List linier (linked list) Dan variasinya_Struktur Data

Agung Kurniman Putra

185100036

Fakultas Komputer

agungkurnimanputra.student@umitra.ac.id

Abstract

Linked list (senarai bertautan; daftar bertautan) adalah salah satu cara untuk menyimpan sekumpulan elemen. Sama halnya dengan array, elemen yang disimpan dapat berupa karakter atau integer. Masing-masing elemen dalam linked list disimpan dalam bentuk sebuah **node**. Node adalah gabungan dua sub-elemen atau komponen; Bagian **data** yang menyimpan elemen dan bagian **next** yang menyimpan link ke node selanjutnya. Linked list terbentuk saat terdapat banyak node yang saling tertaut dan membentuk sebuah rantai. Setiap node menunjuk pada node selanjutnya sesuai urutan. Node pertama yang selalu digunakan sebagai referensi saat melakukan traversing pada list (senarai; daftar) disebut **HEAD**. Node terakhir akan menunjuk ke **NULL**. **Mendeklarasikan Linked list:** Dalam bahasa pemrograman C, linked list dapat diimplementasikan menggunakan struktur dan pointer. Definisi di atas digunakan untuk membuat node pada list. Bidang **data** menyimpan elemen dan **next** menjadi pointer yang menyimpan alamat node selanjutnya.

Apakah Anda memerhatikan ada sesuatu yang berbeda dengan *next*?

Dalam *data type*, **struct LinkedList** ditulis sebelum *next*. Hal tersebut karena *next* merupakan **self-referencing pointer**. Artinya, *next* adalah pointer yang menunjuk ke bagian di mana dirinya berada. Di sini, **next** merupakan bagian dari sebuah node dan dia akan menunjuk ke node selanjutnya.

Kata Kunci : List linier (linked list) Dan variasinya_Struktur Data

A. PENDAHULUAN

LIST LINIER 1. DEFINISI

List linier adalah sekumpulan elemen bertipe sama, yang mempunyai keterurutan tertentu, dan setiap elemennya terdiri dari dua bagian, yaitu informasi mengenai elemennya, dan informasi mengenai alamat elemen suksesornya : type ElmtList : < Info : InfoType, Next :address> dengan InfoType adalah sebuah type terdefinisi yang menyimpan informasi sebuah elemen list; Next adalah address ("alamat") dari elemen berikutnya (suksesor). Dengan demikian, jika didefinisikan adalah alamat elemen pertama list, maka elemen berikutnya dapat diakses secara suksesif dari field Mext elemen tersebut Alamat yang sudah didefinisikan disebut sudah dialokasi. Didefinisikan suatu konstanta Nil, yang artinya alamat yang tidak terdefinisi. Alamat ini nantinya akan didefinisikan secara lebih konkret ketika list linier diimplementasi pada struktur data fisik ElmtList Nil Info Next Jadi, sebuah list linier dikenali : elemen pertamanya, biasanya melalui alamat elemen pertama yang disebut : alamat elemen berikutnya (suksesor), jika kita mengetahui alamat sebuah elemen, yang dapat diakses melalui informasi NEXT. NEXT mungkin ada secara

eksplisit (seperti contoh di atas), atau secara implisit yaitu lewat kalkulasi atau fungsi suksesor. setiap elemen mempunyai alamat, yaitu tempat elemen disimpan dapat diacu. Untuk mengacu sebuah elemen, alamat harus terdefinisi. Dengan alamat tersebut Informasi yang tersimpan pada elemen list dapat diakses. elemen terakhirnya. Ada berbagai cara untuk mengenali elemen akhir Jika L adalah list, dan adalah address: Alamat elemen pertama list L dapat diacu dengan notasi : (L) Elemen yang diacu oleh dapat dikonsultasi informasinya dengan notasi Selektor : Info() Next() Beberapa definisi : - List L adalah list kosong, jika (L) = Nil - Elemen terakhir dikenali, misalnya jika Last adalah alamat element terakhir, maka Next(Last) =Nil Inggriani Liem : "Algoritma dan emrograman rosedural Bagian II.

B. PEMBAHASAN / STUDI KASUS

Struktur Data Linear ialah data yang dipresentasikan berhubungan dengan data lainnya dengan teratur (lurus) sehingga membentuk barisan antara data satu dengan data lainnya.

Linked List ialah suatu record data yang dihubungkan satu dengan lainnya menggunakan pointer.

Fungsi-fungsi yang dapat dipakai dalam operasi linked list adalah:

1. Fungsi menambah simpul di belakang. Langkah-langkahnya:
 - a. membuat simpul baru kemudian diisi info baru.
 - b. simpul paling akhir dihubungkan ke simpul baru.
 - c. penunjuk pointer akhir diarahkan ke simpul simpul baru.
2. Fungsi menambah simpul di depan. Langkah-langkahnya:
 - a. membuat simpul baru, kemudian diisi info baru.
 - b. simpul baru dihubungkan ke simpul awal.
 - c. penunjuk pointer awal diarahkan ke simpul baru.
3. Fungsi menyisipkan simpul di tengah. Langkah-langkahnya:
 - a. membuat simpul bantu, kemudian diisi info baru.
 - b. menentukan di mana simpul baru akan ditambahkan, yaitu dengan menempatkan pointer bantu pada suatu tempat.
 - c. pointer baru dihubungkan ke simpul setelah simpul yang ditunjuk oleh pointer bantu, kemudian penunjuk pointer bantu diarahkan ke simpul baru.
4. Fungsi menghapus simpul di depan. Langkah-langkahnya:
 - a. simpul bantu diarahkan pada simpul awal.
 - b. simpul awal diarahkan ke simpul berikutnya.
 - c. menghapus simpul bantu.
5. Fungsi menghapus simpul di tengah. Langkah-langkahnya:
 - a. meletakkan pointer bantu di sebelah kiri simpul yang akan dihapus.
 - b. simpul yang akan dihapus ditunjuk oleh pointer lain, misalnya pointer hapus.
 - c. pointer bantu diarahkan ke simpul yang ditunjuk oleh hapus, kemudian simpul hapus dihapus.
6. Fungsi menghapus simpul di belakang. Langkah-langkahnya:
 - a. meletakkan pointer bantu di sebelah kiri simpul yang akan dihapus.
 - b. simpul yang akan dihapus ditunjuk oleh pointer lain, misalnya pointer hapus.
 - c. pointer bantu, diarahkan ke simpul yang ditunjuk oleh hapus, kemudian simpul hapus dihapus.
7. Fungsi mencetak list dengan membaca maju. Langkah-langkahnya:
 - a. mengatur agar pointer bantu menunjuk simpul yang ditunjuk oleh pointer awal.
 - b. setelah isi simpul dibaca dan dicetak, pointer bantu digerakkan ke kanan untuk membaca simpul berikutnya.
 - c. proses ini diulang sampai pointer bantu sama dengan pointer akhir.
8. Fungsi mencetak list dengan membaca mundur. Langkah-langkahnya:
 - a. mengatur agar pointer bantu menunjuk simpul yang ditunjuk oleh pointer akhir.
 - b. setelah pointer bantu menunjuk simpul akhir, baru dicetak.

- c. proses pencetakan dilanjutkan dengan mencetak isi simpul di sebelah kiri simpul akhir.
 - d. proses selesai jika isi simpul paling awal telah selesai dicetak.
9. Fungsi mencari data pada list. Langkah-langkahnya:
- a. data yang dicari disimpan dalam suatu variabel, misalkan elemen.
 - b. membuat pointer bantu sama dengan pointer awal.
 - c. isi simpul yang ditunjuk oleh pointer bantu dibandingkan dengan elemen. Jika sama, berarti data yang dicari ditemukan. Jika belum sama, maka pointer bantu dipindahkan ke simpul di sebelah kanannya dan proses perbandingan diulang. Proses ini diulang sampai data yang dicari ditemukan atau sampai pada simpul akhir

1. Single Linked List Non Circular

Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya & ke node sebelumnya. Untuk pembentukan node baru, mulanya pointer next dan prev akan menunjuk ke nilai NULL. Selanjutnya pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node selanjutnya pada list.

Deklasari simpul :

```
Struct simpul{
    int angka;
    simpul *berikut;
};
```

a. Tambah Depan

Prinsipnya ketika data baru datang atau akan ditambahkan maka **head** atau awal akan menunjuk pada data baru tersebut dan data baru tersebut menunjuk pada data yang lama. Sehingga data lama akan menjadi di bawah data baru.

```
void tambahdepan(int x)
{
    struct titik *baru;
    baru=new titik;
    baru->angka=x;
    baru->berikut=awal;
    awal=baru;
}
```

b. Tambah Belakang

Pada prinsipnya sama dengan antrian, siapa yang masuk dialah yang akan keluar terlebih dulu. Maka data lama akan tetap menjadi **awal** bagi data baru, sehingga data pertama kali masuk akan terus menjadi **awal** dan data baru akan terus menjadi setelah data sebelumnya.

```
void tambahbelakang(int x) {
    struct simpul *baru;
    baru=new simpul;
    baru->angka=x;
    if(awal==NULL){
        awal=baru;}
    else{
        akhir->berikut=baru; }
```

```
akhir=baru;
akhir->berikut=NULL;
}
```

c. Tambah List di tengah

Tambah list ditengah ini berguna ketika kita telah menginputkan data ternyata terdapat data yang terlewat untuk diinputkan dan harus berada diantara data yang telah di inputkan. Untuk memudahkan pemahamannya saya akan memberikan langsung dengan pengisian datanya. Yang pertama terdapat fungsi *sisip_isi()* ini berfungsi untuk menginputkan data dan fungsi *sisip_list(simpul *first, int x, int posisi)* untuk menentukan data akan disimpan setelah data yang mana sehingga dapat diisi diantara data-data yang lainnya.

```
void sisip_list(struct simpul *first,int
x,int posisi)
{
    struct simpul *bantu,*baru;
    baru = new simpul;
    baru->angka=x;
    bantu=first;
    do
    {
        if (posisi!=bantu->angka)
        {bantu=bantu->berikut;}
    }
    while (bantu!=NULL &&
posisi!=bantu->angka);
    baru->berikut=bantu->berikut;
    bantu->berikut=baru;
}
```

```
void sisip_isi_list()
{
    int cari;
    int ganti;
    cout<<"\nBilangan : "; cin>>ganti;
    cout<<"\nDisipkan setelah : ";
    cin>>cari;
```

```
sisip_list(awal,ganti,cari);
}
```

d. Hapus Tengah

Tidak jauh beda dengan isi list di tengah, kita membutuhkan 2 fungsi yang pertama untuk mengiput data yang akan dihapus dan fungsi yang akan menghapus list. Fungsi *carihapus()* akan menjadi fungsi untuk membawa parameter atau nilai yang akan dihapus sementara fungsi *hapustengah(int info)* akan menghapus data yang sesuai dengan isi dari *info* yang dibawa oleh fungsi *carihapus()*.

```
void hapustengah(int info)
{
    titik *bantu,*hapus;
    bantu=awal;
    while ((info!=bantu->berikut->angka)
    && (bantu->berikut!=NULL))
    {
        bantu=bantu->berikut;
    }
    hapus=bantu->berikut;
    bantu->berikut=hapus->berikut;
    delete hapus;
}
```

```
void carihapus()
{
    int cari;
    clrscr();
    tampillist();
    cout<<"\nData yang akan dihapus,
Bilangan : ";
    cin>>cari;
    hapustengah(cari);
    cout<<"\nData menjadi : ";
    tampillist();
}
```

e. Hapus Depan

Hapus di depan sebenarnya sangat simple karena kita bisa langsung menghapus **awal*, namun disela itu sebelum tugas kita menghapus **awal* kita harus mengganti **awal* dengan pointer berikutnya sehingga ketika ditampilkan, data masih akan terlihat.

```
Void hapusdepan()
{
    Simpul *hapus;
    Hapus=awal;
    Awal=hapus->berikut;
    Delete hapus;
}
```

f. Hapus Belakang

Tidak jauh beda dengan hapus depan, namu hapus belakang membutuhkan looping untuk mengetahui bahwa dia adalah simpul paling terakhir.

```
Void hapusbelakang()
{
    Simpul *hapus, *bantu;
    Bantu=awal;
    While(bantu->berikut-
    >berikut!=NULL){
        bantu=bantu-
        >berikut;
    }
    hapus=bantu->berikut;
    bantu->berikut=NULL;
    delete hapus;
}
```

g. Hapus Semua

Hapus semua merupakan cara penghapusan paling sederhana, kita hanya menentukan kapan **hapus* berisi null dan disitulah dia berhenti.

```
void hapus()
{
    simpul *hapus;
    hapus=awal;
    while(hapus!=NULL){
        awal=hapus->berikut;
        delete hapus;
        hapus=awal;
    }
}
```

2. Single Linked list Circular

Single Linked List yang pointer nextnya menunjuk pada dirinya sendiri. Jika Single Linked List tersebut terdiri dari beberapa node, maka pointer next pada node terakhir akan menunjuk ke node terdepannya.

Deklarasi simpul :

```
struct simpul{
    int angka;
    simpul *berikut;
};
```

a. Tambah Depan

Pada prinsipnya adalah mengkaitkan data baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan. Untuk menghubungkan node terakhir dengan node terdepan dibutuhkan pointer bantu.

```
void tambahdepan(int x)
{
    simpul *baru, *bantu;
    baru = new simpul;
    baru->angka=x;
    baru->berikut=baru;
    if(awal==NULL){
        awal=baru;
        awal->berikut=awal;
    }
    else{
        bantu=awal;
        while(bantu->berikut!=awal){
            bantu=bantu->berikut;
        }
        baru->berikut=awal;
        awal=baru;
        bantu->berikut=awal;
    }
}
```

b. Tambah Belakang

Penambahan di belakang lebih sulit karena kita membutuhkan pointer bantu untuk mengetahui data terbelakang, kemudian dikaitkan dengan data baru. Untuk mengetahui data terbelakang perlu digunakan perulangan.

```
void tambahbelakang(int x)
{
    simpul *baru, *bantu;
    baru = new simpul;
    baru->angka=x;
    baru->berikut=baru;
    if(awal==NULL){
        awal=baru;
        awal->berikut=awal;
    }
    else{
        bantu=awal;
        while(bantu->berikut!=awal){
            bantu=bantu->berikut;
        }
        bantu->berikut=baru;
        awal=baru;
    }
}
```

```
baru->berikut=awal;
}
}
```

1. 3. Double Linked List Circular

Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya & ke node sebelumnya. Untuk pembentukan node baru, mulanya pointer next dan prev akan menunjuk ke dirinya sendiri. Jika sudah lebih dari satu node, maka pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node sesudahnya.

Deklarasi simpul :

```
Struct simpul{
    int angka;
    simpul *next;
    simpul *prev;
};
```

a. a. Tambahdepan

Pada prinsipnya adalah mengkaitkan data baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan. Dibutuhkan pointer bantu yang digunakan untuk menunjuk node terakhir (head->prev) yang akan digunakan untuk mengikat list dengan node terdepan.

```
void tambahdepan(int x){
```



```

simpul *baru, *bantu;
baru = new simpul;
baru->angka = x;
baru->next = baru;
baru->prev = baru;
if(awal==NULL){
    awal=baru;
    awal->next =awal;
    awal->prev = awal;
}
else {
    bantu = awal->prev;
    baru->next = awal;
    awal->prev = baru;
    awal = baru;
    awal->prev = bantu;
    bantu->next = awal;
}
}

```

b **b. Tambah Belakang**

Penambahan di belakang lebih sulit karena kita membutuhkan pointer bantu untuk mengetahui data terbelakang, namun tidak diperlukan loop karena untuk mengetahui node terbelakang hanya perlu menunjuk pada head->prev saja. Kemudian dikaitkan dengan data baru. Untuk mengetahui data terbelakang perlu digunakan perulangan.

```

void tambahbelakang (int x){
    simpul *baru, *bantu;
    baru = new simpul;
    baru->data = x;
    baru->next = baru;
    baru->prev = baru;
    if(awal==NULL){
        awal=baru;
        awal->next = awal;
        awal->prev = awal;
    }
    else {
        bantu=awal->prev;

```

```

    bantu->next = baru;
    baru->prev = bantu;
    baru->next = awal;
    awal->prev = baru;
}
}

```

1. **4. Double Linked List Non-Circular**

Double Linked List Non Circular adalah linked list dengan menggunakan pointer, dimana setiap node memiliki 3 field, yaitu 1 field pointer yang menunjuk pointer berikutnya (next), 1 field menunjuk pointer sebelumnya (prev), serta sebuah field yang berisi data untuk node tersebut.

Deklarasi simpul :

```

Struct simpul{
    Int angka;
    simpul *next;
    simpul *prev;
};

```

a. **Tambah depan**

Pada prinsipnya adalah mengkaitkan data baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan.

```

void tambahdepan(int x)
{
    titik *baru;
    baru=new simpul;
    baru->nilai=x;
    baru->next=NULL;

```



```
baru->prev=NULL;
if(awal==NULL){
    awal=baru;
    awal->next=NULL;
    awal->prev=NULL;
}
else{
    baru->next=awal;
    awal->prev=baru;
    awal=baru;
}
}
```

b. Tambah belakang

Penambahan di belakang lebih sulit karena kita membutuhkan pointer bantu untuk mengetahui data terbelakang, kemudian dikaitkan dengan data baru. Untuk mengetahui data terbelakang perlu digunakan perulangan.

```
void tambahbelakang(int x)
{
    titik *baru, *bantu;
    baru=new simpul;
    baru->nilai=x;
    baru->next=NULL;
    baru->prev=NULL;
    if(awal==NULL){
        awal=baru;
        awal->next=NULL;
        awal->prev=NULL;
    }
    else{
        bantu=head;
        while(bantu->next!=NULL){
            bantu=bantu->next;
        }
        bantu->next=baru;
        baru->prev=bantu;
    }
}
```

C. ID SECURITY

QWTD4452377-ASP-5244107

D. KESIMPULAN

Struktur Data List Linear:
Struktur data adalah cara menyimpan atau merepresentasikan data didalam komputer agar bisa dipakai secara efisien. Sedangkan data adalah representasi dari fakta dunia nyata. Fakta atau keterangan tentang kenyataan yang disimpan, direkam atau direpresentasikan dalam bentuk tulisan, suara, gambar, sinyal atau simbol.

List linier adalah sekumpulan elemen bertipe sama yang mempunyai keturunan tertentu, dan setiap elemen nya terdiri dari dua bagian, yaitu informasi mengenai elemen nya dan informasi mengenai alamat elemen suksesornya. Penggunaan List dalam Kehidupan Sehari - hari Misalnya:

- Urutan angka pada keyboard komputer
- Urutan lagu pada playlist Mp3 player
- Dll

Linked List (LL) Adalah koleksi data item yang tersusun dalam sebuah barisan secara linear, dengan penyisipan dan pemindahan dapat dilakukan dalam semua tempat di LL tersebut. **Single Linked List** Adalah sebuah LL yang menggunakan sebuah variabel

pointer saja untuk menyimpan banyak data dengan metode LL, suatu daftar isi yang saling berhubungan. **Double Linked List**, Dalam double LL (Linked List berpointer ganda) dapat mengatasi kelemahan-kelemahan single LL tersebut. **Circular Linked List** Adalah double / single LL yang simpul terakhirnya menunjuk ke simpul awal, dan simpul awalnya menunjuk ke simpul akhir, atau dapat disebut LL yang dibuat seakan-akan merupakan sebuah lingkaran dengan titik awal dan titik akhir saling bersebelahan jika LL tersebut masih kosong. List linier memiliki 2 jenis, yaitu:

E. Stack (Tumpukan): Struktur data linear dimana penambahan atau pengurangan komponen dilakukan di satu ujung saja dan merupakan suatu bentuk khusus dari linear list di mana operasi penyisipan dan penghapusan atas elemen-elemennya hanya dapat dilakukan pada satu sisi saja.

F. DISKUSI

Apakah artikel ini berguna?

Fajar: menurut saya artikel ini sangat berguna karena saya dapat mempelajari lebih dalam tentang **List linier (linked list) Dan variasinya.**

Apakah kekurangan artikel ini?

Komang : menurut saya kekurangan artikel ini adalah kurang penjelasan melalui gambar, di artikel ini hanya terdapat 1 penjelasan gambar.

G. REFERENCE

- [1] O. M. Febriani and A. S. Putra, "Sistem Informasi Monitoring Inventori Barang Pada Balai Riset Standardisasi Industri Bandar Lampung," *J. Inform.*, vol. 13, no. 1, pp. 90–98, 2014.
- [2] A. S. Putra, "Paperplain: Execution Fundamental Create Application With Borland Delphi 7.0 University Of Mitra Indonesia," 2018.
- [3] A. S. Putra, "2018 Artikel Struktur Data, Audit Dan Jaringan Komputer," 2018.
- [4] A. S. Putra, "ALIAS MANAGER USED IN

- DATABASE DESKTOP
STUDI CASE DB DEMOS.”
- [5] A. S. Putra, “COMPREHENSIVE SET OF PROFESSIONAL FOR DISTRIBUTE COMPUTING.”
- [6] A. S. Putra, “DATA ORIENTED RECOGNITION IN BORLAND DELPHI 7.0.”
- [7] A. S. Putra, “EMBARCADERO DELPHI XE 2 IN GPU-POWERED FIREMONKEY APPLICATION.”
- [8] A. S. Putra, “HAK ATAS KEKAYAAN INTELEKTUAL DALAM DUNIA TEKNOLOGY BERBASIS REVOLUSI INDUSTRI 4.0.”
- [9] A. S. Putra, “IMPLEMENTASI PERATURAN PERUNDANGAN UU. NO 31 TAHUN 2000 TENTANG DESAIN INDUSTRI BERBASIS INFORMATION TECHNOLOGY.”
- [10] A. S. Putra, “IMPLEMENTATION OF PARADOX DBASE.”
- [11] A. S. Putra, “IMPLEMENTATION OF TRADE SECRET CASE STUDY SAMSUNG MOBILE PHONE.”
- [12] A. S. Putra, “IMPLEMENTATION PATENT FOR APPLICATION WEB BASED CASE STUDI WWW. PUBLIKLAMPUNG.COM.”
- [13] A. S. Putra, “IMPLEMENTATION SYSTEM FIRST TO INVENT IN DIGITALLY INDUSTRY.”
- [14] A. S. Putra, “MANUAL REPORT & INTEGRATED DEVELOPMENT ENVIRONMENT BORLAND DELPHI 7.0.”
- [15] A. S. Putra, “PATENT AS RELEVAN SUPPORT RESEARCH.”
- [16] A. S. Putra, “PATENT FOR RESEARCH STUDY CASE OF APPLE. Inc.”
- [17] A. S. Putra, “PATENT PROTECTION FOR APPLICATION INVENT.”
- [18] A. S. Putra, “QUICK REPORT IN PROPERTY PROGRAMMING.”
- [19] A. S. Putra, “REVIEW CIRCUIT LAYOUT COMPONENT REQUIREMENT ON ASUS NOTEBOOK.”
- [20] A. S. Putra, “REVIEW TRADEMARK PATENT FOR INDUSTRIAL TECHNOLOGY BASED 4.0.”
- [21] A. S. Putra, “TOOLBAR COMPONENT PALLETTE IN OBJECT ORIENTED PROGRAMMING.”
- [22] A. S. Putra, “WORKING DIRECTORY SET FOR PARADOX 7.”
- [23] A. S. Putra, “ZQUERY CONNECTION IMPLEMENTED PROGRAMMING STUDI CASE PT. BANK BCA Tbk.”
- [24] A. S. Putra, D. R. Aryanti, and I. Hartati, “Metode SAW (Simple Additive Weighting) sebagai Sistem Pendukung Keputusan Guru Berprestasi (Studi Kasus: SMK Global Surya),” in *Prosiding Seminar Nasional Darmajaya*, 2018, vol. 1, no. 1, pp. 85–97.

- [25] A. S. Putra and O. M. Febriani, "Knowledge Management Online Application in PDAM Lampung Province," in *Prosiding International conference on Information Technology and Business (ICITB)*, 2018, pp. 181–187.
- [26] A. S. Putra, O. M. Febriani, and B. Bachry, "Implementasi Genetic Fuzzy System Untuk Mengidentifikasi Hasil Curian Kendaraan Bermotor Di Polda Lampung," *SIMADA (Jurnal Sist. Inf. dan Manaj. Basis Data)*, vol. 1, no. 1, pp. 21–30, 2018.
- [27] A. S. Putra, H. Sukri, and K. Zuhri, "Sistem Monitoring Realtime Jaringan Irigasi Desa (JIDES) Dengan Konsep Jaringan Sensor Nirkabel," *IJEIS (Indonesian J. Electron. Instrum. Syst.)*, vol. 8, no. 2, pp. 221–232.
- [28] D. P. Sari, O. M. Febriani, and A. S. Putra, "Perancangan Sistem Informasi SDM Berprestasi pada SD Global Surya," in *Prosiding Seminar Nasional Darmajaya*, 2018, vol. 1, no. 1, pp. 289–294.