MODUL I
ARRAY DUA DIMENSI DAN MULTI DIMENSI

A. TUJUAN

Mendefinisikan dan membuat matriks dengan menggunakan tipe data array

B. ALOKASI WAKTU :

1 X Pertemuan = 120 menit

C. DASAR TEORI

1. PENGERTIAN ARRAY

Array atau biasa disebut larik adalah koleksi data dimana setiap elemen memakai nama yang sama dan bertipe sama dan setiap elemen diakses dengan membedakan indeks arraynya.

2. ARRAY DUA DIMENSI

- Array dua dimensi merupakan array yang terdiri dari m buah baris dan n buah kolom.
- Bentuknya dapat berupa matriks atau tabel.
- Deklarasi array :

```
Tipe_array nama_array[baris][kolom];
```

- Cara mengakses array :
  Untuk mengakses array, misalnya kita ingin mengisi elemen array baris 2 kolom 3 dengan 10 maka perintahnya adalah sbb : X[1][2] = 10;

Contoh :

Sebuah matrik A berukuran 2x3 dapat dideklarasikan sebagai berikut :

int a[2][3] = {{11, 7, 4},{12, 3, 9}} yang akan menempati lokasi memori dengan susunan berikut :

	0	1	2
0	11	7	4
1	12	3	9

Dan definisi variabel untuk setiap elemen tersebut adalah :

	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]

3. PEMROSESAN ARRAY DUA DIMENSI

Pemrosesan matriks dilakukan secara beruntun (sequential). Namun karena disusun dalam baris dan kolom, maka pemrosesan matriks juga dilakukan per baris dan per kolom. Memproses matriks artinya mengunjungi atau menelusuri setiap elemen matriks dan melakukan aksi terhadap elemen tersebut.

Contoh : Tinjaulah matriks MAT[1..5, 1..4] di bawah ini, dengan I [1..5] (ke bawah) adalah indeks baris dan J[1..4] (ke kanan) adalah indeks kolom.

	1	2	3	4
1	12	45	16	9
2	67	15	15	1
3	12	7	0	3
4	21	56	13	19
5	10	15	1	9

a. Jika penelusuran dilakukan per baris per baris (Row Major Order)

Untuk setiap baris I = 1, 2, 3, 4, 5
Telusuri elemen pada kolom J = 1, 2, 3, 4

I = 1, elemen yang diproses pada penelusuran :	12	45	16	9
I = 2, elemen yang diproses pada penelusuran :	67	15	15	1
I = 3, elemen yang diproses pada penelusuran :	12	7	0	3
I = 4, elemen yang diproses pada penelusuran :	21	56	13	19
I = 5, elemen yang diproses pada penelusuran :	10	15	1	9

b. Jika penelusuran dilakukan per kolom per kolom (Column Major Order)

Untuk setiap kolom J = 1, 2, 3, 4
Telusuri elemen pada baris I = 1, 2, 3, 4, 5

J = 1, elemen yang diproses pada penelusuran :	12	67	12	21	10
J = 2, elemen yang diproses pada penelusuran :	45	15	7	56	15
J = 3, elemen yang diproses pada penelusuran :	16	15	0	13	1
J = 4, elemen yang diproses pada penelusuran :	9	1	3	19	9

Contoh Program 1.1.

```
// Program menampilkan isi matriks
#include <iostream>
using namespace std;
void printArray(int [][3]);

int main()
{
   int matrik1 [2][3] = {{1, 2, 3}, {4, 5, 6}};
   int matrik2 [2][3] = {{1, 2, 3}, {4, 5}};
   int matrik3 [2][3] = {{1, 2}, {4}};

   printArray(matrik1);
   printArray(matrik2);
   printArray(matrik3);
   return 0;
}
void printArray(int a[][3])
{
   int i, j;

   for(i=0; i<=1; i++)
   {
      for(j=0; j<=2; j++)
   cout << a[i][j];
      cout << endl;
   }
   cout << endl;
}
```

Contoh Program 1.2.
Program menginput nilai (bilangan) ke dalam array dimensi dua dan menampilkannya.

```
/* Program menginput nilai (bilangan) ke dalam array dimensi dua
dan menampilkannya secara Row Major Order */

#include <iostream>
using namespace std;

int main()
{
   int baris, kolom, matriks[3][4];

   // Input elemen array secara Row Major Order

   cout << "Input elemen Array : \n";
   for(baris=0; baris<3; baris++)
   {
      for(kolom=0; kolom<4; kolom++)
      {
         cout << "matriks[" << baris+1 << "][" << kolom+1 << "] = ";
         cin >> matriks[baris][kolom];
      }
      cout << endl;
   }

   // Tampilkan elemen Array secara Row Major Order

   cout << "Isi array : \n";
   for(baris=0; baris<3; baris++)
   {
      for(kolom=0; kolom<4; kolom++)
         cout << "  " << matriks[baris][kolom];
      cout << endl;
   }
}
```

Contoh Program 1.3. Program penjumlahan matriks dua dimensi

```
// Program penjumlahan matriks dua dimensi

#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
  int A[3][4], B[3][4], X[3][4], Y[3][4], C[3][4], i, j;

  // Masukkan matriks A
  for(i=0; i<3; i++)
  {
    for(j=0; j<4; j++)
    {
      cout << "input data matrik A[" << i+1 << "][" << j+1 << "] : ";
      fflush(stdin); cin >> A[i][j];
    }
    cout << endl;
  }
  cout << endl;

  // Masukkan matriks B
  for(i=0; i<3; i++)
```

```
  {
    for(j=0; j<4; j++)
    {
      cout << "input data matrik B[" << i+1 << "][" << j+1 << "] : ";
      fflush(stdin); cin >> B[i][j];
    }
    cout << endl;
  }

  // Proses penjumlahan matriks A dan B
  for(i=0; i<3; i++)
  {
    for(j=0; j<4; j++)
      X[i][j] = A[i][j] + B[i][j];
  }

  // Cetak isi matriks A
  cout << "\n Matriks A \n";
  for(i=0; i<3; i++)
  {
    for(j=0; j<4; j++)
      cout << "  " << A[i][j];
    cout << endl;
  }
  cout << endl;

  // Cetak isi matriks B
  cout << "\n Matriks B \n";
  for(i=0; i<3; i++)
  {
    for(j=0; j<4; j++)
      cout << "  " << B[i][j];
    cout << endl;
  }
  cout << endl;

  // Cetak hasil penjumlahan matriks A dan B
  cout << "\n Matriks penjumlahan A+B \n";
  for(i=0; i<3; i++)
  {
    for(j=0; j<4; j++)
      cout << "  " << X[i][j];
    cout << endl;
  }
  cout << endl;
}
```

4. ARRAY MULTI-DIMENSI

- Array multi-dimensi merupakan array yang mempunyai ukuran lebih dari dua. Bentuk pendeklarasian array sama saja dengan array dimensi datu maupun array dimesi dua.
- Bentuk umumnya yaitu :

```
Tipe_array nama_array[ukuran1][ukuran2]...[ukuranN] ;
```

Contoh :

```
float X[2][4][3] ;
```

X[0][0][0]	X[0][0][1]	X[0][0][2]
X[0][1][0]	X[0][1][1]	X[0][1][2]
X[0][2][0]	X[0][2][1]	X[0][2][2]
X[0][3][0]	X[0][3][1]	X[0][3][2]

X[1][0][0]	X[1][0][1]	X[1][0][2]
X[1][1][0]	X[1][1][1]	X[1][1][2]
X[1][2][0]	X[1][2][1]	X[1][2][2]
X[1][3][0]	X[1][3][1]	X[1][3][2]

Contoh Program 1.4. Program array multi-dimensi

```
#include <iostream>
using namespace std;

int main()
{
   int i, j, k;
   static int data_huruf[2][8][8] = {
      {
         {1, 1, 1, 1, 1, 1, 1, 0},
        {1, 1, 0, 0, 0, 0, 1, 0},
         {1 ,1, 1, 1, 1, 1, 1, 0},
         {0, 0, 0, 0, 1, 1, 1, 0},
         {1, 0, 0, 0, 1, 1, 1, 0},
         {1, 1, 1, 1, 1, 1, 1, 0},
         {0, 0, 0, 0, 0, 0, 0, 0}
      },
      {
         {1, 1, 0, 0, 0, 1, 1, 0},
         {1, 1, 0, 0, 0, 1, 1, 0},
         {1, 1, 0, 0, 0, 1, 1, 0},
         {1, 1, 1, 1, 1, 1, 1, 0},
         {1, 1, 1, 0, 0, 1, 1, 0},
         {1, 1, 1, 0, 0, 1, 1, 0},
         {1, 1, 1, 0, 0, 1, 1, 0},
         {0, 0, 0, 0, 0, 0, 0, 0}
      }
   };

   // Tampilkan huruf
   for(i=0; i<2; i++)
   {
      for(j=0; j<2; j++)
      {
         for(k=0; k<2; k++)
         {
            if(data_huruf[i][j][k])
               cout << '\xDB';
            else
               cout << " ";  // spasi
         }
       cout << endl;
      }
      cout << endl;
   }
}
```

D. LANGKAH PRAKTIKUM

1. Kerjakan semua contoh program sehingga dapat menampilkan output.
2. Modifikasi **Contoh Program 1.2** menjadi program untuk menginput nilai (bilangan) ke dalam array dimensi dua dan menampilkannya secara **Column Major Order.**
3. Modifikasi **Contoh Program 1.2,** sehingga user dapat bebas membuat matriks dengan ordo yang diinginkan (**jumlah baris dan kolom ditentukan user**) dan **user dapat memasukkan isi dari matriks.**
4. Modifikasi **Contoh Program 1.3,**menjadi program **perkalian dua buah matriks.**

E. TUGAS PRAKTIKUM

1. Dari langkah praktikum No. 2, **cari elemen maksimum** dari matriks yang sudah dibuat. Elemen maksimum matriks dicari dengan menelusuri semua elemen matriks, mulai dari elemen MAT[1,1] sampai elemen MAT[MaksBaris, MaksKolom].

2. Buat program untuk ***Transpose Matriks*.** Misal matriks ordo 2x1 menjadi matrik ordo 1x2.

MODUL II
RECORD / STRUCT

A. TUJUAN

Mendefinisikan dan menggunakan tipe data record/struktur

B. ALOKASI WAKTU : 1 X Pertemuan = 120 menit

C. DASAR TEORI

1. PENGERTIAN RECORD/STRUCT

Seperti halnya Array, Record/Struct mempunyai sejumlah elemen yang disebut field. Kalau semua elemen array harus mempunyai tipe data yang sama, maka tiap-tiap elemen pada Record/Struct dapat memiliki tipe data yang berbeda dan akses individual / obyek dilakukan dengan menyebut namanya (field)

Field 1 ->	Nomhs	Char
Field 2 ->	Nama	Char
Field 3 ->	Nilai	Float

Bentuk umum deklarasi Struct:

```
struct namatipestruct
        {
           tipefield1 namafield1;
           tipefield2 namafield2;
            … …
           tipefieldn namafieldm;
        } namavar1, namavar2, …, namavar;
```

namavar dapat dipisah dari deklarasi tiperecordnya, sehingga menjadi :

```
typedef struct
      {
         tipefield1 namafield1;
         tipefield2 namafield2;
          … …
         tipefieldn namafieldm;
      } namatipestruct;
namatipestruct namavar1;
namatipestruct namavar2 [banyakElm];
```

untuk mengakses elemen record/structur dilakukan dengan cara :

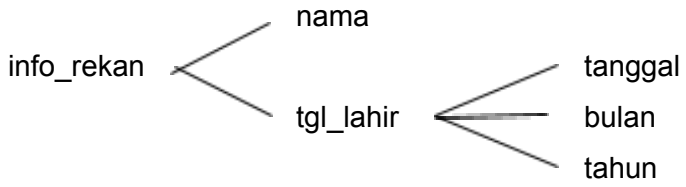
```
namavar.namafield atau namavar[i].namafield
```

Contoh :

```
struct data_tanggal
      {  int tanggal;
         int bulan;
         int tahun;
      };
```

```
struct data_rekan
      {  char nama[21];
         struct data_tanggal tgl_lahir;
      }  info_rekan;
```

Dari contoh diatas diagram struktur data :



Contoh program 2.1.

```
// Contoh variasi program mengakses elemen struktur

#include <iostream>
#include <string.h>
using namespace std;

int main()
{
   struct data_tanggal
   {
      int tanggal, bulan, tahun;
   };

   struct data_rekan
   {
      char nama[30];
      struct data_tanggal tgl_lahir;
   };

   struct data_rekan info_rekan;

   strcpy(info_rekan.nama, "Mr. Sumanto");
   info_rekan.tgl_lahir.tanggal = 3;
   info_rekan.tgl_lahir.bulan = 11;
   info_rekan.tgl_lahir.tahun = 1968;

   cout << "Nama           : " << info_rekan.nama << endl;
   cout << "Tanggal lahir  : " << info_rekan.tgl_lahir.tanggal
<< "-" << info_rekan.tgl_lahir.bulan
<< "-" << info_rekan.tgl_lahir.tahun << endl;
   getchar();
}
```

Contoh program 2.2

```
#include <iostream>
#include <string.h>
using namespace std;

int main()
{
   typedef struct {
      char no_pendaftaran[9];
      char nama_pasien[16];
      float biaya;
   } data_pasien;

   data_pasien pasien;
```

```
   cout << "No Pendaftaran : ";
   cin >> pasien.no_pendaftaran;
   cout << "Nama           : ";
   cin >> pasien.nama_pasien;
   cout << "Biaya          : ";
   cin >> pasien.biaya;

   cout << "\nNo Pendaftaran : " << pasien.no_pendaftaran << endl;
   cout << "Nama           : " << pasien.nama_pasien << endl;
   cout << "Biaya          : " << pasien.biaya << endl;
   getchar();
}
```

2. ARRAY dan STRUCT

Penggunaan struct sering dikaitkan dengan array, membentuk array dari struct.

Contoh Program 2.3.

```
#include <iostream>
#include <string.h>
using namespace std;

int main()
{
   typedef struct
   {
      char nomhs[12];
      char nama[20];
      float nilai;
   } data_nilai;

   data_nilai nilaimhs[10];

   int i;
   for(i=1; i<=2; i++)
   {
      cout << "Nama   : "; cin >> nilaimhs[i].nama;
      cout << "Nomhs  : "; cin >> nilaimhs[i].nomhs;
      cout << "Nilai  : "; cin >> nilaimhs[i].nilai;
      cout << endl;
   }
   for(i=1; i<=2; i++)
   {
      cout << "Nama   : " << nilaimhs[i].nama << endl;
      cout << "Nomhs  : " << nilaimhs[i].nomhs << endl;
      cout << "Nilai  : " << nilaimhs[i].nilai << endl;
   }
}
```

3. STRUCT dan FUNGSI

Contoh Program 2.4

```
#include <iostream>
using namespace std;

struct data_tgl
{
   int tanggal, bulan, tahun;
};

void cetak_info_tgl(struct data_tgl unit_tgl);
```

```
int main()
{
   struct data_tgl saat_proses = {13, 1, 1987};
   cetak_info_tgl(saat_proses);
}

void cetak_info_tgl(struct data_tgl unit_tgl)
{
   static char *nama_bln[] = {
      "Kode bulan salah!", "Januari", "Februari", "Maret",
      "April", "Mei", "Juni", "Juli", "Agustus", "September",
      "Oktober", "November", "Desember"
   };

   cout << unit_tgl.tanggal << " " << nama_bln[unit_tgl.bulan]
<< " " << unit_tgl.tahun << endl;
}
```

4. STRUCT DALAM POINTER

Jika suatu struct mengandung banyak field yang keseluruhannya akan diisi oleh fungsi, maka cara yang efisien adalah dengan melewatkan alamat struct. Sehingga parameter formalnya berupa pointer yang menunjuk ke struct.

Contoh Program 2.5

```
#include <iostream>
using namespace std;

struct koordinat
{
   int x, y;
};

void tukar_xy(struct koordinat *pos_xy);

int main()
{
   struct koordinat posisi = {21, 34};
   cout << "x, y semula = " << posisi.x << ", " << posisi.y;
   tukar_xy(&posisi);
   cout << "\nx, y kini = " << posisi.x<< ", " << posisi.y << endl;
}

void tukar_xy(struct koordinat *pos_xy)
{
   int z;
   z = (*pos_xy).x;
   (*pos_xy).x = (*pos_xy).y;
   (*pos_xy).y = z;
}
```

D. LANGKAH PRAKTIKUM

1. Kerjakan semua Contoh Program yang ada sehingga dapat menampilkan output.
2. Modifikasi **Contoh Program 2.2** dan **Contoh Program 2.3** sehinggadapat menginputkan nama yang terdiri lebih dari satu kata (contoh : Bagus Pramono Perkasa)

E. TUGAS PRAKTIKUM

1. Buat program record yang dapat merekam data pegawai dan menghitung total gaji sesuai hari kerjanya. Contoh output :

```
DATA GAJI PEGAWAI
nama                : Dewi Octaviani
Nip                 : 556563633
Hari kerja          : 24
Gaji perhari        : 55000

Total gaji : 1320000
```

2. Buatlah program menu yang berisi data-data penduduk (nama, alamat, umur, agama, gol darah, status..dll) yang disimpan dalam array dan dapat dilakukan penambahan data, pencarian data, penampilan data dan penghapusan data.

MODUL III
FUNGSI REKURSIF

A. TUJUAN
Mempraktekkan fungsi rekursif

B. ALOKASI WAKTU : 1 X Pertemuan = 120 menit

C. DASAR TEORI
1. PENGERTIAN
Rekursif merupakan alat untuk memecahkan masalah dalam suatu fungsi atau procedure yang memanggil dirinya sendiri.

Mengapa kita memerlukan rekursif? Karena ada beberapa kasus yang akan jauh lebih mudah diselesaikan jika menggunakan rekursif. Secara teori semua masalah yang dapat dipecahkan dengan rekursif dapat dipecahkan dengan tanpa rekursif. Meskipun dalam kenyataan ini, rekursif sangat bermanfaat sebab beberapa masalah mempunyai solusi yang sulit tanpa menggunakan rekursif.

Penggunaan rekursif kadang-kadang harus mengorbankan efisiensi dan kecepatan, disamping itu ada masalah yang sering muncul dalam rekursif, yaitu eksekusi yang tidak pernah berhenti, akibtanya memori tumpukan akan habis dan computer hank.

Pada dasarnya rekursif sering digunakan dalam perhitungan matematika, sebagai contoh pertimbangan fungsi factorial dan juga bilangan Fibonacci

2. FAKTORIAL
Salah satu contoh yang sering digunakan untuk menjelaskan rekursif adalah fungsi fakorial. Fungsi factorial dari bilangan bulat positif n didefinisikan sebagai berikut:

n!= n.(n-1)!, jika n>1

n!= 1, jika n=0, 1

contoh :

3!= 3. 2!

3!= 3. 2. 1!

3!= 3. 2. 1!

3!= 6

Kita dapat menuliskan fungsi penghitung factorial seperti dibawah ini,

```
function Faktorial (input n:byte) : longint
deskripsi
if (n = 0) or (n = 1 ) then
return  (1)
  else
return (n * Faktorial (n-1))
```

Penjelasan:

Pada baris pertama dari fungsi diatas, nilai n dicek sama dengan 0 atau 1, jika ya, maka fungsi mengembalikan nilai 1, jika tidak, fungsi mengembalikan nilai:

n * Faktorial (n -1)

Disinilah letak proses rekursif itu, perhatikan fungsi factorial ini memanggil dirinya sendiri tetapi dengan parameter (n-1)

Cara lain untuk mendefiniskan fungsi tersebut sebagai berikut:

- Dalam definisi ini, nilai 3! diekspsikan sebagai 3 x 2!, dengan kata lain untuk menghitung factorial 3, kita harus menghitung factorial lain yaitu factorial 2.
- Jika definisi nilai 2!=2 x 1!

- Factorial 1 didefinisikan dengan nilai 1.
- Jadi jika factorial 3!= 3 x 2 x 1, yang mana secara pasti mempunyai nilai sama yang diperoleh dari definisi tanpa rekursif.

3. BILANGAN FIBONACCI

Fungsi lain yang dapat diubah kebentuk rekursif adalah perhitungan Fibonacci. Bilangan Fibonacci dapat didefinisikan sebagai berikut:

f(n) = f(n-1) + f(n-2) untuk n>2
f(1) = 1
f(2) = 1
berikut ini adalah barisan bilangan Fibonacci mulai dari n=1
1 1 2 3 5 8 13 21 34

Algoritma yang dipakai

```
Function Fibonacci (input n : integer) LongInt
Deklrasi
      {tidak ada}
Deskripsi
If ((n==1)|| (n==2 ) Then
return (1)
Else
return (Fibonacci(n-1)+Fibonacci(n-2))
Endif
```

Contoh,

Untuk ukuran n= 4, proses perhitungan Fibonacci dapat dilakukan sebagai berikut:

f4 = f3+f2
f4 =(f2+f1) + f2
f4 = ((1+1) +1)
f4 = 3

Definisi rekursif harus tergantung kondisi, yang harus ada keadaan dimana kita dapat menghentikannya. Selama eksekusi program pada semua program yang menggunakan rekursi, sistem tersebut akan membangun suatu daftar nilai, yang disebut Stack (Tumpukan), sehingga ketika rekursi tersebut berakhir, program tersebut dapat kembali kelokasi yang tepat unuk melengkapi perhitungan tersebut yang untuk sementara telah dikesampingkan.

Contoh program 3.1 Program FPB

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

int FPB(int x, int y);

int main()
{
   int bil1, bil2;
   cout << "Masukkan bilangan pertama  : "; cin >> bil1;
   cout << "Masukkan bilangna kedua    : "; cin >> bil2;
   cout << "Faktor Persekutuan Terbesar : " << FPB(bil1, bil2) << endl;
}

int FPB(int x, int y)
{
   float hasil;
   if(y==0)hasil = x;
   else hasil = FPB(y, (x%y));
   return hasil;
}
```

Contoh program 3.2 Program Faktorial

```
#include <iostream>
using namespace std;

int faktorial(int a);

int main()
{
   int n;
   cout << "Masukkan bilangan : "; cin >> n;
   cout << "Faktorial " << n << " = " << faktorial(n) << endl;
}

int faktorial(int a)
{
  int hasil;
   if(a <= 1)
      hasil = 1;
   else
      hasil = a * faktorial(a-1);

   return hasil;
}
```

D. LANGKAH PRAKTIKUM

1. Kerjakan semua Contoh Program yang ada sehingga dapat menampilkan output.

E. TUGAS PRAKTIKUM

1. Buatlah program Fibonacci.
2. Berikut program menghitung deret menggunakan perulangan. Buatlah program untuk menghitung deret tersebut menggunakan function Rekursi.
   (deret S = 1+2+3+4+…+n)

```
#include <iostream>
using namespace std;

int main()
{
   int S, n, i;
   cout << "Masukkan n : "; cin >> n;
   cout << "S(" << n << ") = ";
   i = 1;

   while(i <= n)
   {
      cout << i << " + ";
      S = S + i;
      i++;
   }
   cout << "\nS(" << n << ") = " << S << endl;
}
```

3. Buatlah program untuk menghitung deret menggunakan function Rekursi.

MODUL IV
PENCARIAN DATA *(SEARCHING)*

A. TUJUAN:

Mempraktekkan dan membandingkan metode searching menggunakan tipe data array

B. ALOKASI WAKTU : 1 X Pertemuan = 120 menit

C. DASAR TEORI

1. PENGERTIAN

Searching dalam bahasa Indonesia kita sebut pencarian. Cara mengorganisir data, akan banyak mempengaruhi efisiensi pencarian dalam menemukan suatu item ataupun informasi dari data yang ada.
Contoh dalam kehidupan sehari-hari ;

- Mencari kata di kamus
- Mencari nomor telepon di buku telepon

Contoh dalam komputasi misalnya mencari suatu harga Y apakah ada dalam suatu data Array yang berisi nilai-nilai tertentu. Array memungkinkan untuk menyimpan nilai yang bertipe sama. Metode pencarian yang umum dalam array adalah Sequential Search dan Binary search. Perbedaan dari kedua teknik ini terletak pada keadaan data.

2. PENCARIAN SEQUENTIAL (SEQUENTIAL SEARCH)

Pencarian Sequential digunakan apabila data dalam keadaan acak atau tidak terurut. Pencarian Sequential atau sering disebut Pencarian Linear menggunakan prinsip sebagai berikut data yang ada dibandingkan satu persatu secara berurutan dengan data yang dicari. Pencarian sequential dapat digunakan untuk keadaan data terurut maupun tidak terurut.

a. Sequential Search pada Array yang elemen datanya Belum Terurut

Pada Array yang elemennya belum terurut, Searching hanya bisa menggunakan metode Sequential Search. Ada dua cara yang bisa dibuat yaitu :

- Metode tanpa Sentinel
- Metode dengan sentinel

Proses pencarian sequential data belum terurut tanpa sentinel :

- pada dasarnya pencarian ini hanya melakukan pengulangan dari elemen ke-1 sampai dengan jumlah data.
- pada setiap pengulangan, dibandingkan data ke-i dengan yang dicari,
- apabila sama berarti data telah ditemukan,
- sebaliknya apabila sampai akhir pengulangan tidak ada yang sama, berarti data tidak ada.

```
/* SeqSearch_BelumUrut_nonSentinel
diasumsikan Array X sudah ada dan berisi data yang belum urut
nilai yang dicari adalah nilaiCari dan hanya ada satu
*/

#include <iostream>
using namespace std;

int main() {
   int X[10] = {20, 50, 10, 30, 90, 60, 70, 80, 40, 100};
   bool found;
   int i, nilaiCari;

   cout << "nilai yang dicari = "; cin >> nilaiCari;

   found = false;
```

Contoh Program 4.1. *SeqSearch_BelumUrut_nonSentinel*

Cara lain untuk Sequential Search pada data Array X[n], adalah dengan menyediakan satu tempat setelah elemen terakhir, yaitu pada X[n+1] dan menyimpan harga yang dicari (misal y) pada posisi tersebut. Nilai yang dicari pada posisi elemen terakhir tersebut dinamakan sentinel.

Proses pencarian sequential data belum terurut dengan sentinel :

- pada dasarnya pencarian ini sama dengan proses pencarian sequential data belum terurut tanpa sentinel yaitu melakukan pengulangan dari elemen ke-1 sampai dengan jumlah data.
- pada setiap pengulangan, dibandingkan data ke-i dengan yang dicari,
- apabila sama berarti data telah ditemukan,
- perbedaannya dengan yang tanpa sentinel adalah ketika data ditemukan tapi data tersebut adalah sentinel berarti data tidak ada.

Contoh Program 4.2. *SeqSearch_BelumUrut_Sentinel {cara1}*

```
/* SeqSearch_BelumUrut_Sentinel {cara 1}
diasumsikan Array X[0..10] sudah ada dan indeks ke 0..9 telah berisi data
yang belum urut, nilai yang dicari adalah nilaiCari dan hanya ada satu,
nilaiCari diletakkan di index ke-10
*/
#include <iostream>
using namespace std;

int main()
{
   int X[11] = {20, 50, 10, 30, 90, 60, 70, 80, 40, 100};
   int i, nilaiCari;
   cout << "Nilai yang dicari = "; cin >> nilaiCari;
   X[10] = nilaiCari;
   i=0;
   while(X[i] != nilaiCari)
      i = i + 1;
   if(i > 9)
      cout << "tidak ada " << nilaiCari << " dalam Array" << endl;
   else
      cout << nilaiCari << " ditemukan dalam Array pada index ke-" << i << endl;

}
```

Contoh Program 4.3. *SeqSearch_BelumUrut_Sentinel{cara2}*

```
/* SeqSearch_BelumUrut_Sentinel {cara 2}
diasumsikan Array X[0..10] sudah ada indeks ke 0..9 telah berisi data
yang belum terurut,
nilai yang dicari adalah y dan hanya ada satu, y diletakkan di index ke-10
*/

#include <iostream>
using namespace std;

int main()
{
   int X[11] = {20, 50, 10, 30, 90, 60, 70, 80, 40, 100};
   int i, nilaiCari;
   bool found;

   cout << "Nilai yang dicari = "; cin >> nilaiCari;

   X[10] = nilaiCari;
   found = false;
   i=0;
   while(!found)
   {
      if(X[i] == nilaiCari) found = true;
      else i = i + 1;
}

   if(i == 10)
      cout << "tidak ada " << nilaiCari << " dalam Array" << endl;
   else
      cout << nilaiCari << " ditemukan dalam Array pada index ke-" << i << endl;
}
```

b. Sequential Search pada Array yang elemen datanya Sudah Terurut

Ada dua cara, yaitu :

- Metode Sequential Tanpa Sentinel
- Metode Sequential Search Dengan Sentinel

Secara singkat proses pencarian pada Array yang elemennya sudah diurutkan adalah sebagai berikut :

- Dimulai dari elemen pertama pada Array, dilakukan pembandingan dengan elemen yang dicari. Jika elemen dalam Array masih lebih kecil dari elemen yang dicari maka pencarian diteruskan. Jika sudah lebih besar, pencarian dihentikan, dan bisa dipastikan bahwa elemen yang dicari memang tidak ada.
- Jika digunakan cara pencarian dengan sentinel (elemen yang dicari disisipkan di index setelah data terakhir), dan elemen yang dicari lebih besar dari data terakhir yang ada di Array sehingga data yang dicari sama dengan data sentinel maka dapat disimpulkan bahwa data tidak ditemukan.

Contoh Program 4.4. *SeqSearch_SudahUrut_NonSentinel*

```
/* SeqSearch_SudahUrut_NonSentinel
diasumsikan Array X sudah ada dan berisi data yang sudah terurut,
nilai yang dicari adalah nilaiCari dan hanya ada satu */
#include <iostream>
using namespace std;

int main()
{
   int X[10]={10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
   int i, nilaiCari;
   bool found;
   cout << "Nilai yang dicari = "; cin >> nilaiCari;
   found = false;
   i = 0;
   while((i < 10) & (!found) & (nilaiCari >= X[i]))
   {
      if(X[i] == nilaiCari) found = true;
      else i = i+1;
   }
   if(found)
      cout << nilaiCari << " ditemukan dalam Array pada index ke-" << i << endl;
   else
      cout << "tidak ada " << nilaiCari << " dalam Array" << endl;
}
```

Contoh Program 4.5. *SeqSearch_SudahUrut_Sentinel {cara1}*

```
/* SeqSearch_SudahUrut_Sentinel {cara 1}
diasumsikan Array X[1..nmax] sudah ada dan indeks 1..n telah berisi data yang
sudah terurut, nilai yang dicari adalah nilaiCari dan hanya ada satu */

#include <iostream>
using namespace std;

int main()
{
   int X[11] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
   int i, nilaiCari;
   bool found;

   cout << "nilai yang dicari = "; cin >> nilaiCari;
   X[10] = nilaiCari;
found = false;
   i = 0;
   while((!found) & (X[i] < nilaiCari))
      i = i + 1;  // tidak ada pengecekan ketemu atau tidak
   if(i > 9)
      cout << "tidak ada " << nilaiCari << " dalam Array" << endl;
   else
   {
      if(X[i] == nilaiCari)
         cout << nilaiCari << " ditemukan dalam Array pada index ke-" <<i<< endl;
      else
         cout << "tidak ada " << nilaiCari << " dalam Array" << endl;
   }
}
```

```
/* SeqSearching_SudahUrut_Sentinel {cara 2}
diasumsikan Array X[0..10] sudah ada indeks 1..9 telah berisi data yang sudah
terurut, nilai yang dicari adalah nilaiCari dan hanya ada satu */

#include <iostream>
using namespace std;

int main()
{
   int X[11]={10,20,30,40,50,60,70,80,90,100};
   int i, nilaiCari;
   bool found;

   cout << "Nilai yang dicari = "; cin >> nilaiCari;

   X[10] = nilaiCari;
   found = false;
   i = 0;
   while((!found) & (X[i] <= nilaiCari))
   {
      if(X[i] == nilaiCari) found = true;
      else i = i + 1;
   }
   if(i == 10)
      cout << "tidak ada " << nilaiCari << " dalam Array" << endl;
    else {
      if (found)
         cout <<  nilaiCari << " ditemukan dalam Array pada index ke-" << i <<endl;
      else
         cout << "tidak ada " << nilaiCari << " dalam Array" << endl;
    }
}
```

3. BINARY SEARCH

Binary Search digunakan pada Array yang elemen datanya sudah terurut.

Ide :

1. Urutkan elemen-elemen Array
2. Cari elemen tengah denga mencari indeks tengahnya menggunakan rumus indektengah (indekskiri + indekskanan) div 2
3. Bandingkan nilai yang dicari dengan elemen tengah tersebut

Jika sama berarti ketemu, maka pencarian selesai
Jika tidak sama namun {harga tengah ikut dibuang}

Jika nilai yang dicari < elemen tengah, maka dilakukan pembagian Array sebagai berikut :

SubArray 1 dimulai dari indekskiri s/d indekskananbaru indekstengah-1

SubArray2 dimulai dari indekskiribaru indekstengah s/d indekskanan

Sehingga pencarian diteruskan ke SubArray1

Jika nilai yang dicari >elemen tengah, maka dilakukan pembagian Array sebagai berikut:

SubArray1 dimulai dari indekskiri s/d indkeskananbaru indekstengah

SubArray2 dimulai dari indekskiribaru indekstengah+1 s/d indekskanan

Sehingga pencarian diteruskan keSubArray2

4. Ulangi langkah 2 dan 3, sampai didapat kesimpulan bahwa nilai yang dicari "ditemukan" atau "tidak ditemukan".

Contoh Program 4.7. *Binary Search SudahUrut*

```
/* BinSearch_SudahUrut
diasumsikan Array X sudah ada dan berisi data yang sudah terurut,
nilai yang dicari adalah nilaiCari dan hanya ada satu */

#include <iostream>
using namespace std;

int main()
{
   int X[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
   int i, nilaiCari, j, k;
   bool found;

   cout << "nilai yang dicari = "; cin >> nilaiCari;

   found = false;
   i = 0;
   j = 10;
   while((!found) & (i <= j))
   {
      k = (i + j) / 2;
      if(nilaiCari == X[k])
         found = true;
      else
      {
         if(nilaiCari < X[k])
            j = k - 1;  // i tetap
         else
            i = k + 1;  // j tetap
      }
   }
   if(found)
      cout << nilaiCari << " ditemukan dalam Array pada index ke-" << k << endl;
   else
      cout << "tidak ada " << nilaiCari << " dalam Array" << endl;
}
```

D. LANGKAH PRAKTIKUM

1. Kerjakan contoh-contoh program di atas, simpan untuk tiap-tiap program.

E. TUGAS PRAKTIKUM

1. Modifikasi contoh program di atas dengan : jumlah data diinput, elemen datanya diinput dan ditampilkan (minimal satu contoh program).
2. Buatlah program untuk memodifikasi salah satu contoh program untuk menyisipkan elemen yang dicari ke dalam data array jika elemen yang dicari tersebut tidak ada (asumsi datanya sudah terurut).

MODUL V
PENGURUTAN DATA *(SORTING)*

A. TUJUAN

Mempraktekkan dan membandingkan metode sorting menggunakan tipe data array dan algoritma rekursif

B. ALOKASI WAKTU : 2 X Pertemuan = 120 menit

C. DASAR TEORI

1. PENGERTIAN

Sorting (pengurutan) didefinisikan sebagai suatu proses untuk menyusun kembali himpunan obyek menggunakan metode tertentu. Secara umum ada 2 jenis pengurutan data, yaitu :

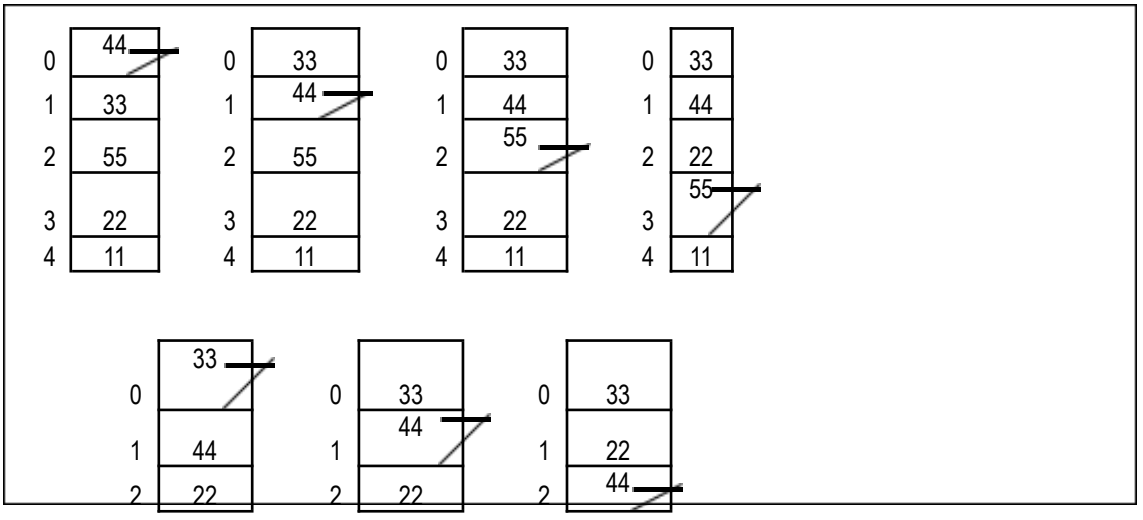
- *Pengurutan naik (ascending)*
  Dari data yang terkecil hingga yang paling besar
- *Pengurutan turun (descending)*
  Pengurutan data dari yang paling besar hingga yang paling kecil.

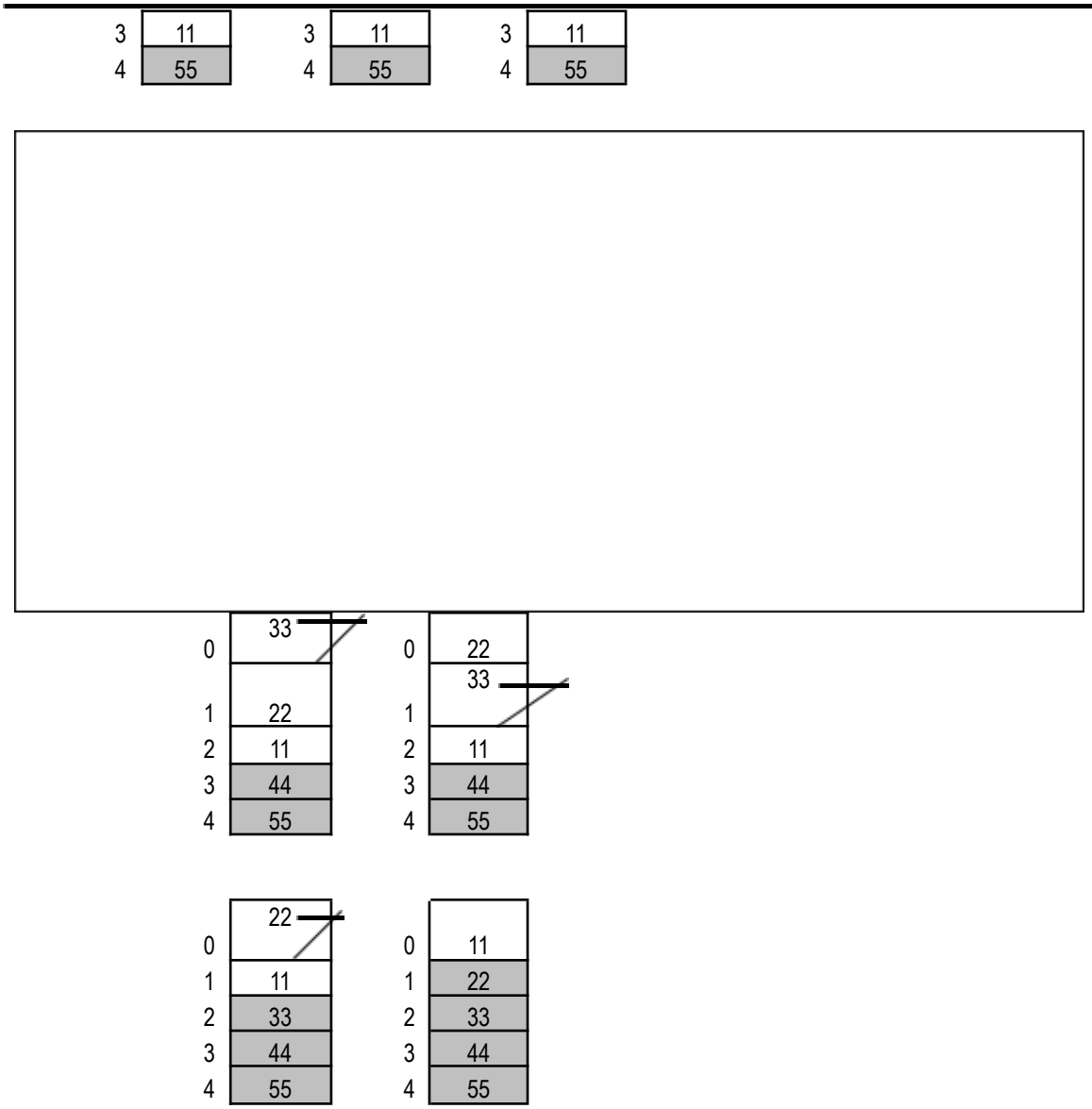
Tujuan pengurutan data adalah untuk lebih mempermudah proses pencarian data kelak di kemudian hari. Pada pratikum ini, akan diberikan beberapa contoh metode pengurutan data terutama pengurutan larik seperti pengurutan langsung *(straight method*). Metode pengurutan langsung ini dapat dikelompokkan menjadi 3 metode, yaitu : *penyisipan (insertion), seleksi (selection) dan penukaran (exchange) / metode gelembung (bubble sort).*

2. METODE GELEMBUNG (*BUBBLE SORT*)

Algoritma bubble adalah teknik pengurutan array yang sederhana yang normalnya merupakan metode pertama yang paling banyak dipelajari pemrogram. Karena kesederhanaannya, bubble sort tidak efisien dan menyita banyak waktu prosessor lebih banyak daripada teknik sorting yang lain. Namun, jika Anda mengurutkan array tidak lebih dari 30 atau kurang dari 30 elemen, penggunaan bubble sort masih sangat baik. Dengan menganggap bahwa Anda mengurutkan dari terkecil ke terbesar, bubble sort mengulang semua nilai dalam array, membandingkan dan memindahkan nilai terbesar array ke puncak array.

Iterasi bubble sort





Contoh ilustrasi di atas ada lima data yang diurutkan yaitu {44,33,55,22,11}. Iterasi pertama memindahkan nilai terbesar array ke puncak array. Iterasi kedua memindahkan nilai terbesar kedua ke posisi kedua dari puncak. Iterasi ketiga memindahkan nilai terbesar ketiga, dan seterusnya.

Contoh Program 5.1. *Bubble Sort*

Program berikut memakai bubble sort untuk mengurutkan data array berisi 30 nilai acak.

```
// Bubble Sort
#include <iostream>
#include <stdlib.h>
using namespace std;

void bubble_sort(int array[], int size)
{
   int temp, i, j;

   for(i=0; i<size-1; i++) {
      for(j=0; j<size-1-i; j++) {
         if(array[j] > array[j+1]) {
            temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
         }
      }
   }
}

int main()
{
   int data_size = 30;
   int values[data_size], i;
```

Perhatian : Fungsi bubble_sort di atas mengurutkan nilai dari terendah ke tertinggi. Untuk membalik urutan, cukup dengan mengubah perbandingan menjadi
`if (array[i]>array[j]).`

b. METODE PENYISIPAN LANGSUNG (*STRAIGHT INSERTION SORT*)

Dapat dibagi menjadi 2 bagian :
- Bagian sebelah kiri data sudah terurut (tujuan)
- Bagian sebelah kanan data belum terurut (sumber)

Ilustrasi :

I	j=i-1	Temp = A[i]	A [0]	A [1]	A [2]	A [3]	A [4]	A [5]
			4	7	9	5	8	6
1	0	7	4	**7**	9	5	8	6
2	1	9	4	7	**9**	5	8	6
3	2	5	4	7	**5**	9	8	6
	1	5	4	**5**	7	9	8	6
4	3	8	4	5	7	**8**	9	6
5	4	6	4	5	7	8	**6**	9
	3	6	4	5	7	**6**	8	9
	2	6	4	5	**6**	7	8	9

Langkah-langkah :

Langkah 1 : Baca array elemen yang akan diurutkan (n)
2 : Kerjakan langkah 3 sampai langkah 6 untuk i : 1 s/d n-1
3 : Tentukan elemen yang akan disisipkan (Temp = A [i] ; j = i-1;)
4 : Kerjakan langkah 5 selama temp <A [j] dan j >= 0;
5 : A [j+1]= A[j] ; j =j-1;
6 : Tempatkan elemen A [j+1] = Temp;
7 : Selesai

Contoh Program 5.2. *Straight Insertion Sort*

```
// Straight Insertion Sort
#include <iostream>
using namespace std;

void straight_inst_sort(int array[], int size)
{
   int i, j, k;
   int Temp;

   for(i=1; i<size; i++) {
      Temp = array[i];
      j = i - 1;
      while((Temp < array[j]) & (j >= 0))
      {
```

anjut

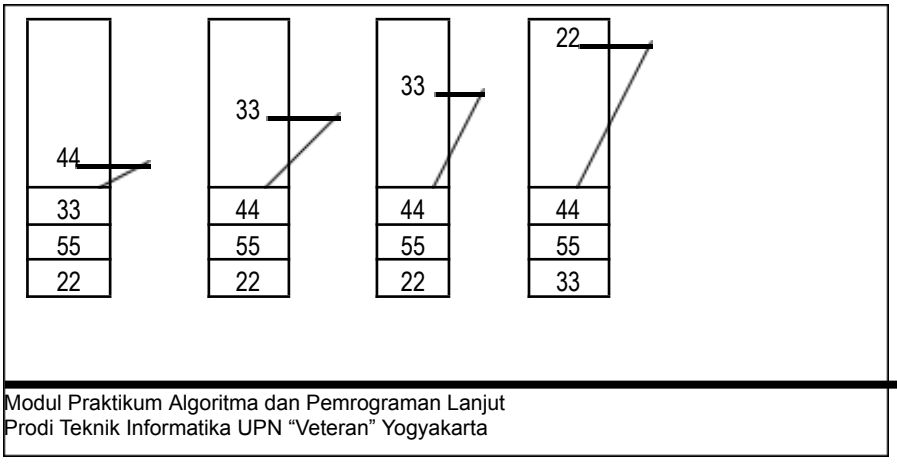
Lanjutan Contoh Program 5.2. *Straight Insertion Sort*

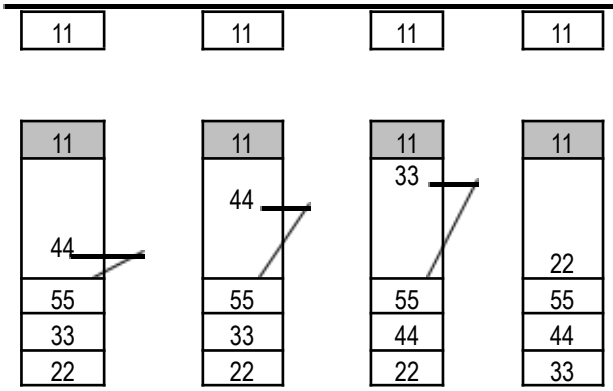
```
    // masukkan data sebanyak n
    for(k=0; k<data_size; k++)
    {
       cout << "values[" << k << "] = ";
       cin >> values[k];
    }
    // data yang belum diurutkan
    cout << "data yang belum diurutkan : " << endl;
    for(k=0; k<data_size; k++)
       cout << values[k] << " ";
    cout << endl;

    straight_inst_sort(values, data_size);
    // data  yang telah diurutkan
    cout << "data yang sudah diurutkan : " << endl;
    for(k=0; k<data_size; k++)
       cout << values[k] << " ";
    getchar();
}
```

c. METODE SELEKSI (*STRAIGHT SELECTION SORT*)

Seperti bubble sort, selection sort merupakan algoritma sorting yang sederhana. Seperti bubble sort, program Anda harus hanya memahami seleksi untuk mengurutkan array yang kecil (misalnya 30 elemen). Selection sort dimulai dengan menyelesaikan elemen array (misalnya elemen pertama). Kemudian sorting mencari keseluruhan array hingga menemukan nilai yang terkecil. Sorting menempatkan nilai terkecil pada elemen tersebut, memilih elemen kedua dan mencari elemen terkecil kedua. Berikut ini adalah illustrasi dua iterasi dari selection sort pada nilai-nilai array.

Ilustrasi mengurutkan nilai dengan selection sort





Program berikut, memakai selection sort untuk mengurutkan array yang berisi 30 nilai acak:

```
// Selection Sort
#include <iostream>
#include <stdlib.h>
using namespace std;

void selection_sort(int array[], int size)
{
   int temp, current, j;

   for(current=0; current<size; current++)
   {
      for(j=current+1; j<size; j++)
      {
         if(array[current] > array[j])
         {
            temp = array[current];
            array[current] = array[j];
            array[j] = temp;
         }
      }
   }
}

int main()
{
   int data_size = 30;
   int values[data_size], i;

   // data yang diurutkan belum diambil dari hasil random
   cout << "data yang belum urut : " << endl;

   for(i=0; i<data_size; i++)
   {
      values[i] = rand() % 100;
   cout << valuSortes[i] << " ";
   }
   cout << endl;

   selection_sort(values, data_size);

   // data yang sudah diurutkan
   cout << "data yang sudah diurutkan : " << endl;
   for(i=0; i<data_size; i++)
      cout << values[i] << " ";
   getchar();
}
```

Contoh Program 5.3. *Selection Sort*

Perhatian: Fungsi selection_sort mengurutkan nilai dari terkecil ke terbesar. Untuk membalik urutan, cukup dengan mengubah pembandingan menjadi `if (array[current] < array[j])`

d. METODE *SHELL SORT*

Shell sort dinamakan berdasarlan nama akhir penciptanya, Donald Shell. Teknik sorting ini bekrja dengan membandingkan elemen-elemen array yang dipisahkan dalam jarak tertentu (celah) sehingga elemen-elemen yang dibandingkan dengan celah sekatang terurutkan. Celah tersebut kemudian dibagi menjadi dua dan pemrosesan dilanjutkan. Pada saat celah akhirnya berupa 1 dan tak ada perubahan terjadi, array telah urut.

Ilustrasi Mengurutkan array dengan shell sort

Jarak	**A[1]**	**A[2]**	**A[3]**	**A[4]**	**A[5]**	**A[6]**	**A[7]**	**A[8]**	**A[9]**
Awal	**24**	46	11	26	**57**	38	27	20	17
Jarak = 9/2=4	**24**	**46**	11	26	**57**	**38**	27	20	17
	24	**38**	**11**	26	57	**46**	**27**	20	17
	24	38	**11**	**26**	57	46	**27**	**20**	17
	24	38	11	**20**	**57**	46	27	**26**	**17**
	24	38	11	20	**17**	46	27	26	**57**
	17	38	11	20	**24**	46	27	26	57
Jarak = 4/2=2	**17**	38	**11**	20	24	46	27	26	57
	11	**38**	**17**	**20**	24	46	27	26	57
	11	**20**	**17**	**38**	**24**	46	27	26	57
	11	20	**17**	**38**	**24**	**46**	27	26	57
	11	20	17	**38**	**24**	**46**	**27**	26	57
	11	20	17	38	**24**	**46**	**27**	**26**	57
	11	20	17	**38**	24	**26**	27	**46**	57
	11	**20**	17	**26**	24	38	27	46	57
	11	**20**	17	**26**	24	38	**27**	46	**57**
	11	20	17	26	24	38	**27**	46	**57**
Jarak = 2/2=1	**11**	**20**	17	38	24	26	27	46	57
	11	**20**	**17**	38	24	26	27	46	57
	11	**17**	**20**	38	24	26	27	46	57
	11	**17**	**20**	**38**	24	26	27	46	57
	11	17	**20**	**38**	**24**	26	27	46	57
	11	17	**20**	**24**	38	26	27	46	57
	11	17	**20**	**24**	**26**	**38**	27	46	57
	11	17	20	24	**26**	**38**	**27**	46	57

	11	17	20	24	**26**	**27**	**38**	46	57
	11	17	20	24	**26**	**27**	**38**	**46**	57
	11	17	20	24	26	27	**38**	**46**	**57**
Hasil	11	17	20	24	26	27	38	**46**	**57**

Program berikut, memakai shell sort untuk mengurutkan array yang diinput.

Contoh Program 5.4. *Shell Sort*

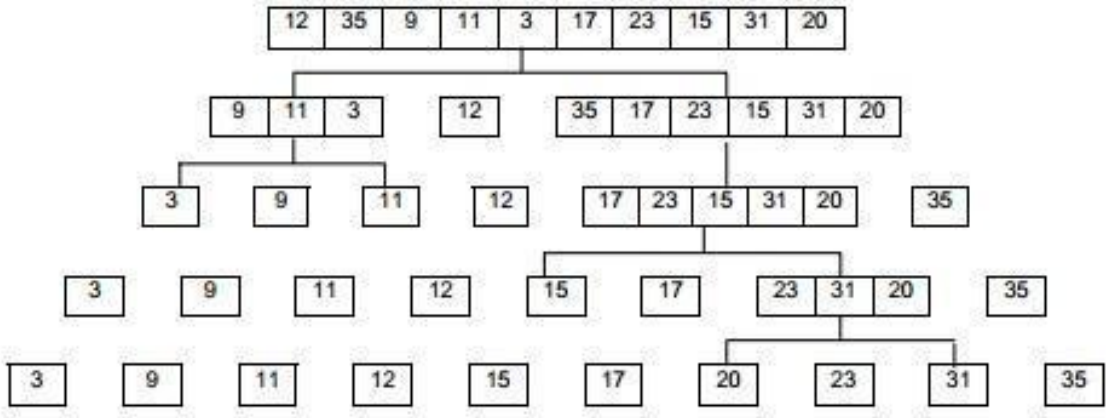
```
// Shell Sort
#include <iostream>
#include <stdio.h>
#define size 9
using namespace std;

int shell_sort(int []);

int main()
{
    int arr[size], i;
    cout << "Enter the elements to be sorted : " << endl;

    for(i = 0; i < size; i++)
    {
        cin>> arr[i];
    }

    shell_sort(arr);

    cout << "The array after sorting is : " << endl;
    for(i = 0; i < size; i++)
        cout << arr[i] << endl;
}

// fungsi shell sort
int shell_sort(int array[])
{
   int i = 0, j = 0, k = 0, mid = 0;
   for(k = size/2; k > 0; k /= 2)
   {
      for(j = k; j < size; j++)
      {
         for(i = j-k; i >= 0; i -= k)
         {
            if(array[i + k] >= array[i])
               break;
            else
            {
               mid = array[i];
               array[i] = array[i + k];
               array[i + k] = mid;
            }
         }
      }
   }
   return 0;
}
```

Perhatian: Fungsi shell_sort mengurutkan nilai terkecil ke terbesar. Untuk membalik urutan, cukup dengan mengubah pembandingan menjadi **`if(array[i] < array[i+gap])`**

e. METODE *QUICK SORT*

Saat jumlah elemen pada array meningkat, quick sort merupakan salah satu teknik sorting yang tercepat yang dapat dipakai program Anda. Quick sort memperlakukan array sebagai senarai nilai. Saat sorting dimulai, ia memilih nilai di tengah-tengah senarai sebagaii pemisah senarai. Kemudian sorting ini membagi senarai menjadi dua buah senarai, satu dengan nilai yang kurang dari pemisah senarai dan senarai kedua yang nilainya lebih besar daripada atau sama dengan pemisah senarai. Kemudian sorting ini secara rekursif memanggil dirinya sendiri dengan kedua senarai tersebut.
Setiapkali sorting dipanggil, ia membagi lebih jauh elemen-elemen ke dalam senarai yang lebih kecil. Gambar berikut mengilustrasikan cara quick sort dapat mengurutkan array dengan 9 nilai.

Ilustrasi Sorting nilai dengan quick sort.



Program berikut, QUICK.C, memakai quick sort untuk mengurutkan array yang berisi 100 nilai acak :

Contoh Program 5.5. *Quick Sort*

```
// Quick Sort
#include <iostream>
#include <stdlib.h>
#define data_size 100
using namespace std;

void quick_sort(int array[], int first, int last)
{
   int temp, low, high, list_separator;

   low = first;
   high = last;
   list_separator = array[(first + last) / 2];

   do {
      while(array[low] < list_separator)
         low++;
      while(array[high] > list_separator)
         high--;

      if(low <= high)
      {
         temp = array[low];
         array[low++] = array[high];
         array[high--] = temp;
```

Lanjutan Contoh Program 5.5. *Quick Sort*

```
int main()
{
   int first = 0;
   int last = data_size - 1;
   int values[data_size], i;

   // data yang belum diurutkan diambil dari hasil random
   cout << "data yang belum urut : " << endl;
   for(i = 0; i < data_size; i++)
   {
      values[i] = rand() % 100;
      cout <<values[i] << " ";
   }
   cout << endl;

   quick_sort(values, first, last);

   // data yang sudah diurutkan
   cout << "data yang sudah diurutkan : " << endl;
   for(i = 0; i < data_size; i++)
      cout << values[i] << " ";
   getchar();
}
```

Perhatian: Fungsi quick_sort mengurutkan nilai dari terkecil ke terbesar. Untuk membalik urutan, cukup dengan mengubah pembandingan pada dua pernyataan while menjadi :

```
while (array[low]>list_separator) low++;
while (array[high]<list_separator) high--;
```

D. LANGKAH PRAKTIKUM

1. Jalankan contoh program 6.1. Bubble Sort. Program tersebut sorting data dari terkecil ke terbesar. Ubahlah menjadi sebuah program sorting data dari terbesar ke terkecil (simpan dengan nama lain).
2. Jalankan contoh program 6.2 Straight Insertion Sort. Tambahkan perintah menampilkan perubahan data tiap iterasi di fungsi straight_inst_sort. Tambahan perintahnya sbb :

```
for(k=0;k<size;k++)
  cout << array[k]<< "  ";
cout << endl;
```

sehingga contoh tampilannya sbb : (program simpan dengan nama lain)

```
banyaknya data = 6
values[0] = 4
values[1] = 7
values[2] = 9
values[3] = 5
values[4] = 8
values[5] = 6
data yang belum diurutkan :
4  7  9  5  8  6
perubahan data dgn straight_inst_sort :
4  7  5  9  8  6
4  5  7  9  8  6
4  5  7  8  9  6
4  5  7  8  6  9
4  5  7  6  8  9
4  5  6  7  8  9
data yang sudah diurutkan :
4  5  6  7  8  9  _
```

3. Jalankan contoh program lainnya.

E. TUGAS PRAKTIKUM

1. Buatlah program menu untuk memilih metode sorting yang digunakan. Contoh tampilan output program : (mengurutkan 50 data acak/random)

PROGRAM SORTING

50 Data Acak Yang Akan Diurutkan :

25 30 12 11 35 60 97 95 45 39 30 12 24 45 50 32 96 95 25 11 25 30 12 11 35 60 97 95 45 39 30 12 24 45 50 32 96 95 25 11 25 30 12 11 35 60 97 95 45 39

Menu Pilihan :

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Shell Sort
5. Quick Sort

Pilihan : 1

Bubble Sort

50 Data Acak Yang Telah Diurutkan :

11 11 11 11 11 12 12 12 12 12 24 24 25 25 25 25 25 30 30 30 30 30 32 32 35 35 35 39 39 39 45 45 45 45 45 50 50 60 60 60 95 95 95 95 95 96 96 97 97 97

Lagi (y/t) ? __

MODUL VI
POINTER

A. TUJUAN:

Memperkenalkan dan mempelajari konsep Pointer yang akan digunakan pada variabel dinamis untuk memakai memori bebas selama eksekusi program

B. ALOKASI WAKTU : 2 X Pertemuan = 120 menit

C. DASAR TEORI

1. PENGERTIAN

Pointer banyak dilibatkan dalam pemrograman C++, misalnya untuk melewatkan string dari suatu fungsi ke fungsi yang lain. Pointer cara yang digunakan untuk mengubah nilai suatu variabel yang dilewatkan ke dalam suatu fungsi. Penerapan pointer yang paling umum yaitu menciptakan variable dinamis, yang memungkinkan untuk memakai memori bebas (memori yang belum dipakai) selama eksekusi program.

Variabel pointer sering disebut sebagai variabel yang menunjuk obyek lain, karena variabel pointer atau pointer adalah variabel yang berisi alamat dari suatu obyek lain, yaitu obyek yang ditunjuk oleh pointer. Suatu variabel pointer dideklarasikan dengan bentuk:

```
tipedata *nama_variabel;
```

Untuk mengatur pointer agar menunjuk ke variabel lain, mula-mula pointer harus diisi dengan alamat dari variabel yang akan ditunjuk. Operator & digunakan untuk menyatakan alamat variabel yang akan ditunjuk. Contoh : &x

Jika suatu varibel sudah ditunjuk oleh pointer, variabel yang ditunjuk oleh pointer dapat diakses melalui variabel itu sendiri (pengaksesan langsung) atau melalui pointer (pengaksesan tidak langsung). Operator indirection, berupa simbol * digunakan untuk pengaksesan tidak langsung.Contoh :`*px`

Bentuk Umum :

```
tipedata  *nama_pointer;
```

Contoh:

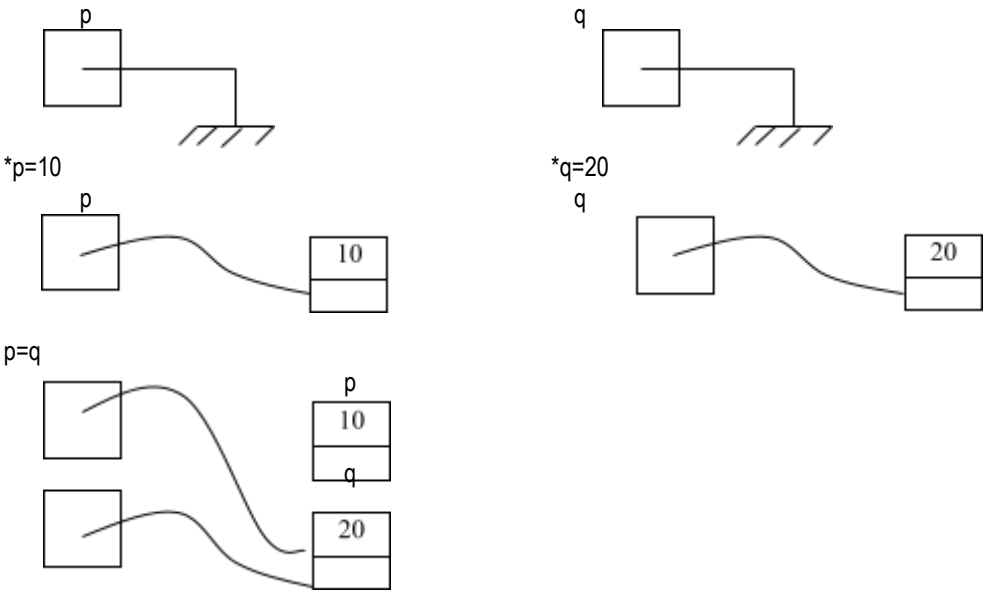
```
int  *p;
float  *nilai;
char  *s;
```

Untuk mendeklarasikan sebuah pointer kosong pada memory digunakan perintah :

```
*malloc(size_t size)
```

Contoh:

```
p=(int *)malloc(sizeof(int));
q=(int *)malloc(sizeof(int));
```



Berikut contoh Implementasi struktur data menggunakan tipe pointer.

Contoh Program 6.1. Contoh Implementasi struktur data menggunakan tipe pointer

```
#include <iostream>
#include <malloc.h>
#include <stdlib.h>
using namespace std;

int main()
{
   int *p, *q, *r;
   int n = 10;
   p = (int *)malloc(sizeof(int));
   q = (int *)malloc(sizeof(int));

   p = &n;
   *q = 120;
   r = p;

   cout << "Isi info pointer : \n";
   cout << "*p = " << *p << endl;
   cout << "*q = " << *q << endl;
   cout << "*r = " << *r << endl;
   cout << "\nAlamat register pointer : \n";
   cout << "p = " << p << endl;
   cout << "q = " << q << endl;
   cout << "r = " << r << endl;

   n = *q;
   p = q;

   cout << "\nKondisi akhir isi info pointer : \n";
   cout << " n = " << n << endl;
   cout << "*p = " << *p << endl;
   cout << "*q = " << *q << endl;
   cout << "*r = " << *r << endl;
}
```

Contoh Program 6.2. Contoh pemakaian pointer yang salah

```
// Contoh pemakaian pointer yang salah
#include <iostream>
using namespace std;

int main()
{
   float *pu;
   float nu;
   int u = 1234;
   pu = &u;  // pernyataan ini salah
             // sebab pu adalah pointer float padahal u bertipe int
   nu = *pu;
   cout << "u = " << u << endl;
   cout << "nu = " << nu << endl;
}
```

Contoh Program 6.3. Contoh pointer yang menunjuk array

```
// Contoh pointer yang menunjuk array
#include <iostream>
using namespace std;

int main()
{
   static int tgl_lahir[] = {27, 8, 1977};
   int *ptgl;
   ptgl = tgl_lahir;  // ptgl berisi alamat array

   cout << "nilai yang ditunjuk oleh ptgl = " << *ptgl << endl;
cout << "nilai dari tgl_lahir[0] = " << tgl_lahir[0] << endl;
}
```

2. POINTER DAN ARRAY

Hubungan antara array dan pointer sangat erat, karena array secara internal akan diterjemahkan ke dalam pointer.

```
#include <iostream>
using namespace std;

int main()
{
   int x, y;  // x & y bertipe int
   int *px;   // px pointer yang menunjuk objek bertipe int

   x = 87;
   px = &x;  // px berisi alamat dari x
   y = *px;  // y berisi nilai yang ditunjuk px

   cout << "alamat x = " <<&x << endl;
   cout << "isi px x = " << px << endl;
   cout << "nilai yang ditunjuk oleh px = " << px << endl;
   cout << "nilai y = " << y << endl;
}
```

Contoh Program 6.5. Contoh program pointer dan array

```
// Contoh pengaksesan isi array dengan pointer
#include <iostream>
using namespace std;

int main()
{
   static int tgl_lahir[] = {27, 8, 1977};
   int i;
   int *ptgl;

   ptgl = tgl_lahir;  // ptgl berisi alamat array

   for(i=0; i<3; i++)
      cout << *(ptgl + i) <<" ";
   cout << endl;
}
```

3. POINTER DAN STRING

Contoh Program 6.6. Program pointer yang menunjuk string

```
#include <iostream>
using namespace std;

int main()
{
   char *pkota = "YOGYAKARTA";  /* pkota menunjuk konstanta string
                                                          "YOGYAKARTA"
                              */
   cout << pkota << endl;
}
```

Contoh Program 6.7. Program menukar isi dua string tanpa memakai pointer

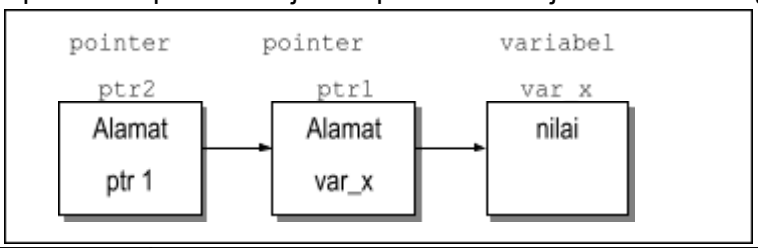
```
// menukar isi dua string tanpa memakai pointer
#include <iostream>
#include <string.h>
#define panjang 20
using namespace std;

char nama1[panjang] = "hanif";
char nama2[panjang] = "fathimah";

int main()
{
   char namax[panjang];
   cout << "semula : " << endl;
   cout << "nama1 -> " << nama1 << endl;
   cout << "nama2 -> " << nama2 << endl;

   // penukaran string
   strcpy(namax, nama1);
   strcpy(nama1, nama2);
   strcpy(nama2, namax);

   cout << "kini : " << endl;
   cout << "nama1 -> " << nama1 << endl;
   cout << "nama2 -> " << nama2 << endl;
}
```

Contoh Program 6.8. Program menukar isi dua string yang ditunjuk pointer

```
// menukar isi dua string yang ditunjuk pointer
#include <iostream>
using namespace std;

char *nama1 = "hanif";
char *nama2 = "fathimah";

int main()
{
   char *namax;
   cout << "semula : " << endl;
   cout << "nama1 -> " << nama1 << endl;
   cout << "nama2 -> " << nama2 << endl;

   // penukaran string yang ditunjuk oleh pointer nama1 dan nama2
   namax = nama1;
   nama1 = nama2;
   nama2 = namax;

   cout << "kini : " << endl;
   cout << "nama1 -> " << nama1 << endl;
   cout << "nama2 -> " << nama2 << endl;
}
```

4. POINTER MENUNJUK POINTER

Suatu pointer dapat menunjuk ke pointer lain dijelaskan dalam gambar berikut:



Perintah	Keterangan
`int var_x;`	`var_x`adalah variabel bertipe integer
`int *ptr1;`	`*ptr1` adalah variabel pointer yang menunjuk ke data bertipe int
`int **ptr2;`	`*ptr2`adalah variabel pointer yang menunjuk ke pointer int
`ptr1=&var_x;`	`ptr1` menunjuk ke variabel `var_x`
`ptr2=&ptr1;`	`ptr2` menunjuk ke `ptr1`

Contoh Program 6.9. Program pointer menunjuk pointer

```
// pointer menunjuk pointer
#include <iostream>
using namespace std;

int main()
{
   int var_x = 273;  // variabel int
   int *ptr1;             // pointer int
   int **ptr2;         // pointer menunjuk pointer int
   ptr1 = &var_x;     // ptr1 berisi alamat dari var_x
   ptr2 = &ptr1;        // ptr2 berisi alamat dari ptr1

   // mengakses nilai var_x melalui ptr1
  cout << "nilai var_x = " << *ptr1 << endl;

  // mengakses nilai var_x melalui ptr2
  cout << "nilai var_x = " << **ptr2 << endl;
}
```

5. POINTER DAN FUNGSI

a. Pointer sebagai parameter fungsi

Penerapan pointer sebagai parameter diinginkan agar nilai suatu variabel internal dapat diubah menjadi fungsi yang dipanggil.

Contoh panggilan	keterangan

`void naikkan nilai (int*x,int*y)`
`{`
`*x = *x+2;`
`*x = *x+2;`
`}`
	Variabel yang berkaitan dengan parameter aktual dapat diubah nilainya, masing-masing dianaikan sebesar 2

`naikkan_nilai(&a,&b);`	Operator & berarti menyatakan alamat variabel, sebab variabel fungsi dalam pendefinisian berupa pointer.

Contoh Program 6.10. Program pointer sebagai parameter fungsi

```
// contoh pointer sebagai parameter fungsi
#include <iostream>
using namespace std;

void naikkan_nilai (int *x, int *y);

int main()
{
   int a=3;
   int b=7;

   cout << "semula : a = "<< a<< " b = " << b << endl;

   naikkan_nilai(&a,&b);
   cout << "kini : a = "<< a << " b = " << b << endl;
}

void naikkan_nilai(int *x, int *y)
{
   *x = *x + 2;
   *y = *y + 2;
}
```

b. Pointer sebagai keluaran (*return value)* fungsi

Suatu fungsi dapat dibuat agar keluarannya berupa pointer

Contoh	keterangan
`char *nama_hari()`	Menyatakan bahwa fungsi `nama_hari()` berupa pointer yang menunjuk ke obyek bertipe char (atau string). Dalam fungsi nama_hari, array dideklarasikan dan diinisialisasikan agar menunjuk sejumlah string yang menyatakan nama_hari.
`return (n<1 || n>7) ? hari[0] :hari[n] );`	

Contoh Program 6.11. Program fungsi dengan keluaran berupa pointer

```
// contoh fungsi dengan keluaran berupa pointer
#include <iostream>
using namespace std;

char *nama_hari(int n);

int main()
{
   int hr;
   cout << "hari(1..7) : ";
   cin >> hr;
   cout << nama_hari(hr) << endl;
}

char *nama_hari(int n)
{
   static char *hari[] = {
      "kode hari salah", "senin","selasa", "rabu",
      "kamis","jumat","sabtu","minggu"
   };
   return (n < 1 || n > 7) ? hari[0] : hari[n];
}
```

D. LANGKAH PRAKTIKUM

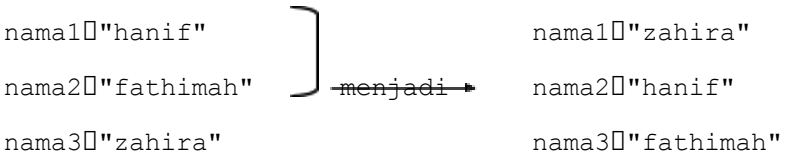
1. Jalankan contoh-contoh program di atas.

E. TUGAS PRAKTIKUM

1. Buatlah suatu program dengan menggunakan pointer untuk menampilkan tulisan atau output :

```
Pemrograman Dasar Turbo C

Dasar Turbo C

Turbo C
```

2. Buatlah program untuk menukar isi tiga string tanpa pointer dan yang ditunjuk pointer (modifikasi contoh program 6.7 dan 6.8). Misal :



MODUL VII
OPERASI FILE

A. TUJUAN

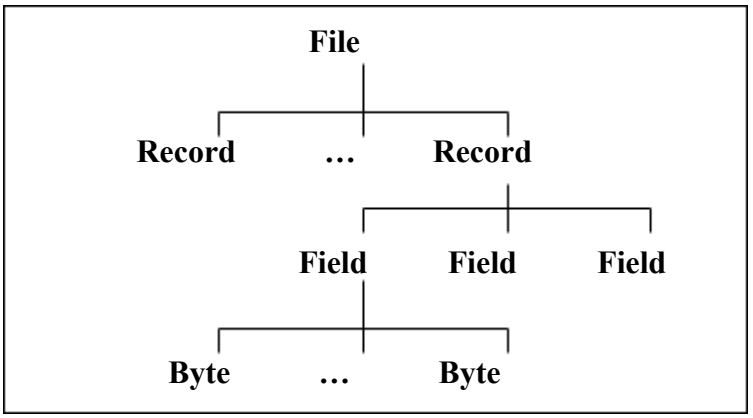
Mengetahui teknik pemrograman yang melibatkan data yang disimpan dalam bentuk file.

B. ALOKASI WAKTU : 2 X Pertemuan = 120 menit

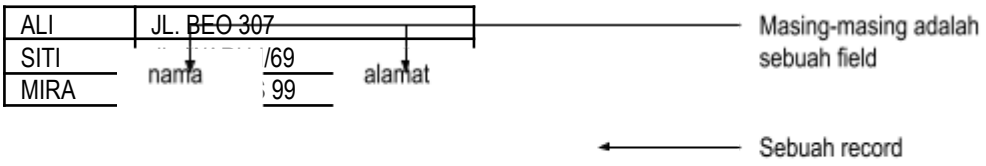
C. DASAR TEORI

1. STRUKTUR FILE

Kebanyakan program melibatkan media disk sebagai tempat untuk membaca atau merekam data. Data sendiri disimpan dalam disk dalam bentuk suatu kesatuan yang disebut **file**. Suatu file merupakan organisasi dari sejumlah **record** dapat terdiri dari satu atau beberapa **field** dan setiap field terdiri dari satu ataua beberapa **byte**. Adapun byte merupakan susunan dari 8 bit.



Struktur data dari file



Gambaran record dan field

2. TAHAPAN OPERASI FILE

a. Membuka / Mengaktifkan File

Bentuk deklarasi pengaktifan file adalah sebagai berikut :

```
FILE fopen(char *namafile, char *mode);
```

Keterangan mode :

- **r** : Read only
- **w** : Menyatakan file baru diciptakan. Operasi yang akan dilakukan adalah operasii perekaman data. Jika file tersebut sudah ada, isi yang lama akan dihapus.
- **a** : Membuka file yang ada pada disk dan operasi yang akan dilakukan adalah operasii penambahan data pada file. Jika file belum ada, secara otomatis file akan dibuat .
- **r+** : Membuka file yang sudah ada, operasi yang dilakukan berupa pembacaan dan penulisan.
- **w+** : Membuka file untuk pembacaan/penulisan. Jika file sudah ada, isinya akan dihapus.
- **a+** : Membuka file, operasi yang dilakukan berupa perekaman dan pembacaan. Jika file sudah ada, isinya tak akan terhapus.

b. Menutup File

Untuk menutup file, deklarasi yang digunakan :

```
int fclose(FILE *pf);
```

3. OPERASI PENYIMPANAN DAN PEMBACAAN FILE PER KARAKTER

Sebuah karakter dapat disimapan dalam file dengan menggunakan deklarasi fungsi :

```
int fputc(int kar, FILE *ptr_file );
```

Contoh program 7.1.

```
// menciptakan & mengisi file dengan data karakter dari keyboard
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#define CTRL_Z 26
using namespace std;

int main()
{
   FILE *pf;                                    // pointer ke FILE
   char kar;

   if((pf = fopen("COBA.TXT", "w")) == NULL)    // ciptakan file
   {
      cout << "File tak dapat diciptakan! \r\n"; // tulis ke layar
      exit(1);  // selesai
   }

   while((kar = getchar()) != CTRL_Z)           // baca kar dari keyboard
      fputc(kar, pf);                           // tulis ke file
   fclose(pf);                                  // tutup file
}
```

Contoh program 7.2.

```
// membaca isi file per karakter
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
using namespace std;

int main()
{
   FILE *pf;                                  // pointer ke file
   char kar;

   if((pf = fopen("COBA.TXT", "r")) == NULL)  // buka file
   {
cout<<" File tak dapat dibuka !!\r\n";  // tulis ke layar
      exit(1);                                // selesai
   }
   while((kar = getc(pf)) != EOF)             // baca kar dari file
cout<<kar;                                 // tampilkan ke layar
   fclose(pf);                                // tutup file
   getchar();
}
```

4. FILE BINER DAN FILE TEKS

File biner adalah file yang pola penyimpanan di dalam disk adalah dalam bentuk biner, yaitu seperti bentuk dalam memori (RAM) komputer. Sedangkan file teks merupakan file yang pola penyimpanan datanya dalam bentuk karakter.
Keterangan mode :

rt : mode file adalah teks dan file hendak dibaca.
rt+ : mode file adalah teks dan file bisa dibaca atau ditulisi (= **r+t**).
rb : mode file adalah biner dan file hendak dibaca.

5. OPERASI PENYIMPANAN DAN PEMBACAAN FILE PER INT

Untuk menyimpan atau membaca data file bertipr **int**, digunakan **getw( )** untuk membaca filedan **putw ( )** untuk menyimpan file. Bentuk deklarasinya :

```
int getw(FILE *ptr_file);

int putw(FILE *ptr_file);
```

Contoh program 7.3.

```
// menyimpan data dengan putw()

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

int main()
{
   FILE *pf;  // ptr ke FILE
   int nilai;
   char jawab;

   if((pf = fopen("BILANGAN.DAT", "wb")) == NULL)  // ciptakan file biner
   {
      cout << "File gagal diciptain !!\n";
      exit(1);
   }
   cout << "Program untuk menyimpan data integer ke file.";
```

Lanjutan Contoh program 7.3.

```
   do {
      cout << "\r\n  Bilangan yang akan disimpan : ";
      cin >> nilai;  // baca nilai dari keyboard
      cin.ignore();
      putw(nilai, pf);  // tulis bilangan ke file
      cout<<"  Memasukkan data lagi (Y/T)? : ";
      jawab = getchar();  // baca jawaban dari keyboard
   } while(jawab == 'y' || jawab == 'Y');

   fclose(pf);  // tutup file
   cout << "\r\nOke. Data sudah disimpan pada file.\r\n";
   getchar();
}
```

Contoh program 7.4.

```
// membaca isi file dengan getw()

#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
   FILE *pf;  // ptr ke file
   int nilai;
   int nomor = 0;

   if((pf=fopen("BILANGAN.DAT","rb"))==NULL) //buka file biner utk dibaca
   {
      cout<<"  File gagal dibuka.\r\n";
      exit(1);
   }
   cout<<"\n Isi file BILANGAN.DAT : \r\n";

   while(1)
   {
      nilai = getw(pf);  // baca sebuah int dr file
      if(feof(pf) != NULL)
         break;  // jika akhir file,keluar loop
      cout << " " << ++nomor << ". " << nilai <<"\r\n";
   }
   fclose(pf);  // tutup file
   getchar();
}
```

6. OPERASI PENYIMPANAN DAN PEMBACAAN FILE PER BLOK

Untuk menyimpan atau membaca data file dalam bentuk kesatuan blok (sejumlah byte), misalnya untuk tipe float atau struct (struktur). Digunakan fungsi fread( ) dan fwrite( ),dengan deklarasi :

```
int fread(void *buffer, int n, FILE *ptr_file);

int fwrite(void *buffer, int jum_byte, int n, FILE *ptr_file);
```

Contoh program 7.5.

```
// menyimpan data bertipe struktur, dengan fwrite()

#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
   FILE *f_struktur;
   char jawab;

   struct data_pustaka
   {
      char judul[26];
      char pengarang[20];
      int jumlah;
   } buku;  // variabel buku bertipe struktur

   if((f_struktur = fopen("DAFBUKU.DAT", "wb")) == NULL)
   {
      cout << "File tak dapat diciptakan !!\r\n";
      exit(1);
   }

   do {
      cout << "  Judul buku         :  "; cin >> buku.judul;
      cout << "  Nama pengarang     :  "; cin >> buku.pengarang;
      cout << "  Jumlah buku        :  "; cin >> buku.jumlah;
      cin.ignore();
      fflush(stdin);  // hapus isi penampung keyboard
      fwrite(&buku, sizeof(buku), 1, f_struktur);  // rekam sebuah data
struktur
      cout << "\r\nMau merekam data lagi (Y/T) : ";
 jawab = getchar();
   }
   while(jawab =='Y' || jawab =='y');
   fclose(f_struktur);  // tutup file
}
```

Contoh program 7.6.

```
// membaca sebuah struktur, dengan fread()

#include <iostream>
#include <iomanip>
#include <stdlib.h>
#define JUM_BLOK 1  // sebuah record
using namespace std;

int main()
{
   FILE *f_struktur;

   struct
   {
      char judul[26];
      char pengarang[20];
      int jumlah;
   } buku;  // variabel struktur

   if((f_struktur = fopen("DAFBUKU.DAT", "rb")) == NULL)
   {
      cout << "File tak dapat dibuka\r\n";
      exit(1);
   }
```

Lanjutan Contoh program 7.6.

```
    cout << "\nDaftar buku :\n";
    cout << "\nJudul" << setw(26) << "Pengarang" << setw(20) << "Jumlah\n";

    while(fread(&buku, sizeof(buku), JUM_BLOK, f_struktur) == JUM_BLOK)
       cout << buku.judul << setw(26) << buku.pengarang << setw(20) <<
                    buku.jumlah<<endl;
    fclose(f_struktur);
    getchar();
}
```

7. MENYIMPAN DAN MEMBACA DATA STRING DARI FILE

Fungsi yang digunakan untuk membaca data string pada file yaitu **`fgets( )`** untuk menyimpan string **`str`** ke dalam file. Dan **`fputs( )`** untuk membaca string dari file sampai ditemukannaya karakter baris-baru '**`\n`**' atau setelah (n-1) karakter, dangan **`n`** adalah panjang maksimal string yang dibaca per wktu-baca.
Bentuk deklarasinya :

```
int fputs(char *str, FILE *ptr_file);

char *fgets(char *str, int n, FILE *ptr_file);
```

Contoh program 7.7.

```
// membaca isi file teks dengan fgets()

#include <iostream>
#include <stdlib.h>
#define PANJANG 256
using namespace std;

int main()
{
   FILE *f_teks;
   char string[PANJANG];
   char namafile[65];

   cout << "\n PROGRAM UNTUK MELIHAT ISI FILE TEKS\r\n";
   cout << "\n Nama file : "; cin >> namafile;

   if((f_teks = fopen(namafile, "rt")) == NULL)
   {
      cout << "File tak dapat dibuka\r\n";
      exit(1);
   }

  // baca string dari file selama masih ada

   while((fgets(string, PANJANG, f_teks)) != NULL)
      cout << "\n Isi file :\n";

   cout << "  " << string << "\r";  //tampilakan string ke layar

   fclose(f_teks);
   getchar();
}
```

Contoh program 7.8.

```
// menyalin isi file teks

#include <iostream>
#include <stdlib.h>
#include <ctype.h>
#define PANJANG 256
using namespace std;

void toUp(char *p);  // fungsi untuk melakukan uppercase

int main()
{
   FILE *pf_input, *pf_output;
   char string[PANJANG];
   char namafile_inp[65], namafile_out[65];

   cout << "\n PROGRAM UNTUK MENYALIN ISI FILE TEKS\r\n";
   cout << "\n HURUF KECIL AKAN DIGANTI HURUF KAPITAL\r\n";
   cout << "  Nama file input : "; cin>>namafile_inp;
   cout << "  Nama file output: "; cin>>namafile_out;

   if((pf_input = fopen(namafile_inp, "rt")) == NULL)  // buka file input
   {
      cout << "File input tak dapat dibuka !!\r\n";
      exit(1);
   }

   if((pf_output = fopen(namafile_out, "wt")) == NULL)  // buka file output
   {
      cout << "File output tak dapat dibuka !!\r\n";
      exit(1);
   }

   // baca string dari file input & konversikan kehuruf kapital
   // kemudian rekam ke file output
   while((fgets(string, PANJANG, pf_input)) != NULL)
   {
      toUp(string);
      fputs(string, pf_output);
   }
   fcloseall();  // tutup file input dan output
}

void toUp(char *p)
{
   while(*p)
   {
      *p = toupper(*p);
      p++;
   }
}
```

8. MENYIMPAN DAN MEMBACA DATA FILE YANG DIFORMAT

Bila dikehendaki, data bilangan dapat disimpan ke dalam file dalam keadaan diformat. Missal bilangan bulat (bertipe int) disimpan dengan panjang 5 digit, dan dibuat rata kanan, berapapun nilainya. Fungsi yang digunakan yaitu **`fprint( )`** dan **`fscan( )`**, dengan bentuk :

```
fprint(ptr_file, "string kontrol", daftar argumen);

fscan(ptr_file, "string kontrol", daftar argumen);
```

Contoh program 7.9.

```
// menyimpan data yang diformat ke file, dengan fprint()
#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
   FILE *pformat;
   char jawab;

   struct
   {
      int x, y;
   } koordinat;
   if((pformat = fopen("KOORDINAT.DAT","w"))==NULL) //buka&ciptakan file
   {
      cout << "File tak dapat dibuat !!";
      exit(1);
   }
   do {
      cout << "\n Masukkan data koordinat (bilangan int)\r\n";
      cout << "  Format : Posisi X Posisi Y\r\n";
      cout << "  Contoh : 20 30 [ENTER]\r\n\n";
      cin >> koordinat.x >> koordinat.y; cin.ignore();
      fprintf(pformat, "%5d%5d\n", koordinat.x, koordinat.y); /* rekam ke

                                                              file */
      cout << "\r\n Menyimpan data lagi (Y/T) : ";
      jawab = getchar();  // baca tombol
   }
   while(jawab == 'Y'||jawab =='y');
   fclose(pformat);
}
```

Contoh program 7.10.

```
// membaca data yang diformat dalam file, dengan fscan()
#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
   struct
   {
      int x, y;
   } koordinat;

   FILE *pkoord;
   char kar;  // menyimpan \n dari file
   int kode;  // memegang keluaran fungsi fscanf()

   if((pkoord = fopen("KOORDINAT.DAT", "rt")) == NULL)
   {
      cout << "File tak dapat dibuka !!";
      exit(1);
   }

   while((kode = fscanf(pkoord,"%5d%5d%c", &koordinat.x, &koordinat.y, &kar))!=EOF)
   {
      if(kode != 3)
      {
         cout << "Program dihentikan.\r\n";
         cout << "Sebab format data dalam file salah !\r\n";
   exit(1);
      }
      cout << koordinat.x << " " << koordinat.y << "\r\n";
   }
   fclose(pkoord);
   getchar();
}
```

9. PENGAKSESAN FILE BINER SECARA ACAK

Untuk keperluan pengaksesan secara random (acak), fungsi yang digunakan adalah **`fseek( )`**. Bentuk deklarasinya :

```
int fseek(FILE *ptr_file, long int offset, int posisi);
```

Konstanta untuk menentukan posisi pada pengaksesan file secara acak :

Konstanta	Nilai	Lokasi file
SEEK_SET	0	Awal file
SEEK_CUR	1	Posisi penunjuk file sedang berada
SEEK_END	2	Akhir file

Contoh program 7.11.

```
// melihat isi file secara random\

#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
   struct
   {
      char judul[26];
      char pengarang[20];
      int jumlah;
   } buku;  // variabel buku bertipe struktur

   FILE *pf;  // pointer file
   char ulang;
   int nmr_record, hasil_baca;
   long int ofset_byte;

   if((pf = fopen("DAFBUKU.DAT", "rb")) == NULL)  /* buka file yg berisi
   {                                                 daftar buku */
      cout << "File tak dapat dibuka !\r\n";
      exit(1);
   }
   ulang = 'Y';  // baca record secara random

   while(ulang == 'y' ||ulang == 'Y')
   {
      cout << "\r\n Nomor record dari data yang ingin diketahui : ";
      cin >> nmr_record; cin.ignore();

      ofset_byte = (nmr_record - 1) * sizeof(buku);
      fseek(pf, ofset_byte, SEEK_SET);
      hasil_baca = fread(&buku, sizeof(buku), 1,pf);

      if(hasil_baca == 0)
      {
         cout << "  Nomor record tak absah !!\r\n";
         continue;  // kembali ke while
      }
      cout << "  Judul buku  : " << buku.judul<<"\r\n";
      cout << "  Pengarang   : " << buku.pengarang<<"\r\n";
      cout << "  Jumlah buku : " << buku.jumlah<<"\r\n";
      cout << "\r\n  Mau mencari informasi lagi (Y/T) : ";
      ulang = getchar();
      cout << endl;
   }
   fclose(pf);
}
```

Contoh program 7.12.

```
// mengganti isi record secara random
#include <iostream>
#include <stdlib.h>
#define SATU_RECORD 1
using namespace std;

int main()
{
   struct
   {
      char judul[26];
      char pengarang[20];
      int jumlah;
   } buku;  // variabel buku bertipe struktur

   FILE *pf;  // pointer file
   char ulang;
   int nmr_record, hasil_baca;
   long int ofset_byte;

   if((pf = fopen("DAFBUKU.DAT", "rb+")) == NULL)  /* buka file yg berisi
   {                                                  daftar buku */
      cout << "File tak dapat dibuka !\r\n";
      exit(1);
   }

   do {
      cout << "\r\n Nomor record dari data yang ingin diubah : ";
      cin >> nmr_record; cin.ignore();
      ofset_byte = (nmr_record - 1) * sizeof(buku);
      fseek(pf, ofset_byte, SEEK_SET);
      hasil_baca = fread(&buku, sizeof(buku), SATU_RECORD, pf);

      if(hasil_baca == 0)
         cout<<"  Nomor record tak absah !!\r\n";
      else
      {
         cout << "  Judul buku  : " << buku.judul <<"\r\n";
         cout << "  Pengarang   : " << buku.pengarang <<"\r\n";
         cout << "  Jumlah buku : " << buku.jumlah <<"\r\n";
         cout << "  Jumlah buku kini : "; cin >> buku.jumlah;
        cin.ignore();

         fseek(pf, ofset_byte, SEEK_SET);
         fwrite(&buku, sizeof(buku), SATU_RECORD, pf);
      }
      cout << "\r\n  Mau mencari informasi lagi (Y/T) : ";
      ulang = getchar();
      cout<< endl;
   }
   while( ulang == 'Y' || ulang == 'y');
   fclose(pf);
}
```

10. OPERASI FILE TAK BERPENAMPUNG

File tak berpenampung (system level I/O atau low level I/O) adalah bahwa system C tidak menyediakan penampung untuk keperluan operasi file, melainkan pemrogram sendirilah yang harus mengaturnya sehingga mengefektifkan kecepatan pengaksesan.

a. Fungsi _open( ), _create( ) dan _close( )

`_open()` berfungsi untuk membuka file, `_create()` berfungsi untuk menciptakan file baru atau mengosongkan file yang sudah ada, dan `_close()` berfungsi untuk menutup file.

Bentuk deklarasinya :

```
it_open(char *filename, int mode);

it_create(char *namafile, int atrib);

it_close(int handle);
```

Keterangan mode untuk _open()

Konstanta	Nilai	Pengaruh
O_RDONLY	1	Hanya bisa dibaca
O_WRONLY	2	Hanya bisa ditulis
O_RDWR	4	Baca dan Tulis

Keterangan atribut file dalam DOS (pada _create()

Nilai	Pengaruh
1	File hanya bisa dibaca
2	File tersembunyi
4	File system
8	Nama lanel volum
16	Nama subdirektori
32	Archieve

b. Membaca Dan Menulis File

Untuk membaca isi file, fungsi yang digunakan yaitu **read()**, dan untuk menulis ke file, fungsi yang digunakan yaitu**_write().** Bentuk deklarasinya :

```
int _read(int handle, void *buffer, unsigned panjang);

int  write(int handle, void *buffer, unsigned panjang);
```

Contoh program 7.13.

```
// menyalin sembarang file teks
#include <iostream>
#include <stdlib.h>
#include <fcntl.h>
#include <io.h>
#define PANJANG 256
#define UKURAN 2048
using namespace std;
char buffer[UKURAN];

int main()
{
   int file_inp, file_out;
   long int byte_terbaca;
   char string[PANJANG];
   char namafile_inp[65], namafile_out[65];

   cout << " PROGRAM UNTUK MENYALIN SEMBARANG FILE\r\n";
   cout << "  Nama file input  : "; cin >> namafile_inp;
   cout << "  Nama file output : "; cin >> namafile_out;
```

Lanjutan Contoh program 7.13.

```
    _fmode = O_BINARY;  // buka file input
    if((file_inp = _rtl_open(namafile_inp, O_RDONLY)) == -1)
    {
       cout << "File input tak dapat dibuka !!\r\n";
       exit(1);
    }

    if((file_out = _create(namafile_out, 0)) == -1 )  // buka file output
    {
       cout<<"File output tak dapat dibuka !!\r\n";
       exit(1);
    }

    for ( ; ; )  // proses menyalin file
    {
       byte_terbaca = _read(file_inp, buffer, sizeof(buffer));
       if(byte_terbaca == 0L) break;
       _write(file_out, buffer, byte_terbaca);
    }
    close(file_inp);  close(file_out);
}
```

11. MENGHAPUS FILE

Fungsi yang berguna untuk menghapus file, yaitu **`unlink()`**. Bentuk deklarasinya :

```
int unlink (char *namafile);
```

Contoh program 7.14.

```
// menghapus file
#include <iostream>
#include <stdio.h>
#include <unistd.h>
using namespace std;

int main()
{
   int kode;
   char namafile[65];

   cout << "\n  Nama file yang akan dihapus : ";
   cin >> namafile;

   kode = unlink(namafile);
   if(kode == -1)
      cout << "  Gagal dalam menghapus !!file\r\n";
   else
      cout << "  Ok.File sudah dihapus.\r\n";

   getchar();
}
```

12. MENGGANTI NAMA FILE

Fungsi yang berguna untuk menghapus file, yaitu **`rename()`**. Bentuk deklarasinya :

```
int rename(char *namafilelama, char *namafilebaru);
```

Contoh program 7.15.

```
// mengganti nama file

#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
   int kode;
   char namafilelama[65], namafilebaru[65];

   cout<<"\n  Nama file yang akan diganti : "; cin>>namafilelama;
   cout<<"\n  Nama file pengganti         : "; cin>>namafilebaru;

   kode = rename( namafilelama, namafilebaru );

   if( kode == -1 )
      cout<<"  Gagal dalam mengganti nama file !!\r\n";
   else
      cout<<"  Ok.Nama file sudah diganti !!\r\n";

  getchar();
}
```

1. Buatlah tiga kelompok di plug Anda. Masing-masing kelompok menjalankan salah satu nomor di bawah ini :
   a. contoh program 7.1, 7.2, 7.7, 7.8, 7.13, 7.14 dan 7.15
   b. contoh program 7.5, 7.6, 7.7, 7.8, 7.11, 7.12 dan 7.13
   c. contoh program 7.3, 7.4, 7.9, 7.10, 7.13, 7.14, dan 7.15

B. TUGAS PRAKTIKUM

Dengan menggunakan metode penyimpanan dan pembacaan file per blok, buatlah sebuah program untuk menyimpan dan membaca sebuah isi file yang bertipe struktur, sebagai berikut:

```
struct NilaiProtek2 {
      int nosiswa[5];
      char nama[20];
      int nilai[3];
};
```

Program Anda buat dengan ketentuan :

a. Buat 2 buah fungsi, yaitu fungsi untuk menuliskan ke file dan fungsi untuk membaca sekaligus menampilkan ke layar isi dari file. Beri nama file tersebut dengan nama “data.txt”
b. Banyaknya data nilai protek 2 yang disimpan ke file fleksibel (min. 10).
c. Buat program dalam bentuk tampilan menu sebagai berikut :

```
/// Program tulis dan baca dari file ///
/// MENU PILIHAN///
[1] Tulis ke file
[2] Baca dari file
[3] Keluar
==================
Masukkan Pilihan Anda [1..3] :
```