Examen Software Ontwikkeling I
2e Bachelor Informatica
Faculteit Wetenschappen
Academiejaar 2010-2011
21 januari, 2011

BELANGRIJK

1. Lees eerst de volledige opgave (inclusief de hints/opmerkingen)!
2. **In dit deel komt enkel C++ aan bod (geen C)**
3. Kladpapier vind je achteraan
4. Veel succes!

Deel I. Open Boek gedeelte op PC (8u30 - 11u00)

Programmeren in C++

[op 12 punten van de 20]

Efficiënte zoekfunctie in een boek

Men wenst alle woorden, die in een boek voorkomen, op te slaan in een datastructuur, zodat men gemakkelijk een woord kan opzoeken. Van elk woord wordt de bladzijde, de lijn en de positie binnen die lijn bijgehouden. Een woord kan uiteraard verschillende keren voorkomen.

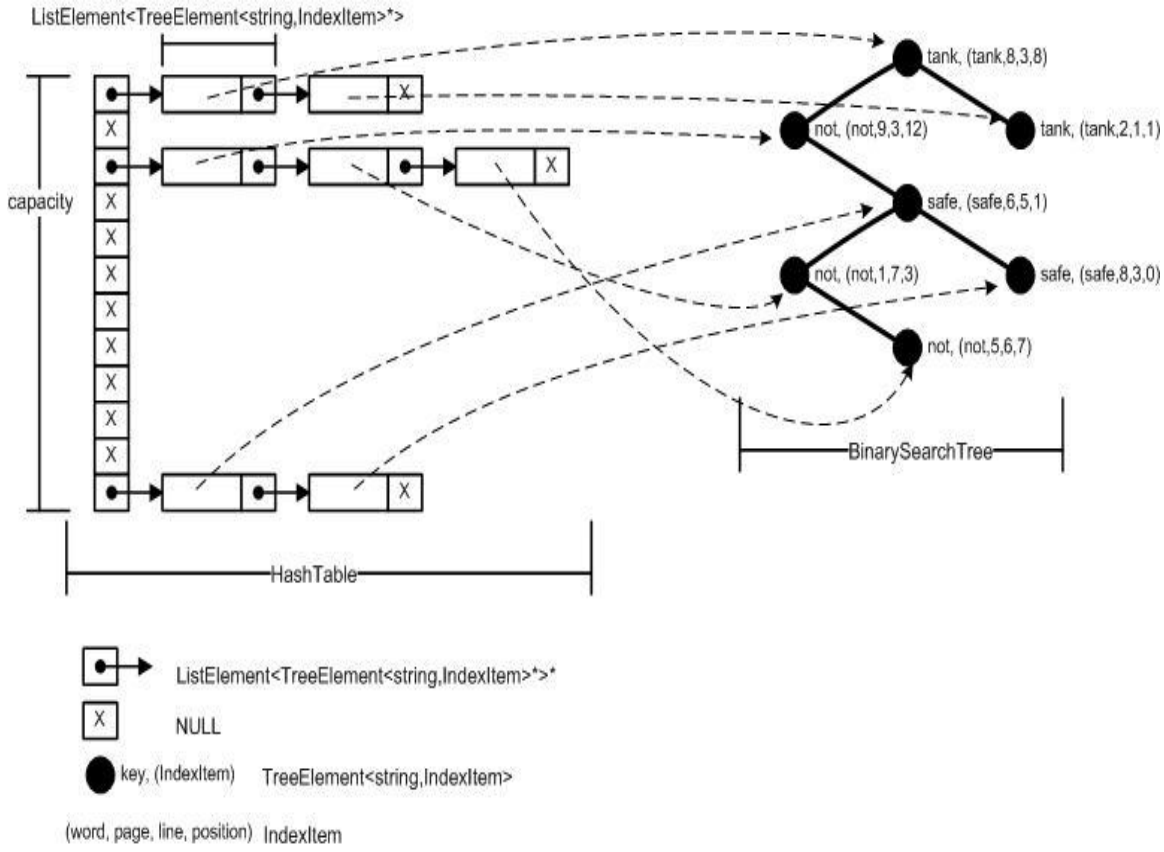
Gegeven de headers IndexItem.h, BookIndex.h, BinarySearchTree.h, HashTable.h en de cpp bestanden BookIndex.cpp, BinarySearchTree.cpp, HashTable.cpp, template_instantiations.cpp en main.cpp.

Een IndexItem stelt een voorkomen voor van een bepaald woord (een `string`) op een bladzijde, lijn en positie binnen die lijn.

BookIndex stelt de volledige index voor, en houdt intern een HashTable en een BinarySearchTree bij. De BinarySearchTree is een gewone binaire zoekboom (cursus Appendix B3.3 blz 190 en volgende) zonder balancering waarbij elk element een pointer naar zijn eventuele linkse en rechtse kind bijhoudt, en een pointer naar zijn ouder in de boom. De BinarySearchTree is opgebouwd uit TreeElements die telkens een sleutel en een waarde bijhouden: een woord en een corresponderend IndexItem. De HashTable gebruikt het woord van een IndexItem als sleutel, en als waarde houdt deze (voor elke sleutel dus) een enkelvoudig gelinkte lijst van TreeElement pointers bij, die verwijzen naar het boomelement dat het IndexItem bevat dat bij deze sleutel hoort. Zie onderstaande figuur met een schematische voorstelling van de datastructuur van BookIndex.

De HashTable bevat ook een extra array die de lengte van elke lijst van de HashTable bijhoudt.

Opzoeken van de IndexItems behorend bij een woord kan dus op twee manieren: via de HashTable of via de BinarySearchTree. Voor een klein aantal IndexItems zal de HashTable efficiënter zijn, voor een groter aantal de BinarySearchTree.



<u>Gevraagd</u>: vul de cpp files verder aan.
Meerbepaald, implementeer

- In BinarySearchTree.cpp
  - **operator[ ]** (2,5 punten) (signatuur in headerbestand, geeft std::list<V> terug)
- In HashTable.cpp:
  - **destructor** (1,5 punten)
  - **put** (2 punten)
  - **operator[ ]** (1,5 punten) (signatuur in headerbestand, geeft std::list<V> terug)
- In BookIndex.cpp:
  - **constructor** (0,5 punten)
  - **destructor** (0,5 punten)
  - **add** (1,5 punten)
  - **operator[ ]** (2 punten) (zie ook <u>hints</u> en header, geeft std::list<IndexItem> terug)

Enkele hints/opmerkingen:

- Schrijf je naam, stamnummer en richting bovenaan elk cpp bestand.
- Je mag de gegeven header bestanden NIET aanpassen! Doe je dit toch, dan verlies je punten.
- Bestudeer aandachtig de gegeven header bestanden; ze bevatten extra informatie over de te implementeren functies en de werking van de code.
- Zoals hierboven vermeld is het opzoeken van de waarden behorend bij een bepaalde sleutel (woord) efficiënter via ofwel de BinarySearchTree ofwel de HashTable, afhankelijk van de lengte van de lijst in de HashTable. Implementeer de [ ] operator in BookIndex zodanig dat er enkel in de BinarySearchTree gezocht wordt indien er meer dan 20 elementen met dezelfde hashwaarde aanwezig zijn in de HashTable. Anders wordt steeds via de HashTable gezocht.
- Voor de implementatie van de HashTable heb je een hashfunctie nodig; die is reeds aanwezig in de klasse: deze neemt een `string` als argument en geeft een `unsigned int` terug
- In de `main` methode kan je gemakkelijk enkel je BinarySearchTree testen door de functie `testBST` uit te voeren. Deze zal een BinarySearchTree van `string`s aanmaken, en er de gevraagde methoden op testen. Hetzelfde geldt voor HashTable, hiervoor is de functie `testHT` voorzien. Eens je de BookIndex klasse hebt afgewerkt kan je die testen via `testBookIndex`.
- De `main` methode hoeft in principe niet aangepast te worden, maar dit is wel toegelaten.

Veel succes!

Appendix (voor de volledigheid):

Project aanmaken in MS Visual C++ EE:

• Kies File – New – Project en selecteer als Project Type Win32.

• Als Template selecteer je Win32 Console Application.

• Onderaan vul je dan een projectnaam, een opslaglocatie en de naam van de (nieuwe) Solution in.

• Na bevestigen wordt een Win32 Application wizard opgestart. Druk op Next, check of Console Application als Application type wordt weergegeven en vink Empty Project aan. Vervolgens kan je deze wizard beëindigen. Er is nu een nieuw project en een nieuwe Solution aangemaakt.

• De header files toevoegen kan je door rechts te klikken op de 'Header files' folder en Add – Existing Item aan te klikken

KLADPAPIER:

Examen Software Ontwikkeling I
2e Bachelor Informatica
Faculteit Wetenschappen
Academiejaar 2010-2011
21 januari, 2011

BELANGRIJK

1. Schrijf je naam onderaan op elk blad.
2. **Dit deel gaat enkel over C, GEEN C++.**
3. Kladpapier vind je achteraan.
4. Veel succes!

Deel II + III. Open Boek gedeelte op papier (11u15 - 12u30)

Programmeren in C

[op 8 punten van de 20]

Deze vraag handelt over een (op positie) gesorteerde **enkelvoudig gelinkte** lijst van elementen die een woord en de positie van dat woord in een tekstbestand bijhouden.

Gegeven volgend header bestand:

```
typedef struct list_element{
      int position;
      char* word;
      struct list_element* next;
}list_element;

/* maakt een nieuw list_element aan met de opgegeven waarden*/
/* geeft NULL terug indien niet genoeg geheugen beschikbaar is*/
list_element* init_list_element(int position, char* word);

/* gesorteerd invoegen van een element to_add*/
/* return waarde :
/*     0 -> toevoegen is gelukt */
/*    -1 -> er is al een element op de betreffende positie */
int add_list_element(list_element* to_add, list_element** head);

/* verwijderen van het element op positie position*/
/* return waarde :
/*     0 -> verwijderen is gelukt */
/*    -1 -> er is geen element op positie position */
int remove_list_element(int position, list_element** head);

/* vernietigt de opgegeven lijst */
/* en geeft het gealloceerde geheugen vrij */
void destroy_list(list_element** head);
```

```
/* concateneert de opgegeven string word achteraan aan de string */
/* op positie position */
/* indien er geen element is op positie position, */
/* wordt een nieuw element toegevoegd */
/*     0 -> achteraan toevoegen is gelukt */
/*    -1 -> er is niet genoeg geheugen beschikbaar */
int append_word(int position, char* word, list_element** head);
```

a. Waarom geeft de functie `init_list_element` een pointer naar `list_element` terug en niet een `list_element` zelf? [op 0,5 punten van de 20]

Geef ook de implementatie van de functie `init_list_element`
[op 1,5 punten van de 20]

```
list_element* init_list_element(int position, char* word)
```

b. Waarom wordt een dubbele pointer meegegeven naar de kop van de lijst in de functies `add_list_element` en `remove_list_element` [op 0,5 punten van de 20]

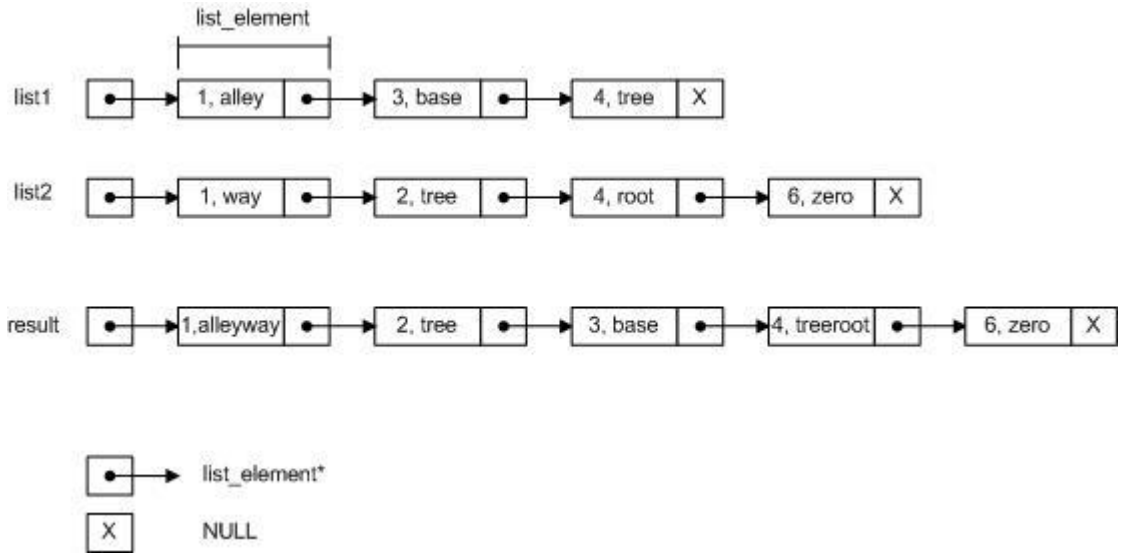
c. Geef de implementatie van de functie `destroy_list` [op 1 punt van de 20]

```
void destroy_list(list_element** head)
```

d. Geef de implementatie van de functie `append_word` [op 1,5 punten van de 20]

```
int append_word(int position, char* word, list_element** head)
```

e. Stel dat men 2 dergelijke lijsten wil samenvoegen. Hiermee wordt bedoeld dat in de resulterende lijst alle elementen uit de 2 oorspronkelijke lijsten aanwezig zijn, en indien er op een bepaalde positie in beide lijsten een element aanwezig is, dan worden de woorden van deze elementen geconcateneerd, zoals hieronder schematisch voorgesteld:



De code op de volgende pagina implementeert deze list merging. Er zitten echter in totaal 4 foutjes in deze code (zowel programmeerfouten als mogelijks algoritmische fouten). Bekijk de code kritisch en corrigeer deze fouten. Indien je niet genoeg plaats hebt mag je de (mogelijks) ontbrekende code onder het codefragment schrijven. [op 2 punten van de 20]

```
int plus(list_element* list1, list_element* list2, list_element** new_list){

char* sum;
list_element* temp = NULL;

while(list1 != NULL || list2 != NULL){

	if(list1 != NULL && list2 != NULL){
		if(list1->position < list2->position){
			temp = init_list_element(list1->position, list1->word);
			list1 = list1->next;
		}else{
			sum =
(char*)malloc(sizeof(char)*(strlen(list1->word)+strlen(list2->word)));
			if(sum == NULL){
				destroy_list(new_list);
				return -1;
			}
			strcpy(sum,list1->word);
			strcpy(sum,strncat(sum,list2->word,sizeof(list2->word)));
			temp = init_list_element(list1->position, sum);
			list1 = list1->next;
			list2 = list2->next;
		}
	}else if(list1 == NULL && list2 != NULL){
		temp = init_list_element(list2->position, list2->word);
		list2 = list2->next;
	}else if(list2 == NULL && list1 == NULL){
		temp = init_list_element(list1->position, list1->word);
		list1 = list1->next;
	}
	if(temp == NULL || add_list_element(temp,new_list)){
		destroy_list(new_list);
		return -1;
	}

}
return 0;


}
```

DEEL III Make bestand

Leg uit hoe een Make bestand automatisch kan gegenereerd worden. Vermeld de voorwaarde waaraan moet voldaan zijn om dit automatisch te kunnen.
Indien deze voorwaarde voldaan is, welke input is dan nog van de gebruiker vereist?
[op 1 punt van de 20]

KLADPAPIER: