1 „Das fehlende Element“

Lösungsidee:

Man addiert alle Werte in dem Array und zeitgleich bildet man eine Summe aus den Indexen vom array (die Zahlen die es sein sollten). Anschließend subtrahiert man von der kompletten summe der Indexes plus der länge des Arrays, die summe des Arrays. Das Ergebnis ergibt die fehlende Zahl.

Testfälle: {2, 1, 3, 4}, {0, 1, 2, 3}, {0, 1 ,3}, {1}, {0}, {}

PseudoCode:

```
getMissingNumber (↓integer[] values, ↑integer missingValue)
    missingNumber = 0

    if (values.length > 0)
        integer sumOfIndexes = 0
        integer sumOfValues = 0

        for (integer i = 0; i < values.length; i++) do
            sumOfIndexes += i
            sumOfValues += values[i]
        end for

        missingNumber = (sumOfIndexes + values.length) - sumOfValues
    end if
end getMissingNumber
```

Schreibtischtests:

Annahme: {2, 1, 3, 4}	missingNumber	values.length > 0	i	i < values.length	sumOfIndexes	sumOfValues
	0	TRUE	0	TRUE	0	2
			1	TRUE	1	3
			2	TRUE	3	6
			3	TRUE	6	10
			4	FALSE		
	0					
Annahme: {0, 1, 2, 3}	0	TRUE	0	TRUE	0	0
			1	TRUE	1	1
			2	TRUE	3	3
			3	TRUE	6	6
			4	FALSE		
	4					
Annahme: {0, 1, 3}	0	TRUE	0	TRUE	0	0
			1	TRUE	1	1
			2	TRUE	3	4
			3	FALSE		
	2					
Annahme: {1}	0	TRUE	0	TRUE	0	1
			1	FALSE		
	0					
Annahme: {0}	0	TRUE	0	TRUE	0	0
			1	FALSE		
	1					
Annahme: {}	0	FALSE				
	0					

2 Bestimmung von lokalen Minima und Maxima

LösungsIdee:

Wenn ein Array n>2 einträge hat läuft ein zähler durch das ganze Array mit Ausnahme der randwerte. An jedem Punkt wird überprüft ob ein lokales Minima oder ein lokales Maxima besteht.

Tesfälle: {1, 3, 5, 4, 6, 5, 1, 2, 1, 1}, {2, 1, -3, 4, 2}, {2, 8}

PseudoCode:

```
findMinMaxValues (↓integer[] values, ↑integer maxCount, ↑integer minCount)
    minCount = 0
    maxCount = 0

    if (values.length > 2)
        for (integer i = 1; i < values.length - 1; i++) do
            integer currValue = values[i]

            if (currValue < values[i - 1] && currValue < values[i + 1])
                minCount++
            else if (currValue > values[i - 1] && currValue > values[i + 1])
                maxCount++
            end if
        end for
    end if
end findMinMaxValues
```

Schreibtischtests:

Annahme: {1, 3, 5, 4, 6, 5, 1, 2, 1, 1}	minCount	maxCount	values.length > 2	i	i < values.length - 1	isMin	IsMax
	0	0	TRUE	1	TRUE	FALSE	FALSE
				2	TRUE	FALSE	TRUE
		1		3	TRUE	TRUE	FALSE
	1			4	TRUE	FALSE	TRUE
		2		5	TRUE	FALSE	FALSE
				6	TRUE	TRUE	FALSE
	2			7	TRUE	FALSE	TRUE
		3		8	TRUE	FALSE	FALSE
				9	FALSE		
	2	3					
Annahme: {2, 3, -3, 4, 2}	0	0	TRUE	1	TRUE	FALSE	TRUE
		1		2	TRUE	TRUE	FALSE
	1			3	TRUE	FALSE	TRUE
		2		4	FALSE		
	1	2					
Annahme: {2, 8}	0	0	FALSE				
	0	0					

3 Russische Bauernmultiplikation

LösungsIdee:

Die Russische Bauernmultiplikation ist ein Multiplikationsverfahren indem der Multiplikator solange halbiert wird bis er 1 erreicht. Der Multiplikant zudem wird immer mit sich selber addiert. Der Multiplikant wird in jedem verlauf, falls der Multiplikator ungerade ist, zu dem endergebnis addiert.

Testfälle: (3, 8) ; (-4, 0) ; (0, 8) ; (10, 1)

C-Program code:

Int main:

```c
int main(int argc, char* argv[]) {
    int test_convertDecimalToBinary = 0;
    int test_russianPeasantMultiplication = 1;

    // convert Decimal to Binary ------------------------------------------------------
    convertDecimalToBinary

    // russian peasant multiplication -------------------------------------------------
    #pragma region russianPeasantMultiplication
    if (test_russianPeasantMultiplication) {
        printf("\n#### russianMul ########################################");

        // test 1 -> values: 3, 8 ---------------
        russianPeasantMultiplication(3, 8);

        // test 2 -> values: -4, 0 --------------
        russianPeasantMultiplication(-4, 0);

        // test 3 -> values: 0, 8 ---------------
        russianPeasantMultiplication(0, 8);

        // test 4 -> values: 10, 1 --------------
        russianPeasantMultiplication(10, 1);

        printf("\n-------------------------------------------------");
    }
    #pragma endregion

    // alternativ
    getMissingNumber

    // just to make it more beautiful -------------------------------------------------
    printf("\n\n");
    // --------------------------------------------------------------------------------

    return 0;
}
```

Int russianPeasantMultiplication:

```c
int russianPeasantMultiplication(int multiplier, int multiplicand) {
    printf("\n-------------------------------------------------");
    printf("\nmultiplier: %d, multiplicand: %d\n", multiplier, multiplicand);

    int result = 0;

    if (multiplier > 0 && multiplicand > 0) {
        printf("\nmultiplier > 0 && multiplicand > 0");

        printf("\ncalculating...");
        while (multiplier >= 1) {
            if ((multiplier % 2) != 0) {
                result += multiplicand;
            }

            multiplier /= 2;
            multiplicand += multiplicand;
        }
        printf("\nfinished");
    } else {
        printf("\nmultiplier, multiplicand or both <= 0");
    }

    printf("\n\nresult: %d", result);
    return result;
}
```

Test Ausgaben: (Start values -> (3, 8), (-4, 9), (0, 8), (10, 1))

```
#### russianMul ############################################
-------------------------------------------------
multiplier: 3, multiplicand: 8

multiplier > 0 && multiplicand > 0
calculating...
finished

result: 24
-------------------------------------------------
multiplier: -4, multiplicand: 0

multiplier, multiplicand or both <= 0

result: 0
-------------------------------------------------
multiplier: 0, multiplicand: 8

multiplier, multiplicand or both <= 0

result: 0
-------------------------------------------------
multiplier: 10, multiplicand: 1

multiplier > 0 && multiplicand > 0
calculating...
finished

result: 10
-------------------------------------------------
```

4 Umrechnung einer Dezimalzahl in eine Binärzahl

LösungsIdee:

Es wird angenommen, dass die Binärzahl max 64Bit groß ist. Wenn die Dezimalzahl > 0 ist wird das binary Array mit 64 ‚0' Einträge erstellt. Zusätzlich wird noch geprüft ob die Dezimalzahl positiv ist. Die Dezimalzahl wird anschließend in einer Schleife immer durch 2 geteilt und der Rest in das binary Array gespeichert. Um die gespeicherten Einträge richtig darzustellen wird in einer Schleife das binary Array ‚rückwärts' durchloffen und in jeder iteration den aktuellen Array wert augegeben.

Testfälle: 0, 33, -10

C-Program code:

Int main:

```c
int main(int argc, char* argv[]) {
    int test_convertDecimalToBinary = 1;
    int test_russianPeasantMultiplication = 0;

    // convert Decimal to Binary ------------------------------------------------------------
    #pragma region convertDecimalToBinary
    if (test_convertDecimalToBinary) {
        printf("\n#### decToBin ##############################################");

        // test 1 -> value: 0 --------------------
        convertDecimalToBinary(0);

        // test 2 -> value: 33 -------------------
        convertDecimalToBinary(33);

        // test 3 -> value: -10 ------------------
        convertDecimalToBinary(-10);

        printf("\n-----------------------------------------------");
    }
    #pragma endregion

    // russian peasant multiplication -------------------------------------------------------
    russianPeasantMultiplication

    // alternativ
    getMissingNumber

    // just to make it more beautiful -------------------------------------------------------
    printf("\n\n");
    // --------------------------------------------------------------------------------------

    return 0;
}
```

Int convertDecimalToBinary:

```c
int convertDecimalToBinary(int decNumber) {
    printf("\n-----------------------------------------------");
    printf("\ndecNumber: %d", decNumber);

    if (decNumber == 0) {
        printf("\ndecNumber == 0");
        printf("\nBinary of decNumber is = 0");
    } else {
        printf("\ndecNumber != 0");

        // prepare binary array
        int binaryValues[64] = { 0 };

        int isPositiv = 0;
        if (decNumber >= 0) {
            printf("\ndecNumber >= 0 --> decNumber is Positive");
            isPositiv = 1;
        } else {
            printf("\ndecNumber < 0 --> decNumber is Negative");
            decNumber *= (-1);
        }

        // convert ------------------------------------------------------------------
        printf("\n\nconverting decNumber to binary...");
        int i = 0;
        for (i = 0; decNumber > 0; i++) {
            binaryValues[i] = decNumber % 2;
            decNumber = decNumber / 2;
        }
        printf("\nconverting finished\n");

        if (isPositiv) {
            printf("\nBinary of decNumber is = ");
        } else {
            printf("\nBinary of decNumber is = -");
        }

        // print binary numbers in the right order ----------------------------------
        for (i = i - 1; i >= 0; i--) {
            printf("%d", binaryValues[i]);
        }
    }

    return 0;
}
```

Test Ausgaben: (Start values -> 0, 33, -10)

```
#### decToBin ##############################################
-----------------------------------------------
decNumber: 0
decNumber == 0
Binary of decNumber is = 0
-----------------------------------------------
decNumber: 33
decNumber != 0
decNumber >= 0 --> decNumber is Positive

converting decNumber to binary...
converting finished

Binary of decNumber is = 100001
-----------------------------------------------
decNumber: -10
decNumber != 0
decNumber < 0 --> decNumber is Negative

converting decNumber to binary...
converting finished

Binary of decNumber is = -1010
-----------------------------------------------
```