Les 4 – Opdrachten

Introductie

In de workshop heb je kennis gemaakt met de basisbeginselen van de programmeertaal C. Om die basiskennis goed te laten bezinken en te kunnen toepassen is het belangrijk om daarmee te oefenen. Daar zijn deze opdrachten voor.

Je zult hopelijk merken dat de taal C meer kennis en begrip van de interne werking van computers vereist dan talen zoals C# en Java waar je eerder in de opleiding TI mee hebt gewerkt. Deze opdrachten zijn bedoeld om je te helpen daarmee vertrouwd te raken.

Instructie

Je krijgt telkens een stukje C source code als startpunt en een beschrijving van wat er geprogrammeerd moet worden. Neem die code over in je programmeeromgeving. In deze les is dat deels Visual Studio of Visual Studio Code + C compiler maar ook ESP-IDF + ESP32 board. Het is waarschijnlijk handig als je voor elke opdracht een aparte map (en evt. een apart project in Visual Studio) maakt.

Schrijf de gevraagde uitbreidingen en/of aanpassingen aan de source code. Compileer en run je code en kijk goed naar de output van zowel de compiler als jouw code.
C compilers zijn (om historische redenen) soms erg tolerant en accepteren code die eigenlijk fout is, maar geven dan wel warnings. Een warning is C is dus soms gewoon een error, **dus neem de warnings serieus**, ook als je ze in Java of C# vaak negeerde!

Doorloop de code ook eens met de debugger en check tussentijds of er gebeurt wat je verwacht. Zet breakpoints op de belangrijke plekken en laat de debugger daar dus stoppen om de waarde van variabelen te kunnen bekijken.

Het is al vaker gezegd maar we herhalen het opnieuw: het belangrijkste van de opdrachten is **niet** de juiste output maar dat wat nodig is om tot die output te komen. Wees dus kritisch op je eigen begrip van wat er gebeurt. Vraag de docent(en) om uitleg als je twijfelt of als je iets niet begrijpt. Een docent helpt je graag verder in je leerproces.

Opdracht 4.1

In de les heb je kennis gemaakt met linked list door middel van pointers. Ook heb je eerder al gehoord over standaardbibliotheken. In deze opdracht gaan we met linked lists aan de slag.

Maak een source file (b.v. linked_list.c) met onderstaande C source code:

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int val;
    struct node * next;
} node_t;

void print_list(node_t * head) {
    node_t * current = head;

    while (current != NULL) {
        printf("%d\n", current->val);
        current = current->next;
    }
}

void main(void) {
    node_t * test_list = (node_t *) malloc(sizeof(node_t));
    //Vul hier je linked list verder in
    print_list(test_list);
}
```

Compileer en run de code in VS Code (met een native compiler zoals de Visual C++ of de GCC compiler). Typ op het keyboard strings in en bekijk de uitvoer.

Noteer voor jezelf (b.v. in commentaar in de code) bij elke regel wat daar gebeurt.

Ook hier geld net als in opdracht 3.1: Waar in de code zou zo'n memory leak kunnen ontstaan als je een regel code weg zou laten? In dit geval is het risico klein want het is een kort programmaatje, maar memory leaks kunnen heel kwalijk zijn, dus let goed op dat geheugen dat gealloceerd wordt ook weer wordt vrijgegeven.

Opdracht 4.2

Maak een nieuw project en kopieer je uitwerking van opdracht 4.1:

```c
#include <stdio.h>

typedef struct node {
    int val;
    struct node * next;
} node_t;

void print_list(node_t * head) {
    node_t * current = head;

    while (current != NULL) {
        printf("%d\n", current->val);
        current = current->next;
    }
}

int pop(node_t ** head) {
    int retval = -1;
    node_t * next_node = NULL;

    if (*head == NULL) {
        return -1;
    }

    next_node = (*head)->next;
    retval = (*head)->val;
    free(*head);
    *head = next_node;

    return retval;
}

int remove_by_value(node_t ** head, int val) {
    // TODO add your code here
}

void main(void) {
    // TODO voeg hier jouw code toe uit de main() van de vorige opdracht
    <Your Code>
    // TODO zorg ervoor dat deze functie aanroep werkt.
    remove_by_value(&test_list, 3);

}
```

Vul de ontbrekende code in op basis van de aanwijzingen in de commentaren (// TODO instructies). Voeg functies toe waar dat handig is. Voor de invoer van de getalwaarden kun je `scanf( )` gebruiken.

Toon met de print_list functie aan dat het gelukt is. Waarom geef je een geheugenadres mee en geen pointer? Ga je bewering na.

Opdracht 4.3

Maak een functie pointer in c. Maak een nieuw project aan en pas de volgende code toe:

```c
#include <stdio.h>

void fun(int a)
{
    printf("Value of a is %d\n", a);
}


void main(void) {
    int a = 16;
    // TODO voeg hier jouw functiepointer declaratie toe

    // TODO zorg ervoor dat je de functiepointer aanroept en het werkt.

}
```

Noteer voor jezelf welke problemen je bent tegengekomen en hoe je die hebt opgelost. Heb je anderen kunnen helpen hierbij en/of hebben anderen jou geholpen?