Notities les 5

Jelle De Bock

October 20, 2015

1 Const

- **Twee posities mogelijk**
  *const int *t; / pointer naar een constante int*
  *int const * t; / is identiek als bovenstaande, de bovenstaande wordt echter meer gebruikt int * const t; / constante pointer*
  Een pointer slaat altijd op het geen wat er voor komt, behalve wanneer het het helemaal vooraan staat, dan wat er net achter komt.

- **Kleine oefening hierop**

```
const int * const ** a;
//a is een pointer naar een pointer naar een constante
//pointer naar een constante int
//A->[]->[]=>[const int]
a++; //oke
(*a)++; //oke
(**a)++; //neen, want constant
(***a)++; //neen, want constante int
```

- **Const bij arrays**
  Niet te ver gaan met const bij pointers. Je moet er enkel zeker van zijn dat de values niet gewijzigd worden.

2 Structs

Tracht bij structs zo veel mogelijk met pointers te werken, want structs kunnen enorm groot zijn.

3 Stack en heap

Bij malloc plaats je iets op de heap. Bij stack wordt deze leeggemaakt op einde van de methode.

4 Linked lists

Nadeel van een array, is dat alles achter elkaar moet zitten.
Invoegen in een array is een tijdsintensieve taak.

4.1 Structuur van de linked list

Einde van de lijst wordt aangegeven door null pointer, tenzij het een circulaire lijst is.
Dubbelgelinkte lijst, heeft ook pointer naar zijn vorige knoop.

4.2 Een implementatiesuggestie

```
typedef struct knoop knoop;
//dit doe je zodat je niet elke keer "struct"
//hiervoor moet zetten (1x bij implementatie)
struct knoop{
    int getal;
    knoop * next;
}
```

4.3 Toevoegen aan een gelinkte lijst

Enkele situaties

- **Achteraan toevoegen** (zie *knoop.c* file)

4.4 Wijzigen

Om iets te wijzigen moet je met een pointer naar een pointer werken.