DIKTAT KULIAH
Pemrograman Berorientasi Objek

Buku Latihan : C++

Oleh :

Inggriani Liem

Departemen Teknik Informatika

Institut Teknologi Bandung

2003

Kata Pengantar

Buku pelengkap diktat ini merupakan buku untuk latihan implementasi, membaca program dan memahami eksekusi program C++, disesuaikan dengan perkuliahan Algoritma dan Pemrograman dan Struktur Data yang diberikan di perkuliahan sebelumnya.

Karena ditujukan untuk latihan memahami eksekusi dan membaca program, maka program-program yang ada di diktat ini mengandung kesalahan/keanehan jika dieksekusi (dan justru merupakan latihan yang harus diselesaikan dengan hanya membaca).
Oleh karena itu, program-program pada buku kecil ini tidak boleh dipakai mentah-mentah sebagai contoh program yang "baik" dan "benar". Kasus keanehan tersebut merupakan bahan yang dibahas secara lisan di kelas.
Buku latihan ini sangat miskin penjelasan tertulis, namun akan menjadi bahan diskusi yang menarik dan bahan latihan eksekusi program bagi mahasiswa .

Program kecil C++: Setiap class harus diletakkan dalam sebuah direktori!!

Contoh pendefinisian ADT POINT (sebagian)
Catatan : implementasi kelas ini sekaligus me-review feature :

- Overloading operator
- Copy constructor
- Default parameter
- Pemanfaatan Current_Object atau tidak

Contoh ADT Point (hanya sebagian, harus anda lengkapi sesuai dengan teks lengkap yang diberikan)

```
// File : Point.h
// ADT Point (sebagian)
#ifndef _Point_H
#define _Point_H
class Point
{ public:
// Konstruktor
    Point ();
    Point (int, int);

// Destruktor
   ~Point ();
// method GET dan SET
  int GetX ();
  int GetY ();
  void SetX (int);
  void SetY (int);

// Relasional
  int LT (Point P1, Point P2);
  // true (bukan 0) jika P1< P2 : absis dan ordinat  lebih kecil
  // Perhatikan Current_Object tidak dipakai!
  int operator<(Point P1);
  // true jika P1< Current_Object : absis dan ordinat lebih kecil
  // Perhatikan Current_Object dipakai!
// Predikat lain
  int IsOrigin ();
  // true (bukan 0) jika Current_Object adalah (0,0)

// method
  void mirror ();
  Point Mirrorof ();
  void PrintObj ();
  private:                    // atribut ADT : private
  int x;                      //absis
  int y;                      // ordinat
};
#endif
```

Point.cc

(sebagian, harus anda lengkapi sesuai dengan teks lengkap yang diberikan)

```
 //File : point.cc
#include "point.h"
#include <iostream.h>
/* method GET dan SET */
int
Point::GetX ()
{
  return x;
}
int
Point::GetY ()
{
  return y;
}
void
Point::SetX (int NewX)
{
  x = NewX;
}
void
Point::SetY (int NewY)
{
  y = NewY;
}

/* Konstruktor */
Point::Point ()
{
  cout << "Point ctor" << endl;
}
Point::Point (int Newx, int Newy)
{
  x = Newx;
  y = Newy;
}
// Destruktor
Point::~Point ()
{
  cout << "Point dtor" << endl;
}
int
Point::LT (Point P1, Point P2)
   // true (bukan 0) jika P1< P2 : absis dan ordinat  lebih kecil
   // Perhatikan Current_Object tidak dipakai!

{
  return (P1.GetX() < P2.GetX()) && (P1.GetY() < P2.GetY());
}
int
Point::operator< (Point P1)
   // true jika P1< Current_Object : absis dan ordinat lebih kecil
   // Perhatikan Current_Object dipakai!
{
  return( (P1.GetX() < x) && (P1.GetY() < y));
}

// Predikat lain
```

```
int
Point::IsOrigin ()
{
  return (x == 0 && y == 0);
}

/* method */

void
Point::mirror ()
{
  x = -x;
  y = -y;
}
Point
Point::Mirrorof ()
{
  int
    tmpx = -x;
  int
    tmpy = -y;
  return (Point (tmpx, tmpy));
}

void
Point::PrintObj ()
{
  cout << "P=( " << x << "," << y << ")" << endl;
}
```

Driver “biasa” untuk test POINT

```
// file mpoint.cc
// driver "biasa" untuk mentest point.cc

#include "point.h"
#include <iostream.h>
int main(){

Point P= Point(1,2);
Point * PtP= new Point(5,5);

cout << "start.." << endl;
P.PrintObj();
PtP->PrintObj();

return 0;
}
```

Driver “kelas” untuk test POINT

```
// File :drvpoint.h
#ifndef _DRVPOINT_H
#define _DRVPOINT_H
class DrvPOINT {
public:
// sebuah meyhod yang isinya adalah test setiap function
 DrvPOINT ();
```

```
 // prosedur yang isi bodynya call thd semua feature yang ada
}; //END DRVPOINT
#endif
```

```
// File : drvpoint.cc
// Body dari drvpoint.h
#include "point.h"
#include "drvpoint.h"
#include <iostream.h>
// test setiap function
DrvPOINT::DrvPOINT ()
{
// Kamus
  Point P1;         // default constructor dengan nilai (0,0)
  Point *PtrP;      // pointer to class, objek belum hidup
  Point& RefP = P1; // referenced objek ke objek lain
                          // yang sudah hidup
  Point Pcopy = P1; // copy constructor

  Point *PtrP1 = new Point (3, 3);
  Point P2 = Point (30, 3);

// ALgoritma
  // prosedur yang isi bodynya call thd semua feature yang ada
  P1.PrintObj ();
  PtrP1->PrintObj ();
  (*PtrP1).PrintObj ();
  RefP.PrintObj();
  Pcopy.PrintObj();

// tes operator assignment
  P1 = *PtrP1;
  P1.PrintObj();

// tes terhadap operator < dan LT()
  if (P1 < *PtrP1) {
    cout << "P1 < *PtrP1\n";
  } else {
    cout << "P1 >= *PtrP1\n";
  }

// tambahkan yang lain
}
```

```
//File : mainpoint.cc
// main program yang tugasnya adalah menghidupkan sebuah objek aktif
// yaitu driver pengetest objek yang dilahirkan dari kelas point
#include "drvpoint.h"
int main()
{
 DrvPOINT Proses;
return 0;
}
```

Cara kompilasi di puntang :

```
c++ –c <nama.cc>
```

Linking :

```
c++  –o <namaexe> <namafile.cc> <nama2.cc> <nama…cc>
```

Contoh constructor dan default constructor

```
// file ctor.h
// header dan sekaligus body file untuk contoh ctor
#ifndef _CTOR_H
#define _CTOR_H
#include <iostream.h>
class ctor
{
  public:
// sebuah method yang isinya adalah test setiap function
  ctor ()
  {
    cout << "ctor()" << endl;
  }
  ctor (int NewX)
  {
    x = NewX;
    cout << "ctor(int x)" << endl;
  }
  ~ctor ()
  {
    cout << "dtor ctor" << endl;
  }
private:
  int x;
  // prosedur yang isi bodynya call thd semua feature yang ada
};                                //END CTOR
#endif
```

```
// File :cctor.h
#ifndef _CCTOR_H
#define _CCTOR_H
#include "string.h"
#include <iostream.h>
class cctor
{
  public:
// sebuah method yang isinya adalah test setiap function
  cctor ()
  {
    cout << "cctor()" << endl;
  }
  cctor (int NewX, char *Ptr)
  {
    x = NewX;
    nama = Ptr;
  }
  cctor (const cctor & C)
  {
    x = C.x;
    nama = new char[strlen (C.nama)];
    strcpy (nama, C.nama);
    cout << "cctor(int x)" << endl;
  }

  ~cctor ()
```

```
  {                                //delete nama;
    cout << "dtor cctor" << endl;
  };

//      stream& operator<< (stream& s, const cctor& C)
  void printobj ()
  {
    if (nama != NULL)
      {
        cout << "cctor= " << x << "," << nama << endl;
      }
    else
      {
        cout << "cctor= " << x << ", NULL" << endl;
      }
  }

private:
  int x;
  char *nama;
  // prosedur yang isi bodynya call thd semua feature yang ada
};                               //END CcTOR
#endif
```

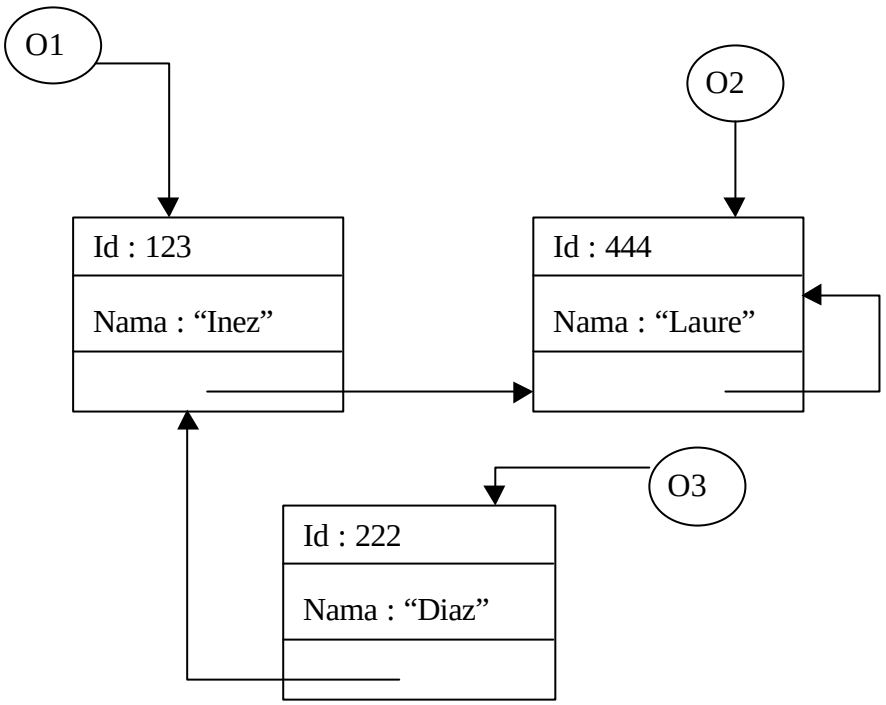
```
// file : mcctor.cpp
// driver untuk mentest kehidupan object
#include <iostream.h>
#include "ctor.h"
#include "cctor.h"
#include "string.h"
int main ()
{
  cout << "hello" << endl;
  ctor Objctor1;                // hidup!!
  ctor Objctor2 (8);            // hidup!!
  ctor & Objctor = Objctor1;    // hanya alias
  // lihat printout : dtor dipanggil
  // ada berapa objek yang hidup ?

  // object cctor mengandung pointer (string)
  char *myname;
  myname = new char[5];
  strcpy (myname, "Laure");

  cctor O1;     // O1.printobj(); bahaya jika diprint. Mengapa ?
  cctor O2 (3, NULL);
  O2.printobj ();
  cctor O3 (8, myname);
  O3.printobj ();
  cctor O4 = O3;
  cout << "end of program " << endl;
  O1 = O2;
  O1.printobj ();

  return 0;
}
```

DINAMIKA OBJEK

Latihan menghidupkan objek : buatlah sebuah kelas dengan data member yang mengandung pointer. Diberikan header file ini, tuliskan bodynya

```
// File : person.h
// Spesifikasi kelas PERSON
#ifndef _PERSON_H
#define _PERSON_H

class PERSON {
public:
// Method public
// ************ Empat sekawan
// Default Constructor
//    PERSON ( ) ;
   // membentuk sebuah PERSON, dg nilai default
// Constructor lain
      PERSON (int Newid, char* Newname="Myname", PERSON* PtrP=NULL);
  // membentuk sebuah PERSON, dg nilai default , atau disuplai

// Copy constructor : HATI-hati
   PERSON (const PERSON&);

// Assignment: HATI HATI…
   PERSON& operator= (const PERSON&);

// Destruktor
   ~PERSON() ;

// Selektor Komponen
   int GetId ();
   char* GetNama ();
   PERSON* GetSahabat();
// Pengubah Komponen
   void SetId (int );
   void SetNama (char*);
   void SetSahabat (PERSON* );
// Operasi I/O
   void PrintObj() ;
   // Print nilai Current Objek


// Data member : untuk ADT adalah Private
private:
  int id;   // identitas
  char* nama;   // nama orang
  PERSON * sahabat; // pointer ke sahabatnya, PERSON juga

}; // END CLASS PERSON
#endif
```

Setelah selesai mengkode bodynya, buatlah sebuah program yang menghidupkan tiga buah objek PERSON, dengan ilustrasi sebagai berikut :



LIST

```
// file elmt.h
// definisi elemen list
#ifndef _ELMT_H
#define _ELMT_H
#include <stdlib.h>
class Elmt;
typedef Elmt* address;
class Elmt
{
friend class List;
private:                          // atribut private
  int info;
  address next;
public:
// konstruktor
    Elmt();                         // ctor
    Elmt (int X = 0, address P=NULL);             // ctor
//    Elmt (const Elmt &);        // cctor tidak perlu
// operator assignment
// Elmt &operator=(const Elmt&);  // tidak perlu utk elmt list

// destruktor
    ~Elmt();                      // sebenarnya tidak perlu

 // Selektor
  int GetInfo ();
  address GetNext ();

  // SET
  void SetInfo (int );
  void SetNext (address );

// method lain
  void PrintObj(); // memprint info dan next

};                                // end class
#endif
```

```
// file elmt.cpp
// body  elemen list
#include "elmt.h"
#include <iostream.h>
// konstruktor
Elmt::Elmt ()
{
  cout << "default ctor elmt " << endl;
}
Elmt::Elmt (int X, address P)
{
  info = X;
  next = P;
}

// destruktor
Elmt::~Elmt ()
{
  cout << "dtor elmt" << endl;
}
```

```
  // Selektor
int
Elmt::GetInfo ()
{
  return info;
}
address Elmt::GetNext ()
{
  return next;
}

  // SET
void
Elmt::SetInfo (int NewInfo)
{
  info = NewInfo;
}
void
Elmt::SetNext (address Ptr)
{
  next = Ptr;
}

// method lain
void
Elmt::PrintObj ()              // memprint info dan next
{
  cout << "print elmt: " << endl;
  cout << "info=" << info << "next= " << next << endl;
}
```

```
// file list.h
// list dengan element yang terdefinisi pada elmt.h
#ifndef _LIST_H
#define _LIST_H
#include "elmt.h"
class List
{
  public:List ();              // ctor : create empty list
  ~List ();                    // dtor : dell all element
  List (const List &);         // cctor : copy lits
    List & operator = (const List &);   // copy list dg assignment

  // get
  address GetFirst ();
  // set
  void SetFirst (address);

  // operasi lain
  int IsEmpty ();              // 0 jika tidak kosong
  void addFirst (address);
  void addLast (address);
  void delFirst (address &);
  int IsMember (int);          // 0 jika tidak ada yg bernilai X
  void PrintObj ();            // print semua elemen list

    private:address First;
};                             //end class list
#endif
```

```
// file : list.cpp
// body dari kelas list yang menggunakan elmt
#include "list.h"
#include "elmt.h"
#include <iostream.h>

List::List ()                   // ctor  : create empty
{
  First = NULL;
}

List::~List ()                  // dtor : dell all
{
  address
    Pdel = First;
  while (First != NULL)
    {
      Pdel = First;
      First = First->next;
      delete
        Pdel;
    }
}

List::List (const List & L)     // cctor
{                               // copy list
  address
    Pt = L.First;
First=NULL;
 while (Pt != NULL)
    {
      addLast (new Elmt (Pt->info, NULL));
      Pt = Pt->next;
    }
}


List & List::operator = (const List & L)
{
  address
    Pt = L.First;

 First = NULL;
  while (Pt != NULL)
    {
      addLast (new Elmt (Pt->info, NULL));
      Pt = Pt->next;
    }

  return *this;
}

  // get
address
List::GetFirst ()
{
  return First;
}
  // set
```

```
void
List::SetFirst (address P)
{
  First = P;
}
  // operasi lain
int
List::IsEmpty ()                // 0 jika tidak kosong
{
  return First == NULL;
}

void
List::addFirst (address P)
{
  P->next = First;
  First = P;
}
void
List::addLast (address Pt)
{
  address
    P = First;
  if (P == NULL)

   {
      First = Pt;
      First->next = NULL;
    }
  else
    {
      while (P->next != NULL)
        {
          P = P->next;
        }
      P->next = Pt;
      Pt->next = NULL;
    }

}
void
List::delFirst (address & P)
{
  P = First;
  First = First->next;
}

int
List::IsMember (int X)
{
  address
    P = First;
  int
    Found = 0;
  while ((P != NULL) && (!Found))
    {
      if (P->info == X)
        {
          Found = 1;
        }
      else
```

```
        {
          P = P->next;
        }
    }
  return Found;
}

void
List::PrintObj ()
{
  if (First == NULL)
    {
      cout << "list kosong " << endl;
    }
  else
    {
      address
        P = First;
      while (P != NULL)
        {
          cout << "info=" << P->info << "next=" << P->next << endl;
          P = P->next;
        }
    }
}
```

```
//file : mlist.cpp
// driver pengetest kelas list
#include <iostream.h>
#include "elmt.h"
#include "list.h"
int main ()
{
  int i;
  cout << "start .." << endl;
  List L;
  L.PrintObj ();
  L.addFirst (new Elmt (88, NULL));
  L.addLast (new Elmt (99, NULL));
  //  L.PrintObj ();
  if (L.IsEmpty ())
    {
      cout << "yes, empty" << endl;
    }
  else
    {
      cout << "No, not empty.." << endl;
    }
 cout << "test member = " << L.IsMember (8) << endl;
  for (i = 1; i < 6; i++)
    {
      L.addFirst (new Elmt (i));
    }
  L.PrintObj ();
  List L2 = L;
  L2.PrintObj ();

  List L1;
  L1 = L;
  L1.PrintObj ();
```

```
address P;
  while (!L.IsEmpty ())
    {
      L.delFirst(P);
    }
  L.PrintObj ();
  return 0;
}
```

Latihan List

Setelah anda memahami dan dapat membuat list dengan elemen integer, buatlah:

1. List dengan elemen bertype “string” (char*)
2. List dengan elemen bertype pointer ke nilai integer
3. List dengan elemen Person dari person.* yang pernah dibuat
4. List dengan elemen bertype pointer ke type (object) apapun

Nested Class

```
// file nlist.h
// nested element terhadap list
#ifndef _NLIST_H
#define _NLIST_H
class List
{
  public:List ();               // ctor
  ~List ();                     // dtor
  //List (const List&); // cctor
  //List &operator= (const List&);
  class elmt;                   // froward declaration
  // get
  elmt *GetFirst ();
  // set
  void SetFirst (elmt *);

  // operasi lain
  int IsEmpty ();               // 0 jika tidak kosong
  void addFirst (const int &);
  void delFirst (int &);
  int IsMember (int);           // 0 jika tidak ada yg bernilai X
  void PrintObj ();             // print semua elemen list

   private:class elmt
  {                             // nested class
  public:
    elmt (const int &X);                // ctor
     ~elmt ();                  //dtor
// atribut
    int info;
    elmt *next;
  };                            // end class elmt

  elmt *First;                  // list dikenali elemen pertama
};                              // end nlist
#endif
```

```
// file nlist.cpp
// body nested element dalam list
#include "nlist.h"
#include <stdlib.h>
#include <iostream.h>
List::List ()                   // ctor
{
  First = 0;
}

List::~List ()                  // dtor
{
  int
    X;
  while (!IsEmpty ())
    {
      delFirst (X);
    }
}
//List::List (const List& L) // cctor
//{}
//List::List &operator= (const List& L)
```

```
//{}
        // get
List::elmt* List::GetFirst ()
{
  return First;
}
        // set
void
List::SetFirst (List::elmt * P)
{
  First = P;
}
    // operasi lain
int
List::IsEmpty ()                // 0 jika tidak kosong
{
  return First == 0;
}

void
List::addFirst (const int& X)
{
  List::elmt * P = new List::elmt (X);
  P->next = First;
  First = P;

}
void
List::delFirst (int &X)
{
  List::elmt * P;
  P = First;
  X = P->info;
  First = First->next;
}
int
List::IsMember (int X)
{
  List::elmt * P = First;
  int
    Found = 0;
  while ((P != NULL) && (!Found))
    {
      if (P->info = X)
        {
          Found = 1;
        }
      else
        {
          P = P->next;
        }
    }
  return Found;
}
void
List::PrintObj ()
{ cout << "Print list : " << endl;
  if (First == NULL)
    {
      cout << "list kosong " << endl;
    }
```

```
  else
    {
      List::elmt * P;
      P = First;
      while (P != NULL)
        {
          cout << "info=" << P->info << "next=" << P->next << endl;
          P = P->next;
        }
    }
}

// Konstruktor elmt
List::elmt::elmt (const int &X) {info=X; next=NULL;}
```

```
//file mnlist.cpp
// driver pengetest nested class utk element list
#include <iostream.h>
#include "nlist.h"
int
main ()
{
  int i;
  cout << "start .." << endl;
  List L;
  L.PrintObj ();

  L.addFirst (5);
  L.PrintObj ();
  if (L.IsEmpty ())
    {
      cout << "yes, empty" << endl;
    }
  else
    {
     cout << "No, not empty.." << endl;
    }
  cout << "test member = " << L.IsMember (8) << endl;
  for (i = 0; i < 6; i++)
    {
      L.addFirst (i);
    }
  L.PrintObj ();
  while (! L.IsEmpty())
   {
      L.delFirst (i);
      cout << "i hsl del=" << i << endl;
    }
  L.PrintObj ();
  return 0;
}
```

Generic Class

```
// File : gpoint.h
// Contoh point dengan elemen generik
#ifndef _GPoint_H
#define _GPoint_H
#include <iostream.h>
template < class t_elmt >       // generic class
  class GPoint
{                               // point generic
public:
// Konstruktor
  GPoint (t_elmt Vx = 0, t_elmt Vy = 0);

// Destruktor
  ~GPoint ();
// method GET dan SET
  t_elmt GetX ();
  t_elmt GetY ();
  void SetX (const t_elmt &);
  void SetY (const t_elmt &);

// method
 void mirror ();
  GPoint Mirrorof ();
  void PrintObj ();
private:
  t_elmt X;                     //absis
  t_elmt Y;                     // ordinat
};                              // end class generic point

/////////////// Body di sini sebab kelas generik
// lihat diktat C++ (HD)
// Konstruktor
template < class t_elmt > GPoint < t_elmt >::GPoint (t_elmt Vx,
t_elmt Vy)
{
  X = Vx;
  Y = Vy;
}

// Destruktor
template < class t_elmt > GPoint < t_elmt >::~GPoint ()
{
  cout << "dtor GPoint" << endl;
}
// method GET dan SET


template < class t_elmt > t_elmt GPoint < t_elmt >::GetX ()
{
  return x;
}
template < class t_elmt > t_elmt GPoint < t_elmt >::GetY ()
{
  return y;
}
template < class t_elmt > void GPoint < t_elmt >::SetX (const t_elmt
& NewX)
{
  X = NewX;
}
```

```
template < class t_elmt > void GPoint < t_elmt >::SetY (const t_elmt
& NewY)
{
  Y = NewY;
}

// method
template < class t_elmt > void GPoint < t_elmt >::mirror ()
{
  X = -X;
 Y = -Y;
}
template < class t_elmt > GPoint < t_elmt > GPoint < t_elmt
>::Mirrorof ()
{
  t_elmt tmpx = -X;
  t_elmt tmpy = -Y;
  return (GPoint (tmpx, tmpy));
}
template < class t_elmt > void GPoint < t_elmt >::PrintObj ()
{
  cout << "Point =(" << X << "," << Y << ")" << endl;
}
#endif
```

```
// File : mgpoint.cpp
// driver untuk mentest mgpoint.h: point dengan elemen generik
#include "gpoint.h"
#include <iostream.h>
int
main ()
{
  cout << "start test gpoint integer " << endl;
  GPoint < int >P;
  GPoint < int >*PtP = new GPoint < int >(8, 8);
  GPoint < int >P1 = GPoint < int >(5, 5);
  P.PrintObj ();
  PtP->PrintObj ();
  P1.PrintObj ();

  cout << "start test gpoint float " << endl;
  GPoint < float >Pf;
  GPoint < float >*PtPf = new GPoint < float >(8.5, 8.8);
  GPoint < float >Pf1 = GPoint < float >(5.88, 5.99);
  Pf.PrintObj ();
  PtPf->PrintObj ();
  Pf1.PrintObj ();
  return 0;
}
```

Generic LIST

```
// file glist.h
// definisi elemen list
#ifndef _GLIST_H
#define _GLIST_H
#include <stdlib.h>
#include <iostream.h>

template < class Type >         // generic type
  class GElmt
{
public:
//  friend class List < Type >;
// konstruktor
  GElmt (const Type & X);       // ctor
// destruktor
  ~GElmt ();
  // sebenarnya tidak perlu
  // Selektor GET
  Type GetInfo ();
  GElmt *GetNext ();
  // SET
  void SetInfo (const Type & X);
  void SetNext (GElmt * P);
// method lain
  void PrintObj ();             // memprint info dan next
// private:                     // atribut private
  Type info;
  GElmt *next;
};                              // end class elmt


/////// Definisi kelas list dengan elemen generic ////////
template < class Type > class List
{
public:
  List ();                      // ctor
  ~List ();                     // dtor
  List (const List &);          // cctor
  List & operator = (const List &);

// method lain
  int IsEmpty () const;
  void AddFirst (const Type &);
  void DelFirst (Type &);

void PrintObj () const;
private:
  GElmt < Type > *First;
};                              // end class list

///////// REALISASI KELAS GENERIK : dalam file berkestensi H
// Body Elmt list
template < class Type > GElmt < Type >::GElmt (const Type & X)  //
ctor
{
  info = X;
  next = NULL;
}
// destruktor
template < class Type > GElmt < Type >::~GElmt ()
```

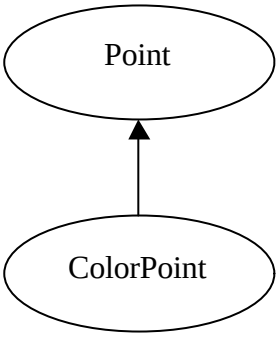
```
{
}                                  // sebenarnya tidak perlu
  // Selektor GET

template < class Type > Type GElmt < Type >::GetInfo ()

{
  return info;
}
template < class Type > GElmt < Type > *GElmt < Type >::GetNext ()
{
  return next;
}

  // SET
template < class Type > void GElmt < Type >::SetInfo (const Type & X)
{
  info = X;
}
template < class Type > void GElmt < Type >::SetNext (GElmt * P)
{
  next = P;
}


////// Body List
template < class Type > List < Type >::List ()

{
  First = NULL;
}

template < class Type > List < Type >::~List ()
{
  Type P;
  while (!IsEmpty ())
    {
      DelFirst (P);
    }
}
template < class Type > int List < Type >::IsEmpty () const
{
  return (First == NULL);
}

template < class Type > void List < Type >::AddFirst (const Type & V)
{
  GElmt < Type > *P = new GElmt < Type > (V);
  P->next = First;
  First = P;
}

template < class Type > void List < Type >::DelFirst (Type & V)
{
  V = First->info;
  First = First->next;
}

template < class Type > void List < Type >::PrintObj () const
{
  GElmt < Type > *P = First;
```

```
  cout << "start print list" << endl;
  while (P != NULL)
    {
      cout << P->info << P->next << endl;
      P = P->next;
    }
  cout << "end print list" << endl;
}
#endif
```

```
// file mglist.cpp
// driver untuk mentest glist.h
#include "glist.h"
#include <iostream.h>
int
main ()
{
  char CC;
  int i;
  List < int >L;
  List < char >L2;

  cout << "start test glist " << endl;
  for (i = 0; i < 6; i++)
    L.AddFirst (i);
  L.PrintObj ();
  while (!L.IsEmpty ())
    {
      L.DelFirst (i);
    }
  L.PrintObj ();

 CC = 'a'; L2.AddFirst (CC);
  CC = 'b'; L2.AddFirst (CC);
  CC = 'C'; L2.AddFirst (CC);
  L2.PrintObj ();
  while (!L2.IsEmpty ())
    {
      L2.DelFirst (CC);
    }
  L2.PrintObj ();
  return 0;
}
```

INHERITANCE

Contoh inheritance sederhana : ColorPoint adalah kelas turunan Point, dengan tambahan atribut Color bertype integer;




```
// File : Point.h
// ADT Point (sebagian)
#ifndef _Point_H
#define _Point_H
class Point
{ public:
// Konstruktor
    Point ();
    Point (int, int);

// Destruktor
   ~Point ();
// method GET dan SET
  int GetX ();
  int GetY ();
  void SetX (int);
  void SetY (int);

// Relasional
  int LT (Point P1, Point P2);
  // true (bukan 0) jika P1< P2 : absis dan ordinat  lebih kecil
  // Perhatikan Current_Object tidak dipakai!
  int operator<(Point P1);
  // true jika P1< Current_Object : absis dan ordinat lebih kecil
  // Perhatikan Current_Object dipakai!
// Predikat lain
  int IsOrigin ();
  // true (bukan 0) jika Current_Object adalah (0,0)

// method
  void mirror ();
  Point Mirrorof ();
  void PrintObj ();
  protected:                        // atribut ADT : private,
                        // tapi supaya dpt diakses anak dijadikan
protected
  int x;                          //absis
  int y;                          // ordinat
};
#endif
```

```
// file point.cpp
// idem dengan sebelumnya
```

```
// file colorpoint.h
// inherit point

class ColorPoint: public Point {

public:
ColorPoint(); // ctor
ColorPoint(int Vx, int Vy, int C=0);      // ctor
  ~ColorPoint (); // dtor

// Get dan Set
int GetColor();
void SetColor (int);

// method lain
void PrintObj();

// atribut spesifik
private:
      int Color;

}; // end class ColorPoint
```

```
#include <iostream.h>
#include "point.h"
#include "colorpoint.h"

ColorPoint::ColorPoint() {cout << "ctor default" << endl;}
ColorPoint::ColorPoint(int Vx, int Vy, int C)   // ctor
{x=Vx; y=Vy; Color=C;
cout << "ctor lain " << endl;}

ColorPoint::~ColorPoint() // dtor
{cout << "dtor of color point" << endl;}

// Get dan Set
int ColorPoint::GetColor(){return Color;}
void ColorPoint::SetColor (int NewC)
{ Color=NewC;}

// method lain
void ColorPoint::PrintObj()
{cout << "x=" << x << "y=" << y << "color=" << Color << endl ;
}
```

```
// file mpoint.cc
// driver "biasa" untuk mentest point.cc

#include "point.h"
#include "colorpoint.h"
#include <iostream.h>
int main(){
```

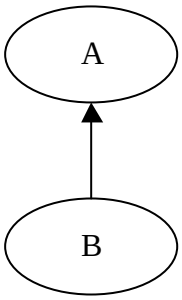
```
      cout << "start.. test point " << endl;
Point P= Point(1,2);
Point * PtP= new Point(5,5);


P.PrintObj();
PtP->PrintObj();

cout << "start test Color Point" << endl;
ColorPoint CP1;
ColorPoint * PtCP = new ColorPoint(6,6,2);
CP1.PrintObj(); // hati-hati, nilainya kacau
PtCP->PrintObj();

return 0;
}
```

Pedoman pemrograman dan testing untuk kelas yang mempunyai hubungan cukup kompleks:

- Buat kode kelas Bapak, lakukan test dengan sebuah driver
- Buat kelas Anak, lakukan test secara independent
- Untuk setiap anak lakukan hal yang sama
- Buat driver untuk mentest semua kelas dalam satu pohon inheritance, untuk mentest ulang semuanya

Test terhadap aspek dinamika kehidupan objek dapat dilakukan dengan melakukan dumb ke sebuah text file. Setelah menguasai semua dinamika kehidupan, barulah anda akan memakai kelas tersebut untuk persoalan sebenarnya

Latihan memahami dinamika kehidupan dan inheritance
Letakkan hubungan ini dalam sebuah direktori



Kelas A

```
// File: A.h
#ifndef __A_H
#define __A_H
#include <stream.h>
class A
{
  public:
// Konstruktor: atribut by default diisi xa=88, pa=0
  A ()
  {
    cout << "konstruktor A \n";
    xa = 88;
    pa = 0;
  };
  A (int x, int z)
  {
    cout << "konstruktor A(x) \n";
    xa = x;
    pa = z;
  };
// Copy constructor
  A (const A & objA)
  {
    cout << "copy constructor A\n";
    xa = objA.xa;
    pa = objA.pa;
  };
// Operator assignment
  A & operator = (const A & objA)
  {
    xa = objA.xa;
    pa = objA.pa;
  };
// Destruktor
  ~A ()
  {
  };
// Method
  void PrintObjek ()
  {
    cout << "Ini atribut A public: xa =" << xa << "\n";
    cout << "Ini atribut A protected: pa =" << pa << "\n";
};
  int f ()
  {
```

```
    cout << " f di A \n";
    protectf (pa);
  };
  int g ()
  {
    cout << " g di A \n";
  };
  virtual int fV ()
  {
    cout << "virtual function fV() di A\n";
  };
// Atribut yg hanya dapat diakses anak
protected:
  int pa;
  void protectf (int x)
  {
    cout << "protectf(x) di A, x=" << x << "\n";
  };
// atribut
private:
  int xa;
};                              // END CLASSA

#endif // __A_H
```

```
// File: mainA.cc
// driver pengetest kelas A
#include "a.h"

int
main ()
{
  A objA1;
  A objA2 (3, 3);
  A objA3 = objA2;

// Protected tidak boleh dipanggil oleh objek
//  objA1.protectf(5);

  objA1.PrintObjek ();
  objA1.f ();
  objA1.g ();

  return 0;
}
```

Hasil eksekusi `maina.cc`

```
konstruktor A
konstruktor A(x)
copy constructor A
Ini atribut A public: xa =88
Ini atribut A protected: pa =0
 f di A
protectf(x) di A, x=0
 g di A
```

Kelas B

```
// File: B.h
#ifndef __B_H
#define __B_H

#include <stream.h>
#include "a.h"

class B:public A
{
// inherit A
  public:
// Konstruktor: atribut by default diisi xb=88
  B ()
  {
    cout << "konstruktor B \n";
    xb = 88;
  };
  B (int x)
  {
    cout << "konstruktor B(x) \n";
    xb = x;
  };
// Copy constructor
  B (const B & objB):A (objB)
  {
    cout << "copy constructor B\n";
    xb = objB.xb;
  };
// Operator  assignment
  B & operator = (const B & objB)
  {
    xb = objB.xb;
  };
// Destruktor
  ~B ()
  {
    // null;
  };
// Method
  void PrintObjek ()
  {
    cout << "Ini atribut B public: xb =" << xb << "\n";
  };
  int g ()

 {
    cout << " g di B \n";
  };
  // ini fungsi tambahan yang hanya ada di kelas B
  int fb ()
  {
    cout << " fb() di B \n";
  };
  // pure virtual function di parent A di definisikan di sini
  virtual int fpureV ()
```

```
  {
    cout << "virtual fpureV() di B\n";
  };
  // redefine function tapi menggunakan virtual function di parent A
  virtual int fV ()
  {
    cout << "virtual fV() di B\n";
  };
// atribut
private:
  int xb;
};                              // END CLBSSB

#endif // __B_H
```

Driver untuk mentest b.h

```
// File: mainB.cc

#include "b.h"

int
main ()
{
  B objB1;
  B objB2 (6);
  B objB3 = objB2;

  objB1.PrintObjek ();
  objB1.fb ();
  objB1.g ();

  return 0;
}
```

Hasil eksekusi mainb.cc

```
konstruktor A
konstruktor B
konstruktor A
konstruktor B(x)
copy constructor A
copy constructor B
Ini atribut B public: xb =88
 fb() di B
 g di B
```

Driver untuk mentest kelas A dan kelas B

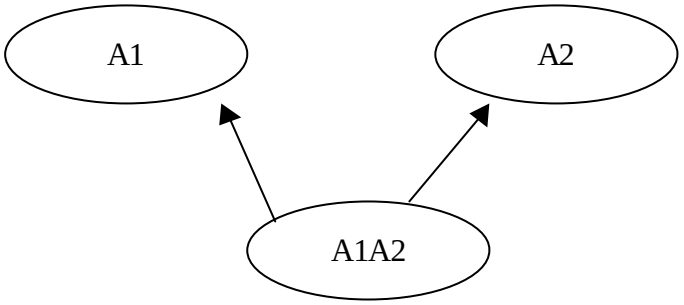
```
// File: mainAB.cc

#include "a.h"
#include "b.h"

int
main ()
{
// KAMUS
  A objA1;
```

```
  A objA2 (3, 3);
  A objA3 = objA2;
  A *PtrObjA;

  B objB1;
  B objB2 (6);
  B objB3 = objB2;
  B *PtrObjB;

// ALGORITMA
  cout << " Call B \n";
  objB1.PrintObjek ();
  objB1.fb ();
  objB1.g ();

  cout << " Call A \n";

// Protected tidak boleh dipanggil oleh objek
//  objA1.protectf(5);

  objA1.PrintObjek ();
  objA1.f ();
  objA1.g ();

  cout << " Polymorphic attachment A = B \n";
  cout << "tidak berlaku sebab HARUS POINTER!!! \n";
  objB2.PrintObjek ();

  objA1 = objB2;
  objA1.f ();
  objA1.g ();
  objA1.PrintObjek ();
  objA1.fV ();
// POlymorphic attachment
  PtrObjA = new A (5, 5);
  PtrObjB = new B (55);

  cout << "Polymorphic attachment lewat pointer \n";
  PtrObjA = PtrObjB;

  PtrObjA->f ();
  PtrObjA->g ();
  PtrObjA->PrintObjek ();
  PtrObjA->fV ();

  PtrObjB->f ();
  PtrObjB->g ();
  PtrObjB->PrintObjek ();
  PtrObjB->fV ();

// Cara kedua
  cout << "Polymorphic attachment lewat pointer langsung B hidup\n";
  PtrObjA = new B (555);
  PtrObjA->f ();
  PtrObjA->g ();
  PtrObjA->PrintObjek ();
  PtrObjA->fV ();

  return 0;
}
```

Hasil eksekusi mab.cc

```
konstruktor A
konstruktor A(x)
copy constructor A
konstruktor A
konstruktor B
konstruktor A
konstruktor B(x)
copy constructor A
copy constructor B
 Call B
Ini atribut B public: xb =88
 fb() di B
 g di B
 Call A
Ini atribut A public: xa =88
Ini atribut A protected: pa =0
 f di A
protectf(x) di A, x=0
 g di A
 Polymorphic attachment A = B
tidak berlaku sebab HARUS POINTER!!!
Ini atribut B public: xb =6
 f di A
protectf(x) di A, x=0
 g di A
Ini atribut A public: xa =88
Ini atribut A protected: pa =0
virtual function fV() di A
konstruktor A(x)
konstruktor A
konstruktor B(x)
Polymorphic attachment lewat pointer
 f di A
protectf(x) di A, x=0
 g di A
Ini atribut A public: xa =88
Ini atribut A protected: pa =0
virtual fV() di B
 f di A
protectf(x) di A, x=0
 g di B
Ini atribut B public: xb =55
virtual fV() di B
Polymorphic attachment lewat pointer langsung B hidup
konstruktor A
konstruktor B(x)
 f di A
protectf(x) di A, x=0
 g di A
Ini atribut A public: xa =88
Ini atribut A protected: pa =0
virtual fV() di B
```

MULTIPLE INHERITANCE



Kelas A1

```
// File: A1.h
#ifndef __A1_H
#define __A1_H
#include <iostream.h>

class A1
{
  public:
// Kostruktor: atribut by default diisi xa=88, pa=0
  A1 ()
  {
    cout << "kostruktor A1 " << endl;
    xa = 88;
    pa = 0;
  }
  A1 (int x, int z)
  {
    cout << "kostruktor A1(x) " << endl;
    xa = x;
    pa = z;
  }
// Copy constructor
  A1 (const A1 & objA1)
 {
    cout << "copy constructor A1" << endl;
    xa = objA1.xa;
    pa = objA1.pa;
  }
// Operator assigment
  A1 & operator = (const A1 & objA1)
  { cout << "Op= A1 " << endl;
    xa = objA1.xa;
    pa = objA1.pa;
  };
// Destruktor
  ~A1 ()
  { cout << "dtor A1" << endl;
  }
// Method
  void PrintObjek ()
  {
    cout << "Ii atribut A1 public: xa =" << xa << endl;
    cout << "Ii atribut A1 protected: pa =" << pa << endl;
  }
int f ()
  {
```

```
    cout << " f di A1 " << endl;
    protectf (pa);
  }
  int g ()
  {
    cout << " g di A1 " << endl;
  }
  virtual int fV ()
  {
    cout << "virtual fuction fV() di A1" << endl;
  }
// A1tribut yg haya dapat diakses anak
protected:
  int pa;
  void protectf (int x)
  {
    cout << "protectf(x) di A1, x=" << x << "";
  }
// atribut
private:
  int xa;
};                              // END CLASSA1

#endif // __A1_H
```

Kelas A2

```
// File: A2.h
#ifndef __A2_H
#define __A2_H

#include <iostream.h>

class A2
{
  public:
// Konstruktor: atribut by default diisi xa=88, pa=0
  A2 ()
  {
    cout << "konstruktor A2 " << endl;
    xa = 88;
    pa = 0;
  }
  A2 (int x, int z)
  {
    cout << "konstruktor A2(x) " << endl;
    xa = x;
    pa = z;
  }
// Copy constructor
 A2 (const A2 & objA2)
  {
    cout << "copy constructor A2" << endl;
    xa = objA2.xa;
    pa = objA2.pa;
  }
// Operator assignment
  A2 & operator = (const A2 & objA2)
```

```
  { cout << "op= A2 " << endl;
    xa = objA2.xa;
    pa = objA2.pa;
  }
// Destruktor
  ~A2 ()
  { cout << "dtor A2" << endl;
  }
// Method
  void PrintObjek ()
  {
    cout << "Ini atribut A2 public: xa =" << xa << endl;
    cout << "Ini atribut A2 protected: pa =" << pa << endl;
 }
  int f ()
  {
    cout << " f di A2 " << endl;
    protectf (pa);
  }
  int g ()
  {
    cout << " g di A2 " << endl;
  }
  virtual int fV ()
  {
    cout << "virtual function fV() di A2" << endl;
  }
// A2tribut yg hanya dapat diakses anak
protected:
  int pa;
  void protectf (int x)
  {
    cout << "protectf(x) di A2, x=" << x << endl;
  }

// atribut
private:
  int xa;
};                              // END CLA2SSA2

#endif // __A2_H
```

Maina1_a2.cc

Sebenarnya file ini dibangun secara sekuensial: mula-mula mentest A1.h, kemudian ditambahkan kode untuk mentest A2.h. Gunanya supaya tidak menambah banyaknya driver

```
// File: maina1a2.cc
// driver pengetest kelas A1 dan A2
#include "a1.h"
#include "a2.h"

int main() {
  A1  objA1_1;
  A1  objA1_2( 3, 3);
  A1  objA1_3 = objA1_2;

  A2  objA2_1;
```

```
  A2  objA2_2( 5, 5);
  A2  objA2_3 = objA2_2;

  objA1_2.PrintObjek();
  objA1_2.f();
  objA1_2.g();

  objA2_2.PrintObjek();
  objA2_2.f();
  objA2_2.g();

  return 0;
}
```

Hasil eksekusi maina1_a2.cc

```
kostruktor A1
kostruktor A1(x)
copy constructor A1
konstruktor A2
konstruktor A2(x)
copy constructor A2
Ii atribut A1 public: xa =3
Ii atribut A1 protected: pa =3
 f di A1
protectf(x) di A1, x=3 g di A1
Ini atribut A2 public: xa =5
Ini atribut A2 protected: pa =5
 f di A2
protectf(x) di A2, x=5
 g di A2
dtor A2
dtor A2
dtor A2
dtor A1
dtor A1
dtor A1
```

Kelas A1A2

```
// file a1a2.h
// Kelas turunan A1 dan A2 (multiple inheritance
#ifndef __A1A2_H
#define __A1A2_H
#include "a1.h"
#include "a2.h"
#include <iostream.h>
class A1A2:public A1, A2
// multiple inheritance A1 dan A2
{
  public:
// Konstruktor: atribut by default diisi xaa=0
  A1A2 ():A1 (-99, -99), A2 (-100, -100)
  {
    cout << "konstruktor A1A2 " << endl;
    xaa = 0;
  }
  A1A2 (int x)
  {
```

```
    cout << "konstruktor A1A2(x) " << endl;
    xaa = x;
  }
// Copy constructor
  A1A2 (const A1A2 & objA1A2)
  {
    cout << "copy constructor A1A2\n";
    xaa = objA1A2.xaa;
  }
// Operator assignment
  A1A2 & operator = (const A1A2 & objA1A2)
  {
    xaa = objA1A2.xaa;
  }
// Destruktor
  ~A1A2 ()
  {
  }

  // Method
  void fA1A2 ()
  {
    cout << "fA1A2()\n";
  }
  void PrintObjek ()
  {
    cout << "Data member di parent class A1: " << endl;
    A1::PrintObjek ();
    cout << "Data member di parent class A2: " << endl;
    A2::PrintObjek ();
    cout << "Data member di class A1A2: " << endl;
    cout << "xaa = " << xaa << endl;
  };

private:
  int xaa;
};
#endif
```

maina1a2.cc

```
// file maina1a2.cc
// driver pengetest kelas turunan A1A2, yaitu kelas
// multiple inherit dari A1 dan A2
#include "a1a2.h"
int main() {
  // Konstruktor parent A1, A2, dan kelas A1A2 hidup semua
  A1A2 oA1A2;
  return 0;
}
```

hasil eksekusi ma1a2.cc

```
kostruktor A1(x)
konstruktor A2(x)
konstruktor A1A2
dtor A2
dtor A1
```

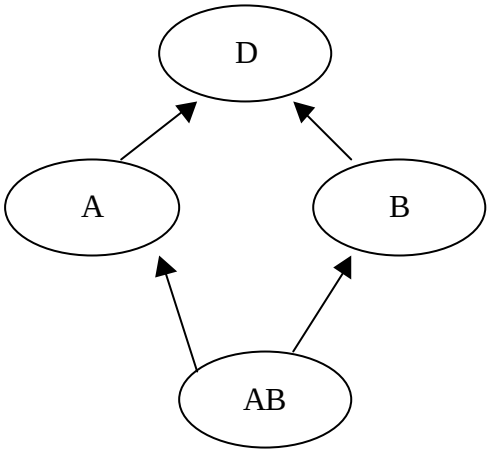
```
// file maina1a2_1.cc
// test
#include "a1a2.h"

int main() {

  // Konstruktor parent A1, A2, dan kelas AA hidup semua
  A1A2 oA1A2;

  A1 oA1(6,6);
  A2 oA2(3,3);

// SALAH, karena xa adalah private
//  oA1A2.xa = 100;

// AMBIGOUS, sebab ada di kedua parent
//  oA1A2.pa = 100;
  oA1A2.PrintObjek();

  return 0;
}
```

```
kostruktor A1(x)
konstruktor A2(x)
konstruktor A1A2
kostruktor A1(x)
konstruktor A2(x)
Data member di parent class A1:
Ii atribut A1 public: xa =-99
Ii atribut A1 protected: pa =-99
Data member di parent class A2:
Ini atribut A2 public: xa =-100
Ini atribut A2 protected: pa =-100
Data member di class A1A2:
xaa = 0
dtor A2
dtor A1
dtor A2
dtor A1
```

Buatlah program utama sehingga menunjukan polymorphic attachment

```
// file maina1a2_2.cc
// test polimorphic attachment
#include "a1a2.h"

int main() {
  // Konstruktor parent A1, A2, dan kelas AA hidup semua
  A1A2 oA1A2;
  A1 oA1(6,6);
  A2 oA2(3,3);
  cout << "Nilai oA1 sebelum attachment : " << endl;
  oA1.PrintObjek();
// attachment
  oA1 = oA1A2;
  cout << "Nilai oA1 setelah attachment : " << endl;
  oA1.PrintObjek();
   oA1.fA1A2();
  return 0;
}
```

REPEATED INHERITANCE




```
// File: D.h
class D {
public:
// Konstruktor
  D(0)
// Destruktor
  ~D;
// Method
     void PrintObjek ();
      int f();
      int g();
// atribut
private:
  Int  xd;
}; // END CLASSA
```

```
// File: A.h
class A {
Inherit D;
public:
// Konstruktor
  A(0)
// Destruktor
  ~A;
// Method
     void PrintObjek ();
      int f();
      int g();
// atribut
```

```
private:
  Int  xa;
}; // END CLASSA
```

```
// File: B.h
class B {
Inherit D;
// Konstruktor
  B(0)
// Destruktor
  ~B;
// Method
     void PrintObjek ();
      int f();
      int g();
// atribut
private:
  Int  xB;
}; // END CLASSB
```

```
class AB {
Inherit A;
Inherit B;
public:
// Konstruktor
  AB(0)
// Destruktor
  ~AB;
// Method
     void PrintObjek ();
// Tambahan method publik
      int fAB ();
// tambahan atribut private
private:
  Int xab;
}; // END CLASSAB
```

Dinamika kehidupan Objek

Untuk menunjukkan kehidupan objek, dalam setiap method harus diawali dengan perintah cout << "Ini method M…" Dengan M adalah nama methodd yang bersangkutan

```
// File: classA.h
class classA {

// Konstruktor
   classA ();

// Destruktor
    ~ classA;

// Assignment
   void = (classA &Ahsl);
   // Assignment current Object ke Ahsl

// Pembandingan
   Boolean "==" (classA A1, A2);
   Boolen Eq (classA A1, A2);

// Print nilai Objek: untuk pengamatan atribut
   void PrintObjek ();

// Atribut
private:
 int ix;
 char CC;
 int * APtrI;
 char * AStr;
} ; // End CLASSA
```

Program yang menghidupkan kelas A

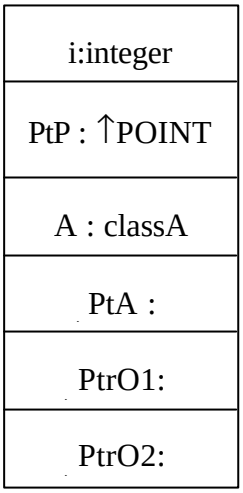
```
// File : mainA.cc
#include “classA.h”
#include “boolean.h”

int main () {
// Kamus
classA a1, a2; // otomatis hidup!
classA * PtrA1;
classA PtrA2=PtrA1;
classA &RefA1=a1; // alias
Int x;


// Algoritma

// hidupkan objek sudah dilakukan pd saat deklarasi


// Cek hasilnya apakah objek yang dihasilkan benar
   a1.PrintObjek ();
   a2.PrintObjek ();
// Alokasi dulu PtrA1
   PtrA1= new classA;
   (* PtrA1).PrintObjek();
   (* PtrA2).PrintObjek();
   PrintObjek (RefA1);

// Assignment objek
   a2=a1;
// assignement antara objek dan reference to object
   a1=PtrA1; // Apa akibatnya ???
// pointer assignment (attachment)
 PtrA2=PtrA1;
// Objek comparaison
 If (a1==a2) { } else {}

// Reference comparaison
 If (PtrA1= PtrA2) {cout << “sama”; } else { cout <<
“beda”;  }

// Bandingkan pointer dengan bukan: bisa SALAH, bisa
benar?!!!
  If (PtrA1=a1) { } else { }


Return 0;
```

```
}
```

Dinamika kehidupan Objek dengan reference yang kompleks

Untuk menunjukkan kehidupan objek (efek copy, clone, deep clode), dalam setiap method harus diawali dengan perintah cout << "Ini method M…"
Dengan M adalah nama methodd yang bersangkutan

Nama kelas : classObj

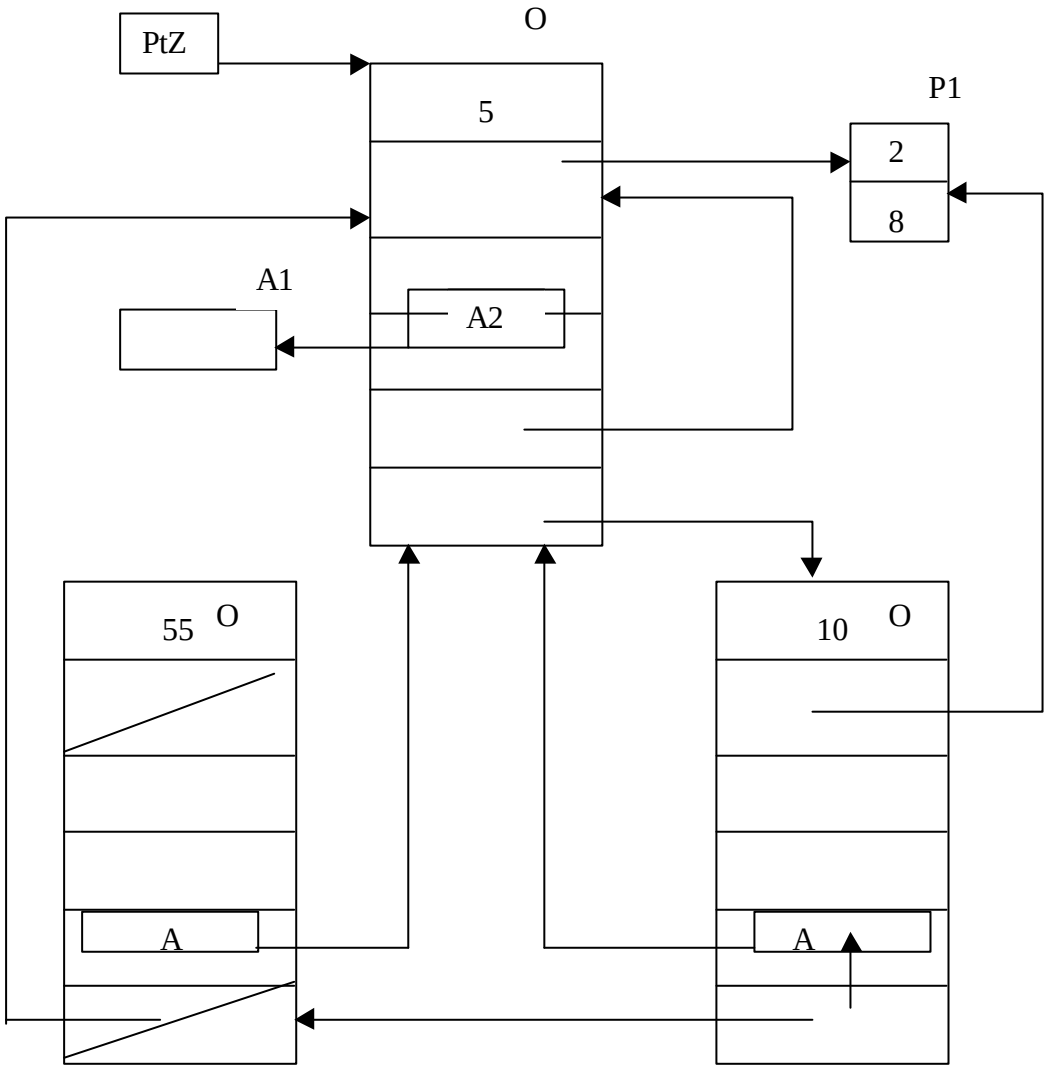



```
// File : "classobj.h"
#include "point.h"
#include "classA.h"
class classObj {
public:
// Konstruktor
   classObj();
// Selektor
   ~classObj();

// PrintObjek: Tampilkan nilai atribut
   void PrintObjek();

// atribut
private:
  int i;
  POINT P;
  classA A; // "expanded" object
  classA * PtA;
  classObj * PtrO1;
  classObj *PtrO2
} ; //END CLASSOBJ
```

Program untuk menghidupkan objek : program ini berasal dari main program untuk menghidupkan kelas A, dan ditambah seperlunya untuk menghidupkan classObj

```
// File : mainObj.cc
#include “classA.h”
#include “boolean.h”
#include “point.h”

int main () {
// Kamus
classA a1, a2, a3, a4; // otomatis hidup!
classA * PtrA1;
classA PtrA2=PtrA1;
classA &RefA1=a1; // alias
Int x;

POINT P1;

classObj * PtO1;
classObj O1(5,PtrP,a1,PtrA1,*PtrO1, NULL);
classObj O2, O3;
classObj * PtO2=PtO1;


// Algoritma

// hidupkan objek


// Cek hasilnya apakah objek yang dihasilkan benar
   a1.PrintObjek ();
   a2.PrintObjek ();
   PrintObjek (PtrA1);
   PrintObjek (PtrA2);
   PrintObjek (RefA1);
// Print hasil cretion classObj
   PrintObj(O1);
   PrintObj(O2);
   PrintObjek(O3);

// Assignment : attachment, value, ref vs value




Return 0;
}
```

Tambahkan instruksi sehingga pada suatu saat terbentuk sbb dan lakukanlah beberapa perintah assignement supaya anda dapat menciptakan reference attachment, copy, clone dan deepclone seperti yang dijelaskan di kuliah:



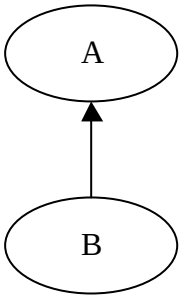
STATIC MEMBER :
Tambahkan sebuah atribut static pada definisi classObj, dengan spesifikasi :

```
private:
 int NbObj=0;  // banyaknya objek yang sudah ada dalam
sistem
```

Amati apa yang terjadi.

INHERITANCE

Letakkan hubungan ini dalam sebuah direktori




```
// File: A.h
#ifndef __A_H
#define __A_H

class A {
public:
// Konstruktor: atribut by default diisi xa=88, pa=0
 A(){ cout << “konstruktor A \n“; xa= 88; pa=0;};
 A( int x, int z ) { cout << “konstruktor A(x) \n“; xa =
x; pa=z;};
// Copy constructor
 A( const A& objA ) {
     cout << “copy constructor\n”;
     xa = objA.xa;
     pa = objA.pa; };
// Destruktor
  ~A();
// Method
      void PrintObjek(){cout << “Ini atribut A: “ xa; };
      int f() { cout << “ f di A \n”;} ;
      int g() { cout << “ g di A \n”;} ;
// Atribut yg hanya dapat diakses anak
protected:
  int pa;
  void protectf( const int  x) {
    cout << “protectf(x) di A, x=” << x << “\n”; };

// atribut
private:
  int  xa;
}; // END CLASSA

#endif // __A_H
```

```
// File: mainA.cc

int main() {
  A  objA1;
```

```
  A  objA2( 3, 3);
  A  objA3 = objA2;

  objA1.protectf(5);

  return 0;
}
```

```
// File: B.h
#ifndef __B_H
#define __B_H

#include <stream.h>
#include "a.h"

class B : public A {
// inherit A
public:
// Konstruktor: atribut by default diisi xb=88
 B() { cout << "konstruktor B \n"; xb= 88; };
 B( int x ) { cout << "konstruktor B(x) \n"; xb = x; };
// Copy constructor
 B( const B& objB ) : A(objB) {
     cout << "copy constructor B\n";
     xb = objB.xb; };
// Operator  assignment
 B& operator= (const B& objB) { xb = objB.xb; };
// Destruktor
  ~B() {
      // null;
  };
// Method
      void PrintObjek(){
       cout << "Ini atribut B public: xb =" << xb <<
"\n"; };
      int g() { cout << " g di B \n";} ;
     // ini fungsi tambahan yang hanya ada di kelas B
      int fb() {
       cout << " fb() di B \n"; };
     // pure virtual function di parent A di definisikan
di sini
      virtual int fpureV() { cout << "virtual fpureV() di
B\n"; };
     // redefine function tapi menggunakan virtual
function di parent A
      virtual int fV() { cout << "virtual fV() di B\n";
};
// atribut
```

```
private:
  int  xb;
}; // END CLBSSB

#endif // __B_H
```

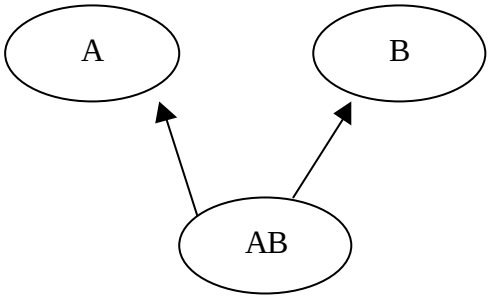
```
// File: mainAB.cc

#include "a.h"
#include "b.h"

int main() {
// KAMUS
  A  objA1;
  A  objA2( 3, 3);
  A  objA3 = objA2;
  A * PtrObjA;

  B  objB1;
  B  objB2(6);
  B  objB3 = objB2;
  B * PtrObjB;

// ALGORITMA
  cout << " Call B \n";
  objB1.PrintObjek();
  objB1.fb();
  objB1.g();

  cout << " Call A \n";
// Protected tidak boleh dipanggil oleh objek
//  objA1.protectf(5);

  objA1.PrintObjek();
  objA1.f();
  objA1.g();

  cout << " Polymorphic attachment A = B \n";
  cout << "tidak berlaku sebab HARUS POINTER!!! \n";
  objB2.PrintObjek();

  objA1 = objB2;
  objA1.f();
  objA1.g();
  objA1.PrintObjek();
  objA1.fV();


// POlymorphic attachment
```

```
 PtrObjA = new A(5,5);
 PtrObjB = new B(55);

 cout << "Polymorphic attachment lewat pointer \n";
  PtrObjA = PtrObjB;

  PtrObjA->f();
  PtrObjA->g();
  PtrObjA->PrintObjek();
  PtrObjA->fV();

  PtrObjB->f();
  PtrObjB->g();
  PtrObjB->PrintObjek();
  PtrObjB->fV();

// Cara kedua
 cout << "Polymorphic attachment lewat pointer langsung B
hidup\n";
  PtrObjA = new B(555);
  PtrObjA->f();
  PtrObjA->g();
  PtrObjA->PrintObjek();
  PtrObjA->fV();


  return 0;
}
```

Program utama

```
// File : mainAB.cc
#include "A.h"
#include "B.h"
#include "boolean.h"

int main () {
// Kamus
  A a1, a2;
  B b1, b2;

// Algoritma

// hidupkan objek: sudah dilakukan pd saat deklarasi

// Call feature
   a1.f();
   a1.g();
   a1.PrintObjek();
// Call feature
   b1.fg();
   b1.PrintObjek();

// Polymorphic attachment
   a1=b2;
   a1.f();
   a1.g();
   a1.PrintObjek();

// Print untuk menunjukkan efek

Return 0;
}
```

MULTIPLE INHERITANCE




```
// File: A.h
class A {
public:
// Konstruktor
  A(0)
// Destruktor
  ~A;
// Method
     void PrintObjek ();
      int f();
      int g();
// atribut
private:
  Int  xa;

}; // END CLASSA
```

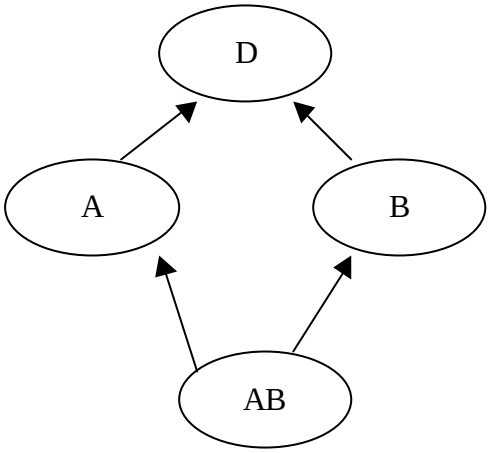
```
// File: B.h
class B {
public:
// Konstruktor
  B(0)
// Destruktor
  ~B;
// Method
     void PrintObjek ();
      int f();
      int g();

// atribut
private:
  Int  xB;

}; // END CLASSB
```

```
class AB {
Inherit A;
Inherit B;
public:
// Konstruktor
  AB(0)
// Destruktor
  ~AB;
// Method
     void PrintObjek ();

// Tambahan method publik
      int fAB ();

// tambahan atribut private
private:
  Int xab;
}; // END CLASSAB
```

Buatlah program utama sehingga menunjukan polymorphic attachment

```
// File : mainAB.cc
int main() {

// Kamus
  A a1, a2;
  B b1, b2;
  AB ab1, ab2;

// Panggil beberapa method
    a1. PrintObjek();
    b1. PrintObjek();
    ab1. PrintObjek();

// Algoritma
  a2=a1;

  ab1=a1; // ilegal!

  a1=b1; // apa akibatnya ?

  a2=ab1; //apa akibatnya
  a2.Printobjek();

return 0;
}
```

REPEATED INHERITANCE




```
// File: D.h
class D {
public:
// Konstruktor
  D(0)
// Destruktor
  ~D;
// Method
     void PrintObjek ();
      int f();
      int g();
// atribut
private:
  Int  xd;
}; // END CLASSA
```

```
// File: A.h
class A {
Inherit D;
public:
// Konstruktor
  A(0)
// Destruktor
  ~A;
// Method
     void PrintObjek ();
      int f();
      int g();
// atribut
private:
```

```
  Int  xa;
}; // END CLASSA
```

```
// File: B.h
class B {
Inherit D;
// Konstruktor
  B(0)
// Destruktor
  ~B;
// Method
     void PrintObjek ();
      int f();
      int g();
// atribut
private:
  Int  xB;
}; // END CLASSB
```

```
class AB {
Inherit A;
Inherit B;
public:
// Konstruktor
  AB(0)
// Destruktor
  ~AB;
// Method
     void PrintObjek ();
// Tambahan method publik
      int fAB ();
// tambahan atribut private
private:
  Int xab;
}; // END CLASSAB
```

POINT GENERIK

```
// File : pointG.h
// Spesifikasi kelas POINT yang generik
#include "Boolean.h"

class POINTG [G] {
public:
// Method public

// Konstruktor
   POINTG (G,G);
   // membentuk sebuah POINTG
// Destruktor
   ~POINTG ;
// Selektor
   int Absis ();
   int Ordinat ();
// Operasi I/O
   void PrintObj();
   // Print nilai Current Objek
// Relasional
   Boolean "<" (POINTG P1,P2);
   // true jika P1< P2 : Kuadrannya lebih kecil
   // Perhatikan Current_Object tidak dipakai!
   Boolean "<" (POINTG P1);
   // true jika P1< Current_Object : Kuadrannya lebih
kecil
   // Perhatikan Current_Object dipakai!
// Predikat lain
   Boolean IsOrigin ();
// Aritmatika
   POINTG "+" (POINTG P1, P2);

// Operasi lain
   int Kuadran();
   // Mengirim kuadran dari Current Obj
   // P bukan Origin dan tidak terletak pd sumbu X atau Y
// Private
private:
  G x;   // absis
  G y;   // ordinat
}; // END CLASS POINTG
```

Program utama yang mengaktifkan POINTG

```
// File mainpg.h
#include “pointG.h”
int main(){
// Kamus
    // instansiasi class sekaligus hidupkan objek
   POINTG (int) Pi;
   POINTG (float) Pf;
   POINTG (int) *Ptri;
   POINTG (float) *Ptrf;
// Algoritma
 Pi.Printobjek();
 Pf.Printobjek();
// Bgmn dengan assignment ini : Casting ??
   Pi=Pf;
// Alokasi Ptrf
   Ptrf= new POINTG(float); // benarkah ??
   (* Ptrf)= Pti;
   (*Ptrf).printObject();
return 0;
}
```