LAPORAN PRAKTIKUM
PERTEMUAN 5
SINGLE LINKED LIST SEARCHING

Nama :

Martryatus Sofia 2311104003

Dosen :

Yudha Islami Sulistya, S.Kom., M.Cs

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO 2024

SOAL TP
Soal 1: Mencari Elemen Tertentu dalam SLL

Deskripsi Soal: Buatlah program yang mengizinkan pengguna memasukkan 6 elemen integer ke dalam list. Implementasikan function **searchElement** untuk mencari apakah sebuah nilai tertentu ada dalam list.

Instruksi

1. Minta pengguna untuk memasukan nilai yang ingin dicari.
2. Jika nilai ditemukan, tampilan alamat dan posisi dalam angka (contoh: urutan ke 4) pada list tersebut.
3. Jika nilai tidak ditemukan, tampilkan pesan bahwa elemen tersebut tidak ada dalam list tersebut.

NB:

1. Gunakan pendekatan linier search untuk mencari elemen.

Sub-Program:

```
Function searchElement( L : list, i : integer)
{ I.S. List tidak kosong.
F.S. Menampilkan alamat dan posisi elemen i jika ditemukan}
Dictionary
        current: address
        position: int
Algorithms
        current ← L.head
        position ← 1

        //melakukan perulangan selama i belum ditemukan dan posisi current belum berada pada
        akhir list
        While ......
                //seiring pointer (current) bergerak, position bertambah
                . . . . .
                //lakukan perpindahan current
                . . . . .
        endwhile
        //jika i ditemukan maka tampilkan alamat dan posisi
        if....
                output(...)
        //jika tidak ditemukan maka tampilkan pesan yang menyatakan hal tsb
        else...
                output(...)
        endif
endfunction
```

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

struct Node {
    int info;
    Node* next;
};

struct List {
    Node* head;
};

void createList(List &L) {
    L.head = nullptr;
}

void insertLast(List &L, int value) {
    Node* newNode = new Node;
    newNode->info = value;
    newNode->next = nullptr;

    if (L.head == nullptr) {
        L.head = newNode;
    } else {
        Node* current = L.head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void searchElement(const List &L, int target) {
    Node* current = L.head;
    int position = 1;
    bool found = false;

    while (current != nullptr) {
        if (current->info == target) {
            cout << "Elemen " << target << " ditemukan pada alamat " << current
                 << " di urutan ke-" << position << "." << endl;
            found = true;
            break;
        }
        current = current->next;
        position++;
    }

    if (!found) {
        cout << "Elemen " << target << " tidak ditemukan dalam list." << endl;
    }
}

int main() {
    List L;
    createList(L);

    cout << "Masukkan 6 elemen integer untuk dimasukkan ke dalam list:" << endl;
    for (int i = 0; i < 6; i++) {
        int value;
        cout << "Elemen " << i + 1 << ": ";
        cin >> value;
        insertLast(L, value);
    }

    int target;
    cout << "Masukkan nilai yang ingin dicari: ";
    cin >> target;

    searchElement(L, target);

    return 0;
}
```

OUTPUT :

```
PS F:\Kuliah\Semester3\Strukdat - praktikum\05_Single_Linked_List_Searching
kum\05_Single_Linked_List_Searching\05_Single_Linked_List_Searching\Unguide
Masukkan 6 elemen integer untuk dimasukkan ke dalam list:
Elemen 1: 1
Elemen 2: 2
Elemen 3: 5
Elemen 4: 7
Elemen 5: 5
Elemen 6: 3
Masukkan nilai yang ingin dicari: 5
Elemen 5 ditemukan pada alamat 0x21194fe1820 di urutan ke-3.
```

Soal 2: Mengurutkan List Menggunakan Bubble Sort

Deskripsi Soal: Buatlah program yang mengizinkan pengguna memasukkan 5 elemen integer ke dalam list. Implementasikan procedure **bubbleSortList** untuk mengurutkan elemen-elemen dalam list dari nilai terkecil ke terbesar.

Instruksi

Setelah mengurutkan, tampilkan elemen-elemen list dalam urutan yang benar.

Langkah-langkah Bubble Sort pada SLL

1. Inisialisasi:
   - Buat pointer current yang akan digunakan untuk menelusuri list.
   - Gunakan variabel boolean swapped untuk mengawasi apakah ada pertukaran yang dilakukan pada iterasi saat ini.
2. Traversing dan Pertukaran:
   - Lakukan iterasi berulang sampai tidak ada pertukaran yang dilakukan:
     - Atur swapped ke false di awal setiap iterasi.
     - Set current ke head dari list.
     - Selama current.next tidak null (masih ada node berikutnya):
       - Bandingkan data pada node current dengan data pada node current.next.
       - Jika data pada current lebih besar dari data pada current.next, lakukan pertukaran:
         - Tukar data antara kedua node (bukan pointer).
         - Set swapped menjadi true untuk menunjukkan bahwa ada pertukaran yang dilakukan.
       - Pindahkan current ke node berikutnya (current = current.next).
3. Pengulangan:
   - Ulangi langkah 2 sampai tidak ada lagi pertukaran yang dilakukan (artinya list sudah terurut).

Contoh Proses Bubble Sort

- List awal : 4 – 2 – 3 – 1 dan akan melakukan sorting membesar / ascending
- Iterasi pertama:
  - Bandingkan 4 dan 2: 4 > 2, lakukan penukaran, 2 – 4 – 3 – 1
  - Bandingkan 4 dan 3: 4 > 3, lakukan penukaran, 2 – 3 – 4 – 1

    - Bandingkan 4 dan 1: 4 > 1, lakukan penukaran, 2 – 3 – 1 – 4
    - Kondisi list di akhir iterasi: 2 – 3 – 1 – 4
- Iterasi kedua:
    - Bandingkan 2 dan 3: 2 < 3, tidak terjadi penukaran
    - Bandingkan 3 dan 1: 3 > 1, lakukan penukaran, 2 – 1 – 3 – 4
    - Bandingkan 3 dan 4: 3 < 4, tidak terjadi penukaran
    - Kondisi list di akhir iterasi: 2 – 1 – 3 – 4
- Iterasi ketiga:
    - Bandingkan 2 dan 1: 2 > 1, lakukan penukaran, 1 – 2 – 3 – 4
    - Bandingkan 2 dan 3: 2 < 3, tidak terjadi penukaran
    - Bandingkan 3 dan 4 : 3 < 4, tidak terjadi penukaran
    - Kondisi list di akhir iterasi: 1 – 2 – 3 – 4

Sub-Program:

```
Procedure bubbleSort( in/out L : list )
{ I.S. List tidak kosong.
F.S. elemen pada list urut membesar berdasarkan infonya}
```

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

struct Node {
    int info;
    Node* next;
};

struct List {
    Node* head;
};

void createList(List &L) {
    L.head = nullptr;
}

void insertLast(List &L, int value) {
    Node* newNode = new Node;
    newNode->info = value;
    newNode->next = nullptr;

    if (L.head == nullptr) {
        L.head = newNode;
    } else {
        Node* current = L.head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void printList(const List &L) {
    Node* current = L.head;
    while (current != nullptr) {
        cout << current->info << " ";
        current = current->next;
    }
    cout << endl;
}

void bubbleSortList(List &L) {
    bool swapped;
    do {
        swapped = false;
        Node* current = L.head;

        while (current != nullptr && current->next != nullptr) {
            if (current->info > current->next->info) {
                // Tukar data antar node
                int temp = current->info;
                current->info = current->next->info;
                current->next->info = temp;
                swapped = true;
            }
            current = current->next;
        }
    } while (swapped);
}

int main() {
    List L;
    createList(L);

    cout << "Masukkan 5 elemen integer untuk dimasukkan ke dalam list:" << endl;
    for (int i = 0; i < 5; i++) {
        int value;
        cout << "Elemen " << i + 1 << ": ";
        cin >> value;
        insertLast(L, value);
    }

    cout << "\nList sebelum diurutkan: ";
    printList(L);

    bubbleSortList(L);

    cout << "List setelah diurutkan: ";
    printList(L);

    return 0;
}
```

OUTPUT

```
Masukkan 5 elemen integer untuk dimasukkan ke dalam list:
Elemen 1: 3
Elemen 2: 7
Elemen 3: 1
Elemen 4: 2
Elemen 5: 3

List sebelum diurutkan: 3 7 1 2 3
List setelah diurutkan: 1 2 3 3 7
PS F:\Kuliah\Semester3\Strukdat - praktikum\05_Single_Linked_List_Searching\05_Single_Lin
```

Soal 3: Menambahkan Elemen Secara Terurut

Deskripsi Soal: Buatlah program yang mengizinkan pengguna memasukkan 4 elemen integer ke dalam list secara manual. Kemudian, minta pengguna memasukkan elemen tambahan yang harus ditempatkan di posisi yang sesuai sehingga list tetap terurut.

Instruksi

1. Implementasikan procedure **insertSorted** untuk menambahkan elemen baru ke dalam list yang sudah terurut.
2. Tampilkan list setelah elemen baru dimasukkan.

Sub-Program:

```
Procedure insertSorted( in/out L : list, in P : address)
{ I.S. List tidak kosong.
F.S. Menambahkan elemen secara terurut}
Dictionary
        Q, Prev: address
        found: bool
Algorithms
        Q ← L.head
        found ← false

        //melakukan perulangan selama found masih false dan Q masih menunjuk elemen pada list
        While ......
                //melakukan pengecekan apakah info dari elemen yang ditunjuk memiliki nilai lebih
                kecil dari pada P
                if ....
                        //jika iya maka Prev diisi elemen Q, dan Q diisi elemen setelahnya
                        ....
                //jika tidak maka isi found dengan nilai 'true'
                else
                        . . .
                Endif
                //lakukan perpindahan Q
                ....
        endwhile


        //melakukan pengecekan apakah Q elemen head
        if ....
                //jika iya, maka tambahkan P sebagai head
                ....
        //melakukan pengecekan apakah Q berisi null (sudah tidak menunjuk elemen pada list
        else if ...
                //jika iya, maka tambahkan P sebagai elemen terakhir
                ...
        //jika tidak keduanya, maka tambahkan P pada posisi diantara Prev dan Q
        else
                ....
        endif
endprocedure
```

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

struct Node {
    int info;
    Node* next;
};

struct List {
    Node* head;
};

void createList(List &L) {
    L.head = nullptr;
}

void insertLast(List &L, int value) {
    Node* newNode = new Node;
    newNode->info = value;
    newNode->next = nullptr;

    if (L.head == nullptr) {
        L.head = newNode;
    } else {
        Node* current = L.head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void printList(const List &L) {
    Node* current = L.head;
    while (current != nullptr) {
        cout << current->info << " ";
        current = current->next;
    }
    cout << endl;
}

void insertSorted(List &L, int value) {
    Node* newNode = new Node;
    newNode->info = value;
    newNode->next = nullptr;

    if (L.head == nullptr || L.head->info >= value) {
        newNode->next = L.head;
        L.head = newNode;
    } else {
        Node* current = L.head;
        Node* prev = nullptr;

        while (current != nullptr && current->info < value) {
            prev = current;
            current = current->next;
        }

        prev->next = newNode;
        newNode->next = current;
    }
}

int main() {
    List L;
    createList(L);

    cout << "Masukkan 4 elemen integer untuk dimasukkan ke dalam list:" <<
endl;
    for (int i = 0; i < 4; i++) {
        int value;
        cout << "Elemen " << i + 1 << ": ";
        cin >> value;
        insertSorted(L, value);
    }

    cout << "\nList setelah 4 elemen dimasukkan: ";
    printList(L);

    int newValue;
    cout << "\n
Masukkan elemen tambahan yang ingin ditambahkan secara terurut: ";
    cin >> newValue;
    insertSorted(L, newValue);

    cout << "\nList setelah elemen tambahan dimasukkan: ";
    printList(L);

    return 0;
}
```

OUTPUT

```
Masukkan 4 elemen integer untuk dimasukkan ke dalam list:
Elemen 1: 2
Elemen 2: 3
Elemen 3: 1
Elemen 4: 2

List setelah 4 elemen dimasukkan: 1 2 2 3

Masukkan elemen tambahan yang ingin ditambahkan secara terurut: 5

List setelah elemen tambahan dimasukkan: 1 2 2 3 5
```