1. Pada saat program pertama kali dijalankan, nilai pointer head adalah nullptr. Jelaskan apa makna kondisi ini dalam konteks linked list dan bagaimana pengaruhnya terhadap operasi insertion dan deletion.
   Artinya linked list kosong.
   insertion
   linked list kosong -> taruh node yang mau diinsert di depan -> head = baru
   deletion
   linked list kosong -> gak bisa dong :'

2. Amati proses insert di awal dan insert di akhir linked list.
   a. Apa perbedaan perubahan nilai pointer head pada kedua operasi tersebut?
   Kalau linked list kosong ya sama aja lah antara insert di awal dan insert di akhir
   Kalau linked list ada isinya
   Insert di awal: posisi yang didepan node baru -> head pindah
   Insert di akhir: posisi yang depan tetep dia -> head gak pindah

   b. Mengapa pada insert di akhir diperlukan proses traversal, sedangkan pada insert di awal tidak?
   Karena pengen naruh di akhir. Berarti perlu nyambungin pointer next punya si node terakhir ke node yang baru. Nah cara ke node terakhir gimana? Kalau Cuma punya head berarti perlu geser satu per satu, current = current->next sampai nyampe akhir. Kalau punya tail gak perlu tuh traversal. Tinggal tail->next = newNode.

3. Saat menjalankan delete di awal, perhatikan perubahan alamat memori yang ditunjuk oleh head.
   a. Node mana yang dihapus dari memori?
   Temp alias node head di linked list awal (sebelum hapus di awal)

   b. Apa yang akan terjadi jika pointer head tidak dipindahkan terlebih dahulu sebelum node dihapus?
   Node paling depan megang node kedua dari depan. Node kedua dari depan megang node ketiga dari depan. DST. Kalau node paling depan tapi head gak dipindahin ke node kedua dari depan, lantas siapa yang megang node kedua dari depan?

   TLDR: gak bisa akses linked list sisanya

4. Pada operasi delete di akhir, perhatikan penggunaan pointer secondLast.
    a. Mengapa kita tidak bisa langsung menghapus node terakhir tanpa mencari node sebelumnya?
    Perlu ngatur node kedua dari terakhir (di linked list awal) biar nunjuk ke null pointer abis kita ngapus node terakhir. Nunjuk ke null pointer = abis dia gak ada node lagi. Kalau gak diapus, nanti dia halu ngiranya ada node lagi dan bakal bisa bikin error.

    b. Apa yang akan terjadi jika linked list hanya memiliki satu node?
    Yaa jadi linked list kosong, as simple as that.

5. Perhatikan fungsi updateData.
    a. Bagaimana proses pencarian data dilakukan pada linked list?
    Cek apakah node punya nilai yang dicari. Ketemu? Selesai. Gak ketemu? Cek node berikutnya. Gitu terus sampai ketemu

    b. Mengapa fungsi ini memiliki kompleksitas O(n)?
    Worst casenya node yang dicari itu di elemen terakhir atau gak ada. Sebenernya kalau gak ada berarti kan O(n+1) yak. Tapi O(n+1) is really just O(n)

    c. Apa yang terjadi jika data yang dicari muncul lebih dari satu kali di linked list?
    Kalau bikinnya kayak yang sekarang, dia pas ketemu return kan. Berarti kalau ada dua, Cuma ngubah yang pertama karena terlanjur return.

6. Bandingkan fungsi printLinkedList dengan fungsi updateData.
    a. Persamaan apa yang terlihat pada cara traversal linked list di kedua fungsi tersebut?
    while (current != nullptr){ current = current -> next}, artinya geser one by one

    b. Apa peran pointer sementara (current) dalam menjaga agar head tidak berubah?
    Head tetep perlu megang node pertama biar node pertama tetep ada yang megangin. Node pertama perlu dipegangin biar satu linked list bisa diakses.

7. Amati fungsi deleteLinkedList yang dijalankan sebelum program berakhir.
    a. Mengapa penghapusan seluruh node perlu dilakukan secara satu per satu?
    Kalau hapus head aja sisanya ada di memori dan gak bisa diakses

b. Apa risiko yang mungkin terjadi jika linked list tidak dihapus sebelum program selesai?
Nyepam di memori. Sistem operasi (OS) yang ngatur pembersihan memori ragu2 mau ngapus karena gak bisa tau siapa yang punya, boleh diapus atau enggak.

8. Selama proses debugging, perhatikan alamat memori dari setiap node yang dibuat.
   a. Apakah node-node tersebut tersimpan berurutan di memori?
   Harusnya enggak sih :v

   b. Apa kesimpulan kalian mengenai perbedaan linked list dan array berdasarkan pengamatan ini?
   Node dari linked list tidak disimpan secara berurut, bisa jadi karena instantiation struct node

9. Jika program ini dijalankan tanpa menu (misalnya langsung memanggil fungsi-fungsi tertentu), operasi mana yang **menurut Anda** paling aman dan paling berisiko menyebabkan error? Jelaskan alasannya berdasarkan kondisi pointer.
10. **Menurut Anda**, untuk kasus apa program linked list seperti ini lebih cocok digunakan dibandingkan array? Sebaliknya, pada kondisi apa array justru lebih efisien daripada linked list?