

PRAKTIKUM 6

SEARCHING & SORTING

A. Tujuan

Setelah praktikum ini, praktikan diharapkan dapat:

1. Memahami tipe data dasar
2. Memahami tipe data bentukan
3. Struktur Program menggunakan bahasa C++

B. Peralatan

1. PC Desktop
2. Windows 7
3. Notepad++ dan MinGW atau Dev++

C. SEARCHING (Pencarian)

Secara umum search diartikan mencari data dengan cara menelusuri tempat penyimpanan data tersebut. Tempat penyimpanan data dapat berupa array dalam memory, bisa juga berada dalam suatu file pada external storage. Searching adalah proses mencari atau pencarian. Data yang disearch adalah data yang berada dalam array satu dimensi. Dalam praktikum ini kita akan membahas proses pencarian data dengan metode :

1. *Sequential Search*
2. *Binary Search*

Sequential Search

Metode Sequential Search dapat digunakan untuk melakukan pencarian data baik pada array yang **sudah terurut** maupun yang **belum terurut**. Proses yang terjadi pada metode pencarian ini adalah sebagai berikut:

1. Menentukan data yang dicari.
2. Data yang dicari akan dibandingkan dengan data dalam array satu persatu mulai dari data ke-0 sampai index terakhir, tetapi
3. Jika perbandingan antara data yang dicari dengan data dalam array cocok maka proses dihentikan.

Binary Search

Metode Binary Search hanya digunakan untuk pencarian data yang sudah terurut. Proses yang terjadi pada pencarian dengan metode ini adalah sebagai berikut:

1. Menentukan data yang akan dicari.
2. Menentukan elemen tengah dari array.
3. Jika nilai elemen tengah sama dengan data yang dicari maka pencarian selesai, tetapi
4. Jika nilai elemen tengah tidak sama dengan data yang dicari maka:
 - a. Jika nilai elemen tengah lebih besar daripada data yang dicari maka pencarian dilakukan pada setengah array pertama, tetapi
 - b. Jika nilai elemen tengah lebih kecil daripada data yang dicari maka pencarian dilakukan pada setengah array berikutnya.

Proses pencarian dengan metode Binary Search selain menggunakan Looping dapat pula dibuat dengan rekursi. Berikut ini contoh penggunaan rekursi untuk pencarian dengan binary search :

```
int binary_search(int a[], int l, int r, int key) {  
    while (l <= r) {  
        int m = l + (r - l) / 2;  
        if (key == a[m])  
            return m;  
        else if (key < a[m])  
            r = m - 1;  
        else  
            l = m + 1;  
    }  
    return -1;  
}
```

Penggunaan algoritma ini harus diawali dengan algoritma pengurutan terlebih dahulu. Program di atas merupakan fungsi binary search yang dapat dipanggil dari main dengan mengirimkan parameter array data, posisi awal, posisi akhir dan nilai yang akan dicari (key).

Fungsi akan mencari elemen key pada array yang sudah diurutkan dari indeks low hingga high. Jika elemen ditemukan, maka program akan mengembalikan indeks dari elemen tersebut. Jika tidak ditemukan, maka program akan mengembalikan nilai -1.

D. PRAKTIKUM 1

Percobaan 1 :

Sebagai ilustrasi ada sebuah array berukuran 7 bertipe `int` dengan nama `ar` berisi data {5, 9, 2, 7, 8, 1, 6}. Data yang akan dicari adalah 7 yang dinyatakan sebagai `x`. Berikut ini adalah contoh proses pencarian yang bisa digunakan.

Proses pencarian secara linier :

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5
6      int x=1; // nilai yg akan dicari
7      int arr[]={5,9,2,7,8,1,6};
8
9      //Proses pencarian secara linier :
10     bool found{false};
11     for (int i = 0 ; i < 7; i++) {
12         if(x == arr[i])
13         {
14             found = true;
15             cout << "Data ditemukan di indeks ke-"<< i;
16             break;
17         }
18     }
19 }
```

Pada potongan kode di atas terdapat variabel `x` sebagai penampung data yang di cari yaitu 7 dan variabel `k` sebagai penanda apakah data yang dicari ketemu. Sebagai nilai awal variable `found` adalah `false` sebagai penanda bahwa data belum ditemukan.

Pada saat proses perulangan ke-`i` nilai `x` akan selalu dibandingkan dengan nilai `arr[i]`. Jika perbandingan sesuai maka nilai `found` akan diset menjadi `true` dan perulangan dihentikan.

Akhir proses dari contoh di atas akan didapat nilai `k` adalah 3, sehingga pencarian dikatakan berhasil yaitu data ketemu pada index yang ke-3.

Hasil :

```
Data ditemukan di indeks ke-5
-----
Process exited after 0.1177 seconds with return value 0
Press any key to continue . . .
```

Latihan 1 :

Cobalah cari data yang tidak ada di dalam array dan munculkan peringatan **"Data tidak ditemukan!"**

Percobaan 2 :

Sebagai ilustrasi terdapat array berukuran **11** bertipe **int** dengan nama **ar** berisi data terurut naik {1, 2, 5, 7, 8, 9, 9, 11, 15, 16, 20}.

Sebagai contoh data yang akan dicari adalah 5, dinyatakan sebagai **x** dan penanda bila data ketemu adalah **found**. Proses pencarian dapat dilakukan seperti berikut.

Proses pencarian dengan binary search :

```
4 int main(){
5     //x adl nilai yg akan dicari
6     int m,x=5;
7     int arr[]={1, 2, 5, 7, 8, 9, 9, 11, 15, 16, 20};
8     int l{0};
9     int r{10};
10
11
12     //Proses pencarian dengan binary search :
13     bool found{false};
14     while(l <= r && !found)
15     {
16         m = l + (r - l) / 2;
17
18         if(x == arr[m]){
19             found = true;
20             cout << "Data ditemukan di indeks ke-"<< m;
21         }
22         if(x < arr[m]) r = m - 1;
23         else if(x > arr[m]) l = m + 1;
24     }
25 }
```

Pengaksesan dapat dilakukan satu persatu langsung merujuk pada indeks atau serentak. Contoh pengaksesan secara serentak pada array 2 dimensi

Latihan 2 :

Modifikasilah program diatas supaya nilai yang dicari dapat diinput saat program berjalan. Setelah itu cobalah dengan data yang **ada** dan data yang **tidak ada** di dalam array serta tambahkan Validasi outputnya **“Data ditemukan”** atau **“Data tidak ditemukan”**.

Contoh output program yang diminta :

```
Data yang tersimpan :
1      2      5      7      8      9      11      13      15      16

Masukkan angka yang ingin dicari : 5
Data ditemukan di indeks ke-2

Ingin cari lagi ? (y/n) y

Masukkan angka yang ingin dicari : 10
Data tidak ditemukan

Ingin cari lagi ? (y/n)
```

E. SORTING (Pengurutan)

Metode pengurutan sekumpulan data menggunakan 3 metode yaitu bubble sort, selection sort dan insertion sort.

Secara umum proses yang terjadi pada pengurutan adalah sebagai berikut:

1. Perbandingan data
2. Pertukaran data

1. Bubble Sort

Bubble Sort mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya.

Pembandingan elemen dapat dimulai dari awal atau mulai dari paling akhir.

Percobaan 3 :

Sebagai ilustrasi ada sebuah array berukuran 6 bertipe `int` dengan nama `data` berisi data {5, 9, 2, 7, 8, 1, 6}. Data yang akan dicari adalah 7 yang dinyatakan sebagai x. Berikut ini adalah contoh proses pencarian yang bisa digunakan.

Kode Program :

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int data[10], data2[10];
5  int n;
6
7  int tukar (int a,int b){
8      int t;
9      t=data[b];
10     data[b]=data[a];
11     data[a]=t;
12     return 0;
13 }
14
15 int input(){
16     cout<<"Masukan Jumlah Data = ";
17     cin>>n;
18
19     cout<<endl;
20
21     for (int i=0;i<n;i++){
22         cout<<"Masukan Data Ke-"<<i+1<<" = ";
23         cin>>data[i];
24
25         data2[i]=data[i];
26     }
27     cout<<endl;
28     return 0;
29 }
```

```

31 int tampil(){
32     for (int i=0;i<n;i++){
33         cout<<"["<<data[i]<<"] ";
34     }
35     cout<<endl;
36     return 0;
37 }
38
39 int bubble_sort(){
40     for (int i=1; i<n;i++){
41         for (int j=n-1; j>=i;j--){
42             if (data[j]<data[j-1]){
43                 tukar(j,j-1);
44             }
45         }
46         tampil();
47     }
48     cout<<endl;
49     return 0;
50 }
51
52
53 int main()
54 {
55     cout<<"ALGORITMA BUBBLE SORT"<<endl;
56     cout<<"-----"<<endl;
57     input();
58     cout<<"Proses Bubble Sort"<<endl;
59     tampil();
60     bubble_sort();
61     getch();
62     return 0;
63 }

```

2. Selection Sort

Merupakan kombinasi antara sorting dan searching. Untuk setiap proses, akan dicari elemen-elemen yang belum diurutkan yang memiliki nilai terkecil atau terbesar akan dipertukarkan ke posisi yang tepat di dalam array. Misalnya untuk putaran pertama, akan dicari data dengan nilai terkecil dan data ini akan ditempatkan di indeks terkecil (data[0]), pada putaran kedua akan dicari data kedua terkecil, dan akan ditempatkan di indeks kedua (data[1]).

Selama proses, perbandingan dan pengubahan hanya dilakukan pada indeks pembandingan saja, pertukaran data secara fisik terjadi pada akhir proses.

Percobaan 4 :

Tambahkan fungsi berikut pada program sebelumnya :

```
int selectionSort(int data[]){
    int i, j, posisi, tukar;
    for(i = 0; i < (n-1); i++){
        posisi = i;
        for (j = i + 1; j < n; j++){
            if(data[posisi] > data[j]){
                posisi = j;
            }
        }
        if(posisi != i){
            tukar = data[i];
            data[i] = data[posisi];
            data[posisi] = tukar;
        }
    }
    return 0;
}
```

3. Insertion Sort

Mirip dengan cara orang mengurutkan kartu, selembat demi selembat kartu diambil dan disisipkan (insert) ke tempat yang seharusnya.

Pengurutan dimulai dari data ke-2 sampai dengan data terakhir, jika ditemukan data yang lebih kecil, maka akan ditempatkan (diinsert) diposisi yang seharusnya.

Pada penyisipan elemen, maka elemen-elemen lain akan bergeser ke belakang.

Percobaan 5 :

```
int insertion_sort(int data[])
{
    int temp,j;
    for (int i=1;i<n;i++){
        temp = data[i];
        j = i-1;
        while (data[j]>temp && j>=0){
            data[j+1] = data[j];
            j--;
        }
        data[j+1] = temp;
        tampil();
    }
    cout<<endl;
}
```

Latihan 3 :

Buatlah algoritma pengurutan menggunakan metode selection dan insertion sort dapat menampilkan proses pertukaran datanya seperti hasil output bubble_sort berikut ini :

```
Masukan Jumlah Data = 7

Masukan Data Ke-1 = 3
Masukan Data Ke-2 = 4
Masukan Data Ke-3 = 9
Masukan Data Ke-4 = 5
Masukan Data Ke-5 = 1
Masukan Data Ke-6 = 7
Masukan Data Ke-7 = 2

ALGORITMA BUBBLE SORT
Proses Bubble Sort
[1] [3] [4] [9] [5] [2] [7]
[1] [2] [3] [4] [9] [5] [7]
[1] [2] [3] [4] [5] [9] [7]
[1] [2] [3] [4] [5] [7] [9]
[1] [2] [3] [4] [5] [7] [9]
[1] [2] [3] [4] [5] [7] [9]
[1] [2] [3] [4] [5] [7] [9]
```


F. Studi Kasus 1

Terdapat sebuah array dengan ukuran maksimal 20. Elemen array diisi dengan input oleh pengguna saat program berjalan. Program harus bisa melakukan pencarian terhadap elemen array dan bila ketemu maka harus menampilkan semua posisi index-nya bila terdapat lebih dari satu elemen yang sama nilainya.

```
/**
 * program sequential search
 * dapat menampilkan semua index data yang ditemukan
 */
#include <iostream>
using namespace std;
const int MAX_SIZE{20}
int data[MAX_SIZE]; // array data
int idx[MAX_SIZE]; // array untuk menyimpan index elemen yang ditemukan
int count{0}; // counter, menghitung ada berapa banyak data yang ditemukan

void search(int x);
int main(void) {
    int n;
    cout << "jumlah data: ";
    cin << n;

    for(auto i = 0; i < n; ++i) {
        cout << "data ke-"<< i ;
        cin >> data[i];
    }

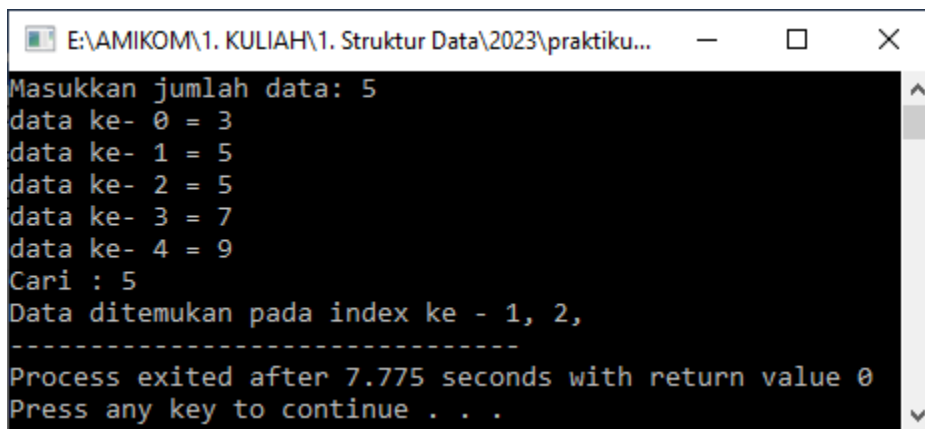
    int x;
    cout << "cari: ";
    cin >> x;
    search(x);
    // jika counter > 0, berarti ada data yang ditemukan
    if(count > 0)
    {
        cout << "ditemukan pada index: ";
        for(auto i = 0; i < count; ++i)
        {
            cout << idx[i] << ", ";
        }
    }
    else
    {
        Cout << "data tidak ditemukan\n";
    }
    return 0;
}
```

```

void search(int x) {
    for(auto i = 0; i < n; ++i)
    {
        // jika x ditemukan pada data[i]
        if(x == data[i])
        {
            // simpan index i ke array idx
            idx[count++] = i;
        }
    }
}

```

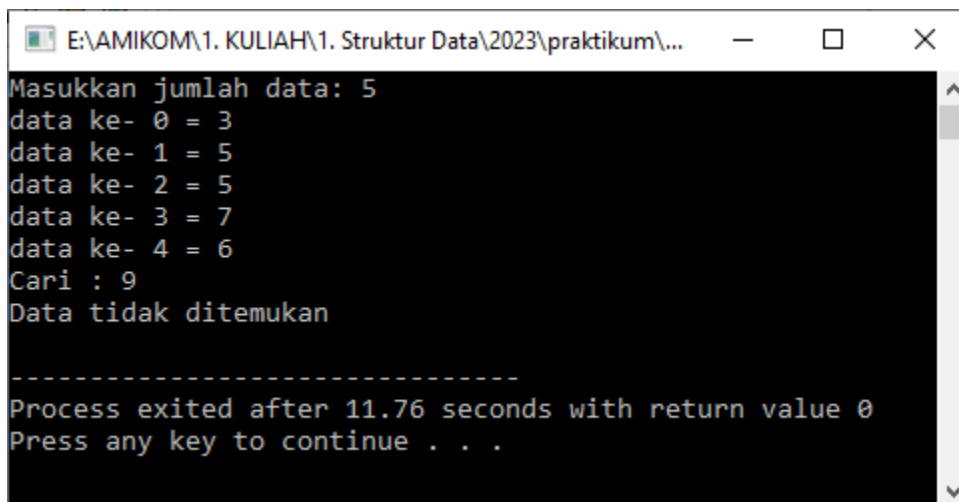
Sesuaikan kode program diatas supaya hasil running seperti berikut :



```

E:\AMIKOM\1. KULIAH\1. Struktur Data\2023\praktiku...
Masukkan jumlah data: 5
data ke- 0 = 3
data ke- 1 = 5
data ke- 2 = 5
data ke- 3 = 7
data ke- 4 = 9
Cari : 5
Data ditemukan pada index ke - 1, 2,
-----
Process exited after 7.775 seconds with return value 0
Press any key to continue . . .

```



```

E:\AMIKOM\1. KULIAH\1. Struktur Data\2023\praktikum\...
Masukkan jumlah data: 5
data ke- 0 = 3
data ke- 1 = 5
data ke- 2 = 5
data ke- 3 = 7
data ke- 4 = 6
Cari : 9
Data tidak ditemukan
-----
Process exited after 11.76 seconds with return value 0
Press any key to continue . . .

```

Studi Kasus 2

Berikut program untuk membandingkan kecepatan pengurutan metode selection, bubble, dan insertion. Data dalam array di-generate secara acak. Tiap metode akan dicatat waktu sebelum dan sesudah proses pengurutan yang kemudian akan diperoleh selisih waktu dalam satuan milisecond. Berikut ini contoh tampilan program waktu berjalan.

```
#include <iostream>
#include <ctime>
#include <windows.h>

using namespace std;
const int SIZE{25};
const int BUBBLE{0};
const int SELECTION{1};
const int INSERTION{2};

int nsrc[SIZE];
int ndata[SIZE];

void init_data();
void load_data();
void view_data();
void bubsort();
void selsort();
void inssort();
void run_sort(int);

int main(void) {
    cout << "Perbandingan Metode Pengurutan\n";
    init_data();
    load_data();

    cout << "Data Acak:\n";
    view_data();
    cout << "\n\n";

    // Bubble Sort
    cout << "Bubble Sort\n";
    load_data();
    run_sort(BUBBLE);

    // Selection Sort
    cout << "Selection Sort\n";
    load_data();
    run_sort(SELECTION);

    // Insertion Sort
```

```

load_data();
cout << "Insertion Sort\n";
run_sort(INSERTION);
return 0;
}
void init_data(void)
{
    srand(time(NULL));
    for(int i = 0; i < SIZE; ++i)
        nsrc[i] = rand() % 100;
}
void load_data(void)
{
    for(int i = 0; i < SIZE; ++i)
        ndata[i] = nsrc[i];
}
void view_data(void)
{
    for(int i = 0; i < SIZE; ++i)
        printf("%d ", ndata[i]);
    cout << endl;
}
void bubsort(void)
{
    for(int i = 0; i < SIZE - 1; ++i)
    {
        for(int j = 0; j < SIZE - i - 1; ++j)
        {
            if(ndata[j] > ndata[j + 1])
            {
                int tmp = ndata[j];
                ndata[j] = ndata[j + 1];
                ndata[j + 1] = tmp;
                Sleep(10);
            }
        }
        Sleep(10);
    }
}
void selsort(void)
{
    for(int i = 0; i < SIZE; ++i)
    {
        int min = i;
        for(int j = i; j < SIZE; ++j)
        {
            if(ndata[j] < ndata[min])
            {
                min = j;
            }
        }
    }
}

```

```

        Sleep(10);
    }
    Sleep(10);
}

int tmp = ndata[i];
ndata[i] = ndata[min];
ndata[min] = tmp;
Sleep(10);
}
}
void inssort(void)
{
    for(int i = 1; i < SIZE; ++i)
    {
        int m = ndata[i];
        s = i;
        while(s >= 0 && m < ndata[s - 1])
        {
            ndata[s] = ndata[--s];
            Sleep(10);
        }
        ndata[s] = m;
        Sleep(10);
    }
}
void run_sort(int m)
{
    auto t1 = time(NULL);
    switch(m)
    {
        case 0: bsort(); break;
        case 1: selsort(); break;
        case 2:
            default: inssort(); break;
    }
    auto t2 = time(NULL);
    auto t = t2 - t1;
    cout << "data terurut:\n";
    view_data();
    cout << "waktu: " << t << " milisecond\n";
    cout << "-----\n\n";
}

```

Tugas Persiapan Responsi :

Buatlah program perbandingan 3 metode pengurutan diatas kemudian berikan kesimpulan dari hasil yang kalian dapatkan.