Exercises and assignments for the course DIT165 Development of Embedded systems _ Part 1

The aim with this course is to give you an overall introduction in developing of software for embedded systems. One way of getting this knowledge is to develop different programs intended for such systems. Today the most common language for embedded systems is C or C++. One reason for this is that C is “closer” to the assembly language than other language, i.e. a C-program generates less assembly instruction then other high level program languages and by that needs less of memory. Another reason is that it’s easier to write program parts including interaction with hardware and its I/O modules.

During the course you will solve a lot of exercises. The exercises are divided into Work package (WP), one for each course week. Each WP consists of different exercises to solve during corresponding week.

Group work: The aim with exercises is that you should solve them in groups of 2 to 3 student s, hand them in as assignments through the course homepage in GUL.

Groups members: You will during the spring work in several projects in different courses. In this course you will work in a group of up to three students and you can choose group members as you want. You register yourself into the project groups at the homepage in GUL for this course (DIT165). The name of the Groups is : group_<grp no 1-40> . Find it in GUL, left menu, Project groups.

Bonus points: For all assignments in this course you will receive bonus points added to the points in the written exam. The bonus points will be added to the first exam and to the two re-exams during 2017. After that the points has expired and will not be able to use. The bonus points will expire at the end of the year regardless of if you take any exam or not.

For each of the handed in and approved exercises you get a number of points from zero up to a maximum specified points. You can get a total sum of 52 points. Your total sum of points will be divided by 10 and the rounded to an integer following the usual rules. This will give you 0 - 5 bonus points to add to the points in the written exam. The written exam will have a maximum of 30 points. Pass level is 15 points.

Terms of assignments and bonus
All exercises that are submitted (possible for all in WP2 - WP6) must meet the following requirements for contribute to bonus points at final exam.
Includes a program header as below:
/ * ====================================
File name: exerc_x_y.c (or cpp)
Date: 2017-mm-dd
Group Number:
Members that contributed:
xxxxxxx xxxxx
yyyyyyy yyyyy
zzzzz zzzzzz
Demonstration code: [<Ass code 1-4> <abc>]   Important , No code no bonus !
====================================== * /

All submissions must be handed in by one of the group members via GUL and before handed in be demonstrated for your course assistant and if he/her approve the program you will get a specific unique demonstration code <xxxx> to put in to the program header.
After that you can hand in the program source code (filename.c) and its header files (filename.h).
You should only hand in one file per WP <#>. All programs for a WP should be included in one xxx.zip file. If there are any solutions containing more than two files you had to put them in a separate directory.

Work package nr 1 / Intro week

Exerc_1_ 0

Install (or test an existing) an IDE to be used when working with the exercises below and for the project in the project course running parallel with this course.

For the project course there is a proposal for using Eclips , gcc, gdb . You should install and test such an environment on some of the group's computers.

In this course we are going to develop rather small programs so it doesn't matter what IDE you use.
Some good and simple are DEV C++ or CodeLite. It is also possible to work directly with the compiler (gcc) and a simple editor (Notepad++) from the console window.

Exerc_1_ 1 : (Filename exerc_1_1.c)

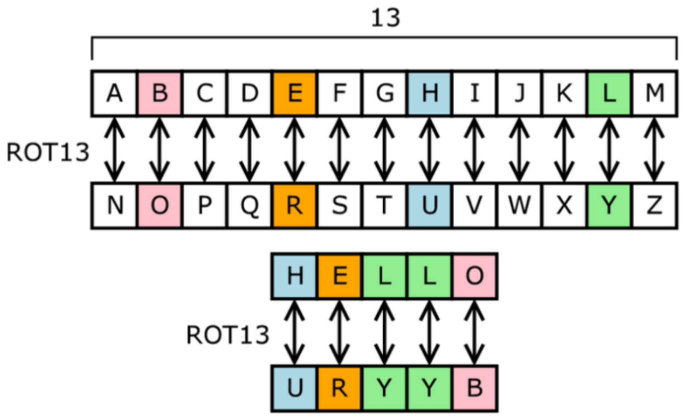
Write a program that reads in an integer number between 1 and 10 as input and prints out one of existing nine sentence in the console depending on what number was entered.
The program exits on number 0.

Exerc_1_ 2 : (Filename exerc_1_2.c)

Write a program that reads in a sentence of MAX characters and counts the number of words in it.
The number of words should then be printed out on the console window.

Exerc_1_ 3: Encryption , (Filename exerc_1_3.c)

Create a very simple encryption program. The program is based on shifts of characters in the ASCII table. In the example below, A has shifted to N, B to O, etc., that will mean 13 steps in the table (only capital letters appear but the same applies to small). The word HELLO becomes URYYB after encryption.



The user enters a text and the program prints the encrypted text. Let the program read character by character, and encrypt it as above. The program is repeated until EOF indicated the program ends. (EOF, the user enters Ctrl +z for Windows and Ctrl + d for Linux system).

Example of a test run :

HELLO (+enter)
URYYB
Banana (+enter)
Onanan
(+Ctrl-z)
(Program ends)

Exerc_1_ 4 : Guess the number , (Filname exerc_1_4.c)

You should develop a very simple game in which the computer create a random integer number between 1..100. The user then tries to guess what number it is. The program should work as below:

- The computer creates a random number
- The user guess a number
- The computer respond by printing one of :
  You have guessed xxx times and your guess is correct, or your guess is to low or to high.
- If wrong the user is asked for a new guess , this will continue until the guess is right or the number of guesses excides a value MAX_NUMBER.
- After the end of one round the user is asked for a new round or to finish.

The program should only except guessed numbers in the range of 1 ...100.

An option, but not a demand, is to secure that the program not fail (crashes) if a user by accident put in any character instead of a number.

Work package nr 2 / General C-programming (Max 12 p)

Exerc_2_ 1 : (Filename exerc_2_1.c) (1p)

Write a program that reads a string with a maximum of 20 characters from the keyboard and store it in a local string variable.
Copy the string to another string by using:
a) The library function strcpy(..)
b) A new function ***void copyString(…)*** not using any library function.

Main program ends by printing out the copied string in the console.

The program should be able to read in from the keyboard or from a textfile 'myfile.txt' containing one string of characters. You create this file with notepad or any other editor. The reading from a text file should be done by redirect the reading from command line when program execution starts as follows:
Exerc_2_1 < myfile.txt

Where Exerc_2_1 is the filename of the compiled program.

Exerc_2_ 2 : (Filename exerc_2_2.c) (1p)

Create an array of integers, array[MAX], and fill it with MAX no of random numbers. Let then the program print out the following:

The value of the label *array* (address) is: xxxxxxxxxx
First integer in the array is (array[0]) : xxxxxxxxxx
The size of an integer (number of bytes) is : xxxxxxxxx
The size of the whole array is : xxxxxxxxx

End the program by printing out the array, then double all values in the array by use of a pointer and print it out again.

Exerc_2_3 : (Filename exerc_2_3.c) (1p)

Create a program that reads two strings from the command line when it starts and checks if it is two strings and if the two strings are identical or not. Do this with and without use of library function strcmp(..). Let the program print out the result in any way.

Start from command line: *exerc_2_3 string1 string2*

Exerc_2_4 : (Filename exerc_2_4.c) (1p)

Create a program that reads in a string and determines if the string is a palindrome. A word is a palindrome if it is the same word reading from left to right or right to left. We can assume that it is a simple strings without any space character in it.

Example of palindrome : level, rotor and racecar.

Exerc_2_5: Pointer exercise (2p)

You should develop a program that calculates some statistical values for an array of integers. Among other things, the program will plot a histogram for the frequency of the different input numbers in the array.
To test it you need first to create an array of integers (*table* [MAX]) with MAX number of random numbers between 0 and MAXNUMBER. Then you should write a function that for each possible number between 0 – MAXNUMBER calculates how many times the number exists in the array. The result is then stored in a new array (*frequency* []).
Finally, you write a function that given the array *frequency []* draw a histogram as followed example:
You should use the function and its function declaration below.

Given an array table[]={ 1,2,12, 5,1,0,0,5,9,12, 0,2,3,0} thhe program will printout:

```
0   xxxx
1   xx
2   xx
3   x
5   xx
9   x
12 xx
```

Note: Numbers with frequency 0 in the array *frequency[]* is not printed out.

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

#define MAX 100
#define MAXNUMBER 20

// ------ Function declaration   ----------
void create_random( int tab[] );
void count_frequency(int tab[], int freq[] );
void draw_histogram(int freq[] );

int main ( void){
   int table[MAX], n ;
   int frequency[MAXNUMBER];
   ….

}
```

Exerc_2_ 6 (2p)

We want to use an array for a queue of positive integers. The numbers must be entered in to the queue which is done by putting them into the first vacant location in the field. (In the figure below from left) . An integer is taken out from the queue by taking the first number in the array. When an integer is taken out the other integers should be shifted one step up to left in the queue.
A vacancy in the queue is represented by the integer -1. In a full queue there is no integer -1.
When taking out an integer from a full queue , the integer -1 must be written in the last position.
Examples below show a queue with five positions.

int que [MAX] ; // MAX equal to 5 in this example

Queue from the start:	-1 -1 -1 -1 -1
Queue after you first put in the numbers 3 and then 4 and 5.	3 4 5 -1 -1
Queue after additional put in of 8 :	3 4 5 8 -1
And after taken out an integer	4 5 8 -1 -1

a) Write a function void ***initQue (int list [], int max)*** that initializes a list to initially containing max nr of vacant positions (-1).

b) Write a function ***int input (int list [] , int number, int max)*** that adds a number in the queue according to the rules , and return 1 if the number could be entered and 0 if the queue is full.

c) Write a function ***int output(int list[], int max)*** that returns the value if any to fetch or 0 if que is empty.

d) Finally write a main-program that sets up an empty queue with 5 positions and then call all functions for test of its function.

Exerc_2_ 7 (2p)

Create a program that checks an entered Swedish person number. The number is entered in the form of : 7107254786 (yymmddxxxc). The number should be read in as a string and converted to integers for year, month , day and number.
The last digit (here 6) is a control digit and is calculated from the other digits by an algorithm that you can find on the internet.

The user inputs the number, the program first checks that the number of month and day is in the right range and after that calculate and checks the control digit. The program then prints out the result and asks for a new person number. This is repeated until the user inputs a ‘q’.

The program should at least consist of the functions: main() , readPersnr(char *person), int controlDigit(const char * persnr) .

Exerc_2_ 8 (2p)

a) The game Nim works as follows: On the table is a stack of 13 coins. Two players take in turn between one and three coins from the stack. Whoever are forced to take the last coin have lost.

Your task is to develop the game Nim by using the code skeleton which is available on the course webpage in Documents / Work package nr 2 .

The program is also available in the form of an executable demo program at the same place.

• Download the code skeleton. Read the code and the comments, and try to understand how the program is structured.

• Compile and run it.

• In the function declarations you can read a description of what the function should do. All function definitions are empty (called stubs). Read through the comments and develop the functions so that the program works. NOT all at once, do ONE by ONE. Try to test each function in any way. For example use the debugger function or print out a value.

TIPS: Create a test area at the beginning of main. Test there to call and print the result of a function at a time. You can write return (0) immediately after printing, so you do not run the entire program. Remove the test parts when the test is complete.

b) When the game works well, you can only play one round. Change the program so you can play several rounds. After each round the program asks if you want to play again. Use function play again()

Work package nr 3/ Extended C-programming (Max 10 p)

For this week tasks we will develop some, a little bit more advanced C-programs for a general computer environment.

Exerc_3_ 1 (Filename exerc_3_1.c) (2p)

Implement a test program for a robot. The program asks for the robot's starting position (x, y coordinates) and then for a string of characters 'm' and 't', where **m** stands for move a step in current direction and **t** for turn of direction as below..

move:	means that the robot takes a step in the current direction.
turn:	means that the robot turns 90 degrees clockwise. Start direction is always north.

The program performs the instructions of the string one by one. When all instructions are executed robot stops and the program prints out the robot's position. The program then asks for new starting position, etc.

Implement the functions move() and turn() as two void functions and use a pointer parameters as arguments so that the function can update the robot position which is a variable in the main function (calling function).
Use enum and a record of type ROBOT as below for the robots position and direction.

```
 enum DIRECTION {N,O,S,W};

typedef struct {
      int xpos;
      int ypos;
      enum DIRECTION dir;
} ROBOT;
```

Exerc_3_ 2 (Filename exerc_3_2.c) (2p)

All sub tasks in this exercise (Searching and sorting) should be implemented and tested in the same program.

a) Write a function that given an integer **n**, an array of integers and the size of the array determines if **n** is in the array. If so the function should return the index for the first position of the number (in case of several) otherwise returns -1.
For testing the function, write a main program that tests the function with help of an array initiated in the main program as below and with a function declaration:

```
int search_number( int number, int tab[], int size);
int test [] = { 1,2,34,5,67,3,23,12,13,10};
```

b) There are a lot of ways to sort a field. For example, bubble sort which not is the fastest but easy to understand and implement. Write a sorting routine that uses the following algorithm to sort an array of integers.

• Find the minimum value in the list.
• Swap the minimum with the first in list.
• Repeat this but exclude the previous minimum on top of the list and search only in the rest of the list.

Implement the sorting function using the function declaration:

```
void sort (int number, int tab []);
```

Test the function by use of a main program and an initiated array as above. For checking purpose print out the sorted array.

Exerc_3_ 3 (Filename exerc_3_3.c) (2p+1p)

a) Write a function that creates a linked list with NUMBER records of type REGTYPE (see below). The value of the variable data is given a random number between 0 and 100.

Function declaration : **REGTYPE * random_list (void);**
Complete the program with a main program that tests the function (a first draft below).

b) Extend the program with a function with the function declaration:
REGTYP * add_first (REGTYPE * temp, int data);
That adds a new record first in the list and assign the field *numbers* the value of *data*.
The function must return a pointer to the new first entry in the list. Extend main so that this function is tested.

```
/*******************************************
    DIT1165 Program      file exerc_3_3.c                  **
    2013-01-04                                           **
*******************************************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

//#### Konstanter #####
#define MAX 5

// ##### Typedefs          ####
typedef struct q{
      int number;
      struct q *next;
      struct q *prev;
} REGTYPE;

// ##### Funcion declarations   #####

REGTYPE* random_list(void);
REGTYPE* add_first(REGTYP* temp, int data);

//###### Main program #######
int main(int argc, char *argv[])
{
   int nr=0;
   REGTYPE *akt_post , *head=NULL;

   srand( time(0));  //   Random seed
   head=random_list();
   akt_post=head;
   while( akt_post!=NULL){
        printf("\n Post nr %d : %d" , nr++, akt_post->number);
        akt_post=akt_post->next;
   }
```

```
       ………
       ………

  // --- Free of allocated memory  ---

   while((akt_post=head)!=NULL){
     head=akt_post->next;
     free(akt_post);
   }

   //------------------
   system("PAUSE");
   return 0;
  }
  // ====     End of main   ======================================
  REGTYPE* random_list(void ){
     int nr,i=0;
     REGTYP *top, *old, *item;
     ……….


     return(top);
  }

   //=========================================================
  REGTYPE* add_first(REGTYPE* temp, int data){
  // Adds a record first i list and set the field tal to data



  }
```

Exerc_3_ 4 (Filename   exerc_3_4.c) (3p)

File managements of a person register

You should write a program for manage a database of people. The database should be stored to the hard disc as a binary file. The function of the program is easiest to understand by reading the description and program skeleton below.

From the **main program** you should be able to choose between these options:

1 Create a new and delete the old file.
2 Add a new person to the file.
3 Search for a person in the file .
4 Print out all in the file.
5 Exit the program.

After entered the choice the program executes the task and returns to the menu for new choices.

1 Create a new and delete the old file.
Program creates a new file with the specified filename (fixed) and writes a first dummy record to the file and then close it.

2 Add a new person to the file.
First gives an opportunity to put in one new person to a temp record and then add this record in the end of the file.

3 Search for a person in the file.
Gives you an opportunity to search for all persons with either a specified first name or family name (by choice).The program prints out all person with that name.

4 Print out all in file.
Prints out the whole list

5 Exit the program.
Just exits the program.

```
/ * ===================================
File name: exerc_3_4.c (or cpp)
Date: 2013-mm-dd
Group Number:xxxx
Members that contributed:
xxxxxxx xxxxxxxx
yyyyyy YYYYYYYYYY
zzzzz ZZZZZZZ
Demonstration code: [<Ass code 1-3> <abc>]       Important !
==================================== * /
#include <stdlib.h>
#include <stdio.h>

// -----Typedefs -------
typedef struct {
      char firstname[20];
      char famnamne[20];
      char pers_number[13]; // yyyymmddnnnc
}PERSON;
```

```
// Function declaration (to be extend)
PERSON input_record( void);           // Reads in a person record.
void write_new_file( PERSON *inrecord); //Creats a file and write a first record
void printfile(void); // print out all persons in the file
void search_by_firstname( char *name);// print out person if in list
void append_file(PERSON *inrecord);// appends a new person to the file


int main( void){
   PERSON ppost;


   return(0);

}
```

Exercises and assignments for the course DIT165 Development of Embedded systems _ Part 2

This is included in a separate document: DIT165 Exercises_P2_v17xxxx.pdf that will be uploaded to GUL during Course Week nr 3.