DESIGN DAN ANALISIS ALGORITMA

TUGAS AKHIR TENGAH SEMESTER 2

Dosen: Jemy Arieswanto, S.Kom., M.Kom.

Asdos: Shania Oktaviani Gunawan

Disusun Oleh:

Rifandy Arnas (232310001)

FAKULTAS INFORMATIKA DAN PARIWISATA

INSTITUT BISNIS DAN INFORMATIKA KESATUAN

2024

DAFTAR ISI

BAB I
SORTING ALGORITHM

1.1 DASAR TEORI

Sorting adalah sebuah proses yang mengurutkan elemen-elemen array yang acak mulai dari yang terkecil ke terbesar atau dari terbesar ke terkecil. Sorting dari yang terkecil ke terbesar disebut *Ascending*, dan sorting dari terbesar ke terkecil disebut *Descending*. Algoritma penyortiran berfungsi untuk mengatur ulang array atau elemen daftar tertentu sesuai dengan operator perbandingan pada elemen tertentu. Algoritma penyortiran bertujuan untuk mengatur elemen data agar lebih mudah ditemukan pada saat proses pencarian (searching).

1.2 BUBBLE SORT

Bubble Sort adalah algoritma penyortiran paling sederhana dan mudah, baik dalam konsep dan penerapannya di dalam sebuah program. Bubble sort juga terinspirasi dari nama 'gelembung' yang berarti berat jenis gelembung lebih kecil dari berat air, sehingga akan mengapung di atas permukaan.

```
1  #include<iostream>
2  using namespace std;
3
4  int main ()
5  {
6      int i, j,temp,pass=0;
7      int a[10] = {10,2,0,14,43,25,18,1,5,45};
8      cout <<"Input list ...\n";
9      for(i = 0; i<10; i++) {
10         cout <<a[i]<<"\t";
11     }
```

#include <iostream> adalah pembuka dari script pada C++ dengan librarynya adalah iostream atau keseluruhan. **Using namespace std** harus digunakan sebelum membuat program. Pada bagian **main**, diketahui **int i, j, temp, pass** = 0 yang menunjukan bahwa nilai dari variable ini harus berupa bilangan bulat. Nilai dari **int pass** saat ini adalah 0.

Terdapat sebuah array dengan nama **a** dan berisi sebanyak 10 angka. Element dari array selalu dimulai dari 0, sehingga urutan nya adalah 0 sampai 9. Program mendeklarasikan **cout** untuk memanggil tulisan "Input list …." Dan **\n** berfungsi untuk Enter sebanyak 1 baris ke bawah. Program melakukan looping **untuk i=0** dan **i** kurang dari 10 (nilai pada array), maka variable **i** akan mengalami increment (pertambahan). Program memanggil array **a** dengan urutan sesuai variable **i** atau bisa ditulis **a[i]** yang berarti, jika saat ini variable **i** bernilai 3, maka **a[3]** adalah 14.

```
for(i = 0; i<10; i++) {
    for(j = i+1; j<10; j++)
    {
        if(a[j] < a[i]) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
pass++;
}
```

Looping ini dilakukan untuk melakukan bubble sorting. Main loop akan dijalankan sampai **i** bernilai 9 dan setiap looping, **pass** akan mengalami increment sebanyak 1. Looping **untuk j=i+1**. Ketika **i** bernilai 4, maka **j** akan berniilai **4+1 = 5**. Jika nilai **a[j]** kurang dari **a[i]** maka lakukan penukaran dengan cara seperti di gambar.

```
cout <<"Sorted Element List ...\n";
for(i = 0; i<10; i++) {
    cout <<a[i]<<"\t";
}

cout<<"\nNumber of passes taken to sort the list:"<<pass<<endl;
return 0;
}
```

Sistem akan memanggil element array yang sudah di sorting dengan menggunakan looping. Sistem juga memanggil berapa kali proses yang dilakukan untuk mengurutkan element-element array. Berikut ini adalah full script dan output yang dihasilkan!

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

int main ()
{
    int i, j,temp,pass=0;
    int a[10] = {10,2,0,14,43,25,18,1,5,45};
    cout <<"Input list ...\n";
    for(i = 0; i<10; i++) {
        cout <<a[i]<<"\t";
    }
cout<<endl;
for(i = 0; i<10; i++) {
    for(j = i+1; j<10; j++)
    {
        if(a[j] < a[i]) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
pass++;
}

cout <<"Sorted Element List ...\n";
for(i = 0; i<10; i++) {
    cout <<a[i]<<"\t";
}

cout<<"\nNumber of passes taken to sort the list:"<<pass<<endl;
return 0;
}
```

```
Input list ...
10      2       0       14      43      25      18      1       5       45
Sorted Element List ...
0       1       2       5       10      14      18      25      43      45
Number of passes taken to sort the list:10

--------------------------------
Process exited after 0.02219 seconds with return value 0
Press any key to continue . . .
```

1.3 INSERTION SORT

Insertion Sort adalah pengurutan yang membandingkan dua element pertama pada sebuah array dan mengurutkannya. Lalu, mengecek element data berikutnya satu persatu dan memabandingkannya dengan element data yang telah diurutkan oleh sistem.

```
void Sorting_Insertion(int ahoy[], int banyak){
    int i, j;

    for (i = 1; i < banyak; i++)
    {
        int variant = ahoy[i];

        for (j = i - 1; j >= 0; j--)
        {

            if (ahoy[j] > variant)
            {

                ahoy[j + 1] = ahoy[j];
            }
            else
            {
                break;
            }
        }
        ahoy[j + 1] = variant;
    }
```

Terdapat sebuah fungsi **void** dengan nama Sorting_Insertion dan memiliki parameter berupa **int ahoy[ ]** (array kosong yang siap di isi dari bagian **int main**) dan **int banyak**. Main loop akan dilakukan sebanyak nilai dari element array itu, jika diketahui sebuah array dengan banyak nilai 10, maka main loop akan dilakukan sebanyak 9 kali. **Variant** saat ini adalah element array ke [i], berarti element ke 1 dari array.

Untuk j=i-1 yang berarti nilai **j** saat ini adalah **1-1 = 0** dan nilai **j** lebih besar dari sama dengan 0, maka **j** akan melakukan decrement (pengurangan) sebanyak 1. Jika isi dari array ahoy ke [j] lebih besar dari variant, lakukan penukaran antara element array ahoy ke [j + 1] (j + 1 = 0 + 1 = 1 berarti element ke 1) dengan element array ahoy ke [j] (element ke 0). Jika isi dari array ahoy ke [j] lebih kecil dari variant, lakukan **break** atau perhentian loop. Karena nilai **j** saat ini adalah -1, maka syarat looping (j >= 0) tidak terpenuhi, sistem langsung keluar dari loop dan menjalankan array **ahoy [j+1] = variant**.

```
31      cout << "Data setelah sorting:" << endl;
32
33      for (int i = 0; i <= banyak - 1; i++)
34      {
35
36          cout << ahoy[i] << " ";
37      }
38  }
```

Sistem akan memanggil array **ahoy [i]** setelah di sorting dengan loop. Ketika nilai dari variable **banyak** adalah 5, maka **5-1 = 4**, akan dilakukan loop sebanyak 4 kali, dimulai dari 0 sampai 4.

```
40  int main()
41  {
42      int data[] = {78, 56, 85, 96, 21, 14};
43
44      int size = 6;
45
46      Sorting_Insertion(data, size);
47  }
```

Pada bagian utama, dideklarasikan sebuah array bernama **data [ ]** yang berisikan sebanyak 6 nilai. Sistem akan memanggil fungsi **void Sorting_Insertion** dengan parameter **data** yang akan diubah menjadi parameter **ahoy[ ]** di dalam fungsi **void** dan parameter **size** yang akan diubah menjadi parameter **banyak** di dalam fungsi **void**. Berikut adalah full script dan output yang dihasilkan!

```cpp

#include <iostream>
using namespace std;


void Sorting_Insertion(int ahoy[], int banyak){

    int i, j;

    for (i = 1; i < banyak; i++)
    {
        int variant = ahoy[i];

        for (j = i - 1; j >= 0; j--)
        {

            if (ahoy[j] > variant)
            {

                ahoy[j + 1] = ahoy[j];
            }
            else
            {
                break;
            }
        }
        ahoy[j + 1] = variant;
    }

    cout << "Data setelah sorting:" << endl;

    for (int i = 0; i <= banyak - 1; i++)
    {

        cout << ahoy[i] << " ";
    }
}

int main()
{
    int data[] = {78, 56, 85, 96, 21, 14};

    int size = 6;

    Sorting_Insertion(data, size);
}
```

```
Data setelah sorting:
14 21 56 78 85 96
--------------------------------
Process exited after 0.03124 seconds with return value 0
Press any key to continue . . .
```

1.4 SELECTION SORT

Selection Sort adalah algoritma pengurutan sederhana yang digunakan untuk mengurutkan element-element pada sebuah array. Algoritma ini bekerja dengan mencari element terkecil atau terbesar dalam array dan menukar posisinya dengan element pertama pada array. Kemudian, mencari element terkecil atau terbesar berikutnya dalam sisa

array yang belum terurut dan menukar posisinya dengan element kedua. Proses ini akan diulang secara berulang hingga seluruh array terurut.

```
3  #include<iostream>
4  using namespace std;
5
6
7  void tampilan(int ahoy[], int jitsu)
8  {
9      int i;
10
11     for (i = 0; i < jitsu; i++)
12
13         cout << ahoy[i] << "\t";
14
15         cout <<endl;
16 }
```

Void tampilan dengan parameter **ahoy[ ]** dan **jitsu** merupakan sebuah fungsi yang akan digunakan untuk menampilkan hasil pengurutan array sebelum di sorting dan sesudah di sorting.

```
19 int main()
20 {
21     int sebuah_data[] = {125, 146, 112, 107, 264, 985, 645, 54};
22
23     int ukuran = sizeof(sebuah_data)/sizeof(sebuah_data[0]);
24
25     cout << "Data Sebelum di Sorting: \n";
26
27     tampilan(sebuah_data, ukuran);
```

Pada bagian main, terdapat sebuah array dengan nama **sebuah_data** yang berisikan 8 element. Diketahui integer **ukuran = sizeof (sebuah_data)** yang berarti variable **ukuran** adalah banyaknya dari element si array, dalam konteks ini element dari array ada sebanyak 8 buah. Sistem akan memanggil fungsi **void** tampilan dengan parameter **sebuah_data** yang akan diganti menjadi **ahoy [ ]** dan parameter **ukuran** yang diganti menjadi **jitsu** pada bagian fungsi **void**.

```
29      int i, j, matematika, save;
30
31
32      for (i = 0; i < ukuran-1; i++)
33      {
34
35          matematika = i;
36
37          for (j = i+1; j < ukuran; j++){
38
39              if (sebuah_data[j] < sebuah_data[matematika])
40
41                  matematika = j;
42          }
43
44
45          save = sebuah_data[matematika];
46
47          sebuah_data[matematika] = sebuah_data[i];
48
49          sebuah_data[i] = save;
50      }
```

Main loop akan melakukan looping sebanyak 7 kali (8 – 1 =7) dimulai dari 0 sampai dengan 6. Di dalam looping ini terdapat deklarasi **matematika = i** yang berarti nilai dari variable **matematika** adalah nilai dari variable **i** dan pada saat ini nilai **i** adalah 0. Loop anak akan melakukan sorting dengan **j=i+1** (j = 0+1, j=1) selama variable j kurang dari 8 dan j akan mengalami increment sebanyak 1. Jika array dari sebuah_data ke [j] kurang dari array sebuah_data dari element [matematika], maka **matematika = j**. Pada konteks ini, variable dari sebuah_data ke [j] (data ke [1]) adalah 146 kurang dari sebuah_data dari element [matematika] (element ke [0]) adalah 125, maka loop berhenti karena syarat tidak terpenuhi.

```
44
45          save = sebuah_data[matematika];
46
47          sebuah_data[matematika] = sebuah_data[i];
48
49          sebuah_data[i] = save;
50      }
51
52      cout << "Data setelah di sorting: \n";
53
54      tampilan(sebuah_data, ukuran);
55
56      return 0;
57  }
```

Setelah loop anak berhenti, sistem akan melakukan penukaran data dengan rumus sesuai di gambar. Pada konteks ini, nilai 125 tidak ditukar dengan nilai apapun. Sistem akan memanggil fungsi **void** tampilan untuk menampilkan hasil data setelah di sorting. Berikut adalah full script dan output yang dihasilkan!

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


void tampilan(int ahoy[], int jitsu)
{
    int i;

    for (i = 0; i < jitsu; i++)

        cout << ahoy[i] << "\t";

        cout<<endl;
}

int main()
{
    int sebuah_data[] = {125, 146, 112, 107, 264, 985, 645, 54};

    int ukuran = sizeof(sebuah_data)/sizeof(sebuah_data[0]);

    cout << "Data Sebelum di Sorting: \n";

    tampilan(sebuah_data, ukuran);

    int i, j, matematika, save;


    for (i = 0; i < ukuran-1; i++)
    {

        matematika = i;

        for (j = i+1; j < ukuran; j++){

            if (sebuah_data[j] < sebuah_data[matematika])

                matematika = j;
        }


        save = sebuah_data[matematika];

        sebuah_data[matematika] = sebuah_data[i];

        sebuah_data[i] = save;
    }

    cout << "Data setelah di sorting: \n";

    tampilan(sebuah_data, ukuran);

    return 0;
}
```

```
Data Sebelum di Sorting:
125     146     112     107     264     985     645     54
Data setelah di sorting:
54      107     112     125     146     264     645     985

--------------------------------
Process exited after 0.1036 seconds with return value 0
Press any key to continue . . .
```

BAB II
SORTING ALGORITHM 2

2.1 QUICK SORT

Sebuah metode pengurutan yang menjadikan sebuah tabel data yang akan diurutkan menjadi dua buah sub bagian yang ditelurusi pada bagian sisi kiri dan sisi kanan disebut dengan Quick Sort. Metode sorting Quick ini mengurutkan data menggunakan strategi *divide and conquer* untuk membagi sebuah array menjadi dua sub-array.

```
3  #include <iostream>
4  using namespace std;
5
6  void sorting_secara_cepat(int, int, int);
7
8  int dipecahkan(int, int, int);
9
10
```

Sistem mendeklarasikan 2 fungsi yaitu fungsi **void sorting_secara_cepat** dengan parameter berupa 3 buah integer dan fungsi rekursif **int dipecahkan** dengan parameter berupa 3 buah integer.

```
13 int dipecahkan(int* ahoy, int waves, int rgb){
14
15     int pivot = ahoy[rgb];
16
17     int hasil = waves;
18
19     int i, save;
20
21     for (i = waves; i < rgb; i++) {
22
23         if (ahoy[i] <= pivot) {
24
25             save = ahoy[i];
26
27             ahoy[i] = ahoy[hasil];
28
29             ahoy[hasil] = save;
30
31             hasil++;
32         }
33     }
34
35     save = ahoy[rgb];
36
37     ahoy[rgb] = ahoy[hasil];
38
39     ahoy[hasil] = save;
40
41     return hasil;
42 }
```

Sistem mendeklarasikan kembali fungsi rekursif **int dipecahkan** dengan parameter pointer dari integer pada fungsi rekursif baris ke 8, parameter **int waves**, dan parameter **int rgb**. Diketahui bahwa nilai pivot adalah nilai dari array **ahoy [rgb]** yang nanti di kirim oleh **int main**. Diketahui bahwa nilai hasil adalah nilai dari parameter **waves** yang nanti di kirim oleh **int main**.

Sistem melakukan looping selama variable i < rgb, maka variable i akan mengalami increment. Jika nilai dari array ahoy [i] kurang dari sama dengan nilai pivot, maka lakukan penukaran dengan rumus pada baris 25 sampai 31. Sistem akan melakukan penempatan balik angka yang sudah di sorting pada posisi semula dengan menggunakan rumus pada baris ke 35 sampai 41.

```
46 void sorting_secara_cepat(int* ahoy, int waves, int rgb){
47
48     if (waves < rgb) {
49
50         int hasil = dipecahkan(ahoy, waves, rgb);
51
52         sorting_secara_cepat(ahoy, waves, hasil - 1);
53
54         sorting_secara_cepat(ahoy, hasil + 1, rgb);
55     }
56 }
```

Sistem mendeklarasikan lagi fungsi **void** dengan parameter yang sama seperti gambar sebelumnya. Jika nilai parameter **waves** lebih kecil dari nilai parameter **rgb**, maka hasil akan memanggil fungsi rekursif **dipecahkan** dengan parameter yang sama. Sistem akan memanggil fungsi **void** sorting_secara_cepat dengan parameter ahoy, waves, dan hasil – 1. Lalu sistem memanggil lagi fungsi **void** dengan parameter ahoy, hasil +1 1, dan rgb. Parameter ahoy, waves, hasil -1, dan hasil + 1 akan mendapatkan tipe data berupa **integer** dari fungsi pada baris ke 6.

```
58 int main(){
59
60     int masukan;
61
62     cout << "Ketikan jumlah element data yang mau di sortir: " ;
63
64     cin >> masukan;
65
66     int wuthering[masukan];
67
68     for (int i = 0; i < masukan; i++) {
69
70     cout << "Ketik element ke  " << i+1 << " : ";
71
72     cin >> wuthering[i];
73
74     }
```

User akan menginput jumlah element data yang akan di sortir, misalkan user memasukan 5 element data. Sistem akan membuat sebuah array bernama **wuthering** dengan element sebanyak 5. User akan menginput kembali nilai masing-masing element, mulai dari element 1 sampai 5. Misalkan user menginput nilai wuthering [5] = {784,9847,65,474,325}.

```
76      sorting_secara_cepat(wuthering, 0, masukan - 1);
77
78      cout << "Data setelah di sorting: ";
79
80      for (int i = 0; i < masukan; i++) {
81
82          cout << wuthering[i] << " ";
83
84      }
85
86      return 0;
87  }
```

Setelah user menginput, sistem akan memanggil fungsi **void** sorting_secara_cepat dengan parameter **wuthering** yang akan mendapatkan tipe data **integer** (baris ke 6). Pada baris ke 46, bisa dilihat bahwa parameter menggunakan pointer yang menuju kepada tipe data integer pada baris ke 6. Oleh karena itu, parameter pada baris ke 6 (int wuthering) akan dikirim ke fungsi **void** pada baris ke 46 dan parameter int wuthering akan diganti menjadi parameter **ahoy**.

Parameter **0** yang akan mendapatkan tipe data **integer** (baris ke 6). Pada baris ke 46, bisa dilihat bahwa parameter menggunakan pointer yang menuju kepada tipe data integer pada baris ke 6. Oleh karena itu, parameter pada baris ke 6 (int 0) akan dikirim ke fungsi **void** pada baris ke 46 dan parameter int 0 akan diganti menjadi parameter **waves**.

Parameter **masukan - 1** yang akan mendapatkan tipe data **integer** (baris ke 6). Pada baris ke 46, bisa dilihat bahwa parameter menggunakan pointer yang menuju kepada tipe data integer pada baris ke 6. Oleh karena itu, parameter pada baris ke 6 (int masukan - 1) akan dikirim ke fungsi **void** pada baris ke 46 dan parameter int masukan – 1 akan diganti menjadi parameter **rgb**.

Sistem akan melakukan proses sorting yang rumusnya sudah dideklarasikan pada fungsi-fungsi di atas. Setelah sorting, sistem akan memanggil data yang telah di sorting dengan menggunakan looping. Berikut adalah full script dan output yang di hasilkan!

```cpp

#include <iostream>
using namespace std;

void sorting_secara_cepat(int, int, int);

int dipecahkan(int, int, int);



int dipecahkan(int* ahoy, int waves, int rgb){

    int pivot = ahoy[rgb];

    int hasil = waves;

    int i, save;

    for (i = waves; i < rgb; i++) {

        if (ahoy[i] <= pivot) {

            save = ahoy[i];

            ahoy[i] = ahoy[hasil];

            ahoy[hasil] = save;

            hasil++;
        }
    }

    save = ahoy[rgb];

    ahoy[rgb] = ahoy[hasil];

    ahoy[hasil] = save;

    return hasil;
}


void sorting_secara_cepat(int* ahoy, int waves, int rgb){

    if (waves < rgb) {

        int hasil = dipecahkan(ahoy, waves, rgb);

        sorting_secara_cepat(ahoy, waves, hasil - 1);

        sorting_secara_cepat(ahoy, hasil + 1, rgb);
    }
```

```
void sorting_secara_cepat(int* ahoy, int waves, int rgb){
    if (waves < rgb) {
        int hasil = dipecahkan(ahoy, waves, rgb);
        sorting_secara_cepat(ahoy, waves, hasil - 1);
        sorting_secara_cepat(ahoy, hasil + 1, rgb);
    }
}
int main(){
    int masukan;
    cout << "Ketikan jumlah element data yang mau di sortir: " ;
    cin >> masukan;
    int wuthering[masukan];
    for (int i = 0; i < masukan; i++) {
    cout << "Ketik element ke  " << i+1 << " : ";
    cin >> wuthering[i];
    }
    sorting_secara_cepat(wuthering, 0, masukan - 1);
    cout << "Data setelah di sorting: ";
    for (int i = 0; i < masukan; i++) {
        cout << wuthering[i] << " ";
    }
    return 0;
}
```

```
Ketikan jumlah element data yang mau di sortir: 5
Ketik element ke  1 : 784
Ketik element ke  2 : 9847
Ketik element ke  3 : 65
Ketik element ke  4 : 474
Ketik element ke  5 : 325
Data setelah di sorting: 65 325 474 784 9847
--------------------------------
Process exited after 32.1 seconds with return value 0
Press any key to continue . . .
```

2.2 MERGE SORT

Sebuah algoritma pengurutan dalam ilmu komputer yang dirancang untuk memenuhi kebutuhan pengurutan atas suatu rangkaian data yang tidak memungkinkan untuk ditampung dalam memory computer, karena jumlahnya yang terlalu besar disebut dengan Merge Sort. Algoritma merge sort sendiri memiliki sebuah set pointer yang mennjuk suatu posisi di dalam satu set daftar.

```
#include <iostream>

#define MAX_SIZE 5

using namespace std;

void merge_sort(int, int);
void merge_array(int, int, int, int);

int arr_sort[MAX_SIZE];
```

#define MAX_SIZE 5 adalah sebuah deklarasi pembuka yang menyatakan bahwa MAX_SIZE pasti bernilai 5 (tidak lebih, tidak kurang). **Void merge_sort** dengan 2 parameter integer dan **void merge_array** dengan 4 parameter integer, merupakan sebuah fungsi. Sistem membuat sebuah array dengan nama **arr_sort** dengan jumlah element sebanyak [MAX_SIZE] (jumlah = 5).

```
int main() {
    int i;

    cout << "\nMasukkan " << MAX_SIZE << " data yang mau sorting: " << endl;
    for (i = 0; i < MAX_SIZE; i++)
        cin >> arr_sort[i];

    cout << "\nData sebelum di sorting:";
    for (i = 0; i < MAX_SIZE; i++) {
        cout << "\t" << arr_sort[i];
    }

    merge_sort(0, MAX_SIZE - 1);

    cout << "\n\nData setelah di sorting:";
    for (i = 0; i < MAX_SIZE; i++) {
        cout << "\t" << arr_sort[i];
    }

    getch();
}
```

Pada bagian utama, sistem membuat sebuah deklarasi supaya user bisa menginputkan data yang mau di sorting, menampilkan data sebelum dan sesudah di sorting, dan sistem memanggil fungsi **void** untuk melakukan proses sorting.

```
void merge_sort(int i, int j) {
    int m;

    if (i < j) {
        m = (i + j) / 2;
        merge_sort(i, m);
        merge_sort(m + 1, j);

        merge_array(i, m, m + 1, j);
    }
}
```

Fungsi **void merge_sort** digunakan untuk melakukan pemecahan data menjadi 2 bagian dengan parameter i dan j. Jika i kurang dari j, maka nilai dari variable m adalah nilai dari (i + j) dibagi dengan 2. Misal nilai **i** adalah 0 dan nilai **j** adalah 6, maka (0 + 6)/2 = 3. Sistem memanggil fungsi **void** merge_sort pada baris ke 8, sehingga parameter i dan m memiliki tipe data integer. Sistem memanggil fungsi **void** merge_sort pada baris ke 8, sehingga parameter m+1 dan j memiliki tipe data integer. Sistem memanggil fungsi **void** merge_array pada baris ke 9, sehingga parameter i, m, m+1, dan j memiliki tipe data integer.

```
48 void merge_array(int a, int b, int c, int d) {
49
50     int t[50];
51
52     int i = a, j = c, k = 0;
53
54     while (i <= b && j <= d) {
55
56         if (arr_sort[i] < arr_sort[j])
57             t[k++] = arr_sort[i++];
58
59         else
60             t[k++] = arr_sort[j++];
61     }
62
63     while (i <= b)
64         t[k++] = arr_sort[i++];
65
66     while (j <= d)
67         t[k++] = arr_sort[j++];
68
69     for (i = a, j = 0; i <= d; i++, j++)
70         arr_sort[i] = t[j];
71 }
```

Fungsi **void merge_array** digunakan untuk melakukan proses sortir data dengan parameter int a, b, c, d. Sistem mendeklarasikan sebuah integer t dengan nilai sebesar 50. **Ketika** nilai i kurang dari sama dengan nilai b dan nilai j kurang dari sama dengan nilai d dan **jika** array [i] kurang dari array ke [j], maka nilai variable **t** bertambah sebanyak 1 dan nilai variable **t** sama dengan nilai array ke [j++] (atau j+1). Berikut adalah full script dan output yang dihasilkan!

```cpp
#include <iostream>

#define size 7

using namespace std;

void sorting_dipecah(int, int);

void array_dipecah(int, int, int, int);

int ahoy_marine[size];

int main() {

    int i;

    cout<<"\nMasukkan "<<size<<" data yang akan di sorting : "<<endl;

    for (i = 0; i < size; i++)

        cin >> ahoy_marine[i];

    cout << "\nData sebelum di sorting: ";

    for (i = 0; i < size; i++) {

        cout << "\t" << ahoy_marine[i];

    }

    sorting_dipecah(0, size - 1);

    cout << "\n\nData setelah di sorting: ";

    for (i = 0; i < size; i++){

        cout << "\t" << ahoy_marine[i];

    }


    return 0;

}
```

```
void sorting_dipecah(int zeya, int zeta) {

    int data;

    if (zeya < zeta) {

        data = (zeya + zeta) / 2;

        sorting_dipecah(zeya, data);

        sorting_dipecah(data + 1, zeta);

        array_dipecah(zeya, data, data + 1, zeya);
    }
}


void array_dipecah(int alfa, int beta, int gama, int sigma) {

    int x[50];

    int zeya = alfa, zeta = gama, ligma = 0;

    while (zeya <= beta && zeta <= sigma) {

        if (ahoy_marine[zeya] < ahoy_marine[zeta])

            x[ligma++] = ahoy_marine[zeya++];

        else

            x[ligma++] = ahoy_marine[zeta++];
    }


    while (zeya <= beta)

        x[ligma++] = ahoy_marine[zeya++];

    while (zeta <= sigma)

        x[ligma++] = ahoy_marine[zeta++];

    for (zeya = alfa, zeta = 0; zeya <= sigma; zeya++, zeta++)

        ahoy_marine[zeya] = x[zeta];
}
```

```
Masukkan 7 data yang akan di sorting :
78
9
47
25
65
14
85

Data sebelum di sorting:        78      9       47      25      65      14      85

Data setelah di sorting:        78      9       47      25      65      14      85
--------------------------------
Process exited after 15.49 seconds with return value 0
Press any key to continue . . .
```

BAB III
CLASS

3.1 DASAR TEORI

Class bisa diartikan sebagai sketsa atau blueprint dari sebuah objek yang akan dibuat. Class memiliki 3 tingkat pengaksesan, yaitu Public, Private, dan Protected. Akses Public berarti semua bagian dari program dapat mengakses semua variable atau rumus yang ada di dalam Public. Akses Private berarti semua bagian dari program tidak dapat mengakses semua variable atau rumus yang ada di dalam Private, kecuali Class itu sendiri.

Akses Protected berarti semua bagian dari program tidak dapat mengakses semua variable atau rumus yang ada di dalam Protected, kecuali Class itu sendiri dan turunan dari Class tersebut (atau biasa disebut anak kelas/adik kelas). Ketiga akses ini sangat penting untuk memastikan bahwa logika aplikasi berjalan sesuai dengan yang diharapkan dan membatasi akses terhadap bagian yang tidak perlu di akses oleh bagian lain dari aplikasi.

3.2 SETTER GETTER

Setter adalah sebuah fungsi atau metode yang dipakai untuk **memberikan** nilai ke dalam sebuah data. Getter adalah fungsi atau metode yang dipakai untuk **menampilkan** nilai data. Perlunya penggunaan setter dan getter diawali dari sebuah saran, bahwasannya semua anggota data dari suatu class harusnya di set sebagai Private dan bukan di set sebagai Public. Jika semua data member di set sebagai private, maka kita tidak bisa memberikan nilai dan mengakses nilai secara langsung.

```
#include <iostream>

using namespace std;

class BigAmbitions {

  private:

    string humanResourceDevelopment;

    double luasGedung;

    string namaKaryawan;
```

Sistem membuat sebuah class dengan nama BigAmbitions dengan bagian Private yang berisikan tipe data string dan double yang masing-masing tipe data memiliki nama.

```
  public:

    void setHumanResourceDevelopment(string bagian1) {

      humanResourceDevelopment = bagian1;

    }

    void setLuasGedung(double bagian2) {

      luasGedung = bagian2;

    }

    void setNamaKaryawan(string bagian3) {

      namaKaryawan = bagian3;

    }

    string getHumanResourceDevelopment() {

      return humanResourceDevelopment;

    }

    double getLuasGedung() {

      return luasGedung;

    }
```

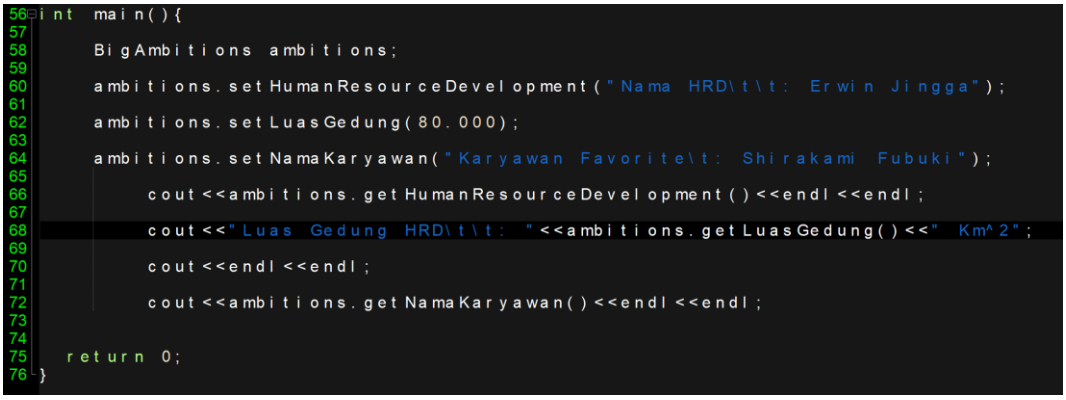
```
      string getNamaKaryawan(){

            return namaKaryawan;

      }

};
```

Karena **string humanResourceDevelopment** berada di Private, user tidak bisa sembarang mengubah. Oleh karena itu, dibuatlah setter pada

baris ke 18 sampai 30 untuk bisa memberikan deklarasi pada variable yang berada di bagian Private. **Void setHumanResourceDevelopment** memiliki parameter **string bagian1**. Pada bagian ini, variable **humanResourceDevelopment** yang berada di Private, dideklarasikan ulang dan membuat variable ini menjadi sama dengan parameter **bagian1**. **Void setLuasGedung** memiliki parameter **double bagian2**.

Pada bagian ini, variable **luasGedung** yang berada di Private, dideklarasikan ulang dan membuat variable ini menjadi sama dengan parameter **bagian2**. **Void setNamaKaryawan** memiliki parameter **string bagian3**. Pada bagian ini, variable **namaKaryawan** yang berada di Private, dideklarasikan ulang dan membuat variable ini menjadi sama dengan parameter **bagian3**.

Sistem membuat getter pada baris ke 36 sampai 48 untuk mengambil variable dari setter yang sudah dibuat. **Return** yang terdapat pada bagian getter, berfungsi untuk mengambil variable pada bagian setter yang sudah di buat. Contohnya mengambil variable **string bagian3** yang nantinya bisa di input oleh user.



Pada bagian utama, nama dari Class ditulis ulang (harus sama persis dengan nama class) yaitu **BigAmbitions** dan sistem akan membuat sebuah objek bernama **ambitions**. Bisa dikatakan bahwa **BigAmbitions** adalah blueprint untuk membuat **ambitions**. Sistem akan mendeklarasikan objek yang dibuat (ambitions) dan mengambil variable dari setter (bisa dilihat pada gambar baris ke 60). Lalu user bisa menginput sebuah kalimat (tidak bisa angka, karena tipe data variable adalah **string**). Pada baris ke 62, sistem mendeklarasikan objek yang dibuat (ambitions) dan mengambil variable

dari setter. Lalu user bisa menginput sebuah angka (harus angka, tidak bisa kalimat atau kata, karena tipe data adalah **double**). Pada baris ke 64, sistem mendeklarasikan objek yang dibuat (ambitions) dan mengambil variable dari setter. Lalu user bisa menginput sebuah kalimat (tidak bisa angka, karena tipe data variable adalah **string**).

Sistem akan menampilkan objek yang sudah selesai dibuat (ambitions) dengan mengambil getter yang variablenya sudah di deklarasikan di bagian setter. Berikut adalah full script dan output yang dihasilkan!

```cpp
#include <iostream>

using namespace std;

class BigAmbitions {

  private:

    string humanResourceDevelopment;

    double luasGedung;

    string namaKaryawan;

  public:

    void setHumanResourceDevelopment(string bagian1) {

      humanResourceDevelopment = bagian1;

    }

    void setLuasGedung(double bagian2) {

      luasGedung = bagian2;

    }

    void setNamaKaryawan(string bagian3) {

      namaKaryawan = bagian3;

    }

    string getHumanResourceDevelopment() {

      return humanResourceDevelopment;

    }

    double getLuasGedung() {

      return luasGedung;

    }

    string getNamaKaryawan(){

        return namaKaryawan;

    }

};
```

```cpp
int main(){

    BigAmbitions ambitions;

    ambitions.setHumanResourceDevelopment("Nama HRD\t\t: Erwin Jingga");

    ambitions.setLuasGedung(80.000);

    ambitions.setNamaKaryawan("Karyawan Favorite\t: Shirakami Fubuki");

        cout<<ambitions.getHumanResourceDevelopment()<<endl<<endl;

        cout<<"Luas Gedung HRD\t\t: "<<ambitions.getLuasGedung()<<" Km^2";

        cout<<endl<<endl;

        cout<<ambitions.getNamaKaryawan()<<endl<<endl;


  return 0;
}
```

```
Nama HRD                : Erwin Jingga

Luas Gedung HRD         : 80 Km^2

Karyawan Favorite       : Shirakami Fubuki


--------------------------------
Process exited after 0.07564 seconds with return value 0
Press any key to continue . . .
```

BAB IV
CLASS 2

4.1 CONSTRUCTOR

Constructor adalah sebuah fungsi khusus yang dijalankan secara otomatis pada saat sebuah object dibuat, yakni saat prosess instansiasi. Constructor biasa dipakai untuk membuat proses awal dalam persiapan object, seperti memberi nilai kepada anggota data, memanggil anggota berdasarkan fungsi internal, dan melakukan beberapa proses lain yang dianggap diperlukan.

Dalam Bahasa C++, biasanya constructor dibuat dengan cara menulis sebuah fungsi yang namanya sama dengan nama class. Sebuah constructor tidak mengembalikan nilai, sehingga tidak perlu menulis tipe data sebelum nama dari sebuah fungsi. Constructor juga harus di set sebagai Public (bukan Private atau Protected), jika constructor tidak di set sebagai Public, maka user tidak bisa menginstansiasi class tersebut. Consturctor hanya boleh ada satu pada setiap class.

```
2
3  #include <iostream>
4
5  using namespace std;
6
7
8  class  Bangunan {
9
10   private:
11
12      double panjang;
13      double lebar;
14      double tinggi;
15
16      int lantai;
17
```

Sistem membuat sebuah class bernama **Bangunan** dengan bagian Private dideklarasikan dengan **double** dan **int** pada masing-masing nama variable.

```
17
18   public:
19
20     Bangunan():panjang{1745.6}, lebar{1249.2}, tinggi{542.42}, lantai{80}
21
```

Sistem membuat Public pada class, lalu membuat sebuah fungsi **constructor** yang bernama sama persis dengan nama class, yaitu **Bangunan**. Sistem mendeklarasikan variable panjang, lebar, tinggi, dan lantai pada constructor yang setiap variablenya diberikan sebuah nilai tertentu. **Bangunan( ) : panjang { 1745.6 }** berarti variable panjang memiliki nilai sebesar 1745,6. Pada program, tanda titik dibaca dengan tanda koma.

```
21
22          cout<<"Loading...."<<endl;
23
24          cout<<"System sedang membuat bangunan....";
25
26          cout<<endl<<endl;
27
28          cout<<"Bagunan telah selesai dibuat!"<<endl;
29
30          cout<<"Deskripsi bangunan:"<<endl;
31
32          cout<<"Panjang\t: "<<panjang<<endl;
33
34          cout<<"Lebar\t: "<<lebar<<endl;
35
36          cout<<"Tinggi\t: "<<tinggi<<endl;
37
38          cout<<"Lantai\t: "<<lantai<<endl;
39
40          }
41
42
43 };
44
```

Sistem mendeklarasikan tulisan yang harus ditampilkan pada program, dan sistem memanggil variable panjang, lebar, tinggi, dan lantai yang sudah dideklarasikan di bagian Public.

```
45 int  main(){
46
47
48      Bangunan fightArea;
49
50
51
52 return 0;
53
54 }
```

Pada bagian utama, sistem menuliskan class dan membuat sebuah objek bernama **fightArea**, sangat disayangkan bahwa sistem tidak membuat

```
public:

    Bangunan():
```

sebuah variable untuk object tersebut (setter) dan sistem secara otomatis tidak bisa memanggil variable untuk object tersebut (getter). Seharusnya program tidak akan menampilkan apapun (kosong). Tetapi, karena adanya **constructor** yang sudah dibuat di bagian Public, program akan secara otomatis

menjalankan constructor, walaupun pada bagian utama tidak ada perintah untuk menampilkan tulisan. Berikut adalah full script dan output yang dihasilkan!

```cpp

#include <iostream>

using namespace std;


class  Bangunan {

   private:

      double panjang;
      double lebar;
      double tinggi;

      int lantai;

   public:

      Bangunan():panjang{1745.6}, lebar{1249.2}, tinggi{542.42}, lantai{80} {

         cout<<"Loading...."<<endl;

         cout<<"System sedang membuat bangunan....";

         cout<<endl<<endl;

         cout<<"Bagunan telah selesai dibuat!"<<endl;

         cout<<"Deskripsi bangunan:"<<endl;

         cout<<"Panjang\t: "<<panjang<<endl;

         cout<<"Lebar\t: "<<lebar<<endl;

         cout<<"Tinggi\t: "<<tinggi<<endl;

         cout<<"Lantai\t: "<<lantai<<endl;

         }


};
int main(){

      Bangunan fightArea;


return 0;
}
```

```
Loading....
System sedang membuat bangunan....

Bagunan telah selesai dibuat!
Deskripsi bangunan:
Panjang : 1745.6
Lebar   : 1249.2
Tinggi  : 542.42
Lantai  : 80

--------------------------------
Process exited after 0.01736 seconds with return value 0
Press any key to continue . . .
```

4.2 DESTRUCTOR

Destructor adalah sebuah fungsi khusus yang akan dijalankan secara otomatis (sama seperti constructor), tetapi destructor akan dijalankan ketika sebuah object dihapus. Dalam pemrograman C++, sebenarnya seluruh objek sudah otomatis dihapus oleh sistem, ketika sebuah script selesai diproses. Oleh karena itu, destructor relatif jarang digunakan pada pembuatan program atau aplikasi.

Kasus yang memerlukan destructor adalah kasus jika sebuah kode pada program yang dibuat, mengakses memory pada komputer yang secara langsung menggunakan pointer, maka destructor bisa dipakai untuk "melepaskan" ruang memory tersebut, supaya tidak terjadi memory leak. Memory leak adalah istilah programming, yang pada setiap aplikasinya terus menggunakan RAM terlalu besar dan terus membesar sepanjang program berjalan. Memory leak adalah *bug* yang harus dihindari, namun selama program dibuat tidak menggunakan pointer, seharusnya tidak akan ada masalah.

```
45    ~Bangunan(){
46
47        cout<<endl<<endl<<"Sistem sedang membersihkan memori, harap menunggu!";
48
49
50    }
51
```

Destructor dibuat menggunakan fungsi khusus yang sama persis dengan nama class (sama seperti constructor), namun destructor diawali dengan tanda *tilde* "~" yang hanya bisa terdapat satu destructor saja pada setiap class.

Sistem membuat destructor yang akan langsung dijalankan Ketika program selesai dijalankan. Walaupun pada bagian utama adalah kosong, karena tidak ada deklarasi untuk menampilkan tulisan apapun, namun destructor akan tetap dijalankan setelah program selesai dijalankan.

```
55 int main(){
56
57
58      Bangunan fightArea;
59
60
61 return 0;
62
63 }
```

```
Sistem sedang membersihkan memori, harap menunggu!
--------------------------------
Process exited after 0.02026 seconds with return value 0
Press any key to continue . . .
```

Berikut adalah full script dan output yang dihasilkan!

```cpp
#include <iostream>

using namespace std;


class  Bangunan {

  private:

     double panjang;
     double lebar;
     double tinggi;

     int lantai;

     string nama;
     string karyawan;

  public:

     Bangunan():panjang{1745.6}, lebar{1249.2}, tinggi{542.42}, lantai{80} {

        cout<<"Loading..."<<endl;

        cout<<"System sedang membuat bangunan....";

        cout<<endl<<endl;

        cout<<"Bagunan telah selesai dibuat!"<<endl;

        cout<<"Deskripsi bangunan:"<<endl;

        cout<<"Panjang\t: "<<panjang<<endl;

        cout<<"Lebar\t: "<<lebar<<endl;

        cout<<"Tinggi\t: "<<tinggi<<endl;

        cout<<"Lantai\t: "<<lantai<<endl;

        }

     ~Bangunan(){

         cout<<endl<<endl<<"Sistem sedang membersihkan memori, harap menunggu!";


     }


};

int main(){


     Bangunan fightArea;


return 0;
}
```

```
Loading....
System sedang membuat bangunan....

Bagunan telah selesai dibuat!
Deskripsi bangunan:
Panjang : 1745.6
Lebar   : 1249.2
Tinggi  : 542.42
Lantai  : 80



Sistem sedang membersihkan memori, harap menunggu!
--------------------------------
Process exited after 0.01948 seconds with return value 0
Press any key to continue . . .
```

BAB V
OBJECT ORIENTED PROGRAMMING

5.1 DASAR TEORI

Pemrograman Berorientasi Objek atau OOP merupakan sebuah pemrograman yang memfokuskan pada banyak objek dan interaksinya dalam menyelesaikan suatu masalah. Pada OOP sendiri, segala sesuatu dianggap sebagai objek, dengan setiap objeknya memiliki atribut dan metode.

Pada konsep OOP, program harus memiliki karakteristik, baik itu karakteristik yang berupa pewarisan sifat (*Inheritance*) atau sifat turunan, karakteristik yang berupa sifat perilaku (*Polymorphism*), ataupun karakteristik yang berupa pembungkusan sifat dari objek yang berbeda (*Encapsulation*).

5.2 ENCAPSULATION

5.2.1 Pengertian Encapsulation

Encapsulation adalah salah satu prisip dasar dalam konsep OOP (Object Oriented Programming) yang bertujuan untuk membungkus data dalam satu unit. Encapsulation biasanya digunakan untuk menyembunyikan detail internal implementasi object dari dunia luar dan menyediakan interface yang dapat berinteraksi dengan object tersebut.

Encapsulation dapat dilakukan dengan mendeklarasikan semua variable di suatu kelas sebagai Private dan menulis metode Public di dalam kelas tersebut untuk mendapatkan dann mengatur nilai dari variable.

5.2.2 Manfaat Encapsulation

Encapsulation sendiri memiliki beberapa manfaat yang mempermudah user dalam membuat sebuah program, antara lain:

1. Menyembunyikan data

   Encapsulation dapat menyembunyikan detail implementasi internal suatu kelas. Hal ini berarti bahwa pengguna kelas tidak perlu mengetahui bagaimana data disimpan dalam variabelnya. Sebaliknya, pengguna berinteraksi dengan kelas melalui metode publik yang dirancang dengan baik, seperti setter dan getter. Dengan demikian, keamanan dapat meningkat dan mencegah manipulasi langsung terhadap data sensitif.

2. Memberi kebebasan bagi programmer

   Enkapsulasi memberdayakan programmer dengan memberikan kebebasan untuk mengubah detail implementasi internal suatu kelas tanpa memengaruhi interface eksternal yang digunakan oleh bagian lain dari program.

Sebagai contoh, seorang GUI programmer awalnya mengimplementasikan operasi cut-and-paste menggunakan gambar layar. Kemudian, jika dia ingin beralih ke sistem penyimpanan berbasis teks yang lebih efisien, dia dapat menggunakan enkapsulasi untuk memastikan bahwa kode lain yang berinteraksi dengan GUI tidak akan terpengaruh.

3. Meningkatkan fleksibilitas

   Dengan enkapsulasi, programmer dapat mengontrol akses ke elemen pada kelas dan menyesuaikan tingkat visibilitas sesuai dengan kebutuhan. Hal ini memungkinkan mereka untuk membuat kelas yang menerapkan aturan khusus untuk mengakses dan memodifikasi data.

4. Dapat digunakan Kembali

   Encapsulation mendukung penggunaan kembali karena detail implementasi internal suatu kelas disembunyikan dari entitas eksternal. Saat kelas yang dienkapsulasi digunakan di berbagai bagian program atau di proyek lain, perubahan pada penerapan kelas kemungkinan besar tidak akan memengaruhi basis kode lainnya.

5. Mudah dalam pengujian kode

   Kode yang dienkapsulasi mudah untuk diuji, terutama selama pengujian unit. Karena kelas memiliki interface yang jelas, fungsionalitas setiap metode dapat diuji secara terpisah dengan lebih mudah.

5.2.3 Kekurangan Encapsulation

Encapsulation sendiri memiliki beberapa kekurangan yang pada dasarnya akan menyulitkan programmer dalam membuat sebuah program, antara lain:

1. Ukuran kode yang meningkat

   Enkapsulasi dapat menyebabkan peningkatan ukuran kode karena kebutuhan akan metode tambahan dengan access modifier. Sebagai akibatnya, file sumber menjadi

lebih besar dan basis kode lebih kompleks serta berpotensi lebih sulit untuk dikelola.

2. Lebih banyak instruksi

   Karena enkapsulasi menggunakan lebih banyak metode untuk mengakses dan memodifikasi elemen data, instruksi tambahan diperlukan dalam kode. Hal ini dapat meningkatkan kerumitan karena setiap metode memerlukan rangkaian instruksinya sendiri.

3. Waktu eksekusi lebih lama

   Instruksi yang ditambahkan karena enkapsulasi dapat menghasilkan waktu eksekusi yang lebih lama untuk suatu program, yang berpotensi memengaruhi kinerja dalam beberapa kasus.

5.2.4 Contoh Encapsulation

```
3  #include <iostream>
4
5  using namespace std;
6
7  class Karyawan {
8
9  private:
10
11      int karyaId;
12
13      string karyaNama;
14
15      float karyaNomor;
16
```

Sistem mendeklarasikan class bernama **Karyawan** dan memberikan Private dengan beberapa tipe data.

```
public:

    void setKaryaId(int id) {

        karyaId = id;

    }

    void setKaryaNama(string nama) {

        karyaNama = nama

    }

    void setKaryaNomorfloat nomor) {

        karyaNomor = nomor;

    }

    int getKaryaId() {

        return karyaId;

    }

    string getKaryaNama()

        return karyaNama;

    }

    float getKaryaNomor() {

        return karyaNomor

    }
};
```

Sistem mendeklarasikan Public dengan implementasi dari setter dan getter.

```
int main() {

    Karyawan fassbear;

    fassbear.setKaryaId(187);
    fassbear.setKaryaNama("Chloe Pawapua");
    fassbear.setKaryaNomor(8977.25);

    cout<<"Nama Karyawan\t: "<<fassbear.getKaryaNama()<<end
    cout<<"Karyawan ID\t: "<<fassbear.getKaryaId()<<endl;
    cout<<"Karyawan Nomor\t: "<<fassbear.getKaryaNomor();


return 0;

}
```

Pada bagian utama, sistem akan memberikan deklarasi tulisan kepada variable yang sudah ditempatkan pada setter. Lalu sistem akan memanggil variable yang sudah di set dengan menggunakan getter.

Ini adalah contoh dari Encapsulation, mungkin beberapa orang akan berfikir bahwa apa perbedaan antara encapsulation dengan setter, getter, construction, dan destruction? Jawabannya, tidak ada perbedaan, karena encapsulation berarti membungkus yang mempermudah programmer dalam menentukan akses dan fungsi dari anggota-anggota dan dapat dengan mudah untuk memodifikasinya. Oleh karena itu, di dalam encapsulation sebuah kelas, pasti terdapat setter, getter, constructor, atau destructor. Berikut ini adalah full script dan output yang dihasilkan!

```
3  #include <iostream>
4
5  using namespace std;
6
7  class Karyawan {
8
9  private:
10
11      int karyaId;
12
13      string karyaNama;
14
15      float karyaNomor;
16
17 public:
18
19      void setKaryaId(int id) {
20
21          karyaId = id;
22
23      }
```

```
25      void setKaryaNama(string nama) {
26
27          karyaNama = nama;
28
29      }
30
31      void setKaryaNomor(float nomor) {
32
33          karyaNomor = nomor;
34
35      }
36
37      int getKaryaId() {
38
39          return karyaId;
40
41      }
42
43      string getKaryaNama() {
44
45          return karyaNama;
46
47      }
48
49      float getKaryaNomor() {
50
51          return karyaNomor;
52
53      }
54  };
55
56  int main() {
57
58      Karyawan fassbear;
59
60      fassbear.setKaryaId(187);
61      fassbear.setKaryaNama("Chloe Pawapua");
62      fassbear.setKaryaNomor(8977.25);
63
64      cout<<"Nama Karyawan\t: "<<fassbear.getKaryaNama()<<endl;
65      cout<<"Karyawan ID\t: "<<fassbear.getKaryaId()<<endl;
66      cout<<"Karyawan Nomor\t: "<<fassbear.getKaryaNomor();
67
```

```
Nama Karyawan   : Chloe Pawapua
Karyawan ID     : 187
Karyawan Nomor  : 8977.25
--------------------------------
Process exited after 0.06149 seconds with return value 0
Press any key to continue . . .
```

5.3 INHERITANCE

Sebuah konsep pemrograman yang membuat sebuah class dapat menurunkan property dan metode yang dimilikinya kepada kelas lain atau kelas di bawahnya, disebut dengan Inheritance. Konsep ini digunakan untuk memanfaatkan fitur *code reuse* (penggunaan kemabli kode) untuk menghindari duplikasi kode program.

Dengan Inheritance, user dapat menggunakan ‘property’ dan ‘metode’ yang dimiliki oleh kelas induk (kakak kelas) tanpa harus mendefinisikan ulang di kelas lainnya. Hal ini akan berdampak pada efisiensi dan struktur kode program yang lebih rapih.

```
#include <iostream>

using namespace std;

 class Utama {

    public:

    string nama = "Kanaki Sayu";

 };
```

Sistem membuat sebuah class dengan nama **Utama**, pada konteks ini menunjukan bahwa kelas ini adalah kelas induk. Terdapat deklarasi variable **nama** pada bagian Public.

```
    class Turunan : public Utama {

    public:

    double gaji = 845000;

    };
```

Sistem membuat class dengan nama **Turunan : public Utama** yang berarti kelas dengan nama **Turunan** adalah *Inheritance* (turunan) dari kelas **Utama**. Pada kelas **Turunan**, kelas ini dapat mengakses variable **nama** yang berada di kelas **Utama**, karena kelas ini merupakan sebuah anak kelas dari induk kelas. Sistem menambahkan variable **gaji** dengan tipe data double.

```
int main(void) {

    Turunan vektor;

        cout<<"Nama Karyawan\t: "<<vektor.nama<<endl;
        cout<<"Gaji Karyawan\t: "<<vektor.gaji<<endl;


    return 0;
}
```

Pada bagian utama, dideklarasikan parameter **void** yang mengarah kepada fungsi **void** untuk semua bagian yang berada pada **int main**. Sistem membuat sebuah object bernama **vector** dan merupakan object dari kelas **Turunan**. Sistem memanggil variable **nama** dan variable **gaji** untuk dijadikan sebagai isi dari object **vector**. Variable dapat dipanggil karena

adanya parameter **void** yang sudah dideklarasikan pada **int main**. Itu adalah contoh dari turunan atau inheritance pada sebuah program. Berikut adalah full script dan output yang dihasilkan!

```
#include <iostream>

using namespace std;

 class Utama {

    public:

    string nama = "Kanaki Sayu";

 };

    class Turunan : public Utama {

    public:

    double gaji = 845000;

    };
int  main(void) {

     Turunan vektor;

          cout<<"Nama Karyawan\t: "<<vektor.nama<<endl;
          cout<<"Gaji Karyawan\t: "<<vektor.gaji<<endl;


     return 0;
}
```

```
Nama Karyawan    : Kanaki Sayu
Gaji Karyawan    : 845000

--------------------------------
Process exited after 0.09225 seconds with return value 0
Press any key to continue . . .
```

5.4 POLYMORPHISM

Polymorphism adalah kemampuan pada sebuah fungsi yang terdapat pada anggota dari sebuah class, untuk memberikan seolah-olah fungsi tersebut memiliki berbagai bentuk yang sesuai berdasarkan kelas masing-masing. ketika menggunakan Polymorphism, jika user mencoba untuk memanggil sebuah anggota dari fungsi, maka program akan merespon dengan anggota dari fungsi yang sesuai berdasarkan yang ada pada object dan class yang digunakan.

Polymorphism terjadi ketika program memiliki banyak class yang memiliki relasi satu sama lain menggunakan *Inheritance*. Polymorphism memiliki beberapa konsep seperti Overloading, konsep Operator Overloading, konsep Templeates, konsep Overriding dan sebagainya merupakan implementasi pada Polymorphism.

```
#include <iostream>

using namespace std;

class Mode {

    public:

        virtual void tingkat(){

            cout<<"Assisted, Normal, Hardcore, Veteran, Extreme, No Hope";

    }

};
```

Sistem membuat class dengan nama **Mode** dan pada bagian Public, dideklarasikan fungsi **virtual** dan fungsi **void** yang digabung menjadi satu. Fungsi **virtual** digunakan untuk menentukan ulang fungsi pada anggota-anggota di class yang berbeda, dengan nama fungsi yang sama. Sistem dapat memanggil fungsi **virtual** untuk object tertentu dan menjalankan versi fungsi kelas turunan. Sistem membuat fungsi yang bernama **virtual void tingkat( )** yang berisi tulisan Assisted, Normal, Hardcore, Veteran, Extreme, No Hope.

```
class Pilihan: public Mode{

    public:

        int pilih;

        void tingkat(){

            cout<<"\nTingkat Kesulitan: "<<endl;

            cout<<"[1] Assisted";
            cout<<"\n[2] Normal";
            cout<<"\n[3] Hardcore";
            cout<<"\n[4] Veteran";
            cout<<"\n[5] Extreme";
            cout<<"\n[6] No Hope";

            cout<<"\n\nPilihan kamu: ";

            cin>>pilih;
```

Sistem membuat kelas turunan dari kelas **Mode** yang bernama class **Pilihan** dan Public yang berisi tipe data int dan juga fungsi **void**. Fungsi **void** yang digunakan pada kelas **Pilihan** adalah fungsi yang sama dengan fungsi **virtual void** yang berada di kelas **Mode**. Program akan melakukan

Overriding pada fungsi **virtual void** yang terdapat pada class **Mode**, sehingga isi dari **virtual void** yang berada di class **Mode** adalah isi dari fungsi **void** yang berada di class **Pilihan**.

```
39                      if(pilih==1){
40
41                          goto Assisted;
42
43                      }
44
45                      if(pilih==2){
46
47                          goto Normal;
48
49                      }
50
51                          Assisted:
52
53                              cout<<"H";
54
55                          Normal:
56
57                              cout<<"K";
58
59          }
60
61  };
```

Jika pilihan user adalah 1, maka program akan membawa (teleport) user ke bagian Assisted pada program. Jika pilihan user adalah 2, maka program akan membawa (teleport) user ke bagian Normal pada program. Hal ini berlaku untuk pilihan yang lainnya. Sangat disayangkan, pada program kali ini sistem tidak membuat bagian lainnya, sistem hanya membuat bagian Assisted dan Normal saja, sehingga jika user memilih pilihan yang bukan 1 atau 2, maka program akan langsung selesai.

```
63  int main(void) {
64
65      Pilihan player;
66
67      player.tingkat();
68
69
70  return 0;
71
72  }
```

Pada bagian utama, sistem mendeklarasikan class yang sudah di buat, yaitu class **Pilihan** dan sistem membuat sebuah object yang bernama **player**. Kemudian, sistem mengisi object tersebut dengan fungsi yang berada di class **Pilihan**, yaitu fungsi **virtual void tingkat ( )**. Berikut adalah full script dan output yang dihasilkan!

```
#include <iostream>

using namespace std;

class Mode {

    public:

        virtual void tingkat(){

            cout<<"Assisted, Normal, Hardcore, Veteran, Extreme, No Hope";

    }

};

class Pilihan: public Mode{

    public:

        int pilih;

        void tingkat(){

            cout<<"\nTingkat Kesulitan: "<<endl;

            cout<<"[1] Assisted";
            cout<<"\n[2] Normal";
            cout<<"\n[3] Hardcore";
            cout<<"\n[4] Veteran";
            cout<<"\n[5] Extreme";
            cout<<"\n[6] No Hope";

            cout<<"\n\nPilihan kamu: ";

            cin>>pilih;

                if(pilih==1){

                    goto Assisted;

                }

                if(pilih==2){

                    goto Normal;

                }

                    Assisted:

                        cout<<"H";

                    Normal:

                        cout<<"K";

    }
```

```
63  int main(void) {
64
65      Pilihan player;
66
67      player.tingkat();
68
69
70  return 0;
71
72  }
```

```
Tingkat Kesulitan:
[1] Assisted
[2] Normal
[3] Hardcore
[4] Veteran
[5] Extreme
[6] No Hope

Pilihan kamu: 2
K
--------------------------------
Process exited after 2.133 seconds with return value 0
Press any key to continue . . .
```

BAB VI
SEARCHING ALGORITHM

6.1 DASAR TEORI

Searching adalah sebuah algoritma pemrograma yang menerima sebuah argumen untuk mencari lokasi dari sebuah data pada kumpulan data yang telah ada. Proses pencarian (searching) menghasilkan dua kemungkinan, yaitu kemugkinan data yang dicari ditemukan (success) atau kemungkinan data yang dicari tidak ditemukan (fail).

Searching memiliki dua jenis teknik pencarian, yaitu *Sequential Search Technic* dan *Binary Search Technic*. Pencarian dengan teknik sekuensial akan dilakukan bila sebuah data tidak berurutan dan akan dicari secara langsung dan bisa juga ketika data sudah terurut; Sedangkan pencarian dengan teknik biner akan dilakukan bila sebuah data sudah terurut rapih.

Teknik pencarian biner juga bisa dilakukan pada sebuah data yang tidak berurutan, tetapi dengan syarat bahwa data yang tidak terurut harus di sorting terlebih dahulu untuk mengurutkan data secara ascending maupun descending dengan menggunakan beberapa metode sorting seperti bubble sorting, insertion sorting, selection sorting, quick sorting, atau merge sorting, barulah setelah terurut data bisa dicari dengan teknik pencarian biner. Intinya, jika data sudah terurut rapih, bisa menggunakan sekuensial ataupun biner searching untuk mencari data. Tetapi, khusus untuk data yang belum terurut, hanya bisa dicari dengan teknik sekuensial.

6.2 SEQUENTIAL SEARCH

Algoritma pencarian sekuensial adalah salah satu algoritma pencarian data yang biasa digunakan untuk data yang berpola acak atau belum terurut dan bisa digunakan juga untuk data yang sudah terurut. Algoritma ini akan mencari data sesuai kata kunci yang diberikan mulai dari elemen awal pada array hingga elemen akhir array. Kemungkinan terbaik (best case) ketika menggunakan algoritma ini adalah jika data yang dicari terletak di indeks awal array sehingga hanya membutuhkan sedikit waktu pencarian. Sedangkan kemungkinan terburuknya (worst case) adalah jika data yang dicari ternyata terletak dibagian akhir dari array sehingga pencarian data akan memakan waktu yang lama. Algoritma pencarian sekuensial memiliki beberapa konsep, antara lain:

- ♣ Membandingkan setiap element pada array satu per satu secara berurutan;
- ♣ Proses pencarian dimulai dari indeks pertama hingga indeks terakhir;

- ♣ Proses pencarian akan berhenti apabila data ditemukan;
- ♣ Jika hingga akhir array data masih juga tidak ditemukan, maka proses pencarian akan tetap dihentikan;
- ♣ Proses perulangan pada pencarian akan terjadi sebanyak jumlah N element pada array tersebut.

```
#include <iostream>

using namespace std;

void array(){

    int ahoy[7]={478,574,698,985,1245,7485,9856};

        cout<<"Data terurut\t\t\t\t 478,574,698,985,1245,7485,9856";

}
```

Sistem membuat fungsi **void array( )** yang di dalamnya terdapat array bernama **ahoy** yang memiliki 7 element data. Sistem menampilkan tulisan “Data terurut” yang bisa dilihat oleh user ketika menjalankan program.

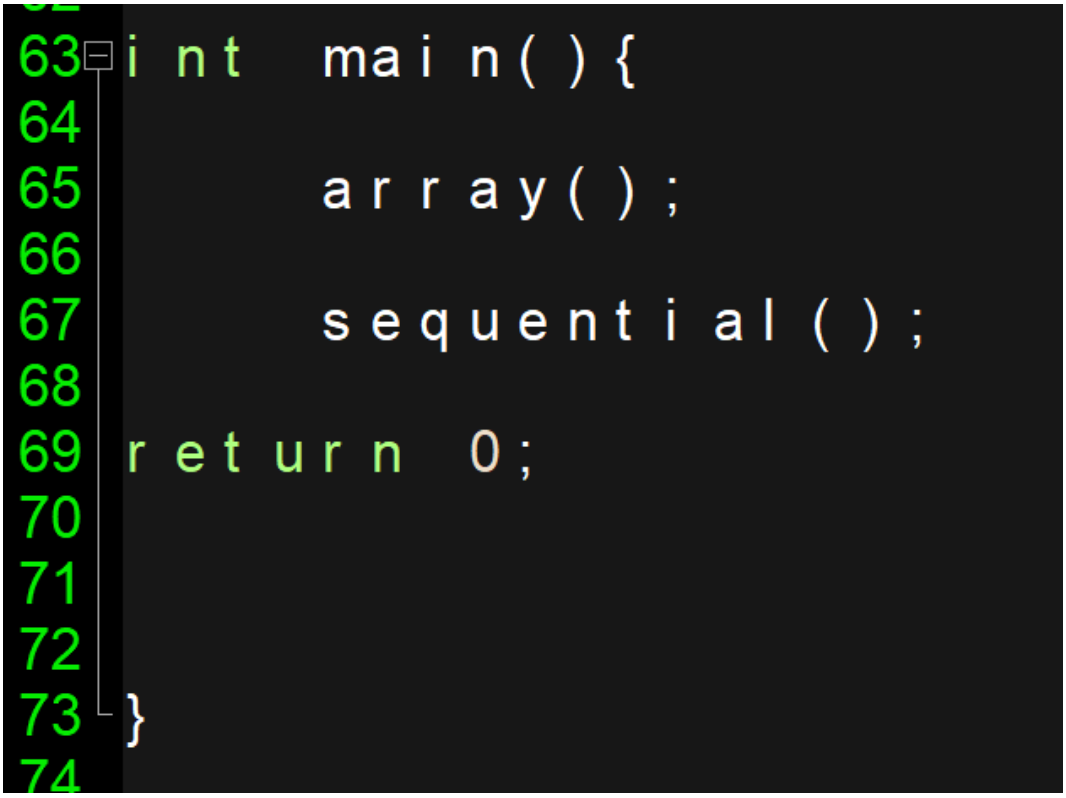
```
void sequential(){

    int ahoy[7]={478,574,698,985,1245,7485,9856};

    int search;

        cout<<endl<<endl;

        cout<<"Masukkan angka yang mau di searching\t: ";

        cin>>search;
```

Sistem membuat fungsi **void sequential( )** yang berisikan array bernama **ahoy** yang memiliki 7 element data. Dideklarasikan variable **search** dengan tipe integer. Lalu sistem membuat tampilan tulisan yang bisa diketikan oleh user untuk mencari angka yang mau dicari. Misalkan user mengetikkan angka 7485 untuk dicari nanti oleh program.

```
    int i; int situasi;

        i=0;

        situasi=0;

            while(i<7){

                if(search==ahoy[i]){

                    situasi=1;

                    break;

                }

                i++;

            }

            if(situasi==0){

                cout<<"\n\nAngka yang dicari tidak ditemukan!";

            }

            else{

                cout<<"\n\nAngka yang kamu cari ada di index ke "<<i;

            }

}
```

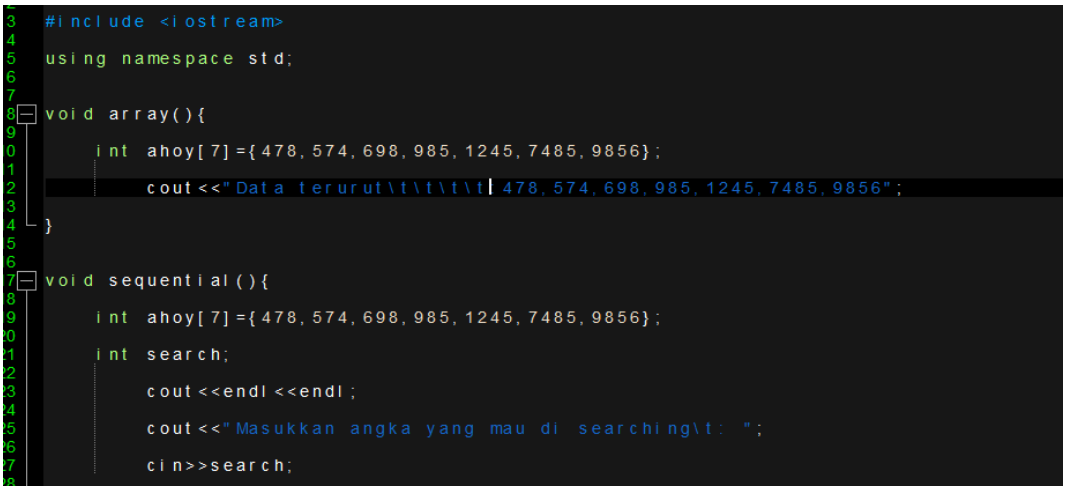
Sistem mendeklarasikan variable **i** dan **situasi** yang pada saat ini nilai dari variable **i** dan **situasi** adalah 0. **Ketika i** kurang dari 7, program akan melakukan looping sebanyak 6 kali, karena **i** dimulai dari 0. **Jika search** sama dengan array **ahoy** ke [i], maka nilai variable **situasi** yang tadinya 0 akan diubah menjadi 1 dan looping akan dihentikan secara paksa, karena adanya perintah **break**. **Jika search** tidak sama dengan array **ahoy** ke [i], maka variable **i** akan bertambah sebanyak 1 di setiap looping sampai maximum nilai variable **i** adalah 6. Setelah looping dihentikan secara paksa oleh perintah **break**, sistem membuat perandaian.

Jika nilai dari variable **situasi** masih sama dengan 0, maka tampilkan tulisan seperti di gambar. Hal ini bisa terjadi apabila user menginput angka yang tidak terdapat di dalam data array **ahoy**, contohnya user menginput angka 20. **Jika** nilai dari variable **situasi** sama dengan 1, maka tampilkan tulisan “Angka yang kamu cari [search] ditemukan pada index ke [i].”



```
63  int  main(){
64
65      array();
66
67      sequential();
68
69  return 0;
70
71
72
73  }
74
```

Pada bagian utama, sistem memanggil fungsi **void array** dan **void sequential**. Fungsi void array dipanggil untuk menampilkan data terurut dari array. Fungsi void sequential dipanggil untuk menampilkan data yang mau di searching dan terdapat proses pencarian data. Berikut adalah full script dan output yang dihasilkan!



```
#include <iostream>

using namespace std;


void array(){
    int ahoy[7]={478,574,698,985,1245,7485,9856};
        cout<<"Data terurut\t\t\t\t 478,574,698,985,1245,7485,9856";
}


void sequential(){
    int ahoy[7]={478,574,698,985,1245,7485,9856};
    int search;
        cout<<endl<<endl;
        cout<<"Masukkan angka yang mau di searching\t: ";
        cin>>search;
```

```
    int i; int situasi;

        i=0;

        situasi=0;

            while(i<7){

                if(search==ahoy[i]){

                    situasi=1;

                    break;

                }

                i++;

            }

            if(situasi==0){

                cout<<"\n\nAngka yang dicari tidak ditemukan!";

            }

            else{

                cout<<"\n\nAngka yang kamu cari ada di index ke "<<i;

            }

}

int main(){

    array();

    sequential();

return 0;


}
```

```
Data terurut                            :478,574,698,985,1245,7485,9856

Masukkan angka yang mau di searching    : 7485


Angka yang kamu cari ada di index ke 5
--------------------------------
Process exited after 7.235 seconds with return value 0
Press any key to continue . . .
```

6.3 BINARY SEARCHING

Binary Search adalah algoritma pencarian yang memiliki cara kerja yang berbeda dari algoritma pencarian sekuensial. Algoritma pencarian biner mencari data dengan cara membagi array menjadi dua bagian ; Sedangkan algoritma pencarian sekuensial mencari data dengan cara data tersebut dicek satu persatu mulai dari element pertama hingga akhir.

Algoritma pencarian biner hanya bisa digunakan untuk mencari data yang sudah terurut, tidak bisa mencari data yang belum terurut. Berbeda

dengan algoritma pencarian sekuensial yang bisa mencari data baik itu data yang masih acak ataupun data yang sudah terurut.

```
#include <iostream>

using namespace std;

int searching(int ahoy[], int p, int r, int search) {
    if (p <= r) {
        int mid = (p + r)/2;
        if (ahoy[mid] == search)
            return mid;
        if (ahoy[mid] > search)
            return searching(ahoy, p, mid-1, search);
        if (ahoy[mid] < search)
            return searching(ahoy, mid+1, r, search);
    }
    return -1;
}
```

Sistem membuat fungsi rekursif **int searching** dengan parameter **ahoy, p, r,** dan **search**. **Jika** variable **p** lebih kecil dari sama dengan variable **r**, maka nilai variable **mid** adalah variable **p** ditambah variable **r**, lalu dibagi dengan dua. Misalkan nilai **p** adalah 0 dan nilai **r** adalah 14, maka (0 + 14) adalah 14, lalu 14 dibagi dengan 2 yang hasilnya adalah 7. Angka 7 akan dijadikan nilai untuk variable **mid**.

Jika array **ahoy** ke [mid] (dalam kontkes ini berarti array ke [7]) sama dengan nilai **search**, lakukan perintah **return** ke variable **mid**. Perintah **return** digunakan untuk mengembalikan suatu nilai atau variable. **Jika** array **ahoy** ke [mid] (dalam kontkes ini berarti array ke [7]) lebih besar dari nilai **search**, maka proses pencarian akan dilakukan ke element bagian kiri array dari posisi element **mid**. Berarti array akan mencari dari 0 sampai 6. **Jika** array **ahoy** ke [mid] (dalam kontkes ini berarti array ke [7]) lebih kecil dari nilai **search**, maka proses pencarian akan dilakukan ke element bagian kanan array dari posisi element **mid**. Berarti array akan mencari dari 8 sampai 14.

Sistem memberikan perintah **return -1** yang berarti, jika program keluar dari looping **if**, maka variable **index** akan bernilai -1, dan jika

program tidak keluar dari looping **if** (masih mencari data), maka variable **index** bernilai sesuai dengan hasil searching.

```
int main(void) {

    int ahoy[] = {20, 56, 78, 98, 187, 689, 951};

    int size = sizeof(ahoy)/sizeof(ahoy[0]);

    int search;

      cout<<"Masukan angka yang mau dicari: ";

      cin>>search;
```

Pada bagian utama, sistem membuat array dengan nama **ahoy** yang berisikan 7 element, sehingga ukuran dari array adalah 7. Variable **search** akan diinputkan oleh user untuk mencari angka yang mau dicari ketika menjalankan program.

```
    cout<<endl<<endl;

    int index = searching(ahoy, 0, size-1, search);

    if(index==-1){

        cout<<"Angka "<<search<<" tidak ada pada data!";

    }

    else{

        cout<<"Angka "<<search<<" berada di index ke "<<index;

    }

return 0;
}
```

Sistem mendeklarasikan bahwa nilai dari variable **index** saat ini adalah nilai dari hasil searching yang sudah dideklarasikan pada fungsi rekursif **int searching**. Apabila nilai **index** adalah -1, maka tampilkan tulisan seperti pada gambar. Hal ini bisa terjadi apabila user menginputkan angka yang tidak terdapat pada data di dalam array **ahoy**. Misalkan user menginput angka 14, maka otomatis nilai **index** menjadi -1. **Jika** nilai **index** tidak sama dengan -1, maka tampilakan tulisan “Angka yang kamu cari [searching] berada pada index ke [index saat ini].” Berikut ini adalah full script dan output yang dihasilkan!

```
#include <iostream>

using namespace std;
int searching(int ahoy[], int p, int r, int search) {
    if (p <= r) {
        int mid = (p + r)/2;
        if (ahoy[mid] == search)
            return mid;
        if (ahoy[mid] > search)
            return searching(ahoy, p, mid-1, search);
        if (ahoy[mid] < search)
            return searching(ahoy, mid+1, r, search);
    }
    return -1;
}

int main(void) {
    int ahoy[] = {20,56,78,98,187,689,951};
    int size = sizeof(ahoy)/sizeof(ahoy[0]);
    int search;
     cout<<"Masukan angka yang mau dicari: ";
     cin>>search;
     cout<<endl<<endl;
    int index = searching(ahoy, 0, size-1, search);
    if(index==-1){
        cout<<"Angka "<<search<<" tidak ada pada data!";
    }
    else{
        cout<<"Angka "<<search<<" berada di index ke "<<index;
    }
return 0;
}
```

```
Masukan angka yang mau dicari: 951


Angka 951 berada di index ke 6
--------------------------------
Process exited after 8.895 seconds with return value 0
Press any key to continue . . .
```