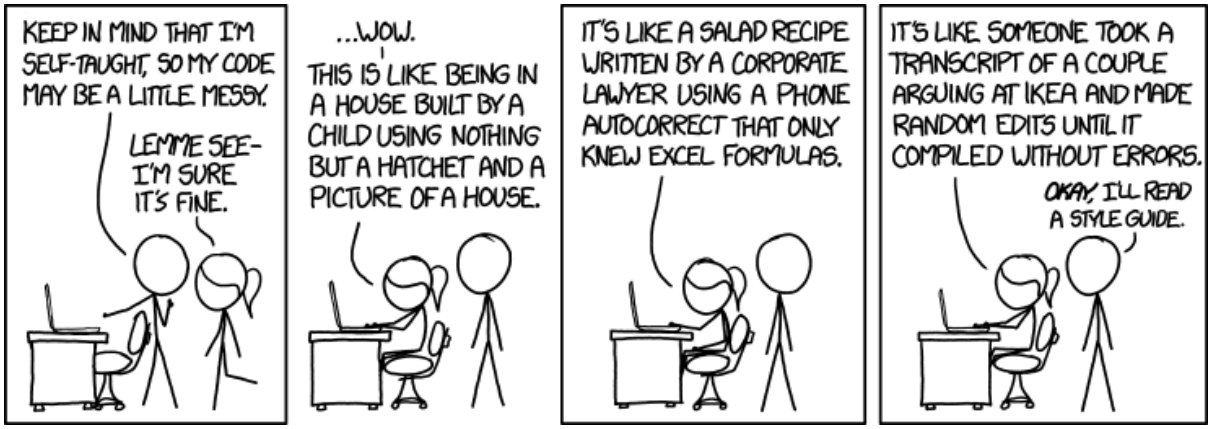
UNIVERSITEIT VAN AMSTERDAM
INLEIDING PROGRAMMEREN

OPDRACHT 1

STEVEN DE ROOIJ EN INGRID KOKKEN

Introductie



1 C en Linux

Als je nog niet gewend bent aan werken met Unix, neem dan voordat je aan de opdracht begint de “Linux introductie” en de “Tips & Tricks” op Canvas door. Deze kun je vinden in de “Aanvullende informatie”-module. Maak vervolgens een map aan voor Inleiding Programmeren. Gebruik hiervoor de terminal. In Linux (en MacOS) is dit namelijk een van de belangrijkste en snelste manieren om je computer te beheren. Gebruik de opdracht `cd` om de nieuwe map als huidige werkmap aan te duiden.

Download het bestand `opdracht1_files.tar.gz` van Canvas en sla het op in deze zelfde map, en pak hem uit met de opdracht `tar xzvf opdracht1.tar.gz`. Daardoor worden de submappen `opdracht0` en `opdracht1` gemaakt, elk met daarin een bestand dat `Makefile` heet.

2 Je eerste programma

Om te controleren of je werkomgeving in orde is, volgt nu het schrijven, compileren en uitvoeren van een testprogramma dat de tekst “Hello world!” afdrukt. Dit programma hoeft niet te worden ingeleverd.

Open je favoriete teksteditor, neem de onderstaande code over en sla het bestand op in de submap `opdracht0` onder de naam `opdracht0.c`.

```c
#include <stdio.h>

int main(void) {

    printf("Hello World!\n");

    return 0;
}
```

Compileer het programma door direct de compiler aan te roepen, zoals is getoond op het hoorcollege:

```
$ cd opdracht0
$ gcc opdracht0.c -o opdracht0
```

Je kan het programma vervolgens uitvoeren met het commando `./<naam uitvoerbaar bestand>`:

```
$ ./opdracht0
Hello World!
```

Pas nu de tekst aan naar “Hello <jouw naam>”. Je moet nu het programma opnieuw compileren, maar gebruik hiervoor deze keer de tool `make` om dat te doen. `make` maakt gebruik van de specificatie in `Makefile` om te zien hoe je programma precies moet worden gecompileerd. Het is makkelijker en veiliger om `make` te gebruiken dan om iedere keer de compiler zelf aan te roepen. Bovendien staan er soms extra specificaties in de meegeleverde Makefile over hoe je programma gecompileerd moet worden waar we bij het nakijken van uitgaan. **Gebruik dus vanaf nu altijd `make` om je programma te compileren**:

```
$ make
$ ./opdracht0
```

Let op: Als je de programmatekst niet hebt veranderd, zal `make` niet opnieuw compileren. Soms wil je toch forceren dat er opnieuw gecompileerd wordt; gebruik dan eerst `make clean` om de gecompileerde bestanden te verwijderen. Vervolgens kan je code opnieuw gecompileerd worden.

3 Voordat je begint

De code die je inlevert moet aan enkele eisen voldoen. Lees voor het inleveren van de eerste opdracht goed de stijlgids door, die vind je ook op Canvas. Hierin staan de regels en richtlijnen voor de layout van je code. Het is niet belangrijk dat je alle details uit de stijlgids meteen uit je hoofd leert, en veel

regels gaan over eigenschappen van de taal die je nu nog niet hoeft te kennen. Maar het is belangrijk dat je af en toe de stijlgids er weer eens bij neemt om in de loop van de cursus een consistente en acceptabele stijl te ontwikkelen. We worden hierover in de loop van de cursus steeds strenger, en het is dus de moeite waard om vanaf het begin jezelf een goede stijl aan te leren. Dit is belangrijk om de samenwerking met andere programmeurs te vergemakkelijken, en het helpt om leesbare code te schrijven. In de stijlgids vind je allerlei voorbeelden kunt vinden van hoe je het juist wel of juist niet moet doen.

Daarnaast moet er bovenaan ieder bestand een kort stukje commentaar staan, de zogenaamde header. Zorg dat jouw persoonlijke informatie altijd in de eerste vijf regels van jouw header staat en dat hier nooit andere informatie in komt te staan. In de header zet je:

- je naam
- je UvAnetID (studentnummer)
- je studie
- een korte omschrijving van de functionaliteit van de code in dit bestand en/of een korte omschrijving van het probleem dat aan de orde is en hoe dit is opgelost.

Hieronder staat een voorbeeld van een goede header.

```
/*
 * Naam : J. Klaassen
 * UvAnetID : 12345678
 * Studie : BSc Informatica
 *
 * Voorbeeld.c:
 * - Dit programma berekent de wortels van de vergelijking
 * ax^2 + bx + c = 0
 * Deze oplossing wordt berekend met de a-b-c formule
 * - De coefficienten moeten in de command line opgegeven worden
 * - Als de discriminant kleiner is dan 0, is er geen oplossing
 */
```

4 Opdracht 1

Deze eerste opdracht bestaat uit drie verschillende onderdelen die ieder in een apart `.c`-bestand geïmplementeerd moeten worden. Het eerste onderdeel moet worden gemaakt in het bestand `deel1.c`, het tweede in `deel2.c`, etc.

De meegeleverde `Makefile` probeert alledrie de delen te compileren als je de opdracht `make` uitvoert. Dit betekent dat `make` een waarschuwing geeft als je nog niet alle delen af hebt. Deze waarschuwingen kun je voorlopig dus negeren. Zorg ervoor dat je `deel1.c`, `deel2.c` en `deel3.c` allemaal in de map `opdracht1` hebt staan, zodat de bestanden kunnen worden gevonden.

4.1 Beoordeling

Let bij dit practicum met name op de volgende aspecten, en blijf hier voor de komende practica op letten:

- Gebruik correcte indentatie voor structuren zoals `if` en `while`, en voor de functie `main`.
- Kies behulpzame variabelenamen.
- Denk na of je raadspel (deel 3) altijd consistente feedback geeft, en een zo lang mogelijk spel garandeert.
- Belangrijk: let erop dat je uitvoer precies overeenkomt met het gegeven voorbeeld. De code die je schrijft wordt automatisch getest, en zelfs kleine afwijkingen kunnen er soms voor zorgen dat de test niet slaagt. Dit is natuurlijk vervelend, maar het is voor ons niet te doen om met de hand voor alle studenten alle tests te controleren.
- Controleer altijd de feedback van CodeGrade voordat je je definitieve versie inlevert en verbeter eventuele warnings en foutmeldingen. Zorg dat je altijd alle warnings oplost, want dit is een onderdeel van je cijfer. (Zie de rubric op Canvas.)

4.2 Deel 1: som en verschil

Neem twee getallen in gedachten, noem ze x en y.

We definiëren nu twee nieuwe getallen: respectievelijk de som en het verschil: $a = x + y$ en $b = x - y$.

Opdracht (1pt). Schrijf een programma dat de gebruiker vraagt om a en b, en dan de oorspronkelijke getallen x en y uitvoert. (Zie de voorbeelduitvoer hieronder.)

- Gebruik `code = scanf("%d", &getal)` om de een integer waarde die de gebruiker intypt in te voeren in de variabele `getal`. Vergeet het `&`-teken niet.
- Gebruik de variabele `code` om te controleren of het inlezen van een getal goed is gelukt. Zo niet, druk dan een foutboodschap af en beëindig je programma. (Gebruik de manual: de opdracht `man scanf` geeft meer informatie over hoe `scanf` precies werkt. Kijk bij “Return values”.)
- Zorg dat het programma zich **precies** zo gedraagt als in de voorbeelduitvoer hieronder, dan kunnen we het gedrag van je programma automatisch testen. Neem dit serieus: in sommige gevallen kan je werk minder gunstig worden beoordeeld als het niet door de automatische tests heen komt.

Hieronder een voorbeelduitvoer:

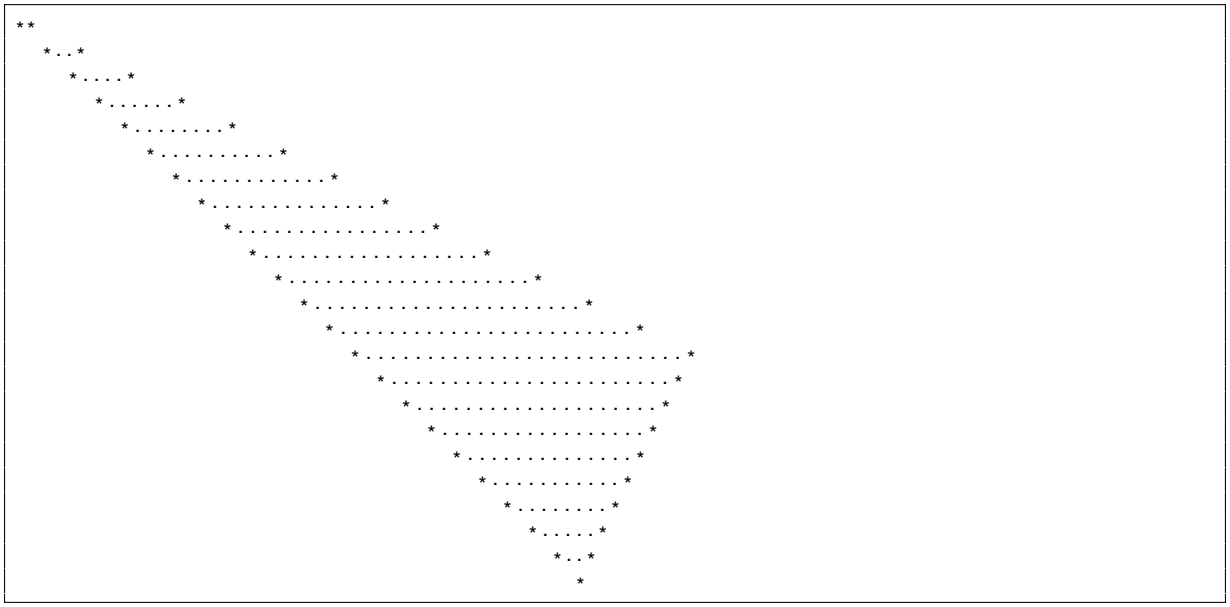
```
$ ./deel1
Geef de waarde voor a:
?
Het lezen van een getal is niet goed gelukt. Ik stop ermee.
$ ./deel1
Geef de waarde voor a:
27
Geef de waarde voor b:
7
x = 17
y = 10
```

Extra oefening (niet verplicht / niet om in te leveren): als je het leuk vindt kun je het programma aanpassen zodat het werkt als je de som en het *product* van de oorspronkelijke getallen invoert, dus $a = x + y$ en $b = xy$. Je zult merken dat programmeervaardigheid en wiskundige kennis in elkaars verlengde liggen.

4.3 Deel 2: driehoek

In deze tweede opdracht gaan we wat oefenen met lussen.

Opdracht (2pt). Schrijf een programma dat de volgende uitvoer oplevert:



Er is een beperking: je mag niet gewoon een reeks `printf`-statements achter elkaar uitvoeren; dat zou te makkelijk zijn (en het zou ook niet handig zijn als we de driehoek bijvoorbeeld heel groot zouden maken, of als we hem een beetje willen veranderen).

In plaats daarvan moet je een lus maken met `while` (of met `for` als je al weet hoe dat werkt), met code die voor elke *regel* van de figuur moet worden uitgevoerd.

Hoewel de regels allemaal van elkaar verschillen, hebben ze wel altijd dezelfde structuur: een aantal spaties, gevolgd door een asterisk, soms gevolgd door een aantal punten en nog een asterisk. Elke regel wordt direct daarna afgesloten met een newline karakter om naar de volgende regel te gaan; daarvoor kun je `printf("\n");` gebruiken.

Probeer te bedenken hoe je met *dezelfde* code in de lus telkens een net iets *andere* rij kunt afdrukken.

Tips:

- Je hebt hiervoor extra variabelen nodig. Bedenk welke informatie je nodig hebt om de rij goed te kunnen afdrukken, en maak dan variabelen die deze informatie bevatten. Geef de variabelen duidelijke namen. De namen hoeven niet lang te zijn maar denk na over wat makkelijk te lezen zou zijn voor iemand die niet in jouw hoofd kan kijken.
- Welke waardes moeten de variabelen hebben helemaal aan het begin? Hoe moeten die waardes veranderen nadat een regel is afgedrukt?
- Je kunt binnenin een lus nog weer een nieuwe lus maken. Dat heb je hier nodig: de buitenste lus voor de rijen, de binnenste lus voor de kolommen. Let op de juiste indentatie van de code binnen de lussen!

4.4 Deel 3

In deze derde opdracht maak je een aangepaste versie van het raadspel dat gedemonstreerd in het hoorcollege. De gebruiker raadt nog steeds een getal, en de computer geeft nog steeds “te hoog” of “te laag” hints. Het verschil is dat de computer nu vals speelt: hij neemt **niet** van tevoren een geheel getal in gedachten, maar probeert op oneerlijke wijze om het spel zo lang mogelijk te laten duren. De computer probeert wel te verbergen dat hij vals speelt, en mag dus alleen antwoorden geven die kloppen met eerder gegeven antwoorden. Het spel stopt wanneer de computer niet langer kan doen alsof het juiste getal nog niet is geraden zonder zichzelf tegen te spreken.

Opdracht (2pt). Implementeer het raadspel waarbij de computer vals speelt. Zorg dat de uitvoer overeenkomt met het onderstaande voorbeeld. Het programma accepteert getallen tussen de 0 en de 100 (inclusief 0 en 100 zelf). Als de gebruiker iets anders invoert moet het programma een foutmelding afdrukken en stoppen.

Dit is het eerste programma waar je echt even moet nadenken over wat het eigenlijk precies moet doen. Doe dit *voordat* je begint met programmeren. Speel eerst het spelletje eens zelf met de hand op papier. Als jij iets raadt, wat moet de computer dan zeggen? Waarom? Kun je goed beargumenteren waarom deze manier van spelen het spel zo lang mogelijk laat duren?

Bedenk dan welke informatie je algoritme nodig heeft, en denk opnieuw na over de namen van de variabelen waarin je deze informatie gaat opslaan. Hoe veranderen deze getallen als er iets wordt geraden?

Zorg er hier weer voor dat foutieve invoer afgehandeld wordt: als de gebruiker iets invoert dat niet kan worden geïnterpreteerd als een getal tussen 0 en 100, moet een foutmelding worden afgedrukt, en moet je programma stoppen.

Zorg opnieuw dat de uitvoer precies overeenkomt met onderstaande voorbeeld: (zie volgende pagina)

```
$ ./deel3
Doe een gok:
?
Ik begrijp de invoer niet.
$ ./deel3
Doe een gok:
90
Te hoog.

Doe een gok:
50
Te hoog.

Doe een gok:
10
Te laag.

Doe een gok:
30
Te hoog.

Doe een gok:
20
Te hoog.

Doe een gok:
15
Te hoog.

Doe een gok:
12
Te laag.

Doe een gok:
14
Te hoog.

Doe een gok:
13
Je hebt het geraden!
```

Soms maakt het niet uit of de computer “te hoog” of “te laag” antwoordt voor de lengte van het spel. In dat geval mag je programma beide antwoorden geven; de automatische tests keuren beide keuzes in zulke gevallen goed. Bijvoorbeeld als je in de eerste ronde `50` raadt, mag je programma zowel “te hoog” als “te laag” antwoorden.

4.5 Inleveren

Stop je bestanden in een archief en lever deze in op Canvas. Gebruik hiervoor de opdracht `make tarball`.