BAB VI
OBYEK DAN KELAS

Dalam C dan bahasa pemrograman prosedural lainnya, pemrogramannya berorientasi kepada aksi, sedangkan pemrograman C++ cenderung berorientasi pada obyek. Disamping itu, unit program dalam C adalah **fungsi**, tetapi di dalam C++, unit program adalah **kelas** (*class*) dimana suatu **obyek** (*object*) secara mendekati kenyataan dapat diciptakan.

Kelas di dalam C++ merupakan pengembangan dari notasi ***struct*** di dalam C biasa, yaitu struktur untuk mendefinisikan elemen-elemen data. Misalnya didefinisikan data tentang waktu yang terdiri dari jam, menit, dan detik sebagai berikut:

```
struct Time {
    int hour;          // nilai jam 0 - 23
    int minute;        // nilai menit 0 - 59
    int second;        // nilai detik 0 - 59
};
```

Definisi **Time** ini terdiri dari tiga buah anggota data (***data members***) yang semuanya bertipe **int**, yaitu **hour**, **minute**, dan **second**. Anggota struktur dapat terdiri dari berbagai macam tipe, kecuali tipe **Time** itu sendiri. Berikut adalah contoh-contoh deklarasi variabel dan akses anggota data dari struktur yang telah didefinisikan sebelumnya (uraian lengkap lihat Buku Catatan Kuliah Algoritme dan Pemrograman):

```
Time timeObyek, timeArray[10], *timePtr;

cout << timeObyek.hour;          // mengakses data hour dari obyek
cout << timePtr -> hour;         // mengakses data hour dari obyek yang ditunjuk
                                 // oleh pointer timePtr
```

Untuk melakukan proses pengolahan terhadap obyek data ini diperlukan fungsi-fungsi yang ditulis secara terpisah dari definisi strukturnya. Sebagai contoh, fungsi berikut disusun untuk menuliskan obyek **Time** dengan bentuk **hh:mm:ss**.

```
void printTime (const Time &t)
{
    cout << (t.hour < 10 ? "0" : "") << t.hour << ":"
         << (t.minute < 10 ? "0" : "") << t.minute << ":"
         << (t.second < 10 ? "0" : "") << t.second << ":" ;
}
```

Pada contoh ini terlihat adanya pemisahan antara data dan prosedur/fungsi, yang berbeda dengan konsep pemrograman beorientasi obyek dimana antara keduanya dibungkus ke dalam satu kesatuan dengan menggunakan tipe ***class***. Pembungkusan (***encapsulation***) inilah yang merupakan salah satu ciri utama dari OOP seperti yang telah dibahas pada bab sebelumnya.

TIPE *CLASS*

Kata kunci ***class*** dalam C++ digunakan untuk memodelkan suatu obyek terdiri dari dua anggota, yaitu atribut yang direpresentasikan sebagai anggota data (***data members***) dan sifat atau operasi-operasi atau prosedur-prosedur yang direpresentasikan sebagai fungsi anggota (***member functions***). Fungsi anggota sering disebut dengan metode pada bahasa OOP lainnya dan digunakan untuk memberi pesan pada obyek yang bersangkutan.

Setiap anggota dari obyek ini dikelompokkan berdasarkan sifat akses terhadap anggota-anggotanya, baik dari dalam obyek itu sendiri maupun dari obyek lainnya. Ada tiga sifat yang dapat diberikan, yaitu ***public***, ***private***, dan ***protected***. Perhatikan contoh berikut:

```
class Time {
public:
    Time();                         // default constructor
    void setTime(int, int, int);    // member function
    voit print();
    ~Time();                        // destructor
private:
    int hour;                       // data members
    int minute;
    int second;
};
```

Anggota kelompok private hanya dapat diakses oleh fungsi anggota dari kelas itu sendiri dan kelas lain yang mempunyai hubungan friend (akan dijelaskan pada bab berikutnya). Hal ini berbeda dengan kelompok public yang dapat diakses dari bagian-bagian lainnya dari program, sedangkan kelompok protected hanya dapat diakses oleh fungsi anggota dari kelas yang bersangkutan dan kelas-kelas lain yang menjadi turunannya (akan dijelaskan pada bab berikutnya).

Pembatasan akses itu merupakan *information hiding* yang berguna antara lain jika representasi data dari kelas berubah, hanya fungsi anggota (bukan program dari pengguna) yang perlu diambil, dan jika ada kesalahan dalam manipulasi anggota data dari kelas, hanya fungsi anggota yang perlu di-*debug*, bukan seluruh program.

Fungsi anggota yang namanya sama dengan nama kelas disebut ***default constructor***, dan fungsi anggota yang namanya terdiri tanda *tilde* ("~") diikuti dengan nama kelas disebut ***destructors***. Constructor akan secara otomatis dipanggil oleh kompilator untuk inisialisasi obyek, sedangkan destructor akan secara otomatis dipanggil oleh kompilator untuk dealokasi memori.

Nama kelas akan berlaku sebagai kata kunci penentu untuk suatu tipe data baru, misalnya dengan kelas Time tersebut dapat dibuat deklarasi variabel sebagai berikut:

```
Time timeObyek1, timeObyek2;
```

Pada saat deklarasi inilah diproses fungsi anggota **Time()** yang merupakan default constructor dari kelas ini. Namun demikian, kelas tersebut belum dapat digunakan

karena implementasi setiap fungsi anggota belum dibuat. Perhatikan contoh implementasi berikut:

```
Time :: Time()
{
    hour = minute = second = 0;
}

void Time :: setTime( int h = 0, int m = 0, int s = 0 )
{
    hour = h;
    minute = m;
    second = s;
}

voit Time :: print()
{
    cout << (hour < 10 ? "0" : "") << hour << ":"
         << (minute < 10 ? "0" : "") << minute << ":"
         << (second < 10 ? "0" : "") << second << ":" ;
}
```

Operator "::" adalah scope-operator, yang menunjukkan bahwa fungsi yang dituliskan di belakangnya merupakan anggota dari kelas yang ditulis di depannya. Dengan demikian, misalnya **void Time :: setTime(h = 0, m = 0, s = 0)** menunjukkan bahwa fungsi setTime merupakan anggota dari kelas Time.

Suatu fungsi dalam C++ dapat dipanggil dengan nol atau lebih argumen, seperti contoh Time() yang tidak mempunyai argumen, dan setTime(h = 0, m = 0, s = 0) mempunyai tiga argumen dengan masing-masing mempunyai nilai *default*. Artinya, fungsi itu dapat dipanggil dengan beberapa cara seperti contoh berikut:

```
setTime( 10, 30, 25 );      // argumen h=10, m=30, s=25
setTime( 10, 30);           // argumen h=10, m=30, s=0
setTime();                  // argumen h=0, m=0, s=0
```

Dengan definisi dan deklarasi kelas Time yang telah dibuat, maka dapat digunakan sebagai tipe baru seperti pada program berikut (misalnya definisi dan deklarasi kelas Time telah disimpan pada file **time.h**):

```
#include <iostream>
#include "time.h"

main() {
    Time t1;
    t1.setTime(10,30,25);       // obyek t1 memiliki hour=10, minute=30, second=25
    t1.print;                   // mencetak 10:30:25
}
```

Mekanisme kelas (*class*) memungkinkan pengguna mendefinisikan tipe data abstrak (ADT). Suatu kelas mempunyai empat atribut, yaitu:

1. Anggota-anggota data (*data members*) yang merupakan representasi dari kelas. Suatu kelas dapat mempunyai nol atau lebih data member dengan tipe apa saja.
2. Fungsi-fungsi anggota (*members function*) yaitu operasi-operasi yang dapat diterapkan pada obyek-obyek dari kelas itu. Sebuah kelas dapat mempunyai nol atau lebih fungsi anggota.
3. Tingkat akses.
   Setiap anggota suatu kelas dapat ditentukan tingkat aksesnya sebagai **private**, **protected** atau **public**. Tingkat-tingkat ini mengontrol akses ke anggota kelas. Biasanya representasi kelas bersifat private, sedang operasi-operasinya bersifat public. Spesifikasi private/public ini membentuk *information hidding*.
4. Nama kelas yang berlaku sebagai type-specifier.

Definisi Kelas

Definisi kelas terdiri dari dua bagian, yaitu:

1. Class head : kata kunci class diikuti oleh nama kelas
2. Class body : diapit kurung kurawal { }, diikuti tanda titik koma atau daftar deklarasi.

Misalnya:

```
class screen (/*……..*/) ;
class screen (/*……..*/) s1 , s2 ;
```

Dalam class body ditentukan data members, member function dan tingkat-tingkat dari information hidding. Deklarasi untuk data members sama dengan deklarasi untuk variabel, kecuali bahwa inisialisasi eksplisit tidak diperbolehkan. Contoh:

```
class screen  {
/* height dan width menyatakan jumlah kolom
/*crusor menunjukkan posisi screen sekarang
/* scr menunjuk ke array height*width
/*
     short height, width ;          // menyatakan jumlah kolom
     char *crusor ;                 // posisi screen saat ini
     char scr ;                     // menunjuk ke array heightwidth
};
```

Kelas dapat mendefinisikan data members berupa reference ke tipe dirinya, misalnya:

```
class linkScreen  {
     screen window ;
     linkScreen *next ;
     linkScreen *prev ;
};
```

Member functions dari suatu kelas tidak dideklarasikan di dalam class body. Contoh:

```
class Screen  {
Public :
      void home ( ) { cursor = scr ; }
      void move ( int, int ) ;
      char get ( ) { return *cusor ;}
      char get (int , int ) ;
      void checkRange ( int , int ) ;
      …
      …
      …
};
```

Member function yang didefinisikan di dalam class-body secara otomatis dianggap inline, dan member function yang lebih dari satu atau dua baris sebaiknya didefinisikan di luar class-body. Contoh:

```
#include “screen.h”
#include <iostream.h>
#include <stdlib.h>

void screen :: checkRange (int row , int col )        // validasi koordinat
{
      if (row < 1 || row > height || col < 1 || col > width )  {
      cerr  <<  “Koordinat screen ( “ << row << “, “<< col << “ )  keluar batas \n “ ; exit (-1);
      }
}
```

Member function yang didefinisikan di luar class body, jika ingin dibuat inline harus secara eksplisit. Contoh:

```
inline void screen :: move ( int r , int c )   // memindahkan cursor ke posisi absolut (r,c)
{
      clockRange (r,c) ;                        // koordinat sah ?
      int row = (r-1) *width ;
      cursor = ser + row + c-1 ;
}
```

Member function dibedakan dari fungsi-fungsi biasa dalam hal-hal:

1. Member function punya hak akses penuh ke anggota-anggota kelas yang bersifat public maupun private. Pada umumnya, fungsi-fungsi biasa dapat di akses hanya oleh anggota-anggota kelas yang public. Tentu saja pada umumnya member functions suatu kelas tidak mempunyai hak akses ke anggota-anggota non public dari kelas-kelas lainnya.
2. Member function didefinisikan dalam scope dari kelasnya, fungsi-fungsi biasa didefinisikan dalam scope file.

Information Hidding

Information hidding adalah suatu mekanisme untuk membatasi akses user ke representasi internal dari suatu kelas. Ada tiga tingkatan akses, yaitu:

1. Public member dapat diakses dari seluruh program. Yang menjadi public member sebaiknya hanya member function yang dimaksudkan untuk mendefinisikan operasi-operasi fungsional dari kelas itu.
2. Protected member berlaku sebagai public member bagi derivided class (kelas turunan) . Protected member berlaku sebagai private member bagi bagian lainnya dari program.
3. Private member dapat diakses hanya oleh member function dan friend dari kelasnya.

Jika tidak diberi label, secara otomatis bagian yang segera mengikuti tanda kurung kurawal pembuka "{" bersifat private.

OPERATOR OVERLOADING

Keistimewaan program C++ lainnya adalah adanya fasilitas *operator overloading*, yaitu penamaan fungsi tidak seperti pada umumnya, melainkan nama fungsi berupa kata kunci **operator** diikuti dengan lambang operator yang digunakan. Misalnya nama fungsi operator= yang dapat dipanggil dengan hanya menuliskan operator = seperti pada operasi penugasan biasa, sehingga dapat dibuat fungsi penugasan versi sendiri untuk tipe-tipe data tertentu, seperti kita dapat menyatakan **Time t2 = t1**, dan sebagainya. Adapun operator yang dapat digunakan seperti tercantum pada daftar berikut:

+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete

Berikut adalah definisi dan deklarasi kelas Time yang lebih lengkap:

```
#include <iostream.h>>

class Time {
friend ostream& operator << (ostream&, Time&);          // untuk fungsi cout << obyek
public:
    Time(int h=0, int m=0, int s=0);                    // default constructor
    Time& operator++();                                 // prefix increment
    Time& operator+=(const Time&);                      // operator +=
    Time& operator+=(const int);
    Time& operator+(const Time&);                       // operator +
    Time& operator+(const int);
    ~Time() {cout << "Program selesai" << endl;}        // destructor
private:
    void format();                                      // memformat nilai obyek
    int hour;                                           // agar sesuai, misalnya:
    int minute;                                         // 12:65:70 → 13:06:10
    int second;
```

```
};

Time::Time(int h=0, int m=0, int s=0) {
	hour = h;
	minute = m;
	second = s;
}

void Time::format() {
	int tm = minute, ts = second;
	if (second >>= 60) {
		second %= 60;
		minute += (int) (ts/60);
		tm = minute;
	}
	if (minute >>= 60) {
		minute %= 60;
		hour += (int) (tm/60);
	}
	if (hour >>= 24)  hour %= 24;
}

Time& Time::operator+= (const Time& t) {
	second = second + t.second;
	minute = minute + t.minute;
	hour = hour + t.hour;
	format;
	return *this;
}

Time& Time::operator+= (const int t) {
	second = second + t;
	format;
	return *this;
}

Time& Time::operator++ () {
	second++;
	format;
	return *this;
}

Time& Time::operator+ (const Time& t) {
	second = second + t.second;
	minute = minute + t.minute;
	hour = hour + t.hour;
	format(hour, minute, second);
	return *this;
}

Time& Time::operator+ (const int t) {
	second = second + t;
	format(hour, minute, second);
	return *this;
}
```

```
ostream& operator << (ostream& ostr, Time& t) {
    return ostr   << (t.hour < 10 ? "0" : "") << t.hour << ":"
                  << (t.minute < 10 ? "0" : "") << t.minute << ":"
                  << (t.second < 10 ? "0" : "") << t.second;
}
```

Untuk lebih memberikan gambaran lagi tentang penanganan suatu obyek, berikut didefenisikan suatu tipe kelas intArray untuk pengelolaan array bilangan bulat yang lebih fleksibel dari yang biasanya.

```
#include <iostream.h>
const int arraySize = 20;                                   //ukuran default

class intArray  {
public :
    intArray (int sz = ArraySize );
    intArray ( const int * , int );
    intArray ( const IntArray & );
    ~intArray ( ) { delete [ ] ia ; }
    intArray & operator  = ( const IntArray & );
    int & operator [ ] ( int );
    int  getSize ( ) { return size  ; }
protected :
    init ( const int *,  int );
    int size;
    int *ia;
};
```

Selanjutnya akan diuraikan bagian-bagian dari implementasi setiap fungsi anggota dari kelas intArray.

```
void intArray :: init ( const int *array , int size ) {
    ia  = new int [ size = sz ] ;              // alokasi array sebanyak sz unsur
    assert (ia  != 0  ) ;                      // new memberikan pointer ke ruangan ini
    for  ( int i  = 0   ; i < size ; ++ i )
        ia[i] = (array != 0   ) ?  array[i] : 0;
}

intArray :: intArray (int sz) {
    init (0, sz);
}

intArray :: intArray  (const int *array , int sz ) {
    init (array, sz );
}

intArray :: intArray ( const intArray &iA)  {
    init (iA.ia, ia.size);
}

intArray&  intArray :: operator= ( const intArray & iA)  {
    if (this ==&iA) return *this;              // assigment ke diri sendiri (iA = iA)
    delete ia;                                 // lepaskan memori dulu
```

```
        init (iA.ia, iA.size);
        retrun *this;
    }

    inline int& intArray :: operator [ ] (int indek) {
        return ia[indek];
    }
```

Contoh penggunaan kelas intArray adalah sebagai berikut:

```
    const int sz = 10;
    int ukuran  = 256;
    intArray arrayKu ( sz) , arrayMu (ukuran);
    intArrray *pA = &arrayKu;
    intArray arrayDia;
    int ia[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    intArray iA3 (ia, 10);
    Intarray   iA4 = arrayKu  ;
    int  ix = arrayKu[3];
    arrayKu[2] = 4;
```