

HOOFDSTUK 4

GELINKTE LIJSTEN

Helga Naessens

Inhoud

- **Gelinkte lijsten: algemeen**
- Definitie en declaratie van een gelinkte lijst
- Een gelinkte lijst overlopen
- Een gelinkte lijst opbouwen
- Een gelinkte lijst vernietigen
- Voordelen/nadelen gelinkte lijst

De array: voordelen

De array is een eenvoudig type waarin je meerdere elementen van hetzelfde type kan opslaan.

- De declaratie is **eenvoudig** en **gemakkelijk uitbreidbaar** naar meerdere dimensies

Voorbeeld: `int k[5][2][3];`

- Elk element in de array is even gemakkelijk en **even snel bereikbaar** (door indices te gebruiken).
- Indien er nog plaats is, kan je een element snel **achteraan toevoegen**

De array: nadelen

- 1) je moet bij constructie de capaciteit van de array opgeven
- 2) deze capaciteit kan nadien niet meer wijzigen
(noch groeien, noch krimpen)
- 3) er is geen index-checking
- 4) een array kent haar eigen grootte niet
(grootte = aantal elementen in de array)

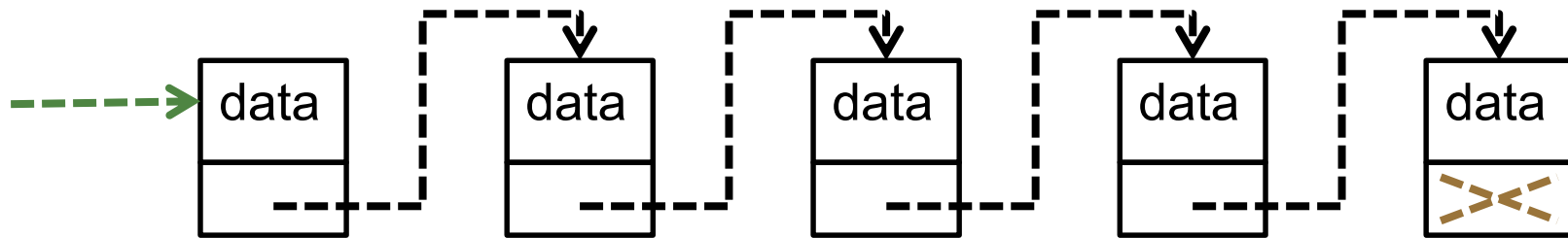
Vraag: *Hoe bereken je de capaciteit?*

- 5) vooraan of tussenin elementen toevoegen of verwijderen is een *dure* (moeilijke?) operatie

De gelinkte lijst: principe

- Een gelinkte lijst bestaat uit **knopen**,
die op willekeurige plaatsen verspreid staan in het geheugen.
- Elke knoop bevat **data** en (een) **pointer**(s) naar (een) buurkno(o)p(en).
 - ⇒ **enkelgelinkte** lijst: elke knoop bevat een wijzer naar
zijn opvolger
 - ⇒ **dubbelgelinkte** lijst: elke knoop bevat een wijzer naar
zijn opvolger en een wijzer naar zijn voorganger

De enkelgelinkte lijst

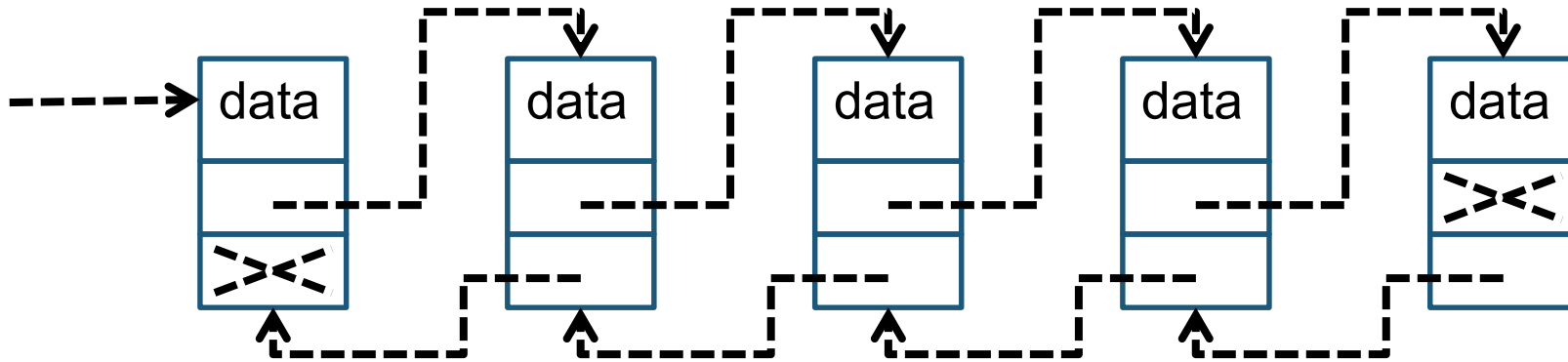


- De lijst zelf moet enkel een **wijzer naar de eerste knoop** (= **kop**) bevatten.

Laatste knoop (= **staart**) heeft geen opvolger (**nullpointer**).

- Men kan de volledige lijst overlopen door telkens naar de opvolger te springen tot men bij de laatste knoop komt.
- De i -de knoop kan men niet zomaar direct bekomen: men moet eerst alle voorgangers overlopen.

De dubbelgelinkte lijst



meer geheugen en **meer werk** (linken verleggen) bij het toevoegen van een knoop,

wissen en toevoegen vóór een knoop is **eenvoudiger** dan bij een enkelgelinkte lijst (niet alle elementen moeten overlopen worden om de voorganger te vinden)

Inhoud


- Gelinkte lijsten: algemeen
- **Definitie en declaratie van een gelinkte lijst**
- Een gelinkte lijst overlopen
- Een gelinkte lijst opbouwen
- Een gelinkte lijst vernietigen
- Voordelen/nadelen gelinkte lijst

Definitie van een gelinkte lijst

Voorbeeld:

```
typedef struct knoop knoop;
```

```
struct knoop {  
    int getal;  
    knoop *next;  
};
```



Elke knoop bestaat uit twee delen: een inhoud (een geheel getal in dit geval) en een verwijzing naar een andere knoop.

Declaratie van een gelinkte lijst

Voorbeeld:

knoop *l;

Variabele l verwijst naar
de eerste knoop van de lijst

...

```
/* eerste getal printen? */
```

```
printf("%d", l->getal);
```

```
/* derde getal printen? */
```

```
printf("%d", l->next->next->getal);
```

Let op als je l
wijzigd/verzet!!



Inhoud

- Gelinkte lijsten: algemeen
- Definitie en declaratie van een gelinkte lijst
- **Een gelinkte lijst overlopen**
- Een gelinkte lijst opbouwen
- Een gelinkte lijst vernietigen
- Voordelen/nadelen gelinkte lijst

Een gelinkte lijst overlopen

Voorbeeld:

Schrijf een procedure `print_lijst(l)` die alle getallen uit de gegeven gelinkte lijst `l` naar het scherm print.

```
void print_lijst(const knoop *l) {  
    while (l) {  
        printf("%d\n", l->getal);  
        l = l->next;  
    }  
}
```

`l++;`

is niet ok! Waarom?

`l` wordt verzet: waarom is dit ok?

Oefening: zet om naar een recursieve procedure

Oefeningen:

1. Schrijf een functie `pointer_naar_knoop(l,g)` die in de gegeven gelinkte lijst `l` op zoek gaat naar het gegeven geheel getal `g` en een pointer naar deze knoop teruggeeft (of de nullpointer indien `l` dit getal niet bevat).
2. Schrijf een procedure `verplaats_naar_knoop(l,g)` die de gegeven pointer `l` verplaatst naar de knoop met het gegeven getal `g` - indien dit getal in de gelinkte lijst te vinden is. Indien het getal niet te vinden is, wordt de gegeven pointer de nullpointer.

Inhoud

- Gelinkte lijsten: algemeen
- Definitie en declaratie van een gelinkte lijst
- Een gelinkte lijst overlopen
- **Een gelinkte lijst opbouwen**
- Een gelinkte lijst vernietigen
- Voordelen/nadelen gelinkte lijst


Een gelinkte lijst opbouwen

1. Achteraan toevoegen

Voorbeeld:

Schrijf een functie `maak_lijst()` die een reeks gehele getallen inleest (de reeks stopt met het getal 0, dat niet meer tot de reeks behoort) en al deze getallen opslaat in een gelinkte lijst (steeds achteraan toevoegen zodat de getallen **in volgorde** van inlezen opgeslagen worden). De functie geeft een pointer naar de eerste knoop van de lijst terug (of een nullpointer indien de reeks leeg is).

```
knoop* maak_lijst() {  
    knoop *l = 0, *h;  
    int g; scanf("%d",&g);  
    if (g) {  
        l = (knoop*) malloc(sizeof(knoop));  
        h = l; h->getal = g;  
        scanf("%d",&g);  
        while (g != 0) {  
            h->next = malloc(sizeof(knoop));  
            h = h->next; h->getal = g; scanf("%d",&g);  
        }  
        h->next = 0;  
    }  
    return l;  
}
```



h = h->next;
h = malloc(sizeof(knoop)); ...
is niet ok! Waarom?

Oefening:

Maak van de functie `maak_lijst()` een procedure `maak_lijst(1)`

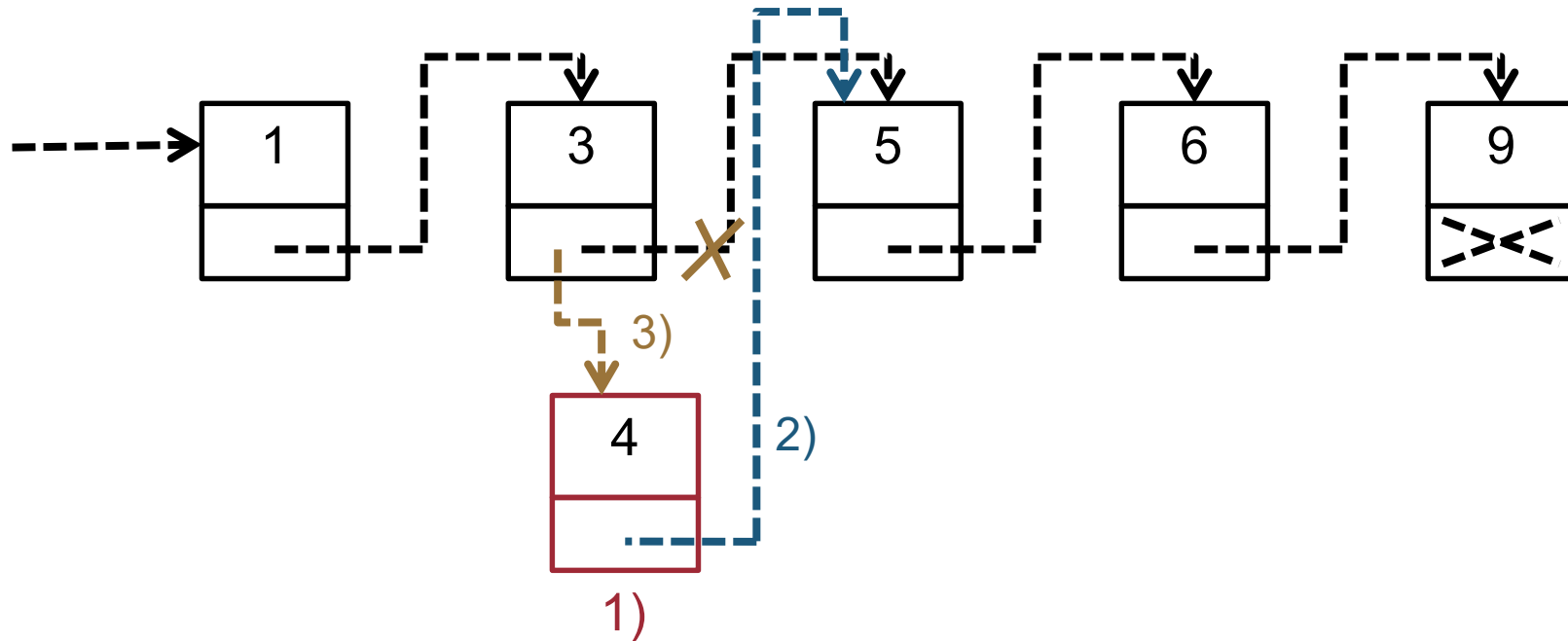
2. Vooraan toevoegen

Oefening:

Schrijf een functie `maak_omgekeerde_lijst()` die een reeks gehele getallen inleest (einde reeks = 0) en al deze getallen opslaat in een gelinkte lijst (vooraan toevoegen zodat de getallen **in omgekeerde volgorde** van inlezen opgeslagen worden). De functie geeft een pointer naar de eerste knoop van de lijst terug (of een nullpointer indien de reeks leeg is).

3. Tussenin toevoegen

- 1) Alloceer de nieuwe knoop.
- 2) Laat de opvolglink van de nieuwe knoop naar de opvolger van de bestaande knoop wijzen.
- 3) Laat de opvolglink van de bestaande knoop naar de nieuwe knoop wijzen.



Voorbeeld:

Schrijf een functie `maak_gesorteerde_lijst()` die een reeks gehele getallen inleest (einde reeks = 0) en **gesorteerd** (= van klein naar groot) opslaat in een gelinkte lijst. De functie geeft een pointer naar de eerste knoop van de lijst terug (of een nullpointer indien de reeks leeg is).

```
knoop* maak_gesorteerde_lijst() {  
    knoop *l = 0; int getal; scanf("%d",&getal);  
    while (getal) {  
        voeg_getal_toe(&l, getal);  
        scanf("%d",&getal);  
    }  
    return l;  
}
```

```
void voeg_getal_toe(knoop **p1, int g) {  
    knoop *h, *nieuw;  
    nieuw = malloc(sizeof(knoop));  
    nieuw->getal = g;  
    if (*p1 == 0 || g <= (*p1)->getal) {  
        nieuw->next = *p1; *p1 = nieuw;  
    }  
    else {  
        h = *p1;  
        while (h->next && h->next->getal < g)  
            h = h->next;  
        nieuw->next = h->next;  
        h->next = nieuw;  
    }  
}
```

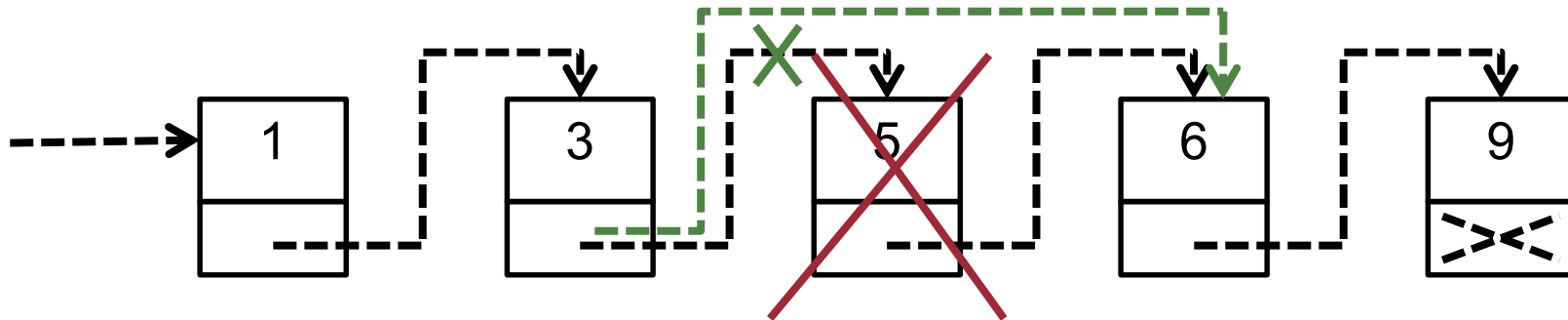
lijst_vb2.c

Inhoud

- Gelinkte lijsten: algemeen
- Definitie en declaratie van een gelinkte lijst
- Een gelinkte lijst overlopen
- Een gelinkte lijst opbouwen
- **Een gelinkte lijst vernietigen**
- Voordelen/nadelen gelinkte lijst

Een gelinkte lijst vernietigen

- Stappen die moeten ondernomen worden om een knoop te wissen:
 - 1) Laat de opvolglink van de voorganger wijzen naar de opvolger van de te verwijderen knoop.
 - 2) De-alloceer de te verwijderen knoop.



- Voorbeeld:

Schrijf een procedure `verwijder_element(l,g)` die in de gegeven gelinkte lijst `l` de eerste knoop met het gegeven geheel getal `g` verwijdert.

Merk op: Indien de gegeven gelinkte lijst het getal `g` niet bevat, wijzigt de lijst niet.

Er zijn 4 mogelijke situaties:

1. De lijst is leeg
2. Het getal bevindt zich in de eerste knoop van de gelinkte lijst
3. Het getal bevindt zich verder in de gelinkte lijst
4. Het getal komt niet voor in de gelinkte lijst

```

void verwijder_element(knoop **p1, int g) {
    knoop *m,*h;
    if (*p1) {
        if ((*p1)->getal == g) {
            m = *p1; *p1 = (*p1)->next; free(m);
        }
        else {
            h = *p1;
            while (h->next && h->next->getal != g)
                h = h->next;
            if (h->next) {
                m = h->next; h->next = m->next; free(m);
            }
        }
    }
}

```

lijst_vb3.c

- Oefening:

Schrijf een procedure `vernietig_lijst(l)` die de gegeven gelinkte lijst `l` volledig vernietigt (en dus het geheugen van alle knopen opnieuw vrijgeeft).

```
void vernietig_lijst(knoop **p1) {  
    knoop *h;  
    while (*p1) {  
        h = *p1;  
        *p1 = h->next;  
        free(h);  
    }  
}
```

Inhoud

- Gelinkte lijsten: algemeen
- Definitie en declaratie van een gelinkte lijst
- Een gelinkte lijst overlopen
- Een gelinkte lijst opbouwen
- Een gelinkte lijst vernietigen
- **Voordelen/nadelen gelinkte lijst**

gelinkte lijst: voordelen

- Vooraan of tussenin (achter een bestaande knoop) een knoop toevoegen kan snel.
- Achteraan een knoop toevoegen kan ook snel als men over een wijzer beschikt naar de laatste knoop.
- Een knoop verwijderen lukt snel als men naast de opvolger ook de voorganger van de knoop kent.

gelinkte lijst: nadelen

- De i-de knoop kan men niet direct bekomen.
Eerst moeten alle voorgangers overlopen worden.
⇒ **indexeren** ([]) is een dure operatie
- Een gelinkte lijst verbruikt meestal **iets meer geheugen** dan een tabel doordat elke knoop pointers moet bijhouden. Bovendien zorgt de verspreiding in het geheugen voor **performantieverlies**.