Pertemuan 6

6. Pointer

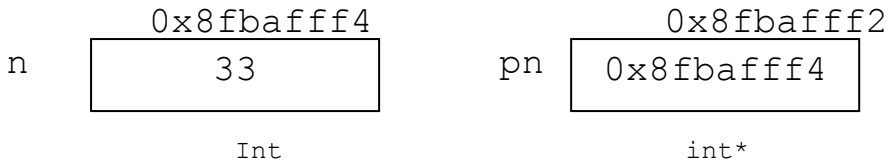
Obyektif Praktikum :

1. Mengerti konsep dasar apa itu pointer dan penggunaannya
2. Mengerti apa itu Reference dan hubungannya dengan pointer
3. Mengerti dan dapat menggunakan pointer pada C++
4. Mengetahui hubungan antara pointer dan memory

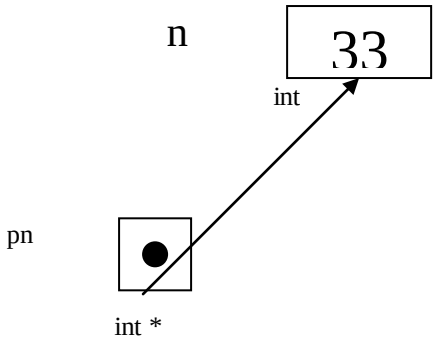
P.6.1 POINTER

Operator reference & mengembalikan alamat memory dari suatu variable ketika digunakan. Kita juga dapat menyimpan suatu alamat pada suatu variable lain. Type dari variable yang dapat menyimpan suatu alamat disebut Pointer. Variabel pointer harus mendapatkan type " pointer to T " , dimana T adalah type dari objek dimana pointer itu ditunjuk. Seperti yang sudah dijelaskan pada akhir penjelasan reference bahwa "pointer to T" disimbolkan dengan T* , sebagai contoh bahwa suatu alamat dari variable dengan tipe int dapat disimpan dalam variabel pointer dengan tipe int*.

Dapat digambarkan sbb :



Variabel n dideklarasi dengan angka 33 , dengan alamat 0x8fbafff4. variable pn dideklarasi sama dengan &n yang merupakan alamat dari n, jadi nilai dari pn adalah 0x8fbafff4 , tetapi pn adalah objek yang berbeda yang memiliki nilai alamat dari n , dikatakan berbeda karena pn memiliki alamat berbeda yaitu 0x8fbafff2.



variabel pn disebut "pointer" karena bernilai "points" atau alamat dari lokasi nilai yang lain. Nilai dari pointer tsb adalah suatu alamat. Alamat tersebut tergantung pada computer dimana program itu berjalan. Dengan gambar diatas dapat pula dijelaskan bahwa 2 variable n dan pn dimana pn adalah pointer dari n dan n memiliki nilai 33. pointer dapat dikatakan juga sebagai "locator" : yang mengalokasikan objek lain melalui alamat.

Operator Alamat (Address operator (&))

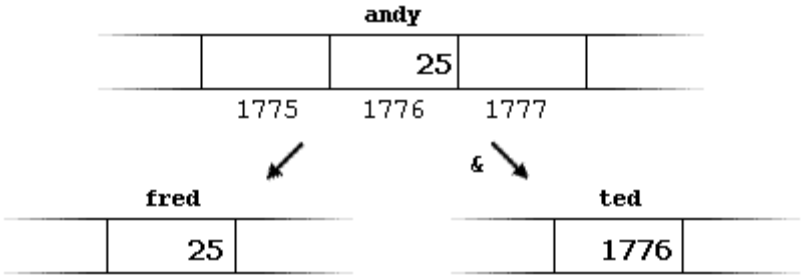
Pada saat pendeklarasian variable, user tidak diharuskan menentukan lokasi sesungguhnya pada memory, hal ini akan dilakukan secara otomatis oleh kompilerdan operating sysem pada saat run-time. Jika ingin mengetahui dimana suatu variable akan disimpan, dapat dilakukan dengan memberikan tanda *ampersand* (&) didepan variable , yang berarti ***"address of"***. Contoh :

```
ted = &andy;
```

Akan memberikan variable **ted** alamat dari variable **andy**, karena variable **andy** diberi awalan karakter *ampersand* (&), maka yang menjadi pokok disini adalah alamat dalam memory, bukan isi variable. Misalkan **andy** diletakkan pada alamat **1776** kemudian dituliskan instruksi sbb :

```
andy = 25;
fred = andy;
ted = &andy;
```

Maka hasilnya :

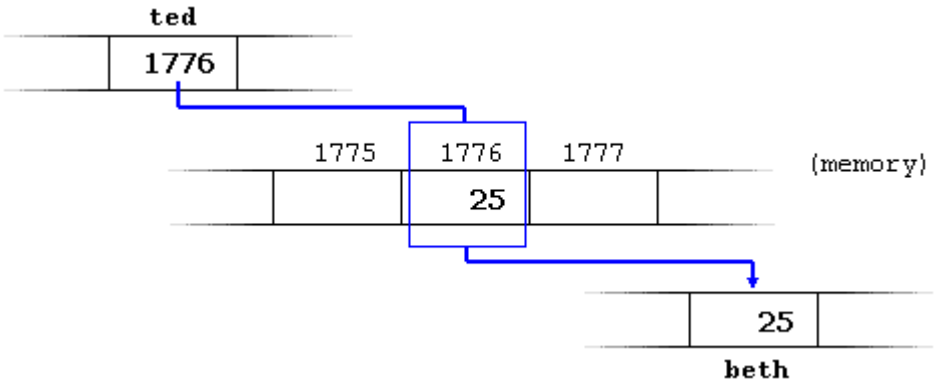


Operator Reference (*)

Dengan menggunakan pointer, kita dapat mengakses nilai yang tersimpan secara langsung dengan memberikan awalan operator *asterisk* (*) pada identifier pointer, yang berarti ***"value pointed by"***. Contoh :

```
beth = *ted;
```

(dapat dikatakan:"beth sama dengan nilai yang ditunjuk oleh ted") **beth** = **25**, karena **ted** dialamat **1776**, dan nilai yang berada pada alamat **1776** adalah **25**.



Ekspresi dibawah ini semuanya benar, perhatikan :

andy == 25
&andy == 1776
ted == 1776
*ted == 25

Ekspresi pertama merupakan *assignation* bahwa **andy=25;**. Kedua, menggunakan operator alamat (address/derefence operator (&)), sehingga akan mengembalikan alamat dari variabel **andy**. Ketiga bernilai

benar karena *assignation* untuk **ted** adalah **ted = &andy;**. Keempat menggunakan reference operator (*) yang berarti nilai yang ada pada alamat yang ditunjuk oleh **ted**, yaitu **25**. Maka ekspresi dibawah ini pun akan bernilai benar :

```
*ted == andy
```

Deklarasi variable bertipe pointer
Format deklarasi pointer :

```
type * pointer_name;
```

Dimana ***type*** merupakan tipe dari data yang ditunjuk, bukan tipe dari pointer-nya. Contoh :

```
int * number;
char * character;
float * greatnumber;
```

Array dan Pointer

Identifier suatu array equivalen dengan alamat dari elemen pertama, pointer equivalen dengan alamat elemen pertama yang ditunjuk. Perhatikan deklarasi berikut :

```
int numbers [20];
int * p;
```

maka deklarasi dibawah ini juga benar :

```
p = numbers;
```

p dan **numbers** equivalen, dan memiliki sifat (*properties*) yang sama. Perbedaannya, user dapat menentukan nilai lain untuk pointer **p** dimana **numbers** akan selalu menunjuk nilai yang sama seperti yang telah didefinisikan. **p**, merupakan *variable pointer*, **numbers** adalah *constant pointer*. Karena itu walaupun instruksi diatas benar, tetapi tidak untuk instruksi dibawah ini :

```
numbers = p;
```

karena **numbers** adalah array (constant pointer), dan tidak ada nilai yang dapat diberikan untuk identifier konstant (*constant identifiers*).

Inisialisasi Pointer
Contoh :

```
int number;
int *tommy = &number;
```

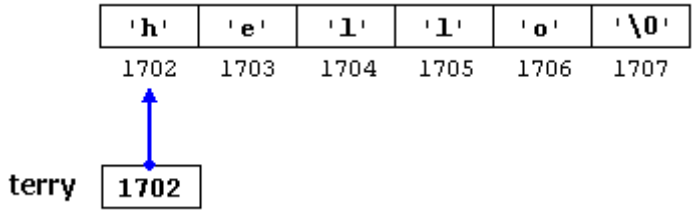
Equivalen dengan :

```
int number;
int *tommy;
tommy = &number;
```

Seperti pada array, inisialisasi isi dari pointer dapat dilakukan dengan deklarasi seperti contoh berikut :

```
char * terry = "hello";
```

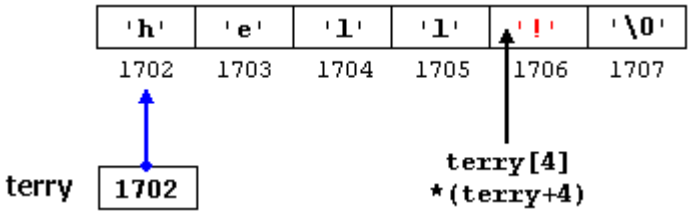
Misalkan "hello" disimpan pada alamat 1702 dan seterusnya, maka deklarasi tadi dapat digambarkan sbb :



terry berisi nilai **1702** dan bukan **'h'** atau **"hello"**, walaupun **1702** menunjuk pada karakter tersebut. Sehingga jika akan dilakukan perubahan pada karakter **'o'** diganti dengan tanda **'!'** maka ekspresi yang digunakan ada 2 macam :

```
terry[4] = '!';
*(terry+4) = '!';
```

Penulisan **terry[4]** dan ***(terry+4)**, mempunyai arti yang sama. Jika digambarkan :



Pointer Arithmatika

Contoh, *char* memerlukan 1 byte, *short* memerlukan 2 bytes dan *long* memerlukan 4. Terdapat 3 buah pointer :

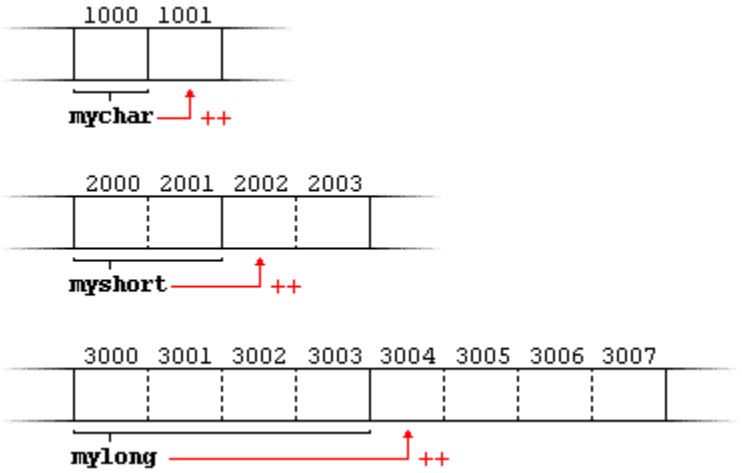
```
char *mychar;
short *myshort;
long *mylong;
```

ekspresi diatas akan menunjuk pada lokasi dimemory masing-masing **1000**, **2000** and **3000**, sehingga jika dituliskan :

```
mychar++;
myshort++;
mylong++;
```

mychar, akan bernilai 1001, myshort bernilai 2002, dan mylong bernilai 3004. Alasannya

adalah ketika terjadi pertambahan maka akan ditambahkan dengan tipe yang sama seperti yang didefinisikan berupa ukuran dalam bytes.



Perhatikan ekspresi dibawah ini :

```
*p++;
*p++ = *q++;
```

Ekspresi pertama equivalen dengan `*(p++)` dan yang dilakukan adalah menambahkan **p** (yaitu alamat yang ditunjuk, bukan nilai yang dikandungnya).
Ekspresi kedua, yang dilakukan pertama adalah memberikan nilai `*q` ke `*p` dan kemudian keduanya ditambahkan 1 atau dengan kata lain :

```
*p = *q;
p++;
q++;
```

void pointer
Tipe pointer *void* merupakan tipe khusus. *void* pointers dapat menunjuk pada tipe data apapun, nilai integer value atau float, maupun string atau karakter. Keterbatasannya adalah tidak dapat menggunakan operator asterisk (*), karena panjang pointer tidak diketahui, sehingga diperlukan operator *type casting* atau *assignations* untuk mengembalikan nilai *void* pointer ketipe data sebenarnya.

Pointer untuk functions
C++ memperbolehkan operasi dengan pointer pada function. Kegunaan yang utama adalah untuk memberikan satu function sebagai parameter untuk function lainnya. Deklarasi pointer untuk function sama seperti prototype function kecuali nama function dituliskan diantara tanda kurung `()` dan operator asterisk `(*)` diberikan sebelum nama.

Structures

Data structures

Struktur data merupakan kumpulan berbagai tipe data yang memiliki ukuran yang berbeda di kelompokan dalam satu deklarasi unik, dengan format sbb :

struct *model_name* **{**
*type1 element1***;**
*type2 element2***;**

```
  type3 element3;
  .
  .
} object_name;
```

dimana *model_name* adalah nama untuk model tipe stukturnya dan parameter optional *object_name* merupakan identifier yang valid untuk objek sturktur. Diantara kurung kurawal { } berupa tipe dan sub-identifier yang mengacu ke elemen pembentuk struktur. Jika pendefinisian stuktur menyertakan parameter *model_name* (optional), maka parameter tersebut akan menjadi nama tipe yang valid ekuivalen dengan struktur. Contoh :

```
struct products {
  char name [30];
  float price;
} ;

products apple;
products orange, melon;
```

Didefinisikan model struktur **products** dengan dua field : **name** dan **price**, dengan tipe yang berbeda. Kemudian tipe struktur tadi (**products**) digunakan untuk mendeklarasikan tiga objek : **apple**, **orange** dan **melon**.

Ketika dideklarasikan, **products** menjadi tnama tipe yang valid seperti tipe dasar *int*, *char* atau *short* dan dapat mendeklarasikan objects (variables) dari tipe tersebut. Optional field yaitu *object_name* dapat dituliskan pada akhir deklarasi struktur untuk secara langsung mendeklarasikan object dari tipe struktur. Contoh :

```
struct products {
  char name [30];
  float price;
} apple, orange, melon;
```

Sangat penting untuk membedakan antara structure **model**, dan structure *object*. *model* adalah *type*, dan *object* adalah *variable*. Kita dapat membuat banyak *objects* (variables) dari satu *model* (type).
Contoh diatas menjelaskan bagaimana menggunakan elemen dari struktur dan struktur itu sendiri sebagai variable normal. Contoh, **yours.year** merupakan variable valid dengan tipe **int**, dan **mine.title** merupakan array valid dari 50 *chars*.

Perhatikan **mine** dan **yours** juga berlaku sebagai valid variable dari tipe **movies_t** ketika di-pass ke-function **printmovie()**. Salah satu keuntungan dari *structures* yaitu kita dapat mengacu pada setiap elemennya atau keseluruhan blok struktur.

Pointer to structure

Sama seperti pada tipe lainnya, struktur juga dapat ditunjuk oleh pointer. Aturannya sama untuk setiap tipe data. Pointer harus dideklarasikan sebagai pointer untuk struktur :

```
struct movies_t {
  char title [50];
  int year;
};

movies_t amovie;
```

```
movies_t * pmovie;
```

amovie merupakan object dari tipe struct **movies_t** dan **pmovie** adalah pointer untuk menunjuk ke objek dari tipe struct **movies_t**. maka, deklarasi dibawah ini juga benar :

```
pmovie = &amovie;
```

Operator **->** merupakan operator penunjuk yang digunakan secara khusus bersama dengan pointer untuk struktur dan pointer untuk class. Memungkinkan kita untuk tidak menggunakan tanda kurung pada setiap anggota struktur yang ditunjuk. Dalam contoh digunakan :

```
pmovie->title
```

Atau dalam penulisan yang lain :

```
(*pmovie).title
```

Kedua ekspresi tersebut diatas : **pmovie->title** dan **(*pmovie).title** benar dan berarti evaluasi elemen **title** dari struktur yang ditunjuk (pointed by) **pmovie**. Harus dibedakan dari :

```
*pmovie.title
```

Yang ekuivalen dengan :

```
*(pmovie.title)
```

Dibawah ini merupaka tabel rangkuman, kombinasi yang mungkin terjadi antara pointer dan struktur :

Expression	Description	Equivalent
pmovie.title	Element **title** of structure **pmovie**	
pmovie->title	Element **title** of structure pointed by **pmovie**	`(*pmovie).title`
***pmovie.title**	Value pointed by element **title** of structure **pmovie**	`*(pmovie.title)`

Nesting structures

Struture juga dapat berbentuk nested (bersarang) sehingga suatu elemen dari suatu struktur dapat menjadi elemen pada struktur yang lain :

```
struct movies_t {
  char title [50];
  int year;
}

struct friends_t {
  char name [50];
  char email [50];
  movies_t favourite_movie;
  } charlie, maria;

friends_t * pfriends = &charlie;
```

Setelah deklarasi diatas, dapat digunakan ekspresi sbb :

```
charlie.name
maria.favourite_movie.title
charlie.favourite_movie.year
pfriends->favourite_movie.year
```

(Dimana 2 ekspresi terakhir ekuivalen)

User defined data types

Definition of own types (`typedef`).
C++ memungkinkan kita untuk mendefinisikan tipe berdasarkan tipe data yang sudah ada. Untuk itu digunakan keyword **`typedef`**, dengan format :

`typedef` *`existing_type new_type_name ;`*

dimana *`existing_type`* adalah tipe data dasar pada C++ dan *`new_type_name`* adalah nama dari tipe baru yang didefinisikan. Contoh :

```
typedef char C;
typedef unsigned int WORD;
typedef char * string_t;
typedef char field [50];
```

Contoh diatas telah mendefinisikan empat tipe data baru : **`C`**, **`WORD`**, **`string_t`** dan **`field`** sebagai **`char`**, **`unsigned int`**, **`char*`** dan **`char[50]`** yang akan digunakan nanti seperti berikut :

```
C achar, anotherchar, *ptchar1;
WORD myword;
string_t ptchar2;
field name;
```

Union

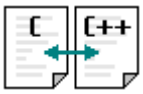
Union memungkinkan bagian dari memory dapat diakses sebagai tipe data yang berbeda, walaupun pada dasarnya mereka berada pada lokasi yang sama di memory. Pendeklarasian dan penggunaanya hampir sama dengan struktur tetapi secara fungsional berbeda :

union *model_name* **{**
*type1 element1***;**
*type2 element2***;**
*type3 element3***;**
.
.
} *object_name***;**

Semua elemen pada deklarasi *union* declaration menempati tempat yang sama dimemory. Ukuran yang

digunakan diambil dari tipe yang paling besar. Contoh :

```
union mytypes_t {
  char c;
  int i;
  float f;
  } mytypes;
```



Mendefinisikan tiga elemen :

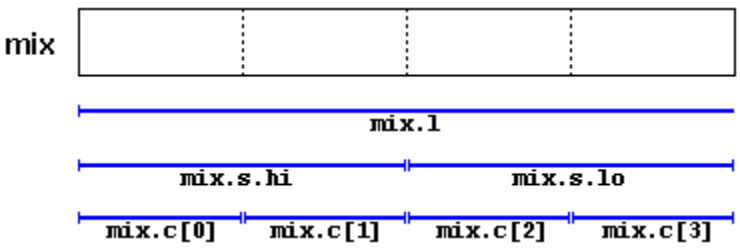
```
mytypes.c
mytypes.i
mytypes.f
```

Tiap data memiliki tipe yang berbeda, karena menempati lokasi yang sama dmemory, maka perubahan terhadap satu elemen akan mempengaruhi elemen yang lain.

Salah satu kegunaan *union*, memungkinkan untuk menggabungkan tipe dasar dengan suatu array atau struktur dari elemen yang lebih kecil. Contoh :

```
union mix_t{
  long l;
  struct {
    short hi;
    short lo;
    } s;
  char c[4];
} mix;
```

Mendefinisikan tiga nama yang memungkinkan kita untuk mengakses grup 4 bytes yang sama : **`mix.l`**, **`mix.s`** dan **`mix.c`** dan dapat digunakan menutut bagaimana kita akan mengaksesnya, sebagai **`long`**, **`short`** atau **`char.`** Tipe data yang sudah digabungkan, arrays dan structures dalam suatu union, maka dibawah ini merupakan cara pengakses-an yang berbeda :



Anonymous unions

Pada C++ terdapat option unions tanpa nama (anonymous union). Jika disertakan union dalam structure tanpa nama objek(yang dituliskan setelah kurung kurawal { }) maka union akan tidak memiliki nama dan kita dapat mengakses elemennya secara langsung dengan namanya. Contoh:

```
union
  struct {
    char title[50];
```

```
    char author[50];
    union {
      float dollars;
      int yens;
    } price;
} book;
```

anonymous union

```
struct {
  char title[50];
  char author[50];
  union {
    float dollars;
    int yens;
  };
} book;
```

Perbedaan deklarasi diatas adalah program pertama diberi nama pada union (**price**) dan yang kedua tidak. Pada saat mengakses anggota **dollars** dan **yens** dari objek. Pada program pertama :

```
book.price.dollars
book.price.yens
```

Sedangkan untuk program kedua :

```
book.dollars
book.yens
```

P.6.2 Contoh Kasus

- program mencetak nilai pointer

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

main()
{
      int n = 33;
      cout << "n  = " << n << endl;    // mencetak nilai n
      cout << "&n = " << &n << endl; // mencetak alamat n
      getch();
      return 0;
}
```

```
Dengan output seperti :
n  = 33
&n = 0x8fd4fff4
```

- Menggunakan pointer variabel

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

main()
{
 clrscr();
 int n = 33;
 cout << " n = " << n << ", &n = " << &n << endl;
 int* pn=&n; // pn memegang alamat n
 cout << "          pn = " << pn << endl;
 cout << " &pn = " << &pn << endl;
 getch();
 return 0;
}
```

Output :

n = 33, &n = 0x8fbafff4

pn = 0x8fbafff4

&pn = 0x8fbafff2

- *my first pointer*

```
// my first pointer
#include <iostream.h>
int main ()
{ int value1 = 5, value2 = 15;
  int * mypointer;
  mypointer = &value1;
  *mypointer = 10;
  mypointer = &value2;
  *mypointer = 20;
  cout << "value1==" << value1 << "/ value2==" << value2;
  return 0;
}
```

Output :

value1==10 / value2==20

Perhatikan bagaimana nilai dari **value1** dan **value2** diubah secara tidak langsung. Pertama **mypointer** diberikan alamat **value1** dengan menggunakan tanda ampersand (&). Kemudian

memberikan nilai **10** ke nilai yang ditunjuk oleh **mypointer**, yaitu alamat dari **value1**, maka secara tidak langsung **value1** telah dimodifikasi. Begitu pula untuk **value2**.

- *pointer to functions*

```
// pointer to functions
#include <iostream.h>

int addition (int a, int b)
{ return (a+b); }

int subtraction (int a, int b)
{ return (a-b); }

int (*minus)(int,int) = subtraction;

int operation (int x, int y, int (*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
  return (g);
}

int main ()
{
  int m,n;
  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  cout <<n;
  return 0;
}
```

Output :

```
8
```

Dari contoh diatas, **minus** merupakan pointer global untuk function yang mempunyai 2 parameters bertipe **int**, kemudian diberikan untuk menunjuk function **subtraction**, ditulis dalam satu baris instruksi :

```
int (* minus)(int,int) = subtraction;
```

- *pointers to structures*

```
// pointers to structures
#include <iostream.h>
#include <stdlib.h>

struct movies_t {
  char title [50];
  int year;
```

```
};

int main ()
{
  char buffer[50];
  movies_t amovie;
  movies_t * pmovie;
  pmovie = & amovie;

  cout << "Enter title: ";
  cin.getline (pmovie->title,50);
  cout << "Enter year: ";
  cin.getline (buffer,50);
  pmovie->year = atoi (buffer);
  cout << "\nYou have entered:\n";
  cout << pmovie->title;
  cout << " (" << pmovie->year << ")\n";
  return 0;
}
```

Output :

```
Enter title: Matrix
Enter year: 1999

You have entered:
Matrix (1999)
```

- *array of structures*

```
// array of structures
#include <iostream.h>
#include <stdlib.h>
#define N_MOVIES 5

struct movies_t {
  char title [50];
  int year;
} films [N_MOVIES];

void printmovie (movies_t movie);

int main ()
{
  char buffer [50];
  int n;
  for (n=0; n<N_MOVIES; n++)
  {
    cout << "Enter title: ";
    cin.getline (films[n].title,50);
    cout << "Enter year: ";
```

```
    cin.getline (buffer,50);
    films[n].year = atoi (buffer);
  }
  cout << "\nYou have entered these movies:\n";
  for (n=0; n<N_MOVIES; n++)
    printmovie (films[n]);
  return 0;
}

void printmovie (movies_t movie)
{
  cout << movie.title;
  cout << " (" << movie.year << ")\n";
}
```

Output :

```
Enter title: Alien
Enter year: 1979
Enter title: Blade Runner
Enter year: 1982
Enter title: Matrix
Enter year: 1999
Enter title: Rear Window
Enter year: 1954
Enter title: Taxi Driver
Enter year: 1975

You have entered these movies:
Alien (1979)
Blade Runner (1982)
Matrix (1999)
Rear Window (1954)
Taxi Driver (1975)
```

- ***example about structures***

```
// example about structures
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
struct movies_t {
  char title [50];
  int year;
} mine, yours;

void printmovie (movies_t movie);
```

```
int main ()
{
  char buffer [50];

  strcpy (mine.title, "2001 A Space Odyssey");
  mine.year = 1968;

  cout << "Enter title: ";
  cin.getline (yours.title,50);
  cout << "Enter year: ";
  cin.getline (buffer,50);
  yours.year = atoi (buffer);

  cout << "My favourite movie is:\n ";
  printmovie (mine);
  cout << "And yours:\n ";
  printmovie (yours);
  return 0;
}
void printmovie (movies_t movie)
{
  cout << movie.title;
  cout << " (" << movie.year << ")\n";
}
```

Output :

```
Enter title: Alien
Enter year: 1979

My favourite movie is:
 2001 A Space Odyssey (1968)
And yours:
 Alien (1979)
```

P.6.3 Latihan

1. Carilah output program di bawah ini :

```
// more pointers
#include <iostream.h>

int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
```

```
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;
}
```

2. Carilah output program di bawah ini :

```
// array of structures
#include <iostream.h>
#include <stdlib.h>
#define N_MOVIES 5

struct movies_t {
  char title [50];
  int year;
} films [N_MOVIES];

void printmovie (movies_t movie);

int main ()
{
  char buffer [50];
  int n;
  for (n=0; n<N_MOVIES; n++)
  {
    cout << "Enter title: ";
    cin.getline (films[n].title,50);
    cout << "Enter year: ";
    cin.getline (buffer,50);
    films[n].year = atoi (buffer);
  }
  cout << "\nYou have entered these movies:\n";
  for (n=0; n<N_MOVIES; n++)
    printmovie (films[n]);
  return 0;
}

void printmovie (movies_t movie)
{
  cout << movie.title;
  cout << " (" << movie.year << ")\n";
}
```

P. 6.4 Daftar Pustaka

1. Ayuliana, modul pengenalan bahasa C++, Gunadarma Jakarta, February 2004

2. Hari, Konsep Dasar Objek Oriented Programming, FTI budiluhur Jakarta, 2003

3. r.hubbard, John , schaum's outline of theory and problems of programming with C++ second edition, mcgraw-hill, New York 2000

4. http://www.cplusplus.com/

5. http://cs.binghamton.edu/~steflik/

6. http://en.wikipedia.org/wiki/c++