Oefenzittingen C: Deel 1

Informatica werktuigen

Academiejaar 2014-2015

1 Opwarmers

1. Maak een programma dat de tekst "hello world" naar het scherm uitprint. Compileer je programma met `gcc -g -Wall main.c -o main` en voer het dan uit met `./main`. Wat is het nut van al de argumenten die we aan gcc meegeven?

2. Verander je programma zodat het de getallen 1 t.e.m. 10 naar het scherm uitprint.

   Tip: in de originele C standaard kan je een variabele enkel in het begin van de functie declareren. Meer bepaald kan je geen variabele declareren in de initialisatie van een for lus. Het volgende is dus niet toegelaten:

   ```
   void function()
   {
       int result = 0;
       ...
       for (int i = 1; i <= 10; i++) {
           ...
       }
   }
   ```

   Het probleem is dat de variabele `i` niet aan het begin van de functie is gedeclareerd (in tegenstelling tot de variabele `result`). Gelukkig laat de nieuwere C99 standaard *wel* toe om eender waar een variabele te declareren (en is zo'n for lus dus ook toegelaten). Je mag de C99 standaard in alle oefeningen gebruiken. Dit doe je door de parameter `-std=c99` aan gcc mee te geven.

2 Lusstructuren

1. Schrijf een functie die twee natuurlijke getallen a en b als parameters meekrijgt, en door middel van een for lus a^b berekent en terug geeft. Je functie zal dus de volgende structuur moeten hebben:

```
int power(int a, int b)
{
        int result = ...;
        ...
        return result;
}
```

2. Roep deze functie op (in `main`) met enkele test waarden. Print het resultaat van de functie oproep uit naar het scherm.

3. Implementeer de `power` functie opnieuw, maar gebruik nu een while lus.

3 Debuggen met gdb

Een overzicht van enkele nuttige gdb commando's kan je in de slides van het hoorcollege terug vinden. Deze staan op Toledo!

3.1 Faculteit

De file `factorial.c` bevat een implementatie van de functie `factorial`. Deze functie moet de faculteit van het meegegeven getal berekenen. Er zitten echter enkele bugs in. Zet met `gdb` een breakpoint op de functie, en voer de functie dan stap voor stap uit om zo alle fouten te achterhalen.

3.2 Average

In de file `average.c` is de functie `average` gegeven. Deze wordt gebruikt om het gemiddelde tussen 1000 MiB en 2000 MiB te berekenen (uitgedrukt in bytes). Het resultaat hiervan is natuurlijk gelijk aan 1500 MiB. Dus moet het programma 1572864000 uitprinten, want de berekeningen en de output gebeuren in bytes. Maar het programma geeft een ander getal als resultaat!

Gebruik `gdb` om door de code te stappen en de fout te vinden:

- Op welke regel gaat het mis?
- Wat gaat er op deze regel mis? Waarom gebeurt dit?
- Verander de `average` functie zodat deze nu twee `double`'s als argument heeft, een `double` terug geeft, en intern ook `double`'s gebruikt om het gemiddelde te gebruiken. Zal de functie nu wel het correcte gemiddelde berekenen? Leg uit.

```c
int main()
{
    int value1 = 3;
    int value2 = 7;

    printf("Value1 = %d en value2 = %d\n", value1, value2);
    swap(&value1, &value2);
    printf("Value1 = %d en value2 = %d\n", value1, value2);

    return 0;
}
```

Figuur 1: Voorbeeld van een oproep naar de swap functie. Als swap correct is geïmplementeerd print het programma eerst "value1 = 3 en value2 = 7" uit gevolgd door "value1 = 7 en value2 = 3".

4 Parameterbinding

Schijf een functie `void swap(int* x, int* y)` die de waardes waarnaar de pointers `x` en `y` verwijst verwisselt. Een voorbeeld van een oproep naar de functie `swap` is gegeven in Fig. 1.

Waarom zou een swap functie met als prototype `void swap(int x, int y)` niet werken[1]? Of anders verwoord: waarom moet de `swap` functie met pointers werken?

5 Gelinkte Lijsten

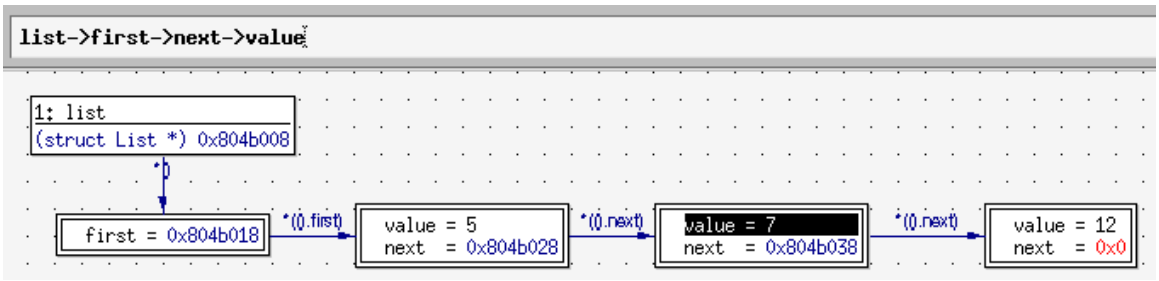
1. Tijdens de les heb je een voorbeeld implementatie van `list_get` gezien. Het nadeel aan deze implementatie is dat de gelinkte lijst tweemaal doorlopen wordt: eerst bij de oproep naar `list_length`, en daarna nog eens om het element te verwijderen. Verander de functie zodat de lijst slechts één keer doorlopen wordt.

   Tip: de originele code van `list_get` en `list_length` staat op Toledo. Zie `demos.zip` onder "Slides → Documenten les C".

2. Alice en Bob hebben `list_pop_front` geïmplementeerd, zie Fig. 4 en Fig. 5. Deze functie moet de waarde van het eerste element (als dit bestaat) toekennen aan het geheugen waarnaar de pointer `value` verwijst. Als de lijst leeg was geeft de functie 0 terug, anders geeft deze 1 terug. Is één van de oplossingen correct? Leg uit waar de fout(en) zitten.

3. Implementeer de `list_remove` functie uit het practicum.

[1] Hint: In dit geval zouden we in Fig. 1 swap moeten oproepen met `swap(value1, value2)`.



Figuur 2: Grafische weergave van de gelinkte lijst uit de les in `ddd`. De lijst bevat de waardes 5, 7, en 12.

6 Grafische front-end voor gdb

Tijdens het debuggen kan je ook gebruik maken van `ddd`, een grafische front-end voor `gdb`. Hoewel de interface er ouderwets uitziet, is dit een zeer handige tool. Om dit op je eigen computer te gebruiken moet je dit eerst installeren via:

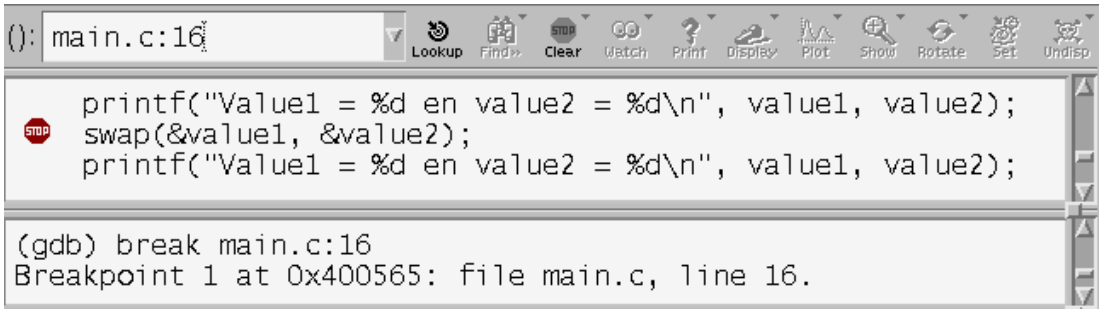
```
sudo apt-get install ddd
```

Daarna kan je `ddd` starten met je programma als argument:

```
ddd ./main
```

Vergeet niet je programma te compileren met de `-g` flag, anders heeft `ddd` geen toegang tot debug informatie. Vooral de *Data Window* in `ddd` is handig. Hiermee kan je een gelinkte lijst (of een andere datastructuur) grafisch voorstellen. Als voorbeeld toont Fig. 2 de gelinkte lijst van tijdens de les. Een goed voorbeeld waar je kan zien hoe je een gelinkte lijst kan weergeven in `ddd` kan je in dit youtube filmpje vinden: `https://www.youtube.com/watch?v=Ahf-PwHJkDc`.

Aangezien `ddd` een front-end voor `gdb` is kan je de gekende `gdb` commando's blijven gebruiken. Een breakpoint kan je bijvoorbeeld nog steeds zetten met het `break` commando (zie Fig. 3). Met `ddd` kan je ook dubbelklikken op een regel om er een breakpoint op te zetten. Gebruik echter niet enkel de grafische interface van `ddd`. Je moet nog altijd de `gdb` commando's kennen! Tijdens de verdediging van het practicum kunnen we je bijvoorbeeld vragen om in `gdb` een breakpoint op een bepaalde regel te zetten. Dit moet je dan kunnen *zonder* `ddd` te gebruiken.



Figuur 3: In ddd kan je via de gekende gdb commando's een breakpoint zetten.

```
1  int list_pop_front1(struct List* list, int* value)
2  {
3          if (list->first == NULL)
4                  return 0;
5
6          // Get value first item and free it
7          *value = list->first->value;
8          free(list->first);
9
10         // Update pointer to first element
11         list->first = list->first->next;
12
13         return 1;
14 }
```

Figuur 4: De implementatie van Alice. Is de code correct?

```
1  int list_pop_front2(struct List* list, int* value)
2  {
3          struct ListNode *pop = list->first;
4          if (pop == NULL)
5                  return 0;
6
7          value = pop->value;
8          list->first = pop->next;
9          free(pop);
10
11         return 1;
12 }
```

Figuur 5: De implementatie van Bob. Is de code correct?