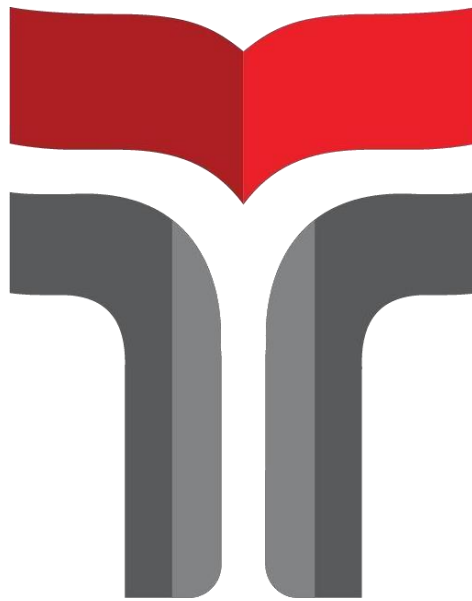


LAPORAN PRAKTIKUM
STRUKTUR DATA 4
"SINGLE LINKED LIST"



Oleh:

NAMA: Ammar Dzaki Nandana

NIM: 2311104071

KELAS: SE 07 B

DOSEN: Wahyu Andi Saputra

PRODI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2023/2024

I. TUJUAN

- Memahami konsep dasar Single Linked List.
- Mengimplementasikan operasi dasar pada Single Linked List, yaitu:
- Menambah node di depan dan belakang.
- Menghapus node berdasarkan nilai.
- Mencari node dengan nilai tertentu.
- Menghitung panjang linked list.
- Mempelajari cara mencetak isi linked list.

II. DASAR TEORI

A. Single Linked List

Single Linked List adalah salah satu jenis struktur data yang terdiri dari kumpulan elemen yang disebut node. Setiap node menyimpan dua bagian: data dan pointer (atau referensi) yang menunjuk ke node berikutnya dalam urutan. Struktur data ini bersifat dinamis, yang berarti ukuran linked list dapat bertambah atau berkurang sesuai dengan kebutuhan, tanpa memerlukan alokasi memori statis seperti array.

B. Struktur Node

Node adalah unit dasar dari linked list. Setiap node biasanya terdiri dari:

1. Data: Informasi atau nilai yang disimpan dalam node.
2. Pointer: Referensi ke node berikutnya. Node terakhir dalam linked list akan memiliki pointer yang menunjuk ke null, menandakan akhir dari list.

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

C. Operasi Dasar pada Single Linked List

1. Insert Node di Depan:

- A. Menambah node baru di awal linked list.
- B. Proses ini melibatkan pembuatan node baru, mengatur pointer next node baru ke node kepala yang ada, dan memperbarui kepala linked list ke node baru.

2. Insert Node di Belakang:

- A. Menambah node baru di akhir linked list.
- B. Proses ini melibatkan penelusuran linked list hingga menemukan node terakhir, kemudian mengatur pointer next node terakhir untuk menunjuk ke node baru.

3. Delete Node:

- A. Menghapus node tertentu berdasarkan nilai yang diberikan.
- B. Proses ini melibatkan penelusuran linked list untuk menemukan node yang ingin dihapus dan mengatur pointer dari node sebelumnya untuk melewati node yang dihapus.

4. Cari Node dengan Nilai Tertentu:

- A. Mencari apakah suatu nilai ada dalam linked list.
- B. Proses ini melibatkan penelusuran dari node kepala hingga akhir linked list, membandingkan nilai setiap node dengan nilai yang dicari.

Hitung Panjang Linked List:

- A. Menghitung jumlah total node dalam linked list.
- B. Proses ini melibatkan penelusuran dari node kepala hingga akhir linked list, menghitung setiap node yang dilalui.

D. Kelebihan dan Kekurangan Single Linked List

Kelebihan:

- Dinamis: Ukuran linked list dapat berubah secara dinamis tanpa memerlukan alokasi memori statis.
- Efisien dalam Penambahan/Penghapusan: Operasi penambahan dan penghapusan dapat dilakukan dengan efisien tanpa perlu memindahkan elemen lain, berbeda dengan array.

Kekurangan:

- Akses Elemen yang Lambat: Akses langsung ke elemen tertentu lambat, karena harus menelusuri dari kepala hingga node yang dicari.
- Memori Tambahan: Setiap node memerlukan ruang memori tambahan untuk pointer.

III. GUIDED

I.

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 // Deklarasi Struct untuk mahasiswa
6 struct mahasiswa {
7     char nama[50];
8     char nim[10];
9 };
10
11 // Deklarasi Struct Node
12 struct Node {
13     mahasiswa data;
14     Node *next;
15 };
16
17 Node *head;
18 Node *tail;
19
20 // Inisialisasi list
21 void init() {
22     head = nullptr;
23     tail = nullptr;
24 }
25
26 // Pengisian apakah list kosong
27 bool isEmpty() {
28     return head == nullptr;
29 }
30
31 // Tambah Depan
32 void insertDepan(const mahasiswa &data) {
33     Node *baru = new Node;
34     baru->data = data;
35     baru->next = nullptr;
36     if (isEmpty()) {
37         head = tail = baru;
38     } else {
39         baru->next = head;
40         head = baru;
41     }
42 }
43
44 // Tambah Belakang
45 void insertBelakang(const mahasiswa &data) {
46     Node *baru = new Node;
47     baru->data = data;
48     baru->next = nullptr;
49     if (isEmpty()) {
50         head = tail = baru;
51     } else {
52         tail->next = baru;
53         tail = baru;
54     }
55 }
56
57 // Hitung Jumlah List
58 int hitungList() {
59     Node *current = head;
60     int jumlah = 0;
61     while (current != nullptr) {
62         jumlah++;
63         current = current->next;
64     }
65     return jumlah;
66 }
67
68 // Hapus Depan
69 void hapusDepan() {
70     if (isEmpty()) {
71         Node *hapus = head;
72         head = head->next;
73         delete hapus;
74         if (head == nullptr) {
75             tail = nullptr; // Jika list menjadi kosong
76         }
77     } else {
78         cout << "List kosong!" << endl;
79     }
80 }
81
82 // Hapus Belakang
83 void hapusBelakang() {
84     if (isEmpty()) {
85         if (head == tail) {
86             delete head;
87             head = tail = nullptr; // List menjadi kosong
88         } else {
89             Node *bantu = head;
90             while (bantu->next != tail) {
91                 bantu = bantu->next;
92             }
93             delete tail;
94             tail = bantu;
95             bantu->next = nullptr;
96         }
97     } else {
98         cout << "List kosong!" << endl;
99     }
100 }
101
102 // Tampilkan List
103 void tampil() {
104     Node *current = head;
105     if (isEmpty()) {
106         while (current != nullptr) {
107             cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
108             current = current->next;
109         }
110     } else {
111         cout << "List masih kosong!" << endl;
112     }
113 }
114
115 // Hapus List
116 void clearList() {
117     Node *current = head;
118     while (current != nullptr) {
119         Node *hapus = current;
120         current = current->next;
121         delete hapus;
122     }
123     head = tail = nullptr;
124     cout << "List berhasil terhapus!" << endl;
125 }
126
127 // Main Function
128 int main() {
129     init();
130
131     // Contoh data mahasiswa
132     mahasiswa m1 = {"Alice", "123456"};
133     mahasiswa m2 = {"Bob", "654321"};
134     mahasiswa m3 = {"Charlie", "112233"};
135
136     // Menambahkan mahasiswa ke dalam list
137     insertDepan(m1);
138     tampil();
139     insertBelakang(m2);
140     tampil();
141     insertDepan(m3);
142     tampil();
143
144     // Menghapus elemen dari list
145     hapusDepan();
146     tampil();
147     hapusBelakang();
148     tampil();
149
150     // Menghapus seluruh list
151     clearList();
152
153     return 0;
154 }
```

II.

```
1 #include <iostream>
2 using namespace std;
3
4 // Definisi struktur untuk elemen list
5 struct Node {
6     int data;        // Menyimpan nilai elemen
7     Node* next;      // Pointer ke elemen berikutnya
8 };
9
10 // Fungsi untuk mengalokasikan memori untuk node baru
11 Node* alokasi(int value) {
12     Node* newNode = new Node; // Alokasi memori untuk elemen baru
13     if (newNode != nullptr) { // Jika alokasi berhasil
14         newNode->data = value; // Mengisi data node
15         newNode->next = nullptr; // Set next ke nullptr
16     }
17     return newNode; // Mengembalikan pointer node baru
18 }
19
20 // Fungsi untuk dealokasi memori node
21 void dealokasi(Node* node) {
22     delete node; // Mengembalikan memori yang digunakan oleh node
23 }
24
25 // Pengecekan apakah list kosong
26 bool isEmpty(Node* head) {
27     return head == nullptr; // List kosong jika head adalah nullptr
28 }
29
30 // Menambahkan elemen di awal list
31 void insertFirst(Node* &head, int value) {
32     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
33     if (newNode != nullptr) {
34         newNode->next = head; // Menghubungkan elemen baru ke elemen pertama
35         head = newNode;      // Menetapkan elemen baru sebagai elemen pertama
36     }
37 }
38
39 // Menambahkan elemen di akhir list
40 void insertLast(Node* &head, int value) {
41     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
42     if (newNode != nullptr) {
43         if (isEmpty(head)) { // Jika list kosong
44             head = newNode; // Elemen baru menjadi elemen pertama
45         } else {
46             Node* temp = head;
47             while (temp->next != nullptr) { // Mencari elemen terakhir
48                 temp = temp->next;
49             }
50             temp->next = newNode; // Menambahkan elemen baru di akhir list
51         }
52     }
53 }
54
55 // Menampilkan semua elemen dalam list
56 void printList(Node* head) {
57     if (isEmpty(head)) {
58         cout << "List kosong!" << endl;
59     } else {
60         Node* temp = head;
61         while (temp != nullptr) { // Selama belum mencapai akhir list
62             cout << temp->data << " "; // Menampilkan data elemen
63             temp = temp->next; // Melanjutkan ke elemen berikutnya
64         }
65         cout << endl;
66     }
67 }
68
69 // Menghitung jumlah elemen dalam list
70 int countElements(Node* head) {
71     int count = 0;
72     Node* temp = head;
73     while (temp != nullptr) {
74         count++; // Menambah jumlah elemen
75         temp = temp->next; // Melanjutkan ke elemen berikutnya
76     }
77     return count; // Mengembalikan jumlah elemen
78 }
79
80 // Menghapus semua elemen dalam list dan dealokasi memori
81 void clearList(Node* &head) {
82     while (head != nullptr) {
83         Node* temp = head; // Simpan pointer ke node saat ini
84         head = head->next; // Pindahkan ke node berikutnya
85         dealokasi(temp); // Dealokasi node
86     }
87 }
88
89 int main() {
90     Node* head = nullptr; // Membuat list kosong
91
92     // Menambahkan elemen ke dalam list
93     insertFirst(head, 10); // Menambahkan elemen 10 di awal list
94     insertLast(head, 20);  // Menambahkan elemen 20 di akhir list
95     insertLast(head, 30);  // Menambahkan elemen 30 di akhir list
96
97     // Menampilkan isi list
98     cout << "Isi List: ";
99     printList(head);
100
101     // Menampilkan jumlah elemen
102     cout << "Jumlah elemen: " << countElements(head) << endl;
103
104     // Menghapus semua elemen dalam list
105     clearList(head);
106
107     // Menampilkan isi list setelah penghapusan
108     cout << "Isi List setelah penghapusan: ";
109     printList(head);
110
111     return 0;
112 }
```

IV. UNGUIDED

1.

```
1  #include <iostream>
2
3  using namespace std;
4
5  struct Node {
6      int data;
7      Node* next;
8
9      Node(int nilai) {
10         data = nilai;
11         next = nullptr;
12     }
13 };
14
15 class SingleLinkedList {
16 private:
17     Node* head;
18
19 public:
20     SingleLinkedList() {
21         head = nullptr;
22     }
23
24     void tambahNodeDiDepan(int nilai) {
25         Node* nodeBaru = new Node(nilai);
26         nodeBaru->next = head;
27         head = nodeBaru;
28     }
29
30     void tambahNodeDiBelakang(int nilai) {
31         Node* nodeBaru = new Node(nilai);
32         if (head == nullptr) {
33             head = nodeBaru;
34             return;
35         }
36         Node* temp = head;
37         while (temp->next != nullptr) {
38             temp = temp->next;
39         }
40         temp->next = nodeBaru;
41     }
42
43     void cetakList() {
44         Node* temp = head;
45         while (temp != nullptr) {
46             cout << temp->data;
47             if (temp->next != nullptr) {
48                 cout << " -> ";
49             }
50             temp = temp->next;
51         }
52         cout << endl;
53     }
54
55     ~SingleLinkedList() {
56         Node* current = head;
57         Node* nextNode;
58         while (current != nullptr) {
59             nextNode = current->next;
60             delete current;
61             current = nextNode;
62         }
63     }
64 };
65
66 int main() {
67     SingleLinkedList linkedList;
68
69     linkedList.tambahNodeDiDepan(10);
70     linkedList.tambahNodeDiBelakang(20);
71     linkedList.tambahNodeDiDepan(5);
72     cout << "Linked List setelah penambahan: ";
73     linkedList.cetakList();
74
75     return 0;
76 }
```

2.

```
1  #include <iostream>
2
3  using namespace std;
4
5  struct Node {
6      int data;
7      Node* next;
8
9      Node(int nilai) {
10         data = nilai;
11         next = nullptr;
12     }
13 };
14
15 class SingleLinkedList {
16 private:
17     Node* head;
18
19 public:
20     SingleLinkedList() {
21         head = nullptr;
22     }
23
24     void tambahNodeDiDepan(int nilai) {
25         Node* nodeBaru = new Node(nilai);
26         nodeBaru->next = head;
27         head = nodeBaru;
28     }
29
30     void tambahNodeDiBelakang(int nilai) {
31         Node* nodeBaru = new Node(nilai);
32         if (head == nullptr) {
33             head = nodeBaru;
34             return;
35         }
36         Node* temp = head;
37         while (temp->next != nullptr) {
38             temp = temp->next;
39         }
40         temp->next = nodeBaru;
41     }
42
43     void hapusNode(int nilai) {
44         if (head == nullptr) {
45             return;
46         }
47         if (head->data == nilai) {
48             Node* temp = head;
49             head = head->next;
50             delete temp;
51             return;
52         }
53         Node* current = head;
54         Node* previous = nullptr;
55         while (current != nullptr && current->data != nilai) {
56             previous = current;
57             current = current->next;
58         }
59         if (current != nullptr) {
60             previous->next = current->next;
61             delete current;
62         }
63     }
64
65     void cetakList() {
66         Node* temp = head;
67         while (temp != nullptr) {
68             cout << temp->data;
69             if (temp->next != nullptr) {
70                 cout << " -> ";
71             }
72             temp = temp->next;
73         }
74         cout << endl;
75     }
76     ~SingleLinkedList() {
77         Node* current = head;
78         Node* nextNode;
79         while (current != nullptr) {
80             nextNode = current->next;
81             delete current;
82             current = nextNode;
83         }
84     }
85 };
86
87 int main() {
88     SingleLinkedList linkedList;
89
90     linkedList.tambahNodeDiDepan(10);
91     linkedList.tambahNodeDiBelakang(20);
92     linkedList.tambahNodeDiDepan(5);
93
94     cout << "Linked List sebelum penghapusan: ";
95     linkedList.cetakList();
96
97     linkedList.hapusNode(10);
98
99     cout << "Linked List setelah penghapusan nilai 10: ";
100    linkedList.cetakList();
101
102    return 0;
103 }
104
```

3.

```
1 #include <iostream>
2
3 using namespace std;
4
5 struct Node {
6     int data;
7     Node* next;
8
9     Node(int nilai) {
10         data = nilai;
11         next = nullptr;
12     }
13 };
14
15 class SingleLinkedList {
16 private:
17     Node* head;
18
19 public:
20     SingleLinkedList() {
21         head = nullptr;
22     }
23
24     void tambahNodeDiDepan(int nilai) {
25         Node* nodeBaru = new Node(nilai);
26         nodeBaru->next = head;
27         head = nodeBaru;
28     }
29
30     void tambahNodeDiBelakang(int nilai) {
31         Node* nodeBaru = new Node(nilai);
32         if (head == nullptr) {
33             head = nodeBaru;
34             return;
35         }
36         Node* temp = head;
37         while (temp->next != nullptr) {
38             temp = temp->next;
39         }
40         temp->next = nodeBaru;
41     }
42
43     bool cariNode(int nilai) {
44         Node* temp = head;
45         while (temp != nullptr) {
46             if (temp->data == nilai) {
47                 return true;
48             }
49             temp = temp->next;
50         }
51         return false;
52     }
53
54     int hitungPanjang() {
55         int panjang = 0;
56         Node* temp = head;
57         while (temp != nullptr) {
58             panjang++;
59             temp = temp->next;
60         }
61         return panjang;
62     }
63
64     void cetakList() {
65         Node* temp = head;
66         while (temp != nullptr) {
67             cout << temp->data;
68             if (temp->next != nullptr) {
69                 cout << " -> ";
70             }
71             temp = temp->next;
72         }
73         cout << endl;
74     }
75
76     ~SingleLinkedList() {
77         Node* current = head;
78         Node* nextNode;
79         while (current != nullptr) {
80             nextNode = current->next;
81             delete current;
82             current = nextNode;
83         }
84     }
85 };
86
87 int main() {
88     SingleLinkedList linkedList;
89
90     linkedList.tambahNodeDiDepan(10);
91     linkedList.tambahNodeDiBelakang(20);
92     linkedList.tambahNodeDiDepan(5);
93
94     cout << "Linked List: ";
95     linkedList.cetakList();
96
97     int nilaiDicari = 20;
98     if (linkedList.cariNode(nilaiDicari)) {
99         cout << "Node dengan nilai " << nilaiDicari << " ditemukan." << endl;
100     } else {
101         cout << "Node dengan nilai " << nilaiDicari << " tidak ditemukan." << endl;
102     }
103
104     int panjang = linkedList.hitungPanjang();
105     cout << "Panjang linked list: " << panjang << endl;
106
107     return 0;
108 }
109
```


V. KESIMPULAN

Praktikum ini berhasil menjelaskan dan mengimplementasikan konsep dasar Single Linked List dalam bahasa C++. Melalui berbagai operasi dasar seperti penambahan node di depan dan belakang, penghapusan node berdasarkan nilai, pencarian node tertentu, dan penghitungan panjang linked list, peserta praktikum dapat memahami struktur dan dinamika dari linked list. Implementasi ini menunjukkan bahwa linked list adalah struktur data yang fleksibel dan efisien, terutama dalam pengelolaan data yang sering berubah. Meskipun memiliki kelebihan dalam hal dinamisitas dan efisiensi operasi penambahan serta penghapusan, linked list juga memiliki kekurangan seperti akses elemen yang lambat dan kebutuhan memori tambahan untuk pointer. Dengan demikian, praktik ini memberikan wawasan penting mengenai cara kerja dan aplikasi Single Linked List dalam pemrograman.

VI. UNGUIDED

NO. 1

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      float bil1, bil2;
7
8      // Input dua buah bilangan dari user
9      cout << "Masukkan bilangan pertama: ";
10     cin >> bil1;
11     cout << "Masukkan bilangan kedua: ";
12     cin >> bil2;
13
14     // Tampilkan hasil penjumlahan
15     cout << "Hasil penjumlahan: " << bil1 + bil2 << endl;
16
17     // Tampilkan hasil pengurangan
18     cout << "Hasil pengurangan: " << bil1 - bil2 << endl;
19
20     // Tampilkan hasil perkalian
21     cout << "Hasil perkalian: " << bil1 * bil2 << endl;
22
23     // Tampilkan hasil pembagian
24     // Pengecekan untuk menghindari pembagian dengan nol
25     if (bil2 != 0) {
26         cout << "Hasil pembagian: " << bil1 / bil2 << endl;
27     } else {
28         cout << "Pembagian tidak bisa dilakukan karena bilangan kedua adalah nol." << endl;
29     }
30
31     return 0;
32 }
33
34
35 //2311104071_AMMAR DZAKI NANDANA
36
```

NO. 2

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 // Fungsi untuk mengubah angka menjadi teks
7 string angkaKeTeks(int angka)
8 {
9     string satuan[] = {"", "satu", "dua", "tiga", "empat", "lima", "enam", "tujuh", "delapan", "sembilan"};
10    string belasan[] = {"sepuluh", "sebelas", "dua belas", "tiga belas", "empat belas", "lima belas", "enam belas", "tujuh belas", "delapan belas", "sembilan belas"};
11    string puluhan[] = {"", "dua puluh", "tiga puluh", "empat puluh", "lima puluh", "enam puluh", "tujuh puluh", "delapan puluh", "sembilan puluh"};
12
13    // Jika angka adalah 0
14    if (angka == 0)
15    {
16        return "nol";
17    }
18
19    // Jika angka adalah 100
20    if (angka == 100)
21    {
22        return "seratus";
23    }
24
25    // Jika angka di bawah 10
26    if (angka < 10)
27    {
28        return satuan[angka];
29    }
30
31    // Jika angka di antara 10 dan 19 (belasan)
32    if (angka >= 10 && angka < 20)
33    {
34        return belasan[angka - 10];
35    }
36
37    // Jika angka di antara 20 dan 99 (puluhan)
38    if (angka >= 20 && angka < 100)
39    {
40        return puluhan[angka / 10] + (angka % 10 != 0 ? " " + satuan[angka % 10] : "");
41    }
42
43    return "";
44 }
45
46 int main()
47 {
48     int angka;
49
50     // Input dari pengguna
51     cout << "Masukkan angka antara 0 sampai 100: ";
52     cin >> angka;
53
54     // Memastikan input valid
55     if (angka < 0 || angka > 100)
56     {
57         cout << "Masukkan angka yang valid (0 sampai 100)." << endl;
58     }
59     else
60     {
61         // Output hasil dalam bentuk teks
62         cout << angka << ": " << angkaKeTeks(angka) << endl;
63     }
64
65     return 0;
66 }
67 //2311104071_AMMAR DZAKI NANDANA
```

NO. 3

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int n;
7
8     // Input dari pengguna
9     cout << "Masukkan angka: ";
10    cin >> n;
11
12    // Loop untuk mencetak pola
13    for (int i = n; i >= 1; i--) {
14        // Bagian kiri: menurun dari i hingga 1
15        for (int j = i; j >= 1; j--) {
16            cout << j << " ";
17        }
18
19        // Cetak tanda bintang *
20        cout << " ";
21
22        // Bagian kanan: menaik dari 1 hingga i
23        for (int j = 1; j <= i; j++) {
24            cout << j << " ";
25        }
26
27        // Baris baru untuk pola berikutnya
28        cout << endl;
29    }
30
31    // Output terakhir hanya tanda *
32    cout << "*" << endl;
33
34    return 0;
35 }
36 #include <iostream>
37 #include <string>
38
39 using namespace std;
40
41 // Fungsi untuk mengubah angka menjadi teks
42 string angkaKeTeks(int angka)
43 {
44     string satuan[] = {"", "satu", "dua", "tiga", "empat", "lima", "enam", "tujuh", "delapan", "sembilan"};
45     string belasan[] = {"sepuluh", "sebelas", "dua belas", "tiga belas", "empat belas", "lima belas", "enam belas", "tujuh belas", "delapan belas", "sembilan belas"};
46     string puluhan[] = {"", "", "dua puluh", "tiga puluh", "empat puluh", "lima puluh", "enam puluh", "tujuh puluh", "delapan puluh", "sembilan puluh"};
47
48     // Jika angka adalah 0
49     if (angka == 0)
50     {
51         return "nol";
52     }
53
54     // Jika angka adalah 100
55     if (angka == 100)
56     {
57         return "seratus";
58     }
59
60     // Jika angka di bawah 10
61     if (angka < 10)
62     {
63         return satuan[angka];
64     }
65
66     // Jika angka di antara 10 dan 19 (belasan)
67     if (angka >= 10 && angka < 20)
68     {
69         return belasan[angka - 10];
70     }
71
72     // Jika angka di antara 20 dan 99 (puluhan)
73     if (angka >= 20 && angka < 100)
74     {
75         return puluhan[angka / 10] + (angka % 10 != 0 ? " " + satuan[angka % 10] : "");
76     }
77
78     return "";
79 }
80
81 int main()
82 {
83     int angka;
84
85     // Input dari pengguna
86     cout << "Masukkan angka antara 0 sampai 100: ";
87     cin >> angka;
88
89     // Memastikan input valid
90     if (angka < 0 || angka > 100)
91     {
92         cout << "Masukkan angka yang valid (0 sampai 100)." << endl;
93     }
94     else
95     {
96         // Output hasil dalam bentuk teks
97         cout << angka << " : " << angkaKeTeks(angka) << endl;
98     }
99
100    return 0;
101 }
102 //2311104071_AMMAR DZAKI NANDANA
```

VII. KESIMPULAN

Penginstalan Code::Blocks beserta kompiler MinGW (untuk pengguna Windows) telah berhasil dilakukan. IDE ini siap digunakan untuk pengembangan program dalam bahasa C atau C++. Dengan menggunakan Code::Blocks, proses pengembangan menjadi lebih mudah karena tersedianya fitur debugging dan kompilasi otomatis.