Nama : Muhammad Abel Jollando

NIM : 2005551073

Kelompok : 12

|  |
| --- |
| MODUL III ARRAY DAN POINTER |

Tujuan

1. Mengetahui konsep dari *array,* cara mendeklarasi *array* dalam C/C++ serta tipe data yang mungkin digunakan dalam *array* dan contohnya beserta dengan keperluan memorinya.
2. Mengetahui konsep dari *pointer* beserta cara pendeklarasiannya dan cara penggunaannya.
3. Mengetahui secara terperinci mengenai perbedaan antara *array* dan *pointer*.
4. Memahami cara untuk membangkitkan bilangan *random* dalam C/C++.
5. Mengetahui apa itu metode *sorting* dan *searching.*
6. Mengetahui mekanisme metode pengurutan dan pencarian data menggunakan berbagai macam metode yang tersedia.
7. Dapat membuat *flowchart* dan *pseudocode* (notasi algoritma) dari masing-masing metode yang ada serta dapat membuat *tracing* secara detail.

Tugas Pendahuluan

1. Jelaskan tentang *array* dan deklarasi *array* dalam C/C++ serta tipe data yang mungkin digunakan dalam *array*. Berikan contoh deklarasi *array* dalam C/C++ lalu hitung keperluan *memory*-nya!
2. Jelaskan apa yang dimaksud dengan *pointer* serta berikan contoh pendeklarasian dan penggunaan *pointer* dalam bahasa C/C++!
3. Jelaskan secara rinci perbedaan *array* dan *pointer*!
4. Buatlah contoh cara membangkitkan bilangan *random* dalam C/C++!
5. Jelaskan apa yang dimaksud dengan *sorting* dan *searching*!
6. Jelaskan mekanisme metode pengurutan dan pencarian berikut serta berikan contoh pengurutan data yang dilakukan dengan metode tersebut.
7. *Insertion sort*
8. *Bubble sort*
9. *Quick sort*
10. *Sequential search*
11. *Binary search*
12. Buatlah *flowchart* dan *pseudocode* (notasi algoritma) dari masing-masing metode di atas, lalu buatlah *trace* dari *flowchart* yang Anda buat

**Jawaban**

# Array

Tipe data *array* adalah sebuah tipe data yang diperuntukkan dalam proses mengelompokan data-data yang memiliki tipe data sama ke dalam sebuah larik yang membentuk baris urutan dengan acuan indeks tertentu yang memakai sebuah nama yang sama dan dideklarasikan pada variabel yang sama. Data-data ini biasanya disimpan serta dialokaasikan ke dalam alamat memori yang berbeda-beda antara data satu dengan data yang lain dan biasanya disebut sebagai elemen pada *array*. Setiap elemen biasanya memiliki nilai indeks yang harus sesuai dengan urutannya serta selanjutnya dapat digunakan untuk memberi akses dalam membuka data-data yang tersimpan. *Array* memiliki dua tipe, yakni sederhana dan dan bentukan. Biasanya *array* sederhana sendiri memuat tipe data *integer*, *boolean*, *string*, *real*, dan *char*. Sedangkan, untuk tipe data dari *array* bentukan salah satu contohnya adalah *record* yang merupakan jenis dari tipe data yang terstruktur.

Variabel *array* dideklarasikan dengan mencantumkan tipe data yang ingin digunakan, kemudian nama variabel serta diikuti dengan banyaknya data yang ingin diinputkan atau singkatnya jumlah elemen. Contoh program akan dipaparkan pada Kode Program 1.1 berikut ini.

|  |
| --- |
| #include <stdio.h>  int main(){  int nilai[4];  nilai[0] = 59;  nilai[1] = 21;  nilai[2] = 13;  nilai[3] = 25;  nilai[4] = 10;  printf("Nilai elemen secara berturut-turut adalah %d, %d, %d, %d, %d\n", nilai[0], nilai[1], nilai[2], nilai[3], nilai[4], nilai[5]);    return 0;  } |

Kode Program 1. Program Sederhana *Array*

Kode Program 1.1 merupakan contoh program sederhana yang menggunakan *array*. Ruang memori yang dibutuhkan pada program ini untuk mendeklarasikan *array* tadi adalah 20 *byte*, dengan rincian perhitungan 5 \* 4 *byte* (4 *byte* sendiri didapatkan berdasarkan ukuran dari tipe data integer yang digunakan sebagai isi dari elemen).

# Pointer

*Pointer* adalah sebuah variabel yang menunjuk ke alamat memori atau referensi dari variabel lain. Cara pendeklarasian *pointer* sendiri cukup dengan menambahkan tanda *asterisk* atau bintang (\*) di depan sebuah variabel. Tanda *asterisk* ini digunakan untuk mengakses nilai dari variabel yang ditunjuk, sedangkan untuk mendapatkan alamat variabel *pointer* *programmer* dapat menggunakan tanda *emphasis* (&).

|  |
| --- |
| #include <stdio.h>  int main(){  int age = 19;  float tinggi = 175.6;  int \*pointer\_age = &umur;  int \*pointer2 = &age;  float \*p\_height = &height;  printf("alamat memori dari variabel 'age' = %d\n", &age);  printf("alamat memori variabel 'height' = %d\n", &height);  printf("referensi alamat memori \*pointer\_age = %d\n", pointer\_age);  printf("referensi alamat memori \*pointer2 = %d\n", pointer2);  printf("referensi alamat memori \*p\_height = %d\n", p\_height);  // mencetak alamat memori pointer  printf("alamat memori \*pointer\_age = %d\n", &pointer\_age);  printf("alamat memori \*pointer2 = %d\n", &pointer2);  printf("alamat memori \*p\_height = %d\n", &p\_height);    return 0;  } |

Kode Program 2. Program Sederhana *Pointer*

Kode Program 2.1 adalah contoh dari sebuah program sederhana implementasi daripada variabel *pointer* dimana disana terdapat prosesi untuk mencetak alamat memori dari variabel, referensi dari alamat memori *pointer* serta kemudian terdapat proses mencetak alamat memori dari *pointer*.

# Perbedaan Array dan Pointer

*Array* merupakan kumpulan dari nilai data bertipe sama dalam sebuah urutan tertentu yang menggunakan sebuah nama yang sama. Data disimpan dalam alamat memori yang berbeda-beda antara satu data dengan data yang lain dan biasanya disebut sebagai elemen *array*. Sedangkan, *pointer* sendiri merupakan variabel yang menunjuk ke alamat memori dari variabel yang lainnya. Perbedaan yang signifikan dapat dilihat adalah dari sifat kedua variabel ini. *Array*, memiliki sifat statis sedangkan *pointer* bersifat dinamis. Pada *array* sendiri, ukuran memorinya harus pasti dan bedanya dengan *pointer* adalah *pointer* memiliki ukuran memori yang sesuai dengan kebutuhan. Selain daripada itu, alokasi variabel *array* harus dijalankan saat program dimulai atau *start* hingga *end*, sedangkan *pointer* dapat dialokasikan sesuai dengan kebutuhan *user*. Jika dilihat dari sifat-sifat tersebut, sudah dipastikan bahwa *pointer* ini memang memiliki sifat yang dinamis dan dapat menyesuaikan dengan kebutuhan si *user* itu sendiri.

# Cara Membangkitkan Bilangan Randompada Program Bahasa C/C++

Saat *programmer* ingin membangkitkan bilangan *random* pada program Bahasa C/C++ *programmer* memiliki dua fungsi bawaan yang dapat melakukannya. Ketika ingin mengacak angka maka *programmer* dapat menggunakan fungsi srand() dan rand()*.* Sebagai fungsi bawaan maka kedua fungsi ini memerlukan sebuah *header* untuk aktif. *Header* yang digunakan adalah <stdlib.h>. Contoh penggunaannya adalah sebagai berikut.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main(){  int i, x;    printf("Masukkan Jumlah Data Bilangan Acak : \t");  scanf("%d", &x);  for(i = 0; i < x; i++){  printf("Angka Acak : %d\t\n", rand()%(25)+100);  }  return 0;  } |

Kode Program 4. Program Sederhana Rand()

Kode Program 4.1 Merupakan program sederhana yang digunakan untuk membangkitkan atau meregenerasi bilangan secara acak. Pada program di atas, saya akan melakukan regenerasi bilangan yang dimulai dari 100 dengan angka terbesar adalah 125 yang berasal dari penjumlahan 25 + 100.

# Sorting & Searching

*Sorting* dan *Searching* adalah algoritma yang dapat digunakan pada pemrograman. Kedua algoritma ini pada umumnya dapat digunakan di dalam sebuah *array*. Definisi lengkap mengenai kedua algoritma tersebut akan diulas sebagai berikut.

## Sorting

*Sorting* adalah sebuah algoritma yang biasanya digunakan untuk melakukan penyortiran data dengan mengurutkan data terbesar ke data yang terkecil maupun sebaliknya berdasarkan keinginan *programmer*. Algoritma *Sorting* memiliki beberapa jenis diantaranya adalah *Bubble Sort, Selection Sort, Heap Sort, Insertion Sort, Quick Sort, Merge Sort, Shell Sort,* dan masih banyak lagi jenis algoritma *Sorting* yang dapat dipakai. Untuk *sorting* sendiri memiliki dua buah metode, yakni *Ascending* untuk melakukan pengurutan data dari terkecil hingga ke terbesar, dan *Descending* adalah prosesi melakukan pengurutan data dari terbesar hingga ke data yang terkecil.

## Searching

*Searching* merupakan sebuah algoritma yang digunakan untuk menemukan sebuah rangkaian data maupun data yang diinginkan pada sebuah program dengan catatan data yang dicari memiliki tipe data yang sama. Algoritma *searching* memiliki banyak jenis, diantaranya adalah *binary search*, *sequential search*,dan lain-lain.

# Pengurutan dan Pencarian Data

Pengurutan adalah upaya untuk mengatur sekumpulan data berdasarkan urutannya. Pencarian data adalah upaya untuk mendapatkan satu atau lebih objek dari sekumpulan data. Berikut ini merupakan penjelasan beberapa metode pengurutan data dan pencarian data.

## Insertion Sort

*Insertion Sort*, adalah sebuah algoritma pengurutan data yang membandingkan dua elemen data pertama, mengurutkannya, melakukan pemeriksaan elemen data berikutnya secara satu persatu dan membandingkannya dengan elemen data yang telah diurutkan oleh program. Penggunaan metode *insertion sort* akan ditunjukkan pada tabel di bawah secara lengkap beserta dengan isi datanya.

Tabel 6. *Insertion Sort*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data Awal | 25 | 13 | 9 | 28 | 26 | 8 |
| Pass 1 | 13 | 25 | 9 | 28 | 26 | 8 |
| Pass 2 | 9 | 13 | 25 | 28 | 26 | 8 |
| Pass 3 | 9 | 13 | 25 | 28 | 26 | 8 |
| Pass 4 | 9 | 13 | 25 | 26 | 28 | 8 |
| Pass 5 | 8 | 9 | 13 | 25 | 26 | 28 |

Tabel 6.1 merupakan contoh *tracing* dari penggunaan metode *insertion sort*. Data yang diambil pertama adalah data kedua, kemudian dilakukan perbandingan dengan data-data yang ada di sebelah kiri dan seterusnya hingga prosesi penyortiran selesai sesuai dengan metode yang diinginkan oleh user entah itu *descending* ataupun *ascending*. Namun, di *tracing* yang menjadi contoh di atas merupakan metode *ascending*.

## Bubble Sort

*Bubble Sort* adalah proses pengurutan sederhana yang bekerja dengan cara membandingkan sebuah bilangan dengan seluruh bilangan yang terletak sesudah bilangan tersebut. Penggunaan metode *bubble sort* akan ditunjukkan pada tabel sebagai berikut.

Tabel 6. *Bubble Sort*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data Awal | 25 | 9 | 13 | 2 | 5 | 1 |
| Pass 1 | 1 | 25 | 9 | 13 | 2 | 5 |
| Pass 2 | 1 | 2 | 25 | 9 | 13 | 5 |
| Pass 3 | 1 | 2 | 5 | 25 | 9 | 13 |
| Pass 4 | 1 | 2 | 5 | 9 | 25 | 13 |
| Pass 5 | 1 | 2 | 5 | 9 | 13 | 25 |

Tabel 6.2 merupakan contoh pengurutan data yang menggunakan algoritma *bubble sort*. Algoritma ini dimulai dari data terakhir dibandingkan dengan data sebelumnya. Jika data yang berada pada indeks sebelumnya lebih besar, maka akan terjadi pertukaran sampai elemen data sebelumnya lebih kecil dari data yang dibandingkan, apabila menggunakan metode *descending* maka akan berlaku sebaliknya.

## Quick Sort

*Quick Sort* mengurutkan data dengan pendekatan *divide* *and conquer*, yaitu sebuah proses dengan membagi masalah menjadi lebih kecil kemudian menyelesaikannya. Elemen pivot adalah kunci utama dari proses sortir data menggunakan algoritma *quick sort* ini. Elemen pivot sendiri merupakan suatu data yang dipilih untuk menjadi dasar dalam pengurutan. Elemen pivot biasanya dipilih secara acak, lalu kemudian dibandingkan dengan seluruh elemen data yang ada. Data yang berada pada sebelah kiri elemen pivot adalah elemen data yang lebih kecil, begitupun sebaliknya dimana elemen data yang berada di sebelah kanan merupakan data ayng lebih besar.

|  |
| --- |
|  |

Gambar 6.3 Cara kerja *Quick Sort*

Gambar 6.3 merupakan contoh penggunaan *quick sort* beserta alur kerja dari algoritma itu sendiri. Algoritma *quick sort* ini akan membagi elemen data menjadi dua bagian dengan elemen pivot sebagai acuan. Proses ini akan berhenti ketika elemen data tidak bisa dibagi lagi.

## Sequential Search

*Sequential Search* atau *Linear Search* merupakan sebuah algoritma untuk melakukan pencarian data dari sebuah himpunan data. Cara kerjanya adalah dengan melakukan komparasi terhadap seluruh data secara satu persatu dengan terurut berdasarkan data yang dicari sampai ditemukan atau tidak. Semisal terdapat 5 buah data dengan nilai 25, 11, 20, 19, 1 dan pengguna ingin mencari letak dari elemen data 20, maka program akan langsung melakukan komparasi elemen data bernilai 20 pada indeks nomor 2 dengan seluruh indeks yang terdapat pada *array* hingga pada akhirnya ditemukanlah elemen data bernilai 20 tersebut pada indeks *array* nomor 2.

|  |
| --- |
| Linear Search - GeeksforGeeks |

Gambar 6. Contoh Program *Sequential Search*

Gambar 6.4 merupakan salah satu contoh implementasi dari metode *sequential search.* Berdasarkan gambar di atas *programmer* sudah dapat menarik kesimpulan bahwa pada *sequential search* atau *linear search* program akan melakukan pencarian dengan mencocokkan tiap elemen yang berada di dalam sebauh indeks untuk menemukan nilai yang dicari.

## Binary Search

*Binary Search* hanya dapat dilakukan pada data yang sudah terurut, data tersebut akan dibagi menjadi dua bagian yang memiliki jumlah elemen yang sama. Prinsipnya adalah data yang diberikan dibagi menjadi dua dengan jumlah yang sama, lalu selanjutnya akan dilakukan pencarian data pada masing-masing bagian. Misalnya, terdapat 6 data yang telah terurut yaitu 1, 2, 3, 4, 5, 6. Kemudian, pengguna ingin mencari letak nilai 5, maka program akan membagi seluruh indeks pada *array* dengan *index* nomor 3 sebagai titik tengah karena elemen data bernilai 5 ada pada indeks ke 5. Selanjutnya, data pada indeks 2 ke bawah atau sebelah kiri tidak akan dipedulikan dan program hanya akan terfokus pada data di sebelah kanan sampai elemen data dengan nilai 5 ditemukan pada indeks nomor 5.

|  |
| --- |
| Binary Search - GeeksforGeeks |

Gambar 6. Mekanisme Program *Binary Search*

Gambar 6.5 merupakan salah satu contoh implementasi dari metode *binary search.* Gambar di atas sudah sangat jelas dan ringkas penjelasannya mengenai *binary search* sehingga dapat digunakan sebagai landasan untuk mempelajari *binary search* lebih lanjut.

# Flowchart, Pseudocode dan Trace Metode Program

Selanjutnya akan ada penjelasan dari metode-metode yang ada pada algoritma *sorting* dan *searching* yang berada paa nomor 6. Penjelasan secara mendetailnya akan dijelaskan sebagai berikut yang akan berawal dari *flowchart* program utama terlebih dahulu.

## Flowchart Program Sorting dan Searching

Pada sub bab ini, pembahasan selanjutnya adalah mengenai *flowchart*. *Flowchart* pada Gambar 7.1 akan memberikan alur kerja secara mendetail dari program *sorting* dan *searching*.

|  |
| --- |
|  |

Gambar 7. *Flowchart* Program *Sorting* dan *Searching*

Gambar 7.1 merupakan *flowchart* *main menu* dari program *sorting* dan *searching* yang telah dibuat. *Sub flowchart* dari berbagai metode *sorting* dan *searching* akan dijabarkan sebagai berikut.

## Pseudocode Program Sorting dan Searching

*Pseudocode* yang berada pada sub bab 7.2 ini merupakan *pseudocode* yang menjadi pondasi utama untuk menyusun kode program dari program *sorting* dan *searching* sehingga pada *pseudocode* ini sudah mencakup seluruh *pseudocode* dari masing-masing metode *sorting* dan *searching* yang ada di atas. *Pseudocode* akan ditampilkan pada Kode Program 7.2 di bawah ini.

|  |
| --- |
| ALGORITMA SORTING DAN SEARCHING  DEKLARASI :  DESKRIPSI :  PROCEDURE seedArray(int arr[], int n)  DEKLARASI :  i : integer  srand(0);  FOR (i = 0; i < n; i++)  arr[i] = rand() % 64000;  arr[i] = rand();  END FOR  END PROCEDURE  PROCEDURE seedArraySearching (int arr[], int n)  DEKLARASI :  i : integer  srand(0);  FOR (i = 0; i < n; i++)  arr[i] = rand() % 1000;  END FOR  END PROCEDURE  PROCEDURE dupArray (int source[], int target[], int n)  DEKLARASI :  i : integer  FOR (i = 0; i < n; i++)  target[i] = source[i];  END FOR  END PROCEDURE  PROCEDURE printArray(int arr[], int n)  DEKLARASI :  i : integer  FOR (i = 0; i < n; i++)  printf("-Angka ke-%d : %d", i, arr[i]);  printf("\n");  END FOR  END PROCEDURE  FUNCTION partition(int arr[], int low, int high)  DEKLARASI :  pivot = arr[high], temp, i = (low-1), j : integer  FOR (j = low; j <= high - 1; j++)  IF (arr[j] <= pivot)  i++;  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  END IF  END FOR  temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  return (i + 1);  END FUNCTION  PROCEDURE quickSort(int arr[], int low, int high)  DEKLARASI :  pi = partition(arr, low, high) : integer  IF(low < high)  pi = partition(arr, low, high)  quickSort(arr, low, pi – 1)  quickSort(arr, pi + 1, high)  END IF  END PROCEDURE  PROCEDURE insertionSort(int arr[], int n)  DEKLARASI :  i, j, key : integer  FOR(i = 1; i < n; i++)  key = arr[i]    j = i – 1  WHILE (j >= 0 AND arr[j] > key)  arr[j + 1] = arr[j]  j—  END WHILE  arr[j + 1] = key  END FOR  END PROCEDURE  PROCEDURE bubbleSort(int arr[], int n)  DEKLARASI :  i, j, temp : integer  FOR (i = 0; i < n-1; i++)  FOR (j = 0; i < n-1-i; j++)  IF (arr[j] > arr[j+1])  temp = arr[j]  arr[j] = arr[j+1]  arr[j+1] = temp  END IF  END FOR  END FOR  END PROCEDURE  FUNCTION Sorting()  DEKLARASI :  detik1, detik2, detik3 : double  i, n, arr[n], arr2[n], arr3[n] : integer  pilihan, enterCheck : char  clock\_t waktu  WRITE (“Masukkan Pilihan : ”)  IF (read(pilihan, enterCheck) != 2 OR enterCheck != ‘\n’)  WRITE(“Input Invalid”)  ELSE  if (pilihan == '1')  n = 1000;  break;  else if (pilihan == '2')  n = 16000;  break;  else if (pilihan == '3')  n = 64000;  break;  else if (pilihan == '4')  main();  break;  else if (pilihan == '5')  return 0;  break;  else  WRITE("Input Invalid!")  END IF  END IF  CALL PROCEDURE seedArray(arr, n)  WRITE(“Array %d data random sebelum di sort :\n”)  READ(n)  CALL PROCEDURE printArray(arr, n)  CALL PROCEDURE dupArray(arr, arr2, n)  CALL PROCEDURE dupArray(arr, arr3, n)  waktu = clock()  CALL PROCEDURE insertionSort(arr, n)  waktu = clock()-waktu  detik1 (double)(waktu)/CLOCKS\_PER\_SEC  waktu = clock();  CALL PROCEDURE bubbleSort(arr2, n);  waktu = clock() - waktu;  detik2 = (double)(waktu) / CLOCKS\_PER\_SEC;  waktu = clock();  CALL PROCEDURE quickSort(arr3, 0, n - 1);  waktu = clock() - waktu;  detik3 = (double)(waktu) / CLOCKS\_PER\_SEC;  WRITE(“Array %d data random sesudah di sort :”)  CALL PROCEDURE printArray(arr, n)  WRITE (“Perbandingan waktu : “)  WRITE(“Insertion Sort Membutuhkan %f Detik Untuk Mensortir Data”)  READ(detik1)  WRITE("Bubble Sort Membutuhkan %f Detik Untuk Mensortir Data”)  READ(detik2)  WRITE("Quick Sort Membutuhkan %f Detik Untuk Mensortir Data")  READ(detik3)  IF(detik1 <= detik2 AND detik1 <= detik3)  WRITE(“Insertion Sort Paling Cepat”)  ELSE IF (detik2 <= detik1 AND detik1 <= detik3)  WRITE(“Bubble Sort Paling Cepat”)  ELSE  WRITE(“Quick Sort Paling Cepat”)  END IF  CALL PROCEDURE akhir()  END FUNCTION  FUNCTION sequentialSearch(int arr[], int n,int angkaDicari)  DEKLARASI :  detik : double  i, counter = 0 : integer  waktu = clock();  FOR (i = 0; i < n; i++)  IF (arr[i] == angkaDicari)  counter++;  END IF  WRITE("Angka ke-%d : %d", i, arr[i])  READ(i)  READ(arr[i])  WRITE (“\n”)  END FOR  waktu = clock() - waktu  detik = (double)(waktu) / CLOCKS\_PER\_SEC  IF (counter == 0)  WRITE("\nTidak Ada Angka Yang Sesuai, pada sequential search!");  ELSE  int indexAngkaDicari[counter];  counter = 0;  FOR (i = 0; i < n; i++) {  IF (arr[i] == angkaDicari)  indexAngkaDicari[counter] = i;  counter++;  END IF  END FOR  WRITE("\n\nAngka Ditemukan Pada Index sequential search: ");  FOR (i = 0; i < counter; i++) {  IF (i == 0 && i == counter - 1) {  WRITE(" %d", indexAngkaDicari[i]);  ELSE  IF (i == counter - 1) {  WRITE(" dan %d.", indexAngkaDicari[i]);  ELSE  WRITE(" %d,", indexAngkaDicari[i]);  END IF  END IF  END FOR  END IF  return detik  END FUNCTION  FUNCTION binarySearch(int arr[], int low, int high, int angkaDicari)  DEKLARASI :  clock\_t waktu  detik : double  i, counter = 0 : integer  WHILE (low <= high) {  int mid = low + (high - low) / 2  IF (arr[mid] == angkaDicari)  counter++  FOR (i = mid + 1; i < high + 1 && arr[i] == angkaDicari; i++)  counter++  END FOR  FOR (i = mid - 1; i < high + 1 && arr[i] == angkaDicari; i++)  counter++  END FOR  break  ELSE IF (arr[mid] > angkaDicari)  high = mid - 1  ELSE  low = mid + 1  END IF  END WHILE  waktu = clock() - waktu  detik = (double)(waktu) / CLOCKS\_PER\_SEC;  IF (counter == 0)  WRITE("\n\nTidak Ada Angka Yang Sesuai pada binary search!");  ELSE  int indexAngkaDicari[counter]  counter = 0;  WHILE (low <= high)  int mid = low + (high - low) / 2;  IF (arr[mid] == angkaDicari) {  indexAngkaDicari[counter] = mid;  counter++;  FOR (i = mid + 1; i < high + 1 && arr[i] == angkaDicari; i++)  indexAngkaDicari[counter] = i;  counter++;  END FOR  FOR (i = mid - 1; i > 0 && arr[i] == angkaDicari; i--)  indexAngkaDicari[counter] = i;  counter++;  END FOR  break;  ELSE IF (arr[mid] > angkaDicari)  high = mid - 1;  ELSE  low = mid + 1;  END IF  END WHILE  WRITE("\n\nAngka Ditemukan Pada Index, binary search: ")  CALL PROCEDURE bubbleSort(indexAngkaDicari, counter)  FOR (i = 0; i < counter; i++)  IF (i == 0 && i == counter - 1)  WRITE(" %d.", indexAngkaDicari[i])  ELSE  IF (i == counter - 1) {  WRITE(" dan %d.", indexAngkaDicari[i])  ELSE  WRITE(" %d,", indexAngkaDicari[i]);  END IF  END IF  END FOR  END IF  return detik;  END FUNCTION  FUNCTION Searching()  DEKLARASI :  clock\_t waktu  detik1, detik2 : double  i, n, menu, arr[n], arr2[n], angkaDicari : integer  enterCheck, pilihan : character  WRITE("\nMasukkan pilihan: ");  IF (READ("%c%c", & pilihan, & enterCheck) != 2 OR enterCheck != '\n')  WRITE("\nInput Invalid!\n");  ELSE  IF (pilihan == '1')  n = 1000  break  ELSE IF (pilihan == '2')  n = 16000  break  ELSE IF (pilihan == '3')  n = 64000  break  ELSE IF (pilihan == '4')  main()  break  ELSE IF (pilihan == '5')  return 0  break  ELSE  printf("\nInput Invalid!\n")  sleep(1)  Searching()  END IF  END IF  CALL PROCEDURE seedArraySearching(arr, n);  CALL PROCEDURE printArray(arr, n);  WRITE("\nMasukkan Angka Yang Ingin Dicari: ");  IF (((READ("%d%c", & angkaDicari, & enterCheck)) != 2 OR enterCheck != '\n'))  WRITE ("\nInput Invalid!\n");  ELSE  IF (angkaDicari < 0) {  WRITE("Inputan tidak boleh dibawah '0'");  ELSE  break;  END IF  END IF  CALL PROCEDURE bubbleSort(arr, n);  detik1 = sequentialSearch(arr, n, angkaDicari);  detik2 = binarySearch(arr, 0, n - 1, angkaDicari);  WRITE("\n\nperbandingan waktu: \n");  WRITE("\nSequential Search Membutuhkan %f Detik Untuk Mencari Data", detik1);  WRITE("\nBinary Search Membutuhkan %f Detik Untuk Mencari Data\n", detik2);  IF (detik1 <= detik2) {  WRITE("Maka Pada Pencarian Kali Ini, Sequential Search Lebih Cepat Dari Binary Search\n");  ELSE  WRITE("Maka Pada Pencarian Kali Ini, Binary Search Lebih Cepat Dari Sequential Search\n");  END IF  CALL PROCEDURE akhir();  END FUNCTION  FUNCTION MAIN  DEKLARASI :  pilihan, enterCheck : character  WRITE("\nMasukkan pilihan: ");  IF (READ("%c%c", & pilihan, & enterCheck) != 2 OR enterCheck != '\n') {  WRITE("\nInput Invalid!\n");  ELSE  IF (pilihan == '1')  Sorting();  break;  ELSE IF (pilihan == '2')  Searching();  break;  ELSE IF (pilihan == '3')  cetakAkhir();  return 0;  break;  ELSE  WRITE("\nInput Invalid!\n");  sleep(1);  main();  END IF  END IF  END FUNCTION  PROCEDURE cetakAkhir()  CLEARSCREEN  WRITE("-----------------------------------------------\n")  WRITE("\* TERIMA KASIH SUDAH MENGGUNAKAN PROGRAM KAMI \*\n")  WRITE("| ADIOS :) |\n")  WRITE("\* FROM KELOMPOK 12 \*\n")  WRITE("-----------------------------------------------\n")  END PROCEDURE  PROCEDURE akhir()  DEKLARASI :  pilihan, enterCheck : character  WRITE("\nMasukkan pilihan: ")  IF ((scanf("%c%c", & pilihan, & enterCheck)) != 2 || enterCheck != '\n')  WRITE("\nInput Invalid!\n")  ELSE  IF (pilihan == '1') {  main();  break;  ELSE IF (pilihan == '2') {  cetakAkhir();  break;  ELSE  WRITE("\nInput Invalid!\n");  END IF  END IF  END PROCEDURE |

Kode Program 7. *Pseudocode* Program *Sorting* dan *Searching*

Pada Kode Program 7.2 di atas, dapat dilihat struktur dari penyusun programnya secara jelas, baik dari variabel serta tipe data yang digunakan. Selain itu, *pseudocode* di atas sudah mencakup berbagai metode *sorting* dan *searching* yang ada.

## Trace Program Sorting dan Searching

*Tracing* adalah proses menganalisis alur kerja program secara langsung dengan menjalankan program tersebut. *Tracing* program *sorting* dan *searching* sendiri akan dijabarkan pada tabel *trace* berikut ini

Tabel 7.3 *Trace* Program *Sorting* dan *Searching*

|  |  |
| --- | --- |
| Input | pilihan ‘1’ : sorting ()  pilihan ‘1’ : n = 1000 |
| Proses | void seed\_array(int arr[],int n){  srand(0);  for (i = 0; i < n ; i++){  arr[i] = rand()%64000;  arr[i] = rand();  }  }  void printArray(int arr[], int n){  int i;  for (i=0; i < n; i++)  printf("%d ", arr[i]);  printf("\n");  }  void insertion\_sort(int arr[], int n){  int i, j, key;  for (i = 1; i < n; i++){  key = arr[i];  j = i-1;  while (j >= 0 && arr[j] > key){  arr[j+1] = arr[j];  j = j--;  }  arr[j+1] = key;  }  }  float count\_insertion\_sort(int arr[],int n){  clock\_t start = clock();  insertion\_sort(arr,n);  clock\_t end = clock();  float seconds = (float)(end - start) / CLOCKS\_PER\_SEC;  return seconds;  }  void bubble\_sort(int arr[], int n){  for (i = 0; i < n-1; i++)  for (j = 0; j < n-i-1; j++)  if (arr[j] > arr[j+1]){  temp = arr[j];  arr[j] = arr[j+1];  arr[j+1] = temp;  }  }  int partition(int arr[], int low, int high) {  int pivot = arr[high];  int temp;  int i = (low - 1);  int j;  for (j = low; j <= high - 1; j++) {  if (arr[j] <= pivot) {  i++;  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  return (i + 1);  }  void quickSort(int arr[], int low, int high) {  int pi = partition(arr, low, high);  if (low < high) {  pi = partition(arr, low, high);  quickSort(arr, low, pi - 1);  quickSort(arr, pi + 1, high);  }  }  int Sorting() {  clock\_t waktu;  double detik1;  double detik2;  double detik3;  int i;  char pilihan;  char enterCheck;  int n;  int arr[n];  int arr2[n];  int arr3[n];  do {  printf("\nMasukkan pilihan: ");  if (scanf("%c%c", & pilihan, & enterCheck) != 2 || enterCheck != '\n') {  printf("\nInput Invalid!\n");  fflush(stdin);  } else {  if (pilihan == '1') {  n = 1000;  break;  } else if (pilihan == '2') {  n = 16000;  break;  } else if (pilihan == '3') {  n = 64000;  break;  } else if (pilihan == '4') {  main();  break;  } else if (pilihan == '5') {  return 0;  break;  } else {  printf("\nInput Invalid!\n");  sleep(1);  system("cls");  Sorting();  }  }  } while (1);  seedArray(arr, n);  printf("Array %d data random sebelum di sort :\n", n);  printArray(arr, n);  dupArray(arr, arr2, n);  dupArray(arr, arr3, n);  //Insertion Sort  waktu = clock();  insertionSort(arr, n);  waktu = clock() - waktu;  detik1 = (double)(waktu) / CLOCKS\_PER\_SEC;  //Bubble Sort  waktu = clock();  bubbleSort(arr2, n);  waktu = clock() - waktu;  detik2 = (double)(waktu) / CLOCKS\_PER\_SEC;  //Quick Sort  waktu = clock();  quickSort(arr3, 0, n - 1);  waktu = clock() - waktu;  detik3 = (double)(waktu) / CLOCKS\_PER\_SEC; |
| Output | Array 1000 data random sebelum di sort :  -Angka ke-0 sampai dengan –Angka ke-999  Array 1000 data random sesudah di sort :  -Angka ke-0 sampai dengan –Angka ke-999  Perbandingan waktu :  Insertion Sort Membutuhkan 0.000000 Detik Untuk Mensortir Data  Bubble Sort Membutuhkan 0.002000 Detik Untuk Mensortir Data  Quick Sort Membutuhkan 0.001000 Detik Untuk Mensortir Data  Insertion sort yang paling cepat |

Tabel 7.3 merupakan hasil dari proses *tracing* mengenai program lengkap dengan contoh *input*, proses, serta *output* yang akan dihasilkan program berdasarkan *input* yang diberikan *user*. Selain itu, *proses* yang berjalan pada program juga berdasarkan metode yang diberikan *programmer* dan berjalan secara terstruktur dan sekuensial.

## Sub Flowchart Metode Insertion Sort dan Pemanggilan Bilangan Acak

Setelah mengetahui dan memahami bagaimana alur kerja program *sorting* dan *searching*, sekarang saatnya untuk memahami alur dari masing-masing metode yang digunakan beserta komponen-komponen tambahan lainnya seperti pemanggilan bilangan secara acak. Berikut ini akan dijabarkan pada Gambar 7.4 mengenai sub *flowchart* dari metode *insertion sort* serta pemanggilan bilangan acak.

|  |
| --- |
|  |

Gambar 7.*Pseudocode Insertion Sort* dan Bilangan Acak

Gambar 7.4 di atas merupakan sub *flowchart* yang digunakan untuk menggambarkan serta mendeskripsikan secara urut alur kerja dari metode *insertion* *sort*. Untuk metode selanjutnya akan dijabarkan pada Gambar 7.5 sub *flowchart bubble sort* dan *quick sort*

## Sub Flowchart Metode Bubble Sort dan Quick Sort

Kemudian, setelah memahami alur kerja dari metode *insertion sort* dengan baik, maka selanjutnya penjelasan rinci menggunakan *flowchart* akan dilanjutkan untuk digunakan menjelaskan alur kerja dari dua metode algoritma pengurutan bilangan. Algoritma pengurutan bilangan yang akan dijelaskan antara lain adalah *Bubble Sort* serta *Quick Sort*.

|  |
| --- |
|  |

**Gambar 7.5** *Subflowchart* Metode *Bubble Sort* dan *Quick Sort*

Gambar 7.5 merupakan *subflowchart* yang menjelaskan alur kerja dari metode *Bubble Sort* dan *Quick Sort*. Metode di atas terdapat pada program utama yang telah di enkapsulasi pada sebuah prosedur yang digunakan untuk meringkas sintaks dan mempermudah pemanggilan dari metode.