Practicum Datastructuren

– voorjaar 2006 –

Extra opgave: *Een stamboom voor de hele familie*

1 Vrijwillige opgave

Deze opgave is een geheel vrijwillig onderdeel van het practicum. Het inleveren is dus niet noodzakelijk. Mocht je de behoefte hebben om nog eens te oefenen met lijsten en bomen, dan raden wij deze opgave aan. Het voordeel van het wel inleveren is dat je uitwerking wordt nagekeken en gecommentariëerd. Bijkomend voordeel is dat we je twee laagste cijfers voor het practicum (inclusief deze opgave) niet meetellen, i.p.v. alleen het laagste cijfer. Hiermee kun je dus je practicumcijfer iets ophalen, mocht je andere opgaven niet of slecht gemaakt hebben.

2 Achtergrond

Een stamboom beschrijft een deel van een familiegeschiedenis en bevat informatie over personen en, impliciet in de vorm van de stamboom, hun onderlinge relatie. Stambomen worden al sinds mensenheugenis gemaakt, maar door het gebruik van Internet om verre familieleden te vinden en om stambomen te publiceren is er een enorme wildgroei ontstaan. Naar mate stambomen groter worden komt er meer vraag naar software die het werken met deze datastructuren beheersbaar houden.

3 Leerdoelen

Na afloop van deze opdracht ben je in staat om te werken met complexe datastructuren die zowel bomen als lijsten bevatten.

4 Instructie

Neem eerst hoofdstuk 16 en 18 van het dictaat nog eens door. Begin pas daarna aan de volgende deelopdrachten. Zorg ervoor, dat je (op papier en aan de hand van een of meer plaatjes) al een ontwerp van je datastructuur en optimalisatiealgoritme hebt gemaakt, vóórdat je aanschuift achter de pc om je code in te voeren en uit te testen.

Probleemschets

We kiezen ervoor om per persoon de naam en die van zijn of haar partner op te slaan. Verder houden we een lijst van kinderen bij, zoals je kunt zien in de structuur `Persoon` hieronder. Voor de lijst van kinderen kun je de `List`-structuur uit het dictaat hergebruiken. Voor de lijstelementen gebruik je verwijzingen naar dynamisch gealloceerde personen (`Persoon *`).

```
typedef Persoon *El;

struct Persoon
{
    char *naam;
    char *partner;
    List kinderen;
```

```
    Persoon (char n[], char p[], List k) : kinderen(k)
    {
        naam = new char [strlen (n) + 1];
        strcpy (naam, n);
        partner = new char [strlen (p) + 1];
        strcpy (partner, p);
    }
};
```

De stamboom kun je implementeren als als een boom bestaande uit knopen die verwijzen naar personen met nul of meer onderbomen: de kinderen. We definiëren twee handige afkortingen: `Boom` voor verwijzingen naar personen en `LegeBoom` voor lege bomen. Als interface voor de stamboom definiëren we de klasse `Stamboom`. De zichtbare interface voor de stamboom definieert een constructor zonder argumenten en enkele methoden om informatie uit de stamboom te halen.

```
typedef Persoon *Boom;

const Boom LegeBoom = NULL;

class Stamboom
{
private:
    Boom wortel;
public:
    Stamboom (): wortel(LegeBoom) { };
    bool trouwtMet (char persoon[], char partner[]);
    bool krijgtKind (char ouder[], char kind[]);
    void toon();
    bool uitgestorven ();
    Stamboom familie (char ouder[]);
    bool telg (char persoon[]);
    bool telgVan (char nazaat[], char voorouder[]);
};
```

Deelopdracht 1

Implementeer de methodes om een stamboom te bouwen. De methode `trouwtMet` vult het `partner` veld in van een `persoon` die al in de stamboom moet voorkomen en nog niet getrouwd is, anders levert de functie **`false`** op. De methode `krijgtKind` voegt een nieuw persoon (`kind`) aan de stamboom toe als kind van een (al dan niet getrouwde) `ouder` die al in de stamboom voorkomt, anders levert de functie **`false`** op.

Bedenk zelf een familie van een ongeveer tien personen en bouw daar de stamboom van. Zorg dat verschillende relationele situaties voorkomen en gebruik deze stamboom om je programma te testen.

Deelopdracht 2

Implementeer de methode toon die de stamboom op een overzichtelijke manier afdrukt. Aangezien je dit op een recursieve manier wilt doen gebruik je een statische privé hulpmethode.

Deelopdracht 3

Implementeer de methodes om informatie uit een stamboom te halen. De methode `uitgestorven` test of een stamboom geen personen bevat. De `familie` methode levert een deel van een stamboom op vanaf een bepaalde `ouder`, en een 'lege stamboom' als die persoon niet voorkomt.

Deelopdracht 4

Maak bij het implementeren van de onderstaande methoden handig gebruik van de methodes die je al geïmplementeerd hebt. De methode `telg` bepaalt of een `persoon` voorkomt in de stamboom. De methode `telgVan` bepaalt of een `nazaat` voorkomt in de deel stamboom vanaf een `voorouder`.

5 Producten

Als producten moet je schetsen hebben van de gebruikte datastructuren en (uitgeteste!) C++-code, die voldoet aan de gevraagde specificaties en aan de kwaliteitscriteria zoals die gesteld worden (zie punt 6: zelfreflectie).

6 Zelfreflectie

Via de cursuswebpagina kom je uit op een aantal criteria die we ook bij deze cursus gebruiken voor de kwaliteit van je ingeleverde werk. Ga voor jezelf na, voor zover dat nu al van toepassing is, in hoeverre je uitwerkingen voldoen aan deze criteria. Je uitwerking van deze zelfreflectie moet je echter niet inleveren.

7 Inleveren van je producten

Vóór maandag 5 juni, 9.00 uur, en wel door jullie programma één keer via e-mail met de tekst `PI2 opgave13` in de `Subject:`-regel naar **`S.Smetsers@science.ru.nl`** te sturen. Vergeet niet om **in** de uitwerking duidelijk jullie namen te vermelden.