1 Main

Oefening 1

```
#include <iostream>
using namespace std;

int main(){
    string naam;   // later zien we de 'echte' basics: char*
    cout << "hallo, wie ben jij? ";
    cin >> naam;
    cout << "Dag " << naam << endl;
    return 0;
}
```

Oefening 2

```
#include <iostream>
#include <iomanip>   // setw(..)   zoek op in de help
using namespace std;

int main(){
    // Opgelet! hier enkel voor [0,10].
    // Probeer met constanten de oefening ook te maken voor het interval
    // [MIN,MAX].
    // Pas OVERAL waar nodig aan, en PROBEER UIT MET HET INTERVAL [-5,7] !!

    int frequentie[11];
    for(int i=0; i<11; i++){   // initialiseren van frequentietabel
        frequentie[i] = 0;
    }

    int x;
    cout << "Geef getallen op; eindig met een getal buiten [0,10]   ";
    cin >> x;
    while( x>=0  && x<=10 ){
        frequentie[x]++;
        cin >> x;
    }

    cout<<endl;
    for(int i=0; i<11; i++){
        cout<<"Het getal "<<setw(3)<<i<<" kwam "<<frequentie[i]<<" keer voor. "<<endl;
    }
    return 0;
}
```

Oefening 3

Wil je een tabel doorlopen op zoek naar iets, dan gebruik je *altijd* een while-lus met *dubbele* voorwaarde. De *eerste* voorwaarde zorgt ervoor dat je niet uit de tabel valt, de *tweede* zorgt dat je stopt indien je vond wat je zocht. (Respecteer je die volgorde niet, dan zal je ooit wel eens een geheugenplaats raadplegen die buiten de tabel valt; dat leidt tot ongewenst gedrag at runtime.)
Na de lus vraag je je dan af waarom je uit de while-lus gevlogen bent, en dit doe je aan de hand van de index: *dat* is immers het item dat je in de lus veranderde. Testen op het tabelelement heeft geen zin (en is, weerom, gevaarlijk).

Oefening 4

Vergelijk de code hieronder met die van jou. Heb je andere structuren gebruikt? Zo ja, ziet je code er even symmetrisch uit als deze? (Want uiteindelijk gaat het om een symmetrisch probleem!) Tel ook het aantal testen dat je moet doen. Indien de ene tabel beduidend langer is dan de andere, doe je dan niet teveel testen?

```
int main(){

    const int I = 5;
    const int J = 4;
    const int K = I+J;   // hangt uiteraard af van I en J !!
    string tab_1[I]={"aap","mies","noot","toon","vuur"};
    string tab_2[J]={"appel","boom","citroen","druif"};
    string tab_3[K];

    int i=0, j=0, k=0;
    while( i<I && j<J ){
        if(tab_1[i] < tab_2[j]){
            tab_3[k] = tab_1[i];
            i++;
            k++;
        }
        else{
            tab_3[k] = tab_2[j];
            j++;
            k++;
        }
    }
    while( i<I ){
        tab_3[k++] = tab_1[i++];
    }
    while( j<J ){
        tab_3[k++] = tab_2[j++];
    }

    // uiteraard controleren !
    cout<<endl;
    for(int i=0; i<K; i++){
        cout<<tab_3[i]<<endl;
    }
    return 0;
}
```

Oefening 5

```
// opmerking: van zodra je pointers kent (en ingeoefend hebt),
// vervang je de paramter "string lijst[]" door "string * lijst"

void voegtoe(const string & woord, string lijst[], int & aantal,
             int maxaantal, bool stijgend){
    if(aantal<maxaantal){
        int i=aantal-1;
        while(i>=0 && (   ( stijgend && lijst[i]>woord )
                       ||(!stijgend && lijst[i]<woord ) ) ){
            lijst[i+1]=lijst[i];
            i--;
        }
        lijst[i+1]=woord;
        aantal++;
    }
}
// Opgelet: door constante controle op de waarde van 'stijgend'
// (stijgend && lijst[i]>woord of
// dalend && lijst[i]<woord)
// wordt de while-lus niet echt efficient.
// Om efficientieredenen zou je beter twee keer de while-lus schrijven,
// afhankelijk van de waarde van 'stijgend'.
// Daar kan je tegenin brengen dat je dan 'duplicated code' hebt,
// maar het is hier kiezen of delen...

bool verwijder(const string & woord, string lijst[], int & aantal, bool stijgend){
    bool gevonden;
    int i=0;
```

```
    while(i<aantal &&  (    ( stijgend && lijst[i]<woord)
                         ||(!stijgend && lijst[i]>woord) ) ){
        i++;
    }
    if(i<aantal && lijst[i]==woord){
        gevonden=true;
        for(int j=i; j<aantal-1; j++){
            lijst[j]=lijst[j+1];
        }
        aantal--;
    }
    else{
        gevonden=false;
    }
    return gevonden;
}
// zelfde opmerking als hierboven ivm constante controle op 'stijgend'

bool is_gesorteerd(const string lijst[], int aantal, bool stijgend){
    int i=0;
    while(i<aantal-1 && (   ( stijgend && lijst[i]<lijst[i+1])
                         ||(!stijgend && lijst[i]>lijst[i+1]) ) ){
        i++;
    }
    return (i==aantal-1);

    /* lelijk alternatief:
    if(i==aantal-1){
        return true;
    }
    else{
        return false;
    }*/
}

void schrijf(const string werktabel [],int aantal){
    cout<<aantal<<" :  ";
    for (int i=0; i<aantal; i++){
        cout<<werktabel[i]<<" ";
    }
}

int main(){
    const bool stijgend=false;
    const int MAX=200;
    const int STARTMAX=5;
    const string startzin[STARTMAX]={"jantje","zag","eens","pruimen","hangen"};
    const string verwijderzin[STARTMAX]={"zag","eens","eens","pruimen","eieren"};
    string werktabel[MAX];

    int aantal=0;
    for (int i=0; i<STARTMAX; i++){
        voegtoe(startzin[i],werktabel,aantal,MAX,stijgend); // stijgend sorteren
        cout<<endl<<"na toevoegen van "<<startzin[i]<<": "<<endl;
        schrijf(werktabel,aantal);
    }

    for (int i=0; i<STARTMAX; i++){
        verwijder(verwijderzin[i],werktabel,aantal,stijgend); // stijgend sorteren
        cout<<endl<<"na verwijderen van "<<verwijderzin[i]<<": "<<endl;
        schrijf(werktabel,aantal);
    }

    if(is_gesorteerd(werktabel,aantal,true)){
        cout<<endl<<"de huidige werktabel is stijgend gesorteerd";
    }
    else if(is_gesorteerd(werktabel,aantal,false)){
```

```
        cout<<endl<<"de huidige werktabel is dalend gesorteerd";
    }
    else{
        cout<<"de huidige werktabel is niet gesorteerd; stijgend noch dalend";
    }

    return 0;
}
```

2 Basishandelingen

3 Programmeren met stijl in C++

Oefening 6

Eerste fout: som niet geinitialiseerd.
Tweede fout: constante niet gebruikt waar nodig.
Echte fout: rand() tweemaal oproepen (in for-lus) wil zeggen tweemaal een ander getal (tenzij HEEL VEEL GELUK); blz 15 in de stijlbrochure waarschuwt niet direct voor deze fout, maar staat ermee in verband.

Oefening 7

1. constanten worden met hoofdletters gedeclareerd, gewone variabelen met kleine letters
2. accolades rond body van binnenste while
3. item voor binnenste while klaarzetten (dus `j` later initialiseren)
4. aantal niet initialiseren; wordt toch ingelezen
5. oneindige lus voor buitenste while: `i` vergeten incrementeren
6. buitenste lus moet een FOR-lus zijn
7. de waarde 40 mag nergens meer gebruikt worden in het programma
8. `j` misschien initialiseren op 1 ipv 0 (hoewel niet noodzakelijk!!)

```
int main(){
    const int LIJNLENGTE=40;
    int aantal;
    int j;
    cout<<"Hoeveel lijnen?"<<endl;
    cin>>aantal;
    for(int i=1;i<=aantal;i++){
        j=1;
        while(j<=i && j<=LIJNLENGTE){
            cout<<"*";
            j++;
        }
        cout<<endl;
    }
    return 0;
}
```

Oefening 8

```
int main(){
    const int AANTAL=4;
    int getal,cijfer;

    // 1. Er is geen reden om 'i' hier te declareren.
    // 2. Er is geen reden om 'som' hier al te initialiseren; moet toch telkens opnieuw gebeuren.
    // 3. Declaratie van som_extremum niet vergeten!
```

```
    int som,som_extremum=0;

    // 4. Beter splitsen over twee regels; << onder elkaar zetten.
    //    Spaties niet vergeten tussen 'negatief' en 'getal'.
    //    Na output 'endl' erbij, zodat input niet aan vraag plakt.
    // 5. Constante overal waar nodig gebruiken, OOK in output-opdracht!!
    //    (niet "100", maar AANTAL uitschrijven).

    cout<<"Geef (max "<<AANTAL<<") positieve getallen  op; ik stop zodra je een negatief "
        <<"getal ingeeft. Ik bepaal welke cijfersom het grootst is, en schrijf dit uit."<<endl;


    // 6. NOOIT aan teller prutsen binnen een for-lus;
    // 7. hier heb je (gezien de opgave) een while-lus nodig.

    //    Bij een for-lus hoort het inlezen van het getal binnen de lus;
    //    bij een while-lus hoort het klaarzetten van het getal vr de lus.

    int teller=0;
    cin>>getal;
    while(teller<AANTAL-1 && getal>0){                      //(&)
        // 8. som niet vergeten initialiseren!
        som=0;

        // 9. while (getal/=10)  : is test en toewijzing in n; niet doen!
        //    de eenheden werden op die manier zelfs overgeslagen.
        while(getal!=0){
            som   += getal%10;
            getal /= 10;
        }
        // 10. na een while(A) geen if(A): dit heeft geen enkele zin.
        if(som_extremum < som){
            som_extremum = som;
        }
        teller++;
        cin>>getal;                                         //(**)
    }

    // 11. Onderstaande code is inderdaad 'herhaling', maar kan vermeden worden
    //     door functies te gebruiken (zie latere lessen).
    //    (Indien je in plaats van deze oplossing, op regel //(**) eerst een if-test zou doen,
    //    doe je de test 'getal>0' twee keer vlak na elkaar (nl. op regel //(**) en op regel //(&)),
    //    en dat is te vermijden.)
    if(getal>0){
        som=0;
        while(getal!=0){
            som   += getal%10;
            getal /= 10;
        }
        if(som_extremum < som){
            som_extremum = som;
        }
    }

    // 12. som_extremum uitschrijven ipv som
    cout<<"De grootste cijfersom is "<<som_extremum<<endl;
    return 0;
}
```

Oefening 9

1. slechte naamgeving (som1,som2), waardoor omwisseling bij het uitschrijven kon gebeuren!
2. initialiseren van `som,som1,som2,product` vergeten
3. structuur van while-lus verkeerd: eerst item klaarzetten
4. dubbele zaken uit if/else-structuur halen
5. `i++` buiten else halen
6. uitschrijfopdrachten buiten while halen

7. getallen hoeven niet in tabel gestoken te worden; tabel overbodig
8. daardoor vervalt fout bij ingave van meer dan 20 getallen
9. nooit `cin>>getal; tabel[i]=getal;` maar meteen `cin>>tabel[i];`
10. `if(tabel[i]%2)` moet zijn `if(tabel[i]%2!=0)` of iets dergelijks
11. bereken som pas achteraf nl. `som=som1+som2` ipv dubbel werk te doen
12. gemiddelde moet komma-getal zijn
13. de variabele `i` kan beter een duidelijke naam krijgen
14. indien er direct een nul wordt ingegeven krijg je run-error (delen door 0)

```
int main(){
    int aantal=0,getal;
    int someven=0,somoneven=0,product=1;
    cin>>getal;
    while(getal!=0){
        product*=getal;
        if(getal%2!=0){
            somoneven+=getal;
        }
        else{
            someven+=getal;
        }
        aantal++;
        cin>>getal;
    }
    cout<<"Som van even getallen is "<<someven<<endl;
    cout<<"Som van oneven getallen is "<<somoneven<<endl;
    cout<<"Product van alle getallen is "<<product<<endl;
    if (aantal>0){
        cout<<"Gemiddelde van alle getallen is "<<(1.0*(someven+somoneven)/aantal)<<endl;
    }
    else{
        cout<<"Er werden geen getallen ingegeven"<<endl;
    }
    return 0;
}
```

Oefening 10

```
int main(){
   int prod,factor;
   int getal;
   cout<<"Dit programma berekent de grootste faculteit die strikt kleiner is"<<endl
       <<"dan het ingelezen getal. Dit voor alle gehele getallen strikt groter dan 1"<<endl
       <<"die je opgeeft; ik stop als je een getal <= 1 ingeeft.";
   getal=2;             // foei! geen nepwaarden ingeven om te starten in een while-lus
   while(getal>1){      // (laatste input wordt dus ook behandeld, wat tot fouten leidt)
      cin>>getal;
      prod=1;   // ontbreekt; zet waarden klaar vlak voor de while-lus; geen meters ervoor
      factor=0;         // ontbreekt; idem
      while(prod<getal){
         factor++;
         prod*=factor;
      }
      prod/=factor;
      cout<<"De grootste faculteit onder dit getal is "<<prod<<endl;
   }
   return 0;
}
```

Oefening 11

```
//er werd gekozen voor duidelijke namen (e en k zijn veranderd in eerste en laatste_oneven)
int main(){
   const int MAX=9;
   int eerste,laatste_oneven;
   int tab[MAX];
```

```
   for(int i=0;i<MAX;i++){
      cin>>tab[i];
   }
   eerste=tab[0];
   for(int i=2;i<MAX;i=i+2){
       tab[i-2]=tab[i];
   }
   int laatste_even_index,laatste_oneven_index;
   laatste_even_index=(MAX-1)/2 * 2;
   tab[laatste_even_index]=eerste;

   laatste_oneven_index=MAX/2 * 2 - 1 ;
   laatste_oneven=tab[laatste_oneven_index];
   for(int i=laatste_oneven_index;i>=3;i=i-2){
      tab[i]=tab[i-2];
   }
   tab[1]=laatste_oneven;

   for(int i=0;i<MAX;i++){
      cout<<tab[i]<<" ";
   }
return 0;
}
/*
   opmerking:
   kan misschien duidelijker door eerst volgende berekeningen te maken:
   if(MAX%2 == 0){
      laatste_even_index=MAX-2;
      laatste_oneven_index=MAX-1;
   }
   else{
      laatste_even_index=MAX-1;
      laatste_oneven_index=MAX-2;
   }
*/
```

4 Pointers: kennismaking

Oefening 12

```
i   = 7;    // ok

ip  = &7;   // rood: een constant getal heeft geen adres
jp  = &i;   // ok
*jp = j;    // groen: j is niet geinitialiseerd, dus nog niet gebruiken!
*ip = i;    // groen: ip is nog zwevende pointer, dus niet gebruiken!
ip  = &j;   // ok

&i  = ip;   // rood: een adres wijzig je niet
j=4;        // ok
(*ip)++;    // ok (4 wordt 5)
&d  = dp;   // rood: een adres wijzig je niet
*ip*= i;    // ok (5 wordt 35)

*jp=*&j;    // ok, hoewel 't korter kan (*& heffen elkaar op) (7 wordt 35)
ip++;       // groen: pointer staat niet in tabel te wijzen, dus nut is twijfelachtig
i  = ip-&i; // groen: ip en &i zijn adressen van 'onafhankelijke' geheugenplaatsen
            //         (behoren niet tot dezelfde tabel)
dp = &i;    // rood: types komen niet overeen
dp = ip;    // rood: types komen niet overeen

&dp = &&d;  // rood: een adres wijzig je niet
*ip+=1;     // ok (35 wordt 36)
*ip++;      // groen: je doet eerst "*ip;" (onzin), daarna "ip++;"
```

```
cout <<endl<<  i  <<  j  <<  d;  // probeer uit
cout <<endl<< *ip << *jp << *dp; // probeer uit
```

5 Verschil tussen lokaal en dynamisch creëren van variabelen

Oefening 13

Oefening 14

Wat er mankeert: vierkante haakjes tussen `delete` en `p`. Anders wordt enkel het eerste element van de tabel aan het geheugen vrijgegeven, en de rest blijft staan (i.e. blijft plaats innemen in het geheugen).

Oefening 15

Wat er mis gaat: je hebt nu wel vierkante haakjes tussen `delete` en `p`, maar ondertussen staat deze pointer achteraan in de tabel te wijzen. Dus het hele stuk ervoor (i.e. de hele tabel) blijft plaats innemen in het geheugen.

Oefening 16

6 Doorgeven van parameters naar (lid)functies

Oefening 17

Voeg je nog op tijd en stond `delete p` toe, dan zijn versie 1 en 6 correct. De andere versies lopen sowieso verkeerd af: een zwevende pointer toch gebruiken om gegevens te bewaren is *not done*!

Oefening 18

```
#include <iostream>
using namespace std;

// kleine wijziging aan opgave: je geeft ook mee hoeveel machten er moeten
// berekend worden

double * new_machten(double x, int aantal){
    double * p = new double[aantal];
    p[0]=1;
    for(int i=1;i<aantal;i++){
        p[i]=p[i-1]*x;
    }
    return p;
}

void bereken_machten_new(double * & p, double x, int aantal){
    p = new double[aantal];
    p[0]=1;
    for(int i=1;i<aantal;i++){
        p[i]=p[i-1]*x;
    }
}
```

```
void schrijf(const double * tabel, int aantal){
    for(int i=0;i<aantal;i++){
        cout<<" "<<tabel[i];
    }
}

int main(){
    double x;
    int aantal;
    cout<<"geef een getal op en een aantal: ";
    cin>>x>>aantal;
    double * machten_1 = new_machten(x,aantal);
    double * machten_2;
    bereken_machten_new(machten_2,x,aantal);

    cout<<endl;
    schrijf(machten_1,aantal);
    cout<<endl;
    schrijf(machten_2,aantal);

    delete [] machten_1;
    delete [] machten_2;

    return 0;
}
```

7 Gelinkte lijsten

Oefening 19

Gelinkte lijst: niet-recursieve basisversie

```
// Deze lijst is NIET recursief gedefinieerd,
// dus ook NIET recursief geimplementeerd.
// We maken voorlopig ook nog geen gebruik van een zoekfunctie;
// zoeken gebeurt nog in de verwijder-lidfunctie ipv apart.

#include <iostream>
using std::endl;
using std::ostream;
using std::cout;

typedef int T;   // type van een sleutel; te vervangen indien nodig
class Lijstknoop;

class Lijst{...};       // zie opgave
class Lijstknoop{...};  // zie opgave

//_____________IMPLEMENTATIE_____________________________

Lijst::Lijst(){
    // in testversie kan hier nog bij: cout<<endl<<"constructor opgeroepen"<<endl;
    eerste = 0;
}
Lijst::~Lijst(){
    // HEEL BELANGRIJK: in testversie ook hier toevoegen: cout<<endl<<"destructor"<<endl;
    delete eerste;
}
void Lijst::voegtoe(const T& t){
    Lijstknoop* schuiftop = eerste;  // andere versie dan op bord; maar ook ok
    eerste = new Lijstknoop(t);
    eerste->volgende = schuiftop;
}
```

```
void Lijst::verwijder(const T& t){  // enkel de eerste knoop met sleutel t wordt verwijderd
                                    // wil de gebruiker dit meerdere keren doen, dan moet
                                    // hij/zij de functie maar meermaals aanroepen.
                                    // (een kleine bouwsteen is vlotter inzetbaar dan een
                                    //  grote!!)
    if(eerste==0){
        cout<<"lijst leeg";
    }
    else if(eerste->sl==t){
        Lijstknoop* magweg=eerste;
        eerste=eerste->volgende;
        magweg->volgende = 0;  // anders.... sneeuwbaleffect (zie blz 20-21 in opgave)
                               // cout<<"ik delete nu "<<magweg->sl;
        delete magweg;
    }
    else{
        Lijstknoop* voorganger=eerste;
        Lijstknoop* magweg=eerste->volgende;
        while(magweg!=0 && magweg->sl!=t){
            voorganger=voorganger->volgende;
            magweg=magweg->volgende;
        }
        if(magweg!=0){
            voorganger->volgende = magweg->volgende;
            magweg->volgende = 0;   // anders.... sneeuwbaleffect
            delete magweg;
        }
        else{
            cout<<"sleutel "<<t<<" niet aanwezig";
        }
    }
}
int Lijst::geefaantal() const{
    int a=0;
    Lijstknoop* hier=eerste;
    while(hier!=0){
        a++;
        hier=hier->volgende;
    }
    return a;
}

void Lijst::schrijf(ostream& os) const{
    Lijstknoop* hier=eerste;
    os<<endl<<"LIJST  met"<<geefaantal()<<" knopen ";
    while(hier!=0){
        os<<hier->sl<<"  ->  ";
        hier=hier->volgende;
    }
    os<<"X  ";  // zo zie je tenminste dat de lijst evt. leeg is!
}
//_____________________________________________________
Lijstknoop::Lijstknoop(){
    volgende = 0;
}
Lijstknoop::Lijstknoop(const T& t){
    sl=t;
    volgende = 0;
}
Lijstknoop::~Lijstknoop(){
    delete volgende;
}

//_________________VOORBEELD VAN TESTPROGRAMMA_________________________

int main(){
    Lijst l;        cout<<l;  // gebruik dit ipv l.schrijf(cout);
```

```
    l.voegtoe(1);   cout<<l;
    l.voegtoe(2);   cout<<l;
    l.verwijder(2); cout<<l;
    l.verwijder(4); cout<<l;
    l.verwijder(1); cout<<l;
    l.verwijder(5); cout<<l;
    l.voegtoe(14);  cout<<l;
    l.voegtoe(16);  cout<<l;
    l.verwijder(16);cout<<l;
    l.verwijder(14);cout<<l;
    l.verwijder(12);cout<<l;
    l.verwijder(12);cout<<l;
    return 0;
}
// OPGELET!
// bovenstaande code (main-gedeelte) is korter te schrijven adhv
// een for-lus, waarbij je tabellen met gegevens overloopt
// bvb    T voegtoe[4]={1,2,14,16};
//        T verwijder[8]={2,4,1,5,16,14,12,12};
// maar steek hier niet teveel tijd in:
// de aandacht moet gaan naar de implementatie van de lidfuncties!!
```

Oefening 20

Gelinkte lijst: recursieve basisversie

```
// Deze lijst is WEL recursief gedefinieerd,
// dus OOK recursief geimplementeerd waar mogelijk.
// We maken voorlopig nog geen gebruik van een zoekfunctie;
// zoeken gebeurt nog in de verwijder-lidfunctie ipv apart.

#include <iostream>
using std::endl;
using std::ostream;
using std::cout;

typedef int T;   // type van een sleutel; te vervangen indien nodig

class Lijstknoop;

class Lijst{

    public:
        Lijst();       // constructor: maak een nieuwe lijst aan
        ~Lijst();      // destructor: geef alles wat je met new aanmaakte,
                       // hier weer vrij aan het geheugen.

        int geefaantal()const;
        void voegtoe (const T &);
        void verwijder (const T &);

        friend ostream& operator<< (ostream& os, const Lijst& l){
            l.schrijf(os);
            return os;
        }

    protected:
        Lijstknoop * k;
        void schrijf(ostream & os) const;
};

class Lijstknoop{
    friend class Lijst;
    private:
        T sl;
```

```
        Lijst volgende;    // enige punt waarop interface recursief/niet-recursief verschilt
        Lijstknoop();
        ~Lijstknoop();
        Lijstknoop(const T&);
};

//_____________IMPLEMENTATIE_____________________________

Lijst::Lijst(){
    k=0;
}
Lijst::~Lijst(){
    delete k;
}
void Lijst::voegtoe(const T& t){
    Lijstknoop* nieuw=new Lijstknoop(t);
    nieuw->volgende.k=k;
    k=nieuw;
}
void Lijst::verwijder(const T& t){
    if(k==0){
        cout<<"lijst is leeg";
    }
    else if(k->sl==t){
        Lijstknoop* magweg=k;
        k=k->volgende.k;
        magweg->volgende.k=0;
        delete magweg;
    }
    else k->volgende.verwijder(t);   // hier zie je de sterkte van recursief programmeren
}

int Lijst::geefaantal()const{
    if (k==0) return 0;
    else return k->volgende.geefaantal()+1;
}

void Lijst::schrijf(ostream& os) const{
//+/ merk op: deze code is identiek aan de vorige versie (niet-recursief)
//+/ probeer deze nu zelf recursief te maken.
//+/ Het einde van een lijst duid je nog altijd aan met 'X'
    os<<endl<<"LIJST met "<<geefaantal()<<" elementen ";
    Lijstknoop* hier=k;
    while(hier!=0){
        os<<hier->sl<<"  ->  ";
        hier=hier->volgende.k;
    }
    os<<"X  ";
}
//_________________________________________________
Lijstknoop::Lijstknoop(){
    // NIHIL in te schrijven !
}
Lijstknoop::Lijstknoop(const T& t){
    sl=t;
}
Lijstknoop::~Lijstknoop(){
    // NIHIL in te schrijven !
}

//_________________VOORBEELD VAN TESTPROGRAMMA_____________________

int main(){
    Lijst l;        cout<<l;
    l.voegtoe(1);   cout<<l;
    l.voegtoe(2);   cout<<l;
    l.voegtoe(3);   cout<<l;
```

```
    l.verwijder(2); cout<<l;
    l.verwijder(4); cout<<l;
    l.verwijder(3); cout<<l;
    l.verwijder(1); cout<<l;
    l.verwijder(5); cout<<l; // zorg dat hele lijst leeg is; en voeg
    l.voegtoe(10);  cout<<l; // dan weer knopen toe
    l.voegtoe(12);  cout<<l;
    l.voegtoe(14);  cout<<l;
    l.voegtoe(16);  cout<<l;
    l.verwijder(16);cout<<l;
    l.verwijder(14);cout<<l;
    l.verwijder(12);cout<<l;
    l.verwijder(12);cout<<l;
}
```

Gelinkte lijst: niet-recursieve uitgebreide versie
Oefening 21

Gelinkte lijst: recursieve uitgebreide versie
Oefening 22

Dubbelgelinkte lijst
Oefening 23

8 Constructor en destructor

9 Gelinkte lijsten: gevorderde features

Oefening 24

Als je een element in een vector (bij)steekt, maak je een copie van dit element. Idem voor een queue, stack, set,...

Oefening 25

Als je zaken copieert, moet je je afvragen wat die 'zaken' precies zijn. Zijn het pointers (of zijn er pointers in het spel)? Dan heb je – default – een probleem met ondiepe copieën. (Zoek meer informatie op indien nodig: *deep copy* versus *shallow copy*.)

Oefening 26

Wat er mis gaat: de code `v.push_back(l)` (regel 12 in main) maakt een copie van de `Lijst l`. Een `Lijst` heeft een pointer als datalid - dus de default copyconstructor levert een ondiepe copie af. Zorg dus zelf voor de implementatie van de copyconstructor.

```
Lijst::Lijst(const Lijst & origineel){
    // geen delete this->k bij copyCONSTRUCTOR;
    // want this->k kan toch nog geen waarde (!=0) hebben.
    if(origineel.k!=0){
        // cout<<endl<<"copyconstructor lijst "<<origineel.k->sl<<endl; // cout in testfase
        k=new Lijstknoop(*(origineel.k));
    }
    else k=0;
}
```

De default copyconstructor van een `Knoop` hoef je niet te wijzigen: een `Knoop` heeft geen enkele pointer als datalid.

Oefening 27

Als je een lijstje `l` bijsteekt in de vector aan de hand van de functie `push_back`, worden de reeds aanwezige lijsten verplaatst naar een andere, langere vector (i.e. ze worden gecopieerd en vernietigd). Dat kan je volgen als je voldoende uitschreef in constructor, destructor en copyconstructor. De functie `push_back` doet dus redelijk wat werk, omdat de vector telkens met een eenheid aangroeit. (Is dit werkelijk zo? Controleer! Misschien verdubbelt de vector altijd van grootte? Misschien is dit compilerafhankelijk? Indien de vector altijd verdubbelt, zal het copieerwerk gaandeweg minder worden. Maar het is er wel!)

Los dit probleem op door de vector van in het begin voldoende groot te maken (eventueel doorloop je het bestand eerst om het aantal sprookjes te tellen?), en dan de toewijzingsoperator te gebruiken: `v[i]=l`. Merk op: deze assignment operator moet je dan ook wel implementeren, want de default `operator=` zal – weerom – een ondiepe copie afleveren.

```
Lijst& Lijst::operator=(const Lijst & origineel){
    if(this!=&origineel){
        delete k;  // eerst opkuisen van lijstje waar this
                   // naar verwijst
    }
    if(origineel.k==0) k=0;
    else{
        k=new Lijstknoop(*(origineel.k)); // rest gaat via defaultcopyconstructor v.e.
            // Lijstknoop (omdat die geen pointers bevat, is hier geen gevaar op fouten)
    }
    return *this;        // verplicht;  om code 'l1=l2=l3' toe te laten
}
```