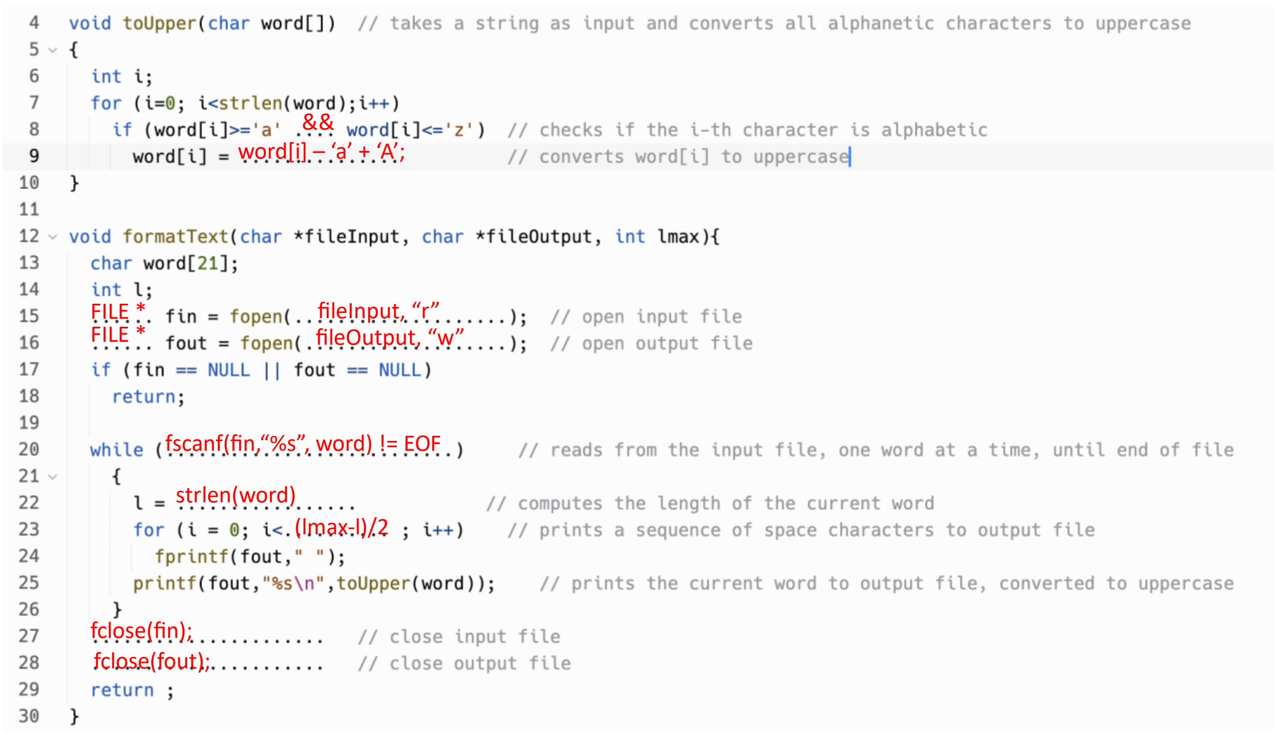
EXAMPLES OF EXAM EXERCISES (PROGRAMMING SECTION)

EXAMPLE (4 points)

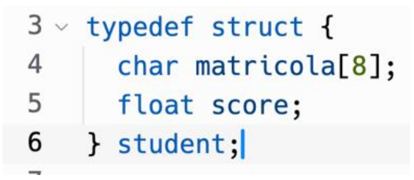
In the following code, the function formatText reads a sequence of words from a text file, whose name is passed as the first parameter. The words are separated by either spaces or newlines, and do not contain any spaces. The function prints the words in a second file, whose name is passed as second parameter. Each word should be converted to all uppercase characters, and then printed one per line and centered within the line. The length of the line, lmax, is passed as the third parameter to the function.
Complete the given code (see dashed lines), as indicated by the comments:



```
void toUpper(char word[])  // takes a string as input and converts all alphanetic characters to uppercase
{
  int i;
  for (i=0; i<strlen(word);i++)
    if (word[i]>='a' && word[i]<='z')  // checks if the i-th character is alphabetic
      word[i] = word[i] – ‘a’ + ‘A’;           // converts word[i] to uppercase
}

void formatText(char *fileInput, char *fileOutput, int lmax){
  char word[21];
  int l;
  FILE * fin = fopen(fileInput, “r”);  // open input file
  FILE * fout = fopen(fileOutput, “w”);  // open output file
  if (fin == NULL || fout == NULL)
    return;

  while (fscanf(fin,“%s”, word) != EOF)     // reads from the input file, one word at a time, until end of file
    {
      l = strlen(word)                 // computes the length of the current word
      for (i = 0; i<(lmax-l)/2 ; i++)     // prints a sequence of space characters to output file
        fprintf(fout," ");
      printf(fout,"%s\n",toUpper(word));     // prints the current word to output file, converted to uppercase
    }
  fclose(fin);             // close input file
  fclose(fout);            // close output file
  return ;
}
```

EXAMPLE (4 points)

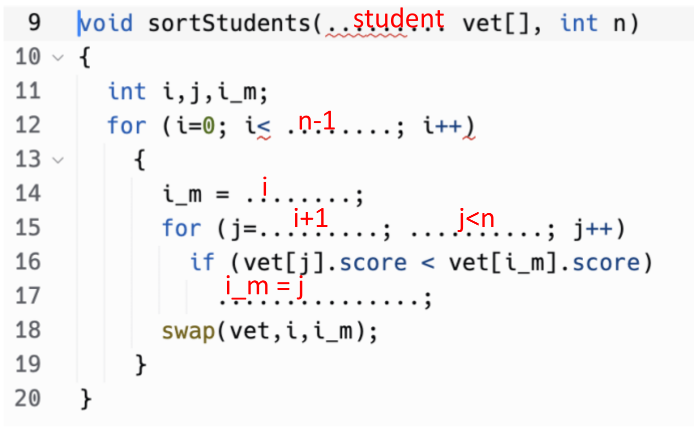
In a C program, a student is represented by the following struct:



```
typedef struct {
  char matricola[8];
  float score;
} student;
```

The function sortStudents receives an array of students and the number of students as the input parameters, and sorts the array by increasing scores, using selection sort as the sorting algorithm.

1) Complete the given code (see dashed lines):



```
 9  void sortStudents(student vet[], int n)
10  {
11    int i,j,i_m;
12    for (i=0; i< n-1; i++)
13     {
14       i_m = i;
15       for (j=i+1; j<n; j++)
16         if (vet[j].score < vet[i_m].score)
17           i_m = j;
18       swap(vet,i,i_m);
19     }
20  }
```

2) Implement the function *swap*, called at line 18:

```
void swap(student v[], int a, int b)
{
   student tmp;
   tmp = v[a];
   v[a] = v[b];
   v[b] = tmp;
}
```

3) Change line 16 to have the students sorted by **decreasing matricola**:

```
if (strcmp(vet[j].matricola,vet[i_m].matricola)>0)
```

EXAMPLE (4 points)

The function *modifyText* receives two filenames (input and output file, respectively) as parameters. The function should read a text from the first file and copy it to a second file, making sure that each word starts with an uppercase character. The code of the function is given, where some parts/instructions are missing. Complete the given code as requested in the comments:

```c
int modifyText(char* fileName1, char* fileName2)
{
  FILE *... fpIn = fopen(fileName1, “r”......);  // open input file
  FILE *... fpOut = fopen(.fileName2, “w”...);  // open output file
  int... n = 0;
  char... ch,tmp;

  if (fpIn == NULL || fpOut == NULL)
    return 0;

  while(fscanf(fpIn, “%c”, &ch) != EOF) {  // reads one character at a time, until the end of the file
    if (ch>='a' && ch<='z')
      if (n==0 || isspace(tmp))
        ch = ch – ‘a’ + ‘A’;..............  // uppercase conversion
    fprintf(fpOut, “%c”, ch);.....          // prints to output file
    tmp = ch;
    n++;
  }
  fclose(fpIn); fclose(fpOut);
  return(n);.....                          // returns the number of printed characters
}
```

EXAMPLE (6 points)

Given the following code:

```c
1   #include <stdio.h>
2
3   ..void... change(...int *..vet, ..int.. n)
4   {
5     int i;
6     for (i=0; i<n; i++)
7       *(vet + 1) = *(vet + i) + 5;
8   }
9
10  int main()
11  {
12    int i, a[] = {2, 4, 6, 8, 10};
13    change(a, 5);
14    for (i=0; i<5; i++)
15      printf("%d ",a[i]);
16    return 0;
17  }
18
19
```

1) Complete the header of the function *change* at line 3:

2) What would the output of the program be?

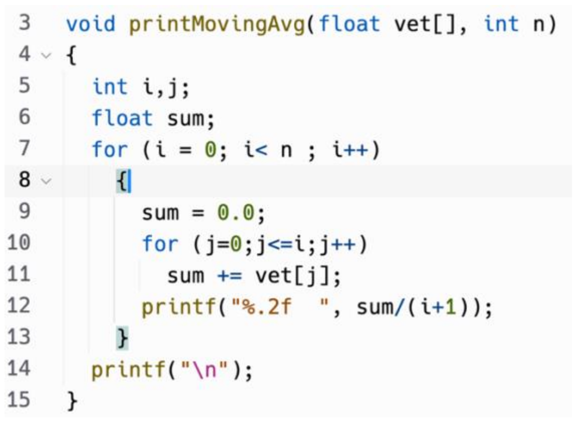
2 15 6 8 10

3) How would you change line 7, so that the new output of the program is: 7 9 11 13 15?

*(vet + i) = *(vet + i) + 5;

EXAMPLE (6 points)

See the following function:



```
void printMovingAvg(float vet[], int n)
{
  int i,j;
  float sum;
  for (i = 0; i< n ; i++)
    {
      sum = 0.0;
      for (j=0;j<=i;j++)
        sum += vet[j];
      printf("%.2f  ", sum/(i+1));
    }
  printf("\n");
}
```

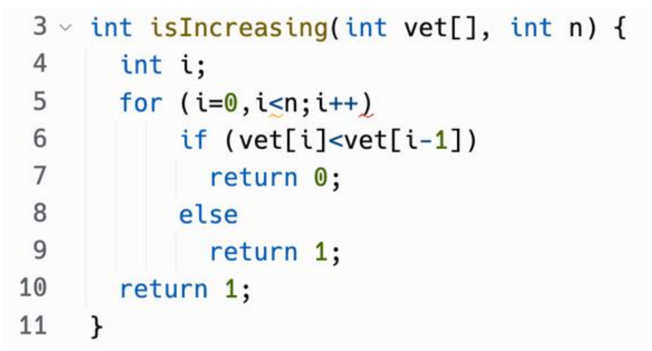
1) What is the output printed by the function when vet = {1,2,3,4} and n=3?
1.00 1.50 2.00

2) Modify the function to reduce its time complexity to O(N). NB You cannot add variables to the ones used in the original version:

```
void printMovingAvg(float vet[], int n)
{
  int i,j;
  float sum = 0.0;
  for (i=0; i<n; i++)
  {
    sum += vet[i];
    printf("%.2f   ",sum/(i+1));
  }
  printf("\n");
}
```

EXAMPLE (6 points)

The following function is supposed to verify whether an input sequence of integers is monotonically increasing:



```
 3  int isIncreasing(int vet[], int n) {
 4    int i;
 5    for (i=0,i<n;i++)
 6        if (vet[i]<vet[i-1])
 7          return 0;
 8        else
 9          return 1;
10    return 1;
11  }
```

1) Is the function correct, syntactically and/or logically? List and explain all the errors, clearly mentioning which line of the code you are referring to.

   - Error at line 5: the initialization of the for loop should be i=1 to avoid accessing the array vet at index -1.
   - Error at lines 8-9: *else return 1* is wrong. The function should return 1 only after all the pairs of consecutive values have been verified.

2) Write a corrected version (do not make any modifications apart from the ones that are strictly necessary for the function to work):

```
int isIncreasing(int vet[], int n) {
 int i;
 for (i=1;i<n;i++)
     if (vet[i]<vet[i-1])
         return 0;
 return 1;
}
```

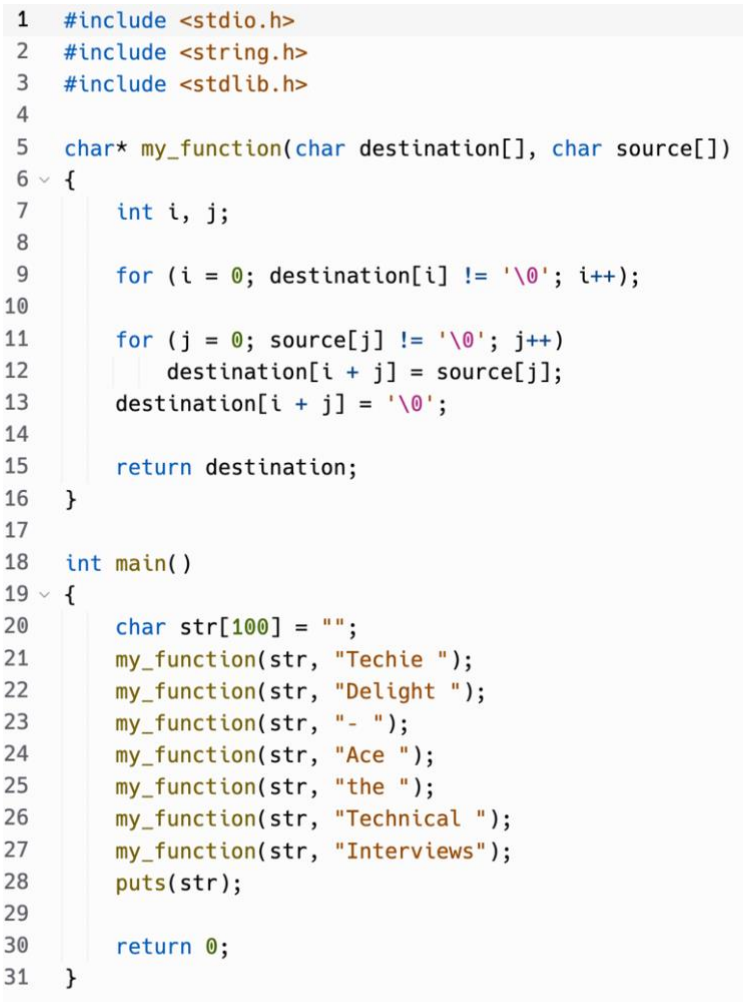
3) Even after the corrections made at point 2, the given function is not structured. Why? Write a structured version of the function:

   The function is not structured, as there are two return statements. It can be made structured by means of a flag:

```
int isIncreasing(int vet[], int n) {
 int i,increasing=1;
 for (i=1;i<n && increasing;i++)
     if (vet[i]<vet[i-1])
         increasing = 0;
 return increasing;
}
```

EXAMPLE (6 points)

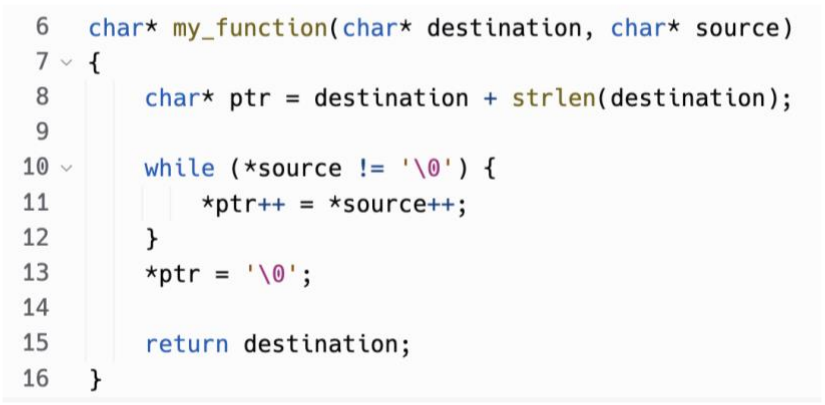
1) Is the following program correct, syntactically and/or logically? What is the output?



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char* my_function(char destination[], char source[])
{
    int i, j;

    for (i = 0; destination[i] != '\0'; i++);

    for (j = 0; source[j] != '\0'; j++)
        destination[i + j] = source[j];
    destination[i + j] = '\0';

    return destination;
}

int main()
{
    char str[100] = "";
    my_function(str, "Techie ");
    my_function(str, "Delight ");
    my_function(str, "- ");
    my_function(str, "Ace ");
    my_function(str, "the ");
    my_function(str, "Technical ");
    my_function(str, "Interviews");
    puts(str);

    return 0;
}
```

The function is correct. The output is the following:
Techie Delight – Ace the Technical Interviews

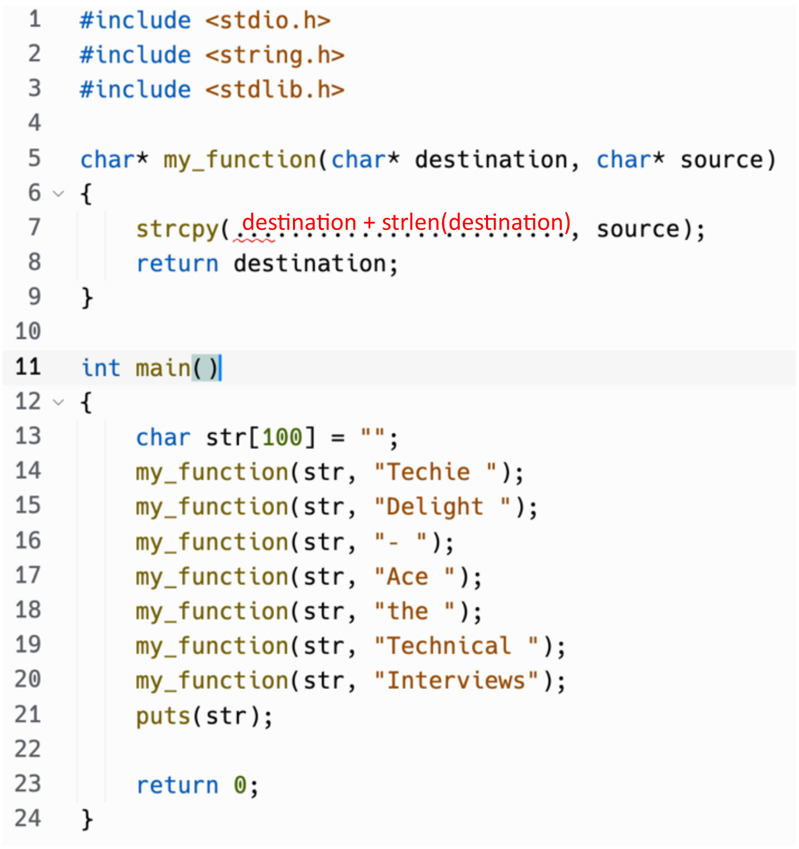
2) Is the following modification of my_function correct, syntactically and/or logically? How does the output of the program change?



```c
char* my_function(char* destination, char* source)
{
    char* ptr = destination + strlen(destination);

    while (*source != '\0') {
        *ptr++ = *source++;
    }
    *ptr = '\0';

    return destination;
}
```

The function is correct. The output is the same as before:
Techie Delight – Ace the Technical Interviews

3) Complete the code at line 7, so that the program provides the same output of the one at point 1:



```c
 1  #include <stdio.h>
 2  #include <string.h>
 3  #include <stdlib.h>
 4
 5  char* my_function(char* destination, char* source)
 6  {
 7      strcpy(destination + strlen(destination), source);
 8      return destination;
 9  }
10
11  int main()
12  {
13      char str[100] = "";
14      my_function(str, "Techie ");
15      my_function(str, "Delight ");
16      my_function(str, "- ");
17      my_function(str, "Ace ");
18      my_function(str, "the ");
19      my_function(str, "Technical ");
20      my_function(str, "Interviews");
21      puts(str);
22
23      return 0;
24  }
```

EXAMPLE (6 points)

In the following, you see six programs. For each of them, you need to say whether it can be compiled and executed or would end up with some error. If you think it can be executed, write in the table below the corresponding output printed on the screen. Otherwise, briefly explain the problem.

```c
#include <stdio.h>
void function1(char* s1, char* s2, char* res) {
    while ((*res++ = *s1++)!='\0');
    res--;
    while ((*res++ = *s2++)!='\0'); }
int main() {
    char a[] = "ab", b[] = "cd", c[50];
    function1(a,b,c);
    printf("%s",c);
    return 0;}
```

1

```c
#include <stdio.h>
void function2(char* s1, char* s2, char* res) {
    while ((*res++ = *s1++)!='\0');
    while ((*res++ = *s2++)!='\0'); }
int main() {
    char a[] = "ab", b[] = "cd", c[50];
    function2(a,b,c);
    printf("%s",c);
    return 0;}
```

2

```c
#include <stdio.h>
char* function3(char* s1, char* s2) {
  while ((*s1++ == *s2++)!='\0');
  return s2-1;}
int main() {
  char a[] = "ab", b[] = "abcd";
  printf("%s",function3(a,b));
  return 0;}
```

3

```c
#include <stdio.h>
int main() {
  char a[] = "ab", b[] = "abcd";
  while ((*a++ == *b++)!='\0');
  printf("%s",b-1);
  return 0;}
```

4

```c
#include <stdio.h>
void function4(char *s, char c) {
    char *w;
    for (w=s;*s!='\0'; s++)
        if (*s != c)
            *w++ = *s;
    *w = '\0'; }
int main() {
    char myS[] = "Hello, World!";
    function4(myS, 'o');
    printf("%s", myS);
    return 0;}
```

5

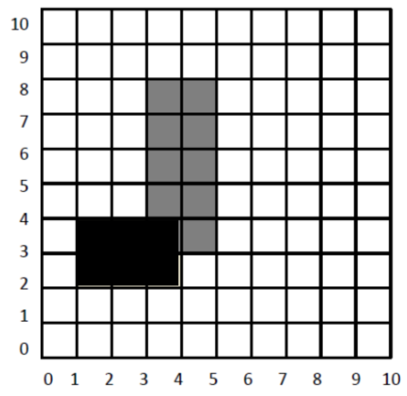
```c
#include <stdio.h>
void function5(char *s, char c) {
    for (char* w=s;*s!='\0'; s++)
        if (*s != c)
            *w++ = *s;
    *w = '\0'; }
int main() {
    char myS[] = "Hello, World!";
    function5(myS, 'o');
    printf("%s", myS);
    return 0;}
```

6

	EXECUTE? (YES/NO)	**IF YES, WRITE THE CORRESPONDING OUTPUT ON THE SCREEN. IF NO, BRIEFLY EXPLAIN THE ERROR.**
1	YES	abcd
2	YES	ab
3	YES	cd
4	NO	a and b are arrays, they cannot be incremented
5	YES	Hell, Wrld!
6	NO	The scope of the variable w ends with the for loop. *w = ‘\0’ will end up into error.

EXAMPLE (8 points)

Consider a square region in the first quadrant of the Cartesian plane whose bottom left and upper right corners have coordinates (0,0) and (100,100), respectively. A text file contains a sequence of rectangles, one per line, with sides parallel to the Cartesian axes. Each rectangle is identified by the x and y coordinates of the bottom left and upper right corners, reported with spaces as separators. The coordinates are integers between 0 and 100, with extremes included. Write the function `int areaTot (FILE * fp);` which receives as a parameter a pointer to the file (already open) and returns the total area covered by the rectangles. In case of intersection of rectangles, the area of the intersecting region should be counted only once.
Example (for the sake of simplicity, we report a smaller region with x and y coordinates in the interval 0..10)



If you assume that the file of the example contains 1 2 4 4 on the first line (see black rectangle) and 3 3 5 8 in the second line (see grey rectangle), the total area returned by the function should be 15.

```
int areaTot(FILE *fp){

  int mat[NMAX][NMAX] = {0};  // Region: 100x100 matrix of 0 elements
  int x1,y1,x2,y2,i1,i2,j1,j2,i,j;
  int area = 0;

  // For each rectangle in the file
  while(fscanf(fp,"%d%d%d%d",&x1,&y1,&x2,&y2)!=EOF)
  {
    // Convert cartesian coordinates to row-column indexes
    i1 = NMAX-y1-1; // row-index of lower-left corner
    j1 = x1;        // column-index of lower-left corner
    i2 = NMAX-y2;   // row index of upper-right corner
    j2 = x2-1;      // column index of upper-right corner
    // For each element of the rectangle
    for (i=i2;i<=i1;i++)
      for (j=j1;j<=j2;j++)
        if (mat[i][j]==0)  // Check if the element is "free"
        {
          mat[i][j]++;   // If so, fill with a 1
```

```
                area++;     // ... and increment the area by 1
            }
        }
        return area;
}
```

EXAMPLE (8 points)

Assume that you have been given a vector `v` of `N` integers whose elements represent, in a compressed format, a sequence of numbers that must be inserted in a matrix `M` of integers of r rows and `c` columns, according to the row-major strategy (i.e. by rows). The vector contains the sequences of values corresponding to the rows of the matrix, with an encoding that tries to compact the repeated data.
To decode the sequence, the integers of the vector must be considered in pairs (v [0], v [1]), (v [2], v [3]), (v [4], v [5]), etc. Given v [i] and v [i + 1], v [i] represents the integer value and v [i + 1] the number of repetitions (i.e. how many times v [i] must repeat on the same row of matrix M, or on the next row if the current row ends).

Implement a C function to decode the vector, store the decoded (i.e. uncompressed) sequences in the matrix and print its content on the screen. The function should have the following prototype:

```
int buildMatrix(int V[], int N, int M[MAXR][MAXC], int nr, int nc);
```

The function should check that the array `V` is valid (i.e., that its size is compatible with the size of the already existing matrix `M`, so that all the nr x nc cells are filled: the function should return 1 if the size of the array is valid, 0 otherwise.

Example. Assume `nr` = 3, `nc` = 5, `N` = 14 and `V` = (1, 2, 17, 2, 3, 1, 8, 4, 6, 1, 7, 3, 5, 2): the matrix `M` should have the following content:

$$M = \begin{pmatrix} 1 & 1 & 17 & 17 & 3 \\ 8 & 8 & 8 & 8 & 6 \\ 7 & 7 & 7 & 5 & 5 \end{pmatrix}$$

```
int buildMatrix(int V[], int N, int M[MAXR][MAXC], int nr, int nc) {
        int i, j, k, x, y;
        k=0; // Counter of the decoded values
        for(i=0;i<N;i=i+2) {
           int val = V[i]; // Value
           int rip = V[i+1];  // Number of repetitions
           for(j=0;j<rip;j++) {  // Per each repetition
              x = k / nc;  // Row-index
              y = k % nc;  // Column-index
              M[x][y] = val;  // Insert value in the matrix
              k++;          // Increment counter
           }
        }
        return k==nr*nc;    // Counter should be equal to number of cells of the matrix
}
```

EXAMPLE (8 points)

In a championship `n` (max value: 20) teams play for `m` (max value: 20) days. The outcome of the matches are stored in a matrix of `n × m` integers, whose elements can only be either 0, 1 or 3. Each `i-th` row of the matrix represents the points acquired by the `i-th` team in the m days of the championship. Each `j-th` column of the matrix represents the points acquired by the n teams in the `j-th` day of championship. The points acquired are 3 for a won game, 1 for tied and 0 for a defeat, respectively.

Write a C function with the following prototype:

```
        void displRanking(int C[MAXN][MAXM], int n, int m);
```

For each matchday of the championship, the function should print on the screen the index (i.e., the corresponding row number) of the team that is currently leading the championship (i.e., the team that has the highest sum of points since the beginning of the championship). In case there is more than one leading team, it should print the first that is found.

<u>Example</u>:
Given the following matrix with n=4 and m=3

3	1	0
0	1	1
1	1	1
1	1	3

The output should be:

```
The leading team in each of the 3 matchdays is: 0 0 3
```

```
    int searchMax(int v[], int n) {
        int max = -1, i, maxi = -1;
        for(i=0;i<n;i++) {
            if (max < v[i]) {
                max = v[i];
                maxi = i;
            }
        }
        return maxi;
    }

    void displRanking(int C[MAXN][MAXM], int n, int m) {
        int i, j, points[MAXN] = {0};

        printf("The leading team for each of the %d days is: ",m);
        /* for each day of championship*/
        for(j=0;j<m;j++) {
            /* for each team */
            for(i=0;i<n;i++)
                points[i] += C[i][j];
            // points now contains the current scores of the different teams in the given day
            printf("%d ",searchMax(points, n));  // find the currently leading team
        }
        return;
    }
```