Software-Ontwikkeling I

Academiejaar 2012 – 2013

SOI@intec.ugent.be

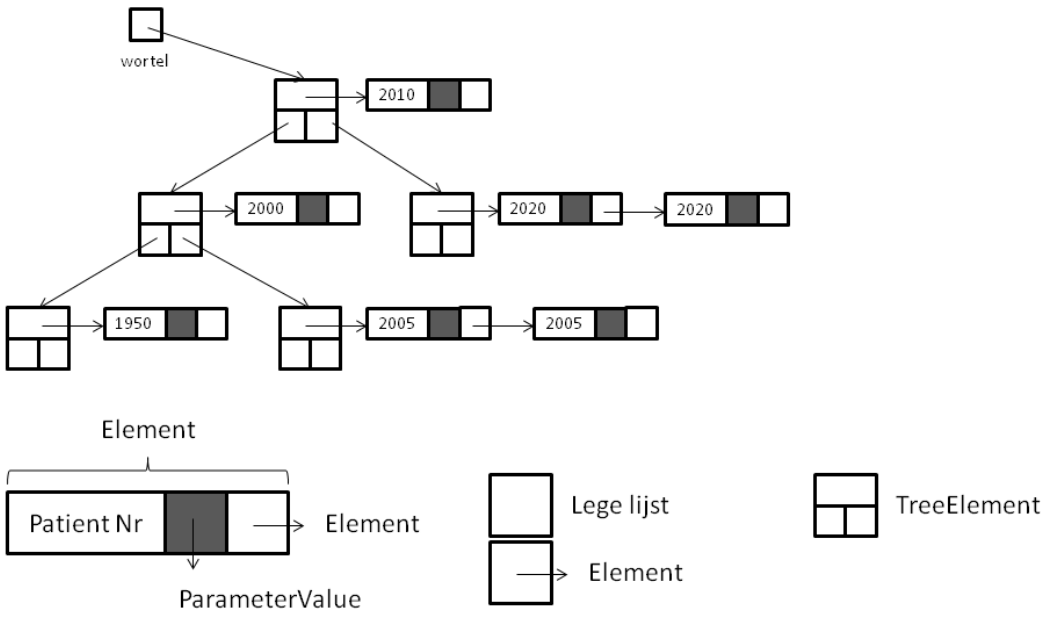
Practicum 5
Polymorfisme – Templates - STL

1. Inleiding

Dit practicum focust op drie geavanceerde principes in C++, nl. het correcte en efficiënte gebruik van polymorfisme en overerving (zie syllabus Hoofdstuk 12), definitie en gebruik van templates (zie syllabus Hoofdstuk 13) en het gebruik van STL (zie syllabus Hoofdstuk 14).

De hierboven aangehaalde onderwerpen zullen gebruikt worden om een datastructuur te ontwerpen en te implementeren voor het bijhouden van medische parameterwaarden.

Om de medische parameters op een efficiënte manier op te slaan, zullen we gebruik maken van een binaire zoekboom met geschakelde lijsten (zie syllabus appendix C, sectie C.3 voor een volledige uitleg). Elk element uit deze boom kan 1 of meerdere medische parameterwaarden van een patiënt bevatten. Als meerdere parameterwaarden in dezelfde knoop van de binaire zoekboom moeten worden opgeslagen, dan worden deze door middel van een gelinkte lijst bijgehouden (zie syllabus appendix C, sectie C.2.2). Een grafische illustratie van deze aanpak kan je vinden in Figuur 1. In deze figuur is een binaire zoekboom met 5 knopen weergegeven. In deze boom werden reeds 7 medische parameterwaarden van in totaal 5 patiënten opgeslagen.



Figuur 1: Visuele voorstelling van de binaire zoekboom

Het eerste deel van dit practicum gaat over het aanmaken van de vereiste datatypes om de medische parameterwaarden te beschrijven. Daarna zullen we in het tweede deel de binaire zoekboom uitwerken om in het derde deel de veralgemening van de zoekboom door te voeren aan de hand van templates, zodat niet alleen medische parameterwaarden kunnen opgenomen worden, maar dat de ontwikkelde datastructuur ook gebruikt kan worden om generieke objecten bij te houden.

Belangrijk: bij het verbeteren wordt rekening gehouden met:

- *efficiënt geheugenbeheer*: hoe minder geheugen je gebruikt, hoe beter,
- *vermijd geheugenlekken*: op Minerva staat in het document 'documenten/oefeningen/FAQ_VisualCppEE.pdf' de manier om je code te testen,
- *hergebruik code*: probeer reeds bestaande methodes te hergebruiken, in plaats van code te kopiëren en plakken,
- *Samenwerken*: Er kan samen met een collega tijdens het practicum aan de oplossing begonnen worden, maar het is belangrijk dat nadien de opgave individueel afgewerkt en ingediend wordt.

Verander niets in de gegeven header bestanden voor de binaire zoekboom. Je mag enkel zaken toevoegen. Vanzelfsprekend mag je aangepaste signaturen gebruiken om

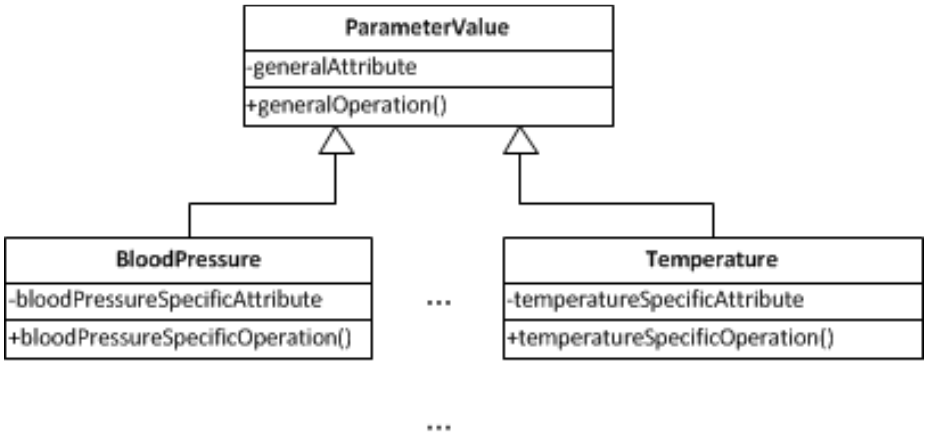
de methodes aan te passen voor de versie van de binaire zoekboom met templates. In `main.cpp` staan bovendien enkele methodes om je code te testen. Neem ook zeker eens een kijkje op http://www.cplusplus.com/reference/ om een oplossing voor je probleem te vinden en een overzicht van bestaande functionaliteit.

2. Overerving en polymorfisme

In de volgende paragraaf kan je een beschrijving vinden, die aangeeft welke functionaliteit moet voorzien worden in de klassen om medische parameterwaarden voor te stellen. Het doel van deze oefening is dat je aan de hand van de beschrijving de overervingstructuur van de klassen bepaalt, alsook welke attributen en methodes in welke klassen voorzien moeten worden. Jullie moeten m.a.w. zelf de declaraties van de klassen opmaken.

"De medische parameterwaarden (`ParameterValue`), die moeten worden opgeslagen, hebben een bepaald tijdstip. Dit tijstip wordt bepaald door de datum en het uur van de meting (d.m.v. een `Timestamp` klasse) en kan opgevraagd worden door de methode `Timestamp getTimestamp()` en aangepast worden met `setTimestamp(Timestamp Timestamp)`. Bovendien beschikken de metingen over 1 welbepaalde waarde, die opnieuw kan opgevraagd worden met `Value getValue()` en aangepast worden met `setValue(Value value)`. Waarden worden gedefinieerd door hun eenheid (`float`) en grootheid (`string`). Bemerk dat jullie zelf de `Timestamp` en `Value` klasse moeten definiëren. Het is verder mogelijk om aan elke medische parameterwaarde een beschrijving, bv. "Dit is de temperatuur van de patiënt" (`getDescription()`) en een eenvoudige diagnose, bv. "Koorts" (`getDiagnosis()`) op te vragen. Ten slotte moeten de metingen ook kunnen herschaald worden volgens een bepaalde reële factor, d.m.v. de methode `rescale(float factor)`. Dit laatste zal er uiteraard ook voor zorgen dat je eventueel de grootheid moet mee veranderen, bv. van ml naar cl."

Je opdracht is nu om de nodige klassen uit te werken in C++, door gebruik te maken van polymorfisme en overerving. Zorg ervoor dat er geen dubbele code geprogrammeerd wordt. Houd voldoende rekening met eventuele virtuele methodes en abstracte basisklassen. Het gebruik van de `const` modifier is voor deze opgave niet nodig (zowel voor de argumenten, return waarden en methoden). Denk dus goed na welke variabelen en operaties je in de basisklasse zal steken, en welke in de kindklassen. Declareer de nodige C++ klassen, zodat minstens de volgende medische parameterwaarden kunnen worden voorgesteld: Lichaamstemperatuur, Hartslag en Bloeddruk. Een eerste aanzet vind je hieronder in Figuur 2. Zorg waar nodig voor (copy) constructoren en destructoren, getters en setters. Implementeer elke klasse in aparte .cpp en .h bestanden.



Figuur 2: Voorbeel overervingstructuur

3. Binaire zoekbomen

Een binaire zoekboom wordt gebruikt om elementen gesorteerd op te slaan volgens het volgende principe:

- de waarde van de sleutel van iedere knoop is groter dan de waarde van alle sleutels in de linker subboom,
- de waarde van de sleutel van iedere knoop is kleiner dan de waarde van alle sleutels in de rechter subboom.

3.1 Zoeken in een binaire zoekboom – search

Vermits de elementen gesorteerd zijn opgeslagen, kan volgend algoritme toegepast worden om een element met een bepaalde waarde op te zoeken in de binaire zoekboom:

vergelijk de waarde van het element met de waarde opgeslagen in de ouderknoop, indien deze kleiner is: ga naar de linker sub-boom en herhaal hetzelfde principe daar, indien de waarden gelijk zijn: element gevonden (wordt search hit genoemd), indien de waarde groter is: ga naar de rechter sub-boom

en herhaal hetzelfde principe daar. Wanneer een gekozen sub-boom leeg is, stopt het algoritme: element niet gevonden (wordt search miss genoemd).

Er wordt dikwijls een recursieve implementatie van dit algoritme gekozen.

3.2 Toevoegen in de wortel - root insertion

Zoals in sectie C.3.3 van de syllabus beschreven, gebeuren alle toevoegingen in een binaire zoekboom aan de toppen van de boom (i.e. onderaan). Voor sommige toepassingen is het aangewezen dat het nieuwe element zich in de wortel bevindt, vermits dit de toegangstijd verkort voor de meest recente knopen. Het is belangrijk om in te zien dat een nieuw element niet zomaar bovenaan kan toegevoegd worden. Daarom wordt de toevoeging onderaan gedaan, en aan de hand van rotaties wordt het nieuwe element gepromoveerd naar de wortel van de boom. Rotaties komen aan bod in volgende sectie.

3.3 Rotatie in binaire zoekbomen - rotation

Een rotatie is een lokale transformatie op een deel van een binaire zoekboom die toelaat om de rol van de wortel en een van zijn kinderen om te wisselen, en dit terwijl de voorwaarden tussen de knopen van een binaire zoekboom behouden blijven. Rotatie is een basisoperatie die gebruikt wordt om de knopen in een binaire zoekboom te herschikken. Zoals hierboven vermeld wordt bij toevoegen van een element in de wortel (Eng.: root insertion) het element onderaan in de binaire zoekboom toegevoegd en dan de boom herschikt tot de nieuwe knoop in de wortel staat. Rotaties worden zeer dikwijls recursief toegepast. Merk op dat in het opgavebestand `BinarySeachTree.cpp` reeds een eerste implementatie van een rotatiemethode werd gegeven. Pas deze methode aan waar nodig.

3.4 Element verwijderen uit een binaire zoekboom - remove

Een element kan niet zomaar uit een binaire zoekboom verwijderd worden (omdat de voorwaarden voor een binaire zoekboom dan kunnen geschonden worden). De beste manier is de volgende:

- promoveer in de rechter sub-boom het kleinste element (i.e. meest linkse) naar de root van deze sub-boom,
- dit element krijgt dezelfde linker sub-boom als het verwijderde element,
- dit element komt in de volledige boom in de plaats van het verwijderde element.

De binaire zoekboom die hier gebruikt wordt, organiseert `long`s (de identifier van een patiënt) met de medische parameterwaarden van die patiënt. Deze binaire zoekboom bestaat uit een verzameling van boomelementen (`TreeElement`) die als waarde gelinkte lijsten van `Element` objecten hebben en bovendien drie `pointer`s, een naar elke

deelboom van de binaire zoekboom en een naar de ouder. De `Element` objecten bestaan op hun beurt uit de identifier van de patiënt, uit een `pointer` naar het volgende element in de lijst en een `pointer` naar de medische parameterwaarde (zie Figuur 1).

Deze binaire zoekboom gebruikt volgend principe; alle elementen waarvan de sleutel dezelfde sleutelwaarde heeft worden in dezelfde gelinkte lijst opgeslagen. De positie van die gelinkte lijst wordt uiteraard bepaald door de correcte positie in de zoekboom te vinden.

Gegeven de header `BinarySearchTree.h` en de bestanden `BinarySearchTree.cpp` en `main.cpp`.

Gevraagd: vul BinarySearchTree.cpp verder aan:

- `constructoren:` alloceren de nodige ruimte voor de binaire zoeknoom; de constructor zonder argument neemt een lege zoekboom als grootte voor de binaire zoekboom,
- `destructor`: geeft alle geheugen vrij ingenomen door de binaire zoekboom; de medische parameterwaarde objecten moeten hierbij niet vrijgegeven worden,
- `void put(long patientID, ParameterValue* e)` voegt een medische parameterwaarde toe aan de binaire zoekboom volgens het volgende algoritme: aan de hand van het opgegeven patiëntennummer wordt de correcte top in de boom voor de toe te voegen medische parameterwaarde bepaald indien het patiëntennummer nog niet voorkomt in de binaire zoekboom (geval 1). Anders wordt de knoop gezocht die overeenkomt met het opgegeven patiëntennummer (geval 2).
    - Geval 2.1: er wordt een nieuwe gelinkte lijst gemaakt en de nieuwe medische parameterwaarde wordt daaraan toegevoegd (voor de structuur, zie Figuur 1). Pas bij het toevoegen de principes van rotatie toe (zie Sectie 3.3),
    - Geval 2.2: de aanwezige gelinkte lijst wordt achteraan uitgebreid met de nieuwe medische parameterwaarde.,
- `ParameterValue* get(long patientID`): geeft een pointer terug naar het eerste voorkomen van een medische parameterwaarde waarvan de sleutel overeenstemt met de opgegeven sleutel. Anders wordt NULL teruggegeven,
- `ParameterValue* remove(long patientID)`: verwijdert een medische parameterwaarde uit de binaire zoekboom waarvan de sleutel overeenstemt met de opgegeven sleutel. Indien er meerdere medische parameterwaarden zijn met die sleutel wordt de eerste dergelijke medische parameterwaarde verwijderd. Uiteraard moeten andere medische parameterwaarden waarvan de sleutel dezelfde waarde heeft als de sleutel van de te verwijderen medische parameterwaarde na verwijdering nog steeds vindbaar zijn. Bovendien moet de knoop uit de binaire zoekboom verwijderd worden

wanneer de laatste medische parameterwaarde die bij deze knoop hoort, verwijderd werd. Deze methode geeft ook de pointer terug naar de verwijderde medische parameterwaarde,

- `printTree()`: print alle (sleutel, waarde) paren af in de binaire zoekboom; voor de waarde moeten ook effectief de kenmerken van de medische parameterwaarden (zie Sectie 2) in de lijst uitgeprint worden en niet de pointer.

4. Templates

In de voorgaande opdracht heb je een binaire zoekboom geïmplementeerd die medische parameterwaarden bijhoudt. Het vergt niet veel verbeelding om te begrijpen dat deze structuur ook perfect zou kunnen gebruikt worden voor het opslaan van arbitraire data, zolang deze maar bestaat uit de combinatie van een `long` met een waarde van een willekeurig type. Herwerk daarom de implementatie van de `Element`, `TreeElement` en `BinarySearchTree` klasse door gebruik te maken van templates om als dusdanig willekeurige waarden te kunnen opslaan in de binaire zoekboom. Meer informatie omtrent templates kan je vinden in de syllabus, Hoofdstuk 13. Implementeer de template variant van de binaire zoekboom in afzonderlijke bestanden, nl. `BinarySearchTreeTemplate.h` en `BinarySearchTreeTemplate.cpp`.

Normaalgezien worden template klassen geïmplementeerd in een header file, omdat voor de compiler zowel de definitie als implementatie zichtbaar moeten zijn op het moment van instantiëring van een template klasse (hier dus in de `BinarySearchTree`, `TreeElement` en `Element` klasse) en omdat template klassen doorgaans klein in omvang zijn.

Hier is dit niet het geval en wordt de implementatie opgesplitst in header en cpp bestanden. Daarom voeg je best bijgevoegd bestand `template_instantiations.cpp` aan je project toe. Dit zal de compiler forceren de code te genereren voor een `BinarySearchTree` van `string`s. Meer informatie kan je ook in de cursus vinden, Hoofdstuk 13, Sectie 3.

Doe je dit niet, dan zal je linker errors krijgen voor elk van de methoden van de `BinarySearchTree` die je probeert te gebruiken. Verder hoef je dit bestand nergens te includeren, deze dient enkel om de compiler te verplichten de juiste code te genereren. Uiteraard zal je in je uiteindelijke ontwerp een `BinarySearchTree` gebruiken met een ander waarde-type dan pointers naar `string`s, namelijk met pointers naar `ParameterValue`s. Dan moet je ook in dit `template_instantiations.cpp` bestand een aanpassing doen naar het juiste type zodat de juiste code voor de datastructuur gegenereerd wordt.

5. STL

STL (Eng.: Standard Template Library) werd in 1994 ontwikkeld als de standaard bibliotheek voor C++: het bevat datatypes, algoritmen en hulpklassen. De belangrijkste onderdelen zijn:

- *containers*: dit zijn klassen, die dienen voor dataopslag en gebaseerd zijn op een bepaalde datastructuur. Er zijn zeven container-types mogelijk in STL: `vector`, `deque`, `list`, `set`, `multiset`, `map` en `multimap`,
- *iteratoren*: dit zijn klassen die dienen om efficiënt door alle aanwezige elementen in de containers te itereren,
- *algoritmen*: dit zijn methodes die een bepaald algoritme implementeren, bijvoorbeeld een sorteer-, zoek- of filteralgoritme.

STL is volledig gebaseerd op templates: de containers zijn allen klasse templates. De achterliggende code is nauwkeurig ontwikkeld om zo snel mogelijke uitvoering toe te laten.

In de voorgaande stappen van dit practicum hebben jullie een eigen implementatie gemaakt van een gelinkte lijst. Hierin werden de verschillende medische parameterwaarden van een bepaalde patiënt, d.m.v. `Element` of `ElementTemplate` objecten aan elkaar geschakeld.

We kunnen echter ook gebruik maken van een bestaande container implementatie uit de STL bibliotheek, nl. `list`. In dit laatste onderdeel wordt jullie gevraagd om in de template variant van de binaire zoekboom de eigen implementatie van de geschakelde lijst om te vormen naar een versie die gebruik maakt van de `list` STL container om de verschillende parameterwaarden op te slaan. Bekijk ook alle methoden van de binaire zoekboom die over de elementen van de gelinkte lijsten itereren, en pas die aan zodat die nu gebruik maken van STL iteratoren om doorheen te elementen te lopen. Pas ook zeker de implementatie van de methode `printTree()` aan, zodat die nu gebruik maakt van STL iteratoren. Onthoud dat een print-methode de uit te printen elementen niet mag/zal aanpassen. Implementeer je aanpassingen in een derde versie van de binaire zoekboom, met als naam `BinarySearchTreeSTL`.

6. Indienen

- Zip je broncode bestanden in een bestand `naam_voornaam_5.zip` bvb. `volckaert_bruno_5.zip`. Gebruik een standaard zipformaat, geen rar of 7zip,
- Zet in je bronbestand bovenaan in commentaar je naam en de naam van het bestand,
- Respecteer de signatuur in de header bestanden. Het aanbrengen van veranderingen hieraan is NIET toegestaan (buiten noodzakelijke aanpassingen aan type-declaraties voor de uitbreidingen naar templates). Doe je dit toch, dan

zal dat zich reflecteren in je score. Uitbreidingen aan de aangeleverde bestanden en extra header bestanden zijn uiteraard wel toegelaten,

- Stuur je oplossing door middel van de dropbox op Minerva door naar Bruno Volckaert, ten laatste op **zondag 25 november**,**23:59**.

Veel succes!