Les 5 – 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 5.1

In deze opdracht ga je een implementatie van een stack datastructuur bestuderen en aanpassen.

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

```c
// Embedded Software Ontwikkeling - Opdracht 5.1
#include "teststack.h"

int main(void) {
    testStack();
    return 0;
}
```

Maak ook een source file stack.c met onderstaande C source code:

```c
#include <stdlib.h>
#include "stack.h"

#define STACK_ENTRIES 16

// This stack can hold either a char or an int in each entry
// TODO modify it so it can also hold a double in each entry
typedef union {
    char valChar;
    int valInt;
} stack_entry_t;

stack_entry_t stack[STACK_ENTRIES];
int stackSize = 0;  // Number of entries used in the stack, index of first empty entry

int stackPushInt(int value) {
    int error = STACK_OK;
    if (stackIsFull()) {    // this function must be added, see below
        error = STACK_ERROR_FULL;
    } else {
        stack[stackSize].valInt = value;
        stackSize += 1;
    }
    return error;
}

int stackPopInt(int *returnValue) {
    int error = STACK_OK;
    if (stackIsEmpty()) {   // this function must be added, see below
        error = STACK_ERROR_EMPTY;
    } else {
        stackSize -= 1;
        *returnValue = stack[stackSize].valInt;
    }
    return error;
}

// TODO add a stackIsFull() function that returns true when the stack is full

// TODO add a stackIsEmpty() function that returns true when the stack is empty

// TODO add stackPushDouble(...) and add stackPopDouble(...) functions so that we can also
// push and pop double values from the stack
// don't forget to update the header file

// end of file
```

Maak de bijbehorende header file stack.h met onderstaande C code:

```c
#ifndef STACK_H
#define STACK_H

#include <stdbool.h>

#define STACK_OK 0
#define STACK_ERROR -1
#define STACK_ERROR_FULL -10
#define STACK_ERROR_EMPTY -11

int stackPushInt(int value);
int stackPopInt(int *returnValue);
bool stackIsEmpty(void);
bool stackIsFull(void);

#endif
```

Maak een source file teststack.c met onderstaande C source code:

```c
#include <stdio.h>
#include "stack.h"
#include "teststack.h"
// TODO add header for assert statements


void testStack(void) {
    int error = STACK_OK;
    int value1 = -1;
    int value2 = 0;
    int nrValuesWritten = 0;
    int nrValuesRead = 0;

    error = stackPushInt(17);
    // TODO convert to an assert( ) statement
    if (error) {
        printf("ERROR in stackPushInt(): %i\n", error);
    }

    error = stackPopInt(&value1);
    // TODO convert to an assert( ) statement
    if (error) {
        printf("ERROR in stackPopInt():%i\n", error);
    }

    error = STACK_OK;
    value1 = 9;
    while (STACK_OK == error) {
        value1 += 3;
        error = stackPushInt(value1);
        if (error) {
            printf("Error received after %d writes\n", nrValuesWritten);
        } else {
            nrValuesWritten += 1;
        }
    }
    printf("Number of values written: %d\n", nrValuesWritten);

    error = STACK_OK;
    while (STACK_OK == error) {
        value1 -= 3;
        error = stackPopInt(&value2);
        if (error) {
            printf("Error received after %d values read\n", nrValuesRead);
        } else {
            nrValuesRead += 1;
```

```
            // TODO convert to an assert( ) statement
            if (value2 != value1) {
                printf("ERROR reading value back from stack: values inequal\n");
            }
        }
    }
    printf("Number of values read: %d\n", nrValuesRead);
    // TODO convert to an assert( ) statement
    if (nrValuesWritten != nrValuesRead) {
        printf("ERROR wrong number of values read back from stack\n");
    }

    // TODO add test code to test pushing and popping double values from the stack

}
```

Maak ook de bijbehorende header file teststack.h met onderstaande C code:

```
#ifndef TESTSTACK_H
#define TESTSTACK_H

void teststack(void);

#endif
```

Voer stap voor stap alle TODO instructies in bovenstaande code uit om tot een werkende applicatie te komen. Begin met de ontbrekende functies stackIsFull() en stackIsEmpty() in stack.c zodat de applicatie gecompileerd kan worden.

Breid de applicatie uit met het kunnen pushen en poppen van double values, zoals in de TODO instructies aangegeven. Daarvoor moet je de union aanpassen. Voeg ook testcode toe om deze nieuwe functies te testen.

Gebruik assert( ) in je testcode, en eventueel ook in je implementatie in stack.c zelf.

Opdracht 5.2

Download de code over menu's met functiepointers (uit de presentatie) van Brightspace, open die in Visual Studio en voer die uit. De code bestaat uit drie projecten die ieder iets geavanceerder zijn dan hun voorganger, zoals getoond in de presentatie in de les.

Pas een paar menuteksten aan, verander de navigatie, voeg een paar functies toe die bij knopdruk worden aangeroepen, etc. Test je wijzigingen om een gevoel te krijgen voor de werking van de code en hoe je dit kunt aanpassen.

De voorbeeldcode is nog verre van voltooid. Voer een aantal verbeteringen door zoals hieronder aangegeven:

```
/* Define the menu items */
MENU_ITEM_STRUCT menu[] = {
       { /* Menu item 0 */
              MENU_MAIN_0_ID,
              { /* New item id (Esc, Ok, Up, Down, Left, Right) */
                     MENU_MAIN_0_ID,
                     MENU_MAIN_0_ID,
                     MENU_MAIN_2_ID,
                     MENU_MAIN_1_ID,
                     MENU_MAIN_0_ID,
                     MENU_MAIN_0_ID
              },
              {
                     /* Menu item text */
                     "HOOFDMENU",
                     "Alarmen",
                     " ",
                     " "
              },
              {
                     NULL, NULL, NULL, NULL, leftPressed, rightPressed
              },
              enterItem,
              exitItem
       },
```

Vervang door -1

Dit zijn de belangrijke overgangen

a. Zorg dat entry en exit functies niet worden aangeroepen als je na een knopdruk in hetzelfde scherm blijft.

b. Vervang in de menu array de overgangen van een scherm naar zichzelf (voor knoppen die je niet naar een ander scherm sturen) door een speciale code (zoals -1), zodat de navigatie tussen schermen duidelijker te herkennen is in de menu array. Zie hieronder.

c. Uitdaging: vervang het lineair zoeken naar het item met het juiste menu ID in de menu array door een betere oplossing waarbij er niet telkens gezocht hoeft te worden. Of maak het zoeken efficiënter. Overleg met medestudenten hoe je dat zou kunnen doen.

d. Maak zelf (met de voorbeeldcode als inspiratiebron) een simulatie op PC (C console application in Visual Studio) van een menu naar eigen ontwerp. Je kunt natuurlijk het menu gebruiken dat jouw projectgroep (IMC Proftaak) voor jullie product aan het ontwerpen is.

e. Zou het mogelijk zijn om de menu data in een linked list op te slaan? Wat zouden voordelen en nadelen zijn? Overleg met medestudenten en weeg samen de voor- en nadelen af. Denk daarbij bijvoorbeeld aan geheugengebruik (stack, heap, programmageheugen) en de noodzaak voor runtime wijzigingen.