Introductie Workshop C

Deze workshop is geen volledige C programmeer cursus. Een van de voorwaarden voor deelnemers om het maximale uit deze workshop te halen is dat zij al kunnen programmeren in een taal. De cursus zal zich focussen op hoe C met geheugen werkt, i.h.b. zal aandacht gegeven worden aan pointers. Programmeren leer je door te doen, daarom zal de workshop een interactief karakter hebben waar we meerdere opdrachten doen. Deze mag je individueel of samen met iemand doen, afhankelijk wat de voorkeur heeft.

Onderwerpen Dag 1

- Variabelen
- Arrays
- Functies
- Condities
- If, else, switch statements
- Loops

Onderwerpen Dag 2

- Structs
- Geheugen
- Pointers
- Linked Lists
- Linux Rootkit

Compilation

In order to run C programs they need to be compiled first. On Linux this can be done using the `gcc` compiler:

```
gcc code.c -o code
```

It can then be ran using:

```
./code
```

Include statements

Include statements worden gebruikt om header files te importeren. In deze header files staan bepaalde definities en functies, vaak van een library, die je kan gebruiken. Afhankelijk van de libraries die je wil gebruiken is het niet genoeg om alleen de header files te includen.

I.h.b. bestaan er header files die je nodig hebt om de 'standard library' te gebruiken (deze heet vaak libc, of glibc (GNU Libc)).

Hieronder een kort overzicht van veelgebruikte libc header files en de functies die erbij horen:

<stdio.h>

Deze include je met de statement: `#include <stdio.h>`. Deze header file gebruik je om de volgende functies te kunnen gebruiken:

- printf()
- fprintf()
- fread()
- fopen()
- fwrite()
- etc. (nog veel meer)

<string.h>

Deze include je met de statement: `#include <string.h>`. Deze header file gebruik je om de volgende functies te kunnen gebruiken:

- strcpy()
- strncpy()
- strcat()
- strncat()
- strdup()
- strndup()
- strtok()
- strntok()
- etc. (nog veel meer)

<stdlib.h>

Deze include je op dezelfde manier. Gebruikt voor:

- malloc()
- calloc()
- free()
- exit()
- etc. (nog veel meer)

RTFM

Voor meer info kan je de man pages gebruiken:

Bijvoorbeeld: `man 3 stdio` `man 3 stdio.h` `man 3 stdlib.h` `man 3 strcpy`

3 omdat dit de sectie over standard libraries is, als je dit weglaat, kan het zijn dat hij een gelijknamige functie uit bash pakt, zoals `printf`.

Voor meer info: `man man` of `man man man`

Variabelen

De basis variabelen in C zijn chars (characters), integers, floats, doubles, arrays, pointers en structs.

Char's en int's

char's worden gebruikt om letters (characters) in op te slaan, of beter gezegd, getallen op te slaan die corresponderen met letters volgens de asciitable, een computers kan immers enkel getallen interpreteren.

chars en integers hebben ook een unsigned variant, deze zijn `unsigned char` en `unsigned int`.

Standaard zijn, in de x86 processorarchitectuur tenminste, (unsigned) int's 32 bit en (unsigned) char's 8 bit. Het is mogelijk om grotere of kleinere varianten te krijgen met de keywords `long` en `short`. Zo is een `long long int` bijvoorbeeld 64 bit en een `short int` 16 bits.

Helaas is dit niet hetzelfde voor andere architecturen, daarom wordt tegenwoordig aangeraden om de library `<stdint.h>` te gebruiken. Deze library definieert de volgende types:

- `int8_t` Een 8 bits signed integer (ookwel bekend als een char)
- `uint8_t` Een 8 bits unsigned integer (ookwel bekend als een unsigned char)
- `int16_t` Een 16 bits singed integer
- `uint16_t` Een 16 bits unsigned integer
- `int32_t` etc.
- `uint32_t`
- `int64_t`
- `uint64_t`

floats en doubles

floats en doubles worden gebruikt om 'kommagetallen' weer te geven, zoals 3,54. floats zijn 32 bits en doubles zijn 64 bits. floats zijn sneller in gebruik, maar doubles zijn preciezer.

pointers en structs

Op deze onderwerpen komen we later terug.

arrays

Hierna aan de beurt!

typecasting

Het is mogelijk de ene variabele expliciet als een andere variabele te interpreteren, dit heet (type) casting. Een belangrijke vorm hiervan is floats als integers interpreteren en vice versa.

Arrays

Arrays zijn rijen variabelen die achter elkaar in het geheugen staan.

zie demo

Arrays hebben altijd een vaste grootte, deze is bepaald bij het declareren van de array. Doordat ze een vaste grootte hebben kan het zijn dat hij vol raakt als je er een onbekend hoeveelheid variabelen in wil opslaan. Dan kan je twee dingen doen: - Een nieuwe array aanmaken die groter is en alles over zetten. - Gebruik maken van een datastructuur zoals de linked list in plaats van een array.

Naast gewone variabelen kan je ook arrays in arrays opslaan, het is dan wel belangrijk dat de opgeslagen arrays dezelfde grootte hebben. Zulke arrays van arrays worden 2D arrays / matrices genoemd.

Op dezelfde manier kan je ook 3D arrays hebben, 4D arrays hebben, etc.

Functies in C

Functies in C hebben parameters en/of return values. Deze parameters en return values hebben een variabele type.

een functie `int sum(float a, float b)` heeft een return type `int` en twee parameters van type `float`. Op het moment dat verkeerde types worden meegegeven als parameter of returned worden dan zal er een poging gedaan worden om het type te converteren, zo wordt een float een integer door alles achter de komma weg te gooien, etc. Als dit niet lukt, crasht het programma.

Anders dan in talen als Python kan een functie in C source code alleen zien wat er eerder in het bestand voorkomt, het volgende zou dus mislukken:

```
void f()
{
    g():
}


void g()
{
    printf("hoi\n");
}
```

Met de mededeling dat `g()` niet bestaat. Het is mogelijk dit toch voor elkaar te krijgen door eerst het type van de functie te declareren:

```
void g();

void f()
{
    g();
}

void g()
{
    printf("hoi\n");
}
```

Dit is handig in situaties waar twee functies elkaar aan moeten roepen.

Condities

Condities in C zijn expressies die boolean waardes geven, True en False. In C bestaat er geen echte True of False, False in C is 0 en True in C is niet 0.

Voorbeelden van condities in C zijn: `- a < b - b == g - c > 6`

Deze geven 1 als ze waar zijn en 0 als ze niet waar zijn. Het is niet zo dat alleen 1 'Waar' is in C, elke waarde die niet 0 is, is 'Waar' in C. Conditionals worden gebruikt in if statements (en loops, later meer). if statements worden uitgevoerd als de conditie niet 0 is. Het maakt daarbij niet uit of het 1, 545 of -23 is. Je kan bijvoorbeeld zoiets doen:

```
void count(int money)
{
    if(money) {
        printf("Amount of money is not zero\n");
    }
}
```

Dit is ook handig als je met pointers gaat werken, de functie `void *malloc(size_t size)` geeft als return-waarde een pointer, een geheugenadres, als er succesvol geheugen geallocceerd kon worden of NULL (0) als er geen geheugen geallocceerd kon worden. Daardoor kan je het volgende stukje code gebruiken om te checken of er succesvol een stuk code geallocceerd is:

```
const size_t BUFSIZE = 512;
char *buffer = (char *) malloc(BUFSIZE);
if (buffer) {
    printf("Succesfully allocated memory!\n");
}
```

Naast if-statements heb je ook else statements, die werkt zoals in andere programmeertalen.

Tevens heb je else if statements:

```
if (cond) {
    some code;
}
else if (cond2) {
    some code;
}
```

Wanneer je heel veel if / else if blokken achter elkaar zou hebben is het soms beter om een switch statement te gebruiken.

Deze hebben het format:

```
switch (INTEGER) {
    case (INTEGER_VALUE):
        some_code;
        break;
    case (INTEGER_VALUE):
        other_code;
        break;
    default:
        default_code;
}
```

Let wel, dat hier in de cases integer values moeten staan, bijvoorbeeld `case 2:`. Het is niet mogelijk iets te doen als `case (n < 5):` Ook moet in het switch argument, waar boven 'INTEGER' staat een integer variabele meegegeven worden, het is niet mogelijk andere types mee te geven. In elke case moet een break statement gebruik worden, doe je dit niet, dan worden alle andere cases uitgevoerd tot er een break statement komt, of dat alle cases op zijn. De default case is een speciaal, optioneel, geval dat uitgevoerd wordt indien geen van de cases matchen.

Loops

Loops werken grotendeels hetzelfde als in andere talen. Wel is de for loop in C iets anders dan in bijvoorbeeld Python. In C heeft een for-loop de volgende structuur:

```
for( <INIT> ; <CONDITIE> ; <END OF ITERATION> ) {
    <code line 1>;
    <code line 2>;
}
```

For example:

```c
for( int i = 0; i < n ; i++ ) {
    array[i]++;
    printf("Incremented array at index %d\n", i);
}
```

This is equivalent to the following while loop:

```c
int i = 0;
while( i < n ) {
    array[i]++;
    printf("Incremented array at index %d\n", i);
    i++;
}
```

Meestal worden for loops in code op deze manier gebruikt, met een iterator i of j die een array afwerkt, maar dat hoeft niet. Zo hebben mensen complete one-liner programmas in for-loops verwerkt voor challenges, of linked list traversals met een for loop gedaan, later daar meer over, maar hier alvast een voorbeeldje daarvan:

```c
Node *curr;
for( curr = list; curr; curr = curr->next ) {
    printf("Value: %d\n", curr->data);
}
```

Assignment

Pick an assignment you want to do:

- Medium: Use a loop to reverse an array of integers.
- Hard: Use loops of choice to print all palindromes in a sentence.
- Harder: Use loops of choice to print all palindromes in a sentence (case insensitive).