LEMBAR ASISTENSI
PRAKTIKUM STRUKTUR DATA
LABORATORIUM TEKNIK KOMPUTER
JURUSAN TEKNIK ELEKTRO FAKULTAS TEKNIK
UNIVERSITAS LAMPUNG

Judul Praktikum : VARIABEL STRUKTUR DATA

Praktikan (NPM) : Fakhry Ramadhan Subur (2415061113)

Asisten (NPM) : M. Favian Rizki (2315061067)

Dimas Faqih Nur A.R. (2315061059)

Alexandrio Ariel Rafidan (2315061071)

Ramadhani Ahmad (2315061001)

Radhitya Agrayasa R. (2315061002)

Kelas : PSTI C

No	Catatan	Tanggal	Paraf

Bandar Lampung, 2025

M. Favian Rizki

NPM. 2315061067

JUDUL PERCOBAAN

JUDUL PERCOBAAN VARIABEL STRUKTUR DATA

I. TUJUAN PERCOBAAN

Adapun tujuan dari percobaan ini adalah sebagai berikut.

1. Dapat membuat dan memahami penggunaan array
2. Dapat membuat dan memahami penggunaan linked list
3. Dapat membuat dan memahami penggunaan vector

II. TEORI DASAR

3.1 Array

Array adalah struktur data yang digunakan untuk menyimpan sekumpulan data yang memiliki tipe yang sama dalam satu variabel. Dengan array, pengelolaan data menjadi lebih mudah karena setiap elemen memiliki indeks yang dapat digunakan untuk mengakses datanya. Menurut Baumann et al. (2021), array sangat penting dalam pengelolaan data berskala besar karena dapat mengorganisasi data secara terstruktur dan mempercepat proses pengolahan informasi. Harris et al. (2020) juga menegaskan bahwa array merupakan dasar pemrograman ilmiah modern, karena efisiensinya dalam menangani data yang jumlahnya besar.

Dalam pemrograman, terdapat beberapa jenis array, namun yang paling umum digunakan adalah array satu dimensi (1D) dan array dua dimensi (2D). Array satu dimensi digunakan untuk menyimpan data linear atau satu baris, seperti daftar nilai atau daftar harga barang. Sedangkan array dua dimensi digunakan untuk

menyimpan data yang berbentuk tabel dengan baris dan kolom, seperti data nilai beberapa mahasiswa dalam berbagai mata kuliah. Perbedaan utama dari kedua jenis array ini terletak pada jumlah indeks yang digunakan untuk mengakses elemen array.

Perbedaan ini dapat dilihat pada contoh kode berikut. Pada array 1D, hanya diperlukan satu indeks untuk mengakses elemen, sedangkan pada array 2D dibutuhkan dua indeks: satu untuk baris dan satu untuk kolom, contoh Perbedaan Implementasi Array 1D dan 2D dalam C++.

Array Satu Dimensi (1D) :
Digunakan untuk data linear, misalnya daftar nilai lima mahasiswa.

```cpp
int nilai[5] = {80, 85, 90, 75, 88}; // Deklarasi array 1D

// Mengakses data menggunakan satu indeks
cout << "Nilai Mahasiswa ke-1: " << nilai[0] << endl;
```

Gambar 3.1.1

Penjelasan:
Pada contoh ini, nilai[5] menyimpan lima nilai mahasiswa. Elemen array diakses menggunakan satu indeks, misalnya nilai[0] untuk mahasiswa pertama. Struktur array ini sederhana dan cocok untuk data yang berbentuk linear.

Array Dua Dimensi (2D) :
Digunakan untuk data berbentuk tabel, misalnya nilai tiga mahasiswa untuk dua mata kuliah.

```
int nilai[3][2] = {
    {80, 85},  // Mahasiswa 1
    {90, 75},  // Mahasiswa 2
    {88, 92}   // Mahasiswa 3
};

// Mengakses data menggunakan dua indeks
cout << "Nilai Mahasiswa ke-1 Mata Kuliah 1: " << nilai[0][0] << endl;
```

Gambar 3.1.2

Penjelasan:
Pada contoh ini, nilai[3][2] memiliki tiga baris dan dua kolom. Elemen array diakses menggunakan dua indeks, misalnya nilai[0][0] untuk mahasiswa pertama di mata kuliah pertama. Bentuk array ini cocok untuk data yang lebih kompleks, seperti tabel nilai atau data penjualan.

Dengan demikian, perbedaan mendasar antara array 1D dan 2D terletak pada jumlah indeks yang digunakan. Array 1D menggunakan satu indeks sehingga cocok untuk data sederhana yang berbentuk linear, sedangkan array 2D menggunakan dua indeks yang memungkinkan pengelolaan data dalam bentuk baris dan kolom. Baumann et al. (2021) menyebutkan bahwa array multidimensi seperti array 2D penting untuk mengelola data berskala besar dengan struktur yang lebih kompleks. Harris et al. (2020) juga menambahkan bahwa pemahaman jenis array akan membantu programmer membuat program yang lebih efisien dan terstruktur.

3.2 LinkedList

Linked list adalah salah satu struktur data dinamis yang tersusun dari rangkaian elemen yang disebut node, di mana setiap node saling terhubung melalui pointer. Berbeda dengan array yang bersifat statis dan menggunakan indeks untuk mengakses elemen, linked list dapat bertambah atau berkurang ukurannya secara dinamis, sehingga sangat efisien untuk aplikasi yang jumlah datanya tidak tetap. Menurut Baumann et al. (2021), linked list memanfaatkan memori lebih fleksibel karena elemen hanya dialokasikan saat dibutuhkan dan dilepas ketika tidak digunakan. Setiap node dalam linked list umumnya memiliki dua komponen utama, yaitu data field untuk menyimpan nilai, dan pointer untuk menyimpan alamat node lain yang terhubung.

Terdapat dua jenis linked list yang paling umum digunakan, yaitu single linked list dan double linked list. Single linked list hanya memiliki satu pointer yang menunjuk ke node berikutnya, sehingga traversal hanya dapat dilakukan dalam satu arah, dari awal menuju akhir. Jenis ini lebih sederhana dan memerlukan memori yang lebih sedikit, namun memiliki keterbatasan ketika ingin bergerak mundur ke node sebelumnya. Sedangkan double linked list memiliki dua pointer, yaitu next untuk menunjuk ke node berikutnya dan prev untuk menunjuk ke node sebelumnya. Hal ini memungkinkan traversal dilakukan dua arah, baik maju maupun mundur, sehingga lebih fleksibel dalam pengelolaan data. Namun, kelemahan double linked list adalah kebutuhan memori yang lebih besar karena setiap node memerlukan dua pointer.

Perbedaan utama antara single dan double linked list terletak pada struktur node dan arah traversal. Perbedaan ini dapat dilihat dari cuplikan kode berikut.

1. Single Linked List

Struktur node hanya memiliki satu pointer next yang mengarah ke node berikutnya, contoh kode :

```
struct Node {
    int data;     // Menyimpan data
    Node* next;   // Menunjuk ke node berikutnya
};
```

Gambar 3.1.3

Penjelasan per baris:

1. struct Node {

- Kata kunci struct digunakan untuk mendefinisikan sebuah struktur data.
- Struktur (struct) ini akan menjadi cetakan (blueprint) untuk membuat elemen dalam linked list yang disebut node.
- Nama struktur adalah Node, yang nantinya digunakan sebagai tipe data baru, mirip seperti int atau float.

Dengan struktur ini, kita hanya bisa bergerak maju dari satu node ke node selanjutnya. Jika ingin kembali ke node sebelumnya, traversal harus dimulai dari awal.

2. Double Linked List

Struktur node memiliki dua pointer: next untuk node berikutnya dan prev untuk node sebelumnya, contoh kode :

```
struct Node {
    int data;     // Menyimpan data
    Node* next;   // Menunjuk ke node berikutnya
    Node* prev;   // Menunjuk ke node sebelumnya
};
```

Gambar 3.1.4

Penjelasan perbaris :

1. struct Node {

- Kata kunci struct digunakan untuk mendefinisikan sebuah struktur data baru dalam C++.
- Nama strukturnya adalah Node.
- Struktur ini digunakan sebagai cetakan (blueprint) untuk membuat elemen-elemen dalam double linked list.
- Dengan struct Node, kita dapat membuat banyak node dengan format yang sama.

Dengan struktur ini, traversal dapat dilakukan maju dan mundur, sehingga memudahkan penghapusan, penambahan, maupun pencarian data pada posisi tertentu dalam linked list.

Secara keseluruhan, single linked list lebih sederhana dan hemat memori, sehingga cocok digunakan jika hanya diperlukan traversal satu arah dan pengelolaan data yang sederhana. Sebaliknya, double linked list lebih fleksibel dalam pengolahan data yang kompleks karena traversal dapat dilakukan dua arah, walaupun memerlukan memori yang lebih besar. Baumann et al. (2021) menyatakan bahwa pemilihan jenis linked list harus disesuaikan dengan kebutuhan aplikasi, sementara Harris et al. (2020) menekankan bahwa memahami struktur linked list sangat penting untuk membangun program yang efisien dan terstruktur.

3.2 Vector

Vector dalam C++ merupakan salah satu struktur data yang disediakan oleh Standard Template Library (STL) dan berfungsi sebagai array dinamis. Berbeda dengan array biasa yang memiliki ukuran tetap, vector dapat bertambah atau berkurang ukurannya secara otomatis selama program berjalan, sehingga lebih fleksibel dalam pengelolaan data. Menurut Harris et al. (2020), vector menyimpan data dalam memori kontigu, sama seperti array statis, sehingga memungkinkan akses acak (random access) ke elemen tertentu dalam waktu konstan, yaitu O(1). Vector memiliki dua properti utama, yaitu size dan capacity.

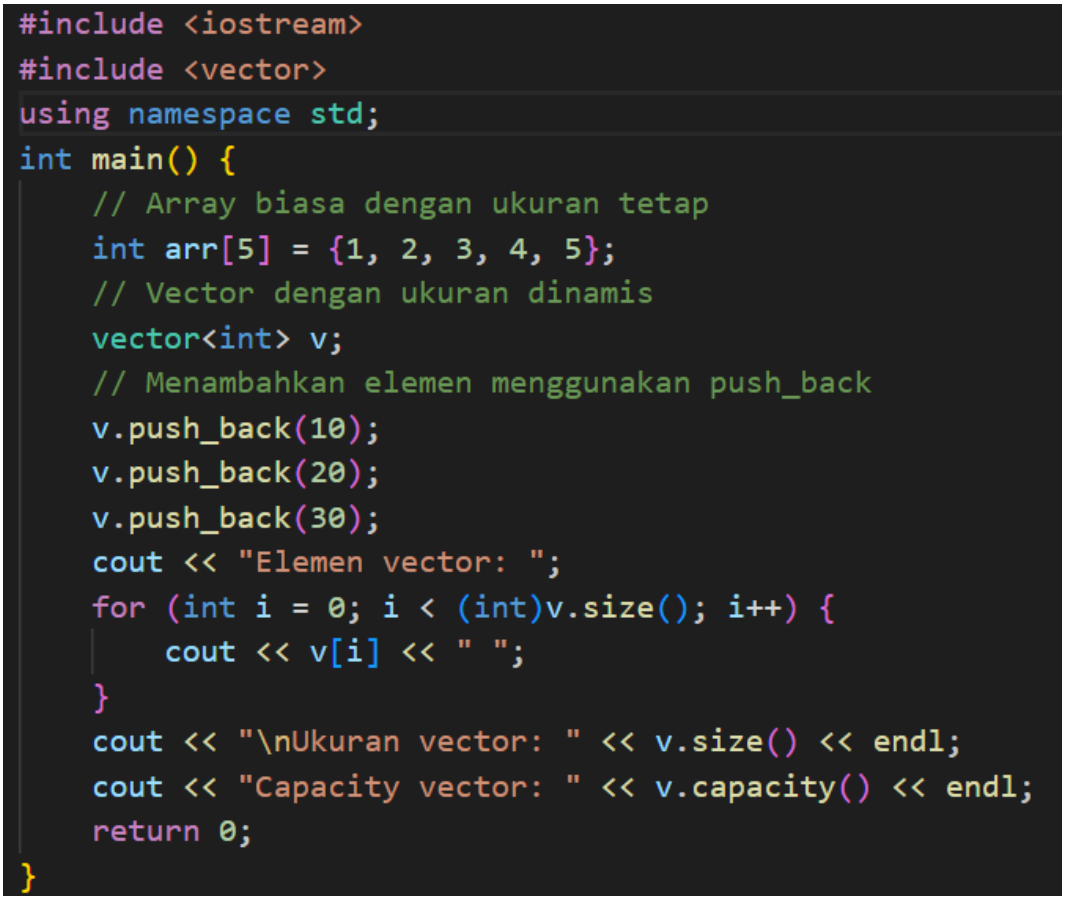
Size menunjukkan jumlah elemen yang sedang digunakan, sedangkan capacity menunjukkan jumlah elemen maksimum yang dapat ditampung sebelum vector melakukan reallocation (pengalokasian ulang memori). Proses reallocation terjadi secara otomatis ketika jumlah elemen yang ditambahkan melebihi kapasitas yang tersedia, di mana vector akan mengalokasikan blok memori yang lebih besar dan menyalin data lama ke lokasi yang baru (Baumann et al., 2021).

Operasi dasar pada vector meliputi penambahan elemen, penghapusan elemen, pengaksesan elemen, serta pengaturan ukuran dan kapasitas. Untuk menambahkan elemen, vector menggunakan fungsi push_back(), yang secara otomatis menempat kan data baru di akhir vector. Jika kapasitas penuh, vector akan melakukan reallocation terlebih dahulu. Untuk menghapus elemen terakhir, digunakan fungsi pop_back(). Selain itu, vector juga memiliki fungsi resize() untuk mengubah ukuran secara manual, dan reserve() untuk mengatur kapasitas awal agar tidak sering terjadi reallocation. Akses elemen dapat dilakukan menggunakan operator[] seperti pada array biasa atau at() yang dilengkapi pengecekan batas (bound checking). Traversal pada vector dapat dilakukan dengan perulangan berbasis indeks ataupun menggunakan iterator yang disediakan oleh STL (Goodrich et al., 2021).

Kelebihan vector adalah efisiensi akses data, karena penyimpanan yang kontigu membuatnya sangat cepat untuk membaca atau menulis elemen, serta memberikan keuntungan dalam hal locality of reference sehingga performa cache menjadi optimal. Selain itu, pengelolaan memori dilakukan secara otomatis sehingga programmer tidak perlu melakukan malloc atau delete secara manual seperti pada array dinamis tradisional. Namun, vector juga memiliki kekurangan, seperti biaya tinggi saat menambahkan atau menghapus elemen bukan di bagian akhir, karena elemen-elemen setelahnya harus digeser satu posisi, yang memakan waktu O(n). Selain itu, setiap kali terjadi reallocation, semua pointer atau iterator yang mengarah ke elemen vector menjadi tidak valid (invalidated), sehingga harus ditangani dengan hati-hati (Wazir, 2020).

Perbedaan utama antara array biasa dan vector terletak pada sifat dinamisnya. Array biasa memiliki ukuran tetap yang ditentukan di awal, sedangkan vector

dapat berubah ukurannya sesuai kebutuhan program. Berikut cuplikan kode sederhana yang memperlihatkan perbedaan mendasar antara array statis dan vector:



```cpp
#include <iostream>
#include <vector>
using namespace std;
int main() {
    // Array biasa dengan ukuran tetap
    int arr[5] = {1, 2, 3, 4, 5};
    // Vector dengan ukuran dinamis
    vector<int> v;
    // Menambahkan elemen menggunakan push_back
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    cout << "Elemen vector: ";
    for (int i = 0; i < (int)v.size(); i++) {
        cout << v[i] << " ";
    }
    cout << "\nUkuran vector: " << v.size() << endl;
    cout << "Capacity vector: " << v.capacity() << endl;
    return 0;
}
```

Gambar 3.1.5

Dalam contoh di atas, array arr memiliki ukuran tetap sebanyak lima elemen yang sudah ditentukan sejak awal. Sementara itu, vector v awalnya kosong namun dapat bertambah ukurannya secara otomatis setiap kali push_back() dipanggil. Fungsi size() digunakan untuk mengetahui jumlah elemen yang ada saat ini, sedangkan capacity() menunjukkan kapasitas memori yang telah dialokasikan. Perbedaan ini menjadikan vector pilihan yang tepat untuk situasi di mana ukuran data tidak dapat diprediksi atau dapat berubah-ubah selama program berjalan.

Dengan fleksibilitas yang dimiliki, vector banyak digunakan dalam berbagai aplikasi modern, mulai dari pemrosesan data, struktur data kompleks, hingga pengembangan perangkat lunak berbasis algoritma. Menurut Wazir (2020), memahami konsep vector sangat penting dalam pemrograman C++ karena vector menjadi dasar dari banyak struktur data lain di STL, seperti stack, queue, dan priority queue.

III. PROSEDUR PERCOBAAN

Adapun prosedur dari percobaan ini adalah sebagai berikut:

III.1 Percobaan 1: Array

4.1.1 Percobaan 1-1: Array 1 Dimensi

```
Array1D.cpp > main()
1   include <iostream>
2   sing namespace std;
3   oid menu () {
4       cout << "1. Tampilkan address array \n";
5       cout << "2. Tampilkan address dari semua index array \n";
6       cout << "3. Masukkan nilai kedalam semua index array \n";
7       cout << "4. Keluar \n";
8
9   nt main () {
10      int a[5];
11      int choice;
12      bool running = true;
13      while (running) {
14          menu ();
15          cin >> choice;
16          switch (choice) {
17              case 1 :
18                  cout << &a << endl;
19                  break;
20              case 2 :
21                  for (int i=0; i<5; i++){
22                      cout << &a[i] << endl;
23                  }
24                  break;
25              case 3 :
26                  for (int i =0; i<5; i++) {
27                      cin >> a[i];
28                  }
29                      break;
30              case 4 :
30              case 4 :
31                  running = false;
32                  return 0;
33          }
34      }
```

Gambar 4.1.1 Source Code Percobaan 1-1: Array 1 Dimensi

4.1.2 Percobaan 1-2 : Array 2 Dimensi

Array2D.cpp > main()

```cpp
#include <iostream>
using namespace std;
int menu () {
    cout << "1. Tampilkan address array \n";
    cout << "2. Tampilkan address dari semua index array \n";
    cout << "3. Masukkan nilai kedalam semua index array \n";
      cout << "4. Keluar \n";
}
void main () {
    int b[3][2];
    int choice;
    bool running = true;
    while (running) {
        menu ();
        cin >> choice;
        switch (choice) {
            case 1 :
                cout << &b << endl;
                break;
            case 2 :
                for (int i=0; i<3; i++){
                    for (int j=0; j < 2; j++){
                        cout << &b[i][j] << endl;
                    }
                }
                break;
            case 3 :
                for (int i =0; i<3; i++) {
                    for (int j=0; j < 2; j++){
                        cin >> b[i][j];
                    }
                }
                    break;
            case 4 :
                running = false;
                return 0;
        }
    }
}
```

*Gambar 4.1.2 Source Code Percobaan 1-2:*Array 2 Dimensi

III.2 Percobaan 2 : LinkedList

4.2.1 Percobaan Single LinkedList

SingleLinkedList.cpp > display(node *)

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

struct node {
    int data;
    node *next;
} *start, *newptr, *save, *ptr, *rear;

node *create_new_node(int);
void insert_at_beg(node *);
void insert_at_end(node *);
void display(node *);
void delete_node();

int main() {
    start = rear = NULL;
    int c;
    char choice = 'y';
    int insert;

    while (choice == 'y' || choice == 'Y') {
        cout << "Nilai baru : ";
        cin >> c;
        cout << "Membuat node baru" << endl;
        newptr = create_new_node(c);
        if (newptr != NULL) {
            cout << "Berhasil membuat node baru" << endl;
        } else {
            cout << "Node baru tidak dapat dibuat" << endl;
            exit(1);
        }

        cout << "1. Depan\n";
        cout << "2. Belakang\n";
        cout << "Masukkan dimana? ";
        cin >> insert;

        switch (insert) {
            case 1:
                insert_at_beg(newptr);
                cout << "Node dimasukkan di awal list" << endl;
                break;
            case 2:
                insert_at_end(newptr);
                cout << "Node dimasukkan di akhir list" << endl;
                break;
        }

        cout << "List : ";
        display(start);

        cout << "Mau membuat node baru? (y/n): ";
        cin >> choice;
    }

    do {
        cout << "List : ";
```

```
58          display(start);
59
60          cout << "Mau menghapus node pertama? (y/n): ";
61          cin >> choice;
62          if (choice == 'y' || choice == 'Y') {
63              delete_node();
64          }
65      } while (choice == 'y' || choice == 'Y');
66
67      return 0;
68  }
69
70  node *create_new_node(int n) {
71      ptr = new node;
72      ptr->data = n;
73      ptr->next = NULL;
74      return ptr;
75  }
76
77  void insert_at_beg(node *np) {
78      if (start == NULL) {
79          start = rear = np;
80      } else {
81          save = start;
82          start = np;
83          np->next = save;
84      }
85  }
```

```
85  }
86
87  void insert_at_end(node *np) {  //sebelumnya yg di modul ini in b
88      if (start == NULL) {
89          start = rear = np;
90      } else {
91          rear->next = np;
92          rear = np;
93      }
94  }
95
96  void delete_node() {
97      if (start == NULL) {
98          cout << "Underflow!!!" << endl;
99      } else {
100         ptr = start;
101         start = start->next;
102         delete ptr;
103     }
104 }
105
106 void display(node *np) {   //ini ga sama kek di modul
107     while (np != NULL) {
108         cout << np->data << " ";
109         np = np->next;
110     }
111     cout << endl;
112 }
```

Gambar 4.2.1 Source Code Percobaan SingleLinkedList

4.2.2 Percobaan DoublyLinkedList

DoublyLinkedList.cpp > traverseBackward(node *)

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

struct node{
    int data;
    node *prev;
      node *next;
} *start, *newptr, *ptr, *rear;

node *create_new_node(int);
void insert_node(node *);
void traverseForward(node *);
void traverseBackward(node *);

int main() {
    start = rear = NULL;
    int c;
    char choice = 'y';
    while (choice == 'y' || choice == 'Y') {
        cout << "Nilai baru = ";
        cin >> c;
        cout << "Membuat node baru" << endl;
        newptr = create_new_node(c);
        if(newptr != NULL) {
            cout << "Berhasil membuat node baru" << endl;
        }
        else {
            cout << "Node baru tidak dapat dibuat";
            exit(1);
        }
```

```cpp
        insert_node(newptr);
        cout << "Node dimasukkan ke list" << endl;
        cout << "Mau membuat node baru? (y/n)";
        cin >> choice;
    }
    int direction;
    cout << "1. Maju" << endl;
    cout << "2. Mundur" << endl;
    cout << "Arah transversal yang mana?";
    cin >> direction;
    switch (direction){
        case 1 :
        cout << "Traversal maju: ";
        traverseForward(start);
        break;
        case 2 :
        cout << "Transversal mundur: ";
        traverseBackward(rear);

    }
    return 0;
}
node *create_new_node(int n){
    ptr = new node;
    ptr->data = n;
    ptr->next = NULL;
    ptr->prev = NULL;
```

```
58      return ptr;
59  }
60
61  void insert_node(node *np) {
62      if(start == NULL){
63          start = rear = np;
64      }
65      else{
66          rear->next = np;
67          rear = np;
68      }
69  }
70  void traverseForward(node* head) {
71      node* current = head;
72      while (current != NULL) {
73          cout << current->data << " ";
74          current = current->next;
75      }
76      cout << endl;
77  }
78
79  void traverseBackward(node* tail) {
80      node* current = tail;
81      while (current != NULL) {
82          cout << current->data << " ";
83          current = current->prev;
84      }
85      cout << endl;
86  }
```

Gambar 4.2.2 Source Code DoublyLinkedList

4.2.2 Percobaan Vector

vector.cpp > ...

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

struct Vector {
    int* data;
    int capacity;
    int length;
};

void init(Vector& v) {
    v.capacity = 2;
    v.length = 0;
    v.data = new int[v.capacity];
}

void resize(Vector& v, int newCap) {
    int* newData = new int[newCap];
    for (int i = 0; i < v.length; i++) {
        newData[i] = v.data[i];
    }
    delete[] v.data;
    v.data = newData;
    v.capacity = newCap;
}

void push_back(Vector& v, int value) {
    if (v.length == v.capacity) {
        resize(v, v.capacity * 2);
    }
    v.data[v.length] = value;
    v.length++;
}

void pop_back(Vector& v) { ···

int get(Vector& v, int index) {
    if (index >= 0 && index < v.length) {
        return v.data[index];
    }
    return -1; // Return a default value if index is invalid
}

void set(Vector& v, int index, int value) {
    if (index >= 0 && index < v.length) {
        v.data[index] = value;
    }
}

int size(Vector& v) {
    return v.length;
}

void display(Vector& v) {
    cout << "[";
    for (int i = 0; i < v.length; i++) {
        cout << v.data[i];
        if (i < v.length - 1) cout << ", ";
    }
    cout << "]\n";
```

```
}

void clear(Vector& v) {
    delete[] v.data;
    v.data = nullptr;
    v.capacity = 0;
    v.length = 0;
}

int main() {
    Vector v;
    init(v);
    push_back(v, 10);
    push_back(v, 20);
    push_back(v, 30);

    cout << "Isi vector: ";
    display(v);

    cout << "Elemen ke-1 = " << get(v, 1) << endl;

    set(v, 1, 99);
    cout << "Set elemen ke-1 jadi 99: ";
    display(v);

    pop_back(v);
    cout << "Setelah pop_back: ";
    display(v);

    cout << "Ukuran vector sekarang = " << size(v) << endl;
    clear(v);
return 0;
}
```

Gambar 4.2.2 Source Code Vector

IV. PEMBAHASAN

Adapun pembahasan dari percobaan ini adalah sebagai berikut:

IV.1 Percobaan Array

5.1.1 Percobaan 1. 1: Array 1 Dimensi

5.1.1.a Source Code Percobaan 1.1

Array1D.cpp > main()

```
1   include <iostream>
2   sing namespace std;
3   oid menu () {
4       cout << "1. Tampilkan address array \n";
5       cout << "2. Tampilkan address dari semua index array \n";
6       cout << "3. Masukkan nilai kedalam semua index array \n";
7       cout << "4. Keluar \n";
8
9   nt main () {
10      int a[5];
11      int choice;
12      bool running = true;
13      while (running) {
14          menu ();
15          cin >> choice;
16          switch (choice) {
17              case 1 :
18                  cout << &a << endl;
19                  break;
20              case 2 :
21                  for (int i=0; i<5; i++){
22                      cout << &a[i] << endl;
23                  }
24                  break;
25              case 3 :
26                  for (int i =0; i<5; i++) {
27                      cin >> a[i];
28                  }
29                      break;
30              case 4 :
30              case 4 :
31                  running = false;
32                  return 0;
33          }
34      }
```

Gambar 5.1.1.a Source Code Percobaan 1.1

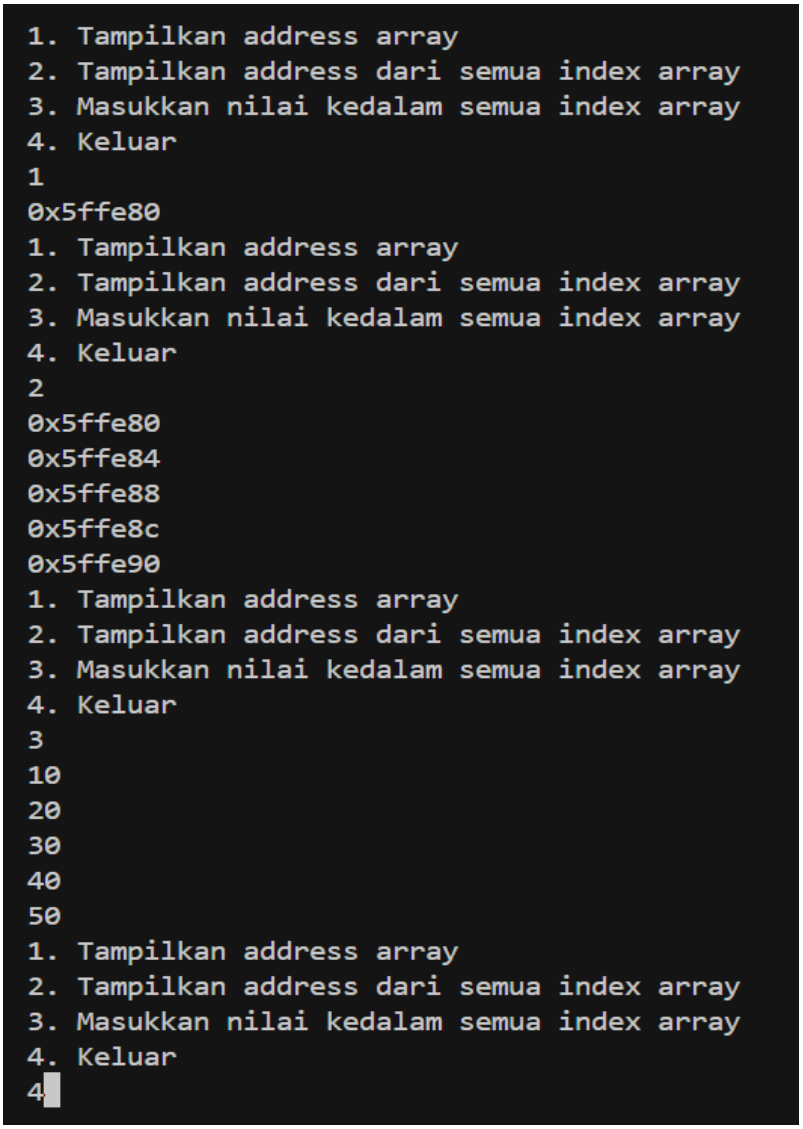
Berdasarkan gambar 5.1.1.a, pada baris pertama program menyertakan pustaka standar iostream yang digunakan untuk menangani proses input maupun output, seperti menampilkan data di layar atau membaca masukan dari keyboard. Selanjutnya, pada baris kedua ditulis perintah using namespace std; yang bertujuan agar pemanggilan fungsi standar seperti cout dan cin bisa dilakukan tanpa menuliskan awalan namespace. Pada baris 3 hingga 8 terdapat fungsi menu() yang berfungsi menampilkan daftar pilihan bagi pengguna.

Program utama dimulai dari fungsi main() pada baris ke-9 sebagai titik awal eksekusi. Di dalamnya, pada baris 10 di deklarasikan sebuah array satu dimensi a yang dapat menampung lima bilangan bulat. Kemudian, pada baris 11 dibuat variabel choice untuk menyimpan input pilihan pengguna, sedangkan baris 12 mendefinisikan variabel running dengan nilai awal true untuk menjaga agar perulangan tetap berjalan.

Perulangan utama program dimulai pada baris 13 menggunakan while, artinya program akan terus berulang selama variabel running bernilai true. Pada setiap putaran, fungsi menu() dipanggil (baris 14) untuk menampilkan daftar opsi, lalu program menunggu masukan pilihan dari pengguna melalui cin (baris 15). Input tersebut kemudian diproses menggunakan struktur switch pada baris 16.

Jika pengguna memilih 1, program mengeksekusi baris 18 untuk menampilkan alamat dasar array a. Apabila pengguna memilih 2, maka program masuk ke case kedua (baris 21–23) yang melakukan perulangan sebanyak lima kali untuk menampilkan alamat setiap elemen array secara berurutan, mulai dari a[0] hingga a[4]. Pada pilihan 3, program menjalankan perulangan (baris 27–29) yang meminta pengguna memasukkan lima nilai untuk mengisi seluruh elemen array. Terakhir, jika pengguna memilih 4, program mengubah nilai running menjadi false (baris 33) sehingga perulangan berhenti, kemudian perintah return 0 pada baris 34 dijalankan untuk mengakhiri program dengan benar.

5.1.1.b Output Percobaan 1.1



Gambar 5.1.1.b Output Percobaan 1.1

Berdasarkan gambar 5.1.1.b, ketika program dijalankan, perulangan while pada baris ke-13 mulai berjalan dan langsung memanggil fungsi menu() di baris ke-14 untuk menampilkan daftar pilihan. Menu tersebut berisi empat opsi: "1. Tampilkan address array" (baris 4), "2. Tampilkan address dari semua index array" (baris 5), "3. Masukkan nilai ke dalam semua index array" (baris 6), dan "4. Keluar" (baris 7). Setelah itu, program menunggu masukan pengguna melalui cin pada baris ke-15.

Pada percobaan pertama, pengguna mengetik angka 1, sehingga program mengeksekusi case 1 di baris ke-18 yang mencetak alamat dasar dari array a (misalnya 0x62fefffb80). Karena kondisi while masih bernilai true, menu kembali dipanggil di baris ke-14.Kemudian, pengguna memilih angka 2 (baris ke-15) yang memicu case 2. Di bagian ini terdapat perulangan for pada baris ke-21 yang dijalankan lima kali, di mana setiap iterasi mengeksekusi baris ke-22 untuk menampilkan alamat memori dari elemen a[0] sampai a[4]. Terlihat setiap alamat berbeda 4 byte, sesuai ukuran tipe data int.

Selanjutnya, menu ditampilkan lagi, lalu pengguna memasukkan angka 3 (baris ke-15). Input ini memanggil case 3, yaitu perulangan for pada baris 27–29. Perintah cin di baris ke-28 dijalankan lima kali untuk mengisi semua elemen array, misalnya dengan nilai 10, 20, 30, 40, dan 50.

Terakhir, saat menu muncul lagi (baris ke-14), pengguna memilih angka 4. Hal ini mengeksekusi case 4 pada baris ke-33, yang mengubah nilai variabel running menjadi false sehingga perulangan while di baris ke-13 berhenti. Program kemudian menutup dengan perintah return 0 di baris ke-34 sebagai tanda eksekusi selesai.

5.1.2 Percobaan 1.2:Array 2 Dimensi

5.1.2.a Source Code Percobaan 1.2

Array2D.cpp > main()

```cpp
#include <iostream>
using namespace std;
int menu () {
    cout << "1. Tampilkan address array \n";
    cout << "2. Tampilkan address dari semua index array \n";
    cout << "3. Masukkan nilai kedalam semua index array \n";
      cout << "4. Keluar \n";
}
void main () {
    int b[3][2];
    int choice;
    bool running = true;
    while (running) {
        menu ();
        cin >> choice;
        switch (choice) {
            case 1 :
                cout << &b << endl;
                break;
            case 2 :
                for (int i=0; i<3; i++){
                    for (int j=0; j < 2; j++){
                        cout << &b[i][j] << endl;
                    }
                }
                break;
            case 3 :
                for (int i =0; i<3; i++) {
                    for (int j=0; j < 2; j++){
                        cin >> b[i][j];
                    }
                }
                    break;
            case 4 :
                running = false;
                return 0;
        }
    }
}
```

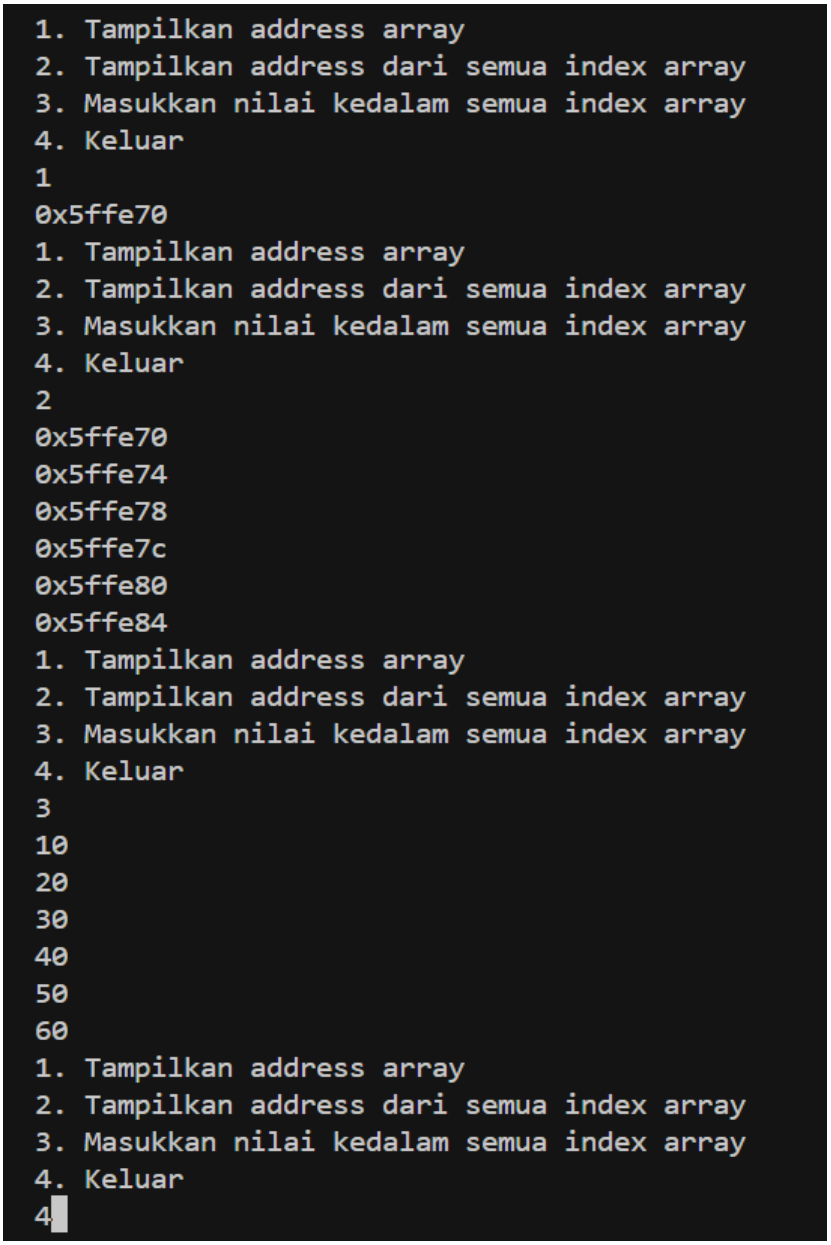
Gambar 5.1.2.a Source Code Percobaan 1.2

Berdasarkan gambar 5.1.2.a Pada kode program yang telah dibuat, terlihat bahwa struktur dasarnya mirip dengan percobaan sebelumnya, hanya saja lebih disusun

rapi. Program dimulai dengan menyertakan pustaka iostream agar dapat melakukan input dan output, serta penggunaan namespace std supaya penulisan perintah lebih sederhana. Selanjutnya, didefinisikan sebuah fungsi menu() yang bertugas menampilkan daftar pilihan kepada pengguna sekaligus mengembalikan input pilihan yang dimasukkan. Di dalam fungsi main(), dibuat sebuah array dua dimensi b[3][2] yang dapat menampung enam data integer dalam bentuk tabel berisi tiga baris dan dua kolom. Sebuah variabel boolean running dipakai untuk mengatur jalannya perulangan. Program berjalan dalam perulangan while, di mana setiap iterasi memanggil fungsi menu untuk menampilkan pilihan.

Jika pengguna memilih opsi pertama, program akan menampilkan alamat dasar array. Pada opsi kedua, program menggunakan dua perulangan bersarang untuk menampilkan alamat memori setiap elemen array, yaitu b[i][j], sehingga terlihat bahwa data disimpan secara kontigu. Pada opsi ketiga, perulangan bersarang digunakan kembali, tetapi kali ini untuk meminta input nilai dari pengguna dan menyimpannya ke setiap elemen array. Terakhir, jika opsi keempat dipilih, variabel running diubah menjadi false sehingga perulangan berhenti dan program mengakhiri eksekusi dengan menampilkan pesan bahwa program selesai.

5.1.2.b Output Percobaan 1.2



Gambar 5.1.2.b Output Percobaan 1.2

Berdasarkan gambar 5.1.2.b, eksekusi program dengan array dua dimensi b[3][2] yang dideklarasikan pada baris ke-8 dimulai melalui perulangan while pada baris ke-11. Perulangan ini pertama kali memanggil fungsi menu() (baris ke-6) untuk menampilkan empat pilihan yang ditulis pada baris 3 hingga 6. Pada langkah awal, pengguna memberikan input angka 1 melalui cin (baris ke-12).

Sesuai pilihan tersebut, program menjalankan case 1 yang mencetak alamat dasar array (base address) dari b, misalnya 0x762cfffb80, sebagaimana diperintahkan pada baris ke-14. Setelah itu, perulangan kembali menampilkan menu yang sama. Pada interaksi kedua, pengguna memilih angka 2, sehingga program masuk ke case 2 yang berisi dua perulangan bersarang (baris 17–22). Perulangan luar menelusuri baris, sedangkan perulangan dalam menelusuri kolom, sehingga perintah pada baris ke-19 dijalankan sebanyak enam kali untuk mencetak alamat memori setiap elemen, dari b[0][0] hingga b[2][1]. Hasilnya menunjukkan bahwa array dua dimensi juga dialokasikan secara berurutan di memori (sekuensial).

Menu kemudian ditampilkan kembali, dan kali ini pengguna memilih angka 3. Pada case 3, perulangan bersarang (baris 27–33) meminta pengguna memasukkan enam data yang dimasukkan ke array, misalnya nilai 10, 20, 30, 40, 50, dan 60. Terakhir, menu tampil untuk keempat kalinya dan pengguna memilih angka 4. Pilihan ini menjalankan case 4, di mana variabel running diubah menjadi false (baris ke-36), menyebabkan perulangan berhenti, dan program berakhir dengan normal pada baris ke-37.

V.2. Percobaan 2: Linked List

5.2.1 Percobaan 2.1: Single Linked List

5.2.1.a Source Code Percobaan 2.1

SingleLinkedList.cpp > display(node *)

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

struct node {
    int data;
    node *next;
} *start, *newptr, *save, *ptr, *rear;

node *create_new_node(int);
void insert_at_beg(node *);
void insert_at_end(node *);
void display(node *);
void delete_node();

int main() {
    start = rear = NULL;
    int c;
    char choice = 'y';
    int insert;

    while (choice == 'y' || choice == 'Y') {
        cout << "Nilai baru : ";
        cin >> c;
        cout << "Membuat node baru" << endl;
        newptr = create_new_node(c);
        if (newptr != NULL) {
            cout << "Berhasil membuat node baru" << endl;
        } else {
            cout << "Node baru tidak dapat dibuat" << endl;
            exit(1);
```

```
        }

        cout << "1. Depan\n";
        cout << "2. Belakang\n";
        cout << "Masukkan dimana? ";
        cin >> insert;

        switch (insert) {
            case 1:
                insert_at_beg(newptr);
                cout << "Node dimasukkan di awal list" << endl;
                break;
            case 2:
                insert_at_end(newptr);
                cout << "Node dimasukkan di akhir list" << endl;
                break;
        }

        cout << "List : ";
        display(start);

        cout << "Mau membuat node baru? (y/n): ";
        cin >> choice;
    }

    do {
        cout << "List : ";
```

```
        display(start);

        cout << "Mau menghapus node pertama? (y/n): ";
        cin >> choice;
        if (choice == 'y' || choice == 'Y') {
            delete_node();
        }
    } while (choice == 'y' || choice == 'Y');

    return 0;
}

node *create_new_node(int n) {
    ptr = new node;
    ptr->data = n;
    ptr->next = NULL;
    return ptr;
}

void insert_at_beg(node *np) {
    if (start == NULL) {
        start = rear = np;
    } else {
        save = start;
        start = np;
        np->next = save;
    }
}
```

```
85  }
86
87  void insert_at_end(node *np) {  //sebelumnya yg di modul ini in b
88      if (start == NULL) {
89          start = rear = np;
90      } else {
91          rear->next = np;
92          rear = np;
93      }
94  }
95
96  void delete_node() {
97      if (start == NULL) {
98          cout << "Underflow!!!" << endl;
99      } else {
100         ptr = start;
101         start = start->next;
102         delete ptr;
103     }
104 }
105
106 void display(node *np) {   //ini ga sama kek di modul
107     while (np != NULL) {
108         cout << np->data << " ";
109         np = np->next;
110     }
111     cout << endl;
112 }
```

Gambar 5.2.a Source Code Percobaan 2.1

Berdasarkan gambar 5.2.1.a, program diawali pada baris 1–2 dengan menambahkan pustaka iostream serta perintah using namespace std agar penulisan perintah input-output lebih sederhana. Struktur data utama ditentukan lewat struct node pada baris 4–7, yang menjadi rancangan setiap elemen list. Masing-masing node memiliki sebuah variabel bertipe int (baris 5) dan pointer next (baris 6) untuk menyambungkan node berikutnya. Beberapa pointer global juga disiapkan pada baris 7, seperti start untuk menunjuk node pertama, rear untuk node terakhir, serta beberapa pointer bantu lainnya. Selanjutnya, pada baris 9–13 dituliskan deklarasi fungsi-fungsi yang akan dipakai dalam program, mulai dari pembuatan node baru, penyisipan di depan atau belakang, menampilkan list, hingga penghapusan node.

Eksekusi dimulai dari fungsi main() pada baris 15. Di baris 16, pointer start dan rear diinisialisasi dengan NULL sebagai tanda bahwa list masih kosong. Variabel c, choice, dan insert (baris 17–19) disiapkan untuk menyimpan data dan pilihan pengguna. Perulangan while pada baris 21 dipakai untuk menambah node baru. Pengguna diminta mengisi nilai (baris 22), lalu nilai itu diproses oleh fungsi

create_new_node() (baris 25). Jika node berhasil dibuat, program menampilkan pesan sukses (baris 27–28), sebaliknya jika gagal akan menampilkan error (baris 30–31) dan berhenti. Setelah node siap, pengguna diminta memilih lokasi penyisipan (baris 34–37). Jika memilih depan, maka insert_at_beg() dipanggil (baris 40). Jika memilih belakang, insert_at_end() dieksekusi (baris 44). Setelah penyisipan, kondisi list ditampilkan melalui display() (baris 50), dan pengguna bisa memutuskan apakah ingin membuat node baru lagi (baris 52–53).

Setelah selesai menambahkan, program melanjutkan ke perulangan do-while pada baris 56 untuk proses penghapusan. Isi list ditampilkan lebih dulu (baris 57), lalu pengguna ditanya apakah ingin menghapus node pertama (baris 59). Jika jawaban y/Y, maka fungsi delete_node() dipanggil (baris 62) untuk menghapus node terdepan. Proses ini diulang sesuai kondisi di baris 64. Program kemudian berakhir dengan return 0 pada baris 66.

Fungsi-fungsi pembantu dituliskan setelah fungsi utama. create_new_node() (baris 69–73) bertugas membuat node baru, mengisi datanya, mengatur next menjadi NULL, dan mengembalikan alamat node tersebut. insert_at_beg() (baris 75–82) menempatkan node di depan; jika list kosong maka node langsung menjadi start dan rear, jika tidak maka node baru disambungkan ke node awal lama. insert_at_end() (baris 85–92) bekerja dengan cara menambahkan node di belakang list dan memperbarui rear. Fungsi delete_node() (baris 94–102) akan menghapus node pertama dengan cara menggeser pointer start dan menghapus node lama dari memori. Terakhir, display() (baris 104–109) menelusuri list dari awal hingga akhir sambil mencetak setiap nilai data yang tersimpan di node.

5.2.b Output Percobaan 2.1

```
Nilai baru : 10
Membuat node baru
Berhasil membuat node baru
1. Depan
2. Belakang
Masukkan dimana? 2
Node dimasukkan di akhir list
List : 10
Mau membuat node baru? (y/n): y
Nilai baru : 20
Membuat node baru
Berhasil membuat node baru
1. Depan
2. Belakang
Masukkan dimana? 1
Node dimasukkan di awal list
List : 20 10
Mau membuat node baru? (y/n): y
Nilai baru : 30
Membuat node baru
Berhasil membuat node baru
1. Depan
2. Belakang
Masukkan dimana? 1
Node dimasukkan di awal list
List : 30 20 10
Mau membuat node baru? (y/n): n
List : 30 20 10
Mau menghapus node pertama? (y/n): n
PS C:\Users\ACER\Downloads\Telegram Desktop\STRUKTUR DATA>
```

Gambar 5.2.b Output Percobaan 2.1

Berdasarkan gambar 5.2.b Pada gambar 5.2.1.b, saat program mulai dijalankan, perulangan while di baris 21 langsung bekerja. Program menampilkan teks "Nilai baru :" (baris 22), lalu pengguna memasukkan angka, misalnya 10 (baris 23). Setelah itu muncul pesan "Membuat node baru" (baris 24). Fungsi create_new_node() dipanggil untuk membuat node baru, dan karena berhasil, program menuliskan "Berhasil membuat node baru" (baris 26–27).

Program kemudian memperlihatkan pilihan lokasi penyisipan node, yaitu "1. Depan" dan "2. Belakang" (baris 34–35), disertai permintaan input "Masukkan

dimana?" (baris 36). Jika pengguna memilih angka 2 (baris 37), maka case 2 akan dijalankan (baris 44) dan program menampilkan "Node dimasukkan di akhir list" (baris 46). Setelah itu, isi list ditampilkan dengan teks "List :" (baris 50), diikuti angka 10 yang dicetak melalui fungsi display() (baris 51). Perulangan ditutup dengan pertanyaan "Mau membuat node baru? (y/n): " (baris 54). Jika jawaban pengguna y (baris 55), maka siklus dilanjutkan.

Pada pengulangan kedua, pengguna memberikan angka 20. Proses pembuatan node kembali dilakukan seperti sebelumnya, lalu menu penyisipan ditampilkan lagi (baris 34–36). Kali ini, pengguna memilih opsi 1 (baris 37), sehingga case 1 dijalankan (baris 40) dan program menampilkan pesan "Node dimasukkan di awal list" (baris 42). Tampilan list yang dihasilkan (baris 50–51) adalah 20 10. Karena jawaban pengguna tetap y (baris 55), program melanjutkan ke siklus berikutnya.

Pada siklus ketiga, nilai yang dimasukkan adalah 30. Node baru dibuat dan dipilih lagi opsi 1 (baris 37). Hasilnya, angka 30 ditambahkan di bagian depan list, sehingga output display() memperlihatkan urutan 30 20 10. Ketika program kembali menanyakan apakah ingin membuat node baru (baris 54), pengguna menjawab n (baris 55), sehingga perulangan while di baris 21 berhenti.

Setelah fase penambahan selesai, program masuk ke bagian penghapusan dengan perulangan do-while di baris 59. Isi list terakhir ditampilkan lebih dulu (baris 61–62), yaitu 30 20 10. Lalu, program menampilkan pertanyaan "Mau menghapus node pertama? (y/n): " (baris 64). Karena pengguna menjawab n (baris 65), kondisi perulangan pada baris 66 tidak terpenuhi, sehingga bagian ini berhenti. Program kemudian ditutup dengan return 0 pada baris 68.

5.2.2 Percobaan 2.2:Double Linked List

5.2.2..a Source Code Percobaan 2.2

DoublyLinkedList.cpp > traverseBackward(node *)

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

struct node{
    int data;
    node *prev;
      node *next;
} *start, *newptr, *ptr, *rear;

node *create_new_node(int);
void insert_node(node *);
void traverseForward(node *);
void traverseBackward(node *);

int main() {
    start = rear = NULL;
    int c;
    char choice = 'y';
    while (choice == 'y' || choice == 'Y') {
        cout << "Nilai baru = ";
        cin >> c;
        cout << "Membuat node baru" << endl;
        newptr = create_new_node(c);
        if(newptr != NULL) {
            cout << "Berhasil membuat node baru" << endl;
        }
        else {
            cout << "Node baru tidak dapat dibuat";
            exit(1);
        }
        insert_node(newptr);
        cout << "Node dimasukkan ke list" << endl;
        cout << "Mau membuat node baru? (y/n)";
        cin >> choice;
    }
    int direction;
    cout << "1. Maju" << endl;
    cout << "2. Mundur" << endl;
    cout << "Arah transversal yang mana?";
    cin >> direction;
    switch (direction){
        case 1 :
        cout << "Traversal maju: ";
        traverseForward(start);
        break;
        case 2 :
        cout << "Transversal mundur: ";
        traverseBackward(rear);

    }
    return 0;
}
node *create_new_node(int n){
    ptr = new node;
    ptr->data = n;
    ptr->next = NULL;
    ptr->prev = NULL;
```

```
58      return ptr;
59  }
60
61  void insert_node(node *np) {
62      if(start == NULL){
63          start = rear = np;
64      }
65      else{
66          rear->next = np;
67          rear = np;
68      }
69  }
70  void traverseForward(node* head) {
71      node* current = head;
72      while (current != NULL) {
73          cout << current->data << " ";
74          current = current->next;
75      }
76      cout << endl;
77  }
78
79  void traverseBackward(node* tail) {
80      node* current = tail;
81      while (current != NULL) {
82          cout << current->data << " ";
83          current = current->prev;
84      }
85      cout << endl;
86  }
```

Gambar 5.2.2.a Source Code Percobaan 2.2

Berdasarkan gambar 5.2.2.a, program diawali pada baris 1–2 dengan menambahkan pustaka iostream dan perintah using namespace std agar operasi input-output bisa dilakukan lebih sederhana. Struktur utama list didefinisikan pada baris 4–8 melalui struct node, yang kini berbeda dari single linked list karena setiap elemen memiliki dua pointer: prev untuk menunjuk node sebelumnya dan next untuk menunjuk node berikutnya. Hal ini memungkinkan penelusuran dilakukan baik dari depan ke belakang maupun sebaliknya. Pada baris 8 juga terdapat deklarasi pointer global seperti start (node pertama) dan rear (node terakhir) beserta pointer bantu lainnya. Fungsi-fungsi penting seperti pembuatan node, penyisipan, dan traversal dideklarasikan di baris 10–13.

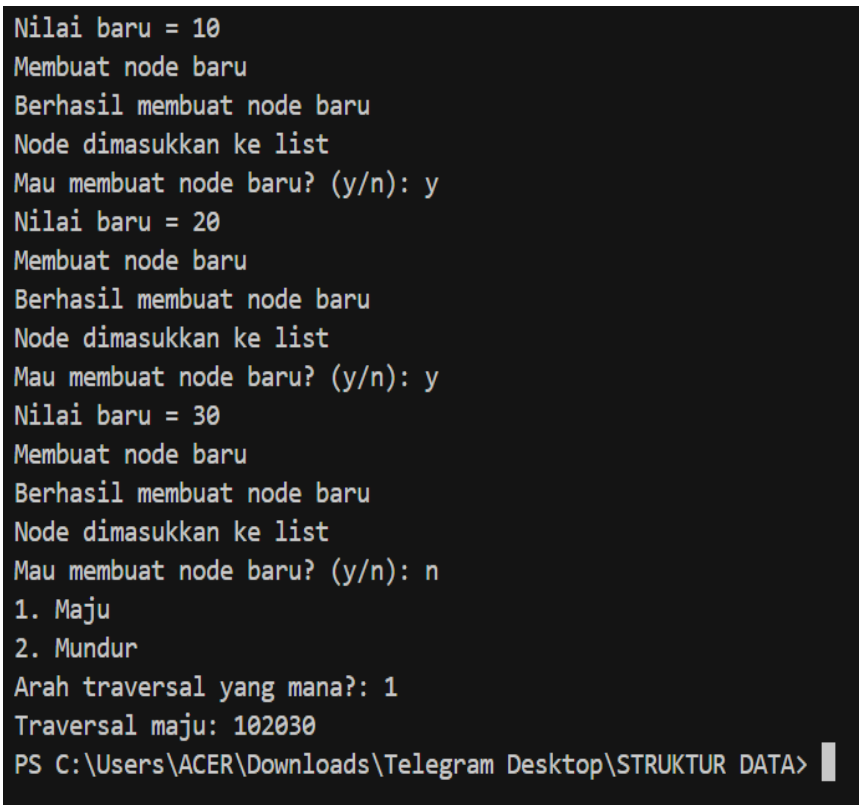
Fungsi utama main() dimulai di baris 15. Pada awalnya, pointer start dan rear diatur ke NULL (baris 16) sebagai penanda list kosong. Program kemudian masuk

ke perulangan while (baris 20) untuk menambahkan elemen baru. Baris 21–22 meminta input angka dari pengguna, lalu menampilkan pesan "Membuat node baru" (baris 23) sebelum memanggil fungsi create_new_node() (baris 24). Jika node berhasil dibuat, pesan keberhasilan muncul (baris 26–27), sedangkan jika gagal program memberi pesan error (baris 28–29) dan berhenti. Node yang valid kemudian dimasukkan ke list lewat insert_node() (baris 32), dan program memberi konfirmasi bahwa node berhasil ditambahkan (baris 33). Pengguna ditanya kembali (baris 35–36) apakah ingin membuat node lain; jawaban ini menentukan apakah perulangan akan berlanjut atau berhenti.

Setelah fase penambahan selesai, program menampilkan menu arah traversal pada baris 40–41. Input arah dari pengguna dibaca pada baris 42–43, lalu diproses oleh switch pada baris 45. Jika memilih opsi 1, program menuliskan "Traversal maju:" (baris 47) dan menjalankan fungsi traverseForward() (baris 48) untuk menampilkan isi list dari awal hingga akhir. Sebaliknya, jika pengguna memilih opsi 2, program menuliskan "Traversal mundur:" (baris 51) dan menjalankan fungsi traverseBackward() (baris 52) untuk menampilkan list dari belakang menuju awal. Program diakhiri dengan perintah return 0 (baris 55).

Fungsi pendukung dituliskan di bagian bawah kode. create_new_node() (baris 59–64) membuat node baru dengan mengalokasikan memori, mengisi data, serta mengatur pointer prev dan next ke NULL, kemudian mengembalikan alamat node. insert_node() (baris 66–73) menangani proses penyisipan: jika list kosong, node baru langsung jadi start dan rear, sedangkan jika list sudah terisi, node baru ditempatkan di belakang dengan memperbarui sambungan next pada node terakhir dan prev pada node baru. Fungsi traverseForward() (baris 75–82) mencetak isi list dari depan dengan menelusuri pointer next, sedangkan traverseBackward() (baris 84–91) mencetak list dari belakang dengan mengikuti pointer prev. Mekanisme traversal dua arah inilah yang menjadi ciri khas dari double linked list dibandingkan single linked list.

5.2.2.b Output Percobaan 2.2



Gambar 5.2.2.b Output Percobaan 2.2

Berdasarkan gambar 5.2.2.b terlihat proses eksekusi program dimulai dari perulangan while di baris ke-19. Program lebih dulu meminta input angka dengan menampilkan teks "Nilai baru = " lalu pengguna memasukkan nilai, misalnya 10 (baris 22). Setelah itu muncul pesan "Membuat node baru" (baris 23), dan fungsi create_new_node() dipanggil untuk membentuk node baru (baris 24). Karena berhasil, program memberikan konfirmasi "Berhasil membuat node baru" (baris 26). Node tersebut kemudian dimasukkan ke list dengan memanggil insert_node() (baris 32). Karena masih merupakan node pertama, pointer start dan rear otomatis menunjuk ke node tersebut. Program lalu mencetak pesan "Node dimasukkan ke list" (baris 33). Akhir siklus ditutup dengan pertanyaan apakah pengguna ingin menambah node lagi (baris 35). Jika pengguna menjawab y (baris 36), maka perulangan dilanjutkan.

Langkah serupa diulangi untuk nilai 20 dan 30. Namun kali ini, fungsi insert_node() bekerja dengan menautkan node baru ke akhir list. Pointer next dari node terakhir diarahkan ke node baru, dan pointer prev dari node baru diarahkan kembali ke node sebelumnya. Setelah itu rear diperbarui agar menunjuk ke node baru. Dengan mekanisme ini terbentuklah hubungan dua arah antar-node. Setelah tiga angka dimasukkan, pengguna menjawab n sehingga perulangan selesai.

Tahap berikutnya adalah traversal. Program menawarkan dua opsi: "1. Maju" dan "2. Mundur" (baris 40–41), lalu menampilkan prompt "Arah traversal yang mana?: " (baris 42). Misalnya, pengguna memilih 1 (baris 43), maka case 1 akan dijalankan. Program menuliskan "Traversal maju:" (baris 48) kemudian memanggil traverseForward() (baris 49). Fungsi ini menelusuri list dari node awal hingga akhir dengan mengikuti pointer next, dan mencetak seluruh data yang tersimpan. Karena output dicetak tanpa spasi (baris 79–80), hasil akhirnya menjadi 102030, yang menggambarkan urutan traversal maju.

5.3 Percobaan 3: Vector

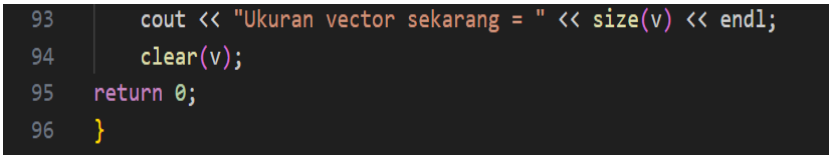
5.3.a Source Code Percobaan 5.3

vector.cpp > ...

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

struct Vector {
    int* data;
    int capacity;
    int length;
};

void init(Vector& v) {
    v.capacity = 2;
    v.length = 0;
    v.data = new int[v.capacity];
}

void resize(Vector& v, int newCap) {
    int* newData = new int[newCap];
    for (int i = 0; i < v.length; i++) {
        newData[i] = v.data[i];
    }
    delete[] v.data;
    v.data = newData;
    v.capacity = newCap;
}

void push_back(Vector& v, int value) {
    if (v.length == v.capacity) {
        resize(v, v.capacity * 2);
    }
    v.data[v.length] = value;
```

```
    v.length++;
}

void pop_back(Vector& v) { ···

int get(Vector& v, int index) {
    if (index >= 0 && index < v.length) {
        return v.data[index];
    }
    return -1; // Return a default value if index is invalid
}

void set(Vector& v, int index, int value) {
    if (index >= 0 && index < v.length) {
        v.data[index] = value;
    }
}

int size(Vector& v) {
    return v.length;
}

void display(Vector& v) {
    cout << "[";
    for (int i = 0; i < v.length; i++) {
        cout << v.data[i];
        if (i < v.length - 1) cout << ", ";
    }
    cout << "]\n";
```

```
}

void clear(Vector& v) {
    delete[] v.data;
    v.data = nullptr;
    v.capacity = 0;
    v.length = 0;
}

int main() {
    Vector v;
    init(v);
    push_back(v, 10);
    push_back(v, 20);
    push_back(v, 30);

    cout << "Isi vector: ";
    display(v);

    cout << "Elemen ke-1 = " << get(v, 1) << endl;

    set(v, 1, 99);
    cout << "Set elemen ke-1 jadi 99: ";
    display(v);

    pop_back(v);
    cout << "Setelah pop_back: ";
    display(v);

```



Gambar 5.3.a Source Code Percobaan 5.3

Berdasarkan gambar 5.3.a, terlihat bahwa program ini merupakan implementasi manual dari sebuah vector yang dapat menyesuaikan ukurannya secara dinamis. Program diawali pada baris 1 dan 2 dengan menyertakan pustaka standar iostream dan penggunaan namespace std untuk mempermudah pemanggilan fungsi input-output. Bagian inti program terletak pada deklarasi struct Vector di baris 4 sampai 8, yang berisi tiga komponen penting: int* data (baris 5) sebagai pointer ke lokasi array di memori, int capacity (baris 6) untuk menyimpan kapasitas maksimum array saat ini, dan int length (baris 7) untuk mencatat jumlah elemen yang terisi.

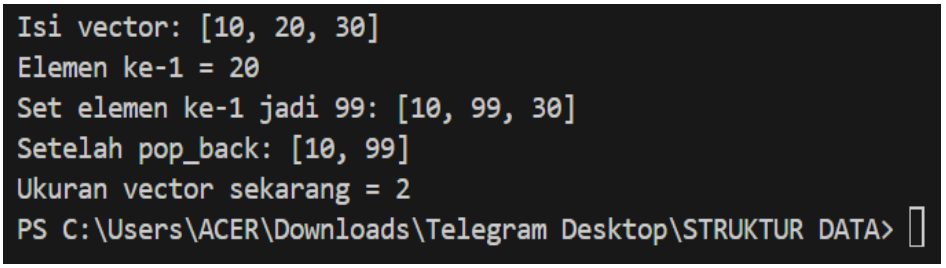
Fungsi-fungsi utama pengelola vector didefinisikan setelahnya. Fungsi init (baris 10–14) berfungsi menginisialisasi vector dengan kapasitas awal 2 (baris 11), panjang 0 (baris 12), dan melakukan alokasi memori baru (baris 13). Fungsi resize (baris 16–23) bertugas menambah kapasitas ketika array penuh. Pada baris 17, memori baru dialokasikan sesuai kapasitas baru, kemudian baris 18–20 menyalin elemen lama ke array baru. Setelah itu, baris 21 menghapus memori lama dengan delete[], baris 22 mengganti pointer data ke memori baru, dan baris 23 memperbarui kapasitas.

Dengan mekanisme resize ini, fungsi push_back (baris 27–32) dapat menambahkan elemen ke akhir. Pada baris 28–29 dilakukan pengecekan apakah array penuh; bila iya, kapasitas digandakan dengan memanggil resize. Selanjutnya, nilai baru disimpan ke array (baris 31) dan panjang ditambah satu (baris 32). Fungsi pop_back (baris 34–38) digunakan untuk menghapus elemen terakhir; pada baris 35–37 hanya dilakukan pengurangan length jika masih ada elemen yang tersisa.

Selain itu terdapat fungsi pendukung. Fungsi get (baris 40–45) mengembalikan nilai pada indeks tertentu bila valid (baris 41–43), atau -1 jika indeks tidak sah (baris 44). Fungsi set (baris 47–51) memungkinkan pengubahan nilai pada indeks tertentu dengan pemeriksaan batas array. Fungsi size (baris 53–55) mengembalikan jumlah elemen saat ini. Fungsi display (baris 57–63) mencetak seluruh elemen vector dalam format array, dengan perulangan pada baris 59–61 yang menampilkan setiap elemen dan memberi pemisah koma di antara elemen. Fungsi clear (baris 65–70) bertugas membersihkan memori: baris 66 menghapus array dengan delete[], lalu baris 67–69 mengatur pointer dan variabel capacity serta length kembali ke kondisi awal.

Bagian utama program ada pada fungsi main (baris 82–101). Pada baris 83 dibuat sebuah variabel Vector v yang diinisialisasi dengan init(v) pada baris 84. Tiga elemen kemudian ditambahkan menggunakan push_back pada baris 85–87, di mana saat menambahkan elemen ketiga (30), kapasitas diperbesar otomatis karena kapasitas awal hanya 2. Selanjutnya, baris 89 mencetak isi vector dengan display, baris 91 menampilkan elemen ke-1 dengan get, lalu baris 93–95 memperbarui elemen ke-1 menjadi 99 dengan set dan menampilkan hasilnya. Setelah itu, baris 97–99 menghapus elemen terakhir dengan pop_back lalu menampilkan sisa elemen. Program kemudian memanggil size pada baris 101 untuk menunjukkan ukuran Vector yang tersisa, dan akhirnya memanggil clear pada baris 102 untuk membersihkan memori sebelum program selesai.

5.2.b Output Percobaan 3



Gambar 5.3.b Output Percobaan 3

Berdasarkan kode pada gambar 5.3.b, program pada fungsi main() dimulai dengan deklarasi Vector v (baris 72) dan inisialisasi menggunakan init(v) pada baris 73. Fungsi ini mengatur kapasitas awal sebesar 2, panjang (length) menjadi 0, dan mengalokasikan array baru sesuai kapasitas. Setelah itu, perintah push_back(v, 10) dan push_back(v, 20) pada baris 74–75 menambahkan dua elemen pertama ke dalam vector. Ketika push_back(v, 30) pada baris 76 dijalankan, kondisi v.length == v.capacity pada baris 26 terpenuhi. Hal ini memicu fungsi resize() (baris 12–20) yang menggandakan kapasitas menjadi 4, menyalin elemen lama ke array baru, lalu menambahkan elemen 30. Hasilnya, perintah cout << "Isi vector: "

(baris 78) diikuti pemanggilan display(v) (baris 79) menampilkan output: Isi vector: [10, 20, 30].

Selanjutnya, baris 81 mencetak teks “Elemen ke-1 =”, yang kemudian dilanjutkan dengan nilai 20 hasil dari pemanggilan fungsi get(v, 1) (baris 37–42). Nilai tersebut diambil dari indeks ke-1 pada array. Pada baris 83, fungsi set(v, 1, 99) mengganti nilai elemen pada indeks ke-1. Perubahan ini ditampilkan melalui cout pada baris 84 dan display(v) pada baris 85, sehingga menghasilkan output: Set elemen ke-1 jadi 99: [10, 99, 30]. Setelah itu, pop_back(v) pada baris 87 mengurangi panjang vector dari 3 menjadi 2, sehingga elemen terakhir (30) tidak lagi ditampilkan. Oleh karena itu, perintah cout pada baris 88 diikuti display(v) pada baris 89 menampilkan hasil: Setelah pop_back: [10, 99].

Terakhir, baris 91 mencetak ukuran vector dengan memanggil fungsi size(v) (baris 47–49), yang mengembalikan nilai 2 sesuai jumlah elemen terkini, sehingga outputnya adalah Ukuran vector sekarang = 2. Program ditutup dengan pemanggilan clear(v) pada baris 92 yang menghapus array dari memori dan mengembalikan kapasitas serta panjang ke 0, meskipun tidak menghasilkan output tambahan di terminal.

V. KESIMPULAN

Adapun kesimpulan dari percobaan ini adalah sebagai berikut:

1. Berdasarkan percobaan 5.1.1 Array 1D, dapat disimpulkan bahwa array satu dimensi menyimpan elemen secara berurutan dalam memori, di mana setiap elemen menempati alamat dengan selisih tetap sesuai ukuran tipe data yang digunakan.
2. Berdasarkan percobaan 5.1.2 Array 2D, diperoleh bahwa array dua dimensi juga tersimpan secara kontigu dalam memori dengan pola row-major, yaitu elemen dalam satu baris ditempatkan terlebih dahulu sebelum baris berikutnya.
3. Berdasarkan percobaan 5.2.1 Single Linked List, diketahui bahwa single linked list merupakan struktur data dinamis yang setiap elemennya (node) tidak harus bersebelahan di memori, melainkan terhubung melalui pointer next, sehingga memudahkan proses penambahan maupun penghapusan data.
4. Berdasarkan percobaan 5.2.2 Double Linked List, dapat dipahami bahwa double linked list merupakan pengembangan dari single linked list dengan penambahan pointer prev pada setiap node, yang memungkinkan penelusuran data dilakukan dua arah, baik maju maupun mundur.
5. Berdasarkan percobaan 5.3 Vector, dapat disimpulkan bahwa vector adalah bentuk array dinamis yang mampu menyesuaikan kapasitasnya secara otomatis ketika penuh, dengan cara mengalokasikan blok memori baru yang lebih besar, menyalin elemen lama, dan tetap memberikan kemudahan akses data seperti array konvensional.

DAFTAR PUSTAKA

Baumann, P., Misev, D., Merticariu, O. & Pham Huu, T., 2021. *Array databases: Concepts, standards, implementations*. *Journal of Big Data*.

Goodrich, M.T., Tamassia, R. & Goldwasser, M.H., 2021. *Data Structures and Algorithms in C++*. 2nd ed. Hoboken, NJ: Wiley.

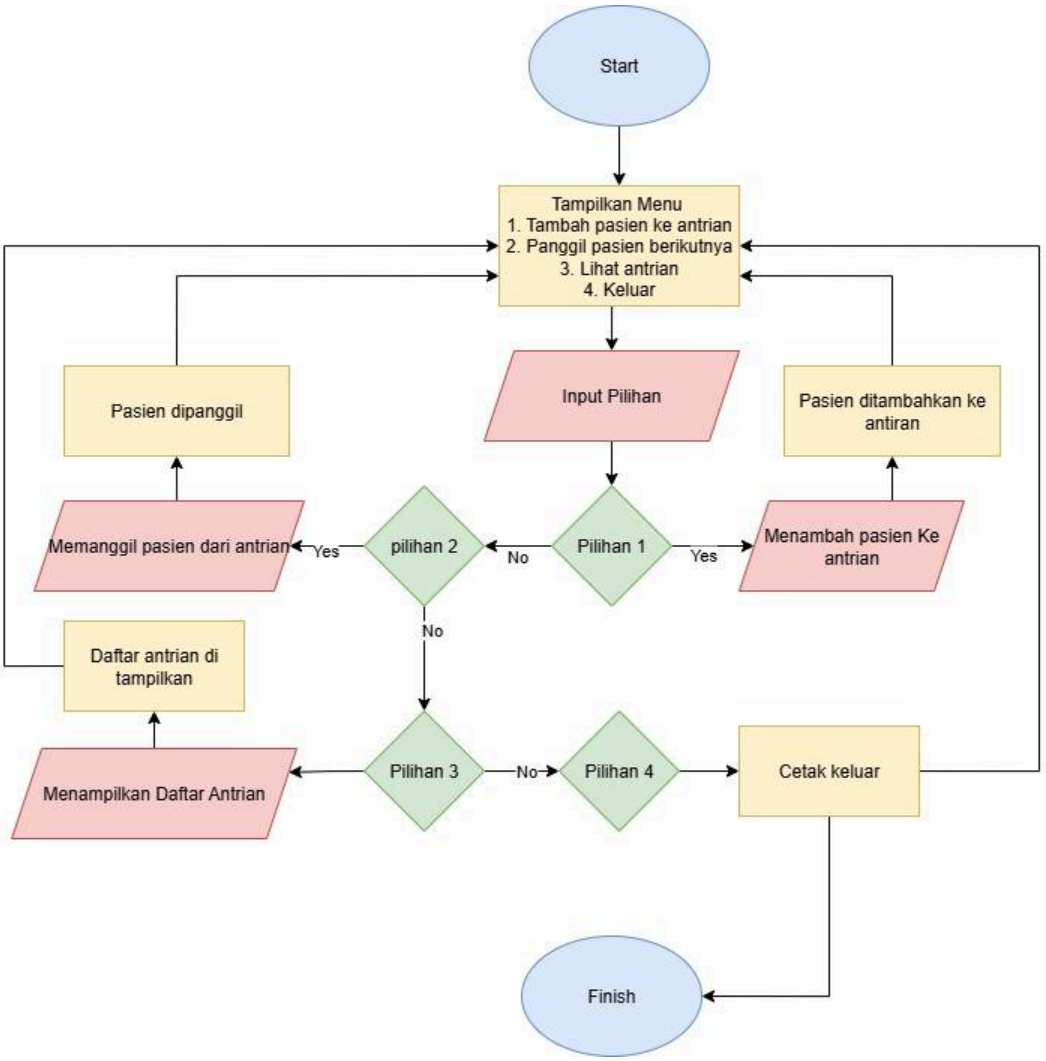
Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D. & Oliphant, T.E., 2020. *Array programming with NumPy*. *Nature*.

Sedgewick, R. & Wayne, K., 2019. *Algorithms*. 4th ed. Boston: Addison-Wesley.

Wazir, Z., 2020. *Implementation and performance analysis of linked list in C++*. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*.

TUGAS AKHIR

A. Flowchart



B. Implementasi Array 1 D

ImplementasiArray.cpp > ...

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

#define MAX 20   // kapasitas maksimal antrian

class Antrian {
private:
    int data[MAX];
    int front, rear;

public:
    Antrian() {
        front = -1;
        rear = -1;
    }

    bool isEmpty() {
        return (front == -1 && rear == -1);
    }

    bool isFull() {
        return (rear == MAX - 1);
    }

    void enqueue(int nomor) {
        if (isFull()) {
            cout << "⚠️  Antrian penuh! Pasien nomor " << nomor << " tidak bisa masuk.\n";
            return;
        }
        if (isEmpty()) {
```

```cpp
        if (isEmpty()) {
            front = rear = 0;
        } else {
            rear++;
        }
        data[rear] = nomor;
        cout << "✅ Pasien nomor " << nomor << " masuk antrian.\n";
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "⚠️  Antrian kosong! Tidak ada pasien yang dipanggil.\n";
            return;
        }
        cout << "🔔 Memanggil pasien nomor " << data[front] << endl;

        if (front == rear) {
            front = rear = -1;
        } else {
            front++;
        }
    }

    void display() {
        if (isEmpty()) {
            cout << "📭 Antrian kosong.\n";
            return;
        }
```

```
        cout << "📄 Antrian sekarang: ";
        for (int i = front; i <= rear; i++) {
            cout << data[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Antrian rs;
    int pilihan, nomor = 1;

    do {
        cout << "\n=== SISTEM ANTRIAN RUMAH SAKIT ===\n";
        cout << "1. Tambah pasien ke antrian\n";
        cout << "2. Panggil pasien berikutnya\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Keluar\n";
        cout << "Pilih menu (1-4): ";
        cin >> pilihan;

        switch (pilihan) {
        case 1:
            rs.enqueue(nomor);
            nomor++;  // nomor pasien bertambah otomatis
            break;
        case 2:
            rs.dequeue();
            break;
        case 3:
            rs.display();
            break;
        case 4:
            cout << "👋 Terima kasih, program selesai.\n";
            break;
        default:
            cout << "⚠️  Pilihan tidak valid!\n";
        }
    } while (pilihan != 4);

    return 0;
}
```

Gambar B Source Code Implementasi Array 1 D

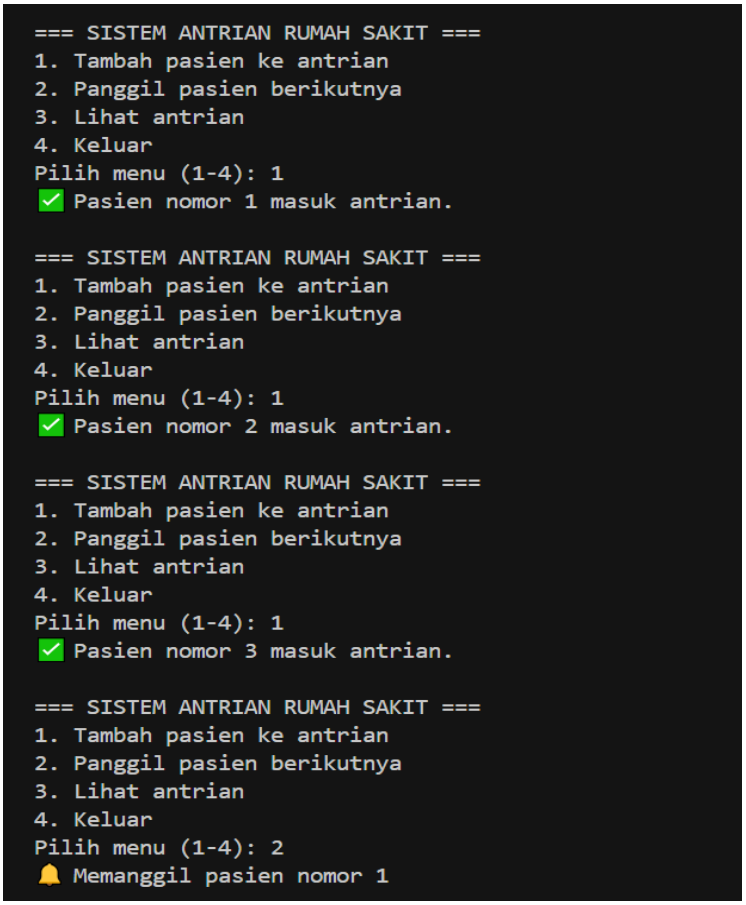
Berdasarkan Gambar B Source Code Implementasi Array 1D, program ini menunjukkan implementasi manual dari struktur data antrian (queue) menggunakan array statis dengan kapasitas maksimum 20 elemen. Pada baris 1 dan 2, program menambahkan pustaka standar iostream dan menggunakan namespace std untuk mempermudah pemanggilan fungsi input–output. Pada baris 4 didefinisikan konstanta MAX bernilai 20 sebagai batas kapasitas antrian. Struktur utama ditulis mulai baris 6 dengan mendefinisikan kelas Antrian. Di dalamnya terdapat tiga anggota utama, yaitu array data[MAX] (baris 8) untuk menyimpan nomor urut pasien, serta variabel front dan rear (baris 9) untuk menandai posisi depan dan belakang antrian.

Konstruktor Antrian() (baris 12–14) menginisialisasi front dan rear dengan nilai -1 sebagai tanda antrian kosong. Fungsi isEmpty (baris 16–18) memeriksa apakah antrian kosong dengan mengecek kondisi front == -1 && rear == -1. Fungsi isFull (baris 20–22) mengecek apakah antrian sudah penuh dengan membandingkan nilai rear terhadap MAX - 1. Fungsi enqueue (baris 24–37) digunakan untuk menambahkan pasien ke dalam antrian. Jika antrian penuh, pesan peringatan ditampilkan (baris 26–28). Jika antrian kosong, maka front dan rear diatur ke 0 (baris 30). Jika sudah ada pasien, maka rear ditambah satu (baris 32). Nomor pasien kemudian dimasukkan ke array pada indeks rear (baris 34), dan ditampilkan pesan bahwa pasien berhasil masuk antrian (baris 35).

Fungsi dequeue (baris 39–49) digunakan untuk memanggil pasien di posisi paling depan. Jika antrian kosong, pesan peringatan ditampilkan (baris 41–43). Jika tidak kosong, pasien pada indeks front dipanggil (baris 45). Jika pasien yang dipanggil adalah satu-satunya elemen, maka front dan rear dikembalikan ke -1 (baris 47). Jika masih ada lebih dari satu pasien, front digeser ke depan (baris 49). Fungsi display (baris 51–60) menampilkan seluruh isi antrian. Jika antrian kosong, pesan "Antrian kosong" ditampilkan (baris 53–55). Jika ada pasien, data ditampilkan mulai dari indeks front hingga rear dengan perulangan (baris 57–59).

Fungsi utama main (baris 62–101) memperlihatkan penggunaan kelas Antrian. Pada baris 63 dibuat objek Antrian rs, lalu baris 64 mendeklarasikan variabel pilihan dan nomor dengan nilai awal 1. Perulangan do–while pada baris 66–99 menampilkan menu utama program. Jika pengguna memilih menu 1 (baris 70–72), fungsi enqueue dipanggil untuk menambahkan pasien baru, dan variabel nomor bertambah otomatis. Jika memilih menu 2 (baris 74–75), fungsi dequeue memanggil pasien berikutnya. Jika memilih menu 3 (baris 77–78), fungsi display menampilkan daftar pasien dalam antrian. Jika memilih menu 4 (baris 80–81), program menampilkan pesan keluar. Untuk input tidak valid, pesan peringatan ditampilkan (baris 83). Program berhenti ketika pilihan bernilai 4 (baris 99).

C. Output Implementasi Array 1 D



Gambar C Output Implementasi Array 1 D

Berdasarkan Gambar C Output Implementasi Array 1 D, Pada saat pengguna memilih menu 1 (Tambah pasien ke antrian), program menjalankan fungsi enqueue() yang terdapat pada baris kode sekitar 39–55. Fungsi ini mengecek

kondisi antrian menggunakan isEmpty() dan isFull(), lalu menambahkan nomor pasien baru ke dalam array data[MAX]. Hal ini ditunjukkan oleh output “Pasien nomor 1 masuk antrian”, “Pasien nomor 2 masuk antrian”, dan seterusnya, di mana nomor pasien diatur oleh variabel nomor pada baris 91 di fungsi main. Selanjutnya, ketika pengguna memilih menu 2 (Panggil pasien berikutnya), program memanggil fungsi dequeue() yang berada pada baris 57–73. Fungsi ini menghapus pasien dari indeks front dan menampilkan pesan “Memanggil pasien nomor 1”.

Karena antrian bersifat FIFO, pasien yang pertama kali masuk dipanggil terlebih dahulu. Saat menu 3 (Lihat antrian) dipilih, fungsi display() pada baris 75–87 dijalankan untuk menampilkan isi antrian mulai dari indeks front hingga rear, sehingga muncul output “Antrian sekarang: 2 3”. Terakhir, menu 4 (Keluar) menghentikan program dengan keluar dari perulangan do-while pada baris 97–108. Dengan demikian, setiap output yang dihasilkan sesuai dengan logika fungsi pada baris kode masing-masing, mulai dari penambahan pasien, pemanggilan pasien, penampilan daftar antrian, hingga keluar dari sistem.