MODUL PRAKTIKUM STRUKTUR DATA



**Tim Penyusun: Dosen Teknik Informatika**

**JURUSAN TEKNIK INFORMATIKA FAKULTAS TEKNIK UNIVERSITAS PALANGKARAYA 2020**

**TATA TERTIB DAN TATA LAKSANA PRAKTIKUM JURUSAN/PROGRAM STUDI TEKNIK INFORMATIKA FAKULTAS TEKNIK UPR**

# TATA TERTIB

1. Praktikan **WAJIB** mengikuti semua modul praktikum.
2. Praktikan hanya boleh tidak mengikuti praktikum **1 (satu) kali DENGAN ATAU TANPA surat izin** dari jumlah pertemuan praktikum.
3. Praktikan yang berhalangan mengikuti praktikum, diwajibkan melaporkan ke asisten praktikum untuk menentukan jadwal praktikum sebagai pengganti jadwal yang berhalangan.
4. Praktikan yang lebih dari 1 (satu) kali tidak mengikuti praktikum, tidak diperbolehkan untuk mengikuti praktikum untuk modul-modul praktikum selanjutnya dan **NILAI AKHIR PRAKTIKUM** adalah **NOL**.
5. Praktikan wajib hadir 15 menit sebelum praktikum dimulai.
6. Tidak diperbolehkan saling bekerjasama.
7. Dilarang menggunakan kaos oblong dan sandal jepit selama praktikum. Bagi yang melanggarpoin ini, tidak diperbolehkan mengikuti praktikum.

# TATA LAKSANA

1. Sebelum praktikum dimulai, setiap praktikan wajib mengumpulkan **LAPORAN RENCANA PRAKTIKUM (Tulis Tangan)** dan **LAPORAN HASIL PRAKTIKUM** modul sebelumnya.
2. Jika praktikan tidak melaksanakan tata laksana poin 1, maka **TIDAK DIPERBOLEHKAN MENGIKUTI PRAKTIKUM**.
3. Setiap modul praktikum, akan dilakukan ***Pre-test***.
4. Pada saat praktikum hanya membawa alat tulis, modul dan Laporan. HP, Laptop & USB di simpan dan tas letakan didepan.
5. Pada saat praktikum tidak diijinkan mengakses Internet.
6. Format laporanmeliputi:

# Laporan Rencana Praktikum :

* + Format A4
  + Tulis Tangan.
  + Tujuan Praktikum dari modul yang akan dilaksanakan
  + Hal-hal yang akan dilakukan selama praktikum.

# Laporan Hasil Praktikum :

* + Halaman Depan
  + Bab I. Tujuandan Landasan Teori
  + Bab II. Pembahasan
  + Bab III. Kesimpulan
  + Daftar Pustaka
  + Lampiran (disertai laporan rencana praktikum)

1. Format Penulisan Laporan Hasil Praktikum
   * Spasi : 1,5
   * Font : Times New Roman
   * Font Size 12
   * Margins : Top 3, Left 4, Right 3, Bottom 4
   * Kertas : A4
2. Penilaian Laporan Hasil Praktikum :
   * Bab I. Tujuan dan LandasanTeori Nilai 10
   * Bab II. Pembahasan Nilai 60
   * Bab III. Kesimpulan Nilai 20
   * Daftar Pustaka Nilai 5
   * Lampiran Nilai 5

# Total 100

1. Praktikan yang mengabaikan format penulisan poin 5, nilai akan dikurangi 5 setiap kesalahan.
2. PenilaianAkhirPraktikum :
   * *Pre-Test* : 10%
   * Praktikum : 50%
   * LaporanPraktikum: 15%
   * Responsi : 25%

# Total 100%

1. PenilaianAkhir Mata Kuliah:
   * Kuliah, meliputi:

Tugas : 20%

UTS : 30%

Praktikum : 50%

# Total : 50%

* + UAS : 50%

# Total 100%

**MODUL I**

**ARRAY, STRUCT DAN POINTER**

1. **TUJUAN**
2. Mahasiswa mampu mengimplementasikan Array, Struct dan Pointer.
3. **LANDASAN TEORI**
4. **Array**

Array adalah suatu tipe data terstruktur yang berupa sejumlah data sejenis (bertipe data sama) yang jumlahnya tetap dan diberi suatu nama tertentu.

Array dapat berupa array 1 dimensi, 2 dimensi, bahkan n-dimensi.

* 1. Deklarasi

|  |
| --- |
| tipe\_data nama\_var [ukuran];  tipe\_data : tipe data elemen larik (int,char,float, dll)  nama\_var : menyatakan nama variabel yang dipakai.  ukuran : menunjukkan jumlah maksimal elemen larik. |

|  |
| --- |
| Contoh :  int nilai[6]; |

* 1. Inisialisasi

Menginisialisasi array sama dengan memberikan nilai awal array pada saat didefinisikan.

|  |
| --- |
| int nilai[5] = {8,7,5,6,4,3};  //atau  int a [2;  a [0] = 10;  a [1] = 3;  a [2] = 1; |

* 1. Pengaksesan

Pengisian dan pengambilan nilai pada indeks tertentu dapat dilakukan dengan mengeset nilai atau menampilkan nilai pada indeks yang dimaksud. Pengaksesan elemen array dapat dilakukan berurutan atau random berdasarkan indeks tertentu secara langsung.

|  |
| --- |
| #include <iostream>  int main (){  int billy [] = {16, 2, 77, 40, 12071};  int n, result=0;  for ( n=0 ; n<5 ; n++ ) {  cout<<”cetak dengan loop : “<<billiy[n];  }  cout<<”cetak nilai indeks 0 : “<<billiy[0];  } |

1. **Struct**

Struct adalah bentuk struktur data yang dapat menyimpan variabel-variabel dalam 1 nama, namun memiliki tipe data yang berbeda ataupun sama. Variable-variabel tersebut memiliki kaitan satu sama yang lain.

* 1. Deklarasi

Ada 2 cara pendeklarasian struct, yaitu :

|  |
| --- |
| typedef struct Mahasiswa { //cara pertama  char NIM[8];  char nama[50];  float ipk;  };//atau  struct { //cara kedua  char NIM[8];  char nama[50];  float ipk;  }mhs; |

Contoh program :

|  |
| --- |
| #include <iostream>  int main(){  struct orang{  char nama[40];  short umur;  }saya;  cout<<”nama : “; cin.getline(saya.nama,40);  cout<<”umur : “; cin>>saya.umur;  cout<<saya.nama<<” berumur “<<saya.umur;  } |

* 1. Array of Struct

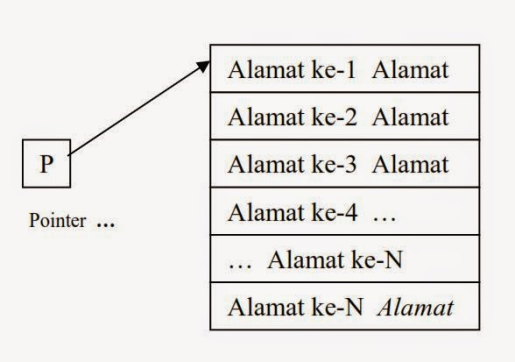
Apabila struct akan digunakan untuk beberapa kali, ada 2 cara :

|  |
| --- |
| #include <stdio.h>  typedef struct Mahasiswa {char NIM[8];  char NIM[8];  char nama[50  float ipk;  };  int main() {  Mahasiswa a,b,c; //cara pertama  //atau  Mahasiswa mhs[3]; //cara kedua  ……  } |

1. **Pointer**

Pointer adalah penunjuk suatu variabel. Karena menunjuk suatu variabel, maka pointer wajib memiliki alamat dari variabel yang ditunjuknya. Dengan mekanisme pointer, suatu variabel dalam suatu fungsi dapat diakses oleh fungsi yang lain.

Sebagai analogi, sebuah nomor halaman dalam buku dapat dianggap sebagai pointer ke halaman yang sesuai; dereferencing seperti pointer akan dilakukan dengan membalik ke halaman dengan nomor halaman yang diberikan.



Gambar 1.1 Ilustrasi Pointer

Ada dua operator pada pointer yaitu operator deference dan reference.

* 1. Deference (&)

Deference dengan simbol & (ampersand) merupakan suatu operator yang berfungsi untuk menanyakan alamat dari suatu variabel. Memberikan simbol & pada awal variabel dan mencetak hasilnya pada jendela CLI, maka yang akan tercetak adalah alamat dari variabel tersebut bukan nilai yang ditampung oleh variabel tersebut. Berikut contoh programnya:

|  |
| --- |
| #include <iostream>  int main(){  int a = 5;  cout<<"Alamat Variabel a adalah :"<<&a<<endl;  cout<<"Nilai Variabel a adalah :"<<a<<endl;  return 0;  } |

* 1. Reference (\*)

Reference dengan simbol \* (asterisk) merupakan suatu operator yang berfungsi menyatakan suatu variabel adalah variabel pointer. Sama halnya dengan operator deference, peletakan simbol operator reference diletakan diawal variabel. Operator reference ini akan membuat suatu variabel pointer untuk menampung alamat. Perhatikan contoh berikut:

|  |
| --- |
| #include <iostream>  int main(){  int a=5; //Memberikan nilai 5 pada variabel a  int \*b; //Mendeklarasikan variabel b sebagai pointer  b = &a; //Mengkopikan alamat a kedalam pointer b  cout<<"Nilai variabel a adalah "<<a<<endl;  cout<<"Alamat variabel a adalah "<<&a<<endl;  cout<<"Isi dari variabel b adalah "<<b<<endl;  cout<<"Nilai yang tertampung dalam variabel b adalah "<<\*b<<endl;  return 0;  } |

Ada beberapa cara atau pengoperasian pointer yang dapat dilakukan, diantaranya :

1. Pointer dan array

Pointer dapat digunakan sebagai penunjuk dari nilai index yang akan digunakan pada sebuah array, bahkan pointer itu sendiri dapat di berikan index sama seperti array.

|  |
| --- |
| #include <iostream>  int main() {  int nilaiArray[3] = {1,2,3};  int \*pointerArray;  pointerArray = nilaiArray;  for (int i = 0; i<3; i++){  cout<<”index [“ << i << ”] = “;  cout<<\*(pointerArray+i)<<endl;  }  } |

1. Pointer dan Struct

|  |
| --- |
| #include <iostream>  struct mhs{  int nim;  string nama;  };  int main() {  mhs m;  mhs \*p = &m;  //struct biasa  m.nama = "Anne";  cout<<m.nama<<endl;  //struct pointer  p->nim = 123;  cout<<p->nim;  } |

1. Pointer sebagai paramater suatu fungsi

Pointer dapat juga digunakan sebagai parameter pada sebuah fungsi. Parameter ini akan bersifat by reference yang artinya apa yang dikirimkan sebagai parameter adalah alamat variabelnya, bukan nilainya. Jadi perubahan nilai yang terjadi didalam fungsi akan mengubah juga dari variabel alamat aslinya.

|  |
| --- |
| #include <iostream>  void tambah(int \*angka){  \*angka +=20;  }  int main() {  int nilai = 10;  cout<<”Nilai = “<<nilai<<endl;  tambah(&nilai);  cout<<”Nilai = “<<nilai<<endl;  return 0;  } |

1. **TUGAS PRAKTIKUM**
2. Buatlah sebuah program untuk memasukan NIM, nama, email mahasiswa menggunakan Array Of Struct sebanyak N inputan. Selain itu, buatlah juga program untuk menampilkan data-data yang telah dimasukkan tadi. Gunakan *looping* untuk menambahkan dan memasukkan datanya!
3. Cobalah source code dibawah ini! Perbaikilah jika ada kesalahan, sehingga program mampu menampilkan output deretan bilangan fibonacci dengan benar. Jelaskan kedalam laporan anda.

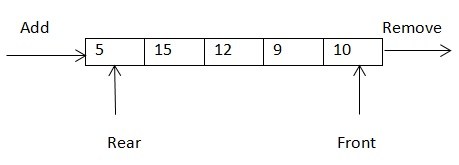
|  |
| --- |
| #include <iostream>  #define max 50  using namespace std;  int\* fibo(int n){  int arr[max];  arr[0] = 1;  arr[1] = 1;  for (int i = 2; i < n; i++){  arr[i] = arr[i - 1] + arr[i - 2];  }  return arr;  }  int main(){  int num, \*ptr;  cout << "jumlah bilangan fibonacci : ";  cin >> num;  ptr = fibo(num);  for (int i = 0; i < num; i++){  cout << (ptr + i) << " ";  }  } |

**MODUL II**

**QUEUE DAN STACK**

1. **TUJUAN**
2. Mahasiswa memahami konsep dari Queue dan Stack.
3. Mahasiswa mampu mengimplementasikan Queue dan Stack.
4. **LANDASAN TEORI**
5. **Queue**

Queue (antrian) dapat diartikan sebagai suatu kumpulan data yang seolah-olah terlihat seperti ada data yang diletakkan di sebelah data yang lain seperti pada gambar 1, dimana data masuk melalui lorong di sebelah kanan dan masuk dari terowongan sebelah kiri. Hal ini membuat antrian bersifat FIFO (First In First Out), beda dengan stack yang berciri LIFO.

Gambar 2.1 Ilustrasi Queue.

Contoh yang paling populer untuk membayangkan sebuah queue adalah antrian pada kasir sebuah bank. Ketika seorang pelanggan datang, akan menuju ke belakang dari antrian. Setelah pelanggan dilayani, antrian yang berada di depan akan maju. Pada saat menempatkan elemen pada ujung (tail) dari queue disebut dengan enqueue, pada saat memindahkan elemen dari kepala (head) sebuah queue disebut dengan dequeue. Pada gambar 2 diperlihatkan sebuah queue serta proses enqueue dan dequeue.

Karakteristik penting dari antrian adalah :

* + - 1. Elemen, antrian yaitu item-item data yang terdapat didalam antrian.
      2. Front (elemen terdepan dari antrian).
      3. Rear (elemen terakhir dari antrian).
      4. Count (jumlah elemen pada antrian).

Operasi yang dapat dilakukan pada queue pada umunya terdapat 6 operasi yaitu *create*, *initialization*, *isEmpty*, *isFull*, *clear*, *enqueue* dan *dequeue*. Namun hanya tiga operasi utama pada queue yang akan dijelaskan, yaitu :

Create

Untuk menciptakan queue menggunakan perintah struct dan menginisialisasi Queue dengan cara membuat Head dan Tail = -1. Beberapa orang memulai index queue dari -1 dan ada juga yang dari 0.

|  |
| --- |
| #ddefine MAX 8  Stsruct Queue{  Int elemen[MAX}  Int front,rear,count;  }; |

Enqueue

Untuk menambahkan elemen ke dalam Antrian, penambahan elemen selalu ditambahkan di elemen paling belakang. Penambahan elemen selalu menggerakan variabel Rear dengan cara increment counter Rear terlebih dahulu.

|  |
| --- |
| void Enqueue(int data, Queue \*Q){  Q->elemen[Q->rear] = data;  Q->rear++;  Q->count++;  } |

Dequeue

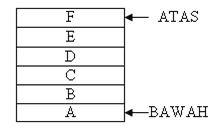
Digunakan untuk menghapus elemen terdepan/pertama (head) dari antrian dengan cara menggeser semua elemen antrian kedepan dan mengurangi Tail dengan 1 Penggeseran dilakukan dengan menggunakan looping.

|  |
| --- |
| void Dequeue(Queue \*Q){  int keeper = Q->elemen[Q->front];  for(int i = Q->front; i<Q->rear; i++){  Q->elemen[i]=Q->elemen[i+1];  }  Q->rear--;  Q->count--;  } |

1. **Stack**

Pengertian Stack pada Struktur Data adalah sebagai tumpukan dari benda, sekumpulan data yang seolah-olah diletakkan di atas data yang lain, koleksi dari objek-objek homogen, atau suatu urutan elemen yang elemennya dapat diambil dan ditambah hanya pada posisi akhir (top) saja.

Stack pada Struktur Data dapat diilustrasikan dengan dua buah kotak yang ditumpuk, kotak yang satu akan ditumpuk diatas kotak yang lainnya. Jika kemudian stack 2 kotak tadi, ditambah kotak ketiga, keempat, kelima, dan seterusnya, maka akan diperoleh sebuah stack kotak yang terdiri dari N kotak.



Gambar 2.2 Ilustrasi Stack.

Stack pada Struktur Data dapat diilustrasikan dengan dua buah kotak yang ditumpuk, kotak yang satu akan ditumpuk diatas kotak yang lainnya. Jika kemudian stack 2 kotak tadi, ditambah kotak ketiga, keempat, kelima, dan seterusnya, maka akan diperoleh sebuah stack kotak yang terdiri dari N kotak.

Stack bersifat LIFO (Last In First Out) artinya Benda yang terakhir masuk ke dalam stack akan menjadi yang pertama keluar dari stack. Operasi-operasi yang biasanya terdapat pada Stack mirip dengan queue. Adapun 2 operasi utama pada stack yaitu:

* + - 1. Push : digunakan untuk menambah item pada stack pada tumpukan paling atas.

|  |
| --- |
| void Push(int data, Queue \*Q){  Q->elemen[Q->top] = data;  Q->top++;  } |

* + - 1. Pop : digunakan untuk mengambil item pada stack pada tumpukan paling atas

|  |
| --- |
| void pop(Queue \*Q){  int keeper=Q->elemen[Q->top];  Q->top--;  } |

Cara mendefenisikan Stack dengan Array of Struct yaitu dengan menggunakan struct, stack hanya memiliki karateristik berupa top dan bottom sebagai penanda awal dan akhir serta elemennya itu sendiri. Definisikan konstanta MAX\_STACK untuk menyimpan maksimum isi stack kemudian buatlah variabel array data sebagai implementasi stack.

|  |
| --- |
| #define MAX\_STACK 10  struct Stack{  int elemen\_array[MAX\_STACK];  int top;  int bottom;  }; |

1. **TUGAS PRAKTIKUM**

Terdapat sebuah struct data dari list musik, seperti judul lagu, penyanyi, dsb. Buatlah sebuah program yang mampu melakukan *Dequeue (FIFO)* dan *Pop (LIFO)* pada struct data musik tersebut*.* Berikut adalah operasi/fungsi yang akan dibuat :

1. *Create*, sebuah procedure untuk membuat struct/record mahasiswa serta menginisialisasikan nilai awalnya.
2. *Input*, procedure untuk memasukan data mahasiswa.
3. *Dequeue/FIFO*, untuk melakukan pengambilan data secara FIFO.
4. *Pop/LIFO*, untuk melakukan pengambilan data secara LIFO.
5. *Clear*, untuk menghapus semua data didalam struct/record mahasiswa.
6. *Print*, mencetak seluruh nama mahasiswa.

**MODUL III**

**LINKED LIST**

1. **Tujuan**
2. Mahasiswa memahami struktur data linked list
3. Mahasiswa mampu menggunakan struktur data linked list dalam menyelesaikan masalah pemrograman
4. **Landasan Teori**

Linked List adalah suatu cara untuk menyimpan data dengan struktur sehingga programmer dapat secara otomatis menciptakan suatu tempat baru untuk menyimpan data kapan saja diperlukan. Linked list dikenal juga dengan sebutan senarai berantai adalah stuktur data yang terdiri dari urutan record data dimana setiap record memiliki field yang menyimpan alamat/referensi dari record selanjutnya (dalam urutan). Elemen data yang dihubungkan dengan link pada linked list disebut Node. Biasanya dalam suatu linked list, terdapat istilah head dan tail.

Jenis Linked List (yang akan dipelajari) adalah :

1. Single Linked List

2. Double Linked List

3. Circular Linked List

4. Multiple Linked List

Ada 5 proses dasar dalam Linked List :

1. **Proses Inisialisasi**

* Proses awal → Menyatakan Linked List belum ada
* Algoritma :

|  |
| --- |
| First = Null ;  Last = Null; |

1. **Proses Simpul Baru**

* Instruksi :

|  |
| --- |
| P = (simpul\*) malloc(sizeof(simpul)); |

* Algoritma :

|  |
| --- |
| void Buat\_Simpul (int x) {  P = (simpul\*) malloc(sizeof(simpul));  if (P!=NULL)  {P → Info=x;}  Else  cout << “simpul gagal dibuat”; } |

1. **Membuat Simpul Awal**

***Syarat :***

1. Linked List belum ada
2. Sudah ada simpul yang akan dijadikan simpul awal

* Algoritma :

|  |
| --- |
| void Awal () {  First = P;  Last =P;  P→Link = NULL; } |

1. **Menambahkan Simpul Baru ke dalam Linked List (INSERT)**

***Syarat :***

1. Linked List sudah ada.
2. Sudah ada simpul yang akan ditambahkan Linked List.
3. Insert Kanan/Akhir

Algoritma :

|  |
| --- |
| void Ins\_Akhir () {  Last → Link = P;  Last = P;  P → Link = NULL; } |

1. Insert Kiri/Awal

Algoritma :

|  |
| --- |
| void Ins\_Awal () {  P → Link = First;  First = P; } |

1. Insert Tengah

Algoritma :

|  |
| --- |
| void Ins\_Tengah () {  P → Link = Q → Link;  Q → Link = P; } |

1. **Menghapus Sebuah Simpul dari Linked List (DELETE)**

***Syarat :***

1. Linked List sudah ada.
2. Delete Kanan/Akhir

Algoritma :

|  |
| --- |
| void Del\_Akhir () {  Free (Last);  Last = Q;  Last → Link = NULL; } |

1. Delete Kiri/Awal

Algoritma :

|  |
| --- |
| void Del\_Awal () {  Q = First;  First = Q → Link;  Free (Q); } |

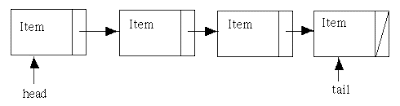
1. Delete Tengah

Algoritma :

|  |
| --- |
| void Del\_Tengah () {  R = Q → Link;  Q → Link = R → Link;  Free (R); } |

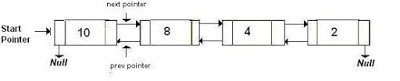
**Single Linked List**

Single Linked List merupakan suatu linked list yang hanya memiliki satu variabel pointer saja. Dimana pointer tersebut menunjuk ke node selanjutnya. Biasanya field pada tail menunjuk ke NULL.



**Double Linked List**

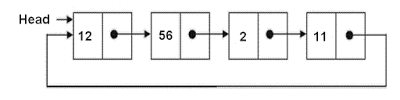
Double Linked List merupakan suatu linked list yang memiliki dua variabel pointer yaitu pointer yang menunjuk ke node selanjutnya dan pointer yang menunjuk ke node sebelumnya. Setiap head dan tailnya juga menunjuk ke NULL.



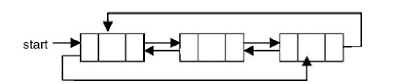
**Circular Linked List**

Circular Linked List merupakan suatu linked list dimana tail (node terakhir) menunjuk ke head (node pertama). Jadi tidak ada pointer yang menunjuk NULL. Ada 2 jenis Circular Linked List, yaitu :

* Circular Single Linked List

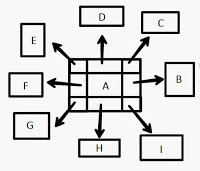


* Circular Double Linked List



**Multiple Linked List**

Multiple Linked List merupakan suatu linked list yang memiliki lebih dar 2 buat variabel pointer.



1. **Tugas Praktikum**

1. Buatlah sebuah linked list non circular yang berisi nim nama lengkap anda dan nim anda !

2. Buatlah sebuah program dengan menggunakan single linked list non circular

dengan fungsi-fungsi: (Buat dengan menggunakan menu).

• Menambah data(dari depan dan dari belakang)

• Menghapus data( dari depan dan dari belakang)

• Mencetak data

3. Buat sebuah program double linked list dengan fungsi-fungsi : (Buat dengan menggunkan menu).

• Menambahkan data (dari depan sesudah simpul dan dari belakang sesudah simpul)

• Menghapus data (dari tengah)

• Data yang dimasukkan langsung ke tampil dalam menu

**MODUL IV**

**BINARY TREE**

1. **TUJUAN PRAKTIKUM**
   1. Mempelajari variasi bagian-bagian dari tree sebagai suatu bentuk struktur tak linier.
   2. Mempelajari beberapa hubungan fakta yang direpresentasikan dalam sebuah tree, sehingga mampu merepresentasikan tree dalam permasalahan aslinya
   3. Memahami bagaimana menulis program untuk tree, dan bagaimana mengartikannya kembali dalam bentuk permasalahan aslinya

# LANDASAN TEORI

Pohon biner adalah sebuah tree yang pada masing-masing simpulnya hanya dapat memiliki maksimum 2 (dua) simpul anak. Tidak boleh lebih. Pada pohon biner, umumnya kedua node anak disebut dengan posisinya, yaitu **kiri** dan **kanan**. Beberapa istilah pada pohon biner:

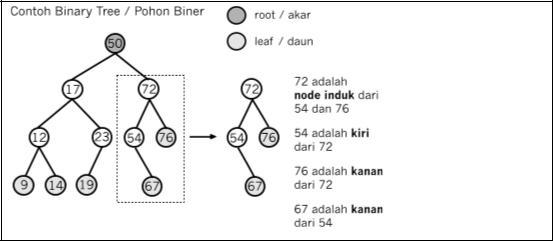
* Size (ukuran): jumlah total node yang terdapat pada pohon biner tersebut.
* Depth (kedalaman): panjang jalur yang menghubungkan sebuah node sampai ke node anaknya yang paling ujung (leaf). Depth biasa juga disebut height.

Full Binary Tree (Pohon Biner Penuh) adalah pohon biner yang setiap nodenya pasti memiliki 0 atau 2 node anak.

Perfect Binary Tree (Pohon Biner Sempurna) adalah pohon biner yang semua node leafnya berada pada kedalaman yang sama dari node root. Juga disebut sebagai Complete Binary Tree (Pohon Biner Lengkap).

Almost Complete Binary Tree (Pohon Biner Hampir Lengkap) adalah pohon biner yang setiap nodenya dapat memiliki 0 node anak, atau memiliki kiri, atau jika memiliki kanan harus memiliki kiri. Tidak boleh memiliki kanan saja.

Ilustrasi Binary Tree :



# Implementasi

Implementasi dalam pemrograman, dalam pokok bahasan ini akan dibicarakan untuk pohon biner saja. Asumsi awal adalah data yang hendak dimasukkan ke dalam node, bertipe data integer.

# Deklarasi Tree

Karena tree tersusun oleh node-node, maka yang perlu kita deklarasikan adalah komponen node itu sendiri. Dalam contoh dibawah, akan kita namai **Node**.

Sebelumnya perlu kita lihat bahwa untuk mewujudkan implementasi node ke dalam bahasa pemrograman, diperlukan sebuah struktur yang memiliki susunan berikut ini:



**typedef struct Node{**

**int data; Node \*kiri; Node \*kanan;**

**};**

Variabel data digunakan untuk menyimpan nilai angka node tersebut, sedangkan kiri dan kanan, bertipe pointer, masing-masing mewakili vektor yang akan menunjuk ke node anak kiri dan kanan.

# Inisialisasi Tree

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

**Node \*pohon; pohon = NULL;**

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

# Menambahkan Node Pada Tree

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

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

Proses penambahan ini diimplementasikan secara rekursif pada fungsi berikut:

**void tambah(Node \*\*root,int databaru){ if((\*root) == NULL){**

**Node \*baru;**

**baru = new Node;**

**baru->data = databaru; baru->kiri = NULL; baru->kanan = NULL; (\*root) = baru; (\*root)->kiri = NULL; (\*root)->kanan = NULL;**

**}**

**else if(databaru < (\*root)->data) tambah(&(\*root)->kiri,databaru);**

**else if(databaru > (\*root)->data) tambah(&(\*root)->kanan,databaru);**

**else if(databaru == (\*root)->data) printf("Data sudah ada!");**

**}**

Variabel **\*\*root** menunjukkan node mana yang sedang dicek saat ini, untuk itu saat pemanggilan fungsi ini, variabel **\*\*root** kita beri nilai pointer yang menunjuk ke node akar, yaitu **pohon**.

**tambah(&pohon,data);**

# Membaca dan Menampilkan Node Pada Tree

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

# Kunjungan Pre-Order.

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

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

**void preOrder(Node \*root){ if(root != NULL){**

**printf("%d ",root->data); preOrder(root->kiri); preOrder(root->kanan);**

**}**

**}**

# Kunjungan In-Order.

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

**void inOrder(Node \*root){ if(root != NULL){**

**inOrder(root->kiri); printf("%d ",root->data); inOrder(root->kanan);**

**}**

**}**

# Kunjungan Post-Order.

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

**void postOrder(Node \*root){ if(root != NULL){**

**postOrder(root->kiri); postOrder(root->kanan); printf("%d ",root->data);**

**}**

**}**

Variabel **\*\*root** pada setiap fungsi diatas menunjukkan node mana yang sedang dikunjungi saat ini, untuk itu saat pemanggilan, variabel **\*\*root** kita beri nilai pointer yang menunjuk ke node akar, yaitu **pohon**.

# Tugas Praktikum

1. Perbaikilah koding program binary tree dibawah

|  |
| --- |
| #include <iostream>  #include <conio.h>  using namespace std  struct \*node{  node \*kiri;  node \*kanan;  };  node \*akar=NULL;char data;  node \*addNode (node \*akar, char isi){  if(akar)==NULL){  node \*baru;  baru = new node;  Baru = data -> isi;  baru -> kiri = NULL;  baru -> kanan = NULL;  (akar) = baru  }  }  node \*preOrder (node akar){  if(akar != NULL){  preOrder(akar -> kanan);  cin << " " << akar -> data  preOrder(akar -> kiri);  }  }  node \*inOrder (node akar){  if(akar != NULL){  inOrder (akar -> kiri);  cout << " " << akar -> data;  inOrder ( akar = kanan);  }  }  node \*postOrder (node akar){  if (akar == NULL){  postOrder (akar -> kanan)  postOrder (akar -> kiri);    cout << " " << akar -> data;  }  }  main (){  int abjad;  cout << "\n\n\t Posisi Awal Tree : \n\n"  cout << "\t \A\n\t /\\\n\t B C\n\t /\n\t D\n\t /\\\n E F\n\n";  addnode(&akar, abjad = 'A');  addnode(&akar -> kanan, abjad = 'B');  addnode(&akar -> kiri, abjad = 'C');  addnode(&akar -> kanan -> kiri, abjad = 'D');  addnode(&akar -> kiri -> kanan -> kanan, abjad = 'E');  addnode(&akar -> kiri -> kanan -> kanan, abjad = 'F');  Cout "Tampilan PreOrder :  preOrder(akar);  cout << "\nTampilan InOrder :  inOrder(akar);  cout << "\nTampilan PostOrder :  postOrder(akar);  } |

1. Buatlah program dengan fungsi untuk menghitung jumlah node keseluruhan pada pohon biner. Penghitungan dilakukan dengan menjelajahi isi pohon, bukan dengan menambahkan counter saat setiap data baru dimasukkan!.
2. Buatlah program dengan fungsi untuk mencetak nilai node minimum (terkecil) pada pohon biner!

**MODUL V**

**PENGURUTAN (SORTING)**

1. **TUJUAN**
2. Mahasiswa memahami bagaimana logika yang digunakan dalam setiap jenis-jenis *sorting*.
3. Mahasiswa mampu menerapkan algoritma *sorting* ke dalam program.
4. **LANDASAN TEORI**

Algoritma pengurutan adalah metode yang dibutuhkan untuk mengurutkan data. Data yang diurutkan dapat berupa tipe dasar atau bertipe tersturktur (*struct*). Jika data bertipe terstruktur, maka harus dispesifikasikan berdasarkan *field* apa data tersebut diurutkan. Terdapat banyak algoritma yang digunakan untuk mengurutkan data. Setiap algoritma mempunyai kekuatan dan kelemahan. Pada banyak cara pemecahan masalah, teknik terbaik untuk menggunakan algoritma pengurutan adalah dengan membangun algoritma pengurutan dalam bentuk fungsi. Jenis algoritma pengurutan yang dibahas pada modul ini adalah algoritma *buble sort, selection sort, shell short,* dan *insertion sort*.

1. *Buble Sort*

Algoritma pengurutan Apung akan membandingkan elemen yang sekarang dengan elemen berikutnya, pada pengurutan ascending jika nilai elemen sekarang > elemen berikutnya, maka dilakukan proses penukaran. Misalkan terdapat L larik yang tiap indeksnya k berisi data belum terurut dengan jumlah elemen n, maka untuk mendapatkan larik yang terurut ascending, algoritmanya secara umum seperti berikut.

Procedure Buble\_sort ( Larik[] array of integer, n > integer )

Begin

For i=1 to n do

For j=1+i to n do

if data[i]>data[j] swap(data[i],data[j])

end.

1. *Selection Sort*

Ide dasar pengurutan Seleksi adalah memilih elemen maksimum/minimum dari larik lalu menempatkan elemen maksimum/minimum. itu pada awal atau akhir larik. Selanjutnya elemen awal/akhir tersebut “diisolasi” dan tidak disertakan dalam proses perbandingan selanjutnya. Seperti pada algoritma pengurutan Apung, proses memilih nilai maksimum/minimum dilakukan pada setiap pass. Jika larik berukuran n, maka jumlah pass adalah n-1. Algoritma pengurutan Seleksi untuk mendapatkan larik yang terurut ascending, secara umum seperti berikut.

Procedure Selection\_Sort (Larik[] array of integer, n > integer)

Begin

For i=1 to n do

Temp = i;

For j=i+1 to n do

If data[j]<data[Temp]

Temp;

Swap(data[j], data[Temp]

End.

1. Insertion Sort

Inspirasi dari algoritma pengurutan Sisip adalah seperti ketika mengurutkan kartu pada permainan kartu. Metode pengurutan dengan cara

menyisipkan elemen larik pada posisi yang tepat. Pencarian posisi yang tepat dilakukan dengan menyisir larik. Selama penyisiran dilakukan pergeseran elemen larik.

Algoritma pengurutan Sisip untuk mendapatkan larik yang terurut

ascending, secara umum seperti berikut.

Untuk setiap pass i=2, ..., n lakukan:

1. y = L[i]

2. Sisipkan y pada tempat yang sesuai di antara L[1]...L[i].

1. *Shell Short*

Pada algoritma pengurutan Shell jarak antar dua elemen yangdibandingkan dan ditukarkan. Langkah pertama, ambil elemen pertama dan bandingkan dengan elemen pada jarak tertentu dari elemen pertama tersebut. Kemudian elemen kedua dibandingkan dengan elemen lain dengan jarak yang sama seperti diatasnya. Demikian seterusnya sampai seluruh elemen dibandingkan. Pada pass berikutnya proses diulang dengan langkah yang lebih kecil, hingga proses dihentikan jika jarak sudah sama dengan satu. Secara umum, algoritma pengurutan Shell dituliskan sebagai berikut.

1. Step = n {n = ukuran larik}

2. While Step > 1 do

a. Step = Step div 3 +1

b. For i=1 to Step do

Insertion Sort setiap elemen ke-Step mulai dari elemen ke-i.

1. **TUGAS PRAKTIKUM**

Gunakanlah keempat jenis algoritma pengurutan diatas untuk mengurutkan data integer sebagai berikut.

15, 8, 0, -2, 19, 42, 31, 81, -72, 1, 2.

**MODUL VI**

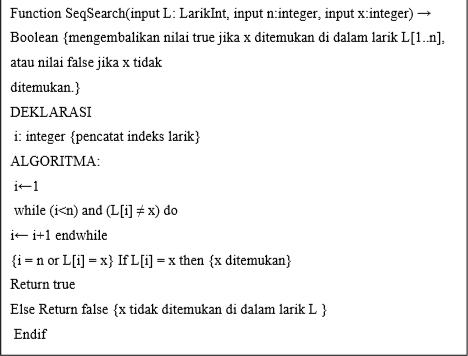
**PENCARIAN (*SEARCHING*)**

1. **TUJUAN**
2. Mahasiswa memahami bagaimana logika yang digunakan dalam setiap jenis-jenis searching.
3. Mahasiswa mampu menerapkan algoritma searching ke dalam program.
4. **LANDASAN TEORI**

Algoritma pencarian atau searching adalah metode untuk menemukan nilai (data) tertentu dalam sekumpulan data yang bertipe sama. Contohnya untuk mengubah (update) data tertentu, langkah pertama yang harus dilakukan adalah mencari keberadaan data tersebut di dalam kumpulannya. Jika data yang dicari ditemukan, maka data tersebut dapat diubah nilainya dengan data yang baru. Pada modul ini akan dibahas algoritma pencarian Beruntun (Sequential Search), Bagi Dua (Binary Search), Knuth Moris Prath String Search.

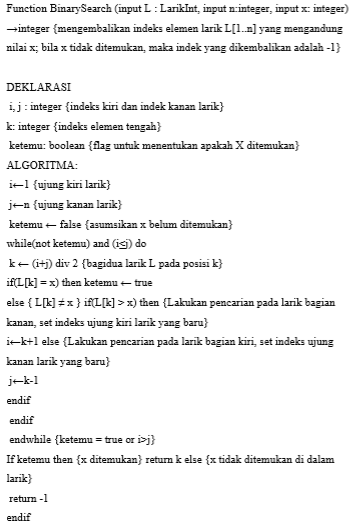
1. Sequential Search

Gagasan pada algoritma pencarian Beruntun adalah membandingkan setiap elemen larik satu per satu secara beruntun, mulai dari elemen pertama, sampai elemen yang dicari ditemukan, atau seluruh elemen larik sudah diperiksa. Berikut ini adalah pseudocode untuk fungsi pencarian beruntun



1. Selection Search

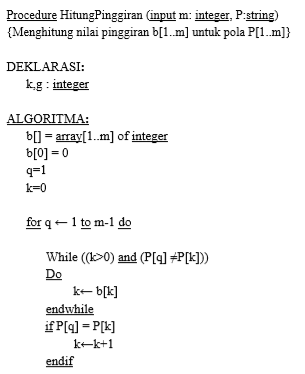
Gagasan pada algoritma pencarian Bagi Dua adalah membagi data atas dua bagian sebelum melakukan pencarian. Untuk memudahkan proses pencarian elemen elemen data sudah dalam kondisi terurut. Dalam proses pencarian Bagi Dua, diperlukan dua buah indeks larik, yaitu indeks terkecil dan indeks terbesar. Berikut ini adalah pseudocode untuk fungsi pencarian Bagi Dua.



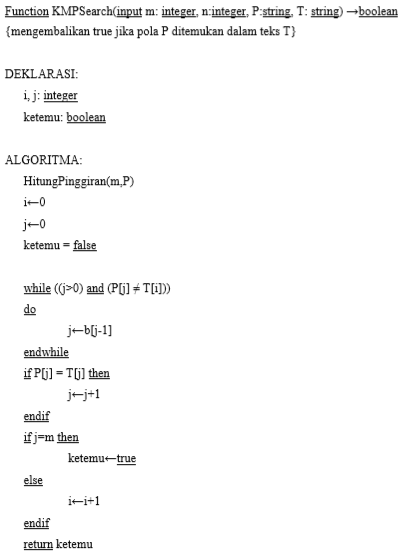
1. Knuth Moris Prath String Search

Knuth Moriss Prath (KMP) adalah salah satu algoritma yang digunakan untuk melakukan pencocokan string atau mencari string tertentu dalam suatu kumpulan teks. Algoritma KMP secara umum adalah seperti berikut.

1. Pencocokan pola dimulai pada awal teks, dari kiri ke kanan. KMP akanmencocokan karakter per karakater di dalam pola dengan karakterkarakter padateks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi.
2. Karakter pada pola dan teks yang dibandingkan tidak cocok.
3. Semua karakter pada pola cocok, kemudian posisi penemuan pola pada teks diberitahukan.
4. Proses pencocokan pola akan dilakukan sampai pola berada di ujung teks. Algoritma KMP melakukan proses awal (preprocessing) terhadap pola P dengan menghitung fungsi pinggiran. Pada beberapa literatur disebut fungsi overlap, fungsi failure, fungsi awalan, dan sebagainya. Fungsi ini mengindikasikan pergeseran Pola yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian string. Berikut ini pseudocode untuk menghitung Pinggiran dalam bentuk prosedur, pada bahasa pemrograman C++ prosedur adalah fungsi yang tidak menghasilkan nilai balik.







1. **TUGAS PRAKTIKUM**
2. Toko A memiliki sistem untuk menyimpan pembeli berdasarkan ID pembeli. ID pembeli dibuat berdasarkan jam menit dan detik. Sebagai contoh, pembeli A membeli barang pada 19:54, maka ID pelanggan adalah 1954. Buatlah program untuk mencari ID pelanggan dari minimal 10 data dengan metode Sequential Search dan Binary Search. Bila tidak ditemukan akan ditampilkan pesan “Data Tidak Ditemukan!”. Bila data ditemukan, akan ditampilkan pesan “Data Ditemukan pada indeks ke-n”.
3. Buatlah program untuk mencari kata “mimpimu” dan “coba” dari kalimat “mimpimu memiliki akhir yang tidak terbatas, tarik napas yang dalam dan coba lagi”.