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

september 2016

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 mee zijn met de recentste ontwikkelingen in het land van C++.
   Een 'roadmap' hiervan vind je op `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 jullie 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.
   Voor beide groepen: jullie krijgen alvast enkele praktische hulpmiddelen — om te beginnen elke week een volledig uitgesponnen theorieles, in het labo een opeenvolging opdrachten van opbouwende moeilijkheidsgraad, geregelde feedback van de aanwezige docenten (al dan niet via testjes), 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 B2.030 tot 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.

Lokaal

Wie kiest om lokaal te werken op de labotoestellen, 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. Vraag onder het tabblad `Display` nog naar de `Line Numbers`.

Je moet ook de juiste compileropties aan- of afvinken. Ga daarvoor in de menubalk naar `Tools - Compiler Options`. Voor C -programma's komt er 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?)

```
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.

Oefening 3

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 4

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 5

Wat doet deze code? Verklaar. Na theorieles 2 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=%i en b=%i.\n",a,b);
     hulp = a;
     a = b;
     b = hulp;
     printf("  Op het einde van de wisselprocedure hebben we a=%i en b=%i.\n",a,b);
}

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

    printf("Eerst hebben we x=%i en y=%i.\n",x,y);
    wissel(x,y);
    printf("Na de wissel hebben we x=%i en y=%i.\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_17_05_wissel.tex`, waarbij `17` staat voor jaargang 2016-2017 en `05` het nummer van de oefening is), en kopieer je het programma (te vinden tussen `\begin{verbatim}` en `\end{verbatim}`) in een .c-bestand.

Oefening 6

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

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

Schrijf je eigen `mijn_sinus`-functie die van een gegeven kommagetal de sinus berekent. (Staat de gevraagde parameter in graden of radialen, denk je?) Opgelet: we kijken naar efficiëntie van je berekeningen! Vergelijk ook met het resultaat van de ingebouwde sinusfunctie uit de bibliotheek `math.h`.

Oefening 7

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. Maak nu een recursieve versie `cijfersom_rec(x)` die hetzelfde doet als `cijfersom_herhaald(x)`.
4. 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, worden meteen drie zaken uitgeschreven: de cijfersom berekend met `cijfersom_herhaald(x)`, de cijfersom berekend met `cijfersom_rec(x)`, en de uitkomst (ok / niet ok) van de vergelijking van deze beide cijfersommen.

Oefening 8

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 9

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 10

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 11

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`?

REEKS 2

Arrays en pointers

Oefening 12

Schrijf voor het eerste deel van deze oefening ENKEL een hoofdprogramma, geen functies of procedures. (Houd je hier aan, of je mist de clou van de oefening.)

1. Maak in het hoofdprogramma een array aan waarin elk element een karakter is. Vul deze array bij declaratie op met volgende letters: `b f r o a u v t o`. Wil je de lengte van een gegeven array kennen in C, dan ben je gedwongen volgende (lelijke) omweg te nemen: je vraagt de geheugenruimte die de array in beslag neemt (`sizeof(array)`), en deelt dit door de geheugenruimte die één element van de array in beslag neemt (in dit geval een `char`, dus `sizeof(char)`). Schrijf nu (in het hoofdprogramma) alle karakters uit die op een even positie in de array staan.
2. Schrijf nu een procedure `schrijf_even_posities(array)` die alle karakters uitschrijft die op een even positie in de gegeven array staan. Roep deze procedure op in het hoofdprogramma en test uit. Wat merk je, en wat is daar de reden van? Los op door een extra parameter te voorzien.

Oefening 13

In deze oefening worden twee gelijkaardige functies gevraagd. Uit de opgave kan je afleiden of je de gegeven array best met indexering dan wel schuivende pointers afloopt.

Schrijf een functie met naam `index_van(...)`, die de index teruggeeft van de plaats waarop een gegeven reëel getal in een gegeven array van reële getallen gevonden wordt. Indien het getal niet aanwezig is, wordt er -1 teruggegeven. Bepaal zelf aantal en aard van de parameters.

2. Schrijf een functie met naam `plaats_van(...)`, die de plaats (pointer) teruggeeft waarop een gegeven reëel getal in een gegeven array van reële getallen gevonden wordt. Indien het getal niet aanwezig is, wordt de nullpointer teruggegeven.
3. Gebruik beide functies in een kort hoofdprogramma, om je code te testen.

Wat verandert er als de gegeven array met zekerheid geordend is? (Niet uitwerken, wel netjes formuleren.)

TIP Het door elkaar gebruiken van indexering en schuivende pointers om een array te overlopen is doorgaans geen goed idee. Je kiest voor elke toepassing de meest geschikte methode en mixt beter niet.

Oefening 14

Keer terug naar oefening 5. Hier werd een `wissel`-procedure gegeven, die echter niet deed wat ze beloofde. Pas de code aan (haal eerst het bestand `opg_17_05_wissel.tex` af, zodat je geen code hoeft over te tikken). Aan het hoofdprogramma verander je niets of zo weinig mogelijk.

Oefening 15

Schrijf een procedure `zoek_extremen(...)` die op zoek gaat naar het minimum en maximum in een array van gehele getallen. Het type en de aard van de parameters bepaal je zelf. In het hoofdprogramma zorg je voor een hardgecodeerde array, en je schrijft het minimum en maximum uit. Test kritisch!

Implementeer beide versies: met indexering en met schuivende pointers.

Extra: schrijf een recursieve versie.

Oefening 16

Schrijf een procedure die alle elementen van een gegeven tabel met lettertekens, één plaats naar links schuift. Het eerste karakter komt achteraan. Roep deze procedure drie keer aan op de tabel

```
char rij[] = {'s','a','p','a','p','p','e','l'};
```

Gebruik in de procedure eerst indexering. Daarna herschrijf je de procedure met schuivende pointers.

Aangezien je de array verschillende keren moet uitschrijven, voorzie je hiervoor uiteraard een procedure.

Oefening 17

Schrap in het rood de regels code die syntactisch niet correct zijn. Schrap in het blauw de bewerkingen die wel kloppen qua syntax, maar in het verloop van het programma zinloos en/of gevaarlijk zijn. Wat zit er uiteindelijk in de variabelen? Beantwoord alles zonder computer.

```
main(){
/* 1*/ int i=7, j;
/* 2*/ double d;
/* 3*/ int *ip, *jp, *tp;
/* 4*/ double *dp;
/* 5*/ const int * p1;

/* deze regel overslaan als theorieles 3 nog niet gepasseerd is */
/* 6*/ int * const p2 = &i;

/* 7*/ int t[25];
/* 8*/ jp = &i;
/* 9*/ j = *jp;
/*10*/ *ip = i;

/*11*/ ip = jp;
/*12*/ &i = ip;
/*13*/ (*ip)++;
/*14*/ *ip *=i;
/*15*/ ip++;

/*16*/ tp = t+2;
/*17*/ j = &t[5] - tp;
/*18*/ t++;
/*19*/ (*t)++;
/*20*/ *tp--;
```

```
/*21*/ j = (*tp)++;
/*22*/ i = *tp++;
/*23*/ p1 = ip;
/*24*/ jp = p1;
/*25*/ (*p1)--;

/*26*/ dp = &i;
/*27*/ dp = ip;
/* deze pas maken na theorieles 3 */
/*28*/ jp = p2;
/*29*/ p2 = p1;
/*30*/ *p2 += i;
}
```

Oefening 18

Wat wordt er uitgeschreven? Uiteraard beantwoord je deze vraag zonder computer.
Bijvraag: hoe zou je de hardgecodeerde bovengrenzen van de drie for-lussen kunnen vervangen door zelf de grootte van de array's te bepalen?

```
#include <stdio.h>
int main(){
    int t[6] = {0,10,20,30,40,50};
    int* pt[3];

    int i;
    for(i=0; i<3; i++){
        pt[i] = &t[2*i];
    }

    pt[1]++;
    pt[2] = pt[1];
    *pt[1] += 1;
    *pt[2] *= 2;

    int ** ppt = &pt[0];
    (*ppt)++;
    **ppt += 1;

    for(i=0; i<6; i++){
        printf("%d ",t[i]);
    }
    printf("\n");
    for(i=0; i<3; i++){
        printf("%d ",*pt[i]);
    }
    printf("\n");
    return 0;
}
```

Oefening 19

Deze oefening geldt als extra uitdaging. Te maken als je (redelijk) vlot door alle oefeningen heen fietste.

Schrijf een procedure `pivoteer(begin,na_einde,pivot)`. De drie parameters zijn pointers naar karakters in dezelfde array. De pointer `pivot` wijst naar een element tussen de elementen waar de pointers `begin` en `na_einde` naar wijzen. De procedure wisselt de elementen symmetrisch rond de pivot. Ligt de pivot niet netjes in het midden tussen de grenzen `begin` en `na_einde`, dan wordt het wisselen beperkt. Een voorbeeld: indien de elementen vanaf `begin` tot net voor `na_einde` gelijk zijn aan `a b c d e f g h` en `pivot` wijst naar de letter `c`, dan wordt dit rijtje na afloop van de procedure `e d c b a f g h`.

Schrijf en procedure `schrijf(begin,na_einde)`. De twee parameters zijn pointers naar karakters in dezelfde array. De procedure schrijft alle elementen in de array uit, te beginnen bij `begin` en eindigend net voor `na_einde`. Gebruik schuivende pointers.

Controleer met het hoofdprogramma hieronder. Is de uitvoer van je programma geen leesbare uitspraak, dan schort er nog wat aan.

```
main(){
    char tekst[] = {'b','d','?','z','g','o','e','z','e','b',
                    ' ','d','i','g','!','h','o','s','v'};
    pivoteer(tekst+7,tekst+12,tekst+9);
    schrijf(tekst+4,tekst+15);
}
```