Nama : Anak Agung Indi Kusuma Putra

NIM : 2205551079

Kelompok : 23

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

**Tujuan**

1. Mengetahui dan memahami tentang *array* serta tipe data yang mungkin digunakan dalam *array,* dan dapat mendeklarasikan *array* dan mengetahui keperluan memorinya.
2. Mengetahui dan memahami tentang *pointer* serta dapat mendeklarasikan *pointer*.
3. Mengetahui dan memahami secara rinci perbedaan dari *array* dan *pointer*.
4. Mengetahui dan memahami cara untuk membangkitkan bilangan *random* dalam C/C++.
5. Mengetahui dan memahami apa itu *sorting* dan *searching.*
6. Mengetahui dan memahami mekanisme pengurutan dan pencarian serta memahami tentang contoh pengurutan data seperti *Insertion sort, Bubble sort, Quick sort, Sequential search, Binary search.* Serta dapat menerapkan *flowchart* dan *pseudocode* dalam metode tersebut.

**Tugas Pendahuluan**

1. Jelaskan tentang *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 diatas, lalu buatlah *trace* dari *flowchart* yang anda buat.

**Jawaban**

1. **Array**

*Array* adalah sekumpulan data dengan tipe data yang sama yang ditempatkan pada suatu lokasi *memory* yang berdekatan dan data akan ditempatkan secara berurutan. Dengan kata lain, *array* dapat digunakan untuk menyimpan banyak nilai dengan tipe data yang sama dalam sebuah variabel. Setiap elemen di dalam *array* tersebut dapat diakses secara acak menggunakan indeks *array* yang merupakan urutan dari data yang ada di dalam *array*. *Array* bisa menyimpan tipe data berupa *Interger, Char, String,* dan  *Boolean.*

* 1. **Pendeklarasian Array dan Perhitungan Memori pada Array**

Pendeklarasian *array*  dengan mencantumkan nama variabel *array*, tipe data yang digunakan, dan panjang *array* yang diapit dengan kurung siku. Perhitungan memori pada *array* dapat dilakukan dengan mengkalikan panjang array dengan jumlah keperluan memori dari tipe data yang digunakan. Contoh pendeklarasian *array* bisa dilihat pada program dibawah.

|  |
| --- |
| #include <stdio.h>  int main()  {  int angka[5] = {0,1,2,3,4};  } |

Kode Program 1.1 Contoh Pendeklarasian *Array*

Kode program 1.1 merupakan contoh pendeklarasian *array* yang menggunakan tipe data *integer*. *Array* tersebut memiliki panjang *array 5.* Yang berarti keperluan memori yang dibutuhkan bisa dihitung dengan mengkalikan panjang *array* (5) dengan keperluan memori dari 1 buah data integer (4 *byte*), jadi keperluan memori dari *array* tersebut adalah 20 *byte.*

1. **Pointer**

*Pointer* dalam bahasa C/C++ adalah tipe data yang nilainya mengacu pada nilai lain yang disimpan di variabel lain di memori komputer lalu menggunakan alamatnya. *Pointer* dilambangkan dengan simbol ‘\*’. Jadi pada dasarnya *pointer* mengambil alamat memori dari suatu variabel dan menggunakannya pada variabel yang diberi pointer. Pendeklarasian pointer dilakukan dengan cara menambahkan simbol ‘\*’ di depan nama variabel yang dideklarasian dan simbol ‘&’ di depan nama variabel yang alamat memorinya akan digunakan. Contoh pendeklarasian dan penerapan *pointer* pada C/C++ bisa dilihat pada program dibawah.

|  |
| --- |
| #include <stdio.h>  int main()  {  int umur = 19;  int \*umurSekarang = &umur;  printf("umur saya sekarang = %d ", \*umurSekarang);  } |

Kode Program 2.1 Contoh Pendeklarasian *Pointer*

Kode Program 2.1 merupakan contoh pendeklarasian dari *pointer* dimana variabel umurSekarang diberi *pointer* di depannya dan mengambil alamat memori dari variabel umur, jadi ketika *print* output dari variabel umurSekarang akan dama dengan variabel umur

1. **Perbedaan Array dan Pointer**

*Array* dan *pointer* merupakan hal yang jelas berbeda dalam C/C++, dimana *array* merupakan kumpulan data yang berisi data dengan tipe data yang sama secara berurutan sedangkan *pointer* merupakan variabel yang sudah diberi tanda dan isinya mengabil dan mengacu pada alamat memori lain.

1. **Membangkitkan Bilangan Random pada C/C++**

Untuk membangkitkan bilangan *random* dalam C/C++, bisa menggunakan fungsi rand() atau srand() yang terdapat pada *library* <stdlib.h> fungsi tersebut akan me-*return* bilangan *random*. Contoh cara untuk membangkitkan bilang *random* pada C/C++ bisa dilihat pada program dibawah.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main()  {  printf("%d", rand());  } |

Kode Program 2.1 Contoh Cara Untuk Membangkitkan Bilangan *Random*

1. **Sorting dan Searching**

*Sorting* dan *searching* merupakan contoh logika algoritma yang digunakan dalam pemrograman untuk menyelesaikan masalah dengan tingkat yang lebih kompleks.

* 1. **Sorting**

Algoritma *sorting* adalah suatu logika algoritma yang digunakan untuk mengurutkan suatu data acak yang terdapat dalam suatu kumpulan data. Algoritma *sorting* banyak digunakan untuk mengurutkan data numerikal, baik itu dari data terkecil ke data terbesar atau yang biasa disebut dengan *ascending* maupun dari data terbesar ke data terkecil atau yang biasa disebut juga *descending.*

* 1. **Searching**

Algoritma *searching* adalah suatu logika algoritma yang digunakan untuk mencari suatu data yang diinginkan dalam kumpulan data acak. Algoritma *searching* akan menerima *input* berupa data yang akan dicari, lalu algoritma *searching* akan memproses dan mulai mencari data yang dicari lalu akan terdapat kemungkinan 2 *output* yaitu data ditemukan atau tidak ditemukan.

1. **Metode Sorting dan Searching**

Ada beberapa mekanisme metode *sorting* dan *searching* yang umum digunakan yaitu *insertion sort, bubble sort, quick sort, sequential search, dan binary search.*

* 1. **Insertion Sort**

*Insertion sort* algoritma merupakan algoritma *sorting* yang memilah data yang akan diurutkan menjadi 2 bagian yaitu data yang telah diurutkan dan yang belum diurutkan. Data pertama pada *array* akan diambil dan dibandingkan dengan data selanjutnya, lalu akan diurutkan sesuai dengan posisinya dan diletakan pada bagian *array* yang telah diurutkan. Data selanjutnya akan dibandingkan dengan bagian *array* yang telah diurutkan dan akan diletakan pada urutan yang seharusnya. Langkah ini terus dilakukan secara berulang hingga tak ada lagi data yang tersedia pada *array.* Contoh proses dari *insertion sort* bisa dilihat pada tabel dibawah.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Awal | 3 | 2 | 5 | 6 | 8 | 9 | 10 | 1 | 4 | 7 |
| Proses 1 | 2 | 3 | 5 | 6 | 8 | 9 | 10 | 1 | 4 | 7 |
| Proses 2 | 2 | 3 | 5 | 6 | 8 | 9 | 10 | 1 | 4 | 7 |
| Proses 3 | 2 | 3 | 5 | 6 | 8 | 9 | 10 | 1 | 4 | 7 |
| Proses 4 | 2 | 3 | 5 | 6 | 8 | 9 | 10 | 1 | 4 | 7 |
| Proses 5 | 2 | 3 | 5 | 6 | 8 | 9 | 10 | 1 | 4 | 7 |
| Proses 6 | 2 | 3 | 5 | 6 | 8 | 9 | 10 | 1 | 4 | 7 |
| Proses 7 | 1 | 2 | 3 | 5 | 6 | 8 | 9 | 10 | 4 | 7 |
| Proses 8 | 1 | 2 | 3 | 4 | 5 | 6 | 8 | 9 | 10 | 7 |
| Proses 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Tabel 6.1 Contoh Proses Algoritma Dari *Insertion Sort*

Tabel 6.1 diatas merupakan proses dari algoritma *insertion sort* yang dimana pada tabel diatas, angka yang berwarna biru menunjukan data yang telah diurutkan, jadi data yang belum diurutkan akan dibandingkan dengan data yang telah diurutkan sebelumnya lalu diletakan pada tempat sesuai dengan urutannya yang benar.

* 1. **Bubble Sort**

Bubble Sort merupakan metode *sorting* dengan membandingkan tiap data pada *array* yang bersebelahan dan menukarnya apabila urutannya salah, proses ini terus berulang sampai data dalam *array* telah terurut. Contoh proses algoritma dari *bubble sort* bisa dilihat pada tabel dibawah.

|  |  |
| --- | --- |
| Proses 1 | 1 8 6 2 menjadi 1 8 6 2 |
| 1 8 6 2 menjadi 1 6 8 2 |
| 1 6 8 2 menjadi 1 6 2 8 |
| Proses 2 | 1 6 2 8 menjadi 1 6 2 8 |
| 1 6 2 8 menjadi 1 2 6 8 |
| 1 2 6 8 menjadi 1 2 6 8 |
| Proses 3 | 1 2 6 8 menjadi 1 2 6 8 |
| 1 2 6 8 menjadi 1 2 6 8 |
| 1 2 6 8 menjadi 1 2 6 8 |

Tabel 6.2 Contoh Proses Algoritma Dari *Bubble Sort*

Tabel 6.2 diatas merupakan contoh proses algoritma *bubble sort* yang dimana algoritma akan membandingkan data dengan data setelahnya lalu jika urutannya salah maka akan ditukar. Data yang berwarna biru adalah data yang dibandingkan.

* 1. **Quick Sort**

*Quick sort* merupakan salah satu metode *sorting* yang bisa mengurutkan data dengan cepat. Pertama metode ini akan mengambil salah satu data acak yang disebut dengan pivot lalu akan mengecek data dan data yang lebih kecil akan ditempatkan di sebelah kiri pivot dan data yang lebih besar akan ditempatkan di sebelah kanan pivot. Hal ini akan dilakukan terus menerus hingga semua data sudah terurut dengan benar. Contoh proses algoritma *quick sort* bisa dilihat pada tabel dibawah.

|  |
| --- |
|  |

Tabel 6.3 Contoh Proses Algoritma dari *Quick Search*

Tabel 6.3 diatas merupakan contoh proses algoritma dari *quick search* yang menunjukan algoritma *quick search* akan menentukan pivot lalu mengecek data dan data yang lebih kecil akan ditempatkan di sebelah kiri pivot dan data yang lebih besar di sebelah kanan pivot.

* 1. **Sequental Search**

*Sequental search* merupakan metode *searching* yang akan mencari suatu data sesuai dengan kata kunci yang diberikan. Algoritma ini akan mencari data dengan membandingkan kata kunci yang dicari dengan data yang ada dalam *array* secara berurut dari data *array* awal hingga data *array* akhir. Contoh proses dari algoritma *sequental search* bisa dilihat pada tabel dibawah.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data | 9 | 1 | 3 | 2 | 4 | 6 | 11 | 5 |
| Proses 1 | 9 | 1 | 3 | 2 | 4 | 6 | 11 | 5 |
| Proses 2 | 9 | 1 | 3 | 2 | 4 | 6 | 11 | 5 |
| Proses 3 | 9 | 1 | 3 | 2 | 4 | 6 | 11 | 5 |
| Proses 4 | 9 | 1 | 3 | 2 | 4 | 6 | 11 | 5 |
| Proses 5 | 9 | 1 | 3 | 2 | 4 | 6 | 11 | 5 |

Tabel 6.4 Contoh Proses Algoritma Dari Sequental Search

Tabel 6.4 diatas merupakan contoh dari algoritma *sequential search* yang akan mencari suatu data. Pada tabel diatas kita mencari angka 4 pada kumpulan data acak. Algoritma *sequential search* akan membandingkan data satu persatu dengan data yang akan dicari. Data yang berwarna biru pada tabel diatas merupakan data yang sudah dibandingkan dengan kata kunci yang diberikan Jika data yang dicari sudah ditemukan maka proses pencarian akan berhenti

* 1. **Binary Search**

*Binary search* merupakan metode *searching*  dengan cara membagi data

menjadi dua bagian dari data keseluruhan terus menerus sampai elemen yang dicari sudah ditemukan atau saat indeks yang berada dikiri lebih besar dari yang di kanan. *binary search* memiliki syarat utama yaitu kumpulan data yang akan dilakukan perurutan sudah diurutan terlebih dahulu, baik dari data terkecil ataupun dari data terbesar. *binary search*  merupakan metode yang bisa terbilang efisien and cepat untuk mencari nilai dalam jumlah data yang besar. Contoh proses dari algoritma *binary search* bisa dilihat pada tabel dibawah.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Proses 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Proses 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Proses 3 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Tabel 6.5 Contoh Proses Algoritma dari *Binary Search*

Tabel 6.5 diatas merupakan contoh proses dari algoritma dari *binary search* yang membagi data membagi 2 bagian secara terus menerus hingga data yang dicari ditemukan, diatas data yang berwarna hijau bertindak sebagai starter atau data awal, data yang berwarna biru sebagi pivot atau batas tengah yang membagi data yang akan dibagi 2, data berwarna kuning data yang dicari, data berwarna merah sebagai data akhir dari kumpulan data yang akan dicari.

## Flowchart dan Pseudocode Metode Pengurutan dan Pencarian

*Flowchart* dan *pseudocode* merupakan dua notasi algoritma yang digunakan sebagai representasi dari suatu tahap penyelesaian masalah atau algoritma. *Flowchart* dan *pseudocode* sering digunakan dikarenakan memiliki penggambaran proses algoritma yang mudah untuk dimengerti dan dapat diimplementasikan ke dalam bahasa program dengan sintaks yang berbeda. Berikut adalah *flowchart* dan *pseudocode* serta *trace* dari metode pengurutan dan pencarian.

### **Flowchart, Pseudocode, dan Trace Pada Insertion Sort**

*Insertion sort* merupakan suatu metode algoritma pengurutan sederhana. *Array* dipisahkan ke dalam *sorted* dan *unsorted sub array.* Nilai-nilai yang berada pada *unsorted sub array* diambil dan ditempatkan ke dalam *sorted sub array* secara satu per satu pada tempat yang benar. *Insertion sort* efektif untuk mengurutkan nilai data yang sedikit dan cocok untuk mengurutkan data yang sudah diurutkan beberapa terlebih dahulu. Berikut adalah *flowchart* dan *pseudocode* serta *trace* dari metode *insertion sort*.

#### Flowchart pada Insertion Sort

Demi memahami suatu algoritma, ada baiknya untuk melihat notasi algoritma *flowchart* untuk mendapatkan visualisasi dari alur algoritma tersebut dengan lebih baik. Berikut adalah *flowchart* dari *insertion sort*.

|  |
| --- |
|  |

Gambar 7.1 *Flowchart* *Insertion Sort*

Gambar 7.1 merupakan penggambaran alur dari metode algoritma *insertion sort* melalui *flowchart.* Setelah deklarasi *array* dan *n* sebagai *sizeof(array)*, *insertion sort* pun dilakukan dengan menggunakan iterasi *nested loop*. Variabel *i* menjadi patokan selama iterasi, yang mana iterasi akan terus dilakukan selama *i* kurang dari *n*. Dua elemen pertama dari *array* akan dibandingkan terlebih dahulu dengan menggunakan bantuan variabel *key* dan *j,* yang mana variabel *key* merupakan variabel *array* ke *i* dan variabel *j* adalah variabel sebelum *i.* Bila variabel *j* lebih atau sama dengan 0 dan *array[j]* lebih dari *key,* (yang berarti elemen awal lebih besar daripada elemen setelahnya) maka elemennya harus ditukar dengan cara menaruh elemen *array[j]* pada *array[j+1]* lalu di-*assign* ke dalam variabel *key* untuk perbandingan selanjutnya. Iterasi ini terus berlanjut hingga kondisi iterasi bernilai salah dan data pun akan terurut.

#### Pseudocode pada Insertion Sort

Setelah pembuatan *flowchart,* ada baiknya untuk menyusun sebuah *pseudocode* sebagai patokan agar dapat merancang kode program yang sesuai dengan algoritma tersebut. *Pseudocode* sendiri umumnya tidak memiliki peraturan *syntax* yang absolut selama dapat dimengerti oleh manusia, sehingga dapat diimplementasikan pada berbagai kode program yang mendukung. Berikut adalah *pseudocode* dari *insertion sort*.

|  |
| --- |
| PROGRAM insertionSort  BEGIN  DECLARE  int array[]  int n=sizeof(array)  int i, j , key  FOR (i = 1 ; i < n ; i++)  key = array[i]  j = i - 1    WHILE (j >= 0 && array[j] > key)  array[j + 1] = array[j]  j = j - 1  END WHILE    array[j + 1] = key    END FOR  END |

Kode Program 7.1 *Pseudocode Insertion Sort*

Kode Program 7.1 merupakan *pseudocode* dari *insertion sort*. *Pseudocode* ini dapat dikatakan sebagai notasi algoritma yang mirip dengan kode program asli, sehingga dapat digunakan sebagai referensi untuk menuliskan metode *insertion sort* ke dalam bahasa program mana pun, khususnya bahasa C.

#### Trace pada Insertion Sort

Ada baiknya untuk melakukan *trace* pada notasi algoritma yang telah dibuat untuk memastikan apakah metode algoritma yang digunakan telah disusun dengan benar atau tidak. Berikut adalah *trace* dari metode algoritma pengurutan *insertion sort*.

Tabel 7.1 *Trace Insertion Sort*

|  |  |
| --- | --- |
| Input | int array[6] = {13, 11, 15, 5, 8, 12} |
| Proses | int n = 6  int i, j , key  FOR (i = 1 ; i < n ; i++)  key = array[i]  j = i - 1    WHILE (j >= 0 && array[j] > key)  array[j + 1] = array[j]  j = j - 1  END WHILE |
| Output | array[6] = {5, 8, 11, 12, 13, 15} |

Tabel 7.1 merupakan contoh *trace* dari *insertion sort.* Setelah pengguna memberikan *input array[]*, program akan memproses *array[]* tersebut di dalam iterasi *nested loop* untuk mengurutkan elemen *array[]* dengan metode algoritma *insertion sort*. *Output* yang diharapkan adalah *array[]* yang sudah terurut. Bila ingin memunculkan *output, programmer* dapat menambahkan *syntax* yang sesuai.

### **Flowchart, Pseudocode, dan Trace pada Bubble Sort**

*Bubble sort* merupakan metode algoritma *sorting* yang paling sederhana, yang mana bekerja dengan cara menukar elemen yang berurutan bila berada pada urutan yang salah. Algoritma ini sesuai untuk data yang sedikit, namun tidak dapat digunakan pada data yang besar. Berikut adalah *flowchart* dan *pseudocode* serta *trace* dari metode *bubble sort*.

#### Flowchart pada Bubble Sort

*Flowchart* berguna dalam memvisualisasikan alur algoritma, sehingga tidak hanya dapat membantu seorang *programmer* dalam menyusun programnya, *flowchart* pun dapat digunakan sebagai patokan bagi klien atau pun pengguna agar dapat lebih memahami program atau algoritma tersebut. Berikut adalah *flowchart* dari *bubble sort*.

|  |
| --- |
|  |

Gambar 7.2 *Flowchart Bubble Sort*

Gambar 7.2 merupakan *flowchart* dari metode algoritma pengurutan *bubble sort*. Metode *bubble sort* memanfaatkan iterasi *nested loop,* yang mana akan berjalan selama variabel *i* sebagai patokan iterasi pertama kurang dari *n-1* atau jumlah elemen yang ada pada *array[]*. Iterasi kedua dengan patokan variabel *j*  akan terus berjalan selama *j* kurang dari *n-i-1*. Bila *array[j]* lebih besar dari *array[j+1]*, maka nilai akan ditukar. Algoritma *bubble sort* akan terus membandingkan setiap elemen *array* dan mengulang pengecekan tersebut hingga seluruh elemen dapat dipastikan telah terurut.

#### Pseudocode pada Bubble Sort

Penyusunan *pseudocode* perlu dilakukan untuk menggambarkan metode algoritma ke dalam suatu notasi yang mirip dengan kode program, hanya saja tidak mengikuti *syntax* kode program apapun sehingga dapat digunakan sebagai referensi algoritma untuk menulis kode program dalam bahasa tertentu dengan cara menambahkan *syntax* yang sesuai. Berikut adalah *pseudocode* dari *bubble sort.*

|  |
| --- |
| PROGRAM bubbleSort  BEGIN  DEKLARASI  int array[]  int n = sizeof(array[])  int i, j, temp  FOR (i = 0 ; i < n-1; i++)  FOR (j = 0 ; j < n – i - 1 ; j++)  IF (array[j] > array[j+1])  temp = array[j]  array[j] = array[j+1]  array[j+1] = temp  END IF  END FOR  END FOR  END |

Kode Program 7.2 *Pseudocode Bubble Sort*

Kode Program 7.2 merupakan *pseudocode* dari *bubble sort*. Proses pada *pseudocode* ini telah mencerminkan bagaimana alur algoritma dari *bubble sort* dapat diimplementasikan pada sebuah program. Selanjutnya, *pseudocode* ini dapat digunakan sebagai referensi saat menulis program dengan menambahkan *syntax* yang diperlukan.

#### Trace pada Bubble Sort

Setelah selesai membuat notasi algoritmanya, ada baiknya untuk melakukan *trace* untuk memastikan apakah implementasi yang diberikan pada algoritma tersebut sudah sesuai. Berikut adalah *trace* dari metode algoritma pengurutan *bubble sort*.

Tabel 7.2 *Trace Bubble Sort*

|  |  |
| --- | --- |
| Input | int array[5] = { 5, 1, 4, 2, 8 } |
| Proses | int n = 5  int i, j, temp  FOR (i = 0 ; i < n-1; i++)  FOR (j = 0 ; j < i - n - 1 ; j++)  IF (array[j] > array[j+1])  temp = array[j]  array[j] = array[j+1]  array[j+1] = temp  END IF  END FOR  END FOR |
| Output | array[5] = {1, 2, 4, 5, 8} |

Tabel 7.2 telah menggambarkan *trace* dari *bubble sort* dan telah membuktikan bahwa algoritma ini sesuai. Setelah *input array[]* oleh pengguna, maka program pun dapat memproses *array[]* tersebut di dalam iterasi *bubble sort* hingga dapat memberikan *output* berupa *array[]* yang sudah diurutkan. Bila ingin memunculkan *output* dalam program, maka *programmer* dapat menambahkan *syntax* yang sesuai.

### **Flowchart, Pseudocode, dan Trace pada Quick Sort**

*Quick sort* merupakan metode algoritma pengurutan yang membutuhkan waktu singkat, berbeda dengan *insertion sort* dan *bubble sort*. Algoritma ini mengambil satu elemen sebagai *pivot* dan menyusun ulang elemen-elemen *array* sehingga elemen yang lebih kecil dari *pivot* dipindahkan di kiri *pivot* tersebut dan elemen yang lebih besar dipindahkan ke kanan. Setelahnya, algoritma ini akan mengurutkan kembali elemen-elemen yang telah dipisahkan secara rekursif. Berikut adalah *flowchart, pseudocode,* dan *trace* dari *quick sort*.

#### Flowchart pada Quick Sort

*Flowchart* dapat membantu dalam memahami suatu algoritma dengan lebih baik karena menggunakan suatu visualisasi alur algoritma dan penerapan simbol-simbol yang mudah untuk dimengerti. Memahami *quick sort* akan menjadi lebih mudah dengan menggunakan *flowchart*, terutama pada bagian rekursif. Berikut adalah *flowchart* dari metode *quick sort.*

|  |
| --- |
|  |

Gambar 7.3 *Flowchart Quick Sort*

Gambar 7.3 merupakan *flowchart* dari metode algoritma pengurutan *quick sort*. Metode ini memanfaatkan algoritma rekursif dan memanggil fungsi bernama *partition()* serta prosedur *quick sort* itu sendiri di dalam metode rekursif dan iterasi tersebut. *Partition()* berguna untuk memisahkan tiap elemen sesuai dengan *pivot* yang telah dipilih (dalam algoritma ini, elemen terakhir atau *high* digunakan sebagai *pivot*). Setelahnya, *partition()* akan memberikan nilai indeks elemen dari *pivot* tersebut yang nantinya akan dipakai sebagai patokan dalam prosedur *quick sort* setelahnya.

#### Pseudocode pada Quick Sort

Setelah selesai menyusun *flowchart* untuk memvisualisasikan alur dari metode algoritma pengurutan *quick sort*, *pseudocode* pun dapat disusun untuk memudahkan implementasi algoritma ini ke dalam kode program. *Pseudocode* tidak terikat dengan *syntax* kode program manapun, sehingga dapat digunakan pada seluruh bahasa program dengan penambahan *syntax* yang sesuai. Berikut adalah *pseudocode* dari *quick sort*.

|  |
| --- |
| PROGRAM quickSort  BEGIN  FUNCTION partition(int array[],int low, int high)  BEGIN  DECLARE int i, pivot, j, temp  pivot = array[high]  i = (low-1)  FOR (j = low; j <= high-1; j++)  IF (array[j]<pivot)  i++  temp=array[i]  array[i]=array[j]  array[j]=temp  ENDIF  ENDFOR  temp=array[i+1]  array[i+1]=array[high]  array[high]=temp  return (i+1)  ENDFUNCTION  PROCEDURE quickSort(int array[], int low, int high)  BEGIN  int pi = CALL partition(array, low, high)  IF (low < high)  pi = CALL partition(array, low, high)  CALL quickSort(array, low, pi-1)  CALL quickSort(array, pi+1, high)  ENDIF  ENDPROCEDURE  END |

Kode Program 7.3 *Pseudocode* *Quick Sort*

Kode Program 7.3 merupakan contoh dari penulisan *pseudocode* pada metode algoritma pengurutan *quick sort*. Dengan menggunakan *pseudocode* ini sebagai patokan, maka akan lebih mudah dalam menuliskan *quick sort* ke dalam program dengan hanya menambahkan *syntax* sesuai dengan bahasa program yang digunakan.

#### Trace pada Quick Sort

Setelah selesai menuliskan notasi algoritma *flowchart* dan *pseudocode* dari metode algoritma pengurutan *quick sort,* maka saatnya untuk memastikan kebenaran prosedur dari algoritma tersebut dengan melakukan *trace.* Berikut adalah hasil *trace* dari *quick sort*.

Tabel 7.3 *Trace Quick Sort*

|  |  |
| --- | --- |
| Input | array[5]={5,2,4,8,3} |
| Proses | CALL quickSort(array, 0, 5) |
| Output | array[5]={2,3,4,5,8} |

Tabel 7.3 merupakan *trace* dari metode algoritma pengurutan *quick sort*. Setelah disediakan *input* berupa *array()*, program pun akan memproses *array()* tersebut dengan memanggil prosedur *quickSort()*. Setelah selesai mengurutkan, maka *array()* pun sudah dapat dikeluarkan sebagai *output*. Pengaksesan *output* dapat dilakukan sesuai dengan *syntax* masing-masing bahasa program.

### **Flowchart, Pseudocode, dan Trace pada Sequential Search**

Metode algoritma pencarian *sequential search* merupakan suatu metode pencarian yang sederhana, yaitu mencari *item* yang diinginkan dengan cara membandingkannya dengan seluruh *item* satu per satu dan secara linear. Metode ini cocok untuk digunakan pada data kecil. Berikut adalah *flowchart, pseudocode,* serta *trace* pada *sequential search*.

#### Flowchart pada Sequential Search

*Flowchart* sering digunakan untuk menggambarkan suatu alur serta notasi algoritma sehingga lebih mudah untuk dipahami, baik untuk *programmer* itu sendiri maupun klien dan pengguna lainnya. *Flowchart* metode algoritma pencarian *sequential search* sendiri cukup mudah untuk dimengerti, Berikut adalah *flowchart* dari *sequential search*.

|  |
| --- |
|  |

Gambar 7.4 *Flowchart Sequential Search*

Gambar 7.4 merupakan *flowchart* dari metode algoritma pencarian *sequential search.* Setelah menerima *array[]*, *n* sebagai elemen terakhir dalam *array*, serta *x* sebagai data atau *item* yang ingin dicari, metode ini pun akan menjalankan suatu iterasi dan membandingkan *x* dengan seluruh elemen yang ada. Bila ditemukan, maka program akan memberi tahu letak dari data yang dicari. Metode iterasi ini dilakukan selama *i* kurang dari *n*. Bila tidak ditemukan, maka program akan memberi tahu bahwa data tidak ditemukan.

#### Pseudocode pada Sequential Search

*Pseudocode* pun dapat disusun sebagai referensi penyusunan program yang menggunakan metode *sequential search* ini. Hal ini dilakukan karena *pseudocode* memiliki kemiripan dengan bahasa program dan tidak terikat pada *syntax* apapun, sehingga bersifat universal dan dapat diimplementasikan ke dalam seluruh bahasa program dengan menambahkan *syntax* yang sesuai. Berikut adalah *pseudocode* dari *sequential search*.

|  |
| --- |
| PROGRAM sequentialSearch  BEGIN  PROCEDURE sequentialSearch(int array[], int n, int x)  BEGIN  DECLARE int i, cocok;  FOR ( i = 0; i<n; i++)  IF (array[i]==x)  WRITE "Angka %d ditemukan pada urutan ke-%d\n", x, i+1  cocok++  ENDIF  ENDFOR  IF (cocok==0)  WRITE "Angka tidak ditemukan."  ENDIF  ENDPROCEDURE  END |

Kode Program 7.4 *Pseudocode Sequential Search*

Kode Program 7.4 merupakan *pseudocode* dari metode algoritma pencarian *sequential search*. Dengan menggunakan *pseudocode* ini sebagai patokan atau referensi, maka *programmer* akan dapat dengan mudah mengimplementasikan algoritma ini ke dalam kode programnya dan memodifikasinya sesuai dengan kebutuhan.

#### Trace pada Sequential Search

*Trace* merupakan bagian penting dalam membuat suatu metode algoritma agar dapat memastikan apakah alur yang telah dibuat sudah sesuai atau belum sebelum mengimplementasikannya ke dalam kode program yang asli. Berikut adalah *trace* dari metode algoritma pencarian *sequential search.*

Tabel 7.4 *Trace Sequential Search*

|  |  |
| --- | --- |
| Input | int x = 20  *array[6]={10,12,48,23,20,43}*  int n = 6 |
| Proses | CALL sequentialSearch(array, 6, 20) |
| Output | Angka 20 ditemukan pada urutan ke-5 |

Tabel 7.4 merupakan contoh *trace* pada *sequential search*. Program diberikan *input* berupa angka yang ingin dicari pada *array* apa dan juga banyaknya elemen pada *array* tersebut yang nantinya akan diberikan ke dalam prosedur *sequential search.* Bila angka ditemukan, maka akan diberikan *output* berupa urutan dari angka tersebut dan sebaliknya, Dalam *trace* ini, *sequential search* berhasil menemukan angka *20* pada urutan ke-*5*.

### **Flowchart, Pseudocode, dan Trace pada Binary Search**

Berbeda dengan *sequential search* yang memiliki metode pencarian sederhana, *binary search* adalah suatu metode algoritma pencarian yang cukup kompleks, tetapi memakan waktu sedikit untuk mengurutkan data dalam jumlah besar. Sebelum menggunakan *binary search*, ada baiknya data diurutkan terlebih dahulu dengan metode algoritma pengurutan yang telah dijelaskan sebelumnya. Berikut adalah *flowchart, pseudocode*, serta *trace* dari metode algoritma pencarian *binary search*.

#### Flowchart pada Binary Search

Memahami serta menyusun alur algoritma dari metode pencarian *binary search* akan lebih mudah dengan menggunakan *flowchart* sebagai medium ilustrasinya. Berikut adalah *flowchart* dari metode algoritma pencarian *binary search.*

|  |
| --- |
|  |

Gambar 7.5 *Flowchart Binary Search*

Gambar 7.5 merupakan *flowchart* dari metode algoritma pencarian *binary search*. Secara singkat, pencarian dimulai dengan menentukan indekstengah sebagai kunci pencarian. Bilamana ternyata indeks tengah tersebut sama dengan angka yang dicari, maka akan diberikan *output* berupa urutan dari angka tersebut di dalam *array*. Bila tidak, maka akan dicari kembali dengan cara menentukan apakah angka tersebut lebih besar dari indeks tengah atau tidak. Bila lebih besar, maka pencarian dilanjutkan pada urutan di kanan indeks dan bagian kiri diabaikan, begitu pula sebaliknya. Pencarian pun dilakukan kembali dengan menentukan indeks tengah dan terus berlanjut hingga seluruh angka ditemukan atau mencapai akhir indeks.

#### Pseudocode pada Binary Search

Cara lain untuk mengilustrasikan alur algoritma selain *flowchart* adalah dengan menggunakan *pseudocode.* Notasi algoritma *pseudocode* hampir menyerupai sebuah kode program, tapi ditulis dengan bahasa yang tidka berpatokan pada *syntax* apapun dan mudah dimengerti oleh manusia. *Pseudocode* dapat diimplementasikan ke dalam bahasa program apa pun dengan menambahkan *syntax* yang sesuai. Berikut adalah *pseudocode* untuk metode algoritma pencarian *binary search*.

|  |
| --- |
| PROGRAM binarySearch  BEGIN  PROCEDURE binarySearch(int array[], int kiri, int kanan, int x)  BEGIN  DECLARE  int bscocok  int urutanAngka[bscocok]  WHILE (kiri<=kanan)  int mid = kiri + (kanan-kiri)/2  IF (array[mid]==x)  urutanAngka[bscocok] = mid  WRITE "Angka %d ditemukan pada urutan ke-%d\n", x, urutanAngka[bscocok]+1  bscocok++  FOR ( int kiri1=mid+1;kiri1<kanan+1 && array[kiri1]==x; kiri1++)  urutanAngka[bscocok] = kiri1  WRITE "Angka %d ditemukan pada urutan ke-%d\n", x, urutanAngka[bscocok]+1  bscocok++  ENDFOR  FOR (int kanan1 = mid - 1; kanan1 >= 0 && array[kanan1] == x; kanan1--)  urutanAngka[bscocok] = kanan1;  WRITE "Angka %d ditemukan pada urutan ke-%d\n", x, urutanAngka[bscocok]+1  bscocok++  ENDFOR  BREAK  ELSE IF (array[mid]>x) {kanan = mid -1;}  ELSE {bscocok=0; kiri = mid +1;}  ENDIF  ENDWHILE  IF (bscocok==0)  WRITE "Angka tidak ditemukan."  ENDIF  ENDPROCEDURE  END |

Kode Program 7.5 *Pseudocode Binary Search*

Kode Program 7.5 merupakan *pseudocode* dari metode algoritma pencarian *binary search*, yang mana memiliki alur algoritma yang sama dengan *flowchart* sebelumnya. *Pseudocode* ini nantinya dapat menjadi patokan dalam implementasi algoritma *binary search* ke dalam kode program dengan menambahkan *syntax* yang sesuai beserta prosedur lain untuk mengurutkan *array* terlebih dahulu.

#### Trace pada Binary Search

Setelah selesai menyusun alur dan notasi algoritma, maka *trace* pun dapat dilakukan untuk memastikan apakah alur dan notasi algoritma yang dibuat sudah sesuai atau tidak. Berikut adalah *trace* pada metode algoritma pencarian *binary search.*

Tabel 7.5 *Trace Binary Search*

|  |  |
| --- | --- |
| Input | int x = 16  array[7]={2,13,16,19.20,23,29} |
| Proses | CALL binarySearch(array, 0, 6, 16) |
| Output | Angka 16 ditemukan pada urutan ke-3 |

Tabel 7.5 merupakan contoh *trace* pada metode algoritma pencarian *binary search*. Setelah diberikan angka yang ingin dicari pada suatu *array* yang sudah diurutkan, prosedur *binary search* pun dapat dipanggil. Bilamana angka tersebut ditemukan maka akan diberikan *output* berupa urutan dari angka tersebut. Bila tidak, maka akan diberikan *output* instruksi bahwa angka tidak ditemukan.

## Flowchart, Pseudocode, dan Trace Soal Praktikum Modul III

*Flowchart* merupakan suatu gambaran atau ilustrasi dari tahapan-tahapan alur algoritma yang menggunakan simbol-simbol khusus dalam ilustrasinya. Penggunaan *flowchart* sering ditemukan dalam pembuatan suatu notasi algoritma dikarenakan dapat memberikan ilustrasi yang jelas mengenai algoritma yang akan digunakan.

*Pseudocode* merupakan suatu notasi algoritma yang ditulis menyerupai suatu kode program. Berbeda dengan kode program asli, *pseudocode* hanya ditulis sebagai notasi algoritma yang dapat dimengerti manusia tapi tidak dapat dijalankan dalam komputer. Penggunaan *pseudocode* sangat membantu dalam menyusun kode program asli dengan cara memodifikasi *pseudocode* tersebut sesuai dengan bahasa pemrograman yang digunakan.

*Trace* merupakan suatu tahapan akhir pada penyusunan suatu alur algoritma, yaitu suatu tahapan pelacakan alur dengan memastikan apakah alur algoritma sudah sesuai atau belum. *Trace* dilakukan dari bagian *input,* proses, serta *output*. Berikut adalah *flowchart, pseudocode,* beserta *trace* dari soal praktikum Modul III.

### **Flowchart, Pseudocode, dan Trace Program Sorting dan Searching**

Program ini merupakan salah satu contoh implementasi program *sorting* dan *searching* ke dalam kode program berbahasa C. Program ini pun diberikan pencatat waktu untuk menghitung waktu proses masing-masing metode pengurutan dan pencarian data. Program memiliki pembangkit bilangan acak yang mana dapat membuat suatu bilangan acak sebanyak 1000, 16.000, dan 64.000 data, yang selanjutnya akan diurutkan dan dilakukan pencarian pada data tersebut. Proses tersebut lantas dicatat waktu prosesnya. Berikut ini adalah *flowchart, pseudocode,* serta *trace* dari program *sorting* dan *searching*.

#### Flowchart Program Sorting dan Searching

*Flowchart* merupakan suatu metode dalam mendeskripsikan notasi algoritma yang menggunakan simbol-simbol tertentu dan alur algoritma disusun secara linear. *Flowchart* biasanya digunakan baik dalam menyusun, mempelajari, atau mengomunikasikan suatu alur algoritma dengan lebih mudah. Berikut adalah *flowchart* program *sorting* dan *searching.*

##### Flowchart Int Main dan Menu Awal

Pada sebuah program berbahasa C, fungsi *int main()* bertindak sebagai fungsi utama. Fungsi ini merupakan awal dan akhir dari program tersebut, yang mana akan mengembalikan nilai *integer* pada saat program berakhir. Pada program ini, prosedur *menuAwal()* dipanggil di dalam fungsi *int main()*. Berikut adalah *flowchart* dari *int main()* dan *menuAwal().*

|  |
| --- |
|  |

Gambar 8.1 *Flowchart Int Main* dan *Menu Awal*

Gambar 8.1 merupakan *flowchart* dari fungsi *int main()* dan prosedur *menuAwal().* Pada prosedur *menuAwal(),* pengguna akan diminta untuk memilih menu yang diinginkan, yaitu *sorting, searching*, kredit, dan keluar. Masing-masing pilihan dari pengguna akan memanggil prosedur yang berbeda. Bila pengguna memberikan *input* pilihan yang salah, maka program akan memberikan peringatan dan meminta pengguna untuk memasukkan pilihan yang valid.

##### Flowchart Prosedur Sorting

Pilihan pertama adalah menu *sorting()*. Pada prosedur ini, akan dilakukan *sorting* serta pencatatan waktu durasi pelaksanaan *sorting.* Berikut adalah *flowchart* dari prosedur *sorting().*

|  |
| --- |
|  |

Gambar 8. 2 *Flowchart* Prosedur *Sorting*

Gambar 8.2 merupakan *flowchart* dari prosedur *sorting().* Pengguna akan kembali diminta untuk memasukkan pilihan angka, yaitu 1.000, 16.000, dan 64.000 atau kembali ke menu awal dan keluar dari program. Sesuai dengan pilihan angka pengguna, bilangan tersebut akan di-*assign* ke dalam variabel n yang nantinya akan diberikan pada *array* sebagai batas elemen dan memanggil *bilAcak()* dan *printArray(), array* disalin dan di-*print* ke layar untuk menunjukkan angka sebelum diurut. Lalu, setiap prosedur *sorting* akan diberikan masing-masing *array* beserta nilai dari variabel n sebagai argumen untuk kemudian melakukan *sorting.* Waktu akan dicatat dan hasilnya akan ditampilkan sebagai *output*.

##### Flowchart Prosedur Searching()

Pada prosedur *searching()*, proses pencarian akan dilakukan sesuai dengan angka yang diminta pengguna dan akan menampilkan hasil pencarian serta waktu yang dibutuhkan untuk melakukan pencarian tersebut. Berikut adalah *flowchart* dari prosedur *searching()*.

|  |
| --- |
|  |

Gambar 8.3 *Flowchart* Prosedur *Searching*

Gambar 8.3 merupakan *flowchart* dari prosedur *searching()*. Pada prosedur ini, pengguna akan diminta untuk memilih banyak bilangan yang ingin ditampilkan sebagai data acak, yaitu sebanyak 1.000, 16.000, serta 64.000 dan ada pula pilihan kembali ke menu dan keluar. Setelah memilih, bilangan tersebut akan di-*assign* ke dalam variabel *n.* Dikarenakan pada prosedur *binary search* angka harus diurutkan terlebih dahulu, maka setelah menjalankan *bilAcak()* dan *printAngka()* akan dijalankan prosedur *insertionArray()*. Setelahnya, seluruh argumen yang dibutuhkan akan diberikan pada masing-masing prosedur pencarian beserta memulai pencatatan waktu untuk masing-masing prosedur. Hasil pencarian dan pencatatan waktu akan diberikan sebagai *output*.

##### Flowchart Prosedur Bilangan Acak

Pada program ini, pembangkit bilangan acak diperlukan untuk mendapatkan nilai *array* yang akan digunakan pada tiap metode. Berikut adalah *flowchart* dari prosedur *bilAcak().*

|  |
| --- |
|  |

Gambar 8.4 *Flowchart* Prosedur Bilangan Acak

Gambar 8.4 merupakan *flowchart* dari prosedur *bilAcak()*. Fungsi *srand()* digunakan untuk inisiasi generator bilangan random, yang mana merupakan titik permulaan dari pembuatan bilangan random tersebut berdasarkan argumen yang diberikan di dalamnya. Setelahnya, fungsi *rand()* akan dipanggil untuk menggenerasi bilangan acak sesuai permintaan.

##### Flowchart Print Angka dan Copy

Prosedur *printAngka()* digunakan untuk memberikan *output array,* sementara *copy* digunakan untuk menyalin satu *array* ke *array*  lainnya. Kedua prosedur ini berguna dalam proses prosedur pengurutan dan pencarian. Berikut adalah *flowchart* dari prosedur *printAngka()* dan *copy().*

|  |
| --- |
|  |

Gambar 8.5 *Flowchart* Prosedur *Print* Angka dan *Copy*

Gambar 8.5 merupakan *flowchart* dari prosedur *printAngka* dan *copy()*. Pada prosedur *printAngka(),* dilakukan proses iterasi untuk mencetak seluruh elemen *array*. Lalu, pada prosedur *copy()*, iterasi kembali dilakukan untuk menyalin seluruh *array* sumber ke dalam *array* target.

##### Flowchart Prosedur Insertion Sort

Prosedur ini mengimplementasikan algoritma *insertionSort()* yang telah dibahas sebelumnya. Prosedur ini akan dipanggil di dalam prosedur *sorting()* bersamaan dengan prosedur pengurutan lainnya Berikut adalah *flowchart* dari prosedur *insertionSort()*.

|  |
| --- |
|  |

Gambar 8.6 *Flowchart* Prosedur *Insertion Sort*

Gambar 8.6 merupakan *flowchart* dari prosedur *insertionSort().* Prosedur ini akan memasukkan angka-angka yang masih ada di dalam *unsorted sub-array* ke dalam *sorted sub-array* dengan menggunakan metode iterasi *nested loop* seperti pada *flowchart.* Penjelasan lengkap dapat dilihat pada sub bab No. 6 dan 7.

##### Flowchart Prosedur Bubble Sort

Prosedur *bubbleSort()* merupakan salah satu prosedur pengurutan yang digunakan pada program ini bersamaan dengan prosedur lainnya. Prosedur *bubbleSort()* sendiri dapat terbilang cukup sederhana. Berikut adalah *flowchart* dari prosedur *bubbleSort().*

|  |
| --- |
|  |

Gambar 8.7 *Flowchart* Prosedur *Bubble Sort*

Gambar 8.7 merupakan *flowchart* dari prosedur *bubbleSort().* Dapat dilihat bahwa prosedur ini memanfaatkan algoritma yang telah dijelaskan sebelumnya. Setelah menerima argumen berupa *array[]* dan *n,* prosedur ini pun akan mengurutkan seluruh elemen pada *array[]* tersebut. Penjelasan lengkap dari algoritma yang digunakan dapat disimak pada nomor 6 dan 7.

##### Flowchart Prosedur Quick Sort dan Partition

Prosedur *quickSort()* merupakan salah satu dari metode pengurutan yang digunakan dalam program ini. Prosedur ini juga memiliki satu prosedur lain yaitu *partition()* yang berguna dalam memisahkan *array* sesuai dengan *pivot-*nya. Berikut adalah *flowchart* dari *quickSort().*

|  |
| --- |
|  |

Gambar 8.8 *Flowchart* Prosedur *Quick Sort* dan *Partition*

Gambar 8.8 merupakan *flowchart* prosedur *quickSort()* dan *partition().* Dapat dilihat bahwa prosedur ini menggunakan algoritma rekursif untuk mengurutkan angka pada *array* yang diberikan sebagai argumen dengan *n* sebagai patokan dalam menjalankan prosedur ini. Penjelasan lengkap mengenai algoritma yang digunakan dapat disimak pada jawaban nomor 6 dan 7.

##### Flowchart Prosedur Sequential Search

Prosedur *sequentialSearch()* merupakan salah satu prosedur pencarian yang digunakan pada program ini. Prosedur *sequentialSearch()* memanfaatkan algoritma pencarian yang sebelumnya sudah dijelaskan. Berikut adalah *flowchart* dari prosedur *sequentialSearch().*

|  |
| --- |
|  |

Gambar 8.9 *Flowchart* Prosedur *Sequential Search*

Gambar 8.9 merupakan *flowchart* dari prosedur *sequentialSearch()*. Prosedur ini akan menerima argumen berupa *array[], n* sebagai banyak elemen *array ,* dan *x* sebagai angka yang ingin ditemukan. Prosedur ini lantas akan mencari angka tersebut dengan algoritma *sequential search* dan lalu menampilkan hasilnya. Penjelasan lengkap dapat dilihat pada nomor 6 dan 7.

##### Flowchart BinarySearch()

Prosedur *binarySearch()* merupakan salah satu dari metode pencarian yang diterapkan pada prosedur ini bersamaan dengan *sequentialSearch()*. Prosedur ini dapat dikatakan cukup kompleks. Berikut adalah *flowchart* dari prosedur *binarySearch*().

|  |
| --- |
|  |

Gambar 8.10 *Flowchart* Prosedur *Binary Search*

Gambar 8.10 merupakan *flowchart* contoh dari implementasi algoritma *binary search* ke dalam prosedur program. Prosedur *binarySearch()* akan menerima argumen berupa *array*, *kiri* untuk elemen *array* paling kiri, *kanan* untuk elemen *array* paling kanan, dan *x* sebagai angka yang dicari. Penjelasan lebih lengkap dapat disimak pada soal nomor 6 dan 7.

##### Flowchart Repeat(), Kredit(), dan Keluar()

Prosedur *repeat(), kredit(),* dan *keluar()* adalah prosedur tambahan yang digunakan pada program ini dan tidak bersifat wajib untuk ditambahkan. Berikut adalah *flowchart* dari prosedur *repeat(), kredit(),* dan *keluar().*

|  |
| --- |
|  |

Gambar 8.11 *Flowchart* Prosedur *Repeat*, *Kredit*, dan *Keluar*

Gambar 8.11 adalah *flowchart* dari prosedur *repeat(), kredit(), dan keluar().* Prosedur *repeat()* digunakan untuk memberikan pilihan pada pengguna, apakah mereka ingin mengulangi program atau tidak. Prosedur *kredit()* digunakan untuk menampilkan nama kelompok. Prosedur *keluar()* digunakan untuk keluar dari program.

#### Pseudocode Program Sorting dan Searching

*Pseudocode* merupakan suatu kode palsu yang digunakan untuk mengimplementasikan suatu algoritma ke dalam notasi yang mirip dengan suatu kode program namun lebih mudah dimengerti oleh manusia. Berikut adalah *pseudocode* dari program *sorting* dan *searching.*

|  |
| --- |
| PROGRAM sortingSearching  BEGIN  FUNCTION validasiInt(int \*var)  BEGIN  fflush(stdin)  DECLARE  char buffer[1024]  char cek  IF (fgets(buffer, sizeof(buffer), stdin) != NULL)  IF (sscanf(buffer, "%d %c", var, &cek) == 1)  return \*var  ENDIF  ENDIF  return 0  ENDFUNCTION  PROCEDURE validasiInput(int \*var)  BEGIN  WHILE (1)  fflush(stdin)  IF CALL (validasiInt(var))  BREAK  WRITE "Masukan salah. Mohon masukkan input sesuai arahan."  WRITE "Masukkan input: "  ENDWHILE  ENDPROCEDURE  PROCEDURE validasiNegatif(int \*var)  BEGIN  WHILE (1)  IF CALL validasiInput(var);  fflush(stdin);  ENDIF  IF (\*var > 0)  BREAK  ENDIF  WRITE "Masukan salah. Mohon masukkan angka positif."  WRITE "Masukkan input: "  ENDWHILE  ENDPROCEDURE  PROCEDURE validasiMenu(int \*var, int batas1, int batas2)  BEGIN  WHILE (1)  CALL validasiInput(var);  fflush(stdin);  IF (batas1 <= \*var && \*var <= batas2)  BREAK;  WRITE "Masukan salah. Mohon masukkan input sesuai arahan."  WRITE "Masukkan input: "  ENDWHILE  ENDPROCEDURE  PROCEDURE bilAcak(int array[], int n)  BEGIN  DECLARE time\_t t  srand((unsigned) time(&t))  FOR (int i = 0; i < n; i++)  array[i] = rand()  ENDFOR  ENDPROCEDURE  PROCEDURE printAngka(int array[], int n)  BEGIN  FOR (int i=0; i<n; i++)  WRITE "Angka dengan urutan ke-%d = %d\n", i+1, array[i])  ENDFOR  ENDPROCEDURE  PROCEDURE copy(int source[], int target[], int n)  BEGIN  FOR (int i=0; i<n; i++)  target[i]=source[i]  ENDPROCEDURE  PROCEDURE insertionSort(int array[], int n)  BEGIN  DECLARE int i, j, key  FOR (i = 1; i < n; i++)  key = array[i]  j = i - 1  WHILE (j >= 0 && array[j] > key) {  array[j + 1] = array[j]  j--  ENDWHILE  array[j + 1] = key  ENDFOR  ENDPROCEDURE  PROCEDURE bubbleSort(int array[], int n)  BEGIN  DECLARE int i, j, temp;  FOR (i = 0 ; i < n-1; i++)  FOR (j = 0 ; j < n - i-1;j++)  IF(array[j] > array[j+1])  temp = array[j]  array[j] = array[j+1]  array[j+1] = temp  ENDIF  ENDFOR  ENDFOR  ENDPROCEDURE  FUNCTION partition(int array[],int low, int high)  BEGIN  DECLARE int i, pivot, j, temp  pivot = array[high]  i = (low-1)  FOR (j = low; j <= high-1; j++)  IF (array[j]<pivot)  i++  temp=array[i]  array[i]=array[j]  array[j]=temp  ENDIF  ENDFOR  temp=array[i+1]  array[i+1]=array[high]  array[high]=temp  return (i+1)  ENDFUNCTION  PROCEDURE quickSort(int array[], int low, int high)  BEGIN  int pi = CALL partition(array, low, high)  IF (low < high)  pi = CALL partition(array, low, high)  CALL quickSort(array, low, pi-1)  CALL quickSort(array, pi+1, high)  ENDIF  ENDPROCEDURE  PROCEDURE sorting()  BEGIN  DECLARE int pilihan, n  WRITE "ALGORITMA SORTING"  WRITE "Metode Pengurutan Bilangan Acak"  WRITE "Mohon pilih banyak bilangan acak"  WRITE "[1] 1000 bilangan"  WRITE "[2] 16000 bilangan"  WRITE "[3] 64000 bilangan"  WRITE "Menu Lainnya"  WRITE "[4] Kembali ke menu awal"  WRITE "[5] Keluar dari program"  WRITE "Pilih menu/bilangan yang diinginkan:"  CALL validasiMenu(&pilihan, 1, 5)  IF (pilihan == 1)  n = 1000  ELSEIF (pilihan == 2)  n = 16000  ELSEIF (pilihan == 3)  n = 64000  ELSEIF (pilihan == 4)  CALL system("cls")  CALL menuAwal()  ELSEIF (pilihan == 5)  CALL system("cls")  CALL keluar()  ENDIF  DECLARE  int arrayI[n]  int arrayB[n]  int arrayQ[n]  CALL bilAcak(arrayI, n);  WRITE "Elemen pada array[%d] sebelum diurutkan:", n  CALL printAngka(arrayI, n)  CALL copy(arrayI, arrayB, n)  CALL copy(arrayI, arrayQ, n)    DECLARE  clock\_t mulai, selesai  double waktuI, waktuB, waktuQ = 0  mulai = clock()  CALL insertionSort(arrayI, n)  selesai = clock()  waktuI = (double)((selesai - mulai)/(CLOCKS\_PER\_SEC))  mulai = clock()  CALL bubbleSort(arrayB, n)  selesai = clock()  waktuB = (double)((selesai - mulai)/(CLOCKS\_PER\_SEC))  mulai = clock()  CALL quickSort(arrayQ, 0, n )  selesai = clock()  waktuQ = (double)((selesai - mulai)/(CLOCKS\_PER\_SEC))  WRITE "Elemen pada array[%d] sebelum diurutkan:", n  CALL printAngka(arrayQ, n);  WRITE "Waktu yang diperlukan untuk mengurutkan data:"  WRITE "> Insertion sort : %lf detik\n", waktuI  WRITE "> Bubble sort : %lf detik\n", waktuB  WRITE "> Quick sort : %lf detik\n", waktuQ    IF (waktuI == waktuB)  WRITE "Waktu Insertion Sort sama dengan Bubble Sort"  ENDIF  IF (waktuI == waktuQ)  WRITE "Waktu Insertion Sort sama dengan Quick Sort"  ENDIF  IF (waktuB == waktuQ)  WRITE "Waktu Bubble Sort sama dengan Quick Sort"  ENDIF  IF (waktuI > waktuB && waktuI > waktuQ)  WRITE "Insertion Sort memakan waktu paling banyak"  IF (waktuB > waktuI && waktuB > waktuQ)  WRITE Bubble Sort memakan waktu paling banyak"  ENDIF  IF (waktuQ > waktuI && waktuQ > waktuB)  WRITE "Quick Sort memakan waktu paling banyak"  ENDIF  IF (waktuI < waktuB && waktuI < waktuQ)  WRITE Insertion Sort memakan waktu paling sedikit"  IF (waktuB<waktuQ)  WRITE "Bubble sort lebih cepat dari Quick Sort"  ELSE IF (waktuQ<waktuB)  WRITE "Quick sort lebih cepat dari Bubble Sort"  ENDIF  ENDIF  IF (waktuB < waktuI && waktuB < waktuQ)  WRITE "Bubble Sort memakan waktu paling sedikit"  IF (waktuI<waktuQ)  WRITE "Insertion sort lebih cepat dari Quick Sort"  ELSE IF (waktuQ<waktuI)  WRITE "Quick sort lebih cepat dari Insertion Sort"  ENDIF  ENDIF  IF (waktuQ < waktuI && waktuQ < waktuB)  WRITE "Quick Sort memakan waktu paling sedikit"  IF (waktuI<waktuB)  WRITE "Insertion sort lebih cepat dari Bubble Sort"  ELSE IF (waktuB<waktuI)  WRITE "Bubble sort lebih cepat dari Insertion Sort"  ENDIF  ENDIF  ENDPROCEDURE  PROCEDURE sequentialSearch(int array[], int n, int x)  BEGIN  DECLARE int i, cocok;  FOR ( i = 0; i<n; i++)  IF (array[i]==x)  WRITE "Angka %d ditemukan pada urutan ke-%d\n", x, i+1  cocok++  ENDIF  ENDFOR  IF (cocok==0)  WRITE "Angka tidak ditemukan."  ENDIF  ENDPROCEDURE  PROCEDURE binarySearch(int array[], int kiri, int kanan, int x)  BEGIN  DECLARE  int bscocok  int urutanAngka[bscocok]  WHILE (kiri<=kanan)  int mid = kiri + (kanan-kiri)/2  IF (array[mid]==x)  urutanAngka[bscocok] = mid  WRITE "Angka %d ditemukan pada urutan ke-%d\n", x, urutanAngka[bscocok]+1  bscocok++  FOR ( int kiri1=mid+1;kiri1<kanan+1 && array[kiri1]==x; kiri1++)  urutanAngka[bscocok] = kiri1  WRITE "Angka %d ditemukan pada urutan ke-%d\n", x, urutanAngka[bscocok]+1  bscocok++  ENDFOR  FOR (int kanan1 = mid - 1; kanan1 >= 0 && array[kanan1] == x; kanan1--)  urutanAngka[bscocok] = kanan1;  WRITE "Angka %d ditemukan pada urutan ke-%d\n", x, urutanAngka[bscocok]+1  bscocok++  ENDFOR  BREAK  ELSE IF (array[mid]>x) {kanan = mid -1;}  ELSE {bscocok=0; kiri = mid +1;}  ENDIF  ENDWHILE  IF (bscocok==0)  WRITE "Angka tidak ditemukan."  ENDIF  ENDPROCEDURE  PROCEDURE searching()  BEGIN  DECLARE  int pilihan, n, cari;  WRITE "ALGORITMA SEARCHING"  WRITE "Metode Pencarian Bilangan Acak"  WRITE "Mohon pilih banyak bilangan acak"  WRITE "[1] 1000 bilangan"  WRITE "[2] 16000 bilangan"  WRITE "[3] 64000 bilangan"  WRITE "Menu Lainnya"  WRITE "[4] Kembali ke menu awal"  WRITE "[5] Keluar dari program"  WRITE "Pilih menu/bilangan yang diinginkan:"  CALL validasiMenu(&pilihan, 1, 5)  IF (pilihan == 1)  n = 1000  ELSEIF (pilihan == 2)  n = 16000  ELSEIF (pilihan == 3)  n = 64000  ELSEIF (pilihan == 4)  CALL system("cls")  CALL menuAwal()  ELSEIF (pilihan == 5)  CALL system("cls")  CALL keluar()  ENDIF  DECLARE  clock\_t mulai, selesai  double waktuS, waktuBs = 0  int array[n]  CALL bilAcak(array, n)  CALL printAngka(array, n)  CALL insertionSort(array, n)  WRITE "Angka yang sudah diurutkan:\n", n  CALL printAngka(array, n)  WRITE "Masukkan angka yang diinginkan: "  CALL validasiNegatif(&cari)  WRITE "Pencarian dengan Sequential Search:\n", n  mulai = clock()  CALL sequentialSearch(array, n, cari)  selesai = clock()  waktuS = (double)(selesai-mulai)/CLOCKS\_PER\_SEC  WRITE "Pencarian dengan Binary Search:\n", n  mulai = clock()  CALL binarySearch(array, 0, n-1, cari)  selesai = clock()  waktuBs = (double)(selesai-mulai)/CLOCKS\_PER\_SEC  WRITE "Waktu yang diperlukan untuk mengurutkan data:\n", n  WRITE "> Sequential Search : %lf detik\n", waktuS  WRITE "> Binary Search : %lf detik\n", waktuBs  IF (waktuS == waktuBs)  WRITE "Waktu Sequential Search sama dengan Binary Search"  ELSEIF(waktuS < waktuBs)  WRITE "Waktu Sequential Search lebih cepat dari Binary Search"  ELSEIF (waktuS > waktuBs)  WRITE "Waktu Binary Search lebih cepat dari Sequential Search"  ENDIF  ENDPROCEDURE  PROCEDURE repeat()  BEGIN  DECLARE  int pilihanUlang;  WRITE "Apakah Anda ingin kembali ke menu utama?"  WRITE "Ketik [1] untuk 'Ya' dan [2] untuk 'Tidak' :"  CALL validasiMenu(&pilihanUlang, 1, 2)  IF (pilihanUlang==1)  system("cls")  CALL menuAwal()  ELSEIF (pilihanUlang==2){  system("cls")  keluar()  ENDIF  ENDPROCEDURE  PROCEDURE kredit()  BEGIN  WRITE "K E L O M P O K 2 3"  WRITE "PROGRAM METODE SORTING DAN SEARCHING"  WRITE "Nama Anggota Kelompok"  WRITE "205551005 | Gede Made Rapriananta Pande"  WRITE "2205551003 | Ida Bagus Paalakaa RNB"  WRITE "2205551069 | Ni Kadek Ari Diah Lestari"  WRITE "2205551072 | I Gusti Ayu Krisna Kusuma Dewi"  WRITE "2205551076 | Kadek Yogi Dwi Putra Utama"  WRITE "2205551079 | Anak Agung Indi Kusuma Putra"    system("pause")  system("cls")  CALL menuAwal()  ENDPROCEDURE  PROCEDURE keluar()  BEGIN  WRITE "TERIMA KASIH"  WRITE "Sampai jumpa di lain waktu!"  exit(0)  ENDPROCEDURE  PROCEDURE menuAwal()  BEGIN  DECLARE  int pilihan  WRITE "SORTING DAN SEARCHING"  WRITE "Algoritma Metode Sorting dan Searching"  WRITE "Pilih Metode yang Diinginkan"  WRITE "[1] Metode Sorting"  WRITE "[2] Metode Searching"  WRITE "Menu Lainnya"  WRITE "[3] Kredit"  WRITE "[4] Keluar dari program"  WRITE "Pilih menu yang diinginkan: "  CALL validasiMenu(&pilihan, 1, 5)  CALL validasiMenu(&pilihan, 1, 5)  IF (pilihan == 1)  system("cls")  CALL sorting()  ELSEIF (pilihan == 2)  system("cls")  CALL searching()  ELSEIF (pilihan == 3)  system("cls")  CALL kredit()  ELSEIF (pilihan == 4)  CALL system("cls")  CALL keluar()  ENDIF  CALL repeat()  ENDPROCEDURE  FUNCTION main(void)  BEGIN  CALL menuAwal()  return 0  END  END |

Kode Program 8.1 *Pseudocode* Program *Sorting* dan *Searching*

Kode Program 8.1 merupakan *pseudocode* dari program *sorting* dan *searching.* Selanjutnya, *pseudocode* dapat dijadikan suatu acuan dalam membuat kode program asli dari program ini dengan lebih matang.

#### Trace Program Sorting dan Searching

*Trace* adalah suatu simulasi eksekusi algoritma untuk memastikan secara manual bahwa algoritma tersebut bekerja dengan benar. *Trace* menggunakan *trace table* yang mencatat *input*, proses, serta *output* yang diekspetasikan dari algoritma tersebut.Kegiatan ini berguna untuk menemukan kesalahan di dalam logika algoritma tersebut. Berikut ini adalah hasil *trace* dari program *sorting* dan *searching.*

Tabel 8.1 *Trace* Program *Sorting* dan *Searching*

|  |  |
| --- | --- |
| *Input* | pilihan = 1  pilihan = 3  pilihanUlang = 1  pilihan = 2  pilihan = 3  cari = 32762  pilihanUlang = 2 |
| Proses | CALL menuAwal()  CALL sorting()  CALL repeat()  CALL menuAwal()  CALL searching()  CALL repeat()  CALL keluar() |
| *Output* | Instruksi pilih menu  Instruksi pilih  Elemen pada array[64000] sebelum diurutkan  Output list elemen dan nilai elemen  Elemen pada array[64000] setelah diurutkan  Output list elemen dan nilai elemen  Waktu yang diperlukan untuk mengurutkan data:  Insertion Sort: 4.000000 detik  Bubble Sort: 13.000000 detik  Quick Sort: 0.000000 detik  Bubble Sort memakan waktu paling banyak  Quick Sort memakan waktu paling sedikit  Insertion sort lebih cepat dari Bubble Sort  Kembali ke menu utama?  Angka yang sudah diurutkan:  Output list elemen dan nilai elemen  Instruksi masukkan angka yang diinginkan  Pencarian dengan Sequential Search:  Angka 32762 ditemukan pada urutan ke-63991  Pencarian dengan Binary Search:  Angka 32762 ditemukan pada urutan ke-63991  Waktu yang diperlukan untuk mengurutkan data:  Sequential Search : 0.000000 detik  Binary Search : 0.000000 detik  Kembali ke menu utama?  Terima kasih |

Tabel 8.1 di atas adalah contoh dari pelaksanaan *trace* untuk program *sorting* dan *searching*. Pengguna meminta untuk melakukan *sorting* pada 64.000 data dan program memproses *input* dari pengguna sebagaimana dipaparkan pada baris *proses* di atas. Program pun memberikan urutan elemen *array* setelah dan sebelum diurutkan serta memberikan hasil dari total waktu pengurutan seluruh algoritma metode pengurutan atau *sorting*. Total waktu dari tiap pengurutan pun dibandingkan dan hasilnya ditunjukkan pada layar pengguna.

Pengguna lalu memilih untuk melakukan *searching* pada 64.000 data dan memasukkan data yang ingin dicari. Program lantas memberikan hasil pencarian menggunakan *sequential search* dan *binary search* serta memberi tahu hasil perhitungan waktu dari tiap metode. Program lalu ditutup.

### **Flowchart, Pseudocode, dan Trace Program Matriks**

Program ini dirancang agar memiliki kemampuan untuk melakukan operasi matematika pada matriks seperti operasi perkalian, penjumlahan, dan *transpose*. Berikut adalah *flowchart*, *pseudocode*, serta *trace* dari program matriks.

#### Flowchart Program Matriks

Diagram alir atau yang sering disebut *flowchart* adalah sebuah digram yang menjelaskan secara detail langkah-langkah atau tahapan berjalannya sebuah program. *Flowchart* memiliki banyak simbol yang masing-masing memiliki fungsinya sendiri. *Flowchart* dari program perhitungan matriks dapat dilihat pada gambar di bawah ini.

|  |
| --- |
|  |

Gambar 8.12 *Flowchart Int Main* Program Matriks

Gambar 8.12 di atas merupakan *flowchart* dari prosedur *main* dan pada program matriks di mana *flowchart main* dari program perhitungan matriks yang dimana pada program ini akan memanggil beberapa fungsi untuk menjalankan program perhitungan matriks,. Untuk *sub-flowchart* menu dapat dilihat pada gambar di bawah ini.

|  |
| --- |
|  |

Gambar 8.13 *Flowchart* Subproses *Menu*

Gambar 8.13 dijelaskan sebuah *sub-flowchart* yang dimana merupakan prosedur dari menu yang ada pada program matriks. Untuk penjelasan *sub-flowchart* dari prosedur operasi dapat dilihat pada gambar di bawah ini.

|  |
| --- |
|  |

Gambar 8.14 *Flowchart* Subproses Operasi

Gambar 8.14 merupakan *sub-flowchart* dari prosedur operasi yang ada pada program matriks sebagai petunjuk jalannya proses penjumlahan pada program matriks ini. Untuk *sub-flowchart* dari prosedur matriks kali dapat dilihat pada gambar di bawah ini.

|  |
| --- |
|  |

Gambar 8.15 *Flowchart* Subproses Matriks Kali

Gambar 8.15 merupakan *sub-flowchart* dari prosedur matriks kali dari program matriks, sama dengan *flowchart* subprogram perkalian matriks dan penjumlahan matriks, subprogram ini berfungsi untuk melakukan perhitungan *transpose* matriks. Untuk sub *flowchart* prosedur matriks jumlah dapat dilihat pada gambar di bawah ini.

|  |
| --- |
|  |

Gambar 8.16 *Flowchart* Subproses Matriks Jumlah

Gambar 8.16 di atas merupakan *sub-flowchart* dari prosedur matriks jumlah dari program matriks yang berfungsi sebagai petunjuk jalannya proses penjumlahan pada program matriks ini. Untuk *sub-flowchart* prosedur matriks perkalian dapat dilihat pada gambar di bawah ini.

|  |
| --- |
|  |

Gambar 8.17 *Flowchart* Subproses Matriks Perkalian

Gambar 8.17 di atas merupakan *sub-flowchart* dari prosedur matriks perkalian dari program matriks yang berfungsi sebagai petunjuk jalannya proses perkalian pada program matriks ini. Untuk *sub-flowchart* prosedur matriks penjumlahan dapat dilihat pada gambar di bawah ini.

|  |
| --- |
|  |

Gambar 8.18 *Flowchart* Subproses Matriks Penjumlahan

Gambar 8.18 di atas merupakan *sub-flowchart* dari prosedur matriks penjumlahan pada program matriks yang berfungsi sebagai petunjuk jalannya proses penjumlahan pada program matriks ini. untuk *sub-flowchart* restart dapat dilihat pada gambar di bawah ini.

|  |
| --- |
|  |

Gambar 8.19 *Flowchart* Subproses *Restart*

Gambar 8.19 di atas merupakan *sub-flowchart* dari prosedur restart yang ada pada program matriks yang berfungsi sebagai petunjuk pada user agar memilih untuk mengulang program atau untuk menyudahi program.

#### Pseudocode Program Matriks

*Pseudocode* adalah sebuah istilah dalam pemrograman untuk menuliskan sintaks, *statement*, algoritma, dan lainnya dalam bahasa yang mudah dipahami manusia. Secara sederhana, *pseudocode* adalah representasi kode program dengan versi mudah dipahami manusia. Kemudian, berikut adalah *pseudocode* dari program perhitungan matriks.

|  |
| --- |
| Program Matriks  DESKRIPSI  {Program mengoperasikan matematika pada matriks }  FUNCTION header() : void  DESKRIPSI :  BEGIN  write("|===========================================|\n");  write("| KELOMPOK 23 |\n");  write("|===========================================|\n");  write("| Nama Anggota Kelompok |\n");  write("|===========================================|\n");  write("|2205551005 Gede Made Rapriananta Pande |\n");  write("|2205551003 Ida Bagus Paalakaa RNB |\n");  write("|2205551069 Ni Kadek Ari Diah Lestari |\n");  write("|2205551072 I Gusti Ayu Krisna Kusuma Dewi |\n");  write("|2205551076 Kadek Yogi Dwi Putra Utama |\n");  write("|2205551079 Anak Agung Indi Kusuma Putra |\n");  write("|===========================================|\n\n");  END  FUNCTION  validasi()  DEKLARASI  input[100] : char  negatif, salah, koma, belakang, deret, depan, a, i : int  hasil, nolKoma : float  BEGIN  negatif <- 0  salah <- 0  koma <- 0  belakang <- 0  deret <- 0  depan <- 0  nolKoma <- 1  read(input)  if(input[a]=='\0') then  salah <- 1  end if  while(input[a]!='\0') do  if(input[a]=='-') then  negatif++  if(a>0 || negatif==2){  salah <- 1  input[a] <- '\0'  end if  if(input[a+1]<'0' || input[a+1]>'9') then  salah <- 1  input[a] <- '\0'  end if  a++  else if(input[a]=='.') then  koma++  if(input[0]=='.' || input[a+1]=='\0' || koma>1 ) then  salah <- 1  input[a] <- '\0'  end if  a++  else if(input[a]>='0' && input[a]<='9') then  if(koma==1) then  belakang <- (belakang\*10)+(input[a]-48)  deret++  a++  else  depan <- (depan\*10)+(input[a]-48);  a++  end if  else  salah <- 1  input[a] <- '\0'  end if  end while  if(koma==1) then  for i <- 0 to deret-1  nolKoma <- nolKoma/10  end for  hasil <- belakang\*nolKoma+depan  else  hasil <- depan  end if  if(negatif==1) then  hasil=hasil-(hasil\*2)  end if  if(salah==1) then  write("Ulangi Input Anda")  return call validasi()  else  return hasil  end if  STOP  PROCEDURE  ulang()  DEKLARASI  a : float  b : int  BEGIN  write("Ingin Mencoba Programnya Lagi")  write("1.Ya")  write("2.Tidak")  while(a!=b || a<1 || a>2) do  a <- validasi()  b <- (int)a  if(a!=b || a<1 || a>2) then  write("Ulangi Input Anda")  end if  end while  call system("cls")  if(a==1) then  call menu1()  else if(a==2){  write("Terima Kasih")  end if  call system("exit")  STOP  PROCEDURE  input2(int b, int x, int y)  DEKLARASI  array[10][10], array1[10][10], array2[10][10], g : float  i, j : int  BEGIN  for i <- 0 to y-1  for j <- 0 to x-1  g <- validasi()  array1[i][j] <- g  end for  end for  if(b==1 || b==2) then  call system("cls")  for i <- 0 to y-1  for j <- 0 to x-1  g <- validasi()  array2[i][j] <- g  end for  end for  end if  call system("cls")  if(b==1) then  for i <- 0 to y-1  for j <- 0 to x-1  array[i][j] <- array1[i][j]+array2[i][j]  end for  end for  else if(b==2) then  for i <- 0 to y-1  for j <- 0 to x-1  array[i][j] <- array1[i][j]\*array2[i][j]  end for  end for  else if(b==3) then  for i <- 0 to x-1  for j <- 0 to y-1  array[i][j] <- array1[j][i]  end for  end for  end if  if(b==1 || b==2) then  for i <- 0 to y-1  for j <- 0 to x-1  write(array[i][j])  end for  end for  else if(b==3) then  for i <- 0 to x-1  for j <- 0 to y-1  write(array[i][j])  end for  end for  end if  call ulang()  STOP  PROCEDURE  input1(int b)  DEKLARASI  c, d : float  x, y : int  BEGIN  while(c!=x || c<1 || c>9) do  c <- validasi()  x <- (int)c  if(c!=x || c<1 || c>9) then  write("Ulangi Input Anda")  end if  end while  while(d!=y || d<1 || d>9) do  d <- validasi()  y <- (int)d  if(d!=y || d<1 || d>9) then  write("Ulangi Input Anda")  end if  end while  call system("cls")  call input2(b, x, y)  STOP  PROCEDURE  menu1()  DEKLARASI  a : float  b : int  BEGIN  write("Pilihan Menu")  while(a!=b || a<1 || a>3) do  a <- validasi()  b <- (int)a  if(a!=b || a<1 || a3) then  write("Ulangi Input Anda")  end if  end while  call system("cls")  call input1(b)  STOP  PROCEDURE  cover()  BEGIN  write("Nama kelompok")  call system("pause")  call system("cls")  call menu1()  STOP  FUNCTION  main()  BEGIN  call cover()  return 0  STOP |

Kode Program 8.2 *Pseudocode* Program Matriks

Kode Program 8.2 di atas dijelaskan mengenai *pseudocode* program matriks. *Pseudocode* ini adalah rancangan dari program yang akan dibuat menggunakan bahasa C, yang nantinya akan di jalankan setelah *user* menginputkan beberapa data.

#### Trace Program Matriks

Dalam pembuatan program, tidak akan bisa berhasil dan pasti terdapat kesalahan di dalam program yang dibuat. Kesalahan dapat berupa ketidaksesuaian alur/logika, ataupun kesalahan dalam penulisan sintaks. Maka dari itu, perlu adanya sebuah penelusuran (*trace*) agar kesalahan dapat dengan cepat ditemukan. *Trace* sendiri terdiri dari *input*, proses dan *output* dari program.

Tabel 8.2 *Trace* Program Matriks

|  |  |
| --- | --- |
| *Input* | pil=3  baris=2  kolom=2  matriks\_1[0][0]=1  matriks\_1[0][1]=1  matriks\_1[1][0]=2  matriks\_1[1][1]=2 |
| Proses | i=0  i<baris : true  j=0  j<kolom : true  matriks\_2[j][i]= matriks\_1[i][j]  matriks\_2[0][0]= matriks\_1[0][0]  matriks\_2[0][0]=1  j++  j<kolom : true  matriks\_2[j][i]= matriks\_1[i][j]  matriks\_2[1][0]= matriks\_1[0][1]  matriks\_2[1][0]=1  j++  j<kolom : false  i++    i<baris : true  j=0  j<kolom : true  matriks\_2[j][i]= matriks\_1[i][j]  matriks\_2[0][1]= matriks\_1[1][0]  matriks\_2[0][1]=2  j++  j<kolom : true  matriks\_2[j][i]= matriks\_1[i][j]  matriks\_2[1][1]= matriks\_1[1][1]  matriks\_2[1][1]=2  j++  j<kolom : false  i++    i<baris : false |
| *Output* | Print :  Matriks Awal :  1 1  2 2  Matriks Akhir :  1 2  1 2  Ingin menghitung kembali? [1] Menghitung Kembali [2] Keluar |

Tabel 8.2 merupakan *trace* atau *tracing* dari program pehitungan matriks. Pada *trace* di atas *user* diminta memilih operasi matematika pada matriks yang diinginkan lalu menginputkan baris, kolom, matriks awal, dan matriks selanjutnya, dimana matriks sebelumnya 1 1 2 2 menjadi 1 2 1 2.

### **Flowchart, Pseudocode, dan Trace Program Statistika**

Program ini merupakan suatu program statistika yang mampu menghitung nilai median, modus, serta *mean* dari *n* data yang di-*input*-kan dengan menggunakan *pointer*. Berikut merupakan *flowchart, pseudocode,* serta *trace* dari program statistika tersebut.

#### Flowchart Program Statistika

*Flowchart* merupakan suatu notasi algoritma yang disusun dengan simbol-simbol untuk menjelaskan suatu alur proses atau langkah-langkah yang berurutan dari program tersebut yang harus diikuti pada saat program disusun. Dapat disimpulkan, bahwa *flowchart* biasanya digunakan untuk menerangkan alur suatu proses dari sebuah program secara sistematis hingga selesai.

|  |
| --- |
|  |

Gambar 8. 20 *Flowchart* Subproses Nama Kelompok

Gambar 8.20 adalah *flowchart* subproses *nama\_kelompok()* pada program statistika. Pada program iniakan ditampilkan nama-nama dari anggota kelompok. Selanjutnya akan dilanjutkan dengan proses *pause* dan *cls*, dan *flowchart* nama kelompok selesai.

|  |
| --- |
|  |

Gambar 8.21 *Flowchart* Subproses Validasi Menu

Gambar 8.21 adalah *flowchart* subproses *validasiMenu()* pada program statistika. Pada program iniakan dilakukan validasi *input*, jika *input* yang dimasukkan tidak sama dengan 1-3, maka *user* akan diminta memasukkan ulang. Jika sudah sesuai dengan kondisi, maka program akan berlanjut dan *flowchart* validasi menu selesai.

|  |
| --- |
|  |

Gambar 8.22 *Flowchart* Subproses Validasi Angka

Gambar 8.22 adalah *flowchart* subproses *validasiAngka()* pada program statistika. Pada program iniakan dilakukan validasi *input*, jika *input* yang dimasukkan tidak lebih dari sama dengan 0, maka *user* akan diminta memasukkan ulang. Jika sudah sesuai dengan kondisi, maka program akan berlanjut dan *flowchart* validasi angka selesai.

|  |
| --- |
|  |

Gambar 8.23 *Flowchart* Subproses *Print Array*

Gambar 8.23 adalah *flowchart* subproses *printArray()* pada program statistika. Program ini akan menampilkan nilai data yang dimasukkan *user*. Setelah menampilkan nilai tersebut, maka proses *flowchart* selesai.

|  |
| --- |
|  |

Gambar 8.24 *Flowchart* Subproses *Bubble Sort*

Gambar 8.24 adalah *flowchart* subproses *bubbleSort()* pada program statistika. Subproses ini merupakan proses pengurutan nilai-nilai data yang dimasukkan oleh *user*. Setelah mengurutkan nilai tersebut, maka proses *flowchart* selesai.

|  |
| --- |
|  |

Gambar 8.25 *Flowchart* Subproses Hitung Median

Gambar 8.25 adalah *flowchart* subproses *hitungMedian()* pada program statistika. Subproses ini merupakan proses perhitungan dari median. Proses perhitungan tersebut berdasarkan suatu rumus, dan jika perhitungan sudah selesai, maka akan dilakukan return median, dan proses *flowchart* selesai.

|  |
| --- |
|  |

Gambar 8.26 *Flowchart* Subproses Hitung Mean

Gambar 8.26 adalah *flowchart* subproses *hitungMean()* pada program statistika. Subproses ini merupakan proses perhitungan dari mean. Proses perhitungan tersebut berdasarkan suatu rumus, dan jika perhitungan sudah selesai, maka akan dilakukan *return* mean, dan proses *flowchart* selesai.

|  |
| --- |
|  |

Gambar 8.27 *Flowchart* Subproses Hitung Modus

Gambar 8.27 adalah *flowchart* subproses *hitungModus()* pada program statistika. Subproses ini merupakan proses perhitungan dari modus. Proses perhitungan tersebut berdasarkan suatu rumus, dan jika perhitungan sudah selesai, maka akan dilakukan return modus, dan proses *flowchart* selesai.

|  |
| --- |
|  |

Gambar 8.28 *Flowchart* Subproses Keluar

Gambar 8.28 adalah *flowchart* subproses *keluar()* pada program statistika. Subproses ini berisi menu untuk mengulang, kembali ke menu, atau keluar dari program. Proses akan dilanjutkan sesuai dengan pilihan *user*, dan proses *flowchart* selesai.

|  |
| --- |
|  |

Gambar 8.29 *Flowchart* Subproses *Input* Angka

Gambar 8.29 adalah *flowchart* subproses *inputAngka()* pada program statistika. Subproses ini merupakan proses memasukkan banyak data yang akan dihitung dan nilai dari masing-masing data tersebut sesuai dengan menu perhitungan yang dipilih. Kemudian nilai data tersebut akan diurutkan menggunakan fungsi *bubbleSort()* dan ditampilkan dengan fungsi *printArray()*. Selanjutnya, program akan masuk ke masing-masing subproses perhitungan, dan proses *flowchart* selesai.

|  |
| --- |
|  |

Gambar 8.30 *Flowchart Int* *Main* Statistika

Gambar 8.30 adalah *flowchart main* pada program statistika. *Flowchart* ini akan menampilkan pilihan menu perhitungan untuk *user*. Kemudian proses perhitungan tersebut akan masuk ke masing-masing subproses perhitungan berdasarkan jenis perhitungan yang dipilih, dan proses *flowchart* selesai.

#### Pseudocode Program Statistika

*Pseudocode* adalah suatu cara menulis algoritma yang mirip dengan bahasa pemrograman, tetapi *pseudocode* lebih sederhana dan mudah dipahami oleh manusia pada umumnya. *Pseudocode* memudahkan user untuk menyelesaikan suatu masalah tanpa memikirkan sintaks dari suatu bahasa pemrograman. Berikut ini *pseudocode* dari program statistika.

|  |
| --- |
| Algoritma Program Statistika  DEKLARASI:  pilihan: int  inputData(int pilih): int  keluar(): void  menu(): void  main: int  FUNCTION namaKelompok(): void  DESKRIPSI:  BEGIN  write "nama kelompok"  system(pause)  system(cls)  END  FUNCTION validasiMenu(): int  DEKLARASI:  select: int  DESKRIPSI:  BEGIN  read select    IF select >= 1 || select <= 3  TRUE  ELSE  read select  END IF  END  FUNCTION validasiAngka(): int  DEKLARASI:  input: int  DESKRIPSI:  BEGIN  read input    IF input >= 0  TRUE  ELSE  read input  END IF  END  FUNCTION printArray(int \*arr, int n): float  DEKLARASI:  i: int  DESKRIPSI:  BEGIN  FOR (i = 0; i < n; i++)  write ("%d ", \*(arr + i))  END FOR    write "\n"  END  FUNCTION bubbleSort(int \*arr, int n): float  DEKLARASI:  i: int  DESKRIPSI:  BEGIN  FOR (i = 0; i < n - 1; i++)    j: int  FOR (j = 0; j < n - 1; j++)  IF arr[j + 1] < arr[j]  int temp = arr[j]  arr[j] = arr[j + 1]  arr[j + 1] = temp  END IF  END FOR  END FOR  END  FUNCTION hitungMedian(int \*arr, int n): float  DEKLARASI:  median = 0: float  DESKRIPSI:  BEGIN  IF n % 2 == 0  median = (float)(\*(arr + ((n - 1) / 2)) + \*(arr + (n / 2))) / 2.0  ELSE  median = (float)\*(arr + (n / 2))  END IF  RETURN MEDIAN  END  FUNCTION hitungMean(int \*arr, int n, int sum)  DEKLARASI:  mean: float  i: int  DESKRIPSI:  BEGIN  FOR (i = 0; i < n; i++)  sum = sum + \*(arr + i)  END FOR  mean = sum / (float)n  RETURN mean  END  FUNCTION hitungModus(int \*arr, int n, int \*barray): int  DEKLARASI:  i, j, t, k = 0, c = 1, nilaiMax = 0, hitung: int  DESKRIPSI:  BEGIN  FOR (i = 0; i < n - 1; i++)  hitung = 0  FOR (j = i + 1; j < n; j++)  IF \*(arr + i) == \*(arr + j)  hitung++  END IF  END FOR  IF (hitung > nilaiMax) && (hitung != 0)  k = 0  nilaiMax = hitung  \*(barray + k) = \*(arr + i)  k++  ELSE IF hitung == nilaiMax  \*(barray + k) = \*(arr + i)  k++  END IF  END FOR  FOR (i = 0; i < n; i++)  IF \*(arr + i) == \*(barray + i)  c++  END FOR  IF c == n  write "tidak ada modus"  ELSE  write "modusnya adalah: "  FOR (i = 0; i < k; i++)  write ("%.2f ", (float)\*(barray + i))  END IF  END  FUNCTION keluar():void  DEKLARASI:  input: int  DESKRIPSI:  BEGIN  write "menu 1. ulang, 2. menu, 3. keluar"    validasiMenu(&input, pilihan: )  IF input == 1  system(cls)  inputData(pilihan)  ELSE IF input == 2  system(cls)  main()  ELSE IF input == 3  system(cls)  exit(0)  END IF  END  FUNCTION inputData(int pilih): int  DEKLARASI:  n, sum = 0: int  DESKRIPSI:  BEGIN  IF pilih == 1  validasiAngka(&n, masukkan banyak data untuk median: ")    arr[100] = {0}: int  i: int  FOR (i = 0; i < n; i++)  write ("data ke-%d ", i + 1)  validasiAngka((arr + i), ": ")  END FOR  bubbleSort(arr, n)  write "data yang diinputkan: "  printArray(arr, n)  barray[100] = {0}: int  write ("median: %.2f", hitungMedian(arr, n))  system(pause)  system(cls)  keluar()  ELSE IF pilih == 2  validasiAngka(&n, masukkan banyak data untuk mean: ")    arr[100] = {0}: int  i: int  FOR (i = 0; i < n; i++)  write ("data ke-%d ", i + 1)  validasiAngka((arr + i), ": ")  END FOR  bubbleSort(arr, n)  write "data yang diinputkan: "  printArray(arr, n)  barray[100] = {0}: int  write ("median: %.2f", hitungMean(arr, n, sum))  system(pause)  system(cls)  keluar()  ELSE IF pilih == 3  validasiAngka(&n, masukkan banyak data untuk modus: ")    arr[100] = {0}: int  i: int  FOR (i = 0; i < n; i++)  write ("data ke-%d ", i + 1)  validasiAngka((arr + i), ": ")  END FOR  bubbleSort(arr, n)  write "data yang diinputkan: "  printArray(arr, n)  barray[100] = {0}: int  hitungModus(arr, n, barray)  system(pause)  system(cls)  keluar()  END IF  END  FUNCTION main(): int  DESKRIPSI:  BEGIN  namaKelompok()  write "menu 1. median, 2. mean, 3. modus"    validasiMenu(&pilihan, "masukkan pilihan: ")  system(cls)  inputData(pilihan)  RETURN 0  END |

Kode Program 8.3 *Pseudocode* Program Statistika

*Pseudocode* pada Kode Program 8.3 akan membantu *programmer* dalam membuat program yang sebenarnya. *Pseudocode* ini sudah dapat diimplementasikan ke dalam sebuah program yang sudah siap digunakan dengan menambah sintaks-sintaks program yang kurang.

#### Trace Program Statistika

*Trace* program adalah suatu proses untuk menemukan *input* yang dimasukkan oleh *user* ke dalam sebuah program, dan menelaah hasil *input* tersebut, serta memahami cara pemrosesan yang ada pada suatu program untuk mendapatkan output yang sesuai. Di bawah ini adalah *trace* program dari program statistika yang dijabarkan dalam bentuk tabel berikut.

Tabel 8.3 *Trace* Program Statistika

|  |  |
| --- | --- |
| *Input* | select = 2 (nilaiAkhir)  n = 4  data ke-1 = 5  data ke-2 = 15  data ke-3 = 10  data ke-4 = 10 |
| Proses | sum = 0 + 5  sum = 5 + 15  sum = 20 + 10  sum = 30 + 10  sum = 40  mean = 40 / 4  mean = 10 |
| *Output* | Print Mean = 10.00 |

Tabel 8.3 merupakan hasil *trace* dari operasi mencari mean. Pada saat menjalankan program, *user* akan diminta untuk memilih menu yang terdiri dari menu median, mean, atau modus. Jika, dipilih menu mean maka akan muncul perintah untuk memasukkan banyak data yang akan dihitung dan memasukkan nilai dari masing-masing data tersebut. Selanjutnya, program akan masuk ke dalam tahap pemrosesan sesuai dengan rumus yang sudah ditentukan serta kemudian program akan menampilkan *output* hasil dari operasi tersebut kepada *user*.

### **Flowchart, Pseudocode, dan Trace Program Bubble Sort**

Program *bubble sort* ini disusun dengan permasalahan sebagai berikut. Suatu data acak berjumlah 1000, 16.000, dan 64.000 dibuat dan lalu disimpan di dalam suatu *array*. *Array* tersebut akan diurutkan menggunakan *bubble sort* dan *pointer*. Waktu yang dibutuhkan pun dicatat untuk masing-masing pengurutan. Berikut merupakan *flowchart, pseudocode,* serta *trace* dari program *bubble sort*.

#### Flowchart Program Bubble Sort

Proses pengurutan dan penghitungan waktu program *bubble sort* bisa dijabarkan dalam *flowchart* sehingga alur program lebih mudah dimengerti oleh orang awam. *Flowchart* ini dibagi menjadi beberapa bagian sesuai prosedur-prosedur yang ada pada program. Berikut adalah *flowchart* fungsi *main* dari program *bubble sort*.

|  |
| --- |
|  |

Gambar 8.31 *Flowchart* *Int Main Function Bubble Sort*

*Main function* pada Gambar 8.31 adalah fungsi *main* dari program mengurutkan menggunakan *bubble sort pointer* dan tanpa *pointer*. Memilih jumlah dari data yang akan diurutkan lalu mendeklarasikan berbagai prosedur untuk memunculkan angka acak, kemudian mengoutput angka-angka acak sesudah maupun sebelum diurut dan akan memanggil prosedur untuk menghitung kecepatan *sorting* dari kedua metode lalu meng*output*kan total waktunya.

|  |
| --- |
|  |

Gambar 8.32 *Flowchart* Subproses *Bubble Sort No Pointer*

Gambar 8.32 adalah *flowchart* kedua yaitu *flowchart* prosedur dari metode pengurutan *bubble sort* tanpa menggunakan *pointer*. *Sub-flowchart* ini menggambarkan bagaimana proses matematis untuk mengurutkan angka acak tanpa memakai *pointer*.

|  |
| --- |
|  |

Gambar 8.33 *Flowchart* Subproses *Bubble Sort Pointer*

Gambar 8.33 ini adalah *flowchart* dari prosedur pengurutan *bubble sort* dengan menggunakan *pointer*. Langkah pengurutan metode *bubble sort* ini menginisiasikan variabel yang berguna sebagai *pointer*.

|  |
| --- |
|  |

Gambar 8.34 *Flowchart* Subproses Menghitung Waktu *Bubble Sort No Pointer*

Gambar 8.34 adalah *flowchart* untuk menggambar langkah dari prosedur menghitung *bubble sort* tanpa menggunakan *pointer*. Pada *flowchart* ini digunakan sebuah variabel yaitu variabel *count* dan pada prosesnya memanggil prosedur dari mengurutkan data *bubble sort* tanpa *pointer*.

|  |
| --- |
|  |

Gambar 8.35 *Flowchart* Subproses Menghitung Waktu *Bubble Sort Pointer*

Gambar 8.35 adalah *flowchart* menghitung waktu pada program pengurutan *bubble sort* dengan *pointer*. Langkah-langkah yang dilakukan sama, hanya saja perbedaan dari Gambar 3.66 adalah prosedur yang dipanggil adalah prosedur *bubble sort pointer*.

#### Pseudocode Program Bubble Sort

Program *bubble sort* dengan *pointer* dan tanpa *pointer* bisa dibuatkan dalam versi *pseudocode*nya agar bahasa pemrograman yang susah dimengerti orang awam menjadi bisa dimengerti karena *pseudocode* adalah program yang syntaksnya menggunakan bahasa manusia (inggris) dalam pengimplementasiannya. Berikut adalah contoh dari *pseudocode* dari program *bubble sort*.

|  |
| --- |
| DESKRIPSI :  PROGRAM BUBBLE SORT  FUNCTION seedArray(arr[] : int, n : int) : void  DEKLARASI  BEGIN :  i : int  srand(0)  for (i = 0 i <= n i++)  arr[i] = rand()  ENDFOR  END  FUNCTION dupplicatedArray(source[] : int, target[] : int, n : int)  DEKLARASI  BEGIN :  i : int  for (i = 0 i <= n i++)  target[i] = source[i]  ENDFOR  END  FUNCTION outputArray(arr[] : int, n : int)  DEKLARASI  BEGIN :  i : int  for (i = 0 i < n i++)  WRITE("Angka acak ke-%d \t: %d", i+1, arr[i])  WRITE("\n")  ENDFOR  END  FUNCTION bubbleNoPointer(arr[] : int, n : int)  DEKLARASI  BEGIN :  i, j, temp : int  for (i = 0 i < n - 1 i++)  for (j = 0 j < n - 1 - i j++)  if (arr[j] > arr[j + 1]) THEN  temp = arr[j]  arr[j] = arr[j + 1]  arr[j + 1] = temp  ENDIF  ENDFOR  ENDFOR  END  FUNCTION swap(\* i1 : int, \* i2 : int)  DEKLARASI  BEGIN :  temp = \* i1 : int  \* i1 = \* i2  \* i2 = temp  END  FUNCTION bubblePointer(arr[] : int, n : int)  DEKLARASI  BEGIN :  i, a : int  for (i = 0 i < n - 1 i++)  flag = 0 : int  for (a = 0 a < n - 1 a++)  if (arr[a] > arr[a + 1]) THEN  swap( & arr[a], & arr[a + 1])  flag = 1  ENDIF  ENDFOR  if (flag == 0) THEN  break  ENDIF  ENDFOR  END  FUNCTION count\_BubbleNoPointer(arr[] : int, n : int) : float  DEKLARASI  BEGIN :  clock\_t start = clock()  CALL bubbleNoPointer(arr, n)  clock\_t end = clock()  detik = (float)(end - start) / CLOCKS\_PER\_SEC : float  return detik  END  FUNCTION count\_BubblePointer(arr[] : int, n : int) float  DEKLARASI  BEGIN :  clock\_t start = clock()  CALL bubblePointer(arr, n)  clock\_t end = clock()  detik = (float)(end - start) / CLOCKS\_PER\_SEC : float  return detik  }  FUNCTION valid\_int(\*var : int) : int  DEKLARASI  BEGIN :  VAR buff[20] : char  VAR cek : char  fflush(stdin)  if(fgets(buff, sizeof(buff), stdin)!=NULL)THEN  if(sscanf(buff, "%d %c", var, &cek)==1)THEN  return 1  ENDIF  ENDIF  return 0  END  FUNCTION input\_pilihan(\*var : int, \*prompt : char) : void  DEKLARASI  BEGIN :  while (1)THEN  WRITE(prompt)  if(valid\_int(var)) THEN  break  WRITE("\nPilih Jumlah Data yang BENAR!\n")  WRITE("\n")  ENDWHILE  END  FUNCTION input\_ulang(\*var : int, \*prompt : char) : void  DEKLARASI  BEGIN :  while (1)THEN  WRITE(prompt)  if(valid\_int(var))THEN  break  WRITE("\nMau mengulang atau tidak?!\n")  ENDWHILE  END  FUNCTION ulang() : void  DEKLARASI  BEGIN :  mengulang : int  input\_ulang(&mengulang, "\nIngin mengulang program?\n1. YA \n2. TIDAK\n""=> ")  if (mengulang == 1 )THEN  system("cls")  CALL main()  else if(mengulang == 2)THEN  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TERIMA KASIH \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")  exit (0)  else THEN  printf("\nMau mengulang atau tidak?!\n")  CALL ulang()  ENDIF  END  FUNCTION main() : int  n : int  pilihan : int  WRITE("======================================\n")  WRITE("||Bubble Sort Pointer and No Pointer||\n")  WRITE("======================================\n")  WRITE("||Pilih Total Data yang Diinginkan! ||\n")  WRITE("--------------------------------------\n")  WRITE("|1 | 1000 ||\n")  WRITE("|2 | 16000 ||\n")  WRITE("|3 | 64000 ||\n")  WRITE("======================================\n")  input\_pilihan(&pilihan, "Masukan Pilihanmu => ")  WRITE("--------------------------------------")  if (pilihan < 1)THEN  WRITE("\n\nMasukan pilihan yang tersedia!\n")  CALL ulang()  else if (pilihan == 1)THEN  n = 1000  arr[n] : int  arr2[n] : int  CALL seedArray(arr, n)  WRITE("\n\n==================================\n")  WRITE("Jumlah %d data sebelum di sort ||\n", n)  WRITE("==================================\n")  CALL outputArray(arr, n)  CALL dupplicatedArray(arr, arr2, n)  detik1 = count\_BubbleNoPointer(arr, n) : float  detik2 = count\_BubblePointer(arr2, n) : float  WRITE("\n==================================\n")  WRITE("Jumlah %d data sesudah di sort ||\n", n)  WRITE("==================================\n")  CALL outputArray(arr, n)  WRITE("\n==============================================\n")  WRITE("Waktu Bubble Sort tanpa pointer: %f\n", detik1)  WRITE("\nWaktu Bubble Sort pointer : %f", detik2)  WRITE("\n==============================================\n")  if (detik1 < detik2) THEN  WRITE("\n=============================================\n")  WRITE("!! Bubble sort tanpa pointer lebih cepat !!\n")  WRITE("=============================================\n")  else if (detik1 > detik2)THEN  WRITE("\n=============================================\n")  WRITE("!! Bubble sort dengan pointer lebih cepat !!\n")  WRITE("==============================================\n")  else if (detik1 == detik2)THEN  WRITE("\n==============================================================\n")  WRITE("!! Kecepatan bubble tanpa pointer dan dengan pointer sama !!\n")  WRITE("==============================================================\n")  ENDIF  CALL ulang()  else if (pilihan == 2)THEN  n = 16000  arr[n] : int  arr2[n] : int  CALL seedArray(arr, n)  WRITE("\n\n==================================\n")  WRITE("Jumlah %d data sebelum di sort ||\n", n)  WRITE("==================================\n")  CALL outputArray(arr, n)  CALL dupplicatedArray(arr, arr2, n)  detik1 = count\_BubbleNoPointer(arr, n) : float  detik2 = count\_BubblePointer(arr2, n) : float  WRITE("\n==================================\n")  WRITE("Jumlah %d data sesudah di sort ||\n", n)  WRITE("==================================\n")  CALL outputArray(arr, n)  WRITE("\n==============================================\n")  WRITE("Waktu Bubble Sort tanpa pointer: %f\n", detik1)  WRITE("\nWaktu Bubble Sort pointer : %f", detik2)  WRITE("\n==============================================\n")  if (detik1 < detik2) THEN  WRITE("\n=============================================\n")  WRITE("!! Bubble sort tanpa pointer lebih cepat !!\n")  WRITE("=============================================\n")  else if (detik1 > detik2)THEN  WRITE("\n=============================================\n")  WRITE("!! Bubble sort dengan pointer lebih cepat !!\n")  WRITE("==============================================\n")  else if (detik1 == detik2)THEN  WRITE("\n==============================================================\n")  WRITE("!! Kecepatan bubble tanpa pointer dan dengan pointer sama !!\n")  WRITE("==============================================================\n")  ENDIF  CALL ulang()  else if (pilihan == 3)THEN  n = 64000  arr[n] : int  arr2[n] : int  CALL seedArray(arr, n)  WRITE("\n\n==================================\n")  WRITE("Jumlah %d data sebelum di sort ||\n", n)  WRITE("==================================\n")  CALL outputArray(arr, n)  CALL dupplicatedArray(arr, arr2, n)  detik1 = count\_BubbleNoPointer(arr, n) : float  detik2 = count\_BubblePointer(arr2, n) : float  WRITE("\n==================================\n")  WRITE("Jumlah %d data sesudah di sort ||\n", n)  WRITE("==================================\n")  CALL outputArray(arr, n)  WRITE("\n==============================================\n")  WRITE("Waktu Bubble Sort tanpa pointer: %f\n", detik1)  WRITE("\nWaktu Bubble Sort pointer : %f", detik2)  WRITE("\n==============================================\n")  if (detik1 < detik2) THEN  WRITE("\n=============================================\n")  WRITE("!! Bubble sort tanpa pointer lebih cepat !!\n")  WRITE("=============================================\n")  else if (detik1 > detik2)THEN  WRITE("\n=============================================\n")  WRITE("!! Bubble sort dengan pointer lebih cepat !!\n")  WRITE("==============================================\n")  else if (detik1 == detik2)THEN  WRITE("\n==============================================================\n")  WRITE("!! Kecepatan bubble tanpa pointer dan dengan pointer sama !!\n")  WRITE("==============================================================\n")  ENDIF  CALL ulang()  else if (pilihan >= 4 )THEN  system("cls")  printf("\n\t!Pilihan hanya sampai 3!\n\n")  CALL main()  ENDIF  END |

Kode Program 8.4 *Pseudocode* Program *Bubble Sort*

Kode Program 8.4 adalah contoh dari *pseudocode* program *bubble sort* yang di mana mengubah sintaks-sintaksnya menggunakan bahasa manusia khususnya bahasa inggris. Seperti contoh mengubah sintaks printf menjadi WRITE dan sintaks memanggil prosedur atau fungsi dengan menambahkan CALL kemudian diikuti dengan prosedur atau fungsi yang ingin dipanggil.

#### Trace Program Bubble Sort

Setelah selesai menyusun alur dan notasi algoritma, *trace* atau pelacakan algoritma pun dapat dilakukan. *Trace* dilakukan untuk melacak alur mulai dari pemberian *input*, memproses *input,* hingga *output* berupa hasil dari proses tersebut, sehingga dapat memastikan apakah implementasi algoritma yang digunakan sudah sesuai atau belum. Berikut ini adalah *trace* dari program *bubble sort*.

Tabel 8.4 *Trace* Program *Bubble Sort*

|  |  |
| --- | --- |
| *Input* | Pilih data = 1  mengulang = 2 |
| Proses | n = 1000  CALL seedArray (int arr[ ], int n)  srand(0);  for (i = 0; i <= n; i++) {  arr[i] = rand();  CALL dupplicatedArray (int source[ ], int target[ ], int n)  for (i = 0; i <= n; i++){  target[i] = source[i];  CALL outputArray(int arr[ ], int n)  for (i = 0; i < n; i++) {  printf("Angka acak ke-%d : %d", i+1, arr[i])  CALL bubbleNoPointer(int arr[ ], int n)  for (i = 0; i < n - 1; i++) {  for (j = 0; j < n - 1 - i; j++) {  if (arr[j] > arr[j + 1]) {  temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  CALL swap (int \* i1, int \* i2)  \* i1 = \* i2;  \* i2 = temp;  CALL bubblePointer(int arr[ ], int n)  for (i = 0; i < n - 1; i++) {  int flag = 0;  for (a = 0; a < n - 1; a++) {  if (arr[a] > arr[a + 1]) {  swap( & arr[a], & arr[a + 1]);  flag = 1;  if (flag == 0) {  break;  float detik1 = count\_BubbleNoPointer (int arr[], int n)  clock\_t start = clock();  bubbleNoPointer(arr, n);  clock\_t end = clock();  float detik = (float)(end - start) / CLOCKS\_PER\_SEC;  return detik;  float detik2 = count\_BubblePointer (int arr[], int n)  clock\_t start = clock();  bubblePointer(arr, n);  clock\_t end = clock();  float detik = (float)(end - start) / CLOCKS\_PER\_SEC;  return detik;  CALL ulang(); |
| *Output* | Intruksi memilih pilihan:  Data acak sebelum di sort: 1-1000  Data acak sesudah di sort: 1-1000  Waktu tanpa pointer : 0.001000  Waktu dengan pointer : 0.002000  Mengulang?  Terima Kasih |

Tabel 8.4 di atas adalah *trace* dari program *bubble sort* dimana *user* memilih pilihan 1000 (seribu) data acak untuk diurutkan. Program kemudian memanggil prosedur untuk melakukan operasi memunculkan angka acak yang kemudian akan diurutkan serta menghitung waktu dari metode *bubble sort pointer* dan tanpa *pointer*.

### **Flowchart, Pseudocode, dan Trace Program Enkripsi dan Deskripsi**

Program enkripsi dan deskripsi ini dapat melaksanakan suatu enkripsi serta deskripsi kata atau kalimat dengan pergeseran abjad sebagai kunci enkripsi serta deskripsi. Program akan membutuhkan suatu *input* berupa jumlah pergeseran serta *plaintext* bila melakukan enkripsi, serta suatu *input* berupa jumlah pergeseran dan *ciphertext* bila melakukan deskripsi. *Output* dari program ini adalah hasil dari proses enkripsi dan deskripsi tersebut. Berikut merupakan *flowchart, pseudocode,* serta *trace* dari program enkripsi dan deskripsi.

#### Flowchart Program Enkripsi dan Deskripsi

Program enkripsi dan dekripsi adalah program yang dibuat untuk melakukan enkripsi dan dekripsi berdasarkan kunci yang diinputkan pengguna. Kunci yang diinputkan pengguna adalah jumlah pergeseran huruf yang diinginkan untuk melakukan enkripsi. Program ini dibuat menggunakan Bahasa pemrograman C. program ini nantinya akan menerima *input* berupa kalimat atau kata yang ingin dienkripsi dan jumlah pergeseran yang diinginkan. Dan akan mengeluarkan *output* berupa hasil enkripsi atau dekripsi.

|  |
| --- |
|  |

Gambar 8.36 *Flowchart* *Int Main* Program Enkripsi dan Dekripsi

Gambar 3.68 di atas merupakan *flowchart* dari fungsi main pada program konversi biner. Alur dari *flowchart* fungsi main program konversi biner dimulai dari *“mulai”.* Lalu akan menampilkan nama kelompok dan lanjut ke fungsi main menu.

|  |
| --- |
|  |

Gambar 8.37 *Flowchart* Subproses *Main Menu*

Gambar 8.37 di atas merupakan *flowchart* dari prosedur *mainMenu()* program Enkripsi dan Dekripsi. pertama akan masuk ke pilihan konversi. Setelah pengguna memilih konversi, maka akan masuk ke fungsi dari masing-masing konversi untuk diproses selanjutnya. Setelah selesai akan masuk ke pilihan apakah pengguna ingin mengulang program atau tidak.

|  |
| --- |
|  |

Gambar 8.38 *Flowchart* Subproses *Plain Text*

Gambar 8.38 di atas merupakan *flowchart* dari prosedur *plain()* yang digunakan untuk menjalankan proses untuk mengenkripsi kalimat yang diinput dengan pergeseran yang telah diinputkan sebelumnya. Fungsi ini akan mengambil *input* berupa kalimat yang akan yang akan dienkripsi oleh program dan akan mengeluarkan hasil yang sudah enkripsi.

|  |
| --- |
|  |

Gambar 8.39 *Flowchart* Subproses *Cipher Text*

Gambar 8.39 di atas merupakan *flowchart* dari prosedur *cipher()* yang digunakan untuk menjalankan proses untuk mendekripsi hasil enkripsi dengan jumlah pergeseran yang sama yang digunakan untuk mengenkripsi, dan hasilnya akan ditampilkan.

|  |
| --- |
|  |

Gambar 8.40 *Flowchart* Subproses Ulang Program

Gambar 8.40 di atas merupakan *flowchart* dari fungsi *ulang()* yang digunakan untuk menampilkan menu ulang yang akan memberikan pilihan kepada pengguna apakah ingin mengulang program atau tidak.

|  |
| --- |
|  |

Gambar 8.41 *Flowchart* Subproses Validasi *Input*

Gambar 8.41 di atas merupakan *flowchart* dari validasi yang digunakan pada program konversi biner. Fungsi *validasiRange()* digunakan untuk memvalidasi bilangan dengan range tertentu. Fungsi *validasiKey()* digunakan untuk memvalidasi bilangan positif.

#### Pseudocode Program Enkripsi dan Deskripsi

*Pseudocode* program enkripsi dan dekripsi berisi perintah perintah yang akan digunakan untuk membuat program enkripsi dan dekripsi . *Pseudocode* tidak terpaku pada peraturan dari Bahasa pemrograman manapun, jadi pada dasarnya *pseudocode* merupakan orek-orek awal bagi *programmer* untuk membuat program yang diinginkan.

|  |
| --- |
| //Program Enkripsi dan Dekripsi  PROCEDURE errorPlain() : void  DESKRIPSI:  BEGIN  CALL system("cls")  WRITE " Maaf Input Plaintext Anda Salah !"  WRITE " Mohon Hanya Inputkan Huruf dalam lowercase"  END  PROCEDURE errorCipher() : void  DESKRIPSI:  BEGIN  CALL system("cls")  WRITE " Maaf Input Ciphertext Anda Salah !"  WRITE " Mohon Hanya Inputkan Huruf dalam Uppercase"  END  FUNCTION validasiRange() : input  DEKLARASI:  input, notValid : char  valid : int  READ input, notValid  DESKRIPSI:  BEGIN  IF (sscanf(input, "%d%c", &valid, &notValid) == 1 && valid == 1 || valid == 2)  return valid  ELSE  CALL system("cls")  WRITE " Maaf Input anda salah!"  WRITE " Masukan Input Dengan Benar"  validasiRange()  END  FUNCTION validasiKey() : int  DEKLARASI:  input, notValid : char  valid : int  READ input, notValid  DESKRIPSI:  BEGIN  IF (sscanf(input, "%d%c", &valid, &notValid) == 1 && valid > 0)  return valid  ELSE  CALL system("cls")  WRITE " Maaf Input anda salah !"  WRITE " Masukan Jumlah Pergeseran Dengan Benar"  CALL validasiKey()  END  PROCEDURE plain (char\* plainText, int key) : void  DEKLRASI:  a,salah : int  hasil : char  DESKRIPSI:  BEGIN  CALL system("cls")  IF ((int)plainText[0] == 10) THEN  CALL errorPlain()  FOR (int i = 0 i < 4 i++ ){  IF ((int)plainText[i] == 9) THEN  errorPlain()  ENDFOR  FOR(int i = 0 i < strlen(plainText)-1i++)  IF ((int)plainText[i] >= 97 && (int)plainText[i] <= 122 || (int)plainText[i] == 32) THEN  IF ((int)plainText[i] == 32) THEN  a = 32  ELSE  a = ((int)plainText[i] - 97 + key) % 26 + 65  hasil[i] = (char)(a)  ELSE  salah++  break  ENDFOR  IF (salah == 0)  WRITE "Plaintext = "  FOR(int i = 0 i < strlen(plainText)-1i++)  WRITE "$plainText[i]"  ENDFOR  WRITE "Ciphertext = "  FOR(int i = 0 i < strlen(plainText)-1i++)  WRITE "$hasil[i]"  ENDFOR  ELSE  CALL errorPlain()  END  PROCEDURE cipher(char\* cipherText, int key)  DEKLARASI:  a, salah : int  hasil[100] : char  DESKRIPSI:  BEGIN  CALL system("cls")  IF ((int)cipherText[0] == 10) THEN  CALL errorCipher()  FOR (int i = 0 i < 4 i++ )  IF ((int)cipherText[i] == 9) THEN  CALL errorCipher()  ENDFOR  FOR(int i = 0 i < strlen(cipherText)-1i++)  IF ((int)cipherText[i] >= 65 && (int)cipherText[i] <= 90 || (int)cipherText[i] == 32) THEN  IF((int)cipherText[i] == 32) THEN  a = 32  ELSE  a = ((int)cipherText[i] - 65 - key) % 26 + 97  hasil[i] = (char)(a)  ELSE salah++  ENDFOR  IF(salah == 0)  WRITE " Ciphertext = "  FOR (int i = 0 i < strlen(cipherText)-1i++)  WRITE "$cipherText[i]"  ENDFOR  WRITE " Plaintext = "  FOR (int i = 0 i < strlen(cipherText)-1i++)  WRITE "$hasil[i]"  ENDFOR  ELSE  CALL errorCipher()  END  PROCEDURE goodBye() : void  DESKRIPSI:  BEGIN  CALL system("cls")  WRITE "\t\t \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n"  WRITE "\t\t \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n"  WRITE "\t\t| | | |\n"  WRITE "\t\t| | | |\n"  WRITE "\t\t| | Terima Kasih Telah Menggunakan Program Ini yaaaaaa!!! | |\n"  WRITE "\t\t| | | |\n"  WRITE "\t\t| | | |\n"  WRITE "\t\t| | \\|||||||||||||||||||| |||||||||||||||||\\ | |\n"  WRITE "\t\t| | \\||||||||||||||||||| ||||||||||||||||||\\ | |\n"  WRITE "\t\t| | \\|||||||||||||||||| |||||||||||||||||||\\ | |\n"  WRITE "\t\t| | |||||||||| | |\n"  WRITE "\t\t| | |||||||||| |||||||||||||||||||/ | |\n"  WRITE "\t\t| | |||||||||| ||||||||||||||||||/ | |\n"  WRITE "\t\t| | |||||||||| |||||||||||||||||/ | |\n"  WRITE "\t\t| | |||||||||| ||||||||| | |\n"  WRITE "\t\t| | |||||||||| ||||||||| | |\n"  WRITE "\t\t| | |||||||||| ||||||||| | |\n"  WRITE "\t\t| | |||||||||| ||||||||| | |\n"  WRITE "\t\t| | \\|||||||| |||||||/ | |\n"  WRITE "\t\t| | \\||||||| ||||||/ | |\n"  WRITE "\t\t| | \\|||||| |||||/ | |\n"  WRITE "\t\t| | \\||||| ||||/ | |\n"  WRITE "\t\t| | \\|||| |||/ | |\n"  WRITE "\t\t| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |\n"  WRITE "\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n"  END  FUNCTION ulang() : int  DEKLARASI:  pilihan : int  DESKRIPSI:  BEGIN  CALL system("cls")  WRITE " Apakah anda ingin mencoba lagi?"  WRITE " (1 = ya / 2 = tidak)"  pilihan = validasiRange()  SWITCH (pilihan)  CASE 1:  CALL system("cls")  return 1  CASE 2:  CALL goodBye()  break  DEFAULT:  WRITE " Maaf Input anda salah !"  WRITE " Masukan Input Dengan Benar"  CALL ulang()  ENDSWITCH  END  PROCEDURE mainMenu(): void  DEKLARASI:  pilihan, read, key : int  plainText, ciphertext : char  DESKRIPSI:  BEGIN  DO  CALL system("cls")  WRITE "\t\t \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n"  WRITE "\t\t \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n"  WRITE "\t\t| | | |\n"  WRITE "\t\t| | ENKRIPSI DAN DEKRIPSI | |\n"  WRITE "\t\t| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |\n"  WRITE "\t\t| | | | |\n"  WRITE "\t\t| | 1. | Enkripsi | |\n"  WRITE "\t\t| | | | |\n"  WRITE "\t\t| | 2. | Dekripsi | |\n"  WRITE "\t\t| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |\n"  WRITE "\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n\n"  WRITE "Pilih Konversi : "  pilihan = validasiRange()  SWITCH (pilihan){  CASE 1:  system("cls")  WRITE " Masukan Plaintext : "  READ plainText  WRITE " Masukan Jumlah Pergeseran : "  key = validasiKey()  CALL plain(plainText, key)  CASE 2:  CALL system("cls")  WRITE " Masukan Ciphertext : "  READ cipherText  WRITE "Masukan Jumlah Pergeseran : "  key = validasiKey()  cipher(ciphertext, key)  DEFAULT:  WRITE " Maaf Input anda salah !"  WRITE " Masukan Input Dengan Benar"  CALL mainMenu()  ENDSWITCH  CALL system("pause")  read = ulang()  WHILE(read == 1)  END  PROCEDURE namaKelompok() : void  DESKRIPSI:  BEGIN  CALL system("cls")  WRITE "namakelompok"  CALL system("cls")  END  FUNCTION main() : int  DESKRIPSI:  BEGIN  CALL namaKelompok()  CALL mainMenu()  END |

Kode Program 8.5 *Pseudocode* Program Enkripsi dan Dekripsi

Kode Program 8.5 di atas merupakan *pseudocode* dari program enkripsi dan dekripsi. Program akan meminta pengguna untuk memilih ingin melakukan enkripsi atau dekripsi. Setelah itu pengguna diminta menginput kata atau kalimat yang akan dikonversi dan jumlah pergeserannya. Setelah itu hasil dari enkripsi atau dekripsi akan ditampilkan sebagai *output*.

#### Trace Program Enkripsi dan Deskripsi

*Trace* pada program merupakan salah satu hal yang penting. *Trace* berguna untuk mengetahui alur program bekerja dan dapat digunakan untuk mendeteksi kesalahan yang ada pada program sehingga kesalahan bisa ditemukan dan diperbaiki. *Trace* pada program *ciphertext* bisa dilihat pada tabel.

Tabel 8.5 *Trace* Program Enkripsi dan Dekripsi

|  |  |
| --- | --- |
| *Input* | Masukan Pilihan Anda (1/2) = 1  Plaintext = kami cinta praktikum  Jumlah pergeseran = 6  Ingin Mencoba Menginput Kembali (1/2) = 2 |
| Proses | CALL plain()  for(int i = 0; i < strlen(plainText)-1;i++){  if ((int)plainText[i] >= 97 && (int)plainText[i] <= 122 || (int)plainText[i] == 32){  if((int)plainText[i] == 32) a = 32;  else a = ((int)plainText[i] - 97 + key) % 26 + 65;  // printf("%c = %d\n",plainText[i],a);  hasil[i] = (char)(a);  }  }  Return binary; |
| *Output* | Tampilkan plain()  Ciphertext = QGSO IOTZG VXGQZOQAS  Terima kasih |

Tabel 8.5 di atas menjelaskan bagaimana program ini berjalan, pertama pengguna memilih opsi nomor 1 yaitu enkripsi, lalu pengguna akan meng-*input* *plain text* yang ingin dienkripsi menjadi *cipher text* dan jumlah pergeseran yang dalam *trace* ini plaintext = “kami cinta praktikum” dan jumlah pergeseran = 6, lalu program akan menjalankan fungsi untuk mengenkripsi *plain text* dengan jumlah pergeseran yang sudah di-*input*, setelah fungsi dijalankan program akan menampilkan hasil dari enkripsi yang dalam *trace* ini “QGSO IOTZG VXGQZOQAS”. Lalu program akan menampilkan menu ulang untuk menanyakan pengguna apakah ingin mengulang program (1 = ya, 2 = tidak), jika pengguna memilih 1 maka program akan kembali berulang, namun jika pengguna memilih 2 maka menu terima kasih akan muncul dan program akan berakhir.

# Daftar Pustaka

GeeksforGeeks, 2022. *Arrays in C/C++.* [Online]   
Available at: https://www.geeksforgeeks.org/arrays-in-c-cpp/#:~:text=An%20array%20in%20C%2FC%2B%2B,using%20indices%20of%20an%20array.  
[Diakses 1 November 2022].

GeeksforGeeks, 2022. *Pointers vs Array in C/C++.* [Online]   
Available at: https://www.geeksforgeeks.org/pointers-vs-array-in-c-cpp/  
[Diakses 2 November 2022].

Hanly, J. R. & Koffman, E. B., 2013. *Problem Solving and Program Design in C.* 7th penyunt. New Jersey: Pearson Education, Inc.

Navone, E. C., 2019. *Data Structures 101: Arrays — A Visual Introduction for Beginners.* [Online]   
Available at: https://www.freecodecamp.org/news/data-structures-101-arrays-a-visual-introduction-for-beginners-7f013bcc355a/#:~:text=Arrays%20are%20classified%20as%20Homogeneous,%2C%20objects%2C%20and%20so%20on.  
[Diakses 1 November 2022].

Pedamkar, P., 2022. *Random Number Generator in C.* [Online]   
Available at: https://www.educba.com/random-number-generator-in-c/  
[Diakses 2 November 2022].

Singh, A., 2018. *Sorting and Searching | Data Structure & Algorithms.* [Online]   
Available at: https://msatechnosoft.in/blog/searching-sorting-data-structure-algorithms/  
[Diakses 2 November 2022].

Srivastava, P., 2021. *Difference between Searching and Sorting Algorithms.* [Online]   
Available at: https://www.geeksforgeeks.org/difference-between-searching-and-sorting-algorithms/  
[Diakses 2 November 2022].

Thompson, B., 2022. *Pointers in C: What is Pointer in C Programming? Types.* [Online]   
Available at: https://www.guru99.com/c-pointers.html  
[Diakses 2 November 2022].

TutorialCup, 2022. *Array Memory Allocation in C Programming.* [Online]   
Available at: https://www.tutorialcup.com/cprogramming/array-memory-allocation.htm  
[Diakses 1 November 2022].