PROG3 – Practicum Week2

Datum: 04-09-2013 Data structures: ADT Linked List, Queue
Versie: 1.0
Auteur: Sander van Heumen

1. Inleiding

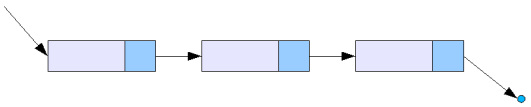
Deze week staan **datastructuren** op het programma. Deze datastructuren komen veelal in de praktijk voor, al dan niet in herkenbare vorm. Het is daarom belangrijk hier voldoende kennis van te bezitten. Voor het maken van de opdrachten is kennis van pointers en structures vereist. Een goed document voor het opfrissen van deze kennis kun je vinden op http://cslibrary.stanford.edu/102/, of overeenkomstig document op Blackboard genaamd */programmeren 3/week1/**PointersAndMemory.pdf***. Alles wat in dit document staat wordt als bekend beschouwd en is onderdeel van de tentamenstof. De nodige documentatie en hulpmiddelen voor deze week bestaan uit:

- Hoorcollege + presentatie
- http://cslibrary.stanford.edu/103/LinkedListBasics.pdf
  (of Blackboard: /programmeren 3/week2/**LinkedListBasics.pdf**)

2. Linked List

In deze opgaven ga je een Linked List(LL) maken. Als je niet weet wat een LL is dan dien je de documentatie genoemd onder punt 1 te raadplegen. Hieronder vind je een stuk code van een header file (genaamd **LL.h**) waarin de datatypes waaruit de LL bestaat zijn gedefinieerd.

```
/* stukje header file LL.h */
#ifndef LL_H
#define LL_H

struct Node {
    int waarde;
    struct Node *volgende;
};

typedef struct Node Knoop;

struct List {
    Knoop *kop;
    int aantal;
};

typedef struct List LL;

// functie prototypes

#endif
```



Figuur 1 - Linked List Structure

Je dient onderstaande functies te schrijven die een LL implementeren. De functies horend bij bovenstaande datatypes vormen samen de LL. De functies **manipuleren** en **creëren** de LL. Kortom zonder deze functies ook geen LL!

- `constructLL` (lijst initialiseren)
- `destructLL` (dynamisch geallokeerd geheugen opruimen)
- `printLL` (lijst op het scherm afdrukken)
- `countLL` (aantal elementen bepalen)
- `addRearLL` (element aan de achterkant van de lijst toevoegen) - **Activity Diagram tekenen**
- `addFrontLL` (element aan de voorkant van de lijst toevoegen) - **Activity Diagram tekenen**
- `deleteLL` (element verwijderen uit de lijst (kies uit: begin of einde van de lijst)
- `copyLL` (kopiëren (***deep-copy***) van een lijst naar een andere lijst)

Om je niet helemaal in het diepe te gooien is hieronder de implementatie van de functie `addFrontLL()` weergegeven. Deze hoef je dus niet meer te maken. Kijk goed naar de argumenten en het return-type.

```
/* stukje LL.cpp */

#include <stdio.h>
#include <stdlib.h>
#include “LL.h”

void addFrontLL (LL *lijst, int getal)
{
        Knoop* nieuw;

        nieuw = (Knoop*)(malloc(sizeof(Knoop)));
        nieuw->waarde   = getal;
        nieuw->volgende = lijst->kop;
        lijst->kop      = nieuw;
        lijst->aantal++;
}
```

Uiteindelijk zal je mij moeten laten zien hoe een LL werkt, en dien je aan te tonen dat de functies die je hebt geschreven correct werken. Hieronder is een voorbeeld programma weergegeven die aantoont dat de LL code die je hebt geschreven correct werkt.

```
/* stukje main.cpp */

#include <stdio.h>
#include <stdlib.h>
#include “LL.h”

int main(void)
{

    LL lijst1, lijst2;// aanmaken van 2 lijsten (ongeïnitialiseerd)

    constructLL (&lijst1);
    constructLL (&lijst2);
    printLL (&lijst1);
    addFrontLL (&lijst1, 3);
    addFrontLL (&lijst1, 5);
    addFrontLL (&lijst1, 2);
    addFrontLL (&lijst1, 18);
    addFrontLL (&lijst1, 12);
    printLL (&lijst1);
    printf("aantal elementen: %d\n", countLL(&lijst1));
    addRearLL (&lijst1,4);
    addRearLL (&lijst1,10);
    printLL (&lijst1);
    printf("aantal elementen: %d\n", countLL(&lijst1));
    copyLL (&lijst2, &lijst1);//dest, src
    printLL (&lijst2);
    printf("aantal elementen: %d\n", countLL(&lijst2));
    destructLL (&lijst1);
    destructLL (&lijst2);

    return 0;
}
```

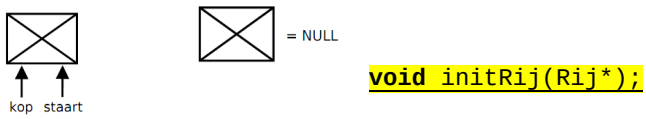
3. Queue

1. Leg in eigen woorden uit wat we verstaan onder een **queue**? Noem de belangrijkste eigenschappen.
2. In deze opgaven moet je een queue gaan implementeren. Probeer eerst de opgaven, zonder gebruik te maken van internet, te maken. Kijk goed naar de plaatjes, en teken deze nog eens na om de werking van een queue goed te begrijpen. Hieronder zijn de nieuwe **datatypes** (dus geen variabelen!) die we nodig hebben om de queue te maken weergegeven:

```
struct Node {
    struct Node* next;
    int data;
};

typedef struct Node Node;
```

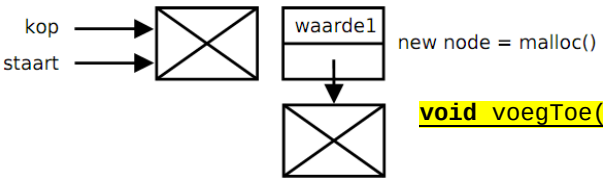
```
struct Rij {
    struct Node* kop;
    struct Node* staart;
};

typedef struct Rij Rij;
```

De queue (rij) bestaat dus uit 2 pointers (genaamd `kop` en `staart`). Eentje die wijst naar het begin, eentje die wijst naar het einde van de rij. In eerste instantie is de queue **leeg**. De pointers wijzen dan beide naar **NULL**. Zie Figuur 2 hieronder.

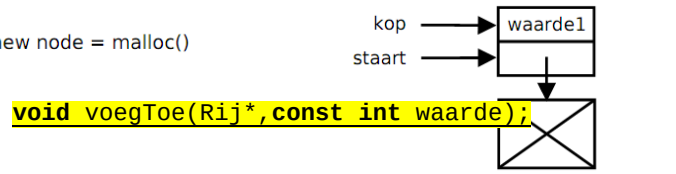


Figuur 2 – lege queue

Je wilt nu een waarde (=nieuw item) aan de queue toevoegen. Je dien t dan eerst geheugen aan te maken voor het nieuwe item. Hierna dien je de lijst-pointers(kop en staart) nog netjes te *updaten*. Zie Figuur 3 + Figuur 4.

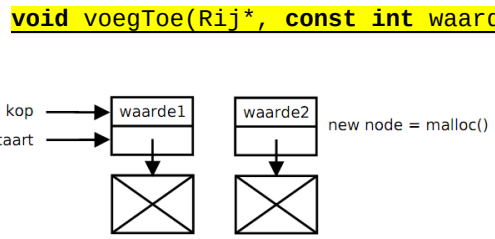


Figuur 3 - aanmaken geheugen nieuw item (=node)

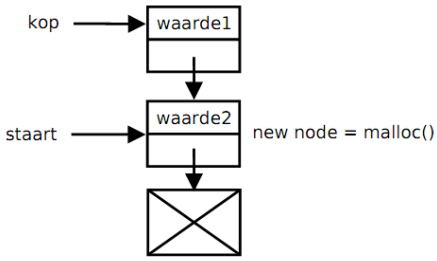


Figuur 4 - nieuw item toevoegen aan lijst

Om het geheel nog duidelijker te maken gaan we aan deze lijst gaan we nog een item toevoegen. Zie Figuur 5 + Figuur 6.

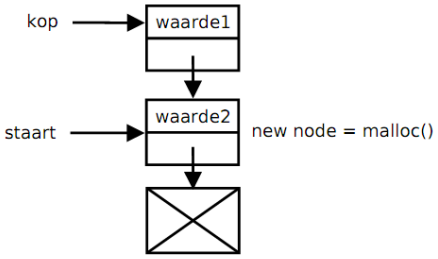


Figuur 5 - aanmaken geheugen nieuw item



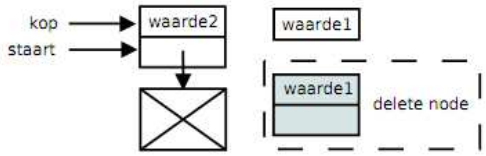
Figuur 6 - nieuw item toevoegen aan lijst

Figuur 6 geeft de eindsituatie weer wanneer er 2 items zijn toegevoegd aan de queue. Let hier op de positie van de kop en staart! We willen nu een waarde (waarde1) van de rij afhalen. Zie Figuur 7 + Figuur 8. Zoals je kunt zien werkt de queue volgens een FIFO structuur: First-In-First-Out.



Figuur 7 - Twee items in lijst vlak voor verwijderen

```
void haalAf(Rij*, int* waarde);
```



Figuur 8 - 1 item verwijderd

a. Implementeer nu de volgende functies, kijk goed naar Figuur 2 t/m Figuur 8.

Functie-prototype	Beschijving functie-definitie
`void initRij (Rij*);`	Initialiseren van de rij.
`int voegToe (Rij*, const int waarde);`	Voegt een nieuwe waarde toe aan het einde van de `Rij`. **Let op!** Deze functie dient ook geheugen aan te maken voor *deze* nieuwe waarde. Deze functie dient een 0 terug te geven als alles gelukt is, een 1 indien er een fout optreed.
`int haalAf(Rij*, int* waarde);`	Haalt de **oudste** waarde, indien beschikbaar, van de `Rij` af. Het geheugen van deze waarde dien je ook op te ruimen. De waarde zelf dien je m.b.v. een argument terug te geven aan de '**caller**' van de functie. De functie (**callee**) dient een 0 terug te geven als alles gelukt is, een 1 indien er een fout is opgetreden of wanneer de `Rij` leeg is. Indien de `Rij` leeg is dient de waarde op '-1' te worden gezet.
`bool isLeeg(Rij*);`	Kijkt of de `Rij` leeg is. Wanneer deze leeg is dient de functie **true** te retourneren, anderzijds **false**.
`bool aantal(Rij*, int* aantal);`	'Retourneert' d.m.v. het argument `aantal` het aantal elementen in de queue. Wanneer alles goed is gegaan moet de functie **true** terug geven, anderzijds **false**. Wanneer de queue leeg blijkt te zijn, moet `aantal` op -1 worden gezet.

b. Schrijf nu een **volledig programma** waarmee de werking van de queue wordt aangetoond.