Tema 1

Implementați următoarele probleme folosind numai operatori pe biți, operatorul de însumare și tipurile de date primitive din Platform_Types.h (nu aveți voie să folosiți altfel de operatori, array-uri, structuri, etc.)

1. Implementați funcția VerifyPalindrome() care verifică dacă reprezentarea în binar a unui număr primit ca parametru este palindrom.

   ```
   boolean VerifyPalindrome(uint8 Number);
   ```

2. Implementați funcția CountBitsOf1() care numără câți biți de unu sunt în reprezentarea în binar a unui număr primit ca parametru.

   ```
   uint8 CountBitsOf1(uint8 Number);
   ```

3. Implementați funcția VerifyPowerOf2() care verifică dacă un număr primit ca parametru este putere a lui 2.

   ```
   boolean VerifyPowerOf2(uint8 Number);
   ```

4. Implementați funcția CalculateSumOfNibbles() care calculează suma dintre low nibble-ul (cei mai puțini semnificativi 4 biți) și high nibble-ul (cei mai semnificativi 4 biți) ale unui byte primit ca parametru, reprezentarea low nibble-ului în binar fiind interpretată ca un număr fără semn și reprezentarea high nibble-ul în binar fiind interpretată ca un număr cu semn (high nibble-ul poate avea valori în intervalul [-8, 7] iar low nibble-ul poate avea valori în intervalul [0, 15]).

   ```
   sint8 CalculateSumOfNibbles(uint8 Byte);
   ```

   Exemple:

   Byte = 0x5C(16) = 01011100(2)

   High nibble = 0101(2) = 5

   Low nibble = 1100(2) = 12

   Suma = 5 + 12 = 17

   Byte = 0x85(16) = 10000101(2)

   High nibble = 1000(2)

   Semn = -1; Modul = ~(1000 – 1) = ~(0111) = 1000 = 8(10)

   => High Nibble = -8

Low nibble = 0101(2) = 5

Suma = -8 + 5 = -3

Byte = 0xB1(16) = 10110001(2)

High nibble = 1011(2)

Semn = -1; Modul = ~(1011 – 1) = ~(1010) = 0101 = 5(10)

=> High Nibble = -5

Low nibble = 0001(2) = 1

Suma = -5 + 1 = -4

Tema 2

Implementați următoarele probleme folosind numai instrucțiuni condiționale (if, switch), instrucțiuni repetitive (for, while, do while), operatori aritmetici, tipurile de date primitive din Platform_Types.h și tipurile de date din cerințe.

1. Implementați funcția CalculateDistance() care calculează distanța dintre două puncte primite ca parametrii.

   Pentru calculul distanței folosiți teorema lui Pitagora.

   Pentru calculul radicalului (din ipotenuză pentru aflarea distanței) calculați pătrate de baze consecutive, începând de la 1, și comparați cu numărul pentru care se dorește aflarea radicalului; la întâlnirea a două pătrate de baze consecutive pentru care un pătrat este mai mic și celălalt este mai mare se efectuează o aproximare, alegându-se baza pentru care modulul diferenței dintre pătratul calculat și numărul pentru care se dorește radicalul este cel mai mic.

   Exemplu: n^2 = 15;

   3^2 = 9; 4^2 = 16;

   9 < 15 < 16;

   |9 – 15| = 6; |15 – 16| = 1;

   1 < 6 => sqrt(n) = 4

   Implementarea trebuie să funcționeze corect pe orice combinații ale valorilor de intrare.

   Observație: coordonatele sunt exprimate cu semn dar distanța calculată este întotdeauna un număr pozitiv.

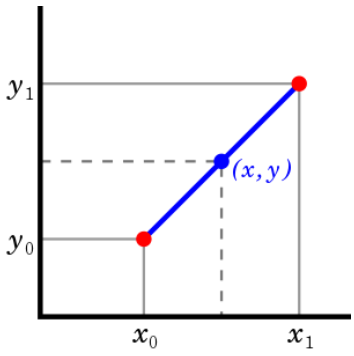
```
typedef struct
{
   sint8 x;
   sint8 y;
} Point_Type;

uint16 CalculateDistance(Point_Type a, Point_Type b);
```

2. Implementați funcția Interpolate() care efectuează interpolarea liniară a unei coordonate pe o dreaptă definită de două puncte.

   Funcția va primi ca parametrii de intrare coordonatele de pe OX și OY ale celor două puncte care definesc dreapta, notate a(x0, y0) și b(x1, y1), și coordonata de pe OX a

punctului pentru care se dorește calculul interpolării, notată cu x. Dacă nu se poate calcula interpolarea atunci funcția va returna 0x80(16) = -128(10) (valoare rezervată pentru eroare).

Funcția va returna valoarea de pe OY, notată cu y, a punctului aflat pe dreapta definită de A și B corespondentă valorii de pe axa OX primită ca parametru.



$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0},$$

Implementarea trebuie să funcționeze corect pe orice combinații ale valorilor de intrare.

```
typedef struct
{
   sint8 x;
   sint8 y;
} Point_Type;

sint8 Interpolate(sint8 x, Point_Type a, Point_Type b);
```

3. Implementați funcția CheckTriangle() care identifică tipul triunghiului definit de dimensiunile a trei laturi primite ca parametru.

   Implementarea trebuie să funcționeze corect pe orice combinații ale valorilor de intrare.

   Observație: valorile de intrare ale lungimilor de laturi pot conține valori <= 0.

```
typedef enum
{
   TRIANGLE_FORM_IMPOSSIBILE,
   TRIANGLE_FORM_SCALENE,
   TRIANGLE_FORM_RIGHT_SCALENE,
   TRIANGLE_FORM_ISOSCELES,
   TRIANGLE_FORM_RIGHT_ISOSCELES,
   TRIANGLE_FORM_EQUILATERAL,
} TriangleForm_Type;
```

```
typedef struct
{
    sint8 a;
    sint8 b;
    sint8 c;
} TriangleDimensions_Type;

TriangleForm_Type CheckTriangle(TriangleDimensions_Type Dimensions);
```

4. Implementați funcția WriteComplexSum() care calculează suma dintre două numere complexe și afișează rezultatul în format "a +- bi".

   Parametrii de intrare specifică coeficienții părților reale (a), coeficienții părților imaginare (b) și puterile numărului imaginar (i) ale două numere complexe.

   Rezultatul afișat trebuie să fie în formă redusă (i-ul să fie la puterea întâi).

   Semnul + se va afișa numai dacă este necesar, iar semnul – se va afișa întotdeauna (de exemplu dacă a = 4 și b = -3 se va afișa "4 - 3i").

   Dacă a = 0 rezultatul se va afișa fără parte reală: "bi".

   Dacă b = 0 rezultatul se va afișa fără parte imaginară: "a".

   Dacă a = 0 și b = 0 rezultatul afișat va fi: "0".

   Implementarea trebuie să funcționeze corect pe orice combinații ale valorilor de intrare.

```
typedef struct
{
    sint8 a;
    sint8 b;
    uint8 iPow;
} ComplexNumber_Type;

void WriteComplexSum(ComplexNumber_Type A, ComplexNumber_Type B);
```

Tema 3

Implementați următoarele module folosind array-uri configurabile static. Nu se permite uzul de pointer-i.

1. Implementați modulul Stack care conține funcții de control ale unei stive (LIFO) configurabile static din punctul de vedere al lungimii și tipului de date folosit pentru elementele stivei.

   Modulul va conține fișierele Stack.h și Stack.c pentru implementarea și exportarea funcționalității modulului și Stack_Cfg.h pentru configurarea lungimii și a tipului de date folosit pentru elementele stivei.

   - Interfețele de implementat sunt:

     ```
     boolean Stack_Push(StackElement_Type tElement);
     boolean Stack_Pop(void);
     void Stack_Print(void);
     ```

   - Tipul de date StackElement_Type trebuie configurat (typedef) în Stack_Cfg.h.
   - Fișierul Stack_Cfg.h va conține macro-ul STACK_LENGTH care va defini lungimea stivei.
   - Toată funcționalitatea modulului poate fi folosită din exterior doar prin includerea fișierului Stack.h. Array-ul care va implementa stiva nu trebuie să fie vizibil din exterior.
   - Dacă stiva este plină și se apelează Stack_Push() funcția va returna FALSE și nu va schimba conținutul stivei. În caz contrar elementul primit ca parametru va fi inserat în stivă și funcția va returna TRUE.
   - Dacă stiva este goală și se apelează Stack_Pop() funcția va returna FALSE și nu va schimba conținutul stivei. În caz contrar se va șterge cel mai nou element adăugat și funcția va returna TRUE.
   - Funcția Stack_Print() va afișa conținutul actual al stivei.

     Exemple:

     - STACK_LENGTH == 8

     Stack_Push(4); Stack_Push(2); Stack_Pop(); Stack_Push(3);

     Stack_Print(); => “4 3”

     - STACK_LENGTH == 2

     Stack_Push(1); Stack_Push(5); Stack_Push(6); Stack_Pop(); Stack_Pop();

Stack_Print(); => “Empty.”

2. Implementați modulul Queue care conține funcții de control ale unei cozi (FIFO) configurabile static din punctul de vedere al lungimii și tipului de date folosit pentru elementele cozii.

   Modulul va conține fișierele Queue.h și Queue.c pentru implementarea și exportarea funcționalității modulului și Queue_Cfg.h pentru configurarea lungimii și a tipului de date folosit pentru elementele cozii.

   - Interfețele de implementat sunt:

     ```
     boolean Queue_Add(QueueElement_Type tElement);
     boolean Queue_Remove(void);
     void Queue_Print(void);
     ```

   - Tipul de date QueueElement_Type trebuie configurat (typedef) în Queue_Cfg.h.
   - Fișierul Queue_Cfg.h va conține macro-ul QUEUE_LENGTH care va defini lungimea cozii.
   - Toată funcționalitatea modulului poate fi folosită din exterior doar prin includerea fișierului Queue.h. Array-ul care va implementa coada nu trebuie să fie vizibil din exterior.
   - Dacă coada este plină și se apelează Queue_Add() funcția va returna FALSE și nu va schimba conținutul cozii. În caz contrar elementul primit ca parametru va fi adăugat în coadă și funcția va returna TRUE.
   - Dacă coada este goală și se apelează Queue_Remove() funcția va returna FALSE și nu va schimba conținutul stivei. În caz contrar se va șterge cel mai vechi element adăugat și funcția va returna TRUE.
   - Funcția Queue_Print() va afișa conținutul actual al cozii.

     Exemple:

     - QUEUE_LENGTH == 8

     Queue_Add(4); Queue_Add(2); Queue_Add(7); Queue_Remove();

     Queue_Print(); => “2 7”

     - QUEUE_LENGTH == 2

     Queue_Add(1); Queue_Add (5); Queue_Add(6); Queue_Remove(); Queue_Remove ();

     Queue_Print(); => “Empty.”

3. Implementați modulul RingBuffer care conține funcții de control ale unui buffer circular configurabil static din punctul de vedere al lungimii și tipului de date folosit pentru elementele buffer-ului circular. Un buffer circular este asemănător cu o coadă cu diferența că reține întotdeauna locația elementului cel mai vechi. Adăugarea și ștergerea de valori se face întotdeauna la / de la locația celui mai vechi element.

   https://en.wikipedia.org/wiki/Circular_buffer

   Modulul va conține fișierele RingBuffer.h și RingBuffer.c pentru implementarea și exportarea funcționalității modulului și RingBuffer_Cfg.h pentru configurarea lungimii și a tipului de date folosit pentru elementele buffer-ului circular.

   - Interfețele de implementat sunt:

     ```
     void RingBuffer_Init(void);
     void RingBuffer_Add(RingBufferElement_Type tElement);
     void RingBuffer_Remove(void);
     void RingBuffer_Print(void);
     ```

   - Tipul de date RingBufferElement_Type trebuie configurat (typedef) în RingBuffer_Cfg.h.
   - Fișierul RingBuffer_Cfg.h va conține macro-ul RINGBUFFER_LENGTH care va defini lungimea buffer-ului.
   - Toată funcționalitatea modulului poate fi folosită din exterior doar prin includerea fișierului RingBuffer.h. Array-ul care va implementa buffer-ul circular nu trebuie să fie vizibil din exterior.
   - Fiecare apel al funcției RingBuffer_Add() va suprascrie locația cea mai veche (chiar dacă buffer-ul este deja plin).
   - Dacă buffer-ul circular este gol și se apelează RingBuffer_Remove() funcția nu va actualiza poziția celui mai vechi element și nici nu va modifica conținutul buffer-ului. În caz contrar poziția actuală a celui mai vechi element adăugat se marchează cu 0xFF (valoare rezervată care specifică locație goală) după care poziția celui mai vechi element adăugat se actualizează.
   - Funcția RingBuffer_Print() va afișa întregul conținut al buffer-ului.
   - Funcția RingBuffer_Init() va inițializa întregul buffer cu 0xFF.

   Exemple:

   - RINGBUFFER_LENGTH == 4

   RingBuffer_Add(4); RingBuffer_Add(2); RingBuffer_Add(7); RingBuffer_Remove();

   RingBuffer_Print(); => “255 2 7 255”

- RINGBUFFER_LENGTH == 4

RingBuffer_Add(1); RingBuffer_Add(3); RingBuffer_Add(7); RingBuffer_Add(2); RingBuffer_Add(9);

RingBuffer_Print(); => “9 3 7 2”

4. Implementați modulul SearchInterval() care identifică dacă un interval specificat ca parametru este inclus total într-o serie de intervale configurabile static. Intervalele în care se caută sunt definite printr-un array numit SearchInterval_Config[] configurabil, sortat crescător în prealabil. Elementele array-ului definesc marginile de interval închise la ambele capete.

   Exemplu:

   SearchInterval_Config[5] = {0, 6, 9, 40, 100} definește intervalele [0, 6], [6, 9], [9, 40], [40, 100] cu ID-urile 0, 1, 2, 3.

   Dacă intervalul de căutat este [12, 36] funcția va returna ID-ul 2 pentru că intervalul căutat este inclus total în intervalului [9, 40]. Dacă intervalul de căutat este [10, 41] funcția va returna cod de eroare.

   Modulul va conține fișierele SearchInterval.h și SearchInterval.c pentru implementarea și exportarea funcționalității modulului și SearchInterval_Cfg.h Și SearchInterval_Cfg.c pentru configurarea lungimii și a conținutului array-ului SearchInterval_Config[]. SearchInterval.c va declara SearchInterval_Config[] ca extern pentru a-l putea accesa.

   - Interfața funcției de implementat este:

     ```
     uint8 Search_Interval(uint8 LeftLimit, uint8 RightLimit);
     ```

   - Fișierul SearchInterval_Cfg.h va conține macro-ul SEARCHINTERVAL_CONFIG_LENGTH care va defini lungimea lui SearchInterval_Config[].
   - Fișierul SearchInterval_Cfg.c va conține valorile array-ului SearchInterval_Config[].
   - Toată funcționalitatea modulului poate fi folosită din exterior doar prin includerea fișierului SearchInterval.h. Array-ul de configurare nu trebuie să fie vizibil din exterior.
   - Funcția SearchInterval() returnează codul de eroare 0xFF dacă intervalul specificat prin limitele date ca parametrii nu este inclus total în nici un interval configurat. În caz contrar funcția returnează ID-ul intervalului în care intervalul de intrare a fost găsit.

Tema 4

Implementați următoarele probleme folosind matrici configurabile static. Nu se permite uzul de pointer-i.

1. Implementați funcția CalcBigMatrixSum() care calculează care este suma cea mai mare dintre suma diagonalei principale, suma diagonalei secundare, suma primei linii, suma ultimei linii, suma primei coloane și suma ultimei coloane ale unei matrice pătratice.

   Funcția va fi implementată în CalcBigMatrixSum.c și exportată prin CalcBigMatrixSum.h. Lungimea matricei pătratice va fi definită în CalcBigMatrixSum_Cfg.h prin macro-ul CALCBIGMATRIXSUM_MATRIX_LENGTH.

   Matricea pe baza căreia se vor calcula sumele va fi numită CalcBigMatrixSum_Matrix[][] și valorile acesteia vor fi configurate în CalcBigMatrixSum_Cfg.c. CalcBigMatrixSum.c va declara CalcBigMatrixSum_Matrix[][] ca externă pentru a o putea accesa.

   - Interfața de implementat este:

     ```
     CalcBigMatrixSum_Type CalcBigMatrixSum(void);
     ```

   - Tipul de date de folosit este:

     ```
     typedef enum
     {
         CALCBIGMATRIXSUM_PRIMARY_DIAGONAL = 0,
         CALCBIGMATRIXSUM_SECONDARY_DIAGONAL = 1,
         CALCBIGMATRIXSUM_FIRST_ROW = 2,
         CALCBIGMATRIXSUM_LAST_ROW = 3,
         CALCBIGMATRIXSUM_FIRST_COLUMN = 4,
         CALCBIGMATRIXSUM_LAST_COLUMN = 5,
     } CalcBigMatrixSum_Type;
     ```

   - Funcția CalcBigMatrixSum() va returna locația unde se află suma cu valoarea cea mai mare. Dacă există două sau mai multe sume egale, funcția va returna valoarea cea mai mică din CalcBigMatrixSum_Type în concordanță cu tipurile de sume implicate. Exemple:

     - CALCBIGMATRIXSUM_MATRIX_LENGTH = 3

       0 1 2

       3 4 5

       6 7 8

       Suma diagonalei principale = 0 + 4 + 8 = 12

       Suma diagonalei secundare = 2 + 4 + 6 = 12

       Suma primei linii = 0 + 1 + 2 = 3

Suma ultimei linii = 6 + 7 + 8 = 21

Suma primei coloane = 0 + 3 + 6 = 9

Suma ultimei coloane = 2 + 5 + 8 = 15

Valoare returnată = CALCBIGMATRIXSUM_LAST_ROW

- CALCBIGMATRIXSUM_MATRIX_LENGTH = 3

  9 0 8

  3 9 1

  8 2 9

  Suma diagonalei principale = 9 + 9 + 9 = 27

  Suma diagonalei secundare = 8 + 9 + 8 = 25

  Suma primei linii = 9 + 0 + 8 = 17

  Suma ultimei linii = 8 + 2 + 9 = 19

  Suma primei coloane = 9 + 3 + 8 = 20

  Suma ultimei coloane = 8 + 1 + 9 = 18

  Valoare returnată = CALCBIGMATRIXSUM_ PRIMARY_DIAGONAL

- CALCBIGMATRIXSUM_MATRIX_LENGTH = 3

  8 0 8

  8 0 8

  8 8 8

  Suma diagonalei principale = 8 + 0 + 8 = 16

  Suma diagonalei secundare = 8 + 0 + 8 = 16

  Suma primei linii = 8 + 0 + 8 = 16

  Suma ultimei linii = 8 + 8 + 8 = 24

  Suma primei coloane = 8 + 8 + 8 = 24

  Suma ultimei coloane = 8 + 8 + 8 = 24

  Valoare returnată = CALCBIGMATRIXSUM_ LAST_ROW

  (conflict între ultima linie, prima coloană și ultima coloană, valoarea lui CALCBIGMATRIXSUM_ LAST_ROW = 3 fiind cea mai mică dintre cele 3 valori corespondente din CalcBigMatrixSum_Type).

2. Implementați modulul TouchMatrix care implementează o funcție de inițializare, o funcție de rotație, o funcție de interschimbare a două linii sau două coloane și o funcție de afișare pentru o matrice pătratică configurabilă static din punctul de vedere al lungimii.

   Funcțiile vor fi implementate în TouchMatrix.c și exportate prin TouchMatrix.h. Lungimea matricei pătratice va fi definită în TouchMatrix_Cfg.h.

   - Interfețele de implementat sunt:

```
void TouchMatrix_Init(void);
void TouchMatrix_Rotate(TouchMatrix_RotateType tRotateType);
void TouchMatrix_Swap(TouchMatrix_SwapType tSwapType, uint8 ucLine0, uint8 ucLine1);
void TouchMatrix_Print(void);
```

   - Tipurile de date de folosit sunt:

```
typedef enum
{
    TOUCHMATRIX_ROTATE_90_CW,
    TOUCHMATRIX_ROTATE_90_CCW,
    TOUCHMATRIX_ROTATE_180,
} TouchMatrix_RotateType;

typedef enum
{
    TOUCHMATRIX_SWAP_ROWS,
    TOUCHMATRIX_SWAP_COLUMNS,
} TouchMatrix_SwapType;
```

   - Fișierul TouchMatrix_Cfg.h. va conține macro-ul TOUCHMATRIX_LENGTH care va defini lungimea matricei pătratice.
   - Matricea folosită va fi declarată și folosită numai în TouchMatrix.c. Folosirea matricei este obligatorie. Declararea matricei se va face astfel:

```
uint8 TouchMatrix_Buffer[TOUCHMATRIX_LENGTH][TOUCHMATRIX_LENGTH];
```

   - Toată funcționalitatea modulului poate fi folosită din exterior doar prin includerea fișierului TouchMatrix.h. Matricea nu trebuie să fie vizibilă din exterior.
   - Toate funcțiile trebuie să funcționeze corect numai pentru valori mai mari sau egale cu 2 ale macro-ului TOUCHMATRIX_LENGTH. Nu este necesar ca funcția să verifice această valoare, presupunându-se că valoarea macro-ului este întotdeauna corectă.
   - Funcția TouchMatrix_Init() va inițializa matricea cu valori de la 0 la ((TOUCHMATRIX_LENGTH * TOUCHMATRIX_LENGTH) – 1). Exemple:
     - TOUCHMATRIX_LENGTH = 3

       0 1 2

       3 4 5

6 7 8

- TOUCHMATRIX_LENGTH = 4

  0 1 2 3

  4 5 6 7

  8 9 10 11

  12 13 14 15

- Funcția TouchMatrix_Rotate() va roti matricea în funcție de parametrul primit (90 de grade în sensul acelor de ceasornic, 90 de grade în sensul invers acelor de ceasornic, sau 180 de grade). Exemple:

  - TOUCHMATRIX_LENGTH = 3

    TouchMatrix_Init();

    0 1 2

    3 4 5

    6 7 8

    TouchMatrix_Rotate(TOUCHMATRIX_ROTATE_90_CW);

    6 3 0

    7 4 1

    8 5 2

    TouchMatrix_Rotate(TOUCHMATRIX_ROTATE_180);

    2 5 8

    1 4 7

    0 3 6

    TouchMatrix_Rotate(TOUCHMATRIX_ROTATE_90_CCW);

    8 7 6

    5 4 3

    2 1 0

- Funcția TouchMatrix_Swap() va interschimba două linii sau două coloane din matrice în funcție de parametrul primit. Dacă cel puțin unul din parametrii care indică indexul de linie sau coloană este mai mare sau egal cu TOUCHMATRIX_LENGTH funcția nu va modifica conținutul matricei în nici un fel. Exemple:

- TOUCHMATRIX_LENGTH = 3

  TouchMatrix_Init();

  0 1 2

  3 4 5

  6 7 8

  TouchMatrix_Swap(TOUCHMATRIX_SWAP_COLUMNS, 0, 1);

  1 0 2

  4 3 5

  7 6 8

  TouchMatrix_Swap(TOUCHMATRIX_SWAP_COLUMNS, 0, 0);

  1 0 2

  4 3 5

  7 6 8

  TouchMatrix_Swap(TOUCHMATRIX_SWAP_ROWS, 1, 4);

  1 0 2

  4 3 5

  7 6 8

  TouchMatrix_Swap(TOUCHMATRIX_SWAP_ROWS, 1, 2);

  1 0 2

  7 6 8

  4 3 5

  TouchMatrix_Swap(TOUCHMATRIX_SWAP_COLUMNS, 1, 2);

  1 2 0

  7 8 6

  4 5 3

- Funcția TouchMatrix_Print() va afișa întregul conținut al matricei, fiecare element fiind afișat pe 4 poziții (printf(“%4d”, TouchMatrix_Buffer[i][j]);).

3. Implementați funcția DrawDiamond() care scrie caractere într-o matrice pătratică configurabilă static din punctul de vedere al lungimii pentru obținerea unei forme de romb, după care o afișează.

   Funcția va fi implementată în DrawDiamond.c și exportată prin DrawDiamond.h. Lungimea matricei pătratice va fi definită în DrawDiamond_Cfg.h.

   - Interfața de implementat este:

     ```
     void DrawDiamond(void);
     ```

   - Fișierul DrawDiamond_Cfg.h. va conține macro-ul DRAWDIAMOND_MATRIX_LENGTH care va defini lungimea matricei pătratice.
   - Matricea folosită va fi declarată și folosită numai în DrawDiamond.c. Folosirea matricei este obligatorie. Declararea matricei se va face astfel:

     ```
     uint8 DrawDiamondMatrix[DRAWDIAMOND_MATRIX_LENGTH][DRAWDIAMOND_MATRIX_LENGTH];
     ```

   - Toată funcționalitatea modulului poate fi folosită din exterior doar prin includerea fișierului DrawDiamond.h. Matricea nu trebuie să fie vizibilă din exterior.
   - Funcția DrawDiamond() trebuie să funcționeze corect numai pentru valori impare mai mari sau egale cu 3 ale macro-ului DRAWDIAMOND_MATRIX_LENGTH. Nu este necesar ca funcția să verifice această valoare, presupunându-se că valoarea macro-ului este întotdeauna corectă.
   - Exemple:
     - DRAWDIAMOND_MATRIX_LENGTH = 3

       ```
       - * -
       * * *
       - * -
       ```

     - DRAWDIAMOND_MATRIX_LENGTH = 5

       ```
       - - * - -
       - * * * -
       * * * * *
       - * * * -
       - - * - -
       ```

     - DRAWDIAMOND_MATRIX_LENGTH = 7

       ```
       - - - * - - -
       - - * * * - -
       - * * * * * -
       * * * * * * *
       - * * * * * -
       - - * * * - -
       - - - * - - -
       ```

- Info:
    - Inițializarea unui element din matrice cu un caracter ‘*’ se poate efectua astfel:

      DrawDiamondMatrix[0][0] = ‘*’;
    - Afișarea în format caracter se poate efectua astfel:

      printf(“%c”, DrawDiamondMatrix[0][0]);

Tema 5

Implementați următoarele probleme folosind pointer-i. Este permisă atât aritmetica pe pointer-i cât și indexarea față de pointer-i.

1. Implementați funcția FindPattern() care identifică pozițiile unui pattern de un octet primit ca parametru în conținutul unei structuri configurabile.

   Funcția va scrie toate pozițiile identificate într-un buffer indicat printr-un pointer primit ca parametru și va returna numărul de apariții. Pozițiile identificate vor reprezenta indecșii octeților din structura dată și se vor numerota crescător de la 0 (poziția adresei primului octet din structură) la lungimea structurii – 1 (poziția adresei ultimului octet din structură). Dacă numărul de apariții este mai mare decât un număr maxim configurabil static atunci funcția va returna o valoare de eroare 0xFF.

   Numărul maxim de poziții identificate va fi configurat în FindPattern_Cfg.h prin macro-ul FINDPATTERN_MAXIMUM.

   Tipul structurii în conținutul căreia se va efectua căutarea se va numi FindPattern_StructType și va fi definită în FindPattern_Cfg.h.

   Funcția FindPattern() va fi implementată în FindPattern.c.

   Funcția FindPattern() și tipul de date FindPattern_StructType vor fi vizibile din exterior numai prin includerea fișierului FindPattern.h.

   Este responsabilitatea user-ului să își declare un buffer în care modulul să scrie valorile pozițiilor unde a fost identificat pattern-ul.

   - Interfața de implementat este:

     ```
     uint8 FindPattern(FindPattern_StructType tStructure, uint8 ucPattern, uint8 * pucPositions);
     ```

   - Exemple:

     a.

     ```
     typedef struct
     {
         uint8 ucSmall0;
         uint8 ucSmall1;
         uint8 ucSmall2;
         uint8 ucSmall3;
         uint8 ucSmall4;
         uint8 ucSmall5;
     } FindPattern_StructType;

     #define FINDPATTERN_MAXIMUM (5U)
     ```

```
uint8 aucTestPositions[100];
FindPattern_StructType tTest =
{ 0x01, 0x02, 0x03, 0xAA, 0x03, 0x03 };

FindPattern(tTest, 0x03, aucTestPositions);
/* Funcția returnează 3, iar primele 3 valori ale lui
aucTestPositions[] sunt {2, 4, 5}. */
```

Funcția ar fi returnat 0xFF dacă FINDPATTERN_MAXIMUM ar fi fost <= 2.

b.

```
typedef struct
{
    uint8 aucArray[4];
    uint32 ulBig;
    uint8 ucSmall;
} FindPattern_StructType;

#define FINDPATTERN_MAXIMUM (5U)

uint8 aucTestPositions[100];
FindPattern_StructType tTest =
{ { 0x01, 0x02, 0x03, 0xAA, }, 0x22AA03AA, 0xAA, };

FindPattern(tTest, 0xAA, aucTestPositions);
/* Funcția returnează 4, iar primele 4 valori ale lui
aucTestPositions[] sunt {3, 4, 6, 8}. */
```

Funcția ar fi returnat 0xFF dacă FINDPATTERN_MAXIMUM ar fi fost <= 3.

Observație: în funcție de tipul de reprezentare de date în memorie este posibil să aveți rezultate diferite. Din moment ce avem aceleași laptop-uri ar trebui totuși ca rezultatele să fie aceleași la toți.

2. Implementați modulul Diagnostics care prelucrează frame-uri primite prin intermediul unui buffer specificat printr-un pointer, decodificându-le și executând comenzi în funcție de conținut, urmând a scrie un frame de răspuns într-un buffer specificat prin intermediul altui pointer.

   Frame-urile (cadre) sunt șiruri de octeți de maximum 8 octeți.

   Frame-urile de intrare vor avea următorul format:

   - Byte 0 – tipul serviciului.
   - Byte 1 – lungimea payload-ului (numărul de octeți de informație utilă); valori permise: [1, 6].
   - Byte 2, …, Byte N (2 <= N <= 8) – informația utilă de procesat.

   Frame-urile de ieșire vor avea următorul format:

   - Byte 0 – răspuns pozitiv (0x00) sau răspuns negativ (0xFF).
   - Byte 1 – lungimea răspunsului (informația utilă) valori permise: [1, 2].
   - Byte 2, …, Byte N (2 <= N <= 3) – informația utilă de răspuns.

   - Interfața de implementat este:

     ```
     void Diagnostics_Handle(uint8 * pucRequest, uint8 * pucResponse);
     ```

   - Tipurile de date de folosit sunt:

     ```
     typedef boolean (DiagnosticsService_Type)(uint8 * pucRequestPayload, uint8 ucRequestLength, uint8 * pucResponse);

     typedef struct
     {
        uint8 ucServiceID;
        DiagnosticsService_Type * pfService;
     } DiagnosticsTableElement_Type;
     ```

   Modulul va conține următoarele fișiere:

   - Fișierul Diagnostics.c va implementa funcția Diagnostics_Handle() care va decodifica informația de control (tipul serviciului și lungimea payload-ului) după care va apela funcția corespunzătoare serviciului decodificat prin intermediul unui function pointer, oferindu-i totodată lungimea payload-ului de request și pointer la adresa de unde serviciul își poate scrie răspunsul (adresă către byte-ul 1 din frame-ul de răspuns). Dacă serviciul identificat nu este găsit sau lungimea payload-ului nu este în intervalul valorilor acceptate atunci funcția va pregăti direct un răspuns negativ (fără a apela vreun serviciu). De asemenea dacă serviciul executat returnează FALSE funcția va pregăti un răspuns negativ. În cazul unui răspuns

negativ (byte 0 = 0x00) conținutul octeților de răspuns 1..N sunt irelevanți (pot avea orice valoare).

- Fișierul Diagnostics.h va exporta funcția Diagnostics_Handle().
- Fișierul Diagnostics_Types.h va conține toate tipurile necesare în interiorul modulului. Aceste tipuri nu trebuie să fie vizibile în afara modulului.
- Fișierul Diagnostics_Cfg.h va conține macro-ul DIAGNOSTICS_NR_OF_SERVICES care configurează numărul de servicii.
- Fișierul Diagnostics_Table.c va configura tabelul folosit pentru decodificarea serviciului: DiagnosticsTableElement_Type Diagnostics_TableCfg[DIAGNOSTICS_NR_OF_SERVICES].
- Fișierul Diagnostics_ServicesImpl.c va implementa serviciile specificate ca function pointer-i în tabelul de decodificare.

ID-urile serviciilor cât și denumirile funcțiilor care vor implementa serviciile vor fi configurabile. Fiecare serviciu va interpreta octeții din payload ca elemente separate (fiecare octet va reprezenta un număr care se va folosi în procesare). Serviciile de implementat sunt: calculul sumei, produsului, maximului, minimului, mediei aritmetice, SAU-ului logic sau ȘI-ului logic între octeții payload-ului de request.

Valoarea returnată de servicii va specifica dacă serviciul s-a executat cu succes sau nu. Dacă serviciul a fost executat cu succes atunci valoarea returnată este TRUE iar lungimea răspunsului este 1 sau 2. Dacă lungimea răspunsului este 2 atunci octeții vor fi stocați în ordine MSB-first. În cazul în care răspunsul nu încape pe 2 octeți atunci serviciul trebuie să returneze FALSE.

- Exemple (numerele marcate cu X sunt irelevante; în fiecare exemplu se cheamă funcția de procesare astfel: Diagnostics_Handle(aucRequest, aucResponse)):

1. ID serviciu de sumă = 0xAA. (celelalte ID-uri nu sunt relevante pentru acest exemplu)

1.1. Frame intrare: aucRequest[8] = {0xAA, 0x03, 0x01, 0x02, 0x09, X, X, X}

Frame ieșire: aucResponse[8] = {0x00, 0x01, 0x0C, X, X, X, X, X}

0x01 + 0x02 + 0x09 = 0x0C (rezultat pe 1 octet -> lungimea răspunsului = 1)

1.2. Frame intrare: aucRequest[8] = {0xAA, 0x04, 0xF0, 0xF1, 0xF2, 0xF3, X, X}

Frame ieșire: aucResponse[8] = {0x00, 0x02, 0x03, 0xC6, X, X, X, X}

0xF0 + 0xF1 + 0xF2 + 0xF3 = 0x03C6 (rezultat pe 2 octeți -> lungimea răspunsului = 2)

2. ID serviciu de produs = 0x12. (celelalte ID-uri nu sunt relevante pentru acest exemplu)

2.1. Frame intrare: aucRequest[8] = {0x12, 0x03, 0x01, 0x02, 0x03, X, X, X}

Frame ieșire: aucResponse[8] = {0x00, 0x01, 0x06, X, X, X, X, X}

0x01 * 0x02 * 0x03 = 0x06 (rezultat pe 1 octet -> lungimea răspunsului = 1)

2.2. Frame intrare: aucRequest[8] = {0x12, 0x04, 0xFF, 0xFF, 0xFF, 0xFF, X, X}

Frame ieșire: aucResponse[8] = {0xFF, X, X, X, X, X, X, X}

0xFF * 0xFF * 0xFF * 0xFF = 0xFC05FC01 (rezultatul nu încape pe 2 octeți -> serviciul returnează FALSE și răspunsul este negativ)

3. ID serviciu de sumă = 0x01.

ID serviciu de produs = 0x02.

ID serviciu de maxim = 0x03.

ID serviciu de minim = 0x04.

ID serviciu de medie aritmetică = 0x05.

ID serviciu de SAU logic = 0x06.

ID serviciu de ȘI logic = 0x07.

3.1. Frame intrare: aucRequest[8] = {0x85, X, X, X, X, X, X, X}

Frame ieșire: aucResponse[8] = {0xFF, X, X, X, X, X, X, X}

Serviciul 0x85 nu există, răspuns negativ.

3.2. Frame intrare: aucRequest[8] = {0x04, 0x08, X, X, X, X, X, X}

Frame ieșire: aucResponse[8] = {0xFF, X, X, X, X, X, X, X}

Lungimea payload-ului de request nu este corect, răspuns negativ.

3.3. Frame intrare: aucRequest[8] = {0x03, 0x06, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06}

Frame ieșire: aucResponse[8] = {0x00, 0x01, 0x06, X, X, X, X, X}

Maximul dintre octeții payload-ul este 0x06.

3.4. Frame intrare: aucRequest[8] = {0x05, 0x06, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06}

Frame ieșire: aucResponse[8] = {0x00, 0x01, 0x03, X, X, X, X, X}

Media aritmetică este (1 + 2 + 3 + 4 + 5 + 6) / 6 = 3.

3.5. Frame intrare: aucRequest[8] = {0x07, 0x02, 0x07, 0x03, X, X, X, X}

Frame ieșire: aucResponse[8] = {0x00, 0x01, 0x03, X, X, X, X, X}

ȘI-ul logic este: 0x07 & 0x03 = 0x03.

1. Software and Hardware Requirements

1.1 General requirements

<USP_GEN_001> Application shall act as a slave with the external tester being the master (PC Application)

<USP_GEN_002> Serial Communication should be used to send messages between ECU and PC.

<USP_GEN_003> Target ECU: 16-Bit CMOS Single-Chip Microcontroller (delivered)

<USP_GEN_004> Project should be delivered as a package with all sources and documentation

1.2 ECU-Application requirements

<USP_ECU_001> C Language should be used.

<USP_ECU_002> Application shall provide a communication interface via Serial

<USP_ECU_003> Use LED's from port P2.8 and P2.9 for Line-change Lights (left / right) and Warning lights.

<USP_ECU_004> The Line-change Lights LED should blink with a 1Hz frequency and 60% Dutycycle. (PWM output lines)

1.3 PC-Application requirements

<USP_PC_002> C# (shall be used

<USP_PC_003> Application shall simulate commands from a real Car dashboar (**Acceleration**, **Warning Lights, Line-change Lights**)

<USP_PC_004> Acceleration shall be simulated with a TrackBar component or EditSpin Component

<USP_PC_005> Each time the acceleration, Line-change or Warning value is changed application shall send a message to the board with this value.

<USP_PC_006> Line-change Lights and Warning lights should be simulated with buttons (left / right)

<USP_PC_007> Dashboard display should be refreshed every 1 second with the current values from the board.

1.4 Quality requirements

<UPS_QA_002> “V-Model” Standard SW development should be used in both parts of the modules.

Requested documents:

- Requirements Document (common for ECU and PC parts)

- Rough design document

i. Use case diagram

- Detail design

i. Sequence diagram

ii. Class diagram

iii. State machine diagram

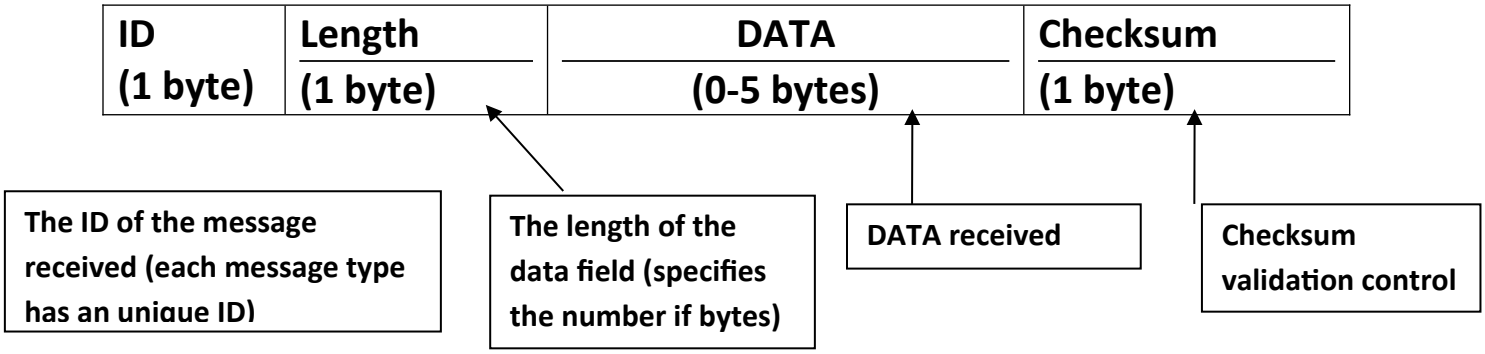
- Integration tests (common for ECU and PC parts)

i. Test cases document

ii. Test report document

- Timer Interrupt service routine used in order to caption an interrupt on every 840 ms
- Serial Receive service routine used in order to receive data frames from the PC using serial mode 19200 bits/second

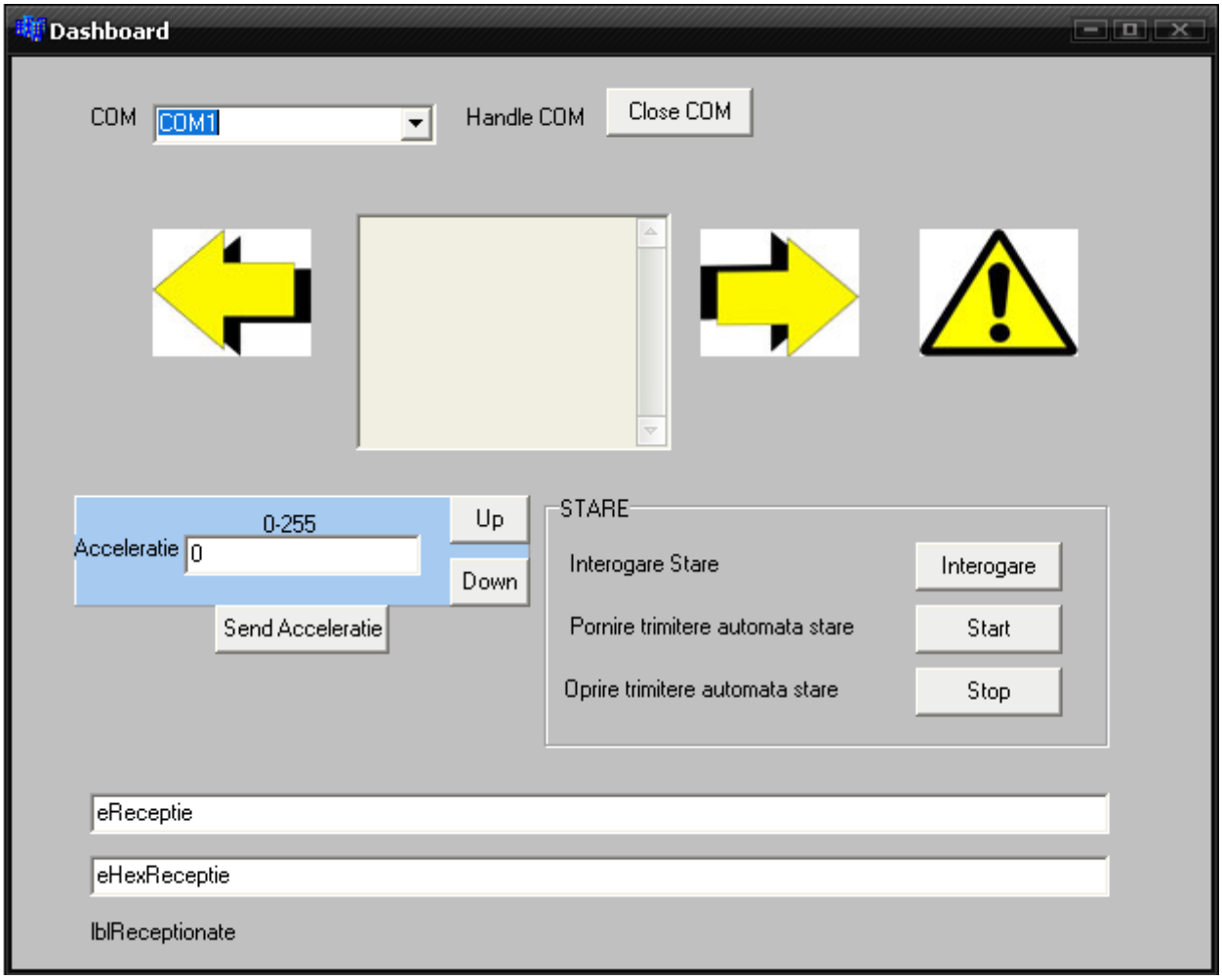
2. Software and Hardware Requirements



- **Commands Table**

Command Type			PC => ECU	ECU=>PC
Line Change Light Or Warning	**SPS**	**1**	Line Change Light Left ON	**ACK SPS (1)**
	SOS	**2**	Line Change Light Left OFF	**ACK SOS (0)**
	SPD	**3**	Line Change Light Right ON	**ACK SPD (5)**
	SOD	**4**	Line Change Light Right OFF	**ACK SOD (0)**
	SPA	**5**	Warning ON	**ACK SPA (2, 3, 4)**
	SOA	**6**	Warning OFF	**ACK SOA (0)**
State	**SPTA**	**7**	Auto send State ON	**ACK SPTA**
	SOTA	**8**	Auto send State OFF	**ACK SOTA**
	SIS	**9**	Interrogate state	**SRI**
				SRA
Acceleration	**AIA**	**10**	Acceleration impose	**ACK AIA**
	ACA	**11**	Increase acceleration	**ACK ACA**
	ASA	**12**	Decrease acceleration	**ACK ASA**

2.1 PC Detailed Design



1. What was introduced by keyboard if the output of the script is eeleeeneee?

```
int i,n;
char c;
read n
for i = 1,n do
{
   read c
   write c,'e'
}
```

2. Compute in base (numeral system) 7:
   13 + 55 = ?

3. What are the differences between Flash and RAM memories?

4. What is the duration of the positive pulse of a PWM signal with a frequency of 70 KHz and a duty cycle of 35%?

5. Write a function that computes the sum of the elements of a square matrix of dimensions N, that are placed below the secondary diagonal.

6. Implement a function to identify the second largest number in an array.

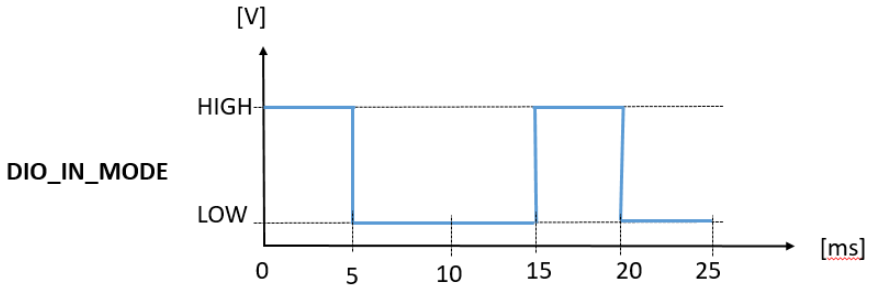
7. Implement a function that returns if an unsigned 16 bit value is a power of 2.

8. An array of size 100 contains distinctive integer values from 1 to 100. The values are randomly placed inside the array. One random value from the array gets corrupted, being replaced by a value of 0. Implement a function for identifying the replaced value.

9. The following C module is part of a software that makes use of a real time operating system. The operating system periodically calls the **TASK(OS_TASK_5MS)** function each 5 ms. The module makes use of two DIO (digital input-output) driver functions for processing digital signals:

```
/* Returns the current level (1 – High, 0 – Low) from the specified DIO channel. */
Dio_LevelType Dio_ReadChannel(Dio_ChannelType ChannelId);

/* Applies the specified level (1 – High, 0 – Low) to the specified DIO channel. */
void Dio_WriteChannel(Dio_ChannelType ChannelId, Dio_LevelType Level);
```

Execute the code and draw the output signals (**DIO_OUT_0** and **DIO_OUT_1**) for the first 25 ms (a total of 6 function calls) for the following input values on the **DIO_IN_MODE** channel:




```c
#include "Std_Types.h"
#include "Os.h"
#include "Dio.h"

static uint8 Task_ucCallCounter = 6U;
static uint8 Task_ucGenerator = 0x07U;
static uint8 Task_ucOutputValue = 0x00U;

TASK(OS_TASK_5MS)
{
   Dio_LevelType tMode;

   if (0U == (Task_ucCallCounter & 0x01U))
   {
      tMode = Dio_ReadChannel(DIO_IN_MODE);
      if (STD_LOW == tMode)
      {
         Task_ucOutputValue = Task_ucCallCounter;
      }
      else if (STD_HIGH == tMode)
      {
         Task_ucOutputValue = Task_ucGenerator;
      }
      else
      {
      }
   }
   else
   {
   }

   Dio_WriteChannel(DIO_OUT_0, (Dio_LevelType) (Task_ucOutputValue & 0x01U));
   Dio_WriteChannel(DIO_OUT_1, (Dio_LevelType) ((Task_ucOutputValue & 0x02U) >> 1U));

   Task_ucOutputValue >>= 2U;

   Task_ucCallCounter++;
   Task_ucGenerator = ((Task_ucGenerator ^ 0x06U) + 1U);
}
```