Duration: 120 minutes.
External Material: you are not allowed to use any written, printed or electronic external material.

Guidelines and Instructions:

- Write your answers only on the exam form, in the intended places. Note that the given place doesn't necessarily indicate the length of the correct answer.
- The even pages of the exam form are empty. You can use them as a draft or to write your answers. Write drafts clearly, so they won't be checked.
- Write your answers tidy, clean and clearly as possible. You're allowed to use pencil and an eraser, but you must fill the title page in pen.
- In all of the questions, you're allowed to define (and implement) your own functions.
- You are not allowed to use global and/or static variables, or pre-compilation commands (include\define).
- You are not allowed to use library functions, or functions implemented in class, without implementing them yourself, unless noted explicitly in the question, excluding input/output functions.
- In every question, you are allowed to use functions defined in previous parts of the same question, even if you didn't solve these parts, though this is not obligated.
- You don't need to check input correction, unless explicitly noted in the question.
- You don't need to implement the main function, and the order of writing the functions is not important.
- the complexity of the solution will not be marked, unless explicitly stated.
- It's recommended to add a written explanation of your algorithm. That explanation will not be marked.
**Question 1 (20 Points)**

**Part 1 (7 Points)**

Explain (general explanation) what the following function will print for the arguments \(a, b\):

```c
void print(int a, int b)
{
    a = a + b;
    b = a - b;
    a = a - b;
    printf("%d %d\n", a, b);
}
```

"the original value in \(b\) will go into \(a\), and the original value in \(a\) will go into \(b\), so what will be printed is the original value of \(b\), a space, then the original value of \(a\):

\(b\ a\)

**Part 2 (13 Points)**

Notice the given function, manipulate \((\text{int } a[], \text{int } n, \text{int } z)\) and the array \(b\), defined after the function.

```c
void manipulate(int a[], int n, int z){
    int x = 0, y = n - 1;
    while (x < y) {
        while (!a[x]%z) ++x;
        while (a[y]%z) y--;
        swap(&a[y], &a[x]); // The usual swap() function
    }
    return;
}
```

```c
int b[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

what will be the content of \(b\) after the following activation of the function:

manipulate\((b, 10, 3)\);

\(b = \)

```
0 9 6 4 3 5 2 7 8 1
```
Question 2 (25 Points)
A number \( n \) will be called Perfect if it's equal to the sum of its divisors. (the ones smaller than itself).

Examples:
- 6 is perfect, because \( 6 = 1 + 2 + 3 \).
- 28 is perfect, because \( 28 = 1 + 2 + 4 + 7 + 14 \).

The perfect number in the following part is not greater than 400.

Part 1 (12 Points)
Write the following function:

\begin{verbatim}
int isPerfect(int n)
{
    int i, sum = 0;
    for(i=1;i<n;i++)
        if(n%i==0)
            sum += i;
    return (sum == n);
}
\end{verbatim}

Its parameter is an integer \( n \). the function returns 1 if \( n \) is a perfect number, and 0 if it's not.

Example – for \( n = 6 \), the function will return 1, and for \( n = 5 \) it will return 0.
Part 2 (13 Points)
Write the following function:

```c
int perfectRowColumn(int a[][N])
```

Its parameter is a **Square** matrix NxN of integers. The function returns a number k. k is a number of a row and a column that contain only perfect numbers. If there is no such k, the function will return N.

**You will get a bonus of 5 points** for an economical implementation (i.e. your solution doesn't fully scan some of the rows/columns).

**You have to use the function isPerfect from the previous part, even if you didn't implement it.**

```c
int perfectRowColumn(int a[][N]){
    int i, j;
    for (i = 0; i < N; ++i) {
        int candidate = 1;
        // if (!isPerfect(a[i,i]) continue;
        for (j = 0; candidate && j < N; ++j) {
            if (!isPerfect(a[i,j]) candidate = 0;
        }
        for (j = 0; candidate && j < N; ++j) {
            if (!isPerfect(a[j,i]) candidate = 0;
        }
        if (candidate) return i;
    }
    return N;
}
```
Question 3 (30 points)

In this question we’ll use strings that contain long addition and subtraction exercises, like "5+6-7+8-9".

Note – the only characters in the string are digits, the operators ‘+’ and ‘-’, and spaces.
You can assume the string represents a legal argument – the are no operators in the beginning or the end, and between every two operators, there’s one number.

Part 1 (12 Points)

Write the following function:

```c
int evaluate(char *s){
    int r=0;
    int op=1;
    while (*s)
    {
        if (*s=='+') op=1;
        else if (*s=='-') op=-1;
        else if (*s!=' ') r+=op*(*s-'0');
        s++;
    }
    return r;
}
```

Its parameter is a string like stated before. The function returns the value of the expression in the string (i.e. the result of the calculation).

Notice that every number is one digit. In this part there are no negative numbers. Every +/- is a mathematical operator.

Examples:

- The string "5+6-7+8-9" will return 3.
- The string "5+4-1 + 2" will return 10.
- The string "8 - 9" will return -1.

```c
int evaluate(char *s){
    int r=0;
    int op=1;
    while (*s)
    {
        if (*s=='+') op=1;
        else if (*s=='-') op=-1;
        else if (*s!=' ') r+=op*(*s-'0');
        s++;
    }
    return r;
}
```
Part 2 (15 points)

Write the following **recursive** function (no loops allowed!):

```c
int count(char *s) {
  Its parameter is a string s. the function counts the number of elements (**numbers**) in the expression.

  Notice that in this part, any number **can be more than one digit**, and a number can have a **negation sign** before it (indicating that it's a negative number).

  **Examples:**
  - The string "5+61-7+38-9" will return 5.
  - The string "5+44-1+2" will return 4.
  - The string "21--3" will return 2.

  ```c
  int count(char *s) {

    if (s[0] == '0') return 0;
    if ((s[0] >= '0' && s[0] <= '9') && (s[1] < '0' || s[1] > '9'))
        return 1 + count(s+1);
    else
        return count(s+1);

  }
  ```
Question 4 (25 Points)
We'll refine an array of pairs as an array of even length n (n=2*m), and pairs of adjacent elements (in indexes (0,1),(2,3)…) are treated as one unit.

Example:

| 1 | 9 | 2 | -5 | 7 | 20 | -10 | -9 | -12 | 12 |

Part 1 (15 Points)
Write the following function:

```c
void pairSort(int a[], int n)
```

Its parameters are an array of pairs a, and it's size n (n is even). The function sorts the pairs in the array, in increasing order of the sums of the pairs.

Example: the array mentioned in the example above will be sorted to:

| -10 | -9 | 2 | -5 | -12 | 12 | 1 | 9 | 7 | 20 |

```c
void pairSort(int a[], int n){
    int i,size;
    for(size = n; size > 2; size -= 2) {
        for (i=2; i< size ; i+=2) {
            if(a[i-2]+a[i-1] > a[i]+ a[i+1]) {
                swap(a+i-2,a+i); /*The regular swap function, had to be written if used */
                swap(a+i-1,a+i+1);
            }
        }
    }
}
```
Part 2 (10 points)

Write the following function:

```c
int uniqCouplesNum(int a[], int n)
```

Its parameters are an array of pairs, a, and its size, n (n is even). The function returns the amount of different sums of its pairs.

**Example**: for the following array:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>-5</th>
<th>-10</th>
<th>7</th>
<th>-10</th>
<th>9</th>
<th>7</th>
<th>20</th>
<th>5</th>
<th>-6</th>
</tr>
</thead>
</table>

The function will return 4, because the **sums of the pairs** are -1, 27, -1, -3, -3, 10, and there are four different sums: 10, -3, -1, 27.

**It's recommended to use the function from part 1.**

```c
int uniqCouplesNum(int a[], int n){
    int i, count = 1;
    if(n==0)
        return 0;
    pairSort(a,n);
    for(i=0; i < n-2; i+=2)
    {
        if(a[i]+a[i+1]!=a[i+2]+a[i+3])
            count++;
    }
    return count;
}
```