MODUL PRAKTIKUM
"STRUKTUR DATA"

Bahasa Pemrograman : C++

Software : Turbo C++ 4.5, Borland C++ Builder 6

Laboran : M. Fachrurrozi

LABORATORIUM DASAR KOMPUTER
PROGRAM ILMU KOMPUTER
UNIVERSITAS SRIWIJAYA
2006

DAFTAR ISI

DAFTAR ISI 1
Bab 1. POINTER 2
Bab 2. ARRAY 5
2.1. Array Satu Dimensi 5
2.2. Array Dua Dimensi 7
Bab 3. STRUCTURE 15
Bab 4. CLASS 18
Bab 5. LINKED LIST 39
5.1. Single Linked List 39
5.2. Operasi Pada Single Linked List 41
5.3. Double Linked List 43
Bab 6. STACK 50
6.1. Definisi Stack 50
6.2. Stack dengan Array 51
6.3. Double Stack dengan Array 52
6.4. Stack dengan Single Linked List 53
Bab 7. QUEUE 62
7.1. Definisi Queue 62
7.2. Implementasi Queue dengan Linear Array 62
7.3. Implementasi Queue dengan Circular Array 64
7.4. Implementasi Queue dengan Double Linked List 65
Bab 8. TREE 73
8.1. Definisi Tree 73
8.2. Jenis-jenis Tree 74
8.3. Operasi-operasi pada Binary Tree 75
8.4. Binary Search Tree 76
REFERENSI 84

mfachrz@gmail.com

Bab 1. POINTER

Pointer merupakan tipe data berukuran 32 bit yang berisi satu nilai yang berpadanan dengan alamat memori tertentu. Sebagai contoh, sebuah variabel P bertipe pointer bernilai 0x0041FF2A, berarti P menunjuk pada alamat memori 0041FF2A. Pointer dideklarasikan seperti variabel biasa dengan menambahkan tanda * (asterik) yang mengawali nama variabel.

Bentuk Umum:

<tipe data> namaVariabel;

Contoh:

float * px;

Statement di atas mendeklarasikan variabel px yang merupakan pointer. Penyebutan tipe data float berarti bahwa alamat memori yang ditunjuk oleh px dimaksudkan untuk berisi data bertipe float.

Contoh Program:

1:

mfachrz@gmail.com

```
#include <iostream.h>

void main()
{
   int x;
   int *px;

   x = 2;
   px = &x;   //membaca alamat dari x

   cout<<"Nilai x              : "<<x<<endl;
   cout<<"Nilai *px            : "<<x<<endl;
   cout<<"Nilai px (alamat x) : "<<px<<endl;
}
```

Output:

```
Nilai x              : 2
Nilai *px            : 2
Nilai px (alamat x) : 0x2467242e
```

2:

```
#include <iostream.h>

void main()
{
   int x[10]={0,1,2,3,4,5,6,7,8,9};
   int *px;
   int i;

   for(i=0;i<10;i++)
   {
      px = &x[i];   //membaca alamat dari x

      cout<<x[i]<<" "<<*px<<" "<<px<<endl;
   }
}
```

Output:

mfachrz@gmail.com

```
0 0 0x250f23da
1 1 0x250f23dc
2 2 0x250f23de
3 3 0x250f23e0
4 4 0x250f23e2
5 5 0x250f23e4
6 6 0x250f23e6
7 7 0x250f23e8
8 8 0x250f23ea
9 9 0x250f23ec
```

3:

```
#include <iostream.h>

void main()
{
   char *nama;

   nama = "Muhammad Fachrurrozi";
   cout<<"Selamat datang "<<nama<<endl;
}
```

Output:

```
Selamat datang Muhammad Fachrurrozi
```


mfachrz@gmail.com

Bab 2. ARRAY

Array adalah suatu struktur yang terdiri dari sejumlah elemen yang memiliki tipe data yang sama. Elemen-elemen array tersusun secara sekuensial dalam memori komputer. Array dapat berupa satu dimensi, dua dimensi, tiga dimensi ataupun banyak dimensi (multi dimensi).

2.1. Array Satu Dimensi

Array Satu dimensi tidak lain adalah kumpulan elemen-elemen identik yang tersusun dalam satu baris. Elemen-elemen tersebut memiliki tipe data yang sama, tetapi isi dari elemen tersebut boleh berbeda.

Elemen ke-	0	1	2	3	4	5	6	7	8	9
Nilai	23	34	32	12	25	14	23	12	11	10

Bentuk umum:

<tipe data> NamaArray[n] = {elemen0, elemen1, elemen2,.....,n};

n = jumlah elemen

Contoh Program:

1.

```
/*Program Mencari bilangan terkecil
  dan terbesar di dalam array */

#include <iostream.h>

void main()
{
   int x[10]={45,34,23,34,32,12,65,76,34,23};
   int i;
   int maks = -1000; //asumsi paling minimum
   int min = 1000;  //asumsi paling maksimum
```


mfachrz@gmail.com

```
    for(i=0;i<10;i++)
    {
       if(x[i]>maks)
       {
          maks = x[i];
       }

       if(x[i]<min)
       {
          min = x[i];
       }
    }

    cout<<"Nilai maksimum : "<<maks<<endl;
    cout<<"Nilai minimum : "<<min<<endl;
}
```

Output:

```
Nilai maksimum : 76
Nilai minimum : 12
```

2.

```
/*Program Mencari bilangan tertentu di dalam array
  mencari jumlah data yang ditemukan dan
  di elemen mana saja bilangan itu ditemukan */

#include <iostream.h>
typedef enum {false=0, true=1} bool;

void main()
{
   int x[10]={45,34,23,34,32,12,65,76,34,23};
   int i,bil,jumlah;
   bool ketemu=false;

   jumlah = 0;

   cout<<"Bilangan yang akan dicari : ";
   cin>>bil;
```


mfachrz@gmail.com

```
    for(i=0;i<10;i++)
    {
       if(x[i]==bil)
       {
          ketemu = true;
          cout<<"Bilangan ditemukan di elemen : "<<i<<endl;
          jumlah++;
       }
    }

    if(ketemu)
    {
       cout<<"Jumlah data : "<<jumlah;
    }
    else
    {
       cout<<"Bilangan tersebut tidak ditemukan";
    }
}
```

Output:

```
Bilangan yang akan dicari : 23
Bilangan ditemukan di elemen : 2
Bilangan ditemukan di elemen : 9
Jumlah data : 2
```

Atau

```
Bilangan yang akan dicari : 30
Bilangan tersebut tidak ditemukan
```

2.2. Array Dua Dimensi

Array dua dimensi sering digambarkan sebagai sebuah matriks, merupakan perluasan dari array satu dimensi. Jika ***array satu dimensi hanya terdiri dari sebuah baris*** dan beberapa kolom elemen, maka ***array dua dimensi terdiri dari beberapa baris*** dan beberapa kolom elemen yang bertipe sama sehingga dapat digambarkan sebagai berikut:

	0	1	2	3	4	5	6
0	10	21	23	43	45	78	65

mfachrz@gmail.com

1	45	43	65	12	21	12	21
2	32	34	23	56	54	34	45
3	11	12	32	23	56	76	45

Bentuk umum:

<tipe data> NamaArray [m][n];

Atau

<tipe data> NamaArray [m][n] = { {a,b,..z},{1,2,...,n-1} };

Contoh:

```
double matrix[4][4];
bool papan[2][2] = { {true,false},{true,false} };
```

Pendeklarasian array dua dimensi hampir sama dengan pendeklarasian array satu dimensi, kecuali bahwa array dua dimensi terdapat dua jumlah elemen yang terdapat di dalam kurung siku dan keduanya boleh tidak sama.

Elemen array dua dimensi diakses dengan menuliskan kedua indeks elemennya dalam kurung siku seperti pada contoh berikut:

```
//papan nama memiliki 2 baris dan 5 kolom
bool papan[2][5];

papan[0][0] = true;
papan[0][4] = false;
papan[1][2] = true;
papan[1][4] = false;
```

Contoh program:

1.

mfachrz@gmail.com

```
/* Penjumlahan 2 buah matriks 2x2 */

#include <iostream.h>
#include <conio.h> //untuk mengaktifkan perintah gotoxy(x,y) dan clrscr()
#include <stdio.h>

#define Nmaks 10
typedef int matrik[Nmaks][Nmaks];

void main()
{
        int n,i,j;
        matrik A,B,C;

        cout<<"Program Penjumlahan Matrik A 2x2 dan B 2x2\n";
        cout<<"\n";
        n = 2;
        cout<<"Masukkan entri-entri matriks A \n";
        for (i=1;i<=n;i++)
        {
            for (j=1;j<=n;j++)
            {
               printf("A[%d,%d] = ",i,j);cin>>A[i][j];
            }
        }

        clrscr();
        cout<<"Masukkan entri-entri matriks B \n";
        cout<<"\n";
        for (i=1;i<=n;i++)
        {
            for (j=1;j<=n;j++)
            {
               printf("B[%d,%d] = ",i,j);cin>>B[i][j];
            }
        }
        clrscr();
        cout<<"\n";
        // proses penjumlahan matrik C = A + B
        for (i=1;i<=n;i++)
        {
            for (j=1;j<=n;j++)
            {
               C[i][j]=A[i][j]+B[i][j];
            }
        }
```


mfachrz@gmail.com

```
clrscr();
cout<<"entri-entri matriks A,B dan C \n";
cout<<"\n";

// proses output matrik A
gotoxy(1,5);
cout<<"A = " ;
for (i=1;i<=n;i++)
{
    for (j=1;j<=n;j++)
    {
       gotoxy(2+4*j,2+2*i);
       cout<<A[i][j];
    }
}
// proses output matrik B
gotoxy(1,10);
cout<<"B = " ;
for (i=1;i<=n;i++)
{
    for (j=1;j<=n;j++)
    {
       gotoxy(2+4*j,2+2*i+5);
       cout<<B[i][j];
    }
}
// proses output matrik C
gotoxy(1,15);
cout<<"C = " ;
for (i=1;i<=n;i++)
{
    for (j=1;j<=n;j++)
    {
       gotoxy(2+4*j,2+2*i+10);
       cout<<A[i][j];
    }
}


gotoxy(12,15);
cout<<" + " ;
for (i=1;i<=n;i++)
{
    for (j=1;j<=n;j++)
    {
       gotoxy(13+4*j,2+2*i+10);
       cout<<B[i][j];
    }
}
```


mfachrz@gmail.com

```
        gotoxy(23,15);
        cout<<" = " ;
        for (i=1;i<=n;i++)
        {
            for (j=1;j<=n;j++)
            {
               gotoxy(24+4*j,2+2*i+10);
               cout<<C[i][j];
            }
        }
}
```

Output:

```
Program Penjumlahan Matrik A 2x2 dan B 2x2

Masukkan entri-entri matriks A
A[1,1] = 1
A[1,2] = 2
A[2,1] = 3
A[2,2] = 4


Masukkan entri-entri matriks B

B[1,1] = 5
B[1,2] = 6
B[2,1] = 7
B[2,2] = 8_


entri-entri matriks A,B dan C

     1   2
A =
     3   4


     5   6
B =
     7   8


     1   2       5   6       6   8
C =          +           =
     3   4       7   8       10  12
```

2.

mfachrz@gmail.com

```
/* Perkalian matrik 2x2 */

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

#define Nmaks 10
typedef int matrik[Nmaks][Nmaks];


void main()
{
        int n,i,j;
        matrik A,B,C;

        cout<<"Program Perkalian Matrik A 2x2 dan B 2x2\n";
        cout<<"\n";
        n = 2;
        cout<<"Masukkan entri-entri matriks A \n";
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++) {
                     printf("A[%d,%d] = ",i,j);cin>>A[i][j];
                     }
                }
        clrscr();
        cout<<"Masukkan entri-entri matriks B \n";


     cout<<"\n";
     for (i=1;i<=n;i++) {
             for (j=1;j<=n;j++) {
                  printf("B[%d,%d] = ",i,j);cin>>B[i][j];
                  }
             }
     clrscr();
     cout<<"\n";
     // proses perkalian matrik C = A x B
     for (i=1;i<=n;i++) {
             for (j=1;j<=n;j++) {
                  C[i][j]=(A[i][1]*B[1][j])+(A[i][2]*B[2][j]);
                  }
             }
```


mfachrz@gmail.com

```
        clrscr();
        cout<<"entri-entri matriks A,B dan C \n";
        cout<<"\n";

        // proses output matrik A
        gotoxy(1,5);
        cout<<"A = " ;
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++) {
                     gotoxy(2+4*j,2+2*i);
                     cout<<A[i][j];
                     }

                }
        // proses output matrik C
        gotoxy(1,15);
        cout<<"C = " ;
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++) {
                     gotoxy(2+4*j,2+2*i+10);
                     cout<<A[i][j];
                     }
                }
        gotoxy(12,15);
        cout<<" x " ;
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++) {
                     gotoxy(13+4*j,2+2*i+10);
                     cout<<B[i][j];
                     }
                }


        gotoxy(23,15);
        cout<<" = " ;
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++) {
                     gotoxy(24+4*j,2+2*i+10);
                     cout<<C[i][j];
                     }
                }
}
```

Output:

```
Program Perkalian Matrik A 2x2 dan B 2x2

Masukkan entri-entri matriks A
A[1,1] = 1
A[1,2] = 2
A[2,1] = 3
A[2,2] = 4
```


mfachrz@gmail.com

```
Masukkan entri-entri matriks B

B[1,1] = 5
B[1,2] = 6
B[2,1] = 7
B[2,2] = 8_
```

```
entri-entri matriks A,B dan C

     1   2
A =
     3   4


     5   6
B =
     7   8


     1   2       5   6       19  22
C =          x           =
     3   4       7   8       43  50
```

Latihan:

1. Buat program menghitung penjumlahan matrik 3x3.
2. Buat program menghitung perkalian matrik 3x3.

mfachrz@gmail.com

Bab 3. STRUCTURE

Structure (struktur) adalah kumpulan elemen-elemen data yang digabungkan menjadi satu kesatuan. Masing-masing elemen data tersebut dikenal dengan sebutan field. Field data tersebut dapat memiliki tipe data yang sama ataupun berbeda. Walaupun field-field tersebut berada dalam satu kesatuan, masing-masing field tersebut tetap dapat diakses secara individual.

Field-field tersebut digabungkan menjadi satu dengan tujuan untuk kemudahan dalam operasinya. Misalnya Anda ingin mencatat data-data mahasiswa dan pelajar dalam sebuah program, Untuk membedakannya Anda dapat membuat sebuah record mahasiswa yang terdiri dari field nim, nama, alamat dan ipk serta sebuah record pelajar yang terdiri dari field-field nama, nonurut, alamat dan jumnilai. Dengan demikian akan lebih mudah untuk membedakan keduanya.

Bentuk umum:

```
struct namastruct
{
   <tipe data> field1;
   <tipe data> field2;
   <tipe data> field3;
};
```

Contoh:

```
struct mahasiswa
{
   char nim[11];
   char nama[30];
   char lamat[50];
   float ipk;
};
```


mfachrz@gmail.com

Untuk menggunakan struktur, tulis nama struktur beserta dengan fieldnya yang dipisahkan dengan tanda titik (“ . “). Misalnya Anda ingin menulis nim seorang mahasiswa ke layar maka penulisan yang benar adalah sebagai berikut:

```
cout<<mahasiswa.nim;
```

Jika Pmhs adalah pointer bertipe mahasiswa* maka field dari Pmhs dapat diakses dengan mengganti tanda titik dengan tanda panah (“ **à** “).

```
cout<<mahasiswa->nim;
```

Contoh program:

1.

```
/* Mengisi Biodata dan Nilai IPK mahasiswa */
#include <iostream.h>

struct mahasiswa
{
   char nim[15];
   char nama[30];
   char alamat[50];
   float ipk;
};


void main()
{
   mahasiswa mhs;

   cout<<"NIM        : "; cin.getline(mhs.nim,15);
   cout<<"Nama       : "; cin.getline(mhs.nama,30);
   cout<<"Alamat     : "; cin.getline(mhs.alamat,50);
   cout<<"Nilai IPK  : "; cin>>mhs.ipk;

   cout<<endl;
```


mfachrz@gmail.com

```
    cout<<endl;

    cout<<"NIM Anda         : "<<mhs.nim<<endl;
    cout<<"Nama Anda        : "<<mhs.nama<<endl;
    cout<<"Alamat Anda      : "<<mhs.alamat<<endl;
    cout<<"Nilai IPK Anda   : "<<mhs.ipk<<endl;
}
```

Output:

```
NIM         : 09983110077
Nama        : M. Fachrurrozi
Alamat      : Jl. Joko Atas No 23 Palembang
Nilai IPK   : 3.00

NIM Anda         : 09983110077
Nama Anda        : M. Fachrurrozi
Alamat Anda      : Jl. Joko Atas No 23 Palembang
Nilai IPK Anda   : 3
```

Latihan:

1. Buat program menghitung durasi rental warnet, dengan ketentuan perhitungannya:

   30 detik = Rp. 130,-

   Satuan waktu : jam : menit : detik

2. Buat program menghitung jumlah nilai akhir mahasiswa dengan ketentuan:

   Nilai akhir = 10%*tugas + 20%*kuis + 30%*mid + 40%*uas

   Nilai Huruf:

Nilai akhir >85	: A
85 >= nilai akhir > 70	: B
70 >= nilai akhir > 55	: C
55 >= nilai akhir > 40	: D
Nilai akhir <=40	: E

mfachrz@gmail.com

Bab 4. CLASS

Pemrograman C++ memerlukan pemahaman yang memadai untuk menterjemahkan desain ke dalam bentuk implementasi, terutama untuk desain yang menggunakan abstraksi class. Fokus pembahasan pada aspek pembentukan obyek (***construction***) sebuah class, dan proses sebaliknya pada saat obyek tersebut sudah tidak digunakan lagi (***destruction***).

DEKLARASI DAN DEFINISI

Deklarasi dan definisi adalah langkah awal dalam setiap penulisan program tidak terkecuali dalam bahasa C++. Deklarasi dan definisi diperlukan untuk semua tipe data termasuk tipe data bentukan user (***user-defined type***).
Bentuk sederhana deklarasi class adalah sebagai berikut,

```
class C {    }; atau
struct C {    };
```

dalam bahasa C++ struct dan class mempunyai pengertian yang sama. Deklarasi class dengan struct mempunyai anggota dengan akses public kecuali jika dinyatakan lain.

```
struct C
{
        int i;
        void f();
}
class C
{
```


mfachrz@gmail.com

```
    public:
    int i;
    void f();
}
```

Kedua deklarasi tersebut mempunyai arti yang sama. Hal ini adalah pilihan desain yang diambil oleh desainer C++ (Bjarne Stroustrup) untuk menggunakan C sebagai basis C++ ketimbang membuat bahasa yang sama sekali baru. Tentunya ada konsekuensi atas pilihan desain ini, salah satu contoh adalah kompatibilitas terhadap bahasa C. Dalam bahasa C deklarasi,

```
struct C { … };
```

menyatakan C sebagai nama tag. Nama tag berbeda dengan nama tipe, sehingga C (nama tag) tidak dapat dipergunakan dalam deklarasi yang membutuhkan C sebagai suatu tipe obyek. Kedua contoh deklarasi berikut ini tidak valid dalam bahasa C,

```
C c;      /* error, C adalah nama tag */
C *pc;    /* error, C adalah nama tag */
```

Dalam bahasa C, kedua deklarasi tersebut harus ditulis sebagai berikut,

```
struct C c;
struct C *pc;
```

atau menggunakan typedef sebagai berikut,

```
struct C { … };
typedef struct C C;
C c;
```

```
C *pc;
```

C++ memperlakukan nama class, C sebagai nama tag sekaligus nama tipe dan dapat dipergunakan dalam deklarasi. Kata class tetap dapat dipergunakan dalam deklarasi, seperti contoh berikut ini,

```
class C c;
```

Dengan demikian C++ tidak membedakan nama tag dengan nama class, paling tidak dari sudut pandang pemrogram (***programmer***), dan tetap menerima deklarasi ***structure*** seperti dalam bahasa C. Kompatibilitas C++ terhadap tidak sebatas perbedaan nama tag dan nama tipe, karena standar C++ masih perlu mendefinisikan tipe POD (Plain Old Data). POD type mempunyai banyak persamaan dengan structure dalam C. Standar C++ mendefinisikan POD type sebagai obyek suatu class yang tidak mempunyai userdefined constructor, anggota protected maupun private, tidak punya base class, dan tidak memiliki fungsi virtual. Dalam desain suatu aplikasi terdiri atas banyak class, dan masing-masing class tidak berdiri sendiri melainkan saling bergantung atau berhubungan satu sama lain. Salah satu contoh hubungan tersebut adalah hubungan antara satu class dengan satu atau lebih ***base class*** atau ***parent class***. Jika class C mempunyai base class B, dikenal dengan ***inheritance***, maka deklarasi class menjadi,

```
class C : public B {}; atau
class C : protected B {}; atau
class C : private B {};
```

akses terhadap anggota base class B dapat bersifat public, protected, maupun private,atau disebut dengan istilah ***public***, ***protected*** atau ***private inheritance***. Class C disebut dengan istilah ***derived class***. Jika tidak dinyatakan bentuk akses secara eksplisit, seperti dalam deklarasi berikut:

mfachrz@gmail.com

```
class C : B
```

maka interpretasinya adalah private inheritance (***default***), tetapi jika menggunakan struct maka tetap merupakan public inheritance. Jika desainer class C tersebut menginginkan hubungan ***multiple inheritance*** (MI) terhadap class B dan A, maka deklarasi class C menjadi,

```
class C : public B, public A { };
```

Sebuah class, seperti halnya class C mempunyai anggota berupa data maupun fungsi (***member function***). Isi class tersebut berada diantara tanda kurung { } dan dipilah-pilah sesuai dengan batasan akses yang ditentukan perancang (desainer) class tersebut.

```
class C : public B
{
    public:
    (explicit) C()(:member-initializer);
    C(const C& );
    C& operator=(const C&);
    (virtual)~C();
    statement lain
    (protected: statement)
    (private: statement)
};
```

Secara ringkas batasan akses (***access specifiers***) mempunyai arti seperti ditunjukkan pada table berikut ini,

Batasan Akses	Arti

Public	Semua class atau bebas
Protected	Class itu sendiri, ***friend***, atau ***derived class***
Private	Class itu sendiri, ***friend***

Sebuah class dapat memberikan ijin untuk class lain mengakses bagian protected maupun private class tersebut melalui hubungan ***friendship*** (dinyatakan dengan ***keyword*** friend). Sebuah class mempunyai beberapa fungsi khusus, yaitu ***constructor***, ***copy constructor***, ***destructor*** dan ***copy assignment operator***.

Constructor

C() adalah anggota class yang bertugas melakukan inisialisasi obyek (***instance***) dari suatu class C. Constructor mempunyai nama yang sama dengan nama class, dan tidak mempunyai ***return value***. Sebuah class dapat mempunyai lebih dari satu constructor. Constructor yang tidak mempunyai argumen, disebut ***default constructor***, sebaliknya constructor yang mempunyai lebih dari satu argumen adalah ***non-default consructor***. Constructor dengan satu default argument tetap merupakan sebuah default constructor,

```
class C
{
        public:
        C(int count=10) : _count(count) {}
        …
        private:
        int _count;
};
```

Compiler C++ dapat menambahkan ***default constructor*** bilamana diperlukan, jika dalam definisi class

- tidak tertulis secara eksplisit sebuah default constructor dan tidak ada deklarasi constructor lain (***copy constructor***).
- tidak ada anggota class berupa data const maupun ***reference***.

Sebagai contoh definisi class C sebagai berikut,

```
class C {…};
C c1; // memerlukan default constructor
C c2(c1); // memerlukan copy constructor
```

Compiler C++ memutuskan untuk menambahkan default dan copy construtor setelah menemui kedua baris program tersebut, sehingga definisi class secara efektif menjadi sebagai berikut,

```
class C
{
    public:
    C(); // default costructor
    C(const C& rhs); // copy constructor
    ~C(); // destructor
    C& operator=(const C& rhs); // assignment operator
    C* operator&(); // address-of operator
    const C* operator&(const C& rhs) const;
};
```

compiler menambahkan public constructor, dan destructor. Selain itu, compiler juga menambahkan ***assignment operator*** dan ***address-of operator***. Constructor (default dan non-default) tidak harus mempunyai akses public, sebagai contoh adalah pola desain (***design pattern***) Singleton.

```
class Singleton
{
        public:
        static Singleton* instance();
        protected:
        Singleton();
        private:
        static Singleton* _instance;
};
```

obyek (***instance***) ***singleton*** tidak dibentuk melalui constructor melainkan melalui fungsi instance. Tidak ada obyek ***singleton*** lain yang dapat dibentuk jika sudah ada satu obyek ***singleton***. Umumnya default constructor bentukan compiler (***generated default constructor***) menggunakan default constructor anggota bertipe class, sedangkan anggota biasa (***builtin type***) tidak diinisialisasi. Demikian halnya dengan obyek yang dibentuk dari obyek lain (copy), maka copy constructor bentukan compiler (***generated copy constructor***) menggunakan copy constructor dari anggota bertipe class pada saat inisialisasi. Sebagai contoh deklarasi class C berikut ini,

```
class C
{
        public:
        C(const char* aName);
        C(const string& aName);
        …
        private:
        std::string name;
};
```

copy constructor bentukan compiler menggunakan copy constructor class string untuk inisialisasi name dari aName. Jika class C tidak mempunyai constructor, maka compiler menambahkan juga default constructor untuk inisialisasi name menggunakan default constructor class string. Inisialisasi obyek menggunakan constructor (non-default) dapat dilakukan dengan ***member initializer*** maupun dengan ***assignment*** sebagai berikut,

member initialization

```
class C
{
    int i,j;
    public:
    C() : i(0),j(1) {}
    …
};
```

assignment

```
class C
{
    int i,j
    public:
    C()
    {
        i=0;j=0;
    }
    …
};
```


mfachrz@gmail.com

Kedua cara tersebut memberikan hasil yang sama, tidak ada perbedaan signifikan antara kedua cara tersebut untuk data bukan tipe class. Cara member initializer mutlak diperlukan untuk data const maupun ***reference***, seperti kedua contoh berikut ini:

```
class C //:1
{
    public:
    C(int hi,int lo) : _hi(hi),_lo(lo) {}
    …
    private:
    const int _hi,_lo; // const member
};
class C //:2
{
    public:
    C(const string& aName) : name(aName) {}
    …
    private:
    std::string& name; // reference member
};
```

Cara ***member initialization*** sebaiknya dilakukan untuk anggota bertipe class (***userdefined type***) seperti ditunjukkan pada contoh berikut ini,

```
class C
{
    public:
    C(const string& aName) : name(aName) { }
    private:
    std::string name; // bukan reference member
```


mfachrz@gmail.com

```
};
```

Pertimbangan menggunakan cara ***member initialization*** terletak pada efisiensi eksekusi program. Hal ini berkaitan dengan cara kerja C++ yang membentuk obyek dalam dua tahap,

- pertama, inisialisasi data
- kedua, eksekusi constructor (assignment)

Dengan demikian jika menggunakan cara assignment sebenarnya eksekusi program dilakukan dua kali, pertama inisialisasi kemudian assignment, sedangkan menggunakan ***member initialization*** hanya memanggil sekali constructor class string. Semakin kompleks class tersebut (lebih kompleks dari class string) semakin mahal (tidak efisien) proses pembentukan obyek melalui cara assignment. Constructor dengan satu argumen berfungsi juga sebagai ***implicit conversion operator***. Sebagai contoh deklarasi class A dan B berikut ini,

```
class A
{
        public:
        A();
};
class B
{
        public:
        B(const A&);
};
```

pada cuplikan baris program di bawah ini terjadi konversi tipe obyek A ke B secara implisit melalui copy constructor class B.

```
A a
B b=a; // implicit conversion
```

explicit

C++ menyediakan satu sarana, menggunakan keyword explicit, untuk mengubah perilaku constructor dengan satu argumen agar tidak berfungsi sebagai ***conversion operator***. Jika class B menyatakan explicit pada copy constructor sebagai berikut,

```
class B
{
    public:
    explicit B(const A& a); // explicit ctor
};
```

maka konversi A ke B secara implisit tidak dapat dilakukan. Konversi A ke B dapat dilakukan secara eksplisit menggunakan ***typecast***,

```
A a;
B b=static_cast<B>(a); atau
B b=(B)a;
```

Konversi secara implisit dapat terjadi melalui argumen fungsi f dengan tipe B

```
void f(const B& );
```

tetapi f diakses dengan variabel tipe A, f(a). Apabila class B menghalangi konversi secara implisit maka argumen fungsi f menjadi,

```
f((B)a); atau
f(static_cast<B>(a));
```


mfachrz@gmail.com

Konversi tipe obyek secara implisit sebaiknya dihindari karena efeknya mungkin lebih besar terhadap aplikasi program secara keseluruhan dan tidak dapat dicegah pada saat kompilasi, karena construcor dengan argumen tunggal adalah suatu pernyataan program yang sah dan memang dibutuhkan.

Copy Constructor dan Copy Assignment

Sejauh ini sudah dibahas mengenai copy constructor sebagai anggota class yang berperan penting pada saat pembentukan obyek. Apabila sebuah class tidak menyatakan secara tegas copy constructor class tersebut, maka compiler menambahkan copy constructor dengan bentuk deklarasi,

```
C(const C& c);
```

Bentuk lain copy constructor adalah sebagai berikut,

```
C(C& c); atau
C(C volatile& c); atau
C(C const volatile& c);
```

Copy constructor class C adalah constructor yang mempunyai satu argumen. Sebuah copy constructor boleh mempunyai lebih dari satu argumen, asalkan argumen tersebut mempunyai nilai default (***default argument***).

```
C(C c); // bukan copy constructor
C(C const& c,A a=b); //copy constructor
```

Constructor dengan argumen bertipe C saja (tanpa ***reference***) bukan merupakan copy constructor. Copy constructor juga dibutuhkan pada saat memanggil suatu fungsi yang menerima argumen berupa obyek suatu class,

```
void f(C x);
```


mfachrz@gmail.com

memerlukan copy onstructor class C untuk mengcopy obyek c bertipe C ke obyek x dengan tipe yang sama, yaitu pada saat memanggil fungsi f(c)(***pass-by-value***).

Hal serupa terjadi pada saat fungsi f sebagai berikut,

```
C f()
{
C c;
…
return c;
}
```

mengirim obyek c ke fungsi lain yang memanggil fungsi f() tersebut. Copy assignment operator class C adalah operator=, sebuah fungsi yang mempunyai satu argumen bertipe C. Umumnya deklarasi copy assignment mempunyai bentuk,

```
C &operator=(const C &c);
```

Bentuk lain yang mungkin adalah,

```
C &operator=(C &c); atau
C &operator=(C volatile &c); atau
C &operator=(C const volatile &c);
```

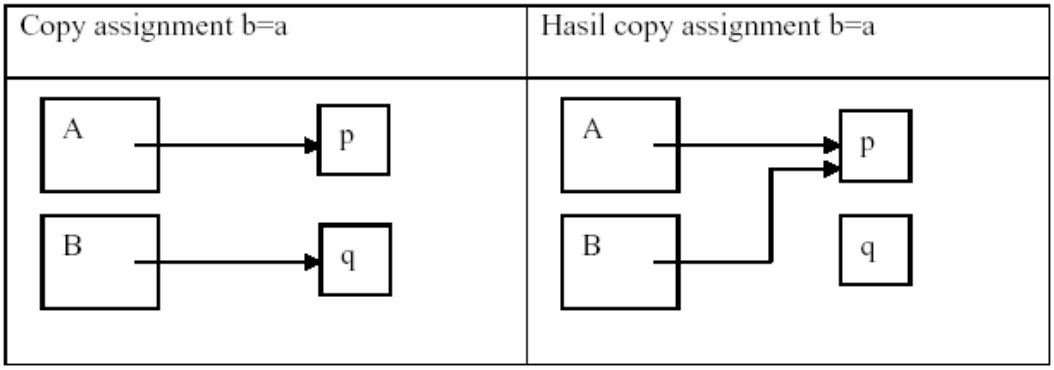
Copy assignment boleh mempunyai argumen dengan tipe C (bukan reference), tetapi tidak boleh mempunyai argumen lebih dari satu walaupun argumen tersebut mempunyai nilai ***default*** (default argument). Seperti halnya copy constructor, compiler akan menambahkan copy assignment jika suatu class tidak mempunyai fungsi tersebut. Copy assignment dibutuhkan untuk membentuk obyek melalui assignment, seperti contoh berikut:

```
class C
```

```
{
    public:
    C(); //ctor
    ~C(); //dtor
    …
};
C c1;
C c2=c1; //copy constructor
C c3;
c3=c1; //copy assignment
```

Class C tidak mempunyai copy constructor maupun copy assignment operator, maka pembentukan obyek c2, dan c3 menggunakan copy constructor dan copy assignment yang ditambahkan oleh compiler ke class C tersebut. Suatu class yang mempunyai data dengan alokasi dinamik (pointer) sebaiknya tidak mengandalkan copy constructor maupun copy assignment operator yang ditambahkan compiler. Copy assignment hasil tambahan compiler mengcopy (***memberwise copy***) pointer dari obyek satu (yang dicopy) ke obyek lainnya (hasil copy), sehingga kedua obyek mengacu ke lokasi memori yang sama. Masalah timbul jika kedua obyek mempunyai masa pakai (***lifetime1***) yang berbeda. Jika salah satu obyek sudah habis masa pakainya maka destructor obyek tersebut mengembalikan memori (***dynamic memory***) yang digunakan obyek tersebut, padahal copy obyek tersebut masih mengacu ke lokasi memori yang sama.

mfachrz@gmail.com



Pada contoh hasil copy assignment b=a (***shalow copy***), menunjukkan kedua obyek a dan b mengacu ke lokasi memori p. Apabila obyek a melepas memori p (melalui destructor), maka obyek b mengacu ke lokasi memori yang sudah tidak valid lagi. Lokasi memori p dapat digunakan obyek lain jika obyek a melepasnya. Demikian pula halnya dengan lokasi memori q, apabila obyek b habis masa pakainya (keluar scope, dihapus dll) maka destructor class B tidak melepas memori q. Akibatnya terjadi pemborosan memori (***memory leak***). Salah satu jalan keluar adalah dengan menyatakan secara tegas copy constructor dan copy assignment yang dibutuhkan suatu class sehingga compiler tidak membuat copy constructor dan copy assignment ke class tersebut. Alternatif lain adalah menempatkan deklarasi copy constructor dan copy assignment operator private sebagai berikut,

```
class C
{ …
    private:
    C(const C&);
    C &operator=(const C&);
};
```

definisi copy constructor dan copy assignment operator class C pada contoh di atas tidak perlu ada, karena tujuannya adalah menghalangi proses penggandaan (copy)

menggunakan kedua fungsi tersebut. Pada tahap kompilasi penggunaan assignment, b=a masih dapat diterima karena deklarasi asignment operator tersebut tersedia. Pada saat link akan gagal karena ***linker*** tidak dapat menemukan definisi copy assignment operator. Teknik ini masih mempunyai kelemahan, karena class lain masih mungkin mempunyai akses ke private copy constructor dan copy assignment operator tersebut (melalui hubungan ***friendship***).

Destructor

Destructor adalah anggota class (***member function***) yang berfungsi melepas memori pada saat suatu obyek sudah tidak diperlukan lagi. Fungsi destructor kebalikan constructor. Destructor tidak mempunyai atau memerlukan argumen. Destructor juga tidak mengembalikan nilai apapun (tidak mempunyai ***return type***). Seperti halnya constructor, compiler dapat menambahkan sebuah destructor jika sebuah class tidak mempunyai destructor.

virtual Destructor

Sebuah destructor dapat berupa fungsi virtual. Hal ini menjadi keharusan jika class B,

- merupakan base class.
- class D yang menggunakan B sebagai base class mempunyai anggota berupa data dengan alokasi memori dinamik (pointer).

```
class B
{
        public:
        B();
        ~B();
};
class D : public B
{
```

```
    public:
    D() : p(new char[256]) {}
    ~D()
    {
        delete[] p;
    }
    …
    private:
    char *p;
};
```

Pada contoh tersebut destructor base class B bukan fungsi virtual. Dalam C++ umumnya obyek class D digunakan secara ***polimorphic*** dengan membentuk obyek class D (***derived class***) dan menyimpan alamat obyek tersebut dalam pointer class B (***base class***) seperti pada contoh berikut ini,

```
void main(void)
{
    B *pB=new D();
    delete pB;
}
```

Dalam standar C++ menghapus obyek D (derived class) melalui pointer class B (base class) sedangkan destructor base class non-virtual mempunyai efek yang tidak menentu (***undefined behaviour***). Apabila standard C++ tidak menetapkan apa yang seharusnya berlaku, maka terserah kepada pembuat compiler menentukan perilaku program pada kondisi semacam ini. Umumnya pembuat compiler mengambil langkah untuk tidak memanggil destructor class D (derived class). Dengan demikian, pada saat menjalankan perintah delete, destructor class D tidak dieksekusi karena destructor base class B

nonvirtual. Akibatnya lokasi memori dinamik yang digunakan class D tidak pernah dilepas. Hal ini adalah contoh lain terjadinya pemborosan memori (***memory leak***) oleh suatu program. Jalan keluarnya adalah membuat destructor base class B virtual,

```
class B
{
    public:
    B();
    virtual ~B();
}
```

Tidak seperti destructor, tidak ada ***virtual constructor*** atau ***virtual copy constructor***. Pada saat membentuk obyek, tipe obyek harus diketahui terlebih dahulu, apakah membentuk obyek class A, B, C dsb. Tidak ada aspek bahasa C++ untuk mewujudkan virtual constructor secara langsung, menempatkan virtual pada deklarasi constructor merupakan kesalahan yang terdeteksi pada proses kompilasi. Efek virtual constructor bukan tidak mungkin dicapai, C++ memungkinkan membuat idiom ***virtual constructor*** yang bertumpu pada fungsi virtual dalam kaitannya dengan hubungan antara sebuah class dengan ***base classnya***.

Ringkasan

Sejauh ini pembahasan artikel masih belum menyentuh aspek praktis pemrograman, namun demikian dalam menterjemahkan suatu desain maupun memahami program yang ditulis orang lain sangatlah penting mengetahui aturan dasar sesuai standarisasi C++. Butir-butir pembahasan dalam artikel ini antara lain:

- Fokus pembahasan adalah aspek pembentukan obyek. Tidak membahas aturan (***rule***) berkaitan dengan class dalam C++ secara komprehensif.

- Constructor merupakan anggota class yang berperan dalam pembentukan obyek. Compiler menambahkan constructor bilamana diperlukan ke class yang tidak mempunyai constructor. Constructor tidak harus mempunyai akses public. Inisialisasi data menggunakan constructor dapat dilakukan dengan cara member initialization dan assignment. Keduanya tidak mempunyai perbedaan signifikan untuk data biasa (***built-in type*** seperti char, int, float, dll). Cara member initialization lebih efisien untuk data berupa class (***user-defined type***).
- Constructor dengan satu argumen dapat digunakan untuk konversi tipe data secara implisit. C++ menyediakan explicit untuk mengubah perilaku ini, karena hal tersebut melonggarkan janji C++ sebagai bahasa yang mengutamakan ***strict type (type safe)***.
- Sebuah class membutuhkan copy constructor dan copy assignment operator untuk menggandakan obyek suatu class. Hal ini terjadi juga pada saat memanggil suatu fungsi dengan cara ***pass-by-value***. Apabila suatu class tidak mempunyai copy constructor dan copy assignment maka compiler menambahkannya. Copy constructor dan copy assignment hasil tambahan compiler bekerja dengan cara ***memberwise copy*** dan menghasilkan ***shalow copy*** untuk data dengan alokasi memori dinamik.
- Destructor merupakan anggota class yang berfungsi pada saat ***lifetime*** suatu obyek habis. Destructor sebuah base class sebaiknya virtual.
- Constructor selalu merupakan fungsi non-virtual. Efek ***virtual constructor*** dan ***virtual copy constructor*** mungkin diperlukan dalam suatu desain. Efek virtual constructor dapat diwujudkan melalui sifat polimorphisme class. Efek virtual copy constructor dapat diwujudkan memanfaatkan aspek ***covariant return type*** sebuah hirarki class. Kedua hal tersebut memerlukan pembahasan khusus.
- Pembahasan pembentukan obyek belum dikaitkan dengan jenis scope yang ada dalam C++. C++ mempunyai jenis scope yang lebih kaya dibandingkan bahasa C, selain file scope, function scope, dan block scope C++ memiliki ***class scope*** dan

mfachrz@gmail.com

namespace scope. Salah satu panduan praktis bahkan menyarankan untuk menunda (***lazy initialization***) pembentukan obyek selagi belum diperlukan.

- Pembentukan suatu obyek mungkin saja gagal. Artikel ini tidak membahas mengenai kegagalan pembentukan obyek, karena pembahasan tersebut berkaitan pembahasan ***exception*** dalam C++. Pembahasan exception C++ (***exception safety***) merupakan topik tersendiri.

Desain dan implementasi class C++ bukanlah hal yang mudah, masih banyak aspek lain yang belum terjangkau pembahasan artikel ini. Pada artikel selanjutnya akan dibahas scope (***visibility***) dalam C++, batasan akses (access specifier) C++, abstract class, function overloading, class relationship, template, dll.

Contoh program:

1.

```
/* Menghitung Nilai Huruf Mahasiswa */

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

class mahasiswa
{
        char nim[12];
        char nama[30];
        char alamat[50];

        public:
        void HitungNilaiAkhir(float tugas,float kuis,float mid,float uas, float &NA);
        void mahasiswa::HitungNilaiHuruf(float NA, char * &NH);
};
void mahasiswa::HitungNilaiAkhir(float tugas,float kuis,float mid,float uas, float &NA)
{
        NA = 0.1*tugas+0.2*kuis+0.3*mid+0.4*uas;
}
void mahasiswa::HitungNilaiHuruf(float NA, char * &NH)
{
        if(NA>85)
        {
                NH = "A";
        }
```


mfachrz@gmail.com

```
        else if(NA<=85&&NA>70)
        {
                NH = "B";
        }
        else if(NA<=70&&NA>55)
        {
                NH = "C";
        }
        else if(NA<=55&&NA>40)
        {
                NH = "D";
        }
        else
        {
                NH = "E";
        }
}


void main()
{
        mahasiswa mhs;
        float NA;
        char * NH;

        mhs.HitungNilaiAkhir(30,50,56,54,NA);
        cout << "Total Nilai Akhir : "<<NA<<endl;
        mhs.HitungNilaiHuruf(NA,NH);

        cout<<"Nilai Huruf : "<<NH<<endl;
}
```

Output:

```
Total Nilai Akhir : 51.4
Nilai Huruf : D
```

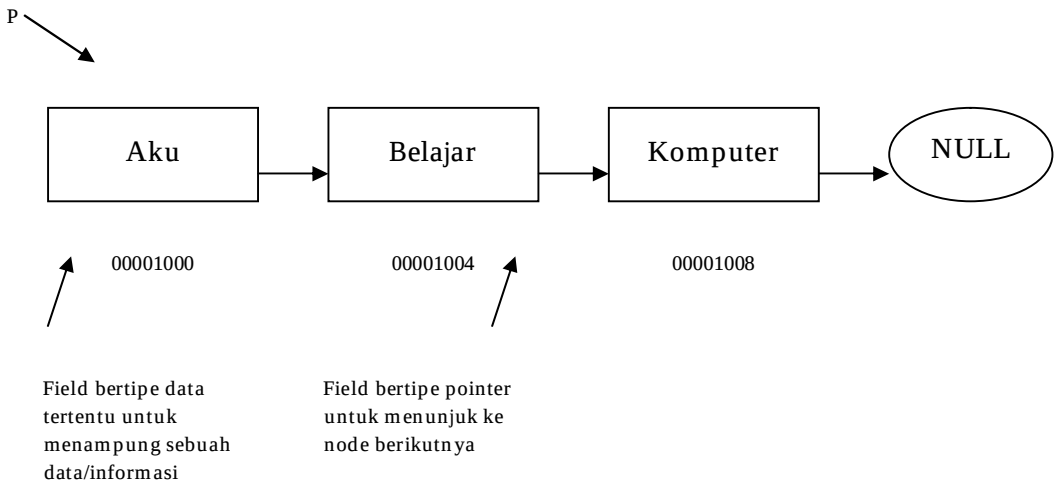

mfachrz@gmail.com

Bab 5. LINKED LIST

Pada bab sebelumnya telah dijelaskan mengenai variabel array yang bersifat statis (ukuran dan urutannya sudah pasti). Selain itu, ruang memori yang dipakai olehnya tidak dapat dihapus bila array tersebut sudah tidak digunakan lagi pada saat program dijalankan. Untuk memecahkan masalah di atas, kita dapat menggunakan variabel pointer. Tipe data pointer bersifat dinamis, variabel akan dialokasikan hanya pada saat dibutuhkan dan sesudah tidak dibutuhkan dapat direlokasikan kembali.

5.1. Single Linked List

Apabila setiap Anda ingin menambahkan data, Anda selalu menggunakan variabel pointer yang baru, Anda akan membutuhkan banyak sekali pointer. Oleh karena itu, ada baiknya jika Anda hanya menggunakan satu variabel pointer saja untuk menyimpan banyak data dengan metode yang kita sebut Linked List. Jika diterjemahkan, ini berarti satu daftar isi yang saling berhubungan. Untuk lebih jelasnya, perhatikan gambar di bawah ini:



Pada gambar di atas tampak bahwa sebuah data terletak pada sebuah lokasi memori area. Tempat yang disediakan pada satu area memori tertentu untuk menyimpan data dikenal dengan sebutan node/simpul. Setiap node memiliki pointer yang menunjuk ke simpul berikutnya sehingga terbentuk satu untaian, degan demikian hanya diperlukan sebuah variabel pointer. Susunan berupa untaian semacam ini disebut Single Linked List (NULL memilik nilai khusus yang artinya tidak menunjuk ke mana-mana. Biasanya Linked List pada titik akhirnya akan menunjuk ke NULL).

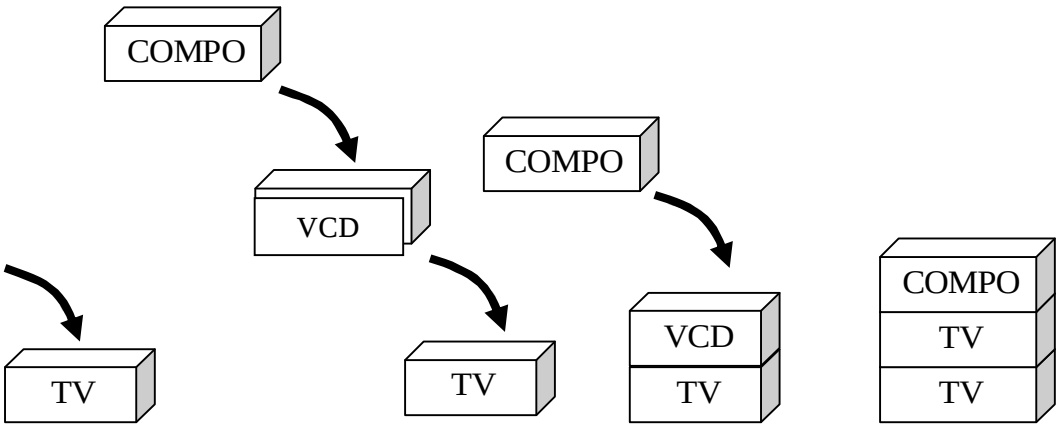
Pembuatan Single Linked List dapat menggunakan 2 metode:

- LIFO (Last In First Out), aplikasinya : Stack (Tumpukan)
- FIFO (First In First Out), aplikasinya : Queue (Antrean)

LIFO (Last In First Out)

Lifo adalah suatu metode pembuatan Linked List di mana data yang masuk paling akhir adalah data yang keluar paling awal. Hal ini dapat di analogikan (dalam kehidupan sehari-hari) dengan saat Anda menumpuk barang seperti digambarkan dibawah ini.

Pembuatan sebuah simpul dalam suatu linked list seperti digambarkan dibawah ii, disebutkan istilah INSERT, Jika linked list dibuat dengan metode LIFO, terjadi penambahan / Insert simpul di belakang.



Keadaan mula-mula adalah kosong

Setelah ditumpuk

Gambar. Ilustrasi Single Linked List dengan metode LIFO

FIFO (Fisrt In Fisrt Out)

FIFO adalah suatu metode pembuatan Linked List di mana data yang masuk paling awal adalah data yang keluar paling awal juga. Hal ini dapat di analogikan (dalam kehidupan sehari-hari), misalnya saat sekelompok orang yang datang (ENQUEUE) mengantri hendak membeli tiket di loket.

Jika linked list dibuat dengan metode FIFO, terjadi penambahan / Insert simpul didepan.

5.2. Operasi Pada Single Linked List

Insert

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

Procedure dan Function Linked List Lainnya

Selain fungsi insert di atas, pada linked list juga terdapat fungsi-fungsi lainnya. Di bawah ini diberikan fungsi-fungsi umum dalam aplikasi linked list.

Konstruktor

Fungsi ini membuat sebuah linked list yang baru dan masih kosong.

IsEmpty

Fungsi ini menentukan apakah linked list kosong atau tidak.

Find First

Fungsi ini mencari elemen pertama dari linked list

Find Next

Fungsi ini mencari elemen sesudah elemen yang ditunjuk now.

Retrieve

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

Update

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

Delete Now

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

Delete Head

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

Clear

Fungsi ini menghapus linked list yang sudah ada. Fungsi ini wajib dilakukan bila anda ingin mengakhiri program yang menggunakan linked list. Jika anda melakukannya,

mfachrz@gmail.com

data-data uang dialokasikan ke memori pada program sebelumnya akan tetap tertiinggal di dalam memori.

5.3. Double Linked List

Salah satu kelemahan single linked list adalah pointer (penunjuk) hanya dapat bergerak satu arah saja, maju/ mundur, atau kanan/kiri sehingga pencarian data pada single linked list hanya dapat bergerak dalam satu arah saja. Untuk mengatasi kelemahan tersebut, anda dapat menggunakan metode double linked list. Linked list ini dikenal dengan nama Linked list berpointer Ganda atau Double Linked List.

Operasi –operasi pada Double Linked List

Insert Tail

Fungsi insert tail berguna untuk menambah simpul di belakang (sebelah kanan) pada sebuah linked list.

Insert Head

Sesuai dengan namanya, fungsi Insert Head berguna untuk menambah simpul di depan (sebelah kiri). Fungsi ini tidak berada jauh dengan fungsi Insert Tail yang telah dijelaskan sebelumnya.

Delete Tail

Fungsi Delete Tail berguna untuk menghapus simpul dari belakang. Fungsi ini merupakan kebalikan dari fungsi Insert Tail yang menambah simpul dibelakang. Fungsi Delete Tail akan mengarahkan Now kepada Tail dan kemudian memanggil fungsi Delete Now.

Delete Head

Fungsi Delete Head merupakan kebalikan dari fungsi Delete Tail yang menghapus simpul dari belakang, sedangkan Delete Head akan menghapus simpul dari depan (sebelah kiri). Fungsi Delete Head akan mengarahkan Now kepada Head dan kemudianm memanggil fungsi Delete Now.

Delete Now

Fungsi Delete Now berguna untuk menghapus simpul pada posisi yang sedang ditunjuk oleh Now.

Circular Double Linked List

Ini adalah double linked list yang simpul terakhirnya menunjuk ke simpul terakhirnya menunjuk ke simpul awalnya menunjuk ke simpul akhir sehingga membentuk suatu lingkaran.

Operasi-operasi pada Circular Double Linked List

Insert Tail

Fungsi insert Tail berguna untuk menambah simpul di belakang (sebelah kanan) pada sebuah circular double linked list.

Insert Head

Sesuai dengan namanya, fungsi Insert Head berguna untuk menambah simpul di depan (sebelah kiri). Fungsi ini tidak berbeda jauh dengan fungsi Insert tail yang telah dijelaskan sebelumnya.

Delete Tail

Fungsi Delete Tail berguna untuk menghapus simpul dari belakang. Fungsi ini kebalikan dari fungsi Insert Tail yang menambah simpul dibelakang.

Delete Head

Fungsi Delete Head merupakan kebalikan dari fungsi Delete Tail yang menghapus simbul dari belakang. Delete Head akan menghapus simpul dari depan (sebelah kiri).

Delete Now

Fungsi Delete Now, sesuai dengan namanya, berguna untuk menghapus simpul pada posisi yang diinginkan.

Contoh Program:

1.

```
//---------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>

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

void CiptaSenarai(NODPTR *s)
{
        *s = NULL;
}
NODPTR NodBaru(int m)
{
        NODPTR n;
        n = (NODPTR) malloc(sizeof(NOD));
        if (n != NULL)
        {
            n -> data = m;
            n -> next = NULL;
        }

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


mfachrz@gmail.com

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

//-----------------------------------------------------
int main()
{
        NODPTR pel;
        NODPTR n;
        int i,k, nilai;

        CiptaSenarai(&pel);
        printf("Masukan banyak data = ");
        scanf("%d",&k);
        for (i=1;i<=k;i++)
        {
            printf("Masukan data senarai ke-%d= ",i);
            scanf("%d",&nilai);
            n = NodBaru(nilai);
            SisipSenarai(&pel, n, NULL);
        }
        CetakSenarai(pel);
        return 0;

}
//-----------------------------------------------------
```

Output:

```
Masukan banyak data = 5
Masukan data senarai ke-1= 5
Masukan data senarai ke-2= 6
Masukan data senarai ke-3= 3
Masukan data senarai ke-4= 8
Masukan data senarai ke-5= 7
7 --> 8 --> 3 --> 6 --> 5 --> NULL
```

2. Pengurutan data secara menaik dan menurun - Menggunakan software (Borland C++ Builder 6).

```
#include <vcl.h>
#pragma hdrstop
#include <stdio>
#include <conio>
#include <list>
using namespace std;
#pragma argsused
int main(int argc, char* argv[])
{
        int n;
        int i;
        list<int> naik,turun;

        printf("Jumlah data=");scanf("%d",&n);
        int data[200];
        for(i=0;i<n;i++)
        {
                printf("Data ke-%i=",i+1);scanf("%d",&data[i]);
        }

        printf ("\n\n");
        //output data awal
        for(i=0;i<n;i++)
        {
                 printf("%d ",data[i]);naik.push_front(data[i]);
        }

        printf ("\n\n");
        //output data awal
        for(i=0;i<n;i++)
        {
                 printf("%d ",data[i]);naik.push_front(data[i]);
        }

        printf ("\n\n");
        naik.sort();
```


mfachrz@gmail.com

```
        //output data naik
        for(i=0;i<n;i++)
        {
                printf("%d ",naik.front());turun.push_back(data[i]);
                naik.pop_front();
        }

        printf("\n\n");
        turun.sort();
        //output data turun
        for(i=0;i<n;i++)
        {
                printf("%d ",turun.back());
                turun.pop_back();
        }

        getch();
        return 0;
}
```

Output:

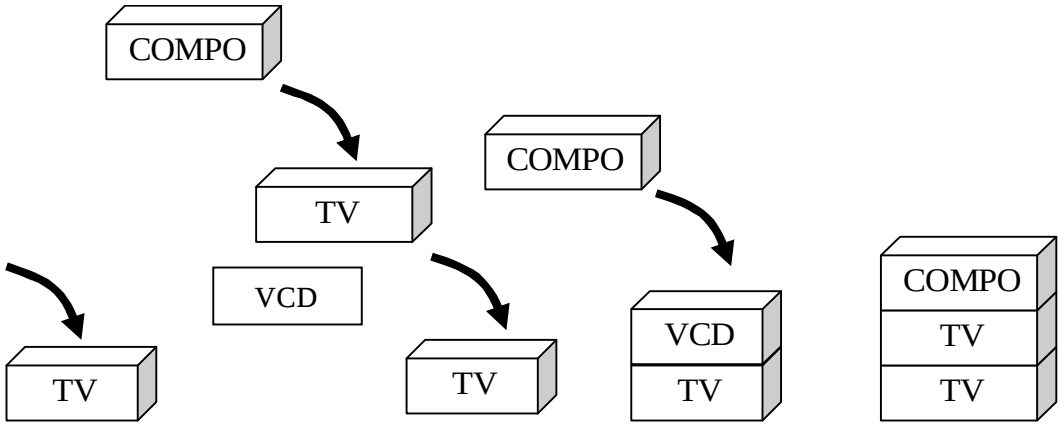
```
Jumlah data=6
Data ke-1=4
Data ke-2=5
Data ke-3=2
Data ke-4=6
Data ke-5=7
Data ke-6=8


4 5 2 6 7 8

2 4 5 6 7 8

8 7 6 5 4 2
```


mfachrz@gmail.com

Bab 6. STACK

6.1. Definisi Stack

Stack adalah suatu tumpukan dari benda. Konsep utamanya adalah LIFO (Last In First Out), benda yang terakhir masuk dalam stack akan menjadi benda pertama yang dikeluarkan dari stack.



Keadaan mula-mula adalah kosong

Setelah ditumpuk

Pada gambar diatas, jika kita ingin mengambil sesuatu dari tumpukan maka kita harus mengambil benda paling atas dahulu, yakni compo. Misalnya jika VCD langsung diambil, compo akan jatuh. Prinsip stack ini bias diterapkan dalam pemrograman. Di C++, ada dua cara penerapan prinsip stack, yakni dengan array dan linked list. Setidaknya stack haruslah memiliki operasi-operasi sebagai berikut.

Push	Untuk menambahkan item pada tumpukan paling atas
Pop	Untuk mengambil item teratas
Clear	Untuk mengosongkan stack
IsEmpty	Untuk memeriksa apakah stack kosong

mfachrz@gmail.com

IsFull	Untuk memeriksa apakah stack sudah penuh

Dalam bab ini penjelasan mengenai stack akan menggunakan kelas stack. Kelima operasi stack diatas akan dideklarasikan secara abstrak dalam kelas ini, sedangkan kelas turunan dari stack akan mengimplementasikan operasi-operasi tersebut.

6.2. Stack dengan Array

Sesuai dengan sifat stack, pengambilan / penghapusan delemen dalam stack harus dimulai dari elemen teratas.

Operasi-operasi pada Stcak dengan Array

Konstruktor

Fungsi ini membuat sebuah stack baru yang masih kosong. Konsepnya adalah bahwa Top menunjukkan elemen stack teratas. Jika Top bernilai -1, berarti tumpukan kosong.

IsFul

Fungsi ini memeriksa apakah stack yang ada sudah penuh. Stack penuh jika stack penuh jika puncak stack terdapat tepat dibawah jumlah maksimum yang dapat ditampung stack atau dengan kata lain Top = MAX_STACK -1.

Push

Fungsi ini menambahkan sebuah elemen ke dalam stack dan tidak bias dilakukan lagi jika stack sudah penuh.

IsEmpty

Fungsi menentukan apakah stack kosong atau tidak. Tanda bahwa stack kosong adalah Top bernilai kurang dari nol.

Pop

Fungsi ini mengambil elemen teratas dari stack dengan syarat stack tidak boleh kosong.

Clear

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

6.3. Double Stack dengan Array

Metode ini adalah teknik khusus yang dikembangkan untuk menghemat pemakaian memori dalam pembuatan dua stack dengan array. Intinya adalah penggunaan hanya sebuah array untuk menampung dua stack.

Tampak jelas bahwa sebuah array dapat dibagi untuk dua stack, stack 1 bergerak ke atas dan stack 2 bergerak ke bawah. Jika Top1 (elemen teratas dari Stack 1) bertemu dengan Top 2 (elemen teratas dari Stack 2) maka double stack telah penuh.

Implementasi double stack dengan array adalah dengan memanfaatkan operasi-operasi yang tidak berbeda jauh dengan operasi single stack dengan array.

Operasi-operasi Double Stack Array

Konstruktor

Fungsi ini membuat stack baru yang masih kosong. Top[0] diset dengan -1 dan Top[1] diset dengan MAX_STACK.

IsFull

Fungsi ini memeriksa apakah double stack sudah penuh. Stack dianggap penuh jika Top[0] dan Top[1] bersentuhan sehingga stack tida memiliki ruang kosong. Dengan kata lain, (Top[0] + 1) $\geq$ Top[1].

Push

Fungsi ini memasukkan sebuah elemen ke salah satu stack.

IsEmpty

Fungsi memeriksa apakah stack pertama atau stack kedua kosong. Stack pertama dianggap kosong jika puncak stack bernilai kurang dari nol, sedangkan stack kedua dianggap kosong jika puncak stack sama atau melebihi MAX_STACK.

Pop

Fungsi ini mengeluarkan elemen teratas dari salah satu stack

Clear

Fungsi ini mengosongkan salah satu stack.

6.4. Stack dengan Single Linked List

Selain implementasi stack dengan array seperti telah dijelasnkan sebelumnya, ada cara lain untuk mengimplementasi stack dalam C++, yakni dengan single linked list. Keunggulannya dibandingkan array tebtu saja adalah penggunaan alokasi memori yang dinamis sehingga menghindari pemborosan memori. Misalnya saja pada stack dengan array disediakan tempat untuk stack berisi 150 elemen, sementara ketika dipakai oleh user stack hanya diisi 50 elemen, maka telah terjadi pemborosan memori untuk sisa 100 elemen, yang tak terpakai. Dengan penggunaan linked list maka tempat yang disediakan akan sesuai dengan banyaknya elemen yang mengisi stack. Oleh karena itu pula dalam stack dengan linked list tidak ada istilah full, sebab biasanya program tidak menentukan jumlah elemen stack yang mungkin ada (kecuali jika sudah dibatasi oleh

pembuatnya). Namun demikian sebenarnya stack ini pun memiliki batas kapasitas, yakni dibatasi oleh jumlah memori yang tersedia.

Operasi-operasi untuk Stack dengan Linked List

Konstruktor

Fungsi ini membuat stack baru yang kosong. Stack adalah kosong jika Top tidak menunjuk apa pun (bernilai NULL).

IsEmpty

Fungsi memeriksa apakah stack yang adamasih kosong.

Push

Fungsi memasukkan elemen baru ke dalam stack. Push di sini mirip dengan insert dalam single linked list biasa.

Pop

Fungsi ini mengeluarkan elemen teratas dari stack.

Clear

Fungsi ini akan menghapus stack yang ada.

Contoh Program:

1. Menggunakan Borland C++ Builder 6

mfachrz@gmail.com

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

int main( )
{
        stack<int> S;

        for(int i=0; i<5; i++)
                S.push(i);

        cout << "Contents:";

        do
        {
                cout << ' ' << S.top();
                S.pop();
        } while( !S.empty());
        cout << '\n';

        getch();
        return 0;
}
```

Output:

```
Contents: 4 3 2 1 0
```

2.

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

class Linked_list_Stack
   {
     private:
    struct node
       {
     int data;

     node *next;
       };
```


mfachrz@gmail.com

```
    node *top;
    node *entry;
    node *print;
    node *bottom;
    node *last_entry;
    node *second_last_entry;

      public:
    Linked_list_Stack( );

    void pop( );
    void push( );
    void print_list( );
    void show_working( );
   };


Linked_list_Stack::Linked_list_Stack ( )
   {
      top=NULL;
      bottom=NULL;
   }

/*************************************************************************/
//-------------------------------  push( )  -------------------------------//
/*************************************************************************/

void Linked_list_Stack::push( )
   {
      int num;

      cout<<"\n\n\n\n\n\t Masukkan angka pada Stack : ";
      cin>>num;

      entry=new node;

      if(bottom==NULL)
    {
       entry->data=num;
       entry->next=NULL;
       bottom=entry;
       top=entry;
    }

      else
    {
       entry->data=num;
       entry->next=NULL;
       top->next=entry;
       top=entry;
    }
```


mfachrz@gmail.com

```
        cout<<"\n\n\t *** "<<num<<" sudah masuk dalam Stack."<<endl;
        cout<<"\n\n\n\t\t Pres any key to return to Menu. ";

        getch( );
    }

/*****************************************************************************/
//---------------------------  pop( )  ----------------------------------//
/*****************************************************************************/

void Linked_list_Stack::pop( )
   {
      if(bottom==NULL)
    cout<<"\n\n\n\t ***  Error : Stack is empty. \n"<<endl;

      else
    {
       for(last_entry=bottom;last_entry->next!=NULL;
                   last_entry=last_entry->next)
     second_last_entry=last_entry;

       if(top==bottom)
     bottom=NULL;

       int poped_element=top->data;

       delete top;

       top=second_last_entry;
       top->next=NULL;

       cout<<"\n\n\n\t *** "<<poped_element<<" sudah diambil dari Stack."<<endl;
    }

      cout<<"\n\n\n\t\t Pres any key to return to Menu. ";

      getch( );
   }

/*****************************************************************************/
//-------------------------------  print_list( )  -------------------------//
/*****************************************************************************/

void Linked_list_Stack::print_list( )
   {
      print=bottom;

      if(print!=NULL)
    cout<<"\n\n\n\n\n\t Angka-angka yang ada di Stack adalah : \n"<<endl;

      else
    cout<<"\n\n\n\n\n\t *** Tidak ada yang ditampilkan. "<<endl;
```


mfachrz@gmail.com

```
        while(print!=NULL)
     {
        cout<<"\t "<<print->data<<endl;

        print=print->next;
     }

       cout<<"\n\n\n\t\t Pres any key to return to Menu. ";

       getch( );
    }

/*************************************************************************/
//---------------------------  show_working( )  ---------------------------//
/*************************************************************************/
```

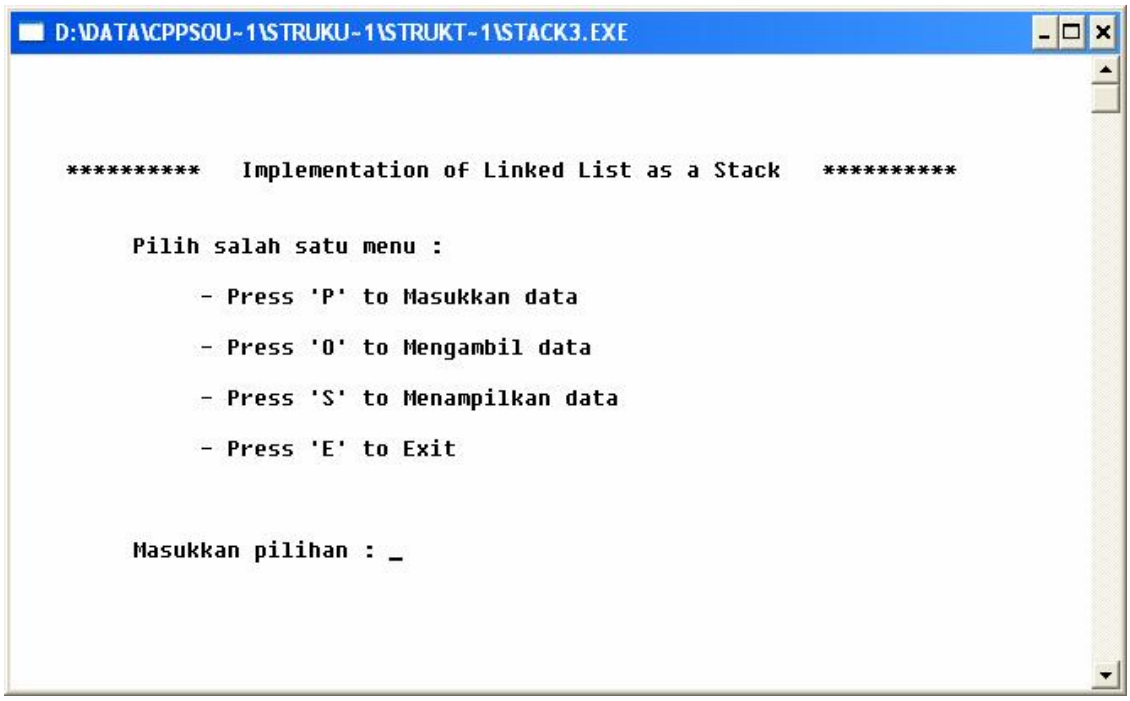
```
void Linked_list_Stack::show_working( )
   {
      char Key=NULL;

      do
     {
       clrscr( );

       gotoxy(5,5);
       cout<<"**********   Implementation of Linked List as a Stack   **********"<<endl;

       gotoxy(10,8);
       cout<<"Pilih salah satu menu :"<<endl;

       gotoxy(15,10);
       cout<<"- Press \'P\' to Masukkan data"<<endl;

         gotoxy(15,12);
         cout<<"- Press \'O\' to Mengambil data"<<endl;

         gotoxy(15,14);
         cout<<"- Press \'S\' to Menampilkan data"<<endl;

         gotoxy(15,16);
         cout<<"- Press \'E\' to Exit"<<endl;

         Input:

         gotoxy(10,20);
         cout<<"                          ";

         gotoxy(10,20);
         cout<<"Masukkan pilihan : ";

         Key=getche( );
```


mfachrz@gmail.com

```
        if(int(Key)==27 || Key=='e' || Key=='E')
      break;

        else if(Key=='p' || Key=='P')
      push( );

        else if(Key=='o' || Key=='O')
      pop( );

        else if(Key=='s' || Key=='S')
      print_list( );

        else
      goto Input;
     }
       while(1);
   }
/*****************************************************************************/
/*****************************************************************************/
//--------------------------------  main( )  ------------------------------//
/*****************************************************************************/
/*****************************************************************************/

int main( )
   {
      Linked_list_Stack  obj;

      obj.show_working( );

      return 0;
   }

/*****************************************************************************/
/*****************************************************************************/
//--------------------------------  THE END  ------------------------------//
/*****************************************************************************/
/*****************************************************************************/
```

Output :

mfachrz@gmail.com



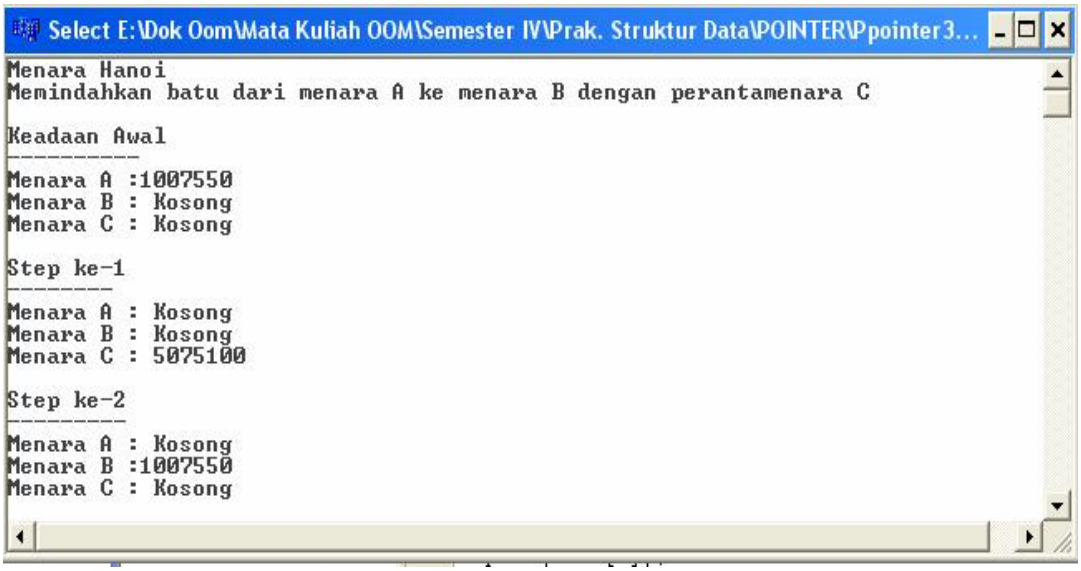
Latihan: Kasus Menara Hanoi – Menggunakan Borland C++ Builder 6

Memindahkan batu dari A ke menara B dengan perantara C dengan jumlah data = 3 (50, 75, 100). Step program:

1. Pindahkan batu 50 dari A ke C
2. Pindahkan batu 75 dari A ke C
3. Pindahkan batu 100 dari A ke B
4. Pindahkan batu 75 dari C ke B
5. Pindahkan batu 50 dari C ke B

Output:

mfachrz@gmail.com



mfachrz@gmail.com

Bab 7. QUEUE

7.1. Definisi Queue

Jika diartikan secara harafiah, queue berarti antrian, queue merupakan salah satu contoh aplikasi dari pembuatan double linked list yang cukup sering kita temui dalam kehiduypan sehari-hari, misalnya saat Anda mengantri di loket untuk membeli tiket.

Istilah yang cukup sering dipakai seseorang masuk dalam sebuah antrian adalah enqueue. Dalam suatu antrian, yang dating terlebih dahulu akan dilayani lebih dahulu.

Istilah yang sering dipakai bila seseorang keluar dari antrian adalah dequeue.

Walaupun berbeda implementasi, struktur data queue setidaknya harus memiliki operasi-operasi sebagai berikut :

EnQueue	Memasukkan data ke dalam antrian
DeQueue	Mengeluarkan data terdepan dari antrian
Clear	Menghapus seluruh antrian
IsEmpty	Memeriksa apakah antrian kosong
IsFull	Memeriksa apakah antrian penuh

7.2. Implementasi Queue dengan Linear Array

Linear Array

Linear array adalah suatu array yang dibuat seakan-akan merupakan suatu garis lurus dengan satu pintu masuk dan satu pintu keluar.

Berikut ini diberikan deklarasi kelas Queue Linear sebagai implementasi dari Queue menggunakan linear array. Dalam prakteknya, anda dapat menggantinya sesuai dengan kebutuhan Anda. Data diakses dengan field data, sedangkan indeks item pertama dan terakhir disimpan dalam field Head dan Tail. Konstruktor akan menginisialisasikan

mfachrz@gmail.com

nilai Head dan Tail dengan -1 untuk menunjukkan bahwa antrian masih kosong dan mengalokasikan data sebanyak MAX_QUEUE yang ditunjuk oleh Data. Destruktor akan mengosongkan antrian kembali dan mendealokasikan memori yang digunakan oleh antrian.

Operasi-operasi Queue dengan Linear Array

Konstruktor

Konstruktor berguna untuk menciptakan queue yang baru dan kosong dengan memberikan nilai awal (head) dan nilai akhir (tail) dengan -1. Nilai -1 menunjukkan bahwa queue (antrian) masih kosong.

IsEmpty

Fungsi IsEmpty berguna untuk mengecek apakah queue masih kosong atau sudah berisi data. hal ini dilakukan dengan mengecek apakah tail bernilai -1 atau tidak. Nilai -1 menandakan bahwa queue masih kosong.

IsFull

Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bisa menampung data dengan cara mengecek apakah nilai tail sudah sama dengan jumlah maksimal queue. Jika nilai keduanya sama, berarti queue sudah penuh.

EnQueue

Fungsi EnQueue berguna untuk memasukkan sebuah elemen dalam queue.

DeQueue

Fungsi DeQueue berguna untuk mengambil sebuah elemen dari queue. Operasi ini sering disebut juga serve. Hal ini dilakukan dengan cara memindahkan sejauh satu

langkah ke posisi di depannya sehingga otomatis elemen yang paling depan akan tertimpa dengan elemen yang terletak di belakangnya.

Clear

Fungsi Clear berguna untuk menghapus semua lemen dalam queue dengan jalan mengeluarkan semua elemen tersebut satu per satu hingga queue kosong dengan memanfaatkan fungsi DEQueue.

7.3. Implementasi Queue dengan Circular Array

Circular Array

Circular array adalah suatu array yang dibuat seakan-akan merupakan sebuah lingkaran dengan titik awal (head) dan titik akhir (tail) saling bersebelahan jika array tersebut masih kosong.

Posisi head dan tail pada gambar diatas adalah bebas asalkan saling bersebelahan. Berikut ini diberikan deklarasi kelas Queue Circular sebagai implementasi circular array. Dalam prakteknya, Anda dapat menggantikanny sesuai dengan kebutuhan Anda. Data diakses dengan field data, sedangkan indeks itemn pertama dan terakhir disimpan dalam field Head dan Tail. Konstruktor akan menginisialisasi nilai Head dan Tail dengan 0 dan MAX-QUEUE-1 untuk menunjukkan bahwa antrian masih kosong dan mengalokasikan data sebanyak MAX-QUEUE yang ditunjuk oleh Data. destruktor akan mengosongkan antrian kembali dan mendealokasikan memori yang digunakan oleh antrian.

Operasi-operasi Queue dengan Circular Array

Konstruktor

Konstruktor berguna untuk menciptakan queue yang baru dan kosong, yaitu dengan cara memberikan nilai awal (head) dengan nol (0) dan nilai akhir (tail) dengan jumlah maksimal data yang akan di tampung/array.

IsEmpty

Fungsi IsEmpty berguna untuk mengecek apakah Queue masih kosong atau sudah berisi. Hal ini dilakukan dengan mengecek apakah tail masih terletak bersebelahan dengan head dan tail lebih besar dari head atau tidak. Jika benar, maka queue masih kosong.

IsFull

Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bias menampung data dengan cara mengecek apakah tempat yang masih kosong tinggal satu atau tidak (untuk membedakan dengan empty dimana semua tempat kosong). Jika benar berarti queue penuh.

EnQueue

Fungsi EnQueue berguna untuk memasukkan sebuah elemen ke dalam queue tail dan head mula-mula bernilai nol (0).

DeQueue

DeQueue berguna untuk mengambil sebuah elemen dari queue. Hal ini dilakukan dengan cara memindahkan posisi head satu langkah ke belakang.

7.4. Implementasi Queue dengan Double Linked List

Selain menggunakan array, queue juga dapat dibuat dengan linked list. Metode linked list yang digunakan adalah double linked list.

Operasi-operasi Queue dengan Double Linked List

Konstruktor

Konstruktor berguna untuk menciptakan queue yang baru dan kosong, yaitu dengan mengarahkan pointer head dan tail kepada NULL.

IsEmpty

Fungsi IsEmpty berguna untuk mengecek apakah queue masih kosong atau sudah berisi data. Hal ini dilakukan dengan mengecek apakah head masih menunjukkan pada Null atau tidak. Jika benar berarti queue masih kosong.

IsFull

Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bias menampung data dengan cara mengecek apakah Jumlah Queue sudah sama dengan MAX_QUEUE atau belum. Jika benar maka queue sudah penuh.

EnQueue

Fungsi EnQueue berguna untuk memasukkan sebuah elemen ke dalam queue (head dan tail mula-mula meunjukkan ke NULL).

DeQueue

Procedure DeQueue berguna untuk mengambil sebuah elemen dari queue. Hal ini dilakukan dengan cara menghapus satu simpul yang terletak paling depan (head).

Contoh Program:

1.

```
//Program : queue1
 # include <iostream.h>
 # include    <conio.h>

 class Linked_list_Queue
    {
       private:
     struct node
        {
      int data;

      node *next;
        };

     node *rear;
     node *entry;
     node *print;
     node *front;

       public:
     Linked_list_Queue( );

     void Delete( );
     void Insert( );
     void print_list( );
     void show_working( );
    };

 Linked_list_Queue::Linked_list_Queue ( )
    {
       rear=NULL;
       front=NULL;
    }

 /*************************************************************************/
 //-----------------------------  Insert( )  -----------------------------//
 /*************************************************************************/

 void Linked_list_Queue::Insert( )
    {
       int num;

       cout<<"\n\n\n\n\n\t Masukkan angka dalam Queue : ";
       cin>>num;

       entry=new node;

       if(rear==NULL)
     {
        entry->data=num;
        entry->next=NULL;
        rear=entry;
        front=rear;
     }
```


mfachrz@gmail.com

```
      else
    {
       entry->data=num;
       entry->next=NULL;
       rear->next=entry;
       rear=entry;
    }

      cout<<"\n\n\t *** "<<num<<" sudah masuk dalam Queue."<<endl;
      cout<<"\n\n\n\t\t Pres any key to return to Menu. ";

      getch( );
   }

/*****************************************************************************/
//------------------------------  Delete( )  ------------------------------//
/*****************************************************************************/

void Linked_list_Queue::Delete( )
   {
      if(front==NULL)
    cout<<"\n\n\n\t ***  Error : Queue is empty. \n"<<endl;

      else
    {
       int deleted_element=front->data;

       node *temp;

       temp=front;
       front=front->next;

       delete temp;

       cout<<"\n\n\n\t *** "<<deleted_element<<" dihapus dari Queue."<<endl;
    }

      cout<<"\n\n\n\t\t Pres any key to return to Menu. ";

      getch( );
   }

/*****************************************************************************/
//---------------------------  print_list( )  ---------------------------//
/*****************************************************************************/

void Linked_list_Queue::print_list( )
   {
      print=front;

      if(print!=NULL)
    cout<<"\n\n\n\n\n\t Angka-angka yang ada dalam Queue adalah : \n"<<endl;
```


mfachrz@gmail.com

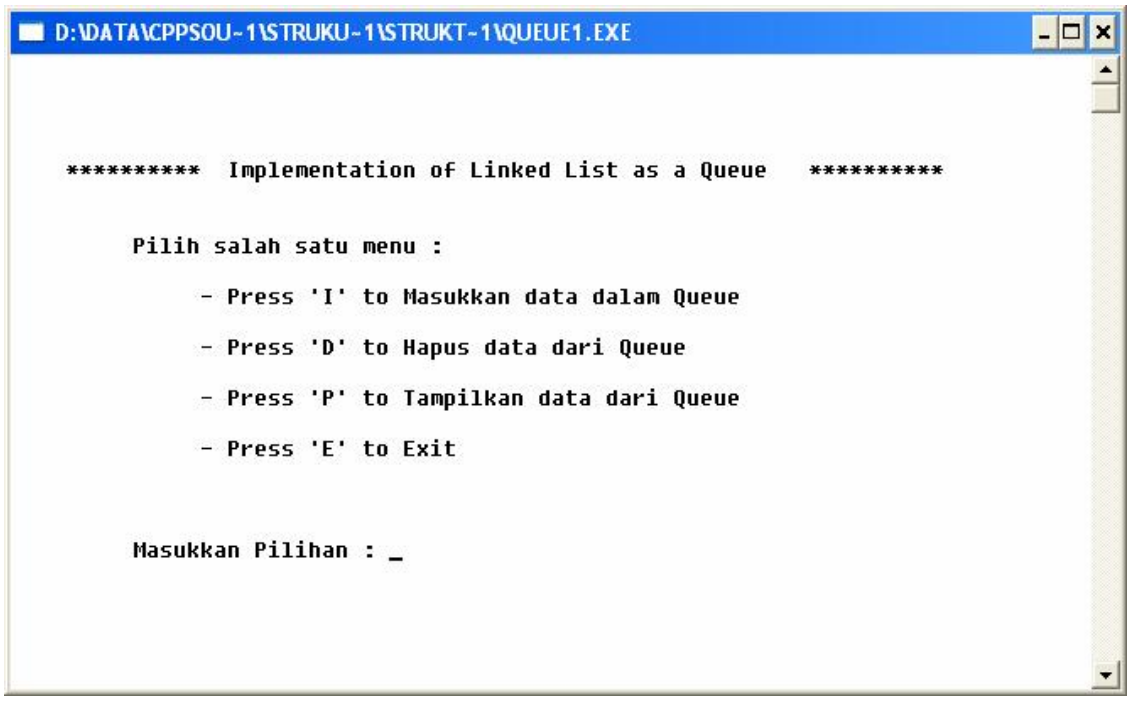
```
      else
    cout<<"\n\n\n\n\n\t *** Tidak ada yang ditampilkan. "<<endl;

      while(print!=NULL)
    {
       cout<<"\t "<<print->data<<endl;

       print=print->next;
    }

      cout<<"\n\n\n\t\t Pres any key to return to Menu. ";

      getch( );
   }

/*****************************************************************************/
//---------------------------  show_working( )  --------------------------//
/*****************************************************************************/

void Linked_list_Queue ::show_working( )
   {
     char Key=NULL;

     do
    {
       clrscr( );

       gotoxy(5,5);
       cout<<"**********  Implementation of Linked List as a Queue   **********"<<endl;

       gotoxy(10,8);
       cout<<"Pilih salah satu menu :"<<endl;

        gotoxy(15,10);
        cout<<"- Press \'I\' to Masukkan data dalam Queue"<<endl;

        gotoxy(15,12);
        cout<<"- Press \'D\' to Hapus data dari Queue"<<endl;

        gotoxy(15,14);
        cout<<"- Press \'P\' to Tampilkan data dari Queue"<<endl;

        gotoxy(15,16);
        cout<<"- Press \'E\' to Exit"<<endl;

        Input:

        gotoxy(10,20);
        cout<<"                          ";

        gotoxy(10,20);
        cout<<"Masukkan Pilihan : ";

        Key=getche( );
```


mfachrz@gmail.com

```
        if(int(Key)==27 || Key=='e' || Key=='E')
      break;

        else if(Key=='i' || Key=='I')
       Insert( );

        else if(Key=='d' || Key=='D')
       Delete( );

        else if(Key=='p' || Key=='P')
       print_list( );

        else
       goto Input;
      }
        while(1);
    }
/*****************************************************************************/
/*****************************************************************************/
//-------------------------------  main( )  -------------------------------//
/*****************************************************************************/
/*****************************************************************************/

int main( )
   {
      Linked_list_Queue  obj;

      obj.show_working( );

      return 0;
   }

/*****************************************************************************/
/*****************************************************************************/
//-------------------------------  THE END  -------------------------------//
/*****************************************************************************/
/*****************************************************************************/
```

Output:

mfachrz@gmail.com



2. Menggunakan software : Borland C++ Builder 6.

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

int main( )
{
        queue<int> Q;

        for( int i = 0; i < 5; i++ )
                Q.push(i);

        cout << "Contents:";
```


mfachrz@gmail.com

```
        do
        {
                cout << ' ' << Q.front();
                Q.pop();
        } while( !Q.empty() );

        cout << '\n';

        getch();
        return 0;
}
```

Output:

```
Contents: 0 1 2 3 4
```


mfachrz@gmail.com

Bab 8. TREE

8.1. Definisi Tree

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hierarkis (hubungan one to many) antara elemen-elemen. Tree bias didefinisikan sebagai kumpulan simpul/node dengan elemen khusus yang disebut Root. Notde lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lain (disebut Subtree). Untuk lebih jelasnya, di bawah akan diuraikan istilah-istilah umum dalam tree.

Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada dibawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendant	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama dengan suatu node
Subtree	Bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan / level dalam suatu tree
Root	Satu-satunya node khusus dalam tree yang tak punyak predecessor
Leaf	Node-node dalam tree yang tak memiliki successor
Degree	Banyaknya child yang dimiliki suatu node

mfachrz@gmail.com

8.2. Jenis-jenis Tree

Binary Tree

Binary Tree adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Sesuai dengan definisi tersebut tiap node dalam binary tree hanya boleh memiliki paling banyak dua child.

Jenis- Jenis Binary Tree :

Full Binary Tree

Jenis binary tree ini tiap nodenya (kecuali leaf) memiliki dua child dan tiap subtree harus mempunyai panjang path yang sama.

Complete Binary Tree

Jenis ini mirip dengan Full Binary Tree, namun tiap subtree boleh memiliki panjang path yang berbeda dan setiap node kecuali leaf hanya boleh memiliki 2 child.

Skewed Binary Tree

Skewed Binary Tree adalah Binary Tree yang semua nodenya (kecuali leaf) hanya memiliki satu child.

Implementasi Binary Tree

Binary tree dapat diimplementasikan dalam C++ dengan menggunakan double linkedlist.

8.3. Operasi-operasi pada Binary Tree

Create	Membentuk binary tree baru yang masih kosong
Clear	Mengosongkan binary tree yang sudah ada
Empty	Function untuk memeriksa apakah binary tree masih kosong
Insert	Memasukkan sebuah node ke dalam tree. Ada tiga pilihan insert : sebagai root, left child, atau right child. Khusus insert sebagai root, tree harus dalam keadaan kosong
Find	Mencari root, parent, left child, atau right child dari suatu node. (tree tidak boleh kosong).
Update	Mengubah isi dari node yang ditunjuk oleh pointer curret (Tree tidak boleh kosong)
Retrieve	Mengetahui isi dari node yang ditunjuk oleh pointer current (Tree tidak boleh kosong)
DeleteSub	Menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk current. Tree tidak boleh kosong. Setelah itu, pointer current dakan berpindah ke parent dari node yang dihapus.
Characteristic	Mengetahui karakteristik dari suatu tree, yakni: size, height, serta average length. Tree tidak boleh kosong.
Traverse	Mengunjungi seluruh node-node pada tree, masing-masing sekali. Hasilnya adalah urutan informasi secara linear yang tersimpan dalam tree. Ada tiga cara traverse,yaitu PreOrder, InOrder, dan PostOrder.

Langkah-langkah Tranverse :

- PreOrder : cetak isi node yang dikunjungi, kunjungi Left Child, kunjungi Right Child
- InOrder : kunjungi Left Child, cetak isi node yang dikunjungi, kunjungi Right Child
- PostOrder : kunjungi Left Child, kunjungi Right Child cetak isi node yang dikunjungi.

8.4. Binary Search Tree

Binary Tree ini memiliki sifat dimana semua left child harus lebih kecil dari pada right child dan parentnya. Semua right child juga harus lebih besar dari left child serta parent-nya. Binary search tree dibuat untuk mengatasi kelemahan pada binary tree biasa, yaitu kesulitan dalam searching / pendarian node tertentu dalam binary tree.

Pada dasarnya operasi dalam Binary Search Tree sama dengan Binary Tree biasa, kecuali pada operasi insert, update, dan delete.

Insert

Pada Binary Search Tree insert dilakukan setelah lokasi yang tepat ditemukan (lokasi tidak ditentukan oleh user sendiri).

Update

Update ini seperti yang ada pada Binary Tree biasa, namun di sini update akan berpengaruh pada posisi node tersebut selanjutnya. Bila update mengakibatkan tree tersebut bukan Binary Search Tree lagi, harus dilakukan perubahan pada tree dengan melakukan rotasi supaya tetap menjadi Binary Search Tree.

Delete

Seperti halnya update, delete dalam Binary Search Tree juga turut mempengaruhi struktur dari tree tersebut.

AVL Tree

AVL Tree adalah Binary Search Tree yang memiliki perbedaan tinggi/ level maksimal 1 antara subtree kiri dan subtree kanan. AVL Tree muncul untuk menyeimbangkan Binary Search Tree. Dengan AVL Tree, waktu pencarian dan bentuk tree dapat dipersingkat dan disederhanakan.

Selain AVL Tree, terdapat pula Height Balanced n Tree, yakni Binary Search Tree yang memiliki perbedaan level antara subtree kiri dan subtree kanan maksimal adalah n sehingga dengan kata lain AVL Tree adalah Height Balanced 1 Tree.

Untuk memudahkan dalam menyeimbangkan tree, digunakan simbol-simbol Bantu :

- (tanda minus) : digunakan apabila Subtree kiri lebih panjang dari Subtree kanan.
+ (tanda plus) : digunakan apabila Subtree kanan lebih panjang dari Subtree kiri.
0 (nol) : digunakan apabila Subtree kiri dan Subtree kanan mempunyai height yang sama.

DAFTAR ISTILAH-ISTILAH

Algoritma	:	Langkah-langkah menyelesaikan suatu masalah yang disusun secara logis dan berurutan
Animasi	:	Gambar yang tampak bergerak, terdiri dari banyak gambar-gambar tunggal (disebut frame) yang ditampilkan satu per satu secara bergantian dengan cepat sehingga objek dalam gambar tampak seolah-olah bergerak.
Array	:	Struktur data yang memiliki banyak elemen di dalamnya, dengan masing-masing elemen memiliki tipe data yang sama.
Clear	:	Menghapus secara keseluruhan, biasanya digunakan sebagai nama fungsi/metode yang bertujuan untuk mengosongkan list atau menghapus keseluruhan elemen.

mfachrz@gmail.com

Console	:	Istilah dalam komputer yang menunjuk pada antarmuka antara pemakai dengan komputer yang berbasiskan teks. Cara kerja konsol sangat sederhana yaitu menggunakan standar input untuk membaca input dari keyboard dan standar output untuk menampilkan teks ke layer monitor.
Data	:	Informasi yang disimpan komputer, dapat berbentuk teks, gambar, suara, video, dan sebagainya.
Delete	:	Menghapus sebuah elemen, biasanya digunakan sebagai nama fungsi/metode yang bertujuan untuk menghapus sebuah elemen dalam suatu list/tree
Deret geometric	:	Deretan bilangan yang setiap bilangan merupakan hasil kali bilangan sebelumnya dengan suatu konstanta.
Destruktor	:	Metode khusus dalam sebuah kelas untuk menghapus objek hasil instansiasi kelas tersebut
Dimensi	:	Jumlah indek yang diperlukan untuk menyatakan sebuah elemen dalam array
Elemen	:	Sebuah data tunggal yang paling kecil dari sebuah array atau list. Data tunggal disini tidak perlu data sederhana, melainkan bisa berupa kumpulan data atau list yang lain.
Empty	:	Keadaan di mana list ada dalam keadaan kosong
Fibonacci	:	Barisan bilangan yang setiap bilangan merupakan jumlah dari dua bilangan sebelumnya.
Field	:	Data yang dimiliki oleh sebuah objek
FIFO	:	First In First Out sifat suatu kumpulan data. jika sebuah elemen A dimasukkan lebih dulu dari B maka A harus dikeluarkan dulu dari B
FPB	:	Faktor Persekutuan terbesar, faktor yang paling besar jika sejumlah bilangan memiliki beberapa faktor yang sama.

mfachrz@gmail.com

Full	:	Keadaan di mana list penuh, tidak boleh menerima data lagi
Fungsi	:	Suatu modul atau bagian program yang mengerjakan suatu program tertentu.
Himpunan	:	Kumpulan dari objek-objek, misalnya sebuah himpunan dari buah-buahan dapat terdiri dari pisang, mangga, jambu dll.
Indeks	:	Bilangan yang digunakan untuk menyatakan posisi suatu elemen dalam array atau list.
Input	:	Data masukkan, dalam fungsi berarti parameter yang dimasukkan, sedangkan dalam program secara keseluruhan berarti data yang dimasukkan pemakai, bias melalui parameter program, file maupun lewat keyboard
Insert	:	Memasukkan sebuah elemen baru ke dalam list. Biasanya insert dilakukan baik di tengah-tengah list, awal, maupun di akhir list.
Iterasi	:	Perulangan dengan struktur perulangan, while, do while, maupun for.
Kelas	:	Suatu struktur yang digunakan sebagai template bagi objek-objek yang sama sifatnya.
Kompilasi	:	Proses menerjemahkan bahasa sumber (source code) ke dalam bahasa lain, biasanya bahasa mesin, untuk dapat dijalankan langsung oleh computer melalui system operasi.
Kompiler	:	Program yang mengerjakan kompilasi.
Konstruktor	:	Metode khusus yang dimiliki suatu kelas untuk membentuk suatu objek baru berdasarkan kelas tersebut
Library	:	Kumpulan fungsi, makro, template, dan kelas yang disediakan bersama compiler C++.
LIFO	:	Last In Fisrt Out, sifat kumpulan data, kebalikan dari FIFO
Linked List	:	List yang didesain dengan cara mendefinisikan sebuah elemen

		yang memiliki hubungan atau link dengan elemen lain yang dihubungkan dengan elemen yang lain lagi.
Matriks	:	Dalam matematika berarti kumpulan bilangan yang disusun dalam bentuk kolom dan baris.
Metode	:	Fungsi yang dimiliki suatu objek
Objek	:	Struktur data yang terdiri dari data yang lebih sederhana yang disebut field yang memiliki operasi sendiri untuk menangani data-data yang dimilikinya.
Output	:	Data yang dihasilkan oleh program
Pointer	:	Type data khusus yang pada umumnya berukuran 32 bit yang berfungsi untuk menampung bilangan tertentu yang menunjuk pada lokasi memory tertentu
Pop	:	Mengeluarkan satu elemen dari dalam list dengan cara menyalin data elemen tersebut, kemudian menghapus elemen tersebut dari list biasanya digunakan untuk stack.
Prima	:	Bilangan yang tidak memiliki faktor selain 1 dan bilangan itu sendiri.
Push	:	Memasukkan sebuah elemen baru ke dalam list.
Queue	:	Struktur list dengan sifat FIFO, cara kerjanya seperti antrian manusia.
Record	:	Struktur data yang terdiri dari satu atau lebih elemen yang tipe data bias berbeda.
Rekursi	:	Jenis perulangan yang tidak menggunakan struktur perulangan, tetapi dengan memanggil fungsi yang bersangkutan.
Sort	:	Menyusun elemen-elemen suatu list secara berurutan.
Source Code	:	Program yang ditulis menggunakan bahasa pemrograman tertentu. Kode sumber belum dapat dijalankan oleh komputer

dan perlu menjalani proses kompilasi sehingga dapat dijalankan.

Stack	:	List yang memiliki sifar LIFO. Data yang hendak di keluarkan dari stack haruslah data yang paling terakhir dari stack.
STL	:	Standar Templete Library merupakan kumpulan yang disertakan dalam setiap compiler C++ yang memenuhi standar ANSI C++ yang menyediakan berbagai struktur data, algoritma dan template yang sering dipakai.
Stream	:	Aliran merupakan konsep dalam C++ untuk input dan ouput data tanpa memperdulikan isi data maupun media penampung data tersebut.
Struktur kontrol	:	Struktur yang digunakan untuk mengontrol jalannya program.
Teks	:	Data yang terdiri dari karakter-karakter yang dapat dibaca (huruf bilangan, tanda baca).
Tree	:	Suatu struktur data yang setiap elemen terhubung sedemikian rupa sehingga berbentuk seperti pohon.

Contoh Program:

1.

```
//Program :tree.cpp
#include <stdio.h>
#include <malloc.h>

struct nod {
     struct nod *left;
     char data;
     struct nod *right;
};

typedef struct nod NOD;
typedef NOD POKOK;
```

```
NOD *NodBaru(char item)
{
            NOD *n;

            n = (NOD*) malloc(sizeof(NOD));

            if(n != NULL) {
            n->data = item;
            n->left = NULL;
            n->right = NULL;
            }
      return n;
}
void BinaPokok(POKOK **T)
{
      *T = NULL;
}

typedef enum { FALSE = 0, TRUE = 1} BOOL;

BOOL PokokKosong(POKOK *T)
{
      return((BOOL)(T == NULL));
}
void TambahNod(NOD **p, char item)
{
      NOD *n;
      n = NodBaru(item);

            *p = n;
}

void preOrder(POKOK *T)
{
      if(!PokokKosong(T)) {
            printf("%c ", T->data);
            preOrder(T->left);
            preOrder(T->right);
      }
}

void inOrder(POKOK *T)
{
      if(!PokokKosong(T)) {
            inOrder(T->left);
            printf("%c ", T->data);
            inOrder(T->right);
      }
}
```


mfachrz@gmail.com

```
void postOrder(POKOK *T)
{
     if(!PokokKosong(T)) {
           postOrder(T->left);
           postOrder(T->right);
           printf("%c ", T->data);
     }
}

int main()
{
     POKOK *kelapa;
     char buah;

     BinaPokok(&kelapa);

     TambahNod(&kelapa, buah = 'M');

     TambahNod(&kelapa->left, buah = 'E');

     TambahNod(&kelapa->left->right, buah = 'I');

     TambahNod(&kelapa->right, buah = 'L');

     TambahNod(&kelapa->right->right, buah = 'O');

     TambahNod(&kelapa->right->right->left, buah = 'D');
     printf("Tampilan secara PreOrder: ");
     preOrder(kelapa);

     printf("\nTampilan secara InOrder: ");
     inOrder(kelapa);

     printf("\nTampilan secara PreOrder: ");
     postOrder(kelapa);

     printf("\n\n");

     return 0;
}
```

Output:

```
Tampilan secara PreOrder: M E I L O D
Tampilan secara InOrder: E I M L D O
Tampilan secara PreOrder: I E D O L M
```

REFERENSI

Heriyanto, Imam, Budi Raharjo (2003). ***Pemrograman Borland C++ Builder***. Informatika Bandung..

Indrajit, Richardus Eko. ***Manajemen Sistem Informasi dan Teknologi Informasi***.

Indrajit, Richardus Eko. ***Kajian Strategis Analisa Cost-Benefit Investasi Teknologi Informasi***.

Lidya, Leoni, rinaldi Munir (2002). ***Algoritama dan Pemrograman dalam Bahas Pascal dan C***. Informatika Bandung.

Sanjaya, Dwi (2005). ***Asyiknya Belajar Struktur Data di Planet C++***. Elex Media Komputindo.

Solichin, Achmad (2003). ***Pemrograman Bahasa C dengan Turbo C***. IlmuKomputer.Com.

Wahono, Romi Satria(2003). ***Cepat MahirBahasa***. IlmuKomputer.Com.

mfachrz@gmail.com