ILK2211P

STRUKTUR DATA ALGORITMA

Disusun oleh:

1. Dr. Elviawaty Muisa Zamzami, ST, MT, MM
2. T. Henny Febriana Harumy, S.Kom., M.Kom
3. Arvin Sebastian Chandra
4. Muhammad Erald Setyaki
5. Muhammad Rivaldi Zulvie
6. Selfhy Agina Ginting

No . Dokumen	: IK-GKM-Fasilkom-TI-005-SDA
Berlaku Efektif	: 19 Oktober 2020
Revisi	: 08
Direvisi	: TIM GKM ILK Fasilkom-TI
Diperiksa/Disetujui	: GJM dan GKM ILK Fasilkom-TI USU
Disahkan Oleh	: Prof. Dr. Opim Salim Sitompul, M.Sc Dekan Fasilkom-TI USU

S-1 ILMU KOMPUTER
FASILKOM-TI
USU

KATA PENGANTAR

Puji syukur kehadirat Allah SWT. Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya sehingga penyusun dapat menyelesaikan buku penuntun praktikum ini. Sholawat dan salam kepada Nabi Muhammad saw. Sosok manusia teladan bagi hidup dan kehidupan manusia sampai akhir zaman.

Modul praktikum ini disusun berdasarkan silabus mata kuliah praktikum Program Studi S1 Ilmu Komputer Fasilkom-TI USU. Buku ini terdiri 8 modul atau judul percobaan yang terdiri dari tujuan, teori, latihan dan tugas praktikum dengan harapan mahasiswa dapat membuat jurnal sebagai tolak ukur bagi keberhasilan pencapaian tujuan praktikum. Buku penuntun praktikum ini merupakan buku pedoman dasar dan bahan ajar dalam pelaksanaan praktikum.

Atas tersusunnya buku ini, penyusun mengucapkan terima kasih kepada Ketua dan Sekretaris Program Studi S1 Ilmu Komputer, Bapak/Ibu Dosen, Staf Tata Usaha Program Studi S1 Ilmu Komputer dan semua pihak yang telah membantu dalam penyusunan buku penuntun praktikum ini. Buku penuntun praktikum digunakan oleh Program Studi S1 Ilmu Komputer Fasilkom-TI USU dan merupakan implementasi dari pelaksanaan Sistem Manajemen Mutu Perguruan Tinggi.

Buku penuntun ini adalah revisi dari buku penuntun praktikum terdahulu. Penyusun menyadari buku ini masih jauh dari kesempurnaan oleh karena itu penyusun sangat mengharapkan kritik dan saran dari para pembaca demi menyempurnakan buku penuntun praktikum ini. Dengan adanya buku penuntun praktikum diharapkan dapat mem bantu mahasiswa dalam pelaksanaan praktikum dan bermanfaat secara maksimal bagi semua pihak.

Medan, 19 Oktober 2020
Program Studi S1 Ilmu Komputer Fasilkom-TI USU
Ketua,

Dr. Poltak Sihombing,M.Kom
NIP. 19620317 199103 1 001

Nama :

NIM :

MODUL I
STRUKTUR DATA STATIS

I. Tujuan

1. Praktikan dapat memahami Struktur Data Statis.
2. Praktikan dapat membuat program yang mengaplikasikan Struktur Data Statis.

II. Teori

1. Struktur Data dan Algoritma

Berbicara mengenai struktur data dalam pemrograman, takkan terlepas dari kata algoritma. Struktur data adalah pengaturan data dalam memori komputer atau terkadang didalam *disk* dengan tujuan agara data dapat diakses secara efisien. Sedangkan algoritma sendiri merupakan langkah-langkah logis untuk memecahkan suatu masalah secara sistematis. Didalam struktur data algoritma digunakan untuk memanipulasi data yang tersimpan, mulai dari insert, delete, update, dan sebagainya.

2. Struct

Struct merupakan komponen yang sangat penting dalam struktur data, karena struct dapat menampung banyak data (variable) dengan tipe yang berbeda.

Dalam mendeklarasikan struct, ada beberapa cara penulisan yang biasa umum digunakan.

Pertama :

```
struct nama_struct {
    tipe_data_1 nama_var_1;
    tipe_data_2 nama_var_2;
    tipe_data_3 nama_var_3;
    ……
};
```

Kedua:

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

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

```
struct tipe_struct nama_variabel;
```

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

```
tipe_struct nama_variabel;
```

3. Struktur Data Statis

Struktur data statis adalah struktur data yang memiliki alokasi data yang tetap. Data yang dapat dimasukkan tidak dapat ditambah lagi jika data yang telah ada telah mencapai nilai maksimum alokasi data dan jika data yang ada kurang dari jumlah maksimum alokasi data, maka alokasi memori yang disediakan tetap (statis).

```
/*  Contoh Program ke : 1                                   */
/*  Program          : struktur data statis mahasiswa    */

#include<conio.h>
#include<stdio.h>
#include<iostream>
#include<string.h>
#define nmax 5        //banyak data yang dapat ditampung
using namespace std;

int n=0;                  //variable n sebagai banyak data yang telah tersimpan

struct data{              // struktur data mahasiswa yg akan diinput
        int nim;
        char nama[20];
        char kom;
        };

struct data maba[nmax];       //deklarasi variabel bertipe struct

void tambah_data();           // prototype fungsi void
void hapus_data();
void tampilkan_data();

main()
{
    int pil;

    menu :
    cout<<" \t\t MENU \n";
    cout<<"1. tambah data\n";
    cout<<"2. hapus data \n";
    cout<<"3. tampilkan data \n";
    cout<<"4. keluar \n";
    cout<<"\n Pilih menu (1/2/3/4) ? ";
    cin>>pil;

    if (pil==1) tambah_data();
    else if (pil==2) hapus_data();
    else if (pil==3) tampilkan_data();
    else if (pil==4) exit(1);                 // atau bisa pakai return(0)
    else {
        cout<<"pilihan tak tersedia...!!!\n";

    }
    goto menu;
          getch();
}
```

```
void tambah_data(){
    if (n<nmax){
      cout<<"nama : ";
      cin>>maba[n].nama;


      cout<<"NIM : ";
      cin>>maba[n].nim;
      cout<<"Kom : ";
      cin>>maba[n].kom;
      n++;
      }
    else cout<<"\n data telah melebihi...!!!\n maksimal data = "<<nmax;
}

void hapus_data(){
    int x;
    cout<<"pilih data yang akan dihapus (1 s.d. 5) : ";
    cin>>x;
    if (x<n && x>0){
      for (int i=x;i<n;i++){
         strcpy(maba[i-1].nama,maba[x].nama);
         maba[i-1].nim = maba[x].nim;
         maba[i-1].kom = maba[x].kom;
         }
      n--;
      }
    else if(x==n) n--;
    else cout<<"\n data yang ingin dihapus tidak ada...!!!\n";
}

void tampilkan_data(){
    if (n==0) cout<<"Tidak ada data yang disimpan...!!!\n";
    else {
        for (int i=0;i<n;i++){
           cout<<"Data ke-"<<i+1<<" : ";
           cout<<"\nNama : "<<maba[i].nama;
           cout<<"\nNIM  : "<<maba[i].nim;
           cout<<"\nKom  : "<<maba[i].kom;
           cout<<endl;
           }
    }
}
```

III. Tugas dan Latihan

1. Pada contoh program 1, tambahkan 4 fungsi lagi :
   a. Fungsi update data
   b. Fungsi bersihkan data
   c. Fungsi menyisipkan data
   d. Fungsi men-swap (menukar posisi) data

MODUL II
SORTING

I. Tujuan

1. Praktikan dapat memahami dan menjelaskan algoritma dari Bubble Sort, Selection Sort, Insertion Sort, Shell Sort dan Quick Sort.
2. Praktikan dapat membuat implementasi pribadi menggunakan algoritma yang ada.

II. Teori

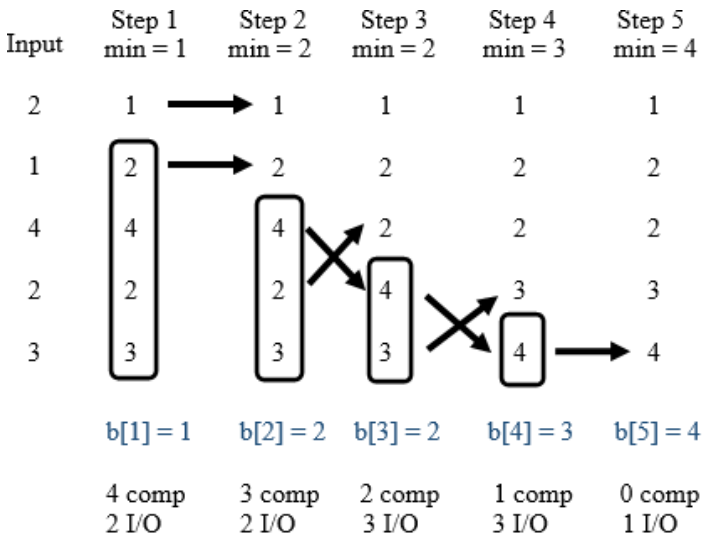
Sorting adalah proses menyusun elemen-elemen dengan tata urut tertentu dan proses tersebut terimplementasi dalam bermacam aplikasi. Beberapa macam algoritma sorting telah dibuat karena proses tersebut sangat mendasar dan sering digunakan. Oleh karena itu, pemahaman atas beberapa algoritma yang ada sangatlah berguna.

1. Bubble Sort

Bubble Sort merupakan cara pengurutan yang sederhana. Konsep dari ide dasarnya adalah seperti "gelembung air" untuk elemen struktur data yang semestinya berada pada posisi awal. Cara kerjanya adalah dengan berulang-ulang melakukan traversal (proses looping) terhadap elemen-elemen struktur data yang belum diurutkan. Di dalam traversal tersebut, nilai dari dua elemen struktur data dibandingkan. Jika ternyata urutannya tidak sesuai dengan "pesanan", maka dilakukan pertukaran (swap). Algoritma sorting ini disebut juga dengan comparison sort dikarenakan hanya mengandalkan perbandingan nilai elemen untuk mengoperasikan elemennya.

Pseudocode Bubble Sort :

```
procedure bubbleSort( A : list of sortable items )
   n = length(A)
   repeat
     swapped = false
     for i = 1 to n-1 inclusive do
       if A[i-1] > A[i] then
         swap(A[i-1], A[i])
         swapped = true
       end  if
     end   for
     n = n - 1
   until not swapped
end procedure
```



Work = (4 + 3 + 2 +1) comparisons + (2 + 2 + 3 + 3 + 1) I/O
= 10 comparisons + 11 I/O operations

Algoritma Bubble Sort

Algoritma bubble sort dapat diringkas sebagai berikut, jika N adalah panjang elemen struktur data, dengan elemen-elemennya adalah T1, T2, T3, …, TN-1,TN, maka:

1. Lakukan traversal untuk membandingkan dua elemen berdekatan. Traversal ini dilakukan dari belakang.
2. Jika elemen pada TN-1 > TN , maka lakukan pertukaran (swap). Jika tidak, lanjutkan ke proses traversal berikutnya sampai bertemu dengan bagian struktur data yang telah diurutkan.
3. Ulangi langkah di atas untuk struktur data yang tersisa.

6	5	3	1	8	7	2	4	tukar
5	6	3	1	8	7	2	4	tukar
5	3	6	1	8	7	2	4	tukar
5	3	1	6	8	7	2	4	tidak ditukar
5	3	1	6	8	7	2	4	tukar
5	3	1	6	7	8	2	4	tukar
5	3	1	6	7	2	8	4	tukar
5	3	1	6	7	2	4	8	Diulangi dari awal lagi sampai tidak ada data yang layak ditukar lagi

Latihan 6.1 – Bubble Sort

```
#include  <iostream>
using namespace std;
int main(){
   cout<<"Bubble sort\n\n";
   cout<<"Input banyak data: ";
   int n;
   cin>>n;
   int data[n];
   cout<<"input data: \n\n";
   for (int i=0;i<n;i++){
      cout<<"Data- "<<i+1<<" : ";cin>>data[i];
   }
// ---- bubble sort
   for (int i=1;i<n;i++){
      for (int j=n-1;j>=i;j--){
      if (data[j]<data[j-1]) //ascending
         swap(data[j],data[j-1]);
      }
   }

     //.....
   cout<<"\n sorted data\n";
   for (int i=0;i<n;i++){
      cout<<data[i]<<" ";
   }
   cout<<endl;
   return 0;
}
```

2. Selection Sort

Algoritma Sorting sederhana yang lain adalah Selection Sort. Ide dasarnya adalah melakukan beberapa kali *pass* untuk melakukan penyeleksian elemen struktur data. Untuk sorting ascending (menaik), elemen yang paling kecil di antara elemen-elemen yang belum urut, disimpan indeksnya, kemudian dilakukan pertukaran nilai elemen dengan indeks yang disimpan tersebut dengan elemen yang paling depan yang belum urut. Sebaliknya, untuk sorting descending (menurun), elemen yang paling besar yang disimpan indeksnya kemudian ditukar.

```
for i = 1 to n,
   k = i
   for j = i+1 to n,
            if a[j] < a[k],
                     k = j
               → invariant: a[k] smallest of a[i..n]
                     swap a[i,k]
         → invariant: a[1..i] in final position
end
```

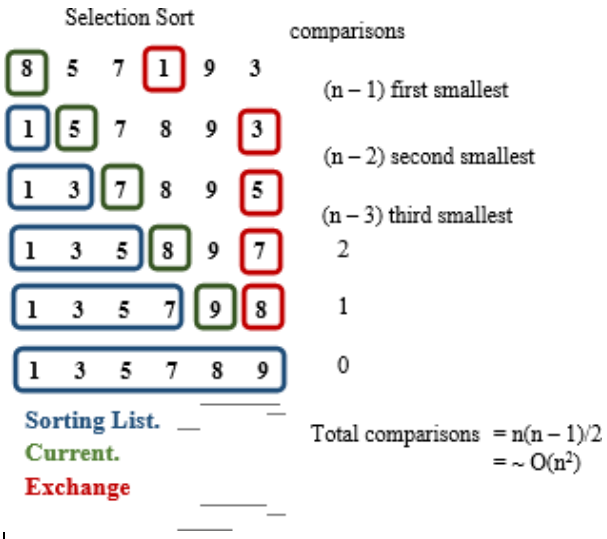
Pseudocode Selection Sort :

Algoritma Selection Sort

1. Temukan nilai yang paling minimum (atau sesuai keinginan) di dalam struktur data. Jika ascending, maka yang harus ditemukan adalah nilai yang paling minimum. Jika descending, maka temukan nilai yang paling maksimum.
2. Tukar nilai tersebut dengan nilai pada posisi pertama di bagian struktur data yang belum diurutkan.
3. Ulangi langkah di atas untuk bagian struktur data yang tersisa.

Latihan 6.2 – Selection Sort :

```
#include <iostream>
```




```
using namespace std;

int main() {
   cout<<"Selection Sort \n\n";
   cout<<"Input banyak data : ";
   int n,pos;
   cin>>n;
   int data[n];
   cout<<"Input Data : \n\n";
   for(int i=0; i<n; i++) {
```

```
    cout<<"Data-"<<i+1<<" : ";
    cin>>data[i];
  }


  //--------------------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]) //Ascending
          pos = j;
     }
     if(pos != i)
          swap(data[pos], data[i]);
  }
  //------------------------------
  cout<<"\nSorted Data\n";
  for(int i=0; i<n; i++) {
    cout<<data[i]<<" ";
  }
  cout<<"\n";
  return 0;
}
```

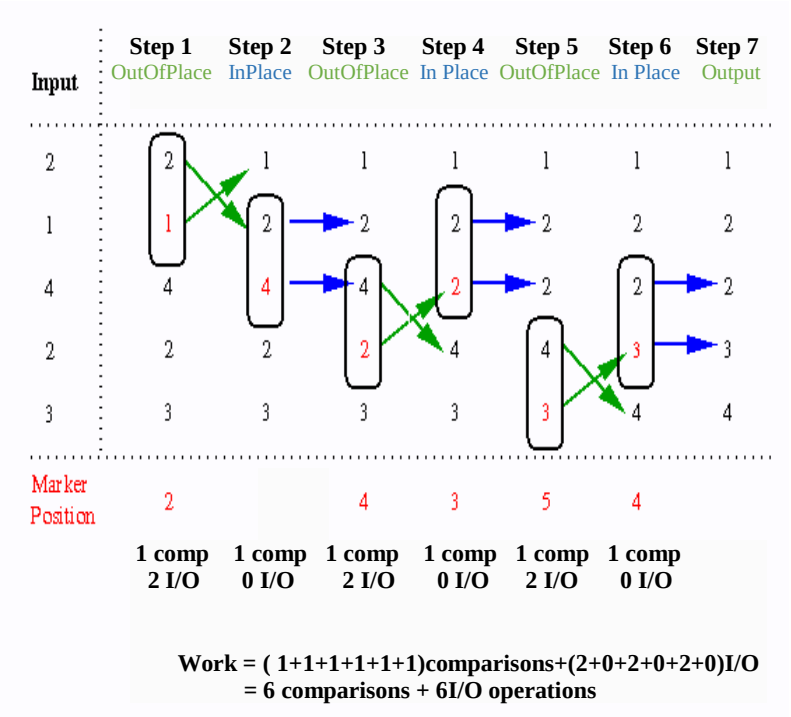
1. Insertion Sort

Insertion sort merupakan salah satu algoritma pengurutan dengan cara menyisipkan / insert jika menemukan nilai yang lebih kecil, sehingga algoritma ini menyelesaikan pengurutan secara bertahap dairi kiri ke kanan atau sebaliknya. Dengan syarat elemen yang akan diurutkan selanjutnya diurutkan dulu pada elemen-elemen yang telah terurut sebelumnya dengan cara langsung menyisipkan diantara elemen tersebut yang sesuai.

Cara kerja insertion sort sebagaimana namanya. Pertama-tama, dilakukan iterasi, dimana di setiap iterasi insertion sort memindahkan nilai elemen, kemudian menyisipkannya berulang-ulang sampai ketempat yang tepat. Begitu seterusnya dilakukan. Dari proses iterasi, seperti biasa, terbentuklah bagian yang telah di-sorting dan bagian yang belum di-sorting.

Pseudocode Insertion Sort :

```
for i = 2:n,
  for (k = i; k > 1 and a[k] < a[k-1]; k--)
    swap a[k,k-1]
    → invariant: a[1..i] is sorted
end
```



Secara sederhana insertion sort dapat dijelaskan dengan deretan angka dibawah ini :

Data awal : 21 32 4 6 2 7 1
Langkah 1 : 21 32 4 6 2 7 1
Langkah 2 : 21 32 4 6 2 7 1
Langkah 3 : 4 21 32 6 2 7 1
Langkah 4 : 4 6 21 32 2 7 1
Langkah 5 : 2 4 6 21 32 7 1
Langkah 6 : 2 4 6 7 21 32 1
Langkah 7 : 1 2 4 6 7 21 32
Hasil : 1 2 4 6 7 21 32

Keterangan :

Data yang berwarna merah merupakan data yang diurutkan (elemen), yang berwarna biru merupakan data yang disisipkan karena nilainya lebih kecil. Data yang berwarna hitam merupakan data yang tidak sorting. Sehingga jika digambarkan akan seperti deretan langkah langkah diatas. Entah itu secara ascending atau descending tetap saja urutan langkahnya seperti diatas, yaitu disisipkan pelan-pelan. Banyak orang analogikan dengan orang yang mengurutkan kartu, karena biasanya kartu diurutkan seperti ini, yaitu disisipkan yang nilainya sesuai.

Algoritma Insertion Sort

Algoritma Insertion Sort dapat dirangkum sebagai berikut:

1. Simpan nilai Ti ke dalam variabel sementara, dengan i = 1.
2. Bandingkan nilainya dengan elemen sebelumnya.
3. Jika elemen sebelumnya (Ti-1) lebih besar nilainya daripada Ti, maka tindih nilai Ti dengan nilai Ti-1 tersebut. Decrement i (kurangi nilainya dengan 1).
4. Lakukan terus poin ke-tiga, sampai Ti-1 ≤ Ti.
5. Jika Ti-1 ≤ Ti terpenuhi, tindih nilai di Ti dengan variabel sementara yang disimpan sebelumnya.
6. Ulangi langkah dari poin 1 di atas dengan i di-increment (ditambah satu).

Latihan 6.3 – Insertion Sort :

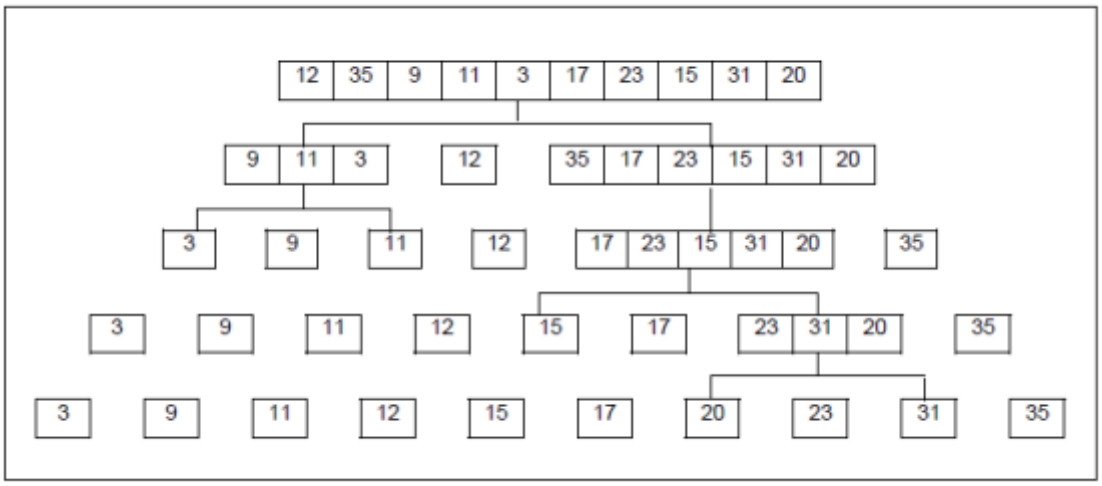
```
#include <iostream>

using namespace std;

int main() {
   cout<<"Insertion Sort \n\n";
   cout<<"Input banyak data : ";
   int n,pos;
   cin>>n;
   int data[n];
   cout<<"Input Data : \n\n";
   for(int i=0; i<n; i++) {
     cout<<"Data-"<<i+1<<" : ";
     cin>>data[i];
   }
   //--------------------Insertion Sort
   int temp,j;
   for(int i=1; i<n; i++) {
     temp = data[i];
     j = i - 1;
     while(data[j] > temp && j>=0) { //Ascending
       data[j+1] = data[j];
       j--;
     }
     data[j+1]=temp;
   }

   //-------------------------------
   cout<<"\nSorted Data\n";
   for(int i=0; i<n; i++) {
     cout<<data[i]<<" ";
   }
   cout<<"\n";
   return 0;
}
```

2. Quick Sort

Quick Sort adalah metode pengurutan data yang dikemukan pertama kali oleh C.AR Hoare pada tahun 1962. Metode ini menggunakan strategi “pecah-pecah” dengan mekanisme seperti berikut : Larik L[p..r] (dengan indeks terkecil adalah p dan indeks terbesar yaitu r) disusun ulang (dipartisi) menjadi dua buah larik A[p..q] dan A[q+1..r] sehingga setiap elemen dalam A[q+1..r]. Selanjutnya kedua larik tersebut diurutkan secara rekursif. Dengan sendirinya kombinasi kedua larik tersebut membentuk larik dengan data yang telah urut.

Pseudocode QuickSort :

```
void quicksort (int a[], int left, int right)
begin procedure
   int i, j, v;

   if (right > left)
   v = a[right]; i = left - 1; j = right;
      for(;;)
         while(a[++i] < v);
         while(a[--j] > v);
         if(i >= j) break;
         swap(a, i, j);

      swap(a, i, right);
      quicksort(a, left, i-1);
      quicksort(a, i+1, right);
end procedure
```



3	9	11	12	15	17	20	23	31	35

Algoritma Quick Sort

Algoritma ini terdiri dari 4 langkah utama:

1. Jika struktur data terdiri dari 1 atau 0 elemen yang harus diurutkan, kembalikan struktur data itu apa adanya.
2. Ambil sebuah elemen yang akan digunakansebagai pivot point (poin poros). (Biasanya elemen yang paling kiri.)
3. Bagi struktur data menjadi dua bagian – satu dengan elemen-elemen yang lebih besar daripada pivot point, dan yang lainnya dengan elemen-elemen yang lebih kecil dari pada pivot point.
4. Ulangi algoritma secara rekursif terhadap kedua paruh struktur data.

Latihan 6.5 – Quick Sort :

```
#include <iostream>
#define n 12
using namespace std;

int A[n] = {32,5,13,2,35,29,17,14,8,20,11,23};

void sort(int l, int r);

main()
{
  int i,kiri, kanan, kali, X;
  cout<<"Sebelum di Sort\n";
  for(i=0; i<=n-1; i++)
    cout<<A[i]<<" ";
  cout<<endl;

  sort(0, n-1);

  cout<<"\nSetelah di Sort\n";
  for(i=0; i<=n-1; i++)
    cout<<A[i]<<" ";
  cout<<endl;
  return 0;
}

void sort(int kiri, int kanan)
{
  int i, ii, j, x, pivot, w,k;
  i = kiri;
  j = kanan;
  pivot = A[i];
  //pivot = A[i + (rand() % (j-i + 1))];
  while(i <= j)
  {
    while(A[i] < pivot)
      i++;
    while(pivot < A[j])
      j--;
    if(i <= j)
    {
      swap(A[i], A[j]);
      i++;
      j--;
    }
  }
```

```
  if(kiri < j)
    sort(kiri, j);
  if(i < kanan)
    sort(i, kanan);
}
```

Tugas dan Latihan

1. Buatlah proses pengurutan secara **descending** pada data 10 11 23 21 0 0 8 7 dengan menggunakan :
   a. Bubble Sort.
   b. Selection Sort.
   c. Insertion Sort.

MODUL III
SEARCHING

I. Tujuan

1. Praktikan dapat menunjukkan beberapa algoritma dalam Pencarian.
2. Praktikan dapat menunjukkan bahwa pencarian merupakan suatu persoalan yang bisa diselesaikan dengan beberapa algoritma yang berbeda.
3. Praktikan dapat memilih algoritma yang paling sesuai untuk menyelesaikan suatu permasalahan pemrograman.

II. Teori

Algoritma searching adalah algoritma yang menerima sebuah argument kunci dan dengan langkah-langkah tertentu akan mencari rekaman dengan kunci tersebut. Setelah proses pencarian dilaksanakan, akan diperoleh salah satu dari dua kemungkinan, yaitu data yang dicari ditemukan (successful) atau tidak ditemukan (unsuccessful).

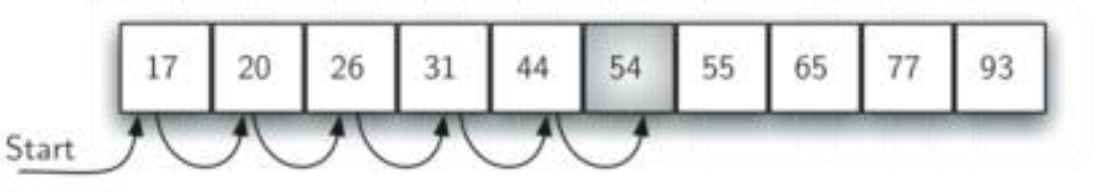
1. Sequential Search / Linear Search

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

Pencarian berurutan menggunakan prinsip sebagai berikut :

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

Kelemahan pada kasus yang paling buruk, untuk N elemen data harus dilakukan pencarian sebanyak N kali pula.



Mencari angka 54 dari deretan angka, angka ditemukan setelah 6 kali pengecekan

Pseudocode :

```
function bool linearSearch (cari,  data[0..n]) {
  for( cek = 0 to data.lenght)   {
     if(data[cek]==cari)
     {
       flag=1;
       break;
     }
  }
  if(flag==1)
     return true;
}
```

Latihan 1.1 : Linier Search

```
#include <iostream>
using namespace std;
int main()
{
        int data[8] = {8,10,6,-2,11,7,1,100};
        int cari;
        int flag=0;
        cout<<"Masukkan angka yang akan dicari ";
        cin>>cari;

        for(int i = 0; i < 8 ; i++)
        {
                if(data[i] == cari)
                    flag=1;
        }
        if(flag==1)
             cout<<"Data ada!\n";
        else
            cout<<"Data tidak ada!\n";
        return 1;
}
```

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

- ✓ Program menggunakan sebuah variabel flag yang berguna untuk menandai ada atau tidaknya data yang dicari dalam array data. Hanya bernilai 0 atau 1.
- ✓ Flag pertama kali diinisialiasasi dengan nilai 0.
- ✓ Jika ditemukan, maka flag akan diset menjadi 1, jika tidak ada maka flag akan tetap bernilai 0.
- ✓ Semua elemen array data akan dibandingkan satu persatu dengan data yang dicari dan diinputkan oleh user.

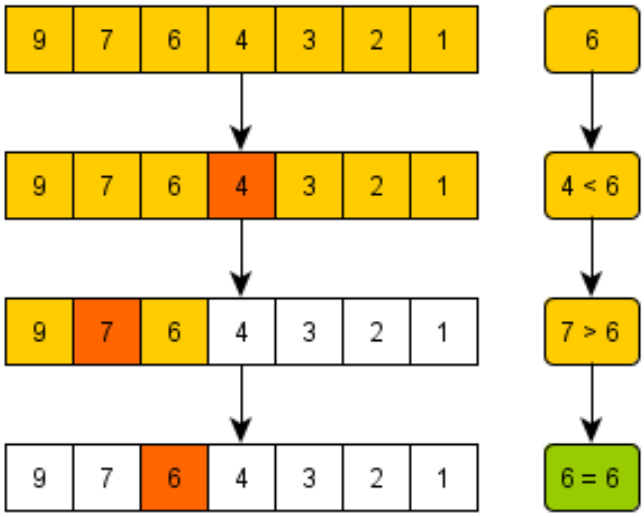
2. Binary Search

Terdapat metode pencarian pada data terurut yang paling efficient, yaitu metode pencarian bagidua atau pencarian biner *(binary search).* Sebuah algoritma pencarian biner (binary search) adalah sebuah teknik untuk menemukan nilai tertentu dalam sebuah larik (array) linear, dengan menghilangkan setengah data pada setiap langkah, dipakai secara luas tetapi tidak secara ekslusif dalam ilmu komputer. Metode ini digunakan untuk kebutuhan pencarian dengan waktu yang cepat.

Sebuah pencarian biner mencari nilai tengah (median), melakukan sebuah pembandingan untuk menentukan apakah nilai yang dicari ada sebelum atau sesudahnya, kemudian mencari setengah sisanya dengan cara yang sama. Sebuah pencarian biner adalah salah satu contoh dari algoritma divide and conquer (atau lebih khusus algoritma decrease and conquer) dan sebuah pencarian dikotomi (lebih rinci di Algoritma pencarian).

Prinsip dari pencarian biner dapat dijelaskan sebagai berikut :

1. mula-mula diambil posisi awal 0 dan posisi akhir = N - 1, kemudian dicari posisi data tengah dengan rumus (posisi awal + posisi akhir) / 2.
2. Kemudian data yang dicari dibandingkan dengan data tengah.
3. Jika lebih kecil, proses dilakukan kembali tetapi posisi akhir dianggap sama dengan posisi tengah –1.
4. Jika lebih besar, porses dilakukan kembali tetapi posisi awal dianggap sama dengan posisi tengah + 1.
5. Demikian seterusnya sampai data tengah sama dengan yang dicari.



Mencari angka 6 dari deretan angka yang terurut secara descending, angka ditemukan setelah 3 kali pengecekan

Pseudocode :

```
function binarySearch( data[0..n], search ) {
        while ( low <= high && flag == false ) {
                mid = ( low + high ) / 2

                if ( data[mid] == search ) {
                        flag == true
                }
                else if ( data[mid] < cari ) {
                        low = mid + 1
                }
                else {
                        high = mid -1
                }
        }
        return flag;
}
```

Pseudocode dengan fungsi rekursif :

```
function binarySearch(a, value, left, right)
   if right < left
      return not found
   mid := floor((right-left)/2)+left
   if a[mid] = value
      return mid
   if value < a[mid]
      return binarySearch(a, value, left, mid-1)
   else
      return binarySearch(a, value, mid+1, right)
```

Latihan 1.2 : Binary Search

```
#include <iostream>
using namespace std;

main()
{
    int data[7] = {10,13,17,34,58,67,99};
    int left = 0, right, middle, cari, flag = 0;
    int N = 7;
    right = N-1;
    cout<<"Masukkan data yang ingin dicari = ";
    cin>>cari;
    while(left<=right && flag==0)
    {
      middle=(left+right)/2;
```

```
        cout<<"Data tengah = "<<middle<<endl;
        if(data[middle]==cari)
            flag=1;
        else if (cari<data[middle])
        {
            cout<<"Cari di kiri\n";
            right=middle-1;
        }
        else
        {
           left=middle+1;
           cout<<"Cari di kanan\n";
        }
    }
    if(flag==1)
       cout<<"Data ditemukan!\n\n";
    else
       cout<<"Data tidak ditemukan!\n\n";
    system("pause");
}
```

3. Interpolation Search

Interpolation Search adalah sebuah algoritma atau metode untuk mencari nilai key yang diberikan dalam array diindeks yang telah diperintahkan oleh nilai-nilai kunci. Metode ini didasari pada proses pencarian nomor telepon pada buku telepon yang mana manusia mencari melalui dengan nilai kunci yang terdapat pada buku. Teknik ini dilakukan pada data yang sudah terurut berdasarkan kunci Tertentu. Teknik searching ini dilakukan dengan perkiraan letak data.

Contoh ilustrasi: jika kita hendak mencari suatu nama di dalam buku telepon, misal yang berawalan dengan huruf T, maka kita tidak akan mencarinya dari awal buku, tapi kita langsung membukanya pada 2/3 atau ¾ dari tebal buku. Jadi kita mencari data secara relatif terhadap jumlah data. Rumus posisi relatif kunci pencarian dihitung dengan rumus:

$$Posisi = \frac{kunci - data\ [low]}{Data\ [high] - \ data\ [low]}\ x\ (high - low) + \ low$$

- Jika data[posisi] > data yg dicari, high = pos – 1
- Jika data[posisi] < data yg dicari, high = pos + 1

Algoritma :

1. Tentukan posisi dengan rumus di atas.
2. Jika data[posisi]=kunci, proses selesai, flag true;
3. Jika data[posisi]>kunci, high=posisi-1;
4. Jika data[posisi]<kunci, low=posisi+1;
5. Ulangi langkah 1 selama low<=high dan flag=false.

Indeks	0	1	2	3	4	5

Nilai	4	5	7	10	12	15

Mencari 7

low=0, high=6-1=5
posisi=(0+((7-4)/(15-4))*(5-0))=1
data[1]=7--->false
data[1]<7 --> low=1+1

low=2, high=5
posisi=2+((7-7)/(15-7))*3=2
data[2]=7---> true
data ditemukan!

Mencari angka 7 dari deretan angka yang terurut secara ascending, angka ditemukan setelah 2 kali pengecekan

Pseudocode :

```
low=0;
high = n-1;
while(low<=high && flag==0){
      posisi=low+((kunci-data[low])/(data[high]-data[low]))*(high- low)
      if(data[posisi]==kunci){
         flag=1;
         break;
      }
      else if (data[posisi]<kunci){
         low=posisi+1;
      }
      else{
         high=posisi-1;
      }
    if(flag==1)
       cout<<"Data ditemukan!\n\n";
    else
       cout<<"Data tidak ditemukan!\n\n";
    system("pause");
}
```

Latihan 1.3 : Interpolation Search

```
#include <iostream>
#include <math.h>
using namespace std;

main()
{
    int data[7] = {10,13,17,34,58,67,99};
    int cari, posisi;
    float posisi1;
    int N = 7, flag = 0, low = 0, high = N-1;
    cout<<"Masukkan data yang ingin dicari = ";
    cin>>cari;
```

```
    do
    {
       posisi1 = (cari-data[low]) / (data[high]-data[low]) * (high-low)+low;
       posisi = floor(posisi1);
       if(data[posisi]==cari)
       {
          flag=1;
          break;
       }
       if(data[posisi]>cari)
          high=posisi-1;
       else if(data[posisi]<cari)
          low=posisi+1;
    }
    while(cari>data[low] && cari <=data[high]);
    if(flag==1)
       cout<<"Data ditemukan!\n\n";
    else
       cout<<"Data tidak ditemukan!\n\n";
    system("pause");
}
```

III. Tugas

1. Buatlah penjelasan singkat menggunakan bahasa sendiri, pseucodoce serta contoh program yang diberi penjelasan dalam bentuk komentar dari:
   a. Sequential Search
   b. Binary Search
   c. Interpolation Search

2. Cari kelemahan dan kelebihan dari :
   a. Sequential Search.
   b. Binary Search.
   c. Interpolation Search.

MODUL IV

STRUKTUR DATA DINAMIS

I. Tujuan

1. Praktikan dapat memahami Struktur Data Dinamis
2. Praktikan dapat membuat program Struktur data Dinamis.
3. Praktikan dapat membandingkan Struktur data Dinamis dengan Struktur data Statis.

II. Teori

Berbeda halnya dengan struktur data statis yang kita pelajari di minggu yang lalu, dalam struktur data dinamis ukuran dari suatu struktur dapat kita tambah atau kurangi pada saat program sedang dijalankan. Namun dalam stuktur data dinamis data tersimpan secara acak dalam memori, sehingga untuk mengakses data tertenu dalam sebuah struktur data dinamis memerlukan waktu yang agak lama karena harus dicari mulai dari data pertama dalam struktur data tersebut. Oleh sebab itu struktur data dinamis baik digunakan dalam pengelolaan data dalam ukuran yang besar.

Sebelum kita membuat program struktur data dinamis ada baiknya kita mengulang sedikit materi mengenai pointer dan struktur karena, materi tersebut adalah dasar pokok yang dibutuhkan untuk memahami keseluruhan materi data dinamis.

1. Pointer

Pointer (penunjuk) adalah suatu variable yang digunakan untuk menyimpan **alamat** dari suatu data. Pointer dideklarasikan dengan menggunakan tanda asteris(*) kemudian diikuti oleh nama pointer tersebut.

Contoh Program :

```
#include<iostream>
using namespace std;

int main(){
   int a, *b;
   a=5;
   b=&a;
   cout<<"Pointer b menyimpan alamat  : "<<b<<endl;
   cout<<"Isi (nilai) dari alamat tersebut : "<<*b<<endl;
```

2. Struktur

Struktur adalah sekumpulan variable yang tiap – tiap variabel tersebut dapat berbeda tipe dan dikelompokkan ke dalam satu nama. Struktur membantu mengatur data – data yang rumit dengan menggelompokkan sekelompok variable menjadi satu kesatuan.

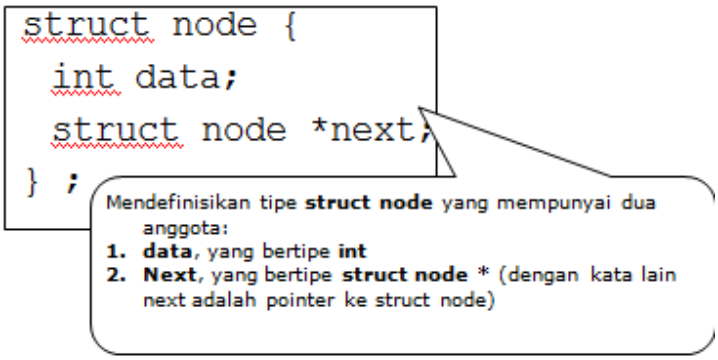
Contoh program :

```
#include<iostream>
using namespace std;

struct mahasiswa{
        string nama;
        int nim;
};

int main(){
    mahasiswa mhs;
    cout<<"Nama Mahasiswa        : ";
    cin>>mhs.nama;
    cout<<"NIM                   : ";
    cin>>mhs.nim;
    cout<<"Nama Mahasiswa        : "<<mhs.nama<<endl;
    cout<<"NIM                   : "<<mhs.nim<<endl;
}
```

3. Linked-List

Sekarang kita akan mempelajari struktur data dinamis yang sangat mendasar, yaitu **linked-list.** penguasaan terhadap linked-list merupakan kunci untuk menguasai struktur data dinamis lainnya. Self-referential structure adalah suatu structure yang mengandung anggota berupa pointer ke structure itu sendiri. Istilah "self-referential" berasal dari kenyataan bahwa pointer next itu menunjuk ke structure di mana next merupakan anggota. Structure-structure self-referential dapat digandengkan bersama *(di-link)* untuk membentuk struktur data seperti *list, stack, queue* dan tree.

Deklarasi Dalam C++.



Terdapat dua jenis operator yang digunakan untuk mengakses anggota dari struktur yaitu operator titik (.) dan operator panah (->). Operator titik digunakan untuk mengakses variabel tunggal sedangkan operator panah digunakan untuk mengakses variable pointer.

Contoh Program :

```
struct node {
  string info;
  struct node *next;
};
typedef struct node simpul;

int main() {
   simpul s1, s2, *psimpul;
   s1.info = "ILKOM";
   s1.next = &s2;
   s2.info = "USU";
   s2.next = NULL;
   psimpul = &s1;
      cout<<" s1.info = "<<s1.info;
      cout<<"\n s2.info = "<<s2.info;
      cout<<"\n psimpul->info = "<<psimpul->info;
      cout<<"\n psimpul->next->info = "<<psimpul->next->info;
}
```

III. Tugas

1. Buatlah Program untuk menukarkan 2 nilai menggunakan pointer.
2. Buatlah Ilustrasi pengalokasian memory secara dinamis menggunakan pointer.
3. Buatlah Sebuah Program yang mimiliki setidaknya 5 data menggunakan metode Linked-List.

MODUL V
STRUKTUR DATA STACK DAN QUEUE

I. Tujuan

1. Praktikan dapat memahami apa itu Stack dan Queue.
2. Praktikan dapat membuat program Stack dan Queue.

II. Teori

1. Stack (Tumpukan)

A. Pengertian

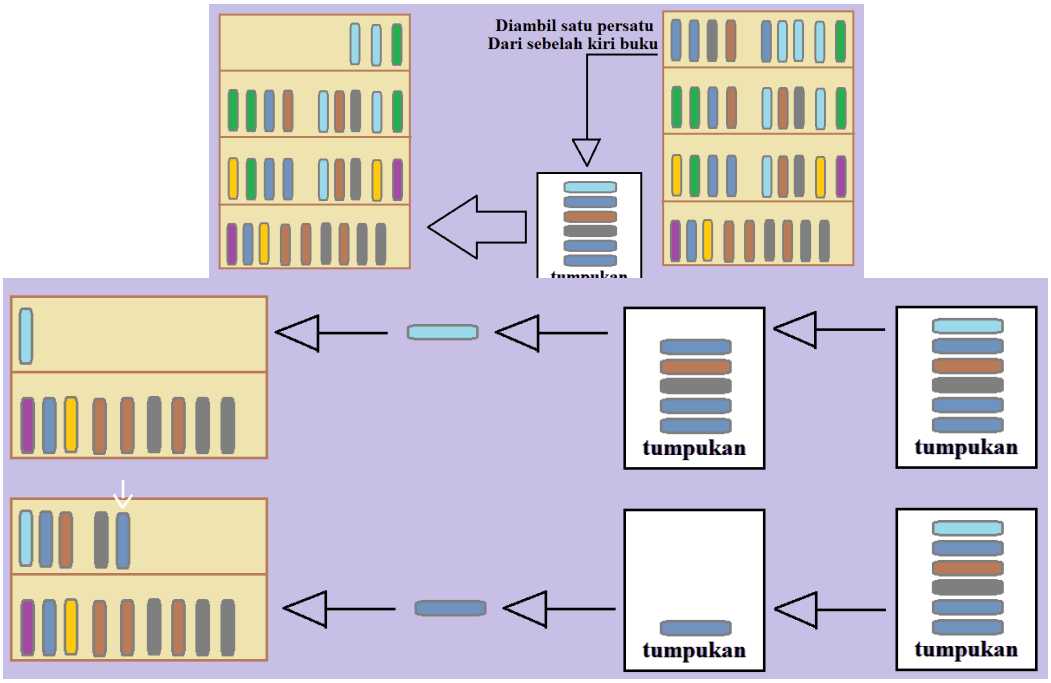
Stack adalah suatu tumpukan dari benda. Konsep Utamanya adalah LIFO(Last In First Out), maksudnya, benda yang terakhir masuk dalam stack akan menjadi benda yang pertama sekali yang akan dikeluarkan dari stack. Ada 2 operasi dasar yang bisa dilaksanakan pada sebuah tumpukan, yaitu operasi menyisipkan data *(push)* dan operasi menghapus data *(pop)*. Selain itu stack mempunyai operasi mengosongkan (*empty*), melihat banyak tumpukan (*size*), dan melihat tumpukan paling atas (*top*). Konsep stack yang anda bisa terapkan dalam c++, dengan konsep stack beberapa peran dan fungsi yang harus dipahami ialah :

push adalah operasi untuk menambahkan *stack*(data tumpukan) dalam program, kita harus bisa menambahkan satu persatu elemen dalam program untuk bisa dijadikan data *stack* , yang otomatis menambahkan data tanpa me-*replace*nya kita bisa menggunakan *array* untuk menampung datangnya.

pop adalah operasi penghapusan elemen yang sudah ditambahkan dalam *stack*(tumpukan) yang kita tambahkan sebelumnya pada posisi paling atas, seperti pada implementasi peletakan buku perpustakaan.

Implementasi dari stack dapat dilakukan dengan membuat linked list, namun untuk bahasa pemograman seperti C++ sudah ada class yang menampung data layaknya stack.

B. Implementasi konsep *stack*



C. Contoh Program

```
#include<iostream>
#include<stack>
using namespace std;
```

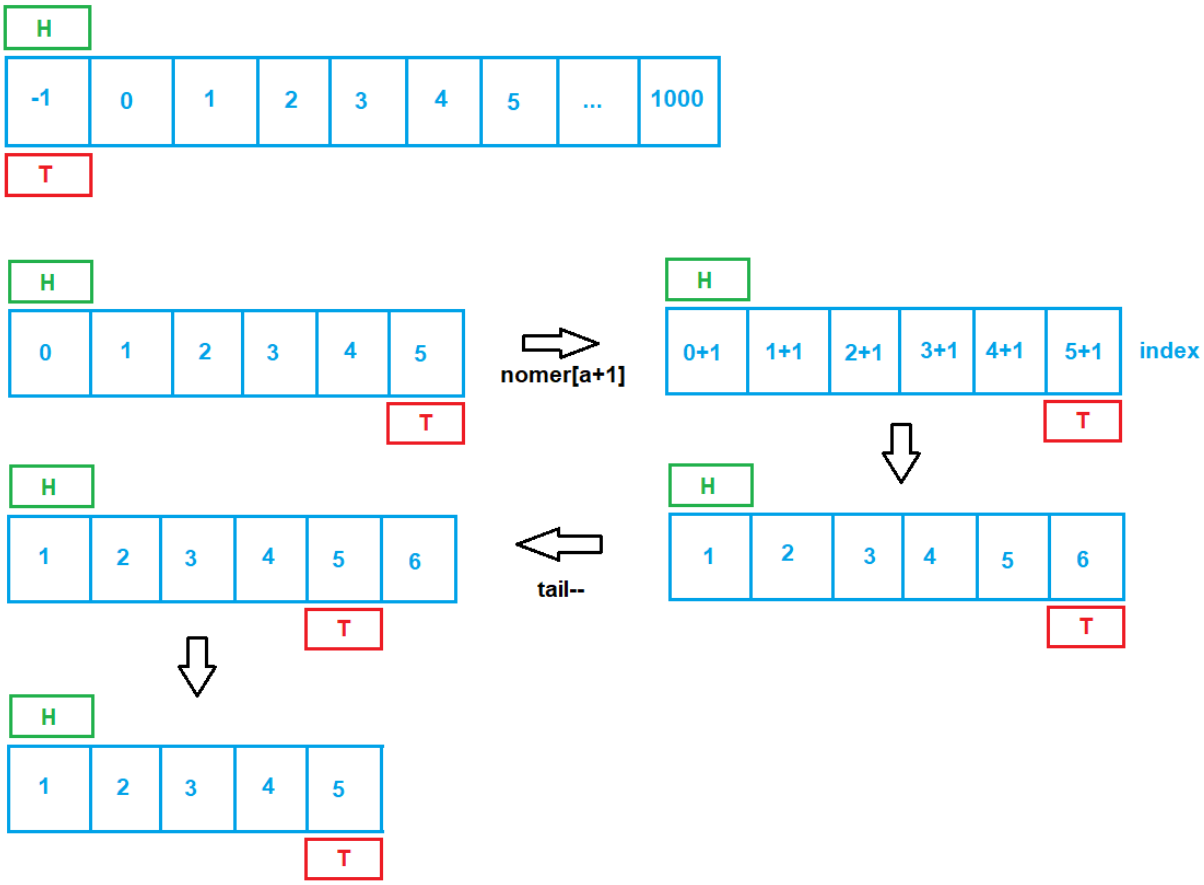
2. Queue

A. Pengertian

Queue secara harafiah diartikan sebagai antrian. Queue merupakan salah satu contoh aplikasi dari pembuatan double linked list yang cukup sering kita temui dalam kehidupan sehari-hari, misalnya pada saat anda mengantri di Mesin ATM. Apabila sesorang masuk dalam sebuah antrian disebut Enqueue. Dalam suatu antrian, ada prinsip FIFO(First In First Out) merupakan antrian biasa dimana yang pertama datang akan menjadi yang pertama keluar dari antrian. Antrian juga merupakan suatu kumpulan data yang penambahan elemennya hanya bisa dilakukan pada suatau ujung(disebut dengan belakang atau rear), dan penghapusan atau pengambilan elemen dilakukan lewat ujung yang lain (disebut dengan isi depan atau front). Dalam antrian juga ada istilah lain yaitu LIFO(Last In First Out) dimana yang duluan masuk akan terakhir keluar dari antrian.

Implementasi dari queue juga dapat dilakukan dengan membuat linked list, namun untuk bahasa pemograman seperti C++ sudah ada class yang menampung data layaknya queue

B. Implementasi



C. Contoh program

```
#include<iostream>
#include<queue>
using namespace std;

int main()
{
	queue<int> q;
	int input;
	//Hentikan input dengan ctrl+z atau tekan huruf
	while(cin>>input)
	{
		q.push(input);
	}
	do
	{
		cout<<q.front()<<" ";
		q.pop();
	} while (q.size() != 0);
	cout<<endl;
	return 0;
}
```

III.Tugas

1. Buatlah program yang dapat mengecek tanda kurung sejajar atau tidak

   Format input :

   Satu baris string yang berisi huruf, angka, simbol '+','-','*','/' dan tanda kurung '(','{', dan '['

   Format output :

   "Ya" untuk string tersebut sejajar, "Tidak" untuk tidak.

   Contoh input :

   ```
   (tanda kurung)
   ```

   Contoh output :

   ```
   Ya
   ```

   Contoh input 2:

   ```
   (tanda (kurung)
   ```

   Contoh output 2:

   ```
   Tidak
   ```

   Contoh input 2:

   ```
   {(2*5) + (4-2)} + {4 – 2 * [2 + 1]}
   ```

   Contoh output 2:

   ```
   Ya
   ```

2. Buatlah program yang dapat mengecek tanda kurung sejajar atau tidak

MODUL VI
SENARAI BERANTAI (LINKED LIST)

I. Tujuan

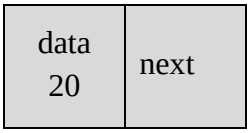
1. Praktikan dapat memahami pengertian linked list, gunanya dan dapat mengimplementasikannya dalam pemrograman.
2. Praktikan dapat mengidentifikasi permasalahan-permasalahan pemrograman yang harus diselesaikan dengan menggunakan linked list, sekaligus menyelesaikannya.

II. Teori

Linked list merupakan suatu struktur data yang membentuk suatu untaian yang saling berhubungan. Tiap untaian tersebut diletakkan pada memory. Tempat yang disediakan pada suatu area memori tertentu untuk menyimpan data dikenal dengan sebutan Node/Simpul. Linked list juga disebut dengan senarai berantai merupakan suatu variabel pointer yang simpulnya bertipe Record.

1. Senarai Berantai Tunggal (Single Linked List)

Single Linked List terdiri dari elemen-elemen individu, dimana masing-masing dihubungkan dengan pointer tunggal. Masing-masing elemen terdiri dari dua bagian, yaitu sebuah data dan sebuah pointer yang disebut dengan pointer next.

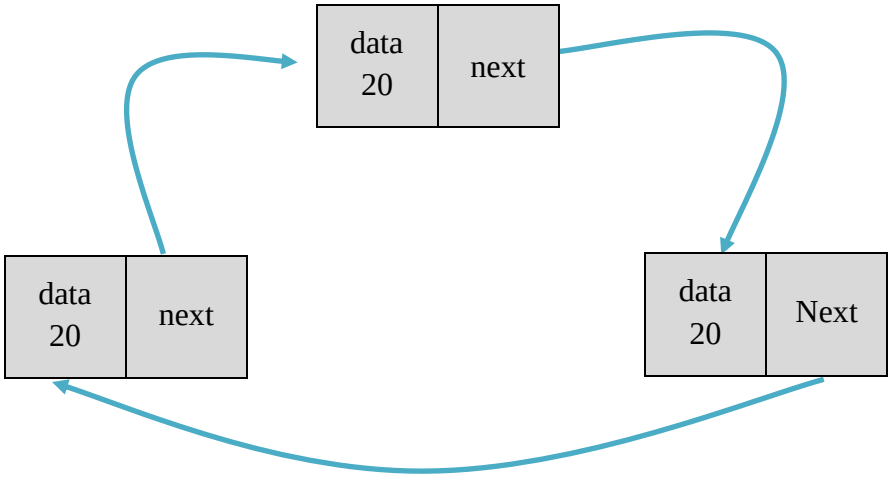



```
//Membuat struktur node
class Node
{
      int data;
      Node* next;

      Node(int n)
      {
            data = n;
            next = NULL;
      }
}

int main()
{
      //buat node kepala dengan nilai 10
      Node *head = new Node(10);
}
```

Single linked list terbagi atas dua jenis yaitu Single Linked List Circular dan Single Linked List non Circular.

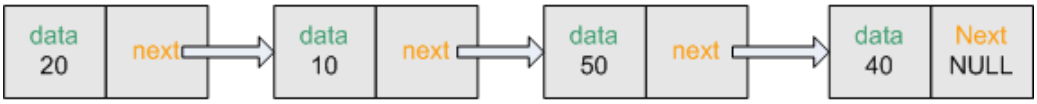
A. Single Linked List Circular

Adalah pointer pada node terakhir terakhir dalam list menunjuk kepada node awal , seakan-akan seperti sebuah lingkaran. Berikut ini adalah ilustrasi dari Single Linked List Circular .



B. Single Linked List NonCircular

Adalah pointer pada node terakhir terakhir dalam list tidak menunjuk kepada lain , sehingga bernilai NULL. Berikut ini adalah ilustrasi dari Single Linked List NonCircular .



Linked list

Elemen pada awal suatu list disebut head, dan elemen terakhir dari suatu list disebut tail. Untuk mengakses elemen dalam linked list, dimulai dari head dan menggunakan pointer next dari elemen selanjutnya untuk berpindah dari elemen ke elemen berikutnya sampai elemen yang diminta dicapai.

Ada beberapa hal yang harus diketahui mengenai linked list, diantaranya adalah :

1. Linked list selalu memiliki pointer petunjuk yang selalu menunjuk pada awal dari list yang disebut Head.
2. Linked list juga selalu memiliki pointer petunjuk menunjuk pada akhir dari list yang disebut Tail, kecuali untuk jenis circular.
3. Setiap simpul yang terbentuk selalu memiliki nilai NULL, kecuali jika simpul tersebut sudah ditunjuk oleh simpul yang lainnya (Linked list belum terhubung).
4. Posisi simpul terakhir pada linked list selalu bernilai NULL karena ia tidak menunjuk pada simpul yang lainnya, kecuali bentuk circular.

Dalam pembuatan Single Linked List dapat menggunakan 2 metode, yaitu:

- **LIFO (Last In First Out), aplikasinya : Stack (Tumpukan)**

  LIFO adalah suatu metode pembuatan Linked List dimana data yang masuk paling akhir adalah data yang keluar paling awal.

- **FIFO (First In First Out), aplikasinya : Queue (Antrian)**

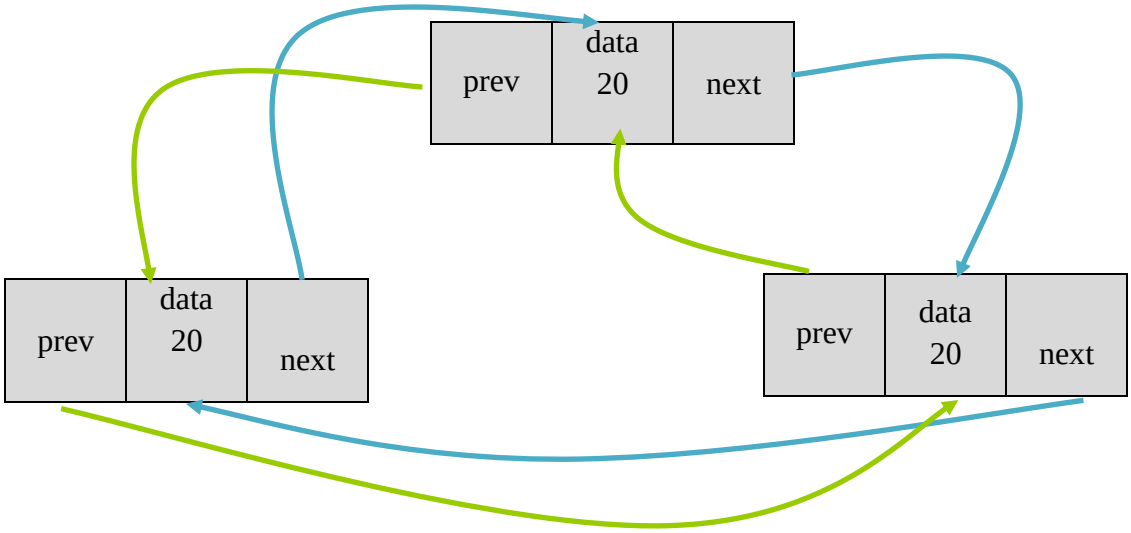
  FIFO adalah suatu metode pembuatan Linked List dimana data yang masuk paling awal adalah data yang keluar paling awal juga.

2. Senarai Berantai Ganda (Double Linked List)

Secara garis besar Double linked list adalah linked list yang memiliki dua buah pointer yang menunjuk ke simpul sebelumnya (Prev) dan yang menunjuk ke simpul sesudahnya (Next). Seperti halnya diatas, double linked list juga memiliki 2 jenis yaitu Double Linked List Circular dan Double Linked List NonCircular.

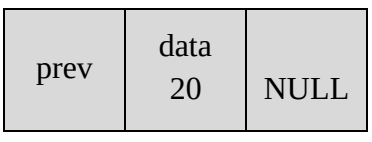
A. Double Linked List Circular

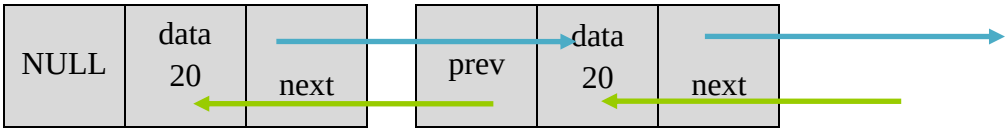
Pada double linked list circular, pointer next pada node terakhir menunjuk kepada node awal dan pointer prev pada node awal menunjuk kepada node akhir dalam rangkaian node-node tersebut.



B. Double Linked List NonCircular

Untuk menunjukkan *head* dari *double linked list circular*, maka pointer *prev* dari node pertama menunjuk NULL dan untuk menunjukkan *tail* dari *double linked list* tersebut, maka pointer *next* dari node terakhir menunjuk NULL. Gambar berikut menunjukkan gambaran Double Linked list Non Circular .



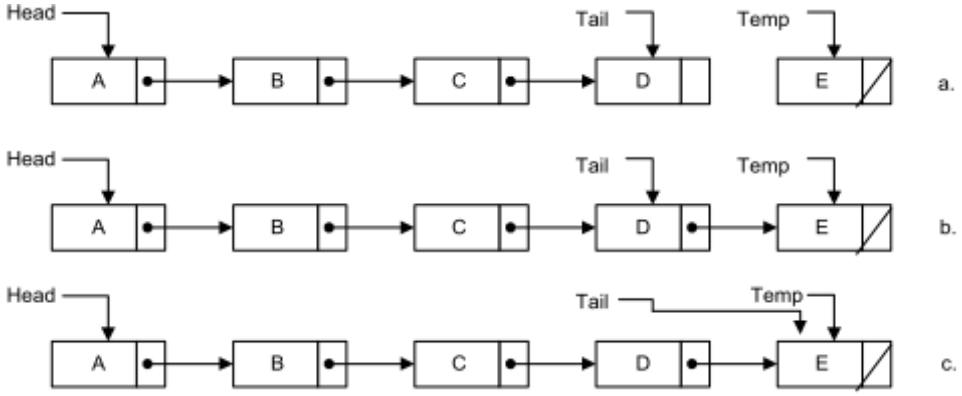


Ada beberapa hal yang harus diketahui mengenai Double linked list, diantaranya adalah :

1. Double Linked list selalu memiliki pointer petunjuk yang selalu menunjuk pada awal dari list yang disebut Head.
2. Double Linked list juga selalu memiliki pointer petunjuk menunjuk pada akhir dari list yang disebut Tail.
3. Setiap simpul yang terbentuk selalu memiliki nilai NIL, kecuali jika simpul tersebut sudah ditunjuk oleh simpul yang lainnya (Double Linked list belum terhubung).
4. Posisi simpul terakhir pada Doube linked list selalu bernilai NIL karena ia tidak menunjuk pada simpul yang lainnya, kecuali bentuk circular.

3. Operasi Linked List dan Beberapa Implementasi

a. Menambah Simpul (di Depan, Belakang dan Tengah).



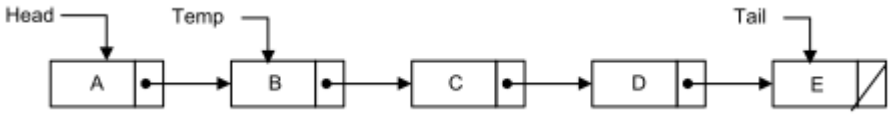
Ilustrasi Penambahan Simpul di Posisi Tail

```
//Fungsi menambah simpul di ekor single linked list
void insertTail(Node** head,int n)
{
      //jika belum ada head
      if (*head==NULL)
      {
            *head = new Node(n);
            return;
      }
      //buat node baru
      Node* temp = new Node(n);
      //cari tail
      Node* tail = *head;
      while(tail->next!=NULL) tail = tail->next;
      //mengarahkan ekor ke node baru
      tail->next = temp;
}
```

b. Menghapus Simpul (di Depan, Belakang dan Tengah).

```
//Fungsi menambah simpul di kepala single linked list
void deleteHead(Node** head)
{
      if (*head==NULL)
      {
            cout<<"Head tidak boleh NULL";
            return;
      }
      //simpan dahulu head yang dihapus
      Node* temp = *head;
      //pindahkan head ke yang baru
      *head = temp->next;
      //hapus head sebelumnya
      delete temp;
}
```

c. Membaca isi linked list (Membaca maju dan mundur).



Ilustrasi Pembacaan dari Head ke Tail

```
//mencetak isi di single linked list
void print(Node* head)
{
      if (head==NULL)
      {
            cout<<"linked list kosong"<<endl;
            return;
      }
      Node* temp = head;
      while(temp!=NULL)
      {
            cout<<(temp->data)<<" ";
            temp = temp->next;
      }
      cout<<endl;
}
```

4. Latihan

```
int main()
{
	Node* head = NULL;
	string input;
	while (cin>>input)
	{
		if (input=="insert")
		{
			int data;
			cin>>data;
			insertTail(&head,data);
			print(head);
		}
		else if (input=="delete")
		{
			deleteHead(&head);
			print(head);
		}
		else if (input=="print")
		{
			print(head);
		}
	}
	return 0;
}
```

5. Tugas

1. Buatlah single linked list dengan operasi seperti stack (push, pop, dll)
2. Buatlah Sebuah program untuk menampilkan deret angka yang tampak bergeser (Membaca dari titik yang ditentukan)
   Contoh :
   Data : 1 2 3 4 5 Pergeseran : 3
   Hasil : 4 5 1 2 3
3. Buatlah Sebuah program double linked list dengan kriteria menu sebagai berikut :
   a. Insert data dari belakang.
   b. Insert data ke indeks yang diberikan
      Contoh :
      List : 1 2 3 4 5 . Hapus indeks 3
      Hasil : 1 2 3 5
   c. Hapus data dari indeks yang diberikan
   d. Tampilkan data dari indeks yang diberikan.
4. Buatlah implementasi insertion sort menggunakan linked list
   Contoh input :
   4 9 2 10 5 8 3
   Contoh output:

2 3 4 5 8 9 10

MODUL VII
TREE

I. Tujuan

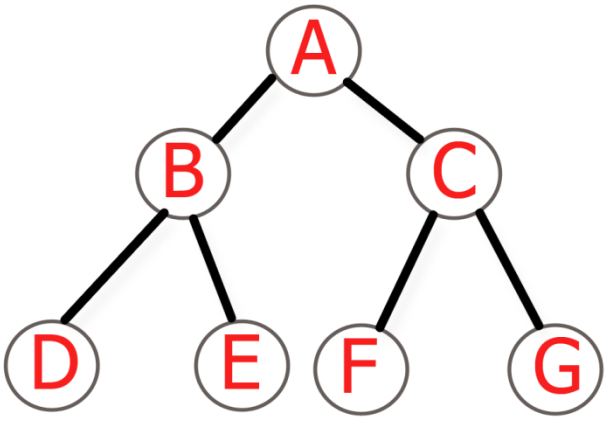
1. Praktikan dapat mengimplementasikan konsep linked list pada Tree .
2. Praktikan dapat memahami konsep Binary Tree.

II. Teori

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis(hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan elemen lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lain yang disebut dengan Subtree.

Istilah-istilah di dalam tree:

Predecessor	: Node yang berada diatas node tertentu
Successor	: Node yang dibawah node tertentu
Ancestor	: Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendant	: Seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama.
Parent	: Predecessor satu level diatas suatu node
Child	: Successor satu level dibawah satu node
Sibling	: Node-node yang memiliki parent yang sama dengan suatu node
Subtree	: Bagian dari tree yang berupa suatu node beserta descendantnya
Size	: Banyaknya node dalam suatu tree
Height	: Banyaknya tingkatan/level dalam suatu tree
Root	: Satu-satunya node khusus dalam tree yang tidak memiliki successor
Degree	: Banyaknya child yang dimiliki suatu node.



Ancestor	: A
Descendant	: F,G
Parent	: B
Child	: B,C
Sibling	: G
Size	: 7
Height	: 3
Root	: A
Leaf	: D,E,F,G
Degree	: 2

Jenis-jenis Tree :

1. Binary Tree

Tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah.

Jenis-jenis Binary Tree :

a. Full Binary Tree
   Binary Tree yang tiap nodenya memiliki dua child dan tiap subtree harus mempunyai panjang path yang sama.
b. Complete Binary Tree
   Mirip dengan Full Binary Tree, namun tiap subtree boleh memiliki panjang path yang berbeda, kecuali leaf memiliki 0 atau 2 child.
c. Skewed Binary Tree
   Binary Tree yang semua nodenya hanya memiliki satu child.

2. Binary Search Tree

Binary Tree dengan sifat bahwa semua right child harus lebih besar dari left child serta parentnya. Binary Search Tree dibuat untuk mengatasi kelemahan pada Binary Tree biasa, yaitu kesulitan dalam searching/pencarian node tertentu.

Algoritma Binary Search Tree :

- Data diambil dari posisi 1 sampai posisi akhir n
- Kemudian cari posisi data tengah dengan rumus: (posisi awal + posisi akhir) / 2
- Kemudian data yang dicari dibandingkan dengan data yang di tengah, apakah sama atau lebih kecil, atau lebih besar?
- Jika lebih besar, maka proses pencarian dicari dengan posisi awal adalah posisi tengah + 1
- Jika lebih kecil, maka proses pencarian dicari dengan posisi akhir adalah posisi tengah – 1
- Jika data sama, berarti ketemu.

```
#include <iostream>
#include <conio.h>
#include <iomanip>
using namespace std;

int data[7] = {1, 8, 2, 5, 4, 9, 7};
int cari;

void selection_sort()
{
      int temp, min, i, j;
      for(i=0; i<7;i++)
      {
            min = i;
            for(j = i+1; j<7; j++)
            {
                  if(data[j]<data[min])
                  {
                        min=j;
                  }
            }
```

```
            temp = data[i];
                  data[i]  = data[min];
                  data[min] = temp;
            }
      }
      void binarysearch()
      {
      //searching
      int awal, akhir, tengah, b_flag = 0;
      awal = 0;
      akhir = 7;
      while (b_flag == 0 && awal<=akhir)
      {
            tengah = (awal + akhir)/2;
            if(data[tengah] == cari)
            {
                  b_flag = 1;
                  break;
            }
            else if(data[tengah]<cari)
                  awal = tengah + 1;
            else
                  akhir = tengah -1;
      }
      if(b_flag == 1)
            cout<<"\nData ditemukan pada index ke-"<<tengah<<endl;
      else
            cout<<"\nData tidak ditemukan\n";
}

int main()
{
      cout<<"\t   'BINARY SEARCH'"<<endl;
      cout<<"\t====================="<<endl;
      cout<<"\nData           : ";
      //tampilkan data awal
      for(int x = 0; x<7; x++)
            cout<<setw(3)<<data[x];
      cout<<endl;
      cout<<"\nMasukkan data yang ingin Anda cari : ";
      cin>>cari;
      cout<<"\nData diurutkan : ";
      //urutkan data dengan selection sort
      selection_sort();
      //tampilkan data setelah diurutkan
      for(int x = 0; x<7;x++)
            cout<<setw(3)<<data[x];
      cout<<endl;
      binarysearch();
      getch();
 }
```

3. AVL Tree

Binary Search Tree yang memiliki perbedaan tingkat tinggi/level antara subtree kiri dan subtree kanan maksimal adalah 1.

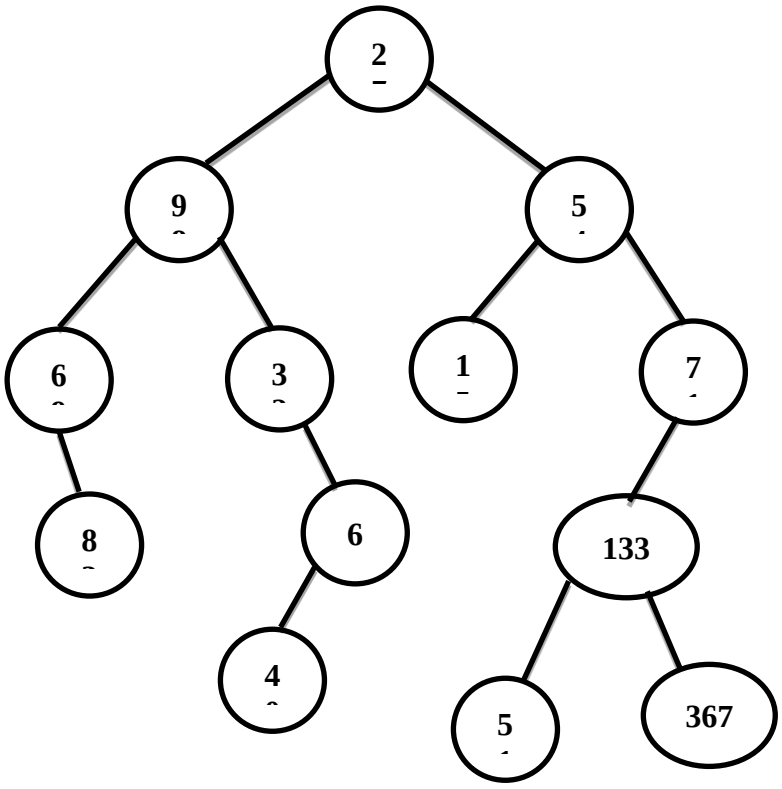
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct Simpul {
    char INFO;
    struct Simpul *LEFT, *RIGHT;
}
*CURR, *BARU, *BANTU, *ROOT, *Q[129];
char X;
```

```
void Inisialisasi() {
    ROOT = NULL;
    BARU = NULL;
}

void buatSimpul(char X) {
    BARU = new Simpul;
    BARU -> INFO = X;
    BARU -> LEFT = NULL;
    BARU -> RIGHT = NULL;
}

void buatSimpulAkar() {
    ROOT = BARU;
}

void preOrder(Simpul *T){
    if(T != NULL){
        printf("%c ", T -> INFO);
        preOrder(T -> LEFT);
        preOrder(T -> RIGHT);
    }
}

void inOrder(Simpul *T) {
    if(T != NULL) {
        inOrder(T -> LEFT);
        printf("%c ", T -> INFO);
        inOrder(T -> RIGHT);
    }
}


void postOrder(Simpul *T) {
    if(T != NULL){
        postOrder(T -> LEFT);
        postOrder(T -> RIGHT);
        printf("%c ", T -> INFO);
    }
}
```

```
main() {
    Inisialisasi();
    buatSimpul('A'); buatSimpulAkar();
    buatSimpul('B'); ROOT -> LEFT = BARU;
    buatSimpul('C'); ROOT -> RIGHT = BARU;
    buatSimpul('G'); ROOT -> RIGHT -> RIGHT = BARU;
    buatSimpul('E'); ROOT -> LEFT -> RIGHT = BARU;
    buatSimpul('F'); ROOT -> RIGHT -> LEFT = BARU;
    buatSimpul('L'); ROOT -> RIGHT -> LEFT -> LEFT = BARU;
    buatSimpul('M'); ROOT -> RIGHT -> LEFT -> RIGHT = BARU;
    printf("PreOrder  : ");
    preOrder(ROOT);
    printf("\n\nInOrder   : ");
    inOrder(ROOT);
    printf("\n\nPostOrder : ");
    postOrder(ROOT);
    getch();
}
```

III. Tugas

1. Buatlah Program untuk struktur pohon berikut ini secara **PreOrder**, **InOrder** dan **PostOrder**:



2. Bentuk program Binary Search Tree dari data berikut 18, 2, 57, 45, 99, 120, 34, 73 dan 61

MODUL VIII
GRAPH

I. Tujuan

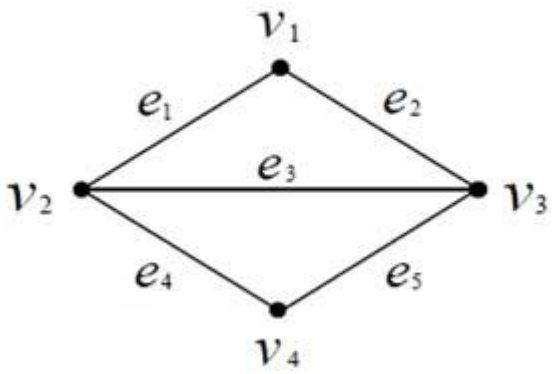
1. Praktikan dapat mengetahui apa itu Graph dan penggunaanya pada struktur data.
2. Praktikan dapat mengimplementasikan bentuk data Graph dengan suatu bahasa pemrograman.

II. Teori

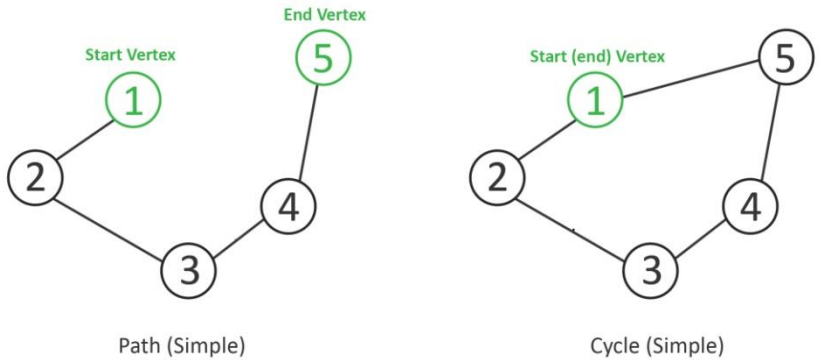
Graf adalah kumpulan noktah (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan garis (sisi). Graph dapat digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari graph adalah dengan menyatakan objek sebagai noktah, bulatan atau titik (Vertex), sedangkan hubungan antara objek dinyatakan dengan garis (Edge). Biasanya untuk suatu graph G digunakan notasi matematis.

G = (V . E)

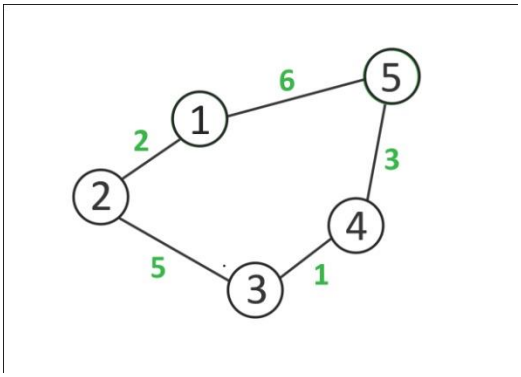
Dimana : G = Graph
V = Simpul atau Vertex, atau Node, atau Titik
E = Busur atau Edge, atau arc



Urutan dari beberapa vertex sehingga terdapat beberapa edge ke setiap vertex yang berurutan disebut Path. Vertex pertama disebut start vertex (vertex awal), vertex terakhir disebut end vertex (vertex akhir). Jika start vertex dan end vertex sama, maka Path disebut Cycle. Path disebut sederhana jika setiap vertex terhubung hanya sekali. Cycle disebut sederhana jika setiap vertex kecuali start vertex terhubung hanya sekali.



Graph berbobot adalah graph yang setiap edgenya mempunyai nilai-nilai real yang disebut bobot edge. Misalnya dalam contoh jaringan atau jalan antar kota, berat masing-masing jalan mungkin panjang atau minimal waktu yang dibutuhkan dalam perjalanan.



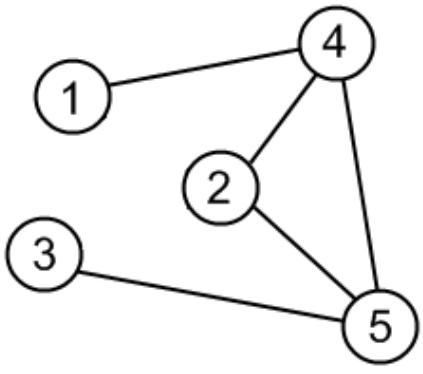
Representasi graph

Banyak cara merepresentasikan graph kedalam komputer. Kali ini kita akan membahas dua diantaranya, adjacency matrix dan adjacency list.

1. Adjacency Matrix

   Pada metode ini graph direpresentasikan dengan matrix dua dimensi, dimana baris menggambarkan vertex dan kolom menggambarkan vertex tujuan. Data pada setiap edge dan vertex disimpan terpisah, hanya nilai/bobot suatu edge yang dapat disimpan antara setiap pasangan vertex.

   Setiap sel a ij dari adjacency matriks mengandung 0, jika ada edge antara i-th dan j-th vertex maka sel a ij mengandung nilai 1.



Graph

	1	2	3	4	5
1	0	0	0	1	0
2	0	0	0	1	1
3	0	0	0	0	1
4	1	1	0	0	1
5	0	1	1	1	0

Adjacency matrix

Metode ini nyaman dan mudah digunakan. Untuk menambah, menghapus atau mengecek edge dapat diselesaikan dengan O(1) kali. Juga sangat mudah diimplementasikan ke program.

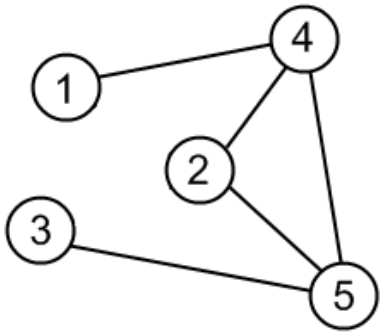
Tetapi metode ini membutuhakn memori yang besar untuk menyimpan graph yang besar. Terutama dense graph, yaitu graph yang mempunyai edge dua kali lipat lebih banyak dari vertex. Untuk mencari nilai dari suatu sel adjacency matriks juga membutuhkan pencarian perbaris, sehingga menimbulkan kompleksitas O (|V|).

```
#include <iostream>

using namespace std;

bool** adjacencyMatrix;
int vertexCount;

void Graph(){
    adjacencyMatrix = new bool*[vertexCount];
    for (int i = 0; i < vertexCount; i++) {
        adjacencyMatrix[i] = new bool[vertexCount];
        for (int j = 0; j < vertexCount; j++)
            adjacencyMatrix[i][j] = false; } }

void addEdge(int i, int j) {
    if (i >= 0 && i < vertexCount && j > 0 && j < vertexCount)
{
        adjacencyMatrix[i][j] = true;
        adjacencyMatrix[j][i] = true; } }

void removeEdge(int i, int j) {
    if (i >= 0 && i < vertexCount && j > 0 && j < vertexCount)
{
        adjacencyMatrix[i][j] = false;
        adjacencyMatrix[j][i] = false; } }

bool isEdge(int i, int j) {
    if (i >= 0 && i < vertexCount && j > 0 && j < vertexCount)
        return adjacencyMatrix[i][j];
    else
        return false; }

void DeleteGraph() {
    for (int i = 0; i < vertexCount; i++)
        delete[] adjacencyMatrix[i];
    delete[] adjacencyMatrix; }
```

```
main()
{
      vertexCount = 5;
      Graph();
      addEdge(0,1);
      addEdge(1,2);
      addEdge(2,3);
      addEdge(3,4);
      cout<<"Apakah Verteks 1 dan 2 terhubung ? "<<isEdge(1,2);
      cout<<"\nApakah Verteks 2 dan 4 terhubung ?
      "<<isEdge(2,4);
}
```

2. Adjacency List

Pada representasi ini, vertex disimpan sebagai record atau objek, dan setiap vertex menyimpan daftar simpul yang berdekatan atau beradjacent. Struktur data ini memungkinkan penyimpanan data tambahan pada vertex. Data tambahan ini dapat disimpan jika edge-edge juga disimpan sebagai objek, dimana dalam setiap vertex menyimpan incident edges dan tiap edge meyimpan incident vertex.



Graph

1 4
2 4 5
3 5
4 1 2 5
5 2 3 4

Adjacency list

Metode ini membuat kita dapat menyimpan graph dalam bentuk yang lebih kompleks dari adjacency matriks, tetapi graph menjadi semakin berubah ke denser graph. Selain itu kita bisa mendapatkan adjancent vertex dalam waktu O(1) kali.

Tetapi dalam dalam hal merepresentasikan ke komputer lumayan sulit. Untuk menambahkan dan mengurangkan adjacent list ataupun edge membutuhkan rata-rata O(|E|/|V|) kali yang dapat menambahkan kompleksitas cubic pada setiap edge.

Singkatnya adjacency list adalah solusi yang baik untuk graph yang tidak terlalu padat dan mengefisienkan pengubahan jumlah vertex daripada adjacency matrix. Walaupun masih ada solusi yang lebih baik lagi untuk menyimpan graph secara lebih dinamis.

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

using namespace std;

struct AdjListNode //Struct untuk Verteks
{
    int dest;
    struct AdjListNode* next;
};

struct AdjList //Struct untuk Adjacent (verteks yg berhubungan)
{
    struct AdjListNode *head;
};

struct Graph //Struct graph
{
    int V;
    struct AdjList* array;
};

struct AdjListNode* newAdjListNode(int dest) {
//Struct untuk membuat kumpulan verteks2
    struct AdjListNode* newNode = (struct AdjListNode*)
malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int V) { // Membuat graph
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct
Graph));
    graph->V = V;
    graph->array = (struct AdjList*) malloc(V * sizeof(struct
AdjList));
    int i;
    for (i = 0; i < V; ++i)
        graph->array[i].head = NULL;
// Membuat head tiap verteks null, verteks belum berhubungan
    return graph;
}
```

```
void addEdge(struct Graph* graph, int src, int dest){// Membuat edge
    struct AdjListNode* newNode = newAdjListNode(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;
    newNode = newAdjListNode(src);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

void printGraph(struct Graph* graph) { //Print graph
    int v;
    for (v = 0; v < graph->V; ++v) {
        struct AdjListNode* pCrawl = graph->array[v].head;
        printf("\n Adjacency list of vertex %d\n head ", v);
        while (pCrawl) {
            printf("-> %d", pCrawl->dest);
            pCrawl = pCrawl->next; }
        printf("\n");
     }
}

main()
{
     struct Graph *graph = createGraph(5);
     addEdge(graph, 1, 2);
     addEdge(graph, 2, 3);
     addEdge(graph, 3, 4);
     printGraph(graph);
}
```

III. Tugas

1. Buatlah implementasi graph daerah tempat tinggal Anda minimal 10 verteks dengan bahasa c++ .