

LAPORAN PRAKTIKUM

ALGORITMA PEMROGRAMAN

“PRE,PRA,POST 10: POINTER”

Diajukan untuk memenuhi salah satu praktikum Mata Kuliah Algoritma Pemrograman yang
di ampu oleh:

Dr. Ardiansyah S.T., M.Cs



Disusun Oleh:

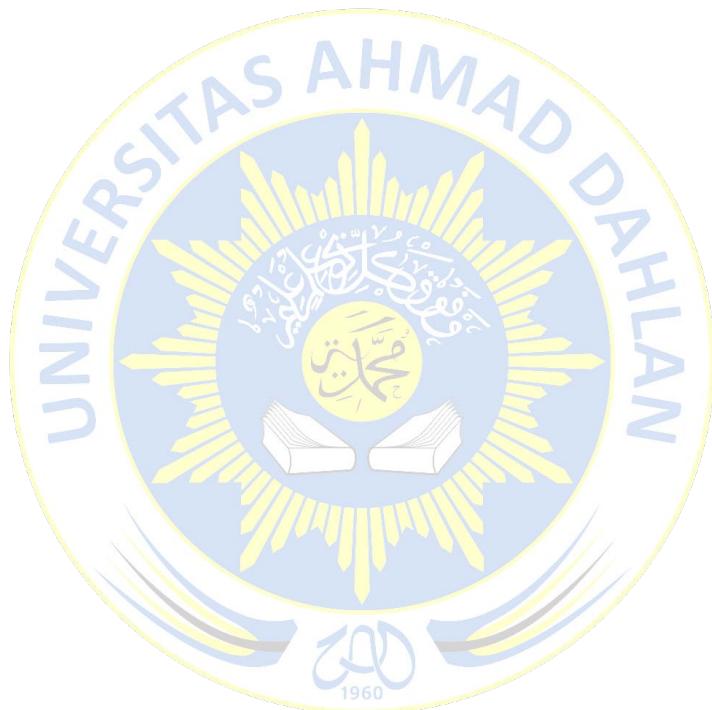
Mohammad Farid Hendianto 2200018401

A / Jumat 13.30 Lab. Jaringan

PROGRAM STUDI INFORMATIKA
UNIVERSITAS AHMAD DAHLAN
FAKULTAS TEKNOLOGI INDUSTRI
TAHUN 2023

DAFTAR BAGIAN

BAGIAN I PRE TEST	3
BAGIAN II LANGKAH_PRAKTIKUM.....	5
BAGIAN III POST TEST	27



BAGIAN I PRE TEST

LEMBAR JAWABAN PRE-TEST DAN POST-TEST PRAKTIKUM

Nama: Mohammad Farid Hendianto NIM: 2200018401	Asisten: Paraf Asisten:	Tanggal: 24/06/2023 Nilai:
---	----------------------------	-------------------------------

- Berikut adalah algoritma untuk menghitung banyaknya elemen dalam linked list!
- 1) Mulai program.
 - 2) Definisikan struktur Node yang berisi data integer dan pointer ke Node berikutnya.
 - 3) Buat fungsi countNodes yang menerima pointer ke Node pertama (head) dari linked list
 - a. Inisialisasi variabel count dengan 0. Ini akan digunakan untuk menghitung jumlah node.
 - b. Buat Pointer current dan atur ke head.
 - c. Selama current tidak NULL, lakukan langkah berikut:
 - Tambahkan 1 ke count.
 - Pindahkan current ke node berikutnya ($current = current \rightarrow next$)
 - d. Setelah loop selesai, kembalikan count sebagai hasil fungsi.
 - 4) Dalam fungsi main, minta pengguna untuk memasukkan jumlah node yang diinginkan dan simpan dalam variabel n.
 - 5) Buat linked list dengan n node.
 - a. Buat Node baru dan atur sebagai head.
 - b. Buat Pointer current dan atur ke head.
 - c. Untuk setiap nilai i dari 2 hingga $n-1$, lakukan langkah berikut.
 - Atur data dari current ke ii.
 - Buat node baru dan atur sebagai next dari current.
 - Pindahkan current ke node berikutnya.
 - Atur data dari current ke n dan next ke NULL
 - 6) Panggil fungsi countNodes dengan head sebagai argumen dan simpan hasilnya dalam variabel count.
 - 7) Cetak count ke layar.
 - 8) Akhiri program.

Berikut adalah algoritma berbentuk pseudocode.

```

DECLARE
    STRUCT Node {
        INTEGER data
        Node POINTER next
    }
    Node POINTER head, current
    INTEGER n, count
ALGORITHM
    FUNCTION countNodes(head: POINTER TO Node) RETURNS INTEGER
        count ← 0
        current ← head
        WHILE current IS NOT NULL DO
            count ← count + 1
            current ← current→next
        END WHILE
        RETURN count
    END FUNCTION
    OUTPUT "Enter the number of nodes:"
    INPUT n
    head ← NEW Node
    current ← head
    FOR i FROM 1 TO n-1 DO
        current→data ← i
        current→next ← NEW Node
        current ← current→next
    END FOR
    current→data ← n
    current→next ← NULL
    count ← CALL countNodes(head)
    OUTPUT "The number of nodes in the linked list is: ", count

```

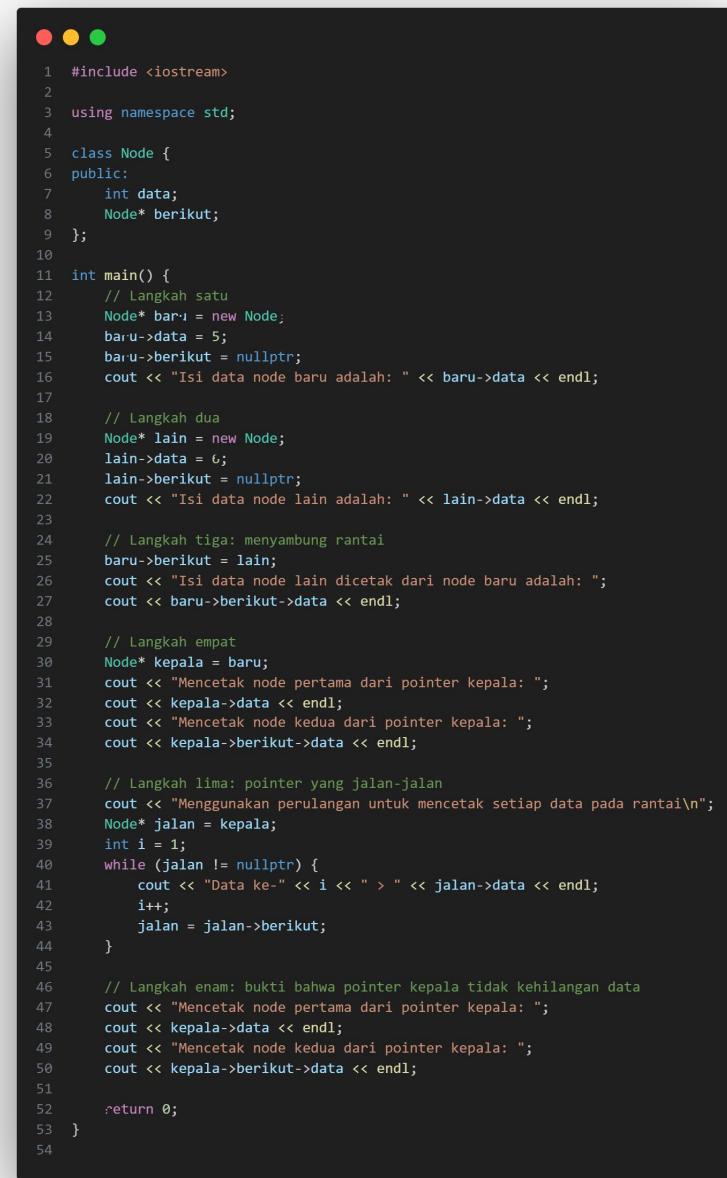
Berikut adalah kegunaan algoritma dari menghitung banyaknya elemen dalam linked list.

- Mengetahui ukuran linked list, algoritma ini memungkinkan kita untuk mengetahui banyak elemen yang ada di linked list. Informasi ini bisa sangat berguna dalam berbagai situasi seperti ketika kita perlu mengetahui seberapa besar data yang kita miliki.
- Pembuktian Flow Program, dalam beberapa kasus, kita mungkin perlu mengetahui jumlah elemen dalam linked list untuk mendukung alur program. Misalkan, jika kita memiliki batas atas atau bawah pada jumlah elemen yang dapat ditambahkan ke linked list, kita dapat menggunakan algoritma ini untuk menemukan apakah kita telah mencapai batas tersebut.
- Pengurutan dan Pencarian, dalam beberapa pengurutan dan pencarian kita perlu mengetahui jumlah elemen dalam linked list. Misalkan, dalam algoritma pengurutan seperti merge sort, kita perlu membagi linked list menjadi dua bagian yang hampir sama. Dalam hal ini, mengetahui jumlah elemen dalam linked list sangat penting.
- Optimalisasi Memori: Dengan mengetahui jumlah elemen dalam linked list, kita dapat memperbaiki penggunaan memori. Misalkan, jika kita tahu bahwa linked list kita memiliki sejumlah besar elemen, kita mungkin memilih untuk menggunakan struktur data yang lebih efisien dalam hal memori.

BAGIAN II

LANGKAH PRAKTIKUM

Buku modul



```

1 #include <iostream>
2
3 using namespace std;
4
5 class Node {
6 public:
7     int data;
8     Node* berikut;
9 };
10
11 int main() {
12     // Langkah satu
13     Node* baru = new Node;
14     baru->data = 5;
15     baru->berikut = nullptr;
16     cout << "Isi data node baru adalah: " << baru->data << endl;
17
18     // Langkah dua
19     Node* lain = new Node;
20     lain->data = 6;
21     lain->berikut = nullptr;
22     cout << "Isi data node lain adalah: " << lain->data << endl;
23
24     // Langkah tiga: menyambung rantai
25     baru->berikut = lain;
26     cout << "Isi data node lain dicetak dari node baru adalah: ";
27     cout << baru->berikut->data << endl;
28
29     // Langkah empat
30     Node* kepala = baru;
31     cout << "Mencetak node pertama dari pointer kepala: ";
32     cout << kepala->data << endl;
33     cout << "Mencetak node kedua dari pointer kepala: ";
34     cout << kepala->berikut->data << endl;
35
36     // Langkah lima: pointer yang jalan-jalan
37     cout << "Menggunakan perulangan untuk mencetak setiap data pada rantai\n";
38     Node* jalan = kepala;
39     int i = 1;
40     while (jalan != nullptr) {
41         cout << "Data ke-" << i << " > " << jalan->data << endl;
42         i++;
43         jalan = jalan->berikut;
44     }
45
46     // Langkah enam: bukti bahwa pointer kepala tidak kehilangan data
47     cout << "Mencetak node pertama dari pointer kepala: ";
48     cout << kepala->data << endl;
49     cout << "Mencetak node kedua dari pointer kepala: ";
50     cout << kepala->berikut->data << endl;
51
52     return 0;
53 }

```

Gambar 1 kodingan yang sesuai dengan modul program linked list. (Sumber: Penulis)

The screenshot shows the Visual Studio Code interface. The left pane displays the code for `linkedListCounter.cpp`, which includes a class `Node` and a function `main`. The right pane shows the terminal output of the program's execution.

```

File Edit Selection View Go Run Terminal Help
modul.cpp - Visual Studio Code - Insiders [Administrator]
D:\Document Ndik\Kuliah\Semester 2\Algoritma Pemrograman\Praktikum\Pertemuan 10> modul.cpp -o "modul.exe"
8     Node* berikut;
9 };
10
11 int main() {
12     // Langkah satu
13     Node* baru = new Node;
14     baru->data = 5;
15     baru->berikut = nullptr;
16     cout << "Isi data node baru adalah: " << baru->data << endl;
17
18     // Langkah dua
19     Node* lain = new Node;
20     lain->data = 6;
21     lain->berikut = nullptr;
22 }

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE
PS D:\Document Ndik\Kuliah\Semester 2\Algoritma Pemrograman\Praktikum\Pertemuan 10> cd "d:\Document Ndik\Kuliah\Semester 2\Algoritma Pemrograman\Praktikum\Pertemuan 10\Langkah Praktikum" ; if ($?) [ p++ -o "modul.cpp" -o "modul.exe" ] ; if ($?) [ & "modul.exe" ]
Isi data node baru adalah: 5
Isi data node lain adalah: 6
Node* node ke-1 dari node baru adalah: 5
Men cetak node pertama dari pointer kepala: 5
Men cetak node kedua dari pointer kepala: 6
Menggunakan perulangan untuk mencetak setiap data pada rantai
Data ke-1 > 5
Data ke-2 > 6
Men cetak node pertama dari pointer kepala: 5
Men cetak node kedua dari pointer kepala: 6
PS D:\Document Ndik\Kuliah\Semester 2\Algoritma Pemrograman\Praktikum\Pertemuan 10\Langkah Praktikum>

```

Gambar 2 Output keseluruhannya. (Sumber: Penulis)

Program ini merupakan contoh implementasi dari linked list menggunakan bahasa pemrograman C++. Linked list adalah struktur data yang terdiri dari sejumlah simpul (node) yang terhubung satu sama lain melalui pointer. Setiap simpul memiliki data dan pointer yang menunjuk ke simpul berikutnya.

Pada program ini, terdapat kelas `Node` yang memiliki dua atribut yaitu data dan pointer berikut. Kemudian, program membuat dua simpul baru (baru dan lain) dan mengisi data pada masing-masing simpul. Selanjutnya, simpul baru dihubungkan dengan simpul lain sehingga membentuk sebuah rantai. Pointer kepala menunjuk ke simpul pertama pada rantai.

Program kemudian mencetak data pada simpul baru dan simpul lain, serta mencetak data pada simpul lain dari simpul baru. Selanjutnya, program mencetak data pada simpul pertama dan kedua dari pointer kepala.

Program juga menggunakan perulangan untuk mencetak setiap data pada rantai dengan menggunakan pointer jalan yang awalnya menunjuk ke simpul pertama pada rantai. Kemudian, pointer jalan digunakan untuk mengunjungi setiap simpul pada rantai dan mencetak datanya.

Terakhir, program mencetak kembali data pada simpul pertama dan kedua dari pointer kepala untuk membuktikan bahwa pointer kepala tidak kehilangan data.

Berikut adalah penjelasan lebih lengkap tiap langkah:

```

5  class Node {
6  public:
7      int data;
8      Node* berikut;
9  };

```

Gambar 3 class Node. (Sumber: Penulis)

Class Node adalah sebuah blueprint atau cetak biru untuk membuat objek-objek Node. Dalam kelas ini, terdapat dua anggota data yaitu data dan berikut. Anggota data data adalah sebuah integer yang akan menyimpan nilai data dari sebuah Node. Sedangkan anggota data berikut adalah sebuah pointer yang akan menunjuk ke Node berikutnya dalam sebuah linked list.

Dalam sebuah linked list, setiap Node akan memiliki anggota data dan anggota pointer berikut yang menunjuk ke Node berikutnya. Dalam kelas Node ini, anggota pointer berikut akan menunjuk ke Node berikutnya dalam linked list. Jika sebuah Node merupakan Node terakhir dalam linked list, maka anggota pointer berikut akan bernilai NULL.

Dalam kelas Node ini, anggota data dan anggota pointer berikut dideklarasikan sebagai public. Hal ini berarti bahwa anggota data dan anggota pointer berikut dapat diakses dari luar kelas Node. Sebagai contoh, jika kita ingin mengakses nilai data dari sebuah Node, kita dapat menggunakan sintaks node.data. Begitu juga dengan anggota pointer berikut, kita dapat mengaksesnya dengan sintaks node.berikut.

Dalam kelas Node ini, tidak terdapat konstruktor atau destruktur yang didefinisikan. Hal ini berarti bahwa ketika sebuah objek Node dibuat, anggota data dan anggota pointer berikut akan memiliki nilai acak. Jika kita ingin menginisialisasi nilai anggota data dan anggota pointer berikut saat membuat objek Node, kita dapat menambahkan konstruktor yang sesuai.

```

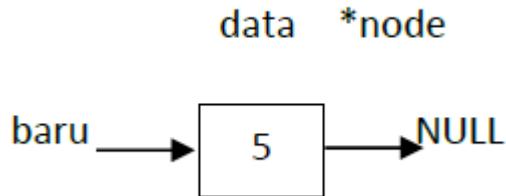
12 // Langkah satu
13 Node* baru = new Node;
14 baru->data = 5;
15 baru->berikut = nullptr;
16 cout << "Isi data node baru adalah: " << baru->data << endl;
17

```

Gambar 4 Langkah 1. (Sumber: Penulis)

Dalam kodingan ini, terdapat beberapa langkah yang dilakukan untuk membuat sebuah node baru. Langkah pertama adalah dengan mendeklarasikan sebuah pointer baru dengan tipe

data Node. Pointer ini akan menunjuk ke alamat memori dari node baru yang akan dibuat. Pada langkah ini, kita menggunakan operator new untuk mengalokasikan memori baru untuk node tersebut.



Gambar 5 Pointer ini akan menunjuk ke alamat memori dari node baru yang akan dibuat (5). (Sumber: Modul)

Setelah pointer baru berhasil dideklarasikan, langkah selanjutnya adalah mengisi data pada node tersebut. Pada kodingan ini, data yang diisikan adalah angka 5. Hal ini dilakukan dengan menggunakan operator "->" untuk mengakses variabel data pada node baru dan mengisinya dengan nilai 5.

Setelah data berhasil diisikan, langkah selanjutnya adalah mengatur pointer berikutnya pada node baru. Pada kodingan ini, pointer berikutnya diatur menjadi null, yang menandakan bahwa node baru ini tidak memiliki node berikutnya. Hal ini dilakukan dengan menggunakan operator "->" untuk mengakses variabel berikut pada node baru dan mengisinya dengan nilai null.

Setelah semua langkah di atas berhasil dilakukan, maka node baru sudah berhasil dibuat. Untuk memastikan bahwa node baru sudah terisi dengan benar, maka dilakukan output pada layar dengan menggunakan perintah cout. Pada kodingan ini, output yang dihasilkan adalah "Isi data node baru adalah: 5". Hal ini menandakan bahwa node baru sudah berhasil dibuat dan data yang diisikan pada node tersebut adalah 5.

Dalam kodingan ini, terdapat beberapa hal yang perlu diperhatikan. Pertama, tipe data yang digunakan adalah Node, yang kemungkinan merupakan tipe data yang sudah didefinisikan sebelumnya. Kedua, penggunaan operator "->" untuk mengakses variabel pada node baru. Operator ini digunakan karena pointer baru yang dideklarasikan merupakan pointer yang menunjuk ke alamat memori dari node baru tersebut. Ketiga, penggunaan operator new untuk mengalokasikan memori baru untuk node baru. Operator ini digunakan untuk menghindari terjadinya kesalahan pada saat penggunaan memori yang tidak teralokasikan.

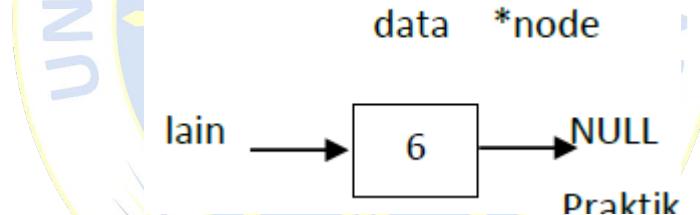
```

18 // Langkah dua
19 Node* lain = new Node;
20 lain->data = 6;
21 lain->berikut = nullptr;
22 cout << "Isi data node lain adalah: " << lain->data << endl;
23

```

Gambar 6 Langkah 2. (Sumber: Penulis)

Pada bagian kodingan ini, terdapat beberapa langkah yang dilakukan untuk membuat sebuah node baru yang berbeda dengan node sebelumnya. Langkah pertama adalah dengan mendeklarasikan sebuah pointer baru dengan tipe data Node. Pointer ini akan menunjuk ke alamat memori dari node baru yang akan dibuat. Pada langkah ini, kita menggunakan operator new untuk mengalokasikan memori baru untuk node tersebut.



Gambar 7 Pointer ini akan menunjuk ke alamat memori dari node baru yang akan dibuat (6). (Sumber: Penulis)

Setelah pointer baru berhasil dideklarasikan, langkah selanjutnya adalah mengisi data pada node tersebut. Pada kodingan ini, data yang diisikan adalah angka 6. Hal ini dilakukan dengan menggunakan operator "->" untuk mengakses variabel data pada node baru dan mengisinya dengan nilai 6.

Setelah data berhasil diisikan, langkah selanjutnya adalah mengatur pointer berikutnya pada node baru. Pada kodingan ini, pointer berikutnya diatur menjadi null, yang menandakan bahwa node baru ini tidak memiliki node berikutnya. Hal ini dilakukan dengan menggunakan operator "->" untuk mengakses variabel berikut pada node baru dan mengisinya dengan nilai null.

Setelah semua langkah di atas berhasil dilakukan, maka node baru sudah berhasil dibuat. Untuk memastikan bahwa node baru sudah terisi dengan benar, maka dilakukan output pada layar dengan menggunakan perintah cout. Pada kodingan ini, output yang dihasilkan

adalah "Isi data node lain adalah: 6". Hal ini menandakan bahwa node baru sudah berhasil dibuat dan data yang diisikan pada node tersebut adalah 6.

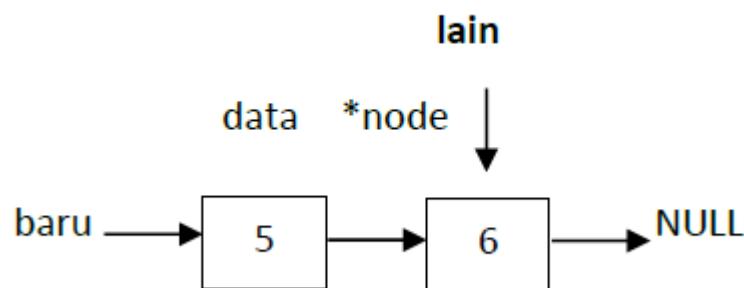
```

24 // Langkah tiga: menyambung rantai
25 baru->berikut = lain;
26 cout << "Isi data node lain dicetak dari node baru adalah: ";
27 cout << baru->berikut->data << endl;

```

Gambar 8 Langkah 3. (Sumber: Penulis)

Pada bagian kodingan ini, terdapat tiga langkah yang dilakukan untuk menyambung rantai. Langkah ketiga dilakukan dengan menggunakan dua baris kode. Pertama, kita menghubungkan node baru dengan node lain menggunakan operator "->". Operator "->" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "baru" menunjuk ke node baru dan pointer "lain" menunjuk ke node lain. Dengan menggunakan operator "->", kita menghubungkan node baru dengan node lain sehingga node baru menjadi node berikutnya dari node lain.



Gambar 9_Menghubungkan node baru dengan node lain. (Sumber: Modul)

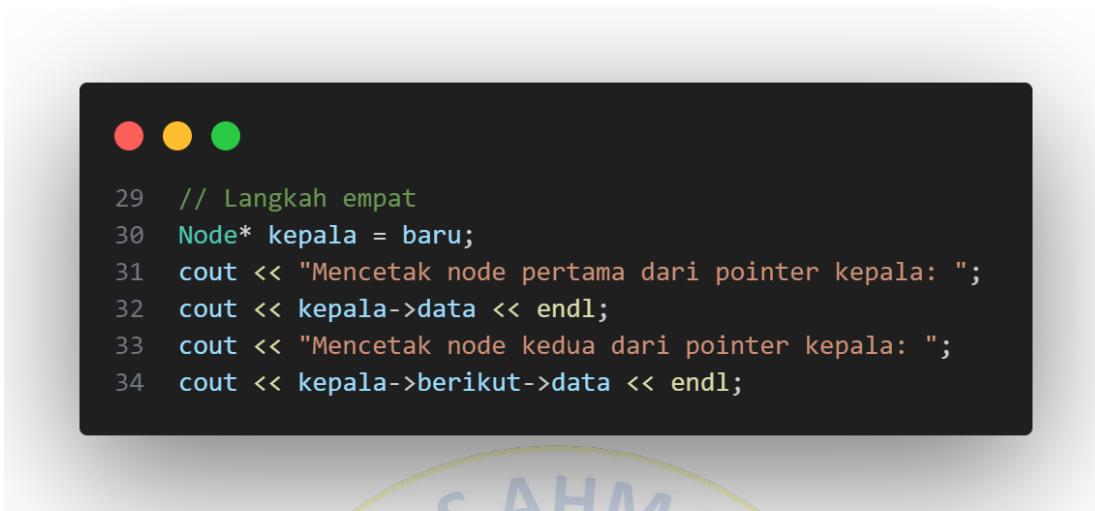
Setelah itu, kita mencetak isi data dari node lain yang dicetak dari node baru. Hal ini dilakukan dengan menggunakan objek "cout" yang digunakan untuk mencetak teks ke layar. Pertama, kita mencetak teks "Isi data node lain dicetak dari node baru adalah: ". Kemudian, kita mencetak data dari node berikutnya dari node baru menggunakan operator "->". Operator "->" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "baru" menunjuk ke node baru dan pointer "berikut" menunjuk ke node berikutnya dari node baru. Dengan menggunakan operator "->", kita mengakses data dari node berikutnya dari node baru dan mencetaknya ke layar.

Dalam konteks yang lebih luas, bagian kodingan ini merupakan bagian dari implementasi dari struktur data linked list. Linked list adalah struktur data yang terdiri dari sejumlah node yang saling terhubung. Setiap node memiliki dua anggota, yaitu data dan pointer ke node berikutnya. Dalam implementasi linked list, kita menggunakan pointer untuk menghubungkan node-node tersebut sehingga membentuk sebuah rantai. Dalam bagian kodingan ini, kita melakukan langkah ketiga dari implementasi linked list, yaitu menyambungkan node baru dengan node lain sehingga membentuk sebuah rantai.

Dalam implementasi linked list, langkah ketiga ini sangat penting karena menyambungkan node baru dengan node lain merupakan cara untuk menambahkan node baru ke dalam linked list. Dalam langkah ini, kita menggunakan operator "`->`" untuk menghubungkan node baru dengan node lain. Operator "`->`" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "baru" menunjuk ke node baru dan pointer "lain" menunjuk ke node lain. Dengan menggunakan operator "`->`", kita menghubungkan node baru dengan node lain sehingga membentuk sebuah rantai.

Setelah kita menyambungkan node baru dengan node lain, kita mencetak isi data dari node lain yang dicetak dari node baru. Hal ini dilakukan untuk memastikan bahwa node baru telah terhubung dengan node lain dengan benar. Dalam hal ini, kita menggunakan objek "cout" yang digunakan untuk mencetak teks ke layar. Pertama, kita mencetak teks "Isi data node lain dicetak dari node baru adalah: ". Kemudian, kita mencetak data dari node berikutnya dari node baru menggunakan operator "`->`". Operator "`->`" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "baru" menunjuk ke node baru dan pointer "berikut" menunjuk ke node berikutnya dari node baru. Dengan menggunakan operator "`->`", kita mengakses data dari node berikutnya dari node baru dan mencetaknya ke layar.

Dalam implementasi linked list, langkah ketiga ini merupakan langkah terakhir dari proses penambahan node baru ke dalam linked list. Setelah langkah ini dilakukan, node baru telah terhubung dengan node lain dan membentuk sebuah rantai. Selanjutnya, kita dapat melakukan operasi-operasi lain pada linked list seperti menghapus node, mencari node, atau mengurutkan node.



```

29 // Langkah empat
30 Node* kepala = baru;
31 cout << "Mencetak node pertama dari pointer kepala: ";
32 cout << kepala->data << endl;
33 cout << "Mencetak node kedua dari pointer kepala: ";
34 cout << kepala->berikut->data << endl;

```

Gambar 10 Langkah 4. (Sumber: Penulis)

Pada bagian kodingan ini, terdapat tiga langkah yang dilakukan untuk mencetak node pertama dan node kedua dari pointer kepala. Langkah keempat dilakukan dengan menggunakan tiga baris kode. Pertama, kita membuat sebuah pointer baru yang menunjuk ke node baru menggunakan tipe data "Node*". Tipe data "Node*" digunakan untuk menyimpan alamat memori dari sebuah node dalam linked list. Dalam hal ini, pointer "kepala" menunjuk ke node baru yang merupakan node pertama dalam linked list.

Setelah itu, kita mencetak node pertama dari pointer kepala. Hal ini dilakukan dengan menggunakan objek "cout" yang digunakan untuk mencetak teks ke layar. Pertama, kita mencetak teks "Mencetak node pertama dari pointer kepala: ". Kemudian, kita mencetak data dari node pertama menggunakan operator "->". Operator "->" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "kepala" menunjuk ke node pertama dan kita mengakses data dari node pertama menggunakan operator "->". Setelah itu, kita mencetak data dari node pertama ke layar.

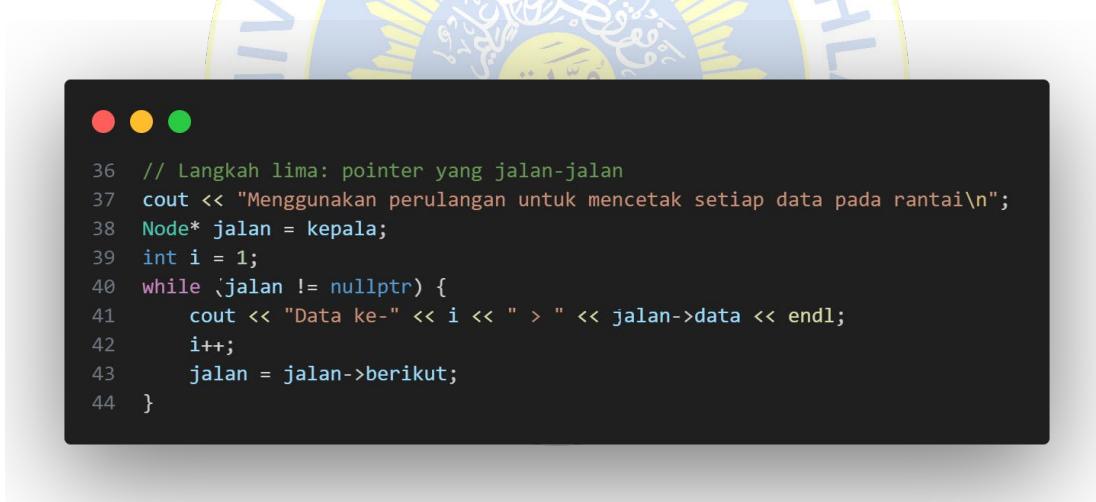
Setelah itu, kita mencetak node kedua dari pointer kepala. Hal ini dilakukan dengan menggunakan objek "cout" yang digunakan untuk mencetak teks ke layar. Pertama, kita mencetak teks "Mencetak node kedua dari pointer kepala: ". Kemudian, kita mencetak data dari node kedua menggunakan operator "->". Operator "->" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "kepala" menunjuk ke node pertama dan kita mengakses data dari node kedua menggunakan operator "->". Karena node kedua merupakan node berikutnya dari node pertama, kita menggunakan anggota "berikut" dari node pertama untuk mengakses node kedua. Setelah itu, kita mencetak data dari node kedua ke layar.

Dalam konteks yang lebih luas, bagian kodingan ini merupakan bagian dari implementasi dari struktur data linked list. Linked list adalah struktur data yang terdiri dari sejumlah node yang saling terhubung. Setiap node memiliki dua anggota, yaitu data dan pointer

ke node berikutnya. Dalam implementasi linked list, kita menggunakan pointer untuk menghubungkan node-node tersebut sehingga membentuk sebuah rantai. Dalam bagian kodingan ini, kita melakukan langkah keempat dari implementasi linked list, yaitu mencetak node pertama dan node kedua dari pointer kepala.

Dalam implementasi linked list, langkah keempat ini sangat penting karena node pertama merupakan titik awal dari linked list. Dalam langkah ini, kita menggunakan pointer "kepala" untuk menunjuk ke node pertama dalam linked list. Dengan menggunakan pointer "kepala", kita dapat mengakses data dari node pertama dan node kedua menggunakan operator "`->`". Operator "`->`" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "kepala" menunjuk ke node pertama dan kita mengakses data dari node pertama dan node kedua menggunakan operator "`->`". Setelah itu, kita mencetak data dari node pertama dan node kedua ke layar.

Dalam implementasi linked list, langkah keempat ini merupakan langkah terakhir dari proses pembuatan linked list. Setelah langkah ini dilakukan, kita telah berhasil membuat linked list yang terdiri dari sejumlah node yang saling terhubung. Selanjutnya, kita dapat melakukan operasi-operasi lain pada linked list seperti menghapus node, mencari node, atau mengurutkan node. Dalam hal ini, kita telah berhasil mencetak node pertama dan node kedua dari pointer kepala yang merupakan titik awal dari linked list.



```

● ● ●
36 // Langkah lima: pointer yang jalan-jalan
37 cout << "Menggunakan perulangan untuk mencetak setiap data pada rantai\n";
38 Node* jalan = kepala;
39 int i = 1;
40 while (jalan != nullptr) {
41     cout << "Data ke-" << i << " > " << jalan->data << endl;
42     i++;
43     jalan = jalan->berikut;
44 }

```

Gambar 11 Langkah 5. (Sumber: Penulis)

Pada bagian kodingan ini, terdapat tiga langkah yang dilakukan untuk mencetak setiap data pada rantai. Langkah kelima dilakukan dengan menggunakan empat baris kode. Pertama, kita mencetak teks "Menggunakan perulangan untuk mencetak setiap data pada rantai" menggunakan objek "cout" yang digunakan untuk mencetak teks ke layar. Hal ini dilakukan untuk memberikan informasi kepada pengguna bahwa kita akan mencetak setiap data pada rantai menggunakan perulangan.

Setelah itu, kita membuat sebuah pointer baru yang menunjuk ke node pertama dalam linked list menggunakan tipe data "Node*". Tipe data "Node*" digunakan untuk menyimpan alamat memori dari sebuah node dalam linked list. Dalam hal ini, pointer "jalan" menunjuk ke node pertama dalam linked list.

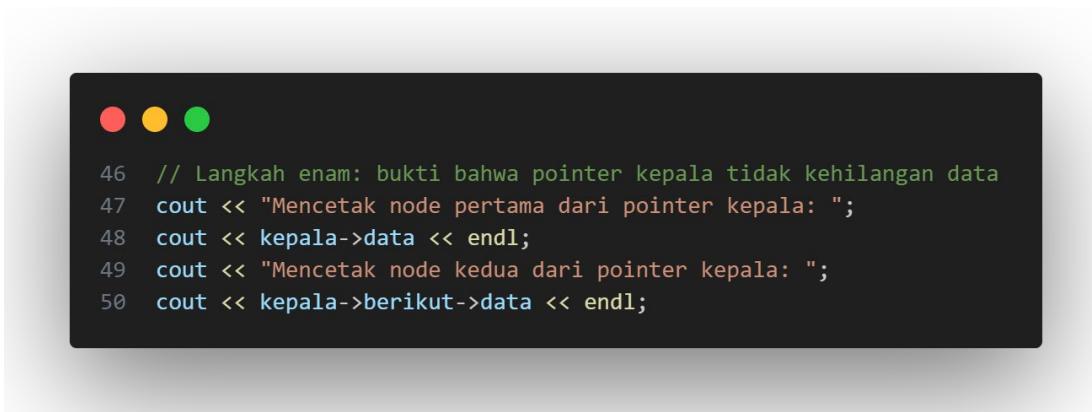
Selanjutnya, kita menggunakan perulangan "while" untuk mencetak setiap data pada rantai. Perulangan "while" digunakan untuk melakukan iterasi selama kondisi yang diberikan bernilai "true". Dalam hal ini, kita menggunakan kondisi "jalan != nullptr" sebagai kondisi perulangan. Kondisi ini berarti perulangan akan terus dilakukan selama pointer "jalan" tidak menunjuk ke alamat memori "nullptr". Alamat memori "nullptr" digunakan untuk menandakan akhir dari linked list.

Dalam setiap iterasi perulangan, kita mencetak data dari node yang ditunjuk oleh pointer "jalan". Hal ini dilakukan dengan menggunakan objek "cout" yang digunakan untuk mencetak teks ke layar. Pertama, kita mencetak teks "Data ke-" dan nomor iterasi "i". Kemudian, kita mencetak data dari node yang ditunjuk oleh pointer "jalan" menggunakan operator "->". Operator "->" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "jalan" menunjuk ke node yang sedang diiterasi dan kita mengakses data dari node tersebut menggunakan operator "->". Setelah itu, kita mencetak data dari node tersebut ke layar.

Dalam konteks yang lebih luas, bagian kodingan ini merupakan bagian dari implementasi dari struktur data linked list. Linked list adalah struktur data yang terdiri dari sejumlah node yang saling terhubung. Setiap node memiliki dua anggota, yaitu data dan pointer ke node berikutnya. Dalam implementasi linked list, kita menggunakan pointer untuk menghubungkan node-node tersebut sehingga membentuk sebuah rantai. Dalam bagian kodingan ini, kita melakukan langkah kelima dari implementasi linked list, yaitu mencetak setiap data pada rantai menggunakan perulangan.

Dalam implementasi linked list, langkah kelima ini sangat penting karena kita dapat mengakses setiap data pada linked list menggunakan perulangan. Dalam hal ini, kita menggunakan perulangan "while" untuk melakukan iterasi selama kondisi yang diberikan bernilai "true". Dalam setiap iterasi perulangan, kita mencetak data dari node yang ditunjuk oleh pointer "jalan". Hal ini dilakukan untuk menampilkan setiap data pada linked list ke layar.

Dalam implementasi linked list, langkah kelima ini merupakan langkah terakhir dari proses pembuatan linked list. Setelah langkah ini dilakukan, kita telah berhasil membuat linked list yang terdiri dari sejumlah node yang saling terhubung. Selanjutnya, kita dapat melakukan operasi-operasi lain pada linked list seperti menghapus node, mencari node, atau mengurutkan node. Dalam hal ini, kita telah berhasil mencetak setiap data pada rantai menggunakan perulangan "while" yang merupakan cara yang efektif untuk mengakses setiap data pada linked list.



```
46 // Langkah enam: bukti bahwa pointer kepala tidak kehilangan data
47 cout << "Mencetak node pertama dari pointer kepala: ";
48 cout << kepala->data << endl;
49 cout << "Mencetak node kedua dari pointer kepala: ";
50 cout << kepala->berikut->data << endl;
```

Gambar 12 Langkah 6. (Sumber: Penulis)

Pada bagian kodingan ini, terdapat tiga baris kode yang digunakan untuk membuktikan bahwa pointer kepala tidak kehilangan data. Langkah keenam dilakukan dengan menggunakan tiga baris kode. Pertama, kita mencetak teks "Mencetak node pertama dari pointer kepala: " menggunakan objek "cout" yang digunakan untuk mencetak teks ke layar. Hal ini dilakukan untuk memberikan informasi kepada pengguna bahwa kita akan mencetak data dari node pertama menggunakan pointer kepala.

Setelah itu, kita mencetak data dari node pertama menggunakan operator "->". Operator "->" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "kepala" menunjuk ke node pertama dan kita mengakses data dari node pertama menggunakan operator "->". Setelah itu, kita mencetak data dari node pertama ke layar.

Selanjutnya, kita mencetak teks "Mencetak node kedua dari pointer kepala: " menggunakan objek "cout" yang digunakan untuk mencetak teks ke layar. Hal ini dilakukan untuk memberikan informasi kepada pengguna bahwa kita akan mencetak data dari node kedua menggunakan pointer kepala.

Setelah itu, kita mencetak data dari node kedua menggunakan operator "->". Operator "->" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "kepala" menunjuk ke node pertama dan kita mengakses data dari node kedua menggunakan operator "->". Karena node kedua merupakan node berikutnya dari node pertama, kita menggunakan anggota "berikut" dari node pertama untuk mengakses node kedua. Setelah itu, kita mencetak data dari node kedua ke layar.

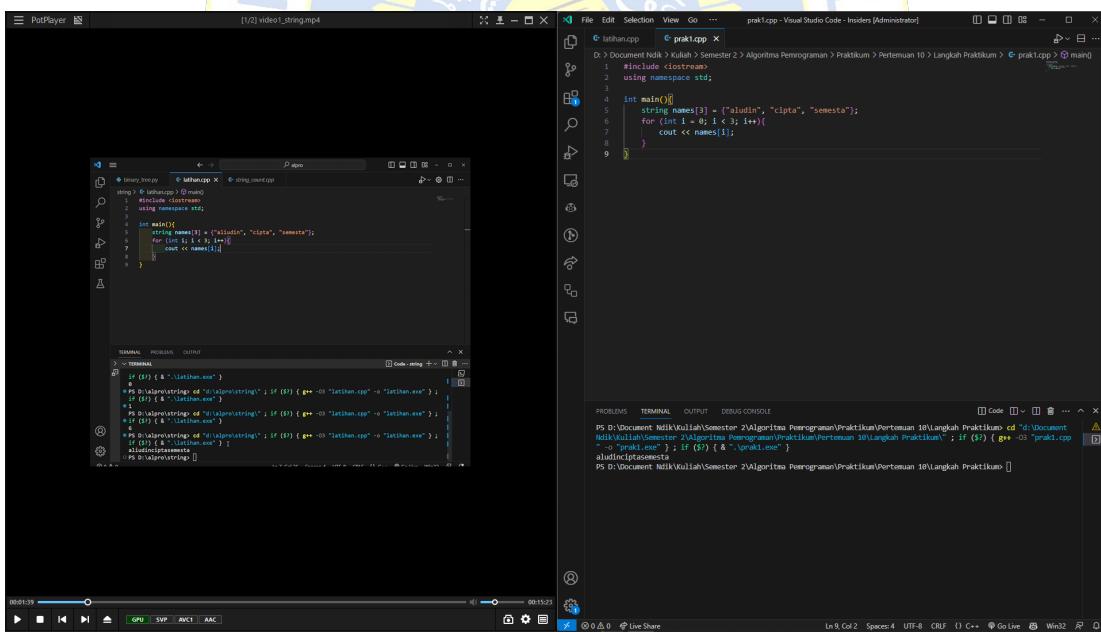
Dalam konteks yang lebih luas, bagian kodingan ini merupakan bagian dari implementasi dari struktur data linked list. Linked list adalah struktur data yang terdiri dari sejumlah node yang saling terhubung. Setiap node memiliki dua anggota, yaitu data dan pointer ke node berikutnya. Dalam implementasi linked list, kita menggunakan pointer untuk menghubungkan node-node tersebut sehingga membentuk sebuah rantai. Dalam bagian

kodingan ini, kita melakukan langkah keenam dari implementasi linked list, yaitu membuktikan bahwa pointer kepala tidak kehilangan data.

Dalam implementasi linked list, langkah keenam ini sangat penting karena pointer kepala merupakan titik awal dari linked list. Dalam langkah ini, kita menggunakan pointer "kepala" untuk membuktikan bahwa data dari node pertama dan node kedua masih tersimpan dengan baik. Dalam hal ini, kita mencetak data dari node pertama dan node kedua menggunakan operator "->". Operator "->" digunakan untuk mengakses anggota dari sebuah objek yang ditunjuk oleh pointer. Dalam hal ini, pointer "kepala" menunjuk ke node pertama dan kita mengakses data dari node pertama dan node kedua menggunakan operator "->". Setelah itu, kita mencetak data dari node pertama dan node kedua ke layar.

Dalam implementasi linked list, langkah keenam ini merupakan langkah terakhir dari proses pembuatan linked list. Setelah langkah ini dilakukan, kita telah berhasil membuat linked list yang terdiri dari sejumlah node yang saling terhubung. Selanjutnya, kita dapat melakukan operasi-operasi lain pada linked list seperti menghapus node, mencari node, atau mengurutkan node. Dalam hal ini, kita telah berhasil membuktikan bahwa pointer kepala tidak kehilangan data dari node pertama dan node kedua yang merupakan titik awal dari linked list.

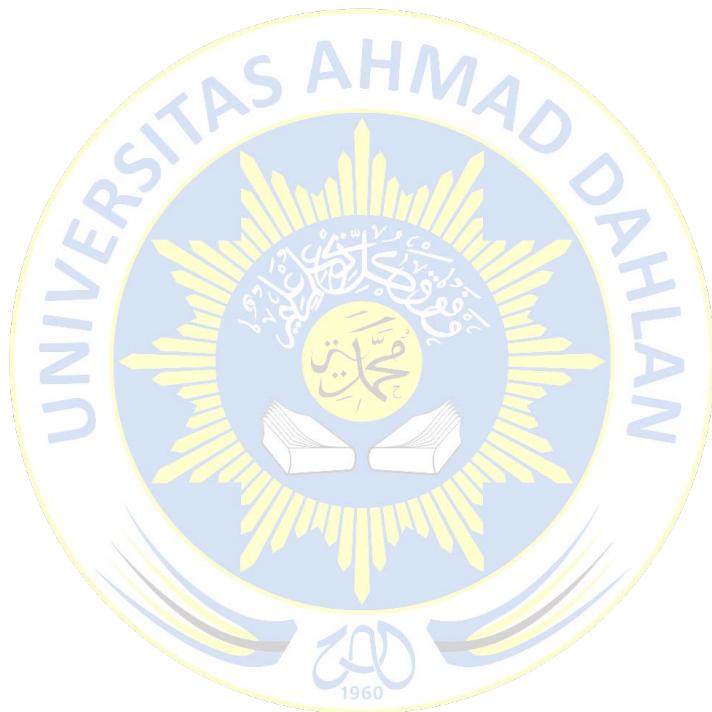
video1_string.mp4



Gambar 13 Mencetak string tanpa spasi. (Sumber: Penulis)

Kode ini adalah program C++ sederhana yang mencetak tiga nama ke layar. Pertama, kita mengimpor pustaka iostream yang berisi fungsi-fungsi input/output standar. Kemudian, kita mendefinisikan fungsi main() yang merupakan titik masuk program. Di dalam fungsi main(),

kita mendeklarasikan sebuah array bernama name yang berisi tiga string. Setelah itu, kita menggunakan loop for untuk mencetak setiap elemen dari array name ke layar menggunakan fungsi cout. Akhirnya, program selesai dan fungsi main() mengembalikan nilai 0.



Kode yang diberikan tidak memiliki spasi antara setiap elemen string pada array name, sehingga ketika program dijalankan, setiap elemen akan dicetak secara berurutan tanpa spasi di antaranya. Sebagai contoh, output yang dihasilkan dari kode ini adalah "alaudinciptasemesta". Jika Anda ingin menambahkan spasi antara setiap elemen, Anda dapat memodifikasi kode dengan menambahkan spasi di antara tanda kutip pada setiap elemen string pada array name.

```

1 #include <iostream>
2
3 using namespace std;
4
5 int main(){
6     string name[3] = {"alaudin", "cipta", "semesta"};
7     for (int i = 0; i < 3; i++){
8         cout << name[i] << " ";
9     }
10 }

```

Gambar 14 Mencetak string dengan spasi. (Sumber: Penulis)

Kode ini adalah program C++ sederhana yang mencetak tiga nama ke layar dengan menambahkan spasi di antara setiap elemen string pada array name. Pertama, kita mengimpor pustaka iostream yang berisi fungsi-fungsi input/output standar. Kemudian, kita mendefinisikan fungsi main() yang merupakan titik masuk program. Di dalam fungsi main(), kita mendeklarasikan sebuah array bernama name yang berisi tiga string. Setelah itu, kita menggunakan loop for untuk mencetak setiap elemen dari array name ke layar menggunakan fungsi cout dan menambahkan spasi di antara setiap elemen menggunakan operator <<. Akhirnya, program selesai dan fungsi main() mengembalikan nilai 0.

Perbedaan antara kode ini dengan kode sebelumnya adalah bahwa kode ini menambahkan spasi di antara setiap elemen string pada array name saat mencetak ke layar, sehingga output yang dihasilkan adalah "alaudin cipta semesta " (dengan spasi di antara setiap elemen). Sedangkan pada kode sebelumnya, output yang dihasilkan adalah "alaudinciptasemesta" (tanpa spasi di antara setiap elemen).

```

1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     string str1 = "hai";
8     string str2 = "Hai";
9     bool result = str1 == str2;
10    cout << result << endl;
11    str2 = "hai";
12    result = str1 == str2;
13    cout << result;
14 }

```

Gambar 15 Pengecekan nilai string. (Sumber: Penulis)

Kode ini adalah program C++ sederhana yang membandingkan dua string dan mencetak hasil perbandingannya ke layar. Program ini menggunakan dua variabel string: str1 dan str2. Variabel str1 diinisialisasi dengan nilai "hai", sedangkan variabel str2 diinisialisasi dengan nilai "Hai". Variabel result adalah boolean yang menunjukkan apakah str1 sama dengan str2 atau tidak.

Di dalam fungsi main(), kita mendeklarasikan dua variabel string bernama str1 dan str2 dengan nilai "hai" dan "Hai". Variabel str1 dan str2 adalah dua string yang akan dibandingkan. Setelah itu, kita menggunakan operator == untuk membandingkan str1 dan str2. Hasil perbandingan disimpan ke dalam variabel result. Setelah itu, kita mencetak nilai result ke layar menggunakan fungsi cout.

Kemudian, kita mengubah nilai variabel str2 menjadi "hai" dan membandingkannya lagi dengan str1 menggunakan operator ==. Hasil perbandingan disimpan ke dalam variabel result dan dicetak ke layar menggunakan fungsi cout.

```

[1/2] video1_string.mp4
File Edit Selection View Go ... prak4.cpp - Visual Studio Code - Insiders [Administrator]
D:\Document NIKK> Kuliah > Semester 2 > Algoritma Pemrograman> Praktikum > Pertemuan 10 > Langkah Praktikum > C\prak4.cpp > main()
1 #include <iostream>
2
3 using namespace std;
4
5 int main(){
6     string name = "Ardiansyah";
7     int length = 0;
8     while (name[length] != '\0'){
9         length = length + 1;
10    }
11 }

TERMINAL PROBLEMS OUTPUT
D:\> <TERMINAL>
$ if ($1) (& ./lithen.exe)
$ ./lithen
PS D:\lithen>string Ardiansyah
panjang string Ardiansyah = 10
PS D:\lithen>string Ardiansyah

```

TERMINAL PROBLEMS OUTPUT

```

PS D:\Document NIKK> Kuliah > Semester 2 > Algoritma Pemrograman> Praktikum > Pertemuan 10 > Langkah Praktikum > C\prak4.cpp > main()
$ if ($1) (& ./prak4.exe )
$ if ($1) (& ./prak4.exe )
panjang string Ardiansyah = 10
PS D:\Document NIKK> Kuliah > Semester 2 > Algoritma Pemrograman> Praktikum > Pertemuan 10 > Langkah Praktikum

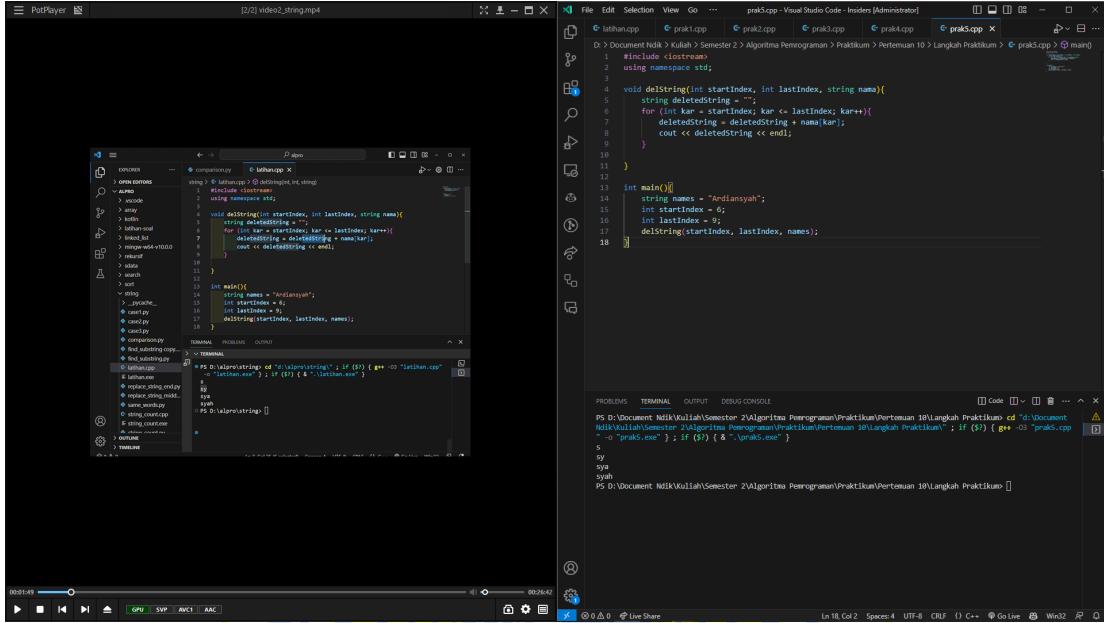
```

Gambar 16 Mengecek jumlah string pada nama. (Sumber: Penulis)

Kode ini adalah program C++ sederhana yang menghitung panjang sebuah string menggunakan loop while. Pertama, kita mengimpor pustaka iostream yang berisi fungsi-fungsi input/output standar. Kemudian, kita mendefinisikan fungsi main() yang merupakan titik masuk program. Di dalam fungsi main(), kita mendeklarasikan sebuah variabel string bernama name dengan nilai "Ardiansyah". Setelah itu, kita mendeklarasikan sebuah variabel integer bernama length dengan nilai 0. Kemudian, kita menggunakan loop while untuk menghitung panjang string name dengan mengecek setiap karakter pada string name menggunakan indeks array dan menambahkan nilai length jika karakter yang diperiksa bukan karakter null ('\0'). Akhirnya, kita mencetak hasil perhitungan ke layar menggunakan fungsi cout.

Output yang dihasilkan dari kode ini adalah "Panjang string Ardiansyah = 10", karena panjang string "Ardiansyah" adalah 10 karakter. Loop while digunakan untuk menghitung panjang string dengan mengecek setiap karakter pada string menggunakan indeks array dan menambahkan nilai length jika karakter yang diperiksa bukan karakter null ('\0'). Karakter null ('\0') digunakan sebagai penanda akhir dari sebuah string, sehingga loop while akan berhenti ketika karakter null ('\0') ditemukan.

Video2_string.mp4



Gambar 17 Mencetak sebagian karakter dari string yang diberikan dengan langkah. (Sumber: Penulis)

Program ini terdiri dari dua bagian utama: fungsi `delString` dan `main`. Berikut adalah penjelasan mengenai kedua bagian tersebut.

Fungsi `delString` memiliki tiga parameter: `startIndex`, `lastIndex`, dan `nama`. Fungsi ini bertujuan untuk mengambil sebagian karakter dari string `nama` yang dimulai dari indeks `startIndex` hingga indeks `lastIndex`. Setelah itu, fungsi ini akan mencetak karakter yang diambil tersebut satu per satu.

Fungsi `main` adalah titik awal program. Dalam fungsi ini, kita mendefinisikan variabel yang akan digunakan sebagai argumen untuk memanggil fungsi `delString`.

Berikut adalah penjelasan langkah demi langkah dari algoritma fungsi `main`:

- Deklarasikan variabel `name` dengan nilai "Ardiansyah".
- Deklarasikan variabel `startIndex` dengan nilai 6.
- Deklarasikan variabel `lastIndex` dengan nilai 9.
- Panggil fungsi `delString` dengan argumen `startIndex`, `lastIndex`, dan `name`.

Sekarang kita akan membahas output yang dihasilkan oleh program ini. Berdasarkan nilai-nilai yang didefinisikan dalam fungsi `main`, fungsi `delString` akan mengambil karakter dari string "Ardiansyah" yang dimulai dari indeks 6 hingga indeks 9. Karakter pada indeks tersebut adalah "nsyah". Fungsi `delString` akan mencetak karakter tersebut satu per satu, seperti berikut:

Program ini bertujuan untuk mencetak sebagian karakter dari string yang diberikan. Program ini terdiri dari dua bagian utama: fungsi delString dan main. Fungsi delString digunakan untuk mengambil dan mencetak karakter dari string, sedangkan fungsi main digunakan untuk mendefinisikan variabel yang akan digunakan sebagai argumen untuk memanggil fungsi delString. Output yang dihasilkan oleh program ini adalah karakter yang diambil dari string "Ardiansyah" yang dimulai dari indeks 6 hingga indeks 9, yang dicetak satu per satu.

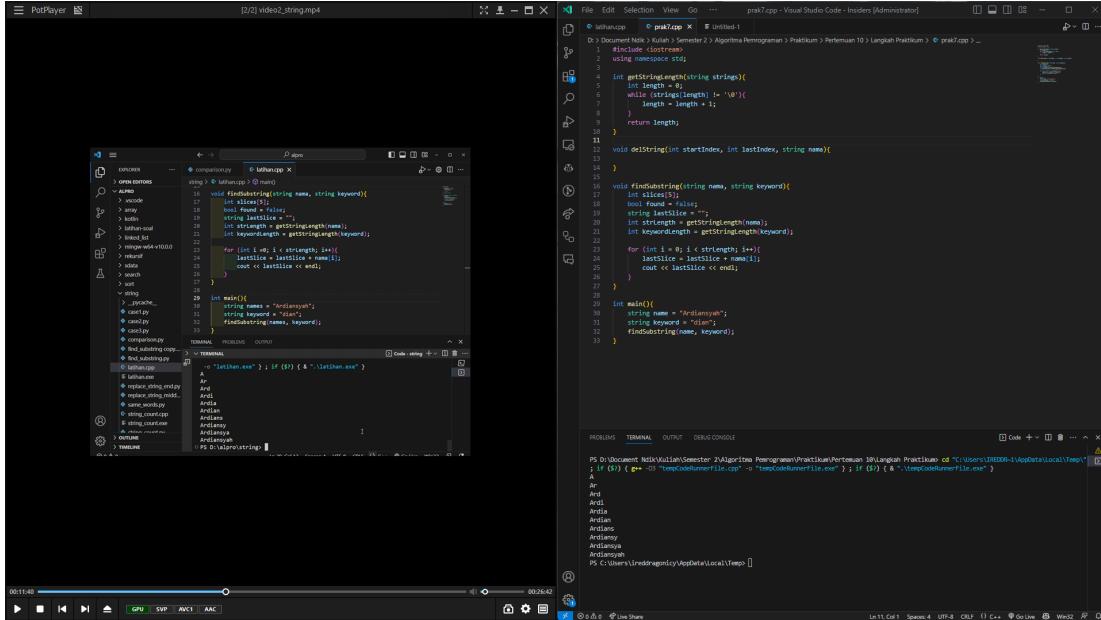
```

1 #include <iostream>
2
3 void delString(int startIndex, int lastIndex, string name){
4     string deletedString = "";
5     for (int kar = startIndex; kar <= lastIndex; kar++){
6         deletedString += name[kar];
7     }
8     cout << deletedString;
9 }
10
11 int main(){
12     string name = "Ardiansyah";
13     int startIndex = 6;
14     int lastIndex = 9;
15     delString(startIndex, lastIndex, name);
16 }
17
18 
```

Gambar 18 Mencetak sebagian karakter dari string yang diberikan yang menampilkan hasil saja. (Sumber: Penulis)

Kode ini adalah program C++ sederhana yang menghapus sebagian string dari sebuah string yang diberikan. Program ini menggunakan fungsi delString() yang menerima tiga parameter: startIndex, lastIndex, dan nama. Parameter startIndex dan lastIndex menunjukkan indeks awal dan akhir dari substring yang akan dihapus dari string nama. Fungsi delString() menggunakan loop for untuk mengambil setiap karakter pada substring yang akan dihapus dan menyimpannya ke dalam variabel deletedString. Setiap karakter yang diambil ditambahkan ke deletedString menggunakan operator +. Setelah loop selesai, fungsi delString() mencetak deletedString ke layar menggunakan fungsi cout.

Di dalam fungsi main(), kita mendeklarasikan sebuah variabel string bernama name dengan nilai "Ardiansyah". Kemudian, kita mendeklarasikan dua variabel integer bernama startIndex dan lastIndex dengan nilai 6 dan 9. Variabel startIndex dan lastIndex menunjukkan indeks awal dan akhir dari substring yang akan dihapus dari string name. Setelah itu, kita memanggil fungsi delString() dengan tiga parameter: startIndex, lastIndex, dan name.



Gambar 19 Mencari substring dalam sebuah string. (Sumber: Penulis)

Kode ini adalah program C++ sederhana yang mencari substring dalam sebuah string. Program ini menggunakan tiga fungsi: `getStringLength()`, `delString()`, dan `findSubstring()`. Fungsi `getStringLength()` digunakan untuk menghitung panjang sebuah string. Fungsi `delString()` digunakan untuk menghapus sebagian string dari sebuah string yang diberikan. Fungsi `findSubstring()` digunakan untuk mencari substring dalam sebuah string.

Di dalam fungsi `main()`, kita mendeklarasikan dua variabel string bernama `name` dan `keyword` dengan nilai "Ardiansyah" dan "dian". Variabel `name` adalah string yang akan dicari substring-nya, sedangkan variabel `keyword` adalah substring yang akan dicari dalam string `name`. Setelah itu, kita memanggil fungsi `findSubstring()` dengan dua parameter: `name` dan `keyword`.

Fungsi `findSubstring()` memiliki lima variabel lokal: `slices`, `found`, `lastSlice`, `strLength`, dan `keywordLength`. Variabel `slices` adalah array integer dengan lima elemen yang digunakan untuk menyimpan indeks awal dari setiap kemunculan substring dalam string `name`. Variabel `found` adalah boolean yang menunjukkan apakah substring ditemukan dalam string `name` atau tidak. Variabel `lastSlice` adalah string yang digunakan untuk menyimpan substring terakhir yang diambil dari string `name`. Variabel `strLength` dan `keywordLength` adalah integer yang menunjukkan panjang string `name` dan `keyword`.

Fungsi `findSubstring()` menggunakan loop `for` untuk mengambil setiap karakter pada string `name` dan menyimpannya ke dalam variabel `lastSlice`. Setiap karakter yang diambil ditambahkan ke `lastSlice` menggunakan operator `+`. Setelah setiap karakter ditambahkan, fungsi `findSubstring()` mencetak `lastSlice` ke layar menggunakan fungsi `cout`.

Output ini menunjukkan setiap substring yang diambil dari string "Ardiansyah". Loop for pada fungsi findSubstring() digunakan untuk mengambil setiap karakter pada string name dan menyimpannya ke dalam variabel lastSlice. Setiap karakter yang diambil ditambahkan ke lastSlice menggunakan operator +. Setelah setiap karakter ditambahkan, fungsi findSubstring() mencetak lastSlice ke layar menggunakan fungsi cout.

```

1 #include <iostream>
2
3 using namespace std;
4
5 int getStringLength(string str);
6
7 void delString(string& str);
8
9 string findSubstring(string name, string keyword);
10
11 string delString(string lastSlice);
12
13 int lastSliceLength = getStringLength(lastSlice);
14
15 for (int kar = 1; kar < lastSliceLength; kar++) {
16     deletedString = deletedString + lastSlice[kar];
17 }
18
19 return deletedString;
20
21 void findSubstring(string name, string keyword) {
22     string slices[5];
23     bool Found = false;
24     int keywordLength = getStringLength(keyword);
25     int nameLength = getStringLength(name);
26
27     for (int i = 0; i < nameLength; i++) {
28         lastSlice = name[i];
29         slices[4] = lastSlice;
30         slices[3] = lastSlice;
31         slices[2] = lastSlice;
32         slices[1] = lastSlice;
33         slices[0] = lastSlice;
34
35         if (lastSliceLength == keywordLength && lastSlice == keyword) {
36             Found = true;
37             cout << lastSlice;
38         }
39     }
40
41 }
42
43 main() {
44
45 }

```

Gambar 20 Mencari substring pada sebuah string. (Sumber: Penulis)

Program di atas merupakan program yang ditulis dalam bahasa pemrograman C++. Program ini bertujuan untuk mencari substring pada sebuah string. Substring yang dicari adalah substring yang tidak sama dengan keyword yang diberikan. Program ini terdiri dari beberapa fungsi dan satu fungsi utama yang akan memanggil fungsi-fungsi tersebut.

Fungsi pertama yang ada pada program ini adalah fungsi getStringLength. Fungsi ini bertujuan untuk menghitung panjang sebuah string. Fungsi ini menerima sebuah parameter berupa string dan akan mengembalikan sebuah bilangan bulat yang merupakan panjang dari string tersebut. Fungsi ini bekerja dengan cara mengiterasi setiap karakter pada string yang diberikan dan menambahkan 1 pada variabel length setiap kali karakter yang diiterasi bukan karakter null.

Fungsi kedua yang ada pada program ini adalah fungsi delString. Fungsi ini bertujuan untuk menghapus karakter pertama dari sebuah string. Fungsi ini menerima sebuah parameter berupa string dan akan mengembalikan sebuah string yang merupakan string yang sama dengan string yang diberikan, namun karakter pertamanya dihapus. Fungsi ini bekerja dengan cara

mengiterasi setiap karakter pada string yang diberikan, dimulai dari karakter kedua, dan menambahkan setiap karakter yang diiterasi pada sebuah string kosong.

Fungsi ketiga yang ada pada program ini adalah fungsi findSubstring. Fungsi ini merupakan fungsi utama pada program ini. Fungsi ini bertujuan untuk mencari substring pada sebuah string. Fungsi ini menerima dua parameter, yaitu string nama dan string keyword. Fungsi ini bekerja dengan cara mengiterasi setiap karakter pada string nama dan menyimpan setiap substring yang terbentuk pada sebuah array slices. Jika panjang dari substring yang terbentuk sama dengan panjang dari string keyword dan substring tersebut tidak sama dengan string keyword, maka substring tersebut akan dihapus karakter pertamanya menggunakan fungsi delString dan kemudian disimpan pada array slices. Setelah selesai melakukan iterasi pada setiap karakter pada string nama, fungsi ini akan mencetak setiap substring yang disimpan pada array slices.

Fungsi terakhir yang ada pada program ini adalah fungsi main. Fungsi ini merupakan fungsi utama yang akan memanggil fungsi findSubstring. Pada fungsi ini, terdapat dua variabel yaitu name dan keyword. Variabel name berisi sebuah string yang akan dicari substring-nya, sedangkan variabel keyword berisi sebuah string yang akan digunakan sebagai acuan untuk mencari substring pada string name. Setelah itu, fungsi findSubstring dipanggil dengan dua parameter yaitu name dan keyword.

Output ini menunjukkan bahwa program berhasil menemukan dua substring pada string name yang tidak sama dengan string keyword. Substring yang ditemukan adalah rdi dan dia.

```

PutPlayer
[0/0] video2_string.mp4

main.cpp
1 #include <iostream>
2
3 using namespace std;
4
5 int findString(string str1, string str2) {
6     int length1 = str1.length();
7     int length2 = str2.length();
8
9     if (length2 > length1) return -1;
10
11     for (int i = 0; i < length1 - length2 + 1; i++) {
12         if (str1[i] == str2[0]) {
13             int j;
14             for (j = 1; j < length2; j++) {
15                 if (str1[i + j] != str2[j]) break;
16             }
17             if (j == length2) return i;
18         }
19     }
20     return -1;
21 }
22
23 int findSubstring(string name, string keyword) {
24     int lastIndex = 0;
25     int keywordLength = keyword.length();
26     int nameLength = name.length();
27
28     while (lastIndex <= nameLength - keywordLength) {
29         if (name[lastIndex] == keyword[0]) {
30             int i;
31             for (i = 1; i < keywordLength; i++) {
32                 if (name[lastIndex + i] != keyword[i]) break;
33             }
34             if (i == keywordLength) {
35                 cout << "Keyword " << keyword << " ditemukan pada indeks " << lastIndex << endl;
36                 lastIndex += keywordLength;
37             }
38         }
39         lastIndex++;
40     }
41     return lastIndex;
42 }
43
44 int main() {
45     string name = "Andriyay";
46     string keyword = "dia";
47
48     cout << findSubstring(name, keyword);
49 }

```

Gambar 21 mencari substring pada sebuah string dengan menampilkan letak indeksnya. (Sumber: Penulis)

Program di atas merupakan program yang ditulis dalam bahasa pemrograman C++. Program ini bertujuan untuk mencari substring pada sebuah string. Substring yang dicari adalah substring yang tidak sama dengan keyword yang diberikan. Program ini terdiri dari beberapa fungsi dan satu fungsi utama yang akan memanggil fungsi-fungsi tersebut.

Fungsi pertama yang ada pada program ini adalah fungsi `getStringLength`. Fungsi ini bertujuan untuk menghitung panjang sebuah string. Fungsi ini menerima sebuah parameter berupa string dan akan mengembalikan sebuah bilangan bulat yang merupakan panjang dari string tersebut. Fungsi ini bekerja dengan cara mengiterasi setiap karakter pada string yang diberikan dan menambahkan 1 pada variabel `length` setiap kali karakter yang diiterasi bukan karakter null.

Fungsi kedua yang ada pada program ini adalah fungsi `delString`. Fungsi ini bertujuan untuk menghapus karakter pertama dari sebuah string. Fungsi ini menerima sebuah parameter berupa string dan akan mengembalikan sebuah string yang merupakan string yang sama dengan string yang diberikan, namun karakter pertamanya dihapus. Fungsi ini bekerja dengan cara mengiterasi setiap karakter pada string yang diberikan, dimulai dari karakter kedua, dan menambahkan setiap karakter yang diiterasi pada sebuah string kosong.

Fungsi ketiga yang ada pada program ini adalah fungsi `findSubstring`. Fungsi ini merupakan fungsi utama pada program ini. Fungsi ini bertujuan untuk mencari substring pada sebuah string. Fungsi ini menerima dua parameter, yaitu string `nama` dan string `keyword`. Fungsi ini bekerja dengan cara mengiterasi setiap karakter pada string `nama` dan menyimpan setiap substring yang terbentuk pada sebuah array `slices`. Jika panjang dari substring yang terbentuk sama dengan panjang dari string `keyword` dan substring tersebut tidak sama dengan string `keyword`, maka substring tersebut akan dihapus karakter pertamanya menggunakan fungsi `delString` dan kemudian disimpan pada array `slices`. Setelah selesai melakukan iterasi pada setiap karakter pada string `nama`, fungsi ini akan mencetak setiap substring yang disimpan pada array `slices`.

Fungsi terakhir yang ada pada program ini adalah fungsi `main`. Fungsi ini merupakan fungsi utama yang akan memanggil fungsi `findSubstring`. Pada fungsi ini, terdapat dua variabel yaitu `name` dan `keyword`. Variabel `name` berisi sebuah string yang akan dicari substring-nya, sedangkan variabel `keyword` berisi sebuah string yang akan digunakan sebagai acuan untuk mencari substring pada string `name`. Setelah itu, fungsi `findSubstring` dipanggil dengan dua parameter yaitu `name` dan `keyword`.

Output ini menunjukkan bahwa program berhasil menemukan substring pada string `name` yang sama dengan string `keyword`. Substring yang ditemukan adalah `dian` dan ditemukan pada indeks 2 hingga 5 dari string `name`.

BAGIAN III POST TEST

1. Ketiklah program dari algoritma saat pretest dan jalankan.

Berikut adalah algoritma untuk menghitung banyaknya elemen dalam linked list berdasarkan jawaban pretest sebelumnya:

- 1) Mulai program.
- 2) Definisikan struktur Node yang berisi data integer dan pointer ke Node berikutnya.
- 3) Buat fungsi `countNodes` yang menerima pointer ke Node pertama (head) dari linked list.
 1. Inisialisasi variabel `count` dengan 0. Ini akan digunakan untuk menghitung jumlah node.
 2. Buat pointer `current` dan atur ke `head`.
 3. Selama `current` tidak NULL, lakukan langkah berikut:
 - Tambahkan 1 ke `count`.
 - Pindahkan `current` ke node berikutnya (`current = current->next`).
 4. Setelah loop selesai, kembalikan `count` sebagai hasil fungsi.
- 4) Dalam fungsi `main`, minta pengguna untuk memasukkan jumlah node yang diinginkan dan simpan dalam variabel `n`.
- 5) Buat linked list dengan `n` node.
 1. Buat Node baru dan atur sebagai `head`.
 2. Buat pointer `current` dan atur ke `head`.
 3. Untuk setiap nilai `i` dari 1 hingga `n - 1`, lakukan langkah berikut:
 - Atur `data` dari `current` ke `i`.
 - Buat Node baru dan atur sebagai `next` dari `current`.
 - Pindahkan `current` ke node berikutnya.
 4. Atur `data` dari `current` ke `n` dan `next` ke NULL.
- 6) Panggil fungsi `countNodes` dengan `head` sebagai argumen dan simpan hasilnya dalam variabel `count`.
- 7) Cetak `count` ke layar.
- 8) Akhiri program.

Berikut adalah bentuk algoritma dalam bentuk pseudocode sebelum diterapkan ke dalam aplikasi.

```

1  DECLARE
2      STRUCT Node {
3          INTEGER data
4          Node POINTER next
5      }
6      Node POINTER head, current
7      INTEGER n, count, i
8
9  ALGORITHM
10     FUNCTION countNodes(head: POINTER TO Node) RETURNS INTEGER
11         count <- 0
12         current <- head
13         WHILE current IS NOT NULL DO
14             count <- count + 1
15             current <- current->next
16         END WHILE
17         RETURN count
18     END FUNCTION
19
20     OUTPUT "Enter the number of nodes: "
21     INPUT n
22
23     head <- NEW Node
24     current <- head
25     FOR i FROM 1 TO n-1 DO
26         current->data <- i
27         current->next <- NEW Node
28         current <- current->next
29     END FOR
30     current->data <- n
31     current->next <- NULL
32
33     count <- CALL countNodes(head)
34     OUTPUT "The number of nodes in the linked list is: ", count

```

Gambar 22 Pseudocode untuk menghitung banyaknya elemen dalam linked list. (Sumber: Penulis)

Pseudocode tersebut merupakan implementasi dari linked list, yaitu struktur data yang terdiri dari kumpulan node yang saling terhubung satu sama lain. Setiap node memiliki data dan pointer ke node berikutnya.

Pseudocode tersebut terdiri dari beberapa bagian. Pertama, dilakukan pendeklarasian struct Node yang memiliki dua variabel, yaitu data bertipe integer dan next bertipe pointer ke Node. Kemudian, dilakukan pendeklarasian variabel head dan current bertipe pointer ke Node, serta variabel n, count, dan i bertipe integer.

Selanjutnya, dibuat sebuah fungsi bernama countNodes yang menerima parameter head bertipe pointer ke Node dan mengembalikan jumlah node dalam linked list. Fungsi ini akan menghitung jumlah node dalam linked list dengan melakukan iterasi pada setiap node dan menambahkan variabel count sebanyak satu setiap kali iterasi.

Setelah itu, program akan meminta input dari pengguna berupa jumlah node yang ingin dibuat. Kemudian, program akan membuat linked list dengan membuat node-node baru sebanyak n, kemudian mengisi data pada setiap node dengan nilai i dan mengatur pointer next pada setiap node ke node berikutnya.

Setelah linked list selesai dibuat, program akan menghitung jumlah node dalam linked list dengan memanggil fungsi countNodes. Terakhir, program akan menampilkan jumlah node dalam linked list.

Implementasi pseudocode tersebut dalam bahasa pemrograman C++ menggunakan dynamic memory allocation untuk membuat linked list. Pertama, dilakukan pendeklarasian struct Node yang sama seperti pada pseudocode. Kemudian, dilakukan pendeklarasian variabel n dan count integer.

Selanjutnya, program akan meminta input dari pengguna berupa jumlah node yang ingin dibuat. Kemudian, program akan membuat linked list dengan membuat node-node baru sebanyak n menggunakan operator new, kemudian mengisi data pada setiap node dengan nilai i dan mengatur pointer next pada setiap node ke node berikutnya.

Setelah linked list selesai dibuat, program akan menghitung jumlah node dalam linked list dengan memanggil fungsi countNodes. Terakhir, program akan menampilkan jumlah node dalam linked list menggunakan cout.

Berikut adalah implementasi algoritma dan pseudocode tersebut ke dalam program C++.



```

1 #include <iostream>
2 using namespace std;
3
4 // Node struct
5 struct Node {
6     int data;
7     Node* next;
8 };
9
10 // Function to count the number of nodes in the linked list
11 int countNodes(Node* head) {
12     int count = 0;
13     Node* current = head;
14     while (current != NULL) {
15         count++;
16         current = current->next;
17     }
18     return count;
19 }
20
21 int main() {
22     int n;
23     cout << "Enter the number of nodes: ";
24     cin >> n;
25
26     // Create a linked list with n nodes
27     Node* head = new Node;
28     Node* current = head;
29     for (int i = 1; i < n; i++) {
30         current->data = i;
31         current->next = new Node;
32         current = current->next;
33     }
34     current->data = n;
35     current->next = NULL;
36
37     // Call the countNodes function and print the result
38     int count = countNodes(head);
39     cout << "The number of nodes in the linked list is: " << count << endl;
40
41     return 0;
42 }
43

```

Gambar 23 Implementasi algoritma ke kodingan C++. (Sumber: Penulis)

Kode di atas merupakan implementasi dari linked list pada bahasa pemrograman C++. Linked list adalah salah satu struktur data yang digunakan untuk menyimpan kumpulan data

yang terdiri dari beberapa node yang saling terhubung. Setiap node pada linked list memiliki dua bagian, yaitu data dan pointer yang menunjuk ke node berikutnya.

Pada kode di atas, terdapat struct Node yang memiliki dua variabel, yaitu data bertipe integer dan next bertipe pointer ke Node. Variabel data digunakan untuk menyimpan data pada setiap node, sedangkan variabel next digunakan untuk menunjuk ke node berikutnya pada linked list.

Selanjutnya, terdapat fungsi countNodes yang digunakan untuk menghitung jumlah node pada linked list. Fungsi ini menerima parameter head yang merupakan pointer ke node pertama pada linked list. Fungsi ini menggunakan variabel count untuk menghitung jumlah node dan variabel current untuk menunjuk ke node saat ini. Fungsi ini akan terus berjalan selama current tidak NULL, dan setiap kali fungsi berjalan, variabel count akan ditambah 1 dan variabel current akan diubah menjadi node berikutnya.

Pada fungsi main, terdapat input dari user berupa jumlah node yang ingin dibuat pada linked list. Selanjutnya, terdapat pembuatan linked list dengan n node menggunakan perulangan for. Pertama-tama, dibuat node pertama pada linked list dengan menggunakan pointer head. Selanjutnya, dilakukan perulangan sebanyak n-1 kali untuk membuat node-node berikutnya. Pada setiap iterasi, variabel current akan diisi dengan data i dan next akan diisi dengan pointer ke node baru yang dibuat menggunakan operator new. Setelah perulangan selesai, node terakhir akan diisi dengan data n dan next akan diisi dengan NULL.

Terakhir, dilakukan pemanggilan fungsi countNodes dengan parameter head dan hasilnya disimpan pada variabel count. Hasil tersebut kemudian dicetak pada layar menggunakan cout.

Berikut adalah penjelasan per bagian kode

```
● ● ●
4 // Node struct
5 struct Node {
6     int data;
7     Node* next;
8 };
9
```

Gambar 24 Bagian pertama. (Sumber: Penulis)

Bagian kode tersebut merupakan definisi dari sebuah struct bernama Node yang digunakan untuk merepresentasikan setiap node pada linked list. Struct Node memiliki dua variabel, yaitu data dan next.

Variabel data bertipe integer dan digunakan untuk menyimpan data pada setiap node pada linked list. Sedangkan variabel next bertipe pointer ke Node dan digunakan untuk menunjuk ke node berikutnya pada linked list.

Linked list adalah salah satu struktur data yang digunakan untuk menyimpan kumpulan data yang terdiri dari beberapa node yang saling terhubung. Setiap node pada linked list memiliki dua bagian, yaitu data dan pointer yang menunjuk ke node berikutnya.

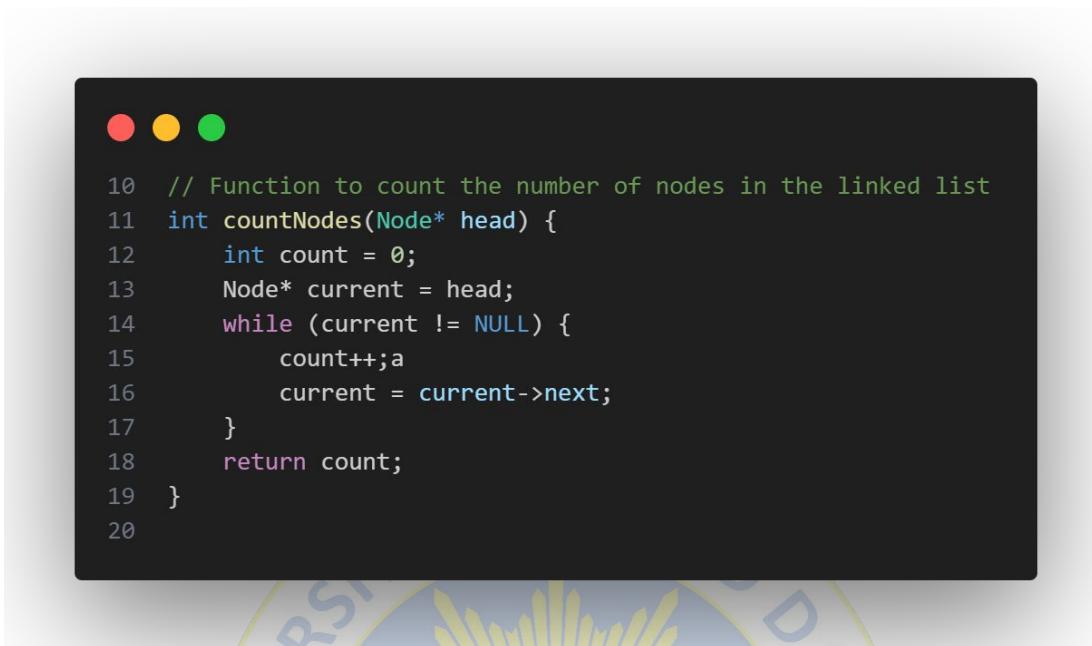
Dalam implementasi linked list, setiap node pada linked list direpresentasikan oleh sebuah struct yang memiliki dua variabel, yaitu data dan next. Variabel data digunakan untuk menyimpan data pada setiap node, sedangkan variabel next digunakan untuk menunjuk ke node berikutnya pada linked list.

Pada linked list, node pertama disebut sebagai head, sedangkan node terakhir disebut sebagai tail. Head adalah node pertama pada linked list dan tidak memiliki node sebelumnya, sehingga variabel next pada head akan menunjuk ke node berikutnya pada linked list. Sedangkan tail adalah node terakhir pada linked list dan tidak memiliki node berikutnya, sehingga variabel next pada tail akan bernilai NULL.

Linked list memiliki beberapa kelebihan dibandingkan dengan array. Pertama, linked list memungkinkan penambahan dan penghapusan elemen dengan mudah tanpa harus memindahkan elemen-elemen lain pada linked list. Kedua, linked list memungkinkan penggunaan memori yang lebih efisien karena setiap node pada linked list hanya memerlukan memori yang dibutuhkan untuk menyimpan data dan pointer ke node berikutnya.

Namun, linked list juga memiliki beberapa kelemahan. Pertama, linked list tidak mendukung akses acak ke elemen pada linked list seperti pada array. Kedua, linked list memerlukan alokasi memori dinamis untuk setiap node pada linked list, sehingga memerlukan waktu eksekusi yang lebih lama dan memerlukan manajemen memori yang lebih kompleks.

Pada kode di atas, struct Node digunakan untuk merepresentasikan setiap node pada linked list yang digunakan untuk menghitung jumlah node pada linked list. Variabel data pada struct Node digunakan untuk menyimpan data pada setiap node, sedangkan variabel next digunakan untuk menunjuk ke node berikutnya pada linked list.



```
10 // Function to count the number of nodes in the linked list
11 int countNodes(Node* head) {
12     int count = 0;
13     Node* current = head;
14     while (current != NULL) {
15         count++;
16         current = current->next;
17     }
18     return count;
19 }
20
```

Gambar 25 Bagian kedua. (Sumber: Penulis)

Kode di atas merupakan implementasi dari fungsi `countNodes` yang digunakan untuk menghitung jumlah node pada linked list. Fungsi ini menerima parameter `head` yang merupakan pointer ke node pertama pada linked list.

Pertama-tama, fungsi ini mendeklarasikan variabel `count` dengan nilai awal 0. Variabel `count` akan digunakan untuk menyimpan jumlah node pada linked list. Selanjutnya, fungsi ini mendeklarasikan variabel `current` dengan nilai awal `head`. Variabel `current` akan digunakan untuk menunjuk ke node saat ini pada linked list.

Selanjutnya, fungsi ini menggunakan perulangan `while` untuk menghitung jumlah node pada linked list. Perulangan akan terus berjalan selama variabel `current` tidak `NULL`. Pada setiap iterasi, variabel `count` akan ditambah 1 dan variabel `current` akan diubah menjadi node berikutnya pada linked list dengan menggunakan variabel `next`.

Setelah perulangan selesai, fungsi ini akan mengembalikan nilai variabel `count` yang merupakan jumlah node pada linked list.

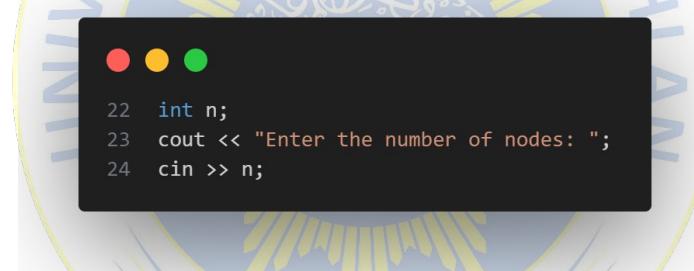
Fungsi `countNodes` sangat penting dalam penggunaan linked list karena seringkali kita perlu mengetahui jumlah node pada linked list. Dalam implementasi linked list, setiap node pada linked list direpresentasikan oleh sebuah struct yang memiliki dua variabel, yaitu `data` dan `next`. Variabel `data` digunakan untuk menyimpan data pada setiap node, sedangkan variabel `next` digunakan untuk menunjuk ke node berikutnya pada linked list.

Linked list adalah salah satu struktur data yang digunakan untuk menyimpan kumpulan data yang terdiri dari beberapa node yang saling terhubung. Setiap node pada linked list memiliki dua bagian, yaitu data dan pointer yang menunjuk ke node berikutnya.

Linked list memiliki beberapa kelebihan dibandingkan dengan array. Pertama, linked list memungkinkan penambahan dan penghapusan elemen dengan mudah tanpa harus memindahkan elemen-elemen lain pada linked list. Kedua, linked list memungkinkan penggunaan memori yang lebih efisien karena setiap node pada linked list hanya memerlukan memori yang dibutuhkan untuk menyimpan data dan pointer ke node berikutnya.

Namun, linked list juga memiliki beberapa kelemahan. Pertama, linked list tidak mendukung akses acak ke elemen pada linked list seperti pada array. Kedua, linked list memerlukan alokasi memori dinamis untuk setiap node pada linked list, sehingga memerlukan waktu eksekusi yang lebih lama dan memerlukan manajemen memori yang lebih kompleks.

Pada kode di atas, fungsi countNodes digunakan untuk menghitung jumlah node pada linked list yang dibuat pada fungsi main. Fungsi ini menerima parameter head yang merupakan pointer ke node pertama pada linked list. Setelah fungsi countNodes dipanggil pada fungsi main, hasilnya akan dicetak pada layar menggunakan cout.



```
22 int n;
23 cout << "Enter the number of nodes: ";
24 cin >> n;
```

Gambar 26 Bagian ketiga. (Sumber: Penulis)

Kode di atas merupakan implementasi untuk meminta input dari pengguna berupa jumlah node yang ingin dibuat pada linked list. Pertama-tama, kode ini mendeklarasikan variabel n dengan tipe data integer. Variabel n akan digunakan untuk menyimpan jumlah node yang dimasukkan oleh pengguna.

Selanjutnya, kode ini menggunakan fungsi cout untuk menampilkan pesan "Enter the number of nodes: " pada layar. Pesan ini akan meminta pengguna untuk memasukkan jumlah node yang ingin dibuat pada linked list.

Setelah pesan ditampilkan, kode ini menggunakan fungsi cin untuk membaca input dari pengguna dan menyimpannya ke dalam variabel n. Fungsi cin akan membaca input dari pengguna sampai pengguna menekan tombol enter.

Setelah variabel n diisi dengan input dari pengguna, variabel n akan digunakan pada kode selanjutnya untuk membuat linked list dengan jumlah node yang sesuai dengan input dari pengguna.

Input dari pengguna sangat penting dalam penggunaan linked list karena jumlah node pada linked list harus disesuaikan dengan kebutuhan pengguna. Jumlah node pada linked list akan mempengaruhi kinerja dan penggunaan memori pada program.

```
26 // Create a linked list with n nodes
27 Node* head = new Node;
28 Node* current = head;
29 for (int i = 1; i < n; i++) {
30     current->data = i;
31     current->next = new Node;
32     current = current->next;
33 }
34 current->data = n;
35 current->next = NULL;
```

Gambar 27 Bagian keempat. (Sumber: Penulis)

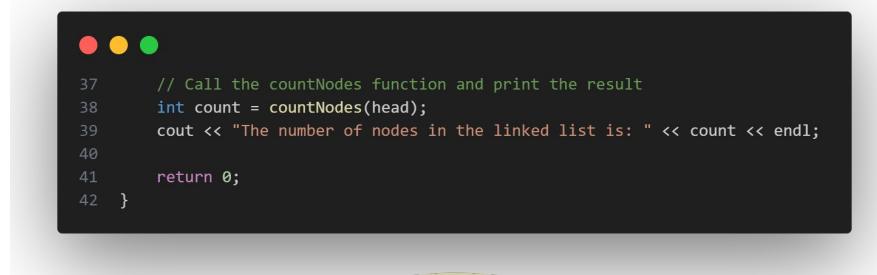
Kode di atas merupakan implementasi untuk membuat linked list dengan n node. Pertama-tama, kode ini mendeklarasikan pointer head dengan tipe data Node dan mengalokasikan memori untuk node pertama pada linked list menggunakan operator new. Pointer head akan digunakan untuk menunjuk ke node pertama pada linked list.

Selanjutnya, kode ini mendeklarasikan pointer current dengan tipe data Node dan menginisialisasi nilai current dengan head. Pointer current akan digunakan untuk menunjuk ke node saat ini pada linked list.

Selanjutnya, kode ini menggunakan perulangan for untuk membuat n-1 node pada linked list. Perulangan akan terus berjalan selama variabel i kurang dari n. Pada setiap iterasi, variabel data pada node yang ditunjuk oleh pointer current akan diisi dengan nilai i, dan variabel next pada node tersebut akan diisi dengan pointer ke node baru yang dialokasikan menggunakan operator new. Pointer current kemudian diubah menjadi pointer ke node baru tersebut.

Setelah perulangan selesai, node terakhir pada linked list akan diisi dengan nilai n dan variabel next pada node tersebut akan diisi dengan NULL. Dengan demikian, linked list dengan n node telah berhasil dibuat.

Pembuatan linked list sangat penting dalam penggunaan linked list karena linked list harus dibuat terlebih dahulu sebelum dapat digunakan. Linked list harus dibuat dengan jumlah node yang sesuai dengan kebutuhan pengguna agar dapat digunakan secara efektif.



```
37 // Call the countNodes function and print the result
38 int count = countNodes(head);
39 cout << "The number of nodes in the linked list is: " << count << endl;
40
41 return 0;
42 }
```

Gambar 28 Bagian kelima. (Sumber: Penulis)

Kode di atas merupakan implementasi untuk memanggil fungsi countNodes dan mencetak hasilnya pada layar. Pertama-tama, kode ini mendeklarasikan variabel count dengan tipe data integer dan memanggil fungsi countNodes dengan parameter head. Hasil dari fungsi countNodes akan disimpan pada variabel count.

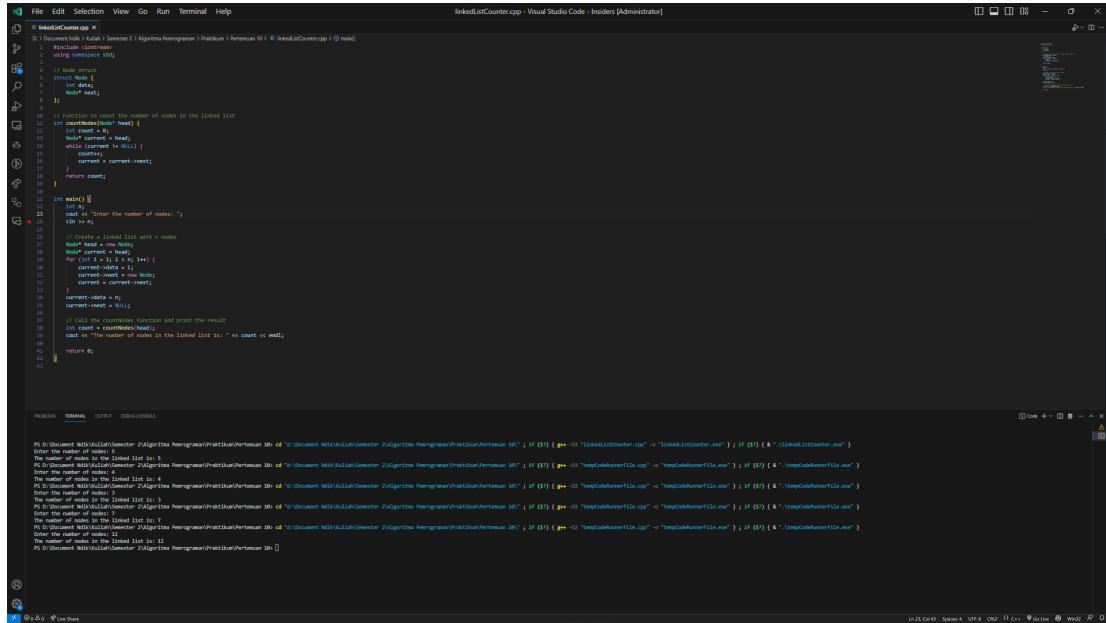
Selanjutnya, kode ini menggunakan fungsi cout untuk mencetak pesan "The number of nodes in the linked list is: " pada layar, diikuti dengan nilai variabel count. Pesan ini akan memberikan informasi kepada pengguna mengenai jumlah node pada linked list.

Setelah pesan ditampilkan, kode ini menggunakan fungsi endl untuk memindahkan kurSOR ke baris baru pada layar. Hal ini dilakukan agar pesan selanjutnya akan ditampilkan pada baris baru.

Terakhir, kode ini mengembalikan nilai 0 dari fungsi main. Hal ini menandakan bahwa program telah selesai dieksekusi dengan sukses.

Pemanggilan fungsi countNodes dan pencetakan hasilnya sangat penting dalam penggunaan linked list karena pengguna sering kali perlu mengetahui jumlah node pada linked list. Dengan mengetahui jumlah node pada linked list, pengguna dapat melakukan operasi pada linked list dengan lebih efektif.

Berikut adalah contoh outputnya



```

#include <iostream>
using namespace std;

// Node structure
class Node {
public:
    int data;
    Node* next;
};

// Function to count the number of nodes in the linked list
int countNodes(Node* current) {
    int count = 0;
    while (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}

int main() {
    cout << "Enter the number of nodes: ";
    cin >> n;
    cout << endl;
    cout << "Enter the linked list with " << n << " nodes" << endl;
    Node* head = new Node();
    head->data = 1;
    for (int i = 1; i < n; i++) {
        Node* current = head;
        current->next = new Node();
        current = current->next;
        current->data = i + 1;
    }
    cout << endl;
    cout << "The number of nodes in the linked list is: " << countNodes(head);
    cout << endl;
    cout << "The number of nodes in the linked list is: " << count;
    cout << endl;
    return 0;
}

```

Gambar 29 Output program menghitung linked list. (Sumber: Penulis)

Untuk mengakses kodingan, dapat melihat link berikut:

<https://github.com/IRedDragonICY/Programming-Algorithms>