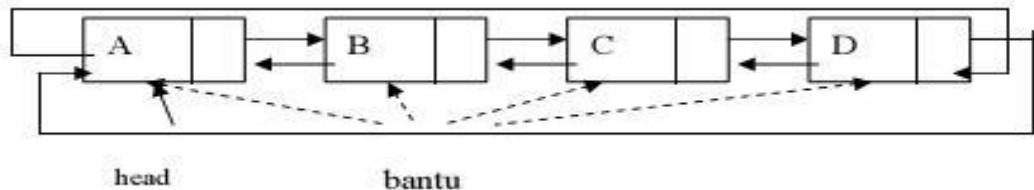


3. Datang data baru, misalnya 20 (penambahan di belakang)



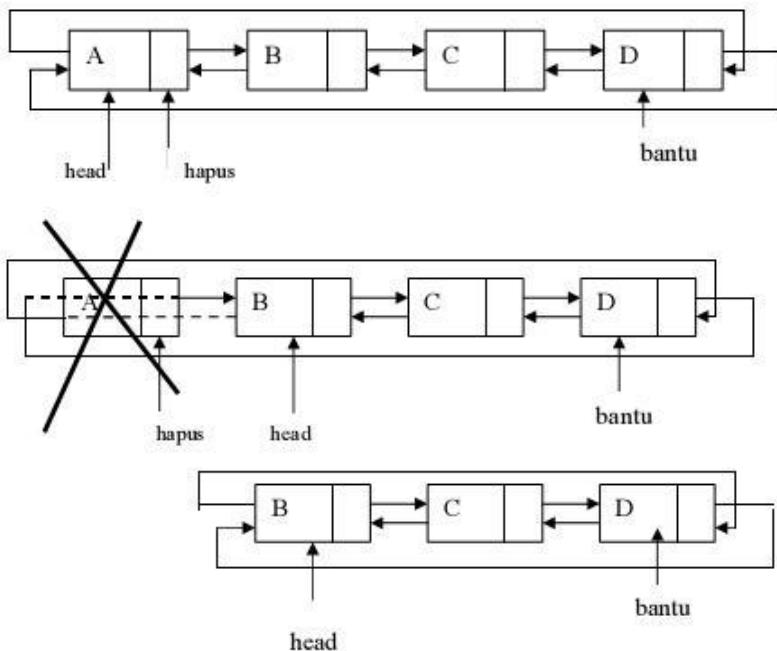
Function untuk menampilkan isi linked list

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



Function untuk menghapus node terbelakang

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



Function untuk menghapus node terbelakang

```
void hapusBelakang(){
    TNode *hapus,*bantu;
```

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

Function untuk menghapus semua elemen

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

Latihan

```
1 #include <iostream>
2 #include <conio.h>
3 #include <iomanip> //setw()
4 using namespace std;
5
6 struct node
7 {
8     int data;
9     node* next; //untuk menghubungkan dengan node lain, tipe data dibuat sm sprt aturan penggunaan pointer
10 };
11
12 node* head;
13 node* tail;
14 node* curr;
15 node* entry;
16 node* del;
17
18 void inisialisasi()
19 {
20     head = NULL;
21     tail = NULL;
22 }
23
24 void input(int dt)
25 {
26     entry = (node* )malloc(sizeof(node)); //alokasi memori
27     entry->data = dt;
28     entry->next = NULL;
29     if(head==NULL)
30     {
31         head = entry;
32         tail = head;
33     }
34     else
35     {
36         tail->next = entry;
37         tail = entry;
38     }
39 }
40
41 void hapus()
42 {
43     int simpan;
44     if(head==NULL)
45     {
46         cout<<"\nlinked list kosong, penghapusan tidak bisa dilakukan"<<endl;
47     }
48     else
49     {
50         simpan = head->data;
51         cout<<"\ndata yang dihapus adalah "<<simpan<<endl;
52         //hapus depan
53         del = head;
54         head = head->next;
55         delete del;
56     }
57 }
58
59 void cetak()
60 {
61     curr = head;
62     if(head == NULL)
```

```

63     cout<<"\ntidak ada data dalam linked list"<<endl;
64 else
65 {
66     cout<<"\nData yang ada dalam linked list adalah"<<endl;
67     cout<<setw(6);
68     while(curr!=NULL)
69     {
70         cout<<curr->data<<"->";
71         curr = curr->next;
72     }
73     cout<<endl;
74 }
75
76
77 void menu()
78 {
79     char pilih, ulang;
80     int data;
81
82     do
83     {
84         system("cls");
85         cout<<"SINGLE LINKED LIST NON CIRCULAR"<<endl;
86         cout<<"-----"<<endl;
87         cout<<"Menu : "<<endl;
88         cout<<"1. Input data"<<endl;
89         cout<<"2. Hapus data"<<endl;
90         cout<<"3. Cetak data"<<endl;
91         cout<<"4. Exit"<<endl;
92         cout<<"Masukkan pilihan Anda :";
93         cin>>pilih;
94         switch(pilih)
95         {
96             case '1':
97                 cout<<"\nMasukkan data :";
98                 cin>>data;
99                 input(data);
100                break;
101            case '2':
102                hapus();
103                break;
104            case '3':
105                cetak();
106                break;
107            case '4':
108                exit(0);
109                break;
110            default :
111                 cout<<"\nPilih ulang"<<endl;
112         }
113         cout<<"Kembali ke menu?(y/n)";
114         cin>>ulang;
115     }while(ulang=='y' || ulang=='Y');
116 }
117
118 int main()
119 {
120     inisialisasi();
121     menu();
122
123     return EXIT_SUCCESS;
124 }
```

Tugas Praktikum

TROUBLESHOOT PROGRAM :

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 //definisikan struct
5 struct SNode{
6     int data;
7     struct SNode *next;
8     struct SNode *before;
9 };
10
11 struct SNode *n_awal,    //node awal dari linked list
12     *n_tampil,           //node bantu untuk tampilkan data
13     *n_bantu;            //node bantu untuk insert node baru
14
15 int iterasi
16
17 //inisialisasi linked list
18 void inisialisasi(){
19     n_awal = NULL;
20 }
21
22 //tampilkan linked list
23 void tampil_list(){
24     n_tampil = n_awal
25
26     while(n_tampil != NULL){
27         printf("%d", n_tampil->data);
28         n_tampil = n_tampil->before;
29     }
30     printf("\n");
31 }
```

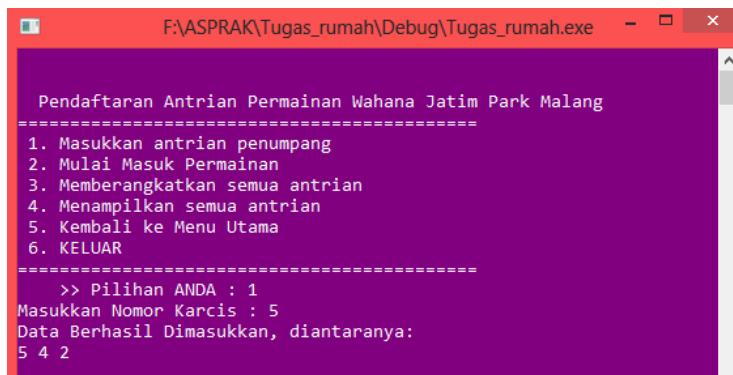
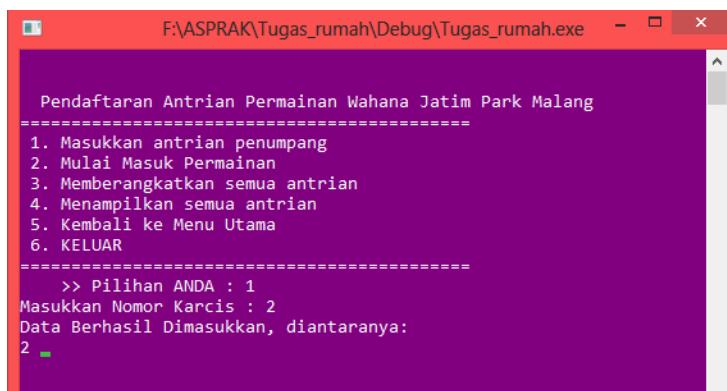
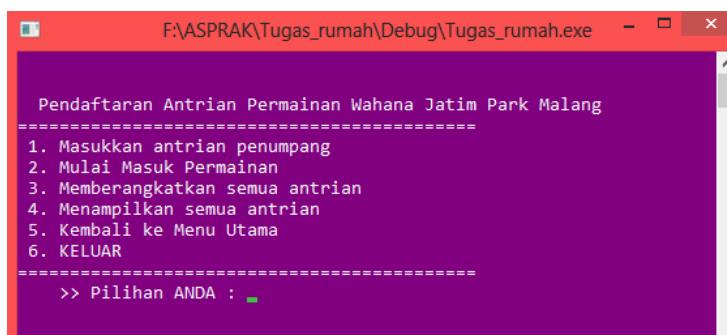
```

32      /*
33      //method untuk insert node baru
34      void insert_node(int data){
35
36          //node baru yang akan di insert
37          struct SNode *n_insert;
38          n_insert = (struct SNode *)malloc(sizeof(struct SNode));
39          n_insert->data = iterasi;
40          n_insert->next = NULL;
41          n_insert->before = NULL;
42
43          //kondisi linked list masih kosong
44          if(n_awal == NULL){
45              n_awal = n_insert;
46          }else{
47              n_bantu = n_awal;
48
49              //cari node terakhir
50              while(n_bantu->next != NULL){
51                  n_bantu = n_bantu->next;
52              }
53
54              //hubungkan node terakhir dengan node baru
55              if(n_bantu->next==NULL){
56                  {
57                      n_bantu->next = n_insert;
58                      insert->before = n_bantu;
59                  }
60              }
61          }
62      }
63
64      int main(int argc, char *argv[])
65      {
66          inisialisasi();
67          for(iterasi = 11; iterasi<=15; iterasi++)
68          {
69              insert_node(iterasi);
70
71              tampil_list();
72
73              system("PAUSE");
74          }
75      }

```

TUGAS RUMAH

Buatlah program seperti berikut ini menggunakan metode Linked List :



```
F:\ASPRAK\Tugas_rumah\Debug\Tugas_rumah.exe

Pendaftaran Antrian Permainan Wahana Jatim Park Malang
=====
1. Masukkan antrian penumpang
2. Mulai Masuk Permainan
3. Memberangkatkan semua antrian
4. Menampilkan semua antrian
5. Kembali ke Menu Utama
6. KELUAR
=====
>> Pilihan ANDA : 2
Antrian Pertama Telah Masuk ke Permainan Wahana Jatim Park
Antrian Sekarang, diantaranya:
5 4
```

```
F:\ASPRAK\Tugas_rumah\Debug\Tugas_rumah.exe

Pendaftaran Antrian Permainan Wahana Jatim Park Malang
=====
1. Masukkan antrian penumpang
2. Mulai Masuk Permainan
3. Memberangkatkan semua antrian
4. Menampilkan semua antrian
5. Kembali ke Menu Utama
6. KELUAR
=====
>> Pilihan ANDA : 3
Semua Antrian Masuk Dalam Permianan Wahana Jatim Park
```

MODUL VII

TREE (POHON)

A. TUJUAN

- Mahasiswa mampu menjelaskan mengenai algoritma Tree
- Mahasiswa mampu membuat dan mendeklarasikan struktur algoritma Tree
- Mahasiswa mampu menerapkan dan mengimplementasikan algoritma Tree

B. DASAR TEORI

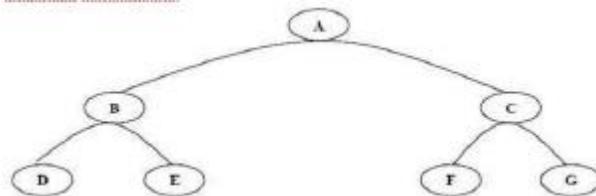
1. PENGERTIAN TREE

Kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layaknya struktur sebuah pohon. Struktur pohon adalah suatu cara merepresentasikan suatu struktur hirarki (one-to-many) secara grafis yang mirip sebuah pohon, walaupun pohon tersebut hanya tampak sebagai kumpulan node-node dari atas ke bawah. Suatu struktur data yang tidak linier yang menggambarkan hubungan yang hirarkis (one-to-many) dan tidak linier antara elemen-elemennya.

2. ISTILAH DALAM TREE

Predecesor	Node yang berada diatas node tertentu.
Successor	Node yang berada dibawah node tertentu.
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendant	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node.
Child	Successor satu level di bawah suatu node.
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendantnya.
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan dalam suatu tree
Root	Node khusus yang tidak memiliki predecessor.
Leaf	Node-node dalam tree yang tidak memiliki successor.
Degree	Banyaknya child dalam suatu node

Ilustrasi Algoritma Tree



Ancestor (F) = C,A

Descendant (C) = F,G

Parent(D) = B

Child(A) = B,C

Sibling(F) = G

Size = 7

Height = 3

Root = A

Leaf = D,E,F,G

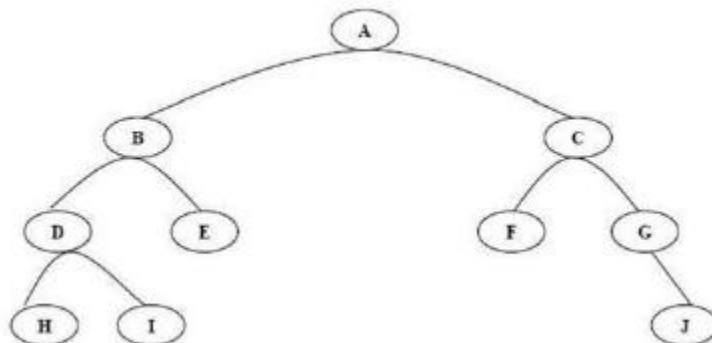
Degree = 2

3. JENIS - JENIS BINARY TREE

Tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua sub pohon dan kedua subpohon harus terpisah.

Kelebihan struktur Binary Tree :

- Mudah dalam penyusunan algoritma sorting
- Searching data relatif cepat
- Fleksibel dalam penambahan dan penghapusan data



4. IMPLEMENTASI ALGORITMA TREE

Insert : menambahkan node dalam tree

Jika data yang akan dimasukkan lebih besar daripada elemen root, maka akan diletakkan di node sebelah kanan, sebaliknya jika lebih kecil maka akan diletakkan di node sebelah kiri. Untuk data pertama akan menjadi elemen root.

```

void insert(int data,int node = 1){

    if(tree[node] == int()){ //int() same NULL
        tree[node] = data;
        curr++;
    }
    else if(data < tree[node]){
        insert(data, 2*node); //do recursion
    }
    else if(data > tree[node]){
        insert(data, 2*node+1); //do recursion
    }
}
  
```

PreOrder: cetak node yang dikunjungi, kunjungi left, kunjungi right

```
void preorder(int node = 1){
    int left = 2 * node;
    int right = 2 * node+1;
    if(empty() == true){
        cout<<"Node Masih Kosong"<<endl;
        return;
    }

    if(tree[node] != int()){
        cout<<tree[node]<< " ";

        if(tree[left] != int()) preorder(2*node); //do recursion
        if(tree[right] != int()) preorder(2*node+1); //do recursion
    }
}
```

InOrder: kunjungi left, cetak node yang dikunjungi, kunjungi right

```
void inorder(int node = 1){
    int left = 2 * node;
    int right = 2 * node+1;

    if(empty() == true){
        cout<<"Node Masih Kosong"<<endl;
        return;
    }

    if(tree[node] != int()){

        if(tree[left] != int()) {
            inorder(2*node); //do recursion
        }

        cout<<tree[node]<< " ";

        if(tree[right] != int()) {
            inorder(2*node+1); //do recursion
        }
    }
}
```

PostOrder: kunjungi left, kunjungi right, cetak node yang dikunjungi

```
void postorder(int node = 1){
    int left = 2 * node;
    int right = 2 * node+1;
    if(empty() == true){
        cout<<"Node Masih Kosong"<<endl;
        return;
    }

    if(tree[node] != int()){
        if(tree[left] != int()) {
            postorder(2*node);
        }

        if(tree[right] != int()) {
            postorder(2*node+1);
        }

        cout<<tree[node]<< " ";
    }
}
```

Searching Data : untuk searching data kita tinggal menjelajahi node yang ada di dalam tree, algoritmanya adalah jika data yang di cari sama dengan root maka data di temukan, jika tidak maka akan mengecek kembali apakah data lebih besar dari root maka secara rekursif akan mencari ke kanan

```
void searchTree(int data,int node = 1){
    if(tree[node] == int()){
        cout<<"data tidak di temukan"<<endl;
        return;
    }

    if(data == tree[node]){
        cout<<"data ditemukan pada node ke : "<<node<<endl;
        return;
    }
    else if(data < tree[node]){
        searchTree(data,node*2);
    }
    else if(data > tree[node]){
        searchTree(data,2*node+1);
    }
}
```

C. LATIHAN

```
1 #include <iostream>
2 #include <vector> //library for vector
3 #include <windows.h>
4
5 using namespace std;
6
7 vector<int> tree; //like array but size is dynamic
8 int curr = 0; //size of node
9
10 bool empty(){
11     if(curr == 0) return true;
12     else return false;
13 }
14
15 void clear(){
16     curr = 0;
17     tree.assign(1000000,int());
18 }
19
20 //function to insert node in tree
21 void insert(int data,int node = 1){
22
23     if(tree[node] == int()){//int() same NULL
24         tree[node] = data;
25         curr++;
26     }
27     else if(data < tree[node]){
28         insert(data, 2*node); //do recursion
29     }
30     else if(data > tree[node]){
31         insert(data, 2*node+1); //do recursion
32     }
33 }
34
35 void preorder(int node = 1){
36     int left = 2 * node;
37     int right = 2 * node+1;
38     if(empty() == true){
39         cout<<"Node Masih Kosong"<<endl;
40         return;
41     }
42
43     if(tree[node] != int()){
44         cout<<tree[node]<<" ";
45
46         if(tree[left] != int()) preorder(2*node); //do recursion
47         if(tree[right] != int()) preorder(2*node+1); //do recursion
48     }
49 }
50 }
```

```
51
52 ▼ void inorder(int node = 1){
53     int left = 2 * node;
54     int right = 2 * node+1;
55
56 ▼     if(empty() == true){
57         cout<<"Node Masih Kosong"<<endl;
58         return;
59     }
60
61 ▼     if(tree[node] != int()){
62
63         if(tree[left] != int()) {
64             inorder(2*node); //do recursion
65         }
66
67         cout<<tree[node]<<" ";
68
69         if(tree[right] != int()) {
70             inorder(2*node+1); //do recursion
71         }
72     }
73 }
74
75 ▼ void postorder(int node = 1){
76     int left = 2 * node;
77     int right = 2 * node+1;
78 ▼     if(empty() == true){
79         cout<<"Node Masih Kosong"<<endl;
80         return;
81     }
82
83 ▼     if(tree[node] != int()){
84
85         if(tree[left] != int()) {
86             postorder(2*node);
87         }
88
89         if(tree[right] != int()) {
90             postorder(2*node+1);
91         }
92
93         cout<<tree[node]<<" ";
94     }
95 }
96 }
```

```
97 //function search binary tree
98 void searchTree(int data,int node = 1){
99    if(tree[node] == int()){
101       cout<<"data tidak di temukan"<<endl;
102       return;
103    }
104
105   if(data == tree[node]){
106      cout<<"data ditemukan pada node ke : "<<node<<endl;
107      return;
108   }
109   else if(data < tree[node]){
110      searchTree(data,node*2);
111   }
112   else if(data > tree[node]){
113      searchTree(data,2*node+1);
114   }
115 }
116
117 int main(){
118   int data,menu;
119   tree.assign(1000000,int()); //provide value to the vector tree
120
121 do{
122   //system("cls");
123   cout<<"MENU"<<endl;
124   cout<<"1. Insert Data"<<endl;
125   cout<<"2. Lihat Pre-Order"<<endl;
126   cout<<"3. Lihat In-Order"<<endl;
127   cout<<"4. Lihat Post-Order"<<endl;
128   cout<<"5. Lihat Jumlah Node "<<endl;
129   cout<<"6. Hapus Semua Node"<<endl;
130   cout<<"7. Cari Data"<<endl;
130   cout<<"7. Cari Data"<<endl;
131   cout<<"8. Keluar"<<endl;
132   cout<<"Pilih Menu : ";
133   cin>>menu;
134   cout<<endl;
135
136   switch(menu){
137     case 1:
138       cout<<"Masukkan Data : ";
139       cin>>data;
140       cout<<endl<<endl;
141       insert(data);
142       break;
143     case 2:
144       preorder();
145       cout<<endl<<endl;
146       break;
147     case 3:
148       inorder();
149       cout<<endl<<endl;
150       break;
```

```
151 ▼
152     case 4:
153         postorder();
154         cout<<endl<<endl;
155         break;
156     case 5:
157         cout<<curr<<endl<<endl;
158         break;
159     case 6:
160         clear();
161         cout<<"data sudah terhapus"<<endl<<endl;
162         break;
163     case 7:
164         cout<<"Masukkan Data yang akan di cari : ";
165         cin>>data;
166         searchTree(data);
167         cout<<endl;
168         break;
169     default :
170         cout<<"pilihan yang anda masukan salah"<<endl;
171     }
172 }while(menu != 8 );
```

D. TUGAS RUMAH

1. Buatlah sebuah fungsi atau prosedur yang dapat menghapus suatu node dari struktur data tree. Untuk algoritmanya silahkan lihat di visualgo.net
2. Buatlah sebuah fungsi untuk mencari Parent dan child dari suatu node dalam tree