

Tentamen Programmeermethoden

Dinsdag 5 januari 1999, 19.00–22.00 uur

Universiteit Leiden — Informatica

Bij alle te schrijven functies moeten de variabelen in de heading voorkomen (niet stiekem globale variabelen gebruiken). De opgaven tellen alle vier even zwaar mee. Veel succes!

1. Gegeven zijn `const int n = 1000`; en een array `A` met een rij van `n` onderling verschillende oplopend gesorteerde getallen: `int A[n]`. Neem in het vervolg steeds aan dat de integer `k` een deler is van `n`.

a. Schrijf een `int` C++-functie `Komtvoor (A,X,k)` die het eerste getal uit het deelrijtje `A[k-1], A[2*k-1], A[3*k-1], ..., A[n-1]` opzoekt dat groter dan of gelijk aan `X` is, en daarvan de array-index oplevert; als `X` groter dan `A[n-1]` is, moet de functie `-1` geven.

b. Schrijf een `int` C++-functie `Lineair (A,X,links,rechts)` die met lineair zoeken bepaalt of `X` voorkomt tussen de getallen `A[links], ..., A[rechts]` (grenzen inbegrepen), en zo ja de array-index `i` met `X == A[i]` oplevert, en zo nee `-1`.

Neem aan dat $0 \leq \text{links} \leq \text{rechts} \leq n-1$.

c. Schrijf een `int` C++-functie `Zoeken (A,X,k)` die het getal `X` in `A` als volgt opzoekt. Eerst wordt de functie van **a** gebruikt om het zoeken tot een `k`-tal opeenvolgende array-elementen te kunnen beperken; vervolgens wordt hiervoor de functie van **b** ingezet. Als `X` niet voorkomt wordt `-1` opgeleverd, anders de index `i` met `X == A[i]`.

d. Hoeveel vergelijkingen tussen `X` en array-elementen doet het algoritme van **c** maximaal, en wanneer is dat? Is dit minder dan / evenveel als / meer dan bij binair zoeken?

2. Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Verder heb je ook nog *formele* en *actuele* parameters.

a. Leg deze zes begrippen aan de hand van een klein voorbeeld duidelijk uit.

b. Gegeven zijn de twee volgende functies:

```
int berg (int a, int b, int c) {
    a--; b--; c += 2; cout << a << b << c << endl; return a + b + c;
} // berg
int dal (int a, int b) {
    a = a * b; return berg (a,a,b) + berg (b,b,a);
} // dal
```

Neem aan dat de waardes van de globale variabelen `x` en `y`, beide `int`, 2 respectievelijk 5 zijn. Wat gebeurt er bij `cout << dal (y,x) << x << y << endl`? Wat wordt er afgedrukt? Probeer duidelijke uitleg te geven.

c. We voegen vijf maal een `&` toe, en wel bij alle parameters in de headings van `berg` en `dal`. Beantwoord opnieuw vraag **b**. Als er verschillende antwoorden mogelijk zijn: geef deze.

d. Neem nu aan dat de waarde van de globale variabele `a`, een `int`, 0 is. Wat gebeurt er bij `cout << dal (a,berg (a,a,a)) << a << endl`? Wat wordt er afgedrukt? De `&`'s staan er niet bij, de situatie is dus verder als bij **b**.

e. Stel dat in het programma ergens de test `if (berg (a,a,a) == berg (a,a,a))...` staat. Levert deze test altijd `true` op? Maak hierbij onderscheid tussen de situatie met en zonder `&`'s.

3. In een array `D` (`int D[n][n]` met `n` een constante) worden de onderlinge afstanden tussen een `n`-tal plaatsen bijgehouden. Zo betekent `D[1][5] == 70` dat de afstand tussen de plaatsen 1 en 5 precies 70 kilometer is. Neem aan dat alle voorkomende afstanden verschillend zijn, en dat `D[i][j] == D[j][i]` voor alle `i` en `j`. De `D[i][i]`'s zijn voorlopig allemaal 0.

Een voorbeeld (met `n == 3`):

```
0 70 30
70 0 35
30 35 0
```

a. Schrijf een `void` C++-functie `Dicht` (`D`) die voor iedere stad `i` de *index* van de dichtstbijzijnde stad opbergt in `D[i][i]`. In het voorbeeld moet onder andere gaan gelden: `D[0][0] == 2`, want stad 2 is voor stad 0 de dichtstbijzijnde.

b. Schrijf een `bool` C++-functie die controleert of `D` aan de *driehoeksongelijkheid* voldoet, dat wil zeggen of altijd geldt dat de afstand van plaats `i` naar plaats `j` kleiner dan of gelijk is aan de afstand die een “omweg” via een willekeurige andere plaats `k` kost. In het voorbeeld is de rechtstreekse afstand van 0 naar 1, 70, groter dan de afstand van 0 naar 2 plus de afstand van 2 naar 1, 65: `false` dus.

c. Schrijf een `bool` C++-functie `Loop` (`D,i,m`) die precies dan `true` oplevert als beginnende in stad `i`, je door steeds naar de dichtstbijzijnde stad te gaan (en van daar uit weer naar de dichtstbijzijnde ...) in *hooguit* `m` stappen weer in `i` terug bent. Gebruik **a**.

4. Gegeven is:

```
class Dorp { public: int inwo;    // aantal inwoners
              Dorp* vorige;    // wijst voorganger aan
              Dorp* volgende;  // en het volgende dorp
}; // Dorp
```

Met behulp hiervan kan een dubbelverbonden lijst van dorpjes worden opgebouwd. Zo'n lijst bestaat dus uit vakjes met een geheel getal, en twee pointers naar respectievelijk vorige en volgende dorp. Het eerste dorp (waar de ingangspointer naar wijst) heeft als voorganger `NULL`.

a. De aantallen inwoners van het eerste en het tweede dorp van een niet-lege lijst met ingang `Serie` van type `Dorp*` worden verwisseld. Schrijf een C++-functie `Wissel` (`Serie`) die dit doet. Denk aan de test op het bestaan van het tweede dorpje.

b. Schrijf een C++-functie `Voegtoe` (`Serie,zoveel`) die een nieuw dorpje met `zoveel` inwoners toevoegt vooraan de eventueel lege lijst `Serie` (van type `Dorp*`).

c. Schrijf een C++-functie die het eerste dorpje uit de lijst `Serie` verwijdert. Als de lijst leeg is of als het eerste dorpje nog inwoners heeft hoeft er niets te worden gedaan.

d. In de functies bij **a**, **b** en **c** staat in de heading de parameter `Serie`. Deze heb je call by value of call by reference doorgegeven (met een `&`). Maakt het voor de werking van deze functies verschil uit of die `&` erbij staat? Leg ook uit wat er bij deze twee mogelijkheden precies gebeurt tijdens executie van de betreffende functie.

e. Bepaal hoeveel dorpjes uit de lijst met ingang `Serie` in hun `inwo`-veld precies het gemiddelde van dat van hun voorganger en hun opvolger bevatten—mits ze die hebben natuurlijk. Schrijf hiertoe een `int` C++-functie `Aantal` (`Serie`).

Zodra het tentamen is nagekeken staat worden de resultaten toegestuurd.