UNIVERSITEIT GENT

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

LABO
Programmeren in C en C++
Oefeningenbundel

Leen Brouns

Helga Naessens

Wim Van den Breen

Opleiding Industrieel Ingenieur Informatica / Elektronica / Automatisering

september 2017

Doelstelling van de labo's C en C++

Met het aanbieden van de labo's C en C++ willen we inhoudelijk volgende zaken op een rijtje krijgen:

1. **inzicht in het verschil tussen C /C++ en Java**
   Zodat je op het juiste moment de juiste redenering en syntax gebruikt. Daarom zullen we je (voor dit ene vak) ook aanmoedigen om af en toe zonder IDE te werken: veel oefenen en zélf de code intikken laat je bewuster omgaan met de verschillen tussen Java en C /C++ .

2. **een reflexmatige voorkeur voor leesbare en efficiënte code**
   Je krijgt in deze cursus heel wat programmeeropgaven. Uiteraard gaan we er vanuit dat afgewerkte code de gevraagde output levert. Dat is echter nog maar de startvoorwaarde. Daarnaast is ook leesbaarheid en efficiëntie zeer belangrijk. Tip: vergelijk onze oplossingen altijd kritisch met je eigen code. Zit er een verschil in leesbaarheid en/of efficiëntie? Leer daarvan — of laat ons weten wat beter kan in de voorbeeldoplossing.

3. **een zekere vlotheid in je basishandelingen**
   Bepaalde programmaonderdelen, zoals aanbieden van een keuzemenu, (handmatig) zoeken in een tabel, bewerkingen op cijfers van gehele getallen,... komen dikwijls voor. Toch zeker als je met low-level-talen werkt. Even dikwijls zien we echter teveel ballast verschijnen in de geproduceerde code: teveel hulpvariabelen, overbodige testen, storende bewerkingen tussendoor. We geven je enkele stijltips, en sommen op welke basishandelingen je vlot uit je mouw moet kunnen schudden. (Dit staat uiteraard los van de gebruikte programmeertaal, maar we hebben hier de gelegenheid enkele van onze aandachtspunten aan jullie voor te stellen.)

4. **een goed begrip van het hot topic in C(++) : pointers**
   Dit komt jullie nog van pas in andere vakken (o.a. netwerkprogrammatie), daarom schakelen we in de oefeningen vrij snel pointers in.

5. **feeling met de 'low level' features in C**
   Hier denken we onder andere aan het gebruik van `char*` in plaats van `string`. Even doorbijten in het begin — het is écht niet gedateerd, en nog altijd nuttig.

6. **een gepast gebruik van de techniek van het recursief programmeren**
   (zowel in interface als functie-opbouw), met afweging van de voor- en nadelen ten opzichte van niet-recursief programmeren.

7. **een eerste overwinning op C++11**
   zodat jullie zelf verder kunnen in de *New Standard War*, die in *C++Lands* wordt uitgevochten. Zie voor de roadmap hiervan `http://fearlesscoder.blogspot.be/2012/01/c11-lands.html`.

8. ...

Meer algemeen streven we naar

1. **een denk- in plaats van tik-reflex**
   Vooral bij het gebruik van pointers, gelinkte lijsten, boomstructuren, pointers naar pointers,... is het van belang dat je de zaken al bij de eerste poging juist op papier zet. (Jawel, PAPIER, uitvinding van voor onze jaartelling, maar sedertdien o zo geduldig.) Begin je gehaast aan een kladpoging op je scherm, en sla je de bal of de pointer mis, dan kan elke kleine aanpassing (poging tot verbetering) je verder af drijven van het juiste pad. Dit straatje zonder eind kan zeer frustrerend werken, en 'tabula rasa' maken is dan dikwijls het enige aangewezen hulpmiddel.
   Is de oefening wat groter, en wil je aan alles tegelijk beginnen? Maak eerst een gerangschikt (!) to-do-lijstje dat je tijdens de loop van de opdracht kan afvinken. Dat vraagt soms dat je procedures of functies voorlopig 'schetst' (bvb. een hardgecodeerde waarde laat teruggeven, in afwachting van betere implementatie).

2. **propere manieren wat communicatie betreft**
   Hulp nodig tijdens een labo? Let op hoe je die vraagt. De zinsnede *Meneer/mevrouw, het werkt niet!* is geen vraag, maar een vaststelling waarop je lesgever enkel instemmend kan knikken. Zorg voor

   - een duidelijke omschrijving van de context
   - een stukje code
   - het probleem dat zich voordoet (compileerfout, error at runtime, onverwachte resultaten,...)
   - wat je eventueel zelf al probeerde

   We helpen je graag bij het juist formuleren van je vraag — dikwijls vind je hierdoor ook zelf al de oplossing.

3. **een goede inschatting van je eigen vorderingen en mogelijkheden**
   Voor de reguliere bachelorstudenten onder jullie: allicht hebben jullie nog een pak programmeerervaring op te doen. Op twee fronten tegelijk: Java én C(++) . Probeer gelijke tred te houden in beide vakken, en een speekmedaille af en toe kan alleen maar deugd doen!
   Voor de schakelstudenten: dit jaar wordt er veel van jullie verwacht — veel algemene vakken die na lange tijd weer op de agenda staan, ernstig voorbereid naar alle labo's komen, eventuele leemtes tussen vooropleiding en nieuwe leerstof zelf opvullen. Je zou voor minder het overzicht verliezen, of jezelf foutief inschatten. Ook hier: houd de pas erin, en wees fier op elk bereikt deelresultaat.
   Voor beide groepen: jullie krijgen alvast enkele praktische hulpmiddelen — om te beginnen elke week twee volledig uitgesponnen theorielessen, in het labo een opeenvolging opdrachten van opbouwende moeilijkheidsgraad, geregelde feedback van de aanwezige docenten, oplossingen die je kritisch naast je eigen code dient te leggen. Doe je voordeel met de aangeboden hulp om zo vlot mogelijk de eindmeet te halen.

4. **een zicht op jullie kijk op de zaak**
   Aarzel dus niet om op- of aanmerkingen door te geven. IEDEREEN moet mee zijn met deze leerstof!
   Voor de schakelstudenten: ben je niet mee met deze leerstof, dan is er geen beginnen aan in het tweede semester (labo bij de cursus Algoritmen I).
   Voor de bachelorstudenten: jullie hebben bovendien nog een 'gap' van 12 maanden te overbruggen voor jullie aan de cursus Algoritmen I beginnen. Zorg dus dat de leerstof goed verankerd is!

Veel succes!

Software

Voor dit labo kan je kiezen: lokaal werken (dat kan op eigen laptop of op de labotoestellen in lokalen B2.031 en B2.035) of via Athena. Zoek zelf uit wat voor jou het snelst/handigst werkt.

Via Athena

Lees aandachtig hoe je Athena best gebruikt op `www.helpdesk.ugent.be/athena/gebruik.php`. Zo kan je software gebruiken zonder die zelf te installeren.

Je beschikt over een Netwerkshare die je via Athena overal kan raadplegen (lees het onderdeel `Centrale Schijfruimte`). Nadat Athena opgestart is kan je met `File Explorer` een verkenner openen vanuit Athena, waar je ook toegang hebt tot de lokale computer. Op je centrale schijfruimte (of netwerkdrive, `H`-drive) staat alles veilig, er worden backups gemaakt, en je kan het van overal terug bereiken.

Toepassingen die je in Athena start werken veel sneller als je de bestanden ook opslaat op de netwerkdrive.

Open in het labo Athena zodat je alles kan uitproberen; voeg `Dev-C++` en `Command Shell` (beide onder `Academic/Development` te vinden) alvast toe aan `MyAthena`.

Maak op je `H`-drive een map aan voor deze cursus, en een submap voor de eerste reeks. Dan kan je starten. Gezien we in deze cursus vooral focussen op korte oefeningen, is het niet nodig om telkens een nieuw project aan te maken. Allicht heb je aan aparte bestanden genoeg — in `Dev-C++` raden we het gebruik van projecten trouwens af!

Lokaal

Wie kiest om lokaal te werken op de labotoestellen in B2.031 en B2.035, moet weten dat de U-drive in onbruik is, en je dus de UGent-cloud moet mounten (icoon met huisje op je bureaublad) zodat je op je H-drive kan werken. Het mounten laat je toe om de lokaal geïnstalleerde software te gebruiken (snelkoppeling Dev-C++ op je bureaublad).

Wie op een eigen Windows-toestel werkt, downloadt de laatste versie van Orwell Dev-Cpp op `http://sourceforge.net/projects/orwelldevcpp/`.

Instellingen Dev-C++

Om je bladschikking in `Dev-C++` deftig te krijgen, pas je volgende zaken aan: onder `Tools - Editor Options - tabblad General` vink je (`SmartTabs` uit en) `Use Tab Character` aan.

Je moet ook de juiste compileropties aan- of afvinken. Ga daarvoor in de menubalk naar `Tools - Compiler Options`. Voor C -programma's komt er in het tabblad `General` onder de compileropties de optie `-pedantic` terwijl de linkeropties uitgevinkt moeten worden. Voor C++ -programma's komt er onder de compileropties de optie `-std=c++11` (met een kleine c) en de linkeropties vink je aan.

Nog een weetje: via Athena staan de instellingen bij de start misschien op `qwerty`-klavier. Verander dit met `Alt-Shift`.

REEKS 1

Kennismaking met C
main(), tabellen en eenvoudige functies / procedures

Oefening 1

Schrijf een programma dat volgende tekst op het scherm brengt (delay bij uitschrijven van de verschillende getallen is niet nodig). (En wat als we laten aftellen vanaf 100? Heb je de output `10 9 8 7 6 5 4 3 2 1` hardgecodeerd? Niet doen!)

```
Hello world!
10 9 8 7 6 5 4 3 2 1
START
```

Oefening 2

Schrijf een programma dat alle (gehele) getallen van 0 tot en met 64 uitschrijft. Per regel komt zowel octale, decimale, als hexadecimale voorstelling van één getal. Zorg ervoor dat de getallen rechts gealligneerd zijn, dus zo:

```
  0    0    0
  1    1    1
  2    2    2
  3    3    3
  4    4    4
  5    5    5
  6    6    6
  7    7    7
 10    8    8
 11    9    9
 12   10    a
 13   11    b
 14   12    c
 15   13    d
 16   14    e
 17   15    f
 20   16   10
 21   17   11
 ...
 77   63   3f
100   64   40
```

Oefening 3

Gegeven volgend programmafragment, als oplossing voor oefening 1. Dit levert de gevraagde output. Het levert echter op een examen geen punten op. Waarom niet?

```
    int i;
    for(i=10; i>0; i--){
        if(i==10){
            printf("Hello world!\n");
        }
        printf("%d ",i);
        if(i==1){
            printf("\nSTART");
        }
    }
```

Oefening 4

Gegeven de opgave *schrijf alle machten van 2 (beginnend bij $2^0 = 1$), kleiner dan 10.000 uit.* Onderstaande code is foutief. Geef aan hoe je op zicht ziet dat er iets loos is.

```
#include <stdio.h>
int main(){
    int macht = 1;
    while(macht < 10000){
        macht *= 2;
        printf("%d ",macht);
    }
    return 0;
}
```

Oefening 5

Deze code levert, in tegenstelling tot vorige oefening, wél de gevraagde output. Toch levert deze amper meer punten op dan de vorige oplossing. Waarom?

```
#include <stdio.h>
int main(){
    int macht = 1;
    int i;
    for(i=0; i<20; i++){
        printf("%d ",macht);
        macht *= 2;
        if(macht > 10000){
            break;
        }
    }
    return 0;
}
```

Oefening 6

Bij het berekenen van de sinus van een gegeven hoek, gebruikt je computer of zakrekenmachine onderstaande reeksontwikkeling.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = \frac{1}{1!}x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots$$

1. Schrijf een programma dat de sinus berekent van 0.23 radialen. Opgelet: we kijken naar efficiëntie van je berekeningen! Vergelijk dit met het resultaat van de ingebouwde sinusfunctie uit de bibliotheek `math.h`.
2. Pas het programma aan. In plaats van de sinus van 0.23 radialen, bereken je de sinus van een getal dat (door het programma) willekeurig gekozen wordt in het interval $[-3.200, 3.200[$. Het getal bevat dus drie beduidende decimalen. Gebruik de methode `rand`; voor meer informatie zoek je online.
3. De derde versie die je schrijft, vraagt aan de gebruiker een (reëel) getal, waarna er opnieuw twee sinuswaarden berekend worden: via de zelfgeschreven reeksontwikkeling en via de ingebouwde sinusfunctie van `math.h`.
4. Na theorieles 2 kan je de berekening van de reeksontwikkeling in een functie verpakken; geef deze de naam `mijn_sinus`. Pas ook het hoofdprogramma aan.

Het efficiënt implementeren van een reeksontwikkeling is niet evident. Vergelijk bijvoorbeeld het aantal berekeningen dat jouw code uitvoert met het aantal berekeningen van de voorbeeldoplossing. Heb je graag nog wat extra oefeningen op reeksontwikkelingen, dan vind je er op `https://nl.wikipedia.org/wiki/Taylorreeks` nog een hele resem (zie 'Ontwikkelingen'), die bovendien eenvoudig te controleren zijn. Zo kan je bijvoorbeeld de uitkomst die jouw zelfgeprogrammeerde reeksontwikkeling

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{n+1} x^{n+1} = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \dots$$

van `ln(1+x)` geeft, vergelijken met de uitkomst van `ln(1+x)`. Kies in dit geval `x` dan wel tussen -1 en 1 zoals de voorwaarde eist, en zoek uit hoe `ln` in C-taal geschreven wordt.

Oefening 7

Schrijf een programma dat aan de gebruiker vijf positieve gehele getallen vraagt. Geeft de gebruiker echter een negatief getal in, dan stopt het programma met getallen opvragen. Nadien schrijft het programma uit of de gebruiker inderdaad vijf positieve gehele getallen opgaf. Tot slot wordt de som van de ingegeven positieve gehele getallen uitgeschreven (ook als er niet genoeg ingegeven werden). Test uit. Wat met de getallenreeks `1 2 3 4 -5`?

Vanaf de volgende oefeningen heb je theorieles 2 nodig: er wordt gebruik gemaakt van functies (zowel implementatie als oproepen van functies) en van recursie.

Oefening 8

1. Schrijf een functie `cijfersom(x)` die van een gegeven geheel getal `x` de som van de cijfers berekent. Zo is `cijfersom(12345)` gelijk aan 15. Doe dit zonder bewerkingen op karaktertekens, gebruik enkel het type `int` (en wat wiskunde uit de lagere graad).
2. Gebruik deze functie in de functie `cijfersom_herhaald(x)`; deze blijft de som van de cijfers berekenen tot een getal kleiner dan 10 bekomen wordt. Zo is `cijfersom_herhaald(12345)` gelijk aan 6.
3. Schrijf een hoofdprogramma dat 10 strikt positieve gehele getallen inleest. (Indien een getal niet aan de voorwaarden voldoet, dan blijft het programma vragen naar een strikt positief geheel getal tot het gegeven wordt.) Bij elk positief geheel getal dat wordt ingegeven, wordt meteen geantwoord met de cijfersom.

4. Maak een recursieve versie `cijfersom_rec(x)` die hetzelfde doet als `cijfersom_herhaald(x)`. Test uit in het hoofdprogramma dat je als volgt aanpast: beantwoord elke input van de gebruiker met drie zaken, te weten (1) de cijfersom berekend met `cijfersom_herhaald(x)`, (2) de cijfersom berekend met `cijfersom_rec(x)` en (3) de uitkomst (ok / niet ok) van de vergelijking van deze beide cijfersommen.

Oefening 9

Schrijf een functie `faculteit(x)` die de faculteit van een gegeven geheel getal `x` berekent.
Schrijf een procedure `schrijf_faculteit(x)` die de faculteit van een gegeven geheel getal `x` uitschrijft.
Roep deze procedure op voor het getal 5. Daarna doe je dit voor alle getallen van 0 (0!=1) tot en met 40. Wat merk je? (Probleem dat zich stelt hoef je niet op te lossen, enkel te constateren.)

Oefening 10

Om de grootste gemende deler (ggd) van twee getallen te berekenen, werd je allicht aangeleerd om de twee getallen eerst te ontbinden in priemfactoren, om dan de gemeenschappelijke factoren te ontdekken. Het kan ook anders, met het algoritme van Euclides dat gebruik maakt van volgende gelijkheid:

```
ggd(a , b) = ggd(b , a mod b);
```

De uitdrukking `a mod b` lees je als 'a modulo b' en stelt de rest van `a` bij deling door `b` voor. In C (en C++) gebruik je de notatie `%` in plaats van `mod`. Het algoritme van Euclides vervangt de getallen `a` en `b` (herhaaldelijk) door de getallen `b` en `a mod b`. Indien `a>b`, zullen `b` en `a mod b` kleiner zijn dan `a` en `b` - en dus werd het probleem vereenvoudigd. Dit vervangen stopt van zodra een van de getallen 0 is: `ggd(a,0) = a`.

Schrijf een recursieve functie `ggd(a,b)` die de grootste gemene deler van twee gehele getallen berekent. Test uit met

```
ggd(-6,-8)==2
ggd(24,18)==6
ggd( 0,-5)==5
ggd(6,-35)==1
```

Oefening 11

Wat doet deze code? Verklaar. Na theorieles 3 kan je de code zo omvormen, dat ze ook doet wat ze belooft.

```
void wissel(int a, int b){
     int hulp;
     printf("  Bij start van de wisselprocedure hebben we a=%d en b=%d.\n",a,b);
     hulp = a;
     a = b;
     b = hulp;
     printf("  Op het einde van de wisselprocedure hebben we a=%d en b=%d.\n",a,b);
}

int main(){
    int x, y;
    x = 5;
    y = 10;

    printf("Eerst hebben we x=%d en y=%d.\n",x,y);
    wissel(x,y);
    printf("Na de wissel hebben we x=%d en y=%d.\n",x,y);

    return 0;
}
```

Deze vraag beantwoord je eerst ZONDER de code uit te proberen. Daarna kan je jouw antwoord controleren, door de code te copiëren, compileren en uit te voeren. Omdat het copiëren vanuit een .pdf-bestand de kantlijnen niet behoudt, kan je copiëren vanuit een andere bron. (Zeker nuttig als we je later langere code geven!) Op Minerva vind je een map met .tex-bestanden. Daar haal je het juiste bestand op (`opg_18_11_wissel.tex`, waarbij `18` staat voor jaargang 2017-2018 en `11` het nummer van de oefening is), en kopieer je het programma (te vinden tussen `\begin{verbatim}` en `\end{verbatim}`) in een .c-bestand.