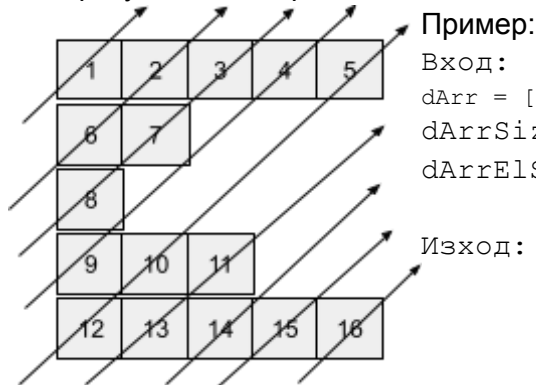


\* **Работата с паметта (заделяне и освобождаване) е ваша отговорност. В задачите е позволено използване на `cstring` и помощни функции. За решението на задачите НЕ е позволено използването на `std::string` или други структури от данни. При оценяването на решенията ще се вземе предвид подредеността и спазването на добрите практики за писане на код.**

### Задача 1.

- а) (1 т.) Да се напише функция `<тип> getDiagArr(<тип> result, <тип> dArr, <тип> dArrSize, <тип> dArrElSizes)`, която по подаден динамичен масив от динамични масиви от цели числа (`dArr`) и масив от техните размери (`dArrElSizes`), създава нов масив, който съдържа елементите на `dArr` след диагоналното му обхождане. Резултатният масив да се записва в `result`.
- б) (0.5 т.) Да се напише програма, която създава динамичен масив от масиви със стойности въведени от клавиатурата, заделя памет за резултатния масив, извиква функцията `getDiagArr` и отпечатва резултата на екрана.



Пример:

Вход:

`dArr = [[1, 2, 3, 4, 5], [6, 7], [8], [9, 10, 11], [12, 13, 14, 15, 16]]`

`dArrSize = 5`

`dArrElSizes = [5, 2, 1, 3, 5]`

Изход: [1, 6, 2, 8, 7, 3, 9, 4, 12, 10, 5, 13, 11, 14, 15, 16]

**Задача 2. (1.5 т.)** Да се напише **функция**, която приема като вход изречение с **максимална дължина 128**, създава нова криптирана версия на изречението и връща цената от криптирането по следните правила:

- премахват се всички малки и главни отворени гласни ('a', 'o'). Цената на изтриване на гласна буква е 1;
- заместват се всички **числа** с 'X'. Цената на заместване на число е сумата на цифрите му;

\* Криптираната версия се записва в нов статичен низ с нужния капацитет, който се подава като параметър на функцията.

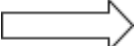
Пример:

"I was born in 1989 in the city of YORK" ⇒ "I ws brn in X in the city f YRK", цена: 31

(цена 4 за гласните, цена 27 за цифрите)

**Задача 3. (1 т.)** Да се напише булева функция, която **по указател** към началото на низ от малки латински букви, връща дали има последователност от еднакви символи в низа. Функцията да записва И указателите към началото и края на най-дългата последователност от еднакви символи. Ако има две последователности с еднаква дължина, указателите да сочат към първата от тях. Стойностите на указателите да остават видими и след края на изпълнение на функцията.

Пример:

"pbfdaaaabcssll"  връща "истина", указател към първото 'a' и последното 'a'.

\* **Работата с паметта (заделяне и освобождаване) е ваша отговорност. В задачите е позволено използване на `cstring` и помощни функции. За решението на задачите НЕ е позволено използването на `std::string` или други структури от данни. При оценяването на решенията ще се вземе предвид подредеността и спазването на добрите практики за писане на код.**

**Задача 1. (1.5 т.)** Да се напише **функция**, която приема като вход изречение с **максимална дължина 128**, създава нова криптирана версия на изречението и връща цената от криптирането по следните правила:

- премахват се всички малки и главни затворени гласни ('u', 'e', 'i'). Цената на изтриване на гласна буква е 1;
- заместват се всички **числа** с 'X'. Цената на заместване на число е произведението на цифрите му;

\* **Криптираната версия се записва в нов статичен низ с нужния капацитет, който се подава като параметър.**

Пример:

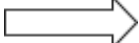
"I was born in 1981 in the city of YORK" ⇒ " was born n X n th cty of YORK", цена: 77

(цена 5 за гласните, цена 72 за цифрите)

**Задача 2. (1 т.)** Да се напише булева функция, която **по указател** към началото на низ от главни латински букви, връща дали има последователност от еднакви символи в низа. Функцията да записва И указателите към началото и края на най-дългата последователност от еднакви символи. Ако има две последователности с еднаква дължина, указателите да сочат към втората от тях.

\* **Стойностите на указателите да остават видими и след края на изпълнение на функцията.**

Пример:

"PBDFAAABCSSL"  връща "истина", указател към първото 'A' и последното 'A'.

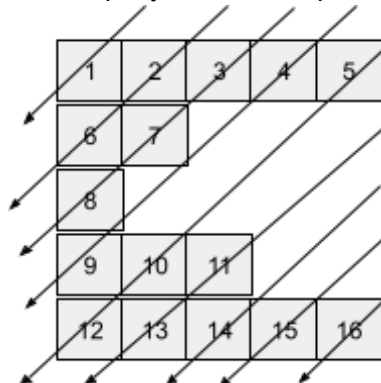
**Задача 3.**

- **(1 т.)** Да се напише функция `<тип> getDiagArr(<тип> result, <тип> dArr, <тип> dArrSize, <тип> dArrElSizes)`, която по подаден динамичен масив от динамични масиви от цели числа (`dArr`) и масив от техните размери (`dArrElSizes`), създава нов масив, който съдържа елементите на `dArr` след диагоналното му обхождане. Резултатният масив да се записва в `result`.
- **(0.5 т.)** Да се напише програма, която създава динамичен масив от масиви със стойности въведени от клавиатурата, заделя памет за резултатния масив, извиква функцията `getDiagArr` и отпечатва резултата на екрана.

Пример:

Вход:  
`dArr = [[1, 2, 3, 4, 5], [6, 7], [8], [9, 10, 11], [12, 13, 14, 15, 16]]`  
`dArrSize: 5`  
`dArrElSizes: [5, 2, 8, 3, 5]`

Изход: [1, 2, 6, 3, 7, 8, 4, 9, 5, 10, 12, 11, 13, 14, 15, 16]



Примерни решения:

1)

```
#include <iostream>

// for testing purposes, should be int**
void findDiagonalOrder(int *result, int dArr[][5], int *dArrLengths, int size)
{
    int maxLen = 0;
    for (int i = 0; i < size; i++)
    {
        if (maxLen < dArrLengths[i])
        {
            maxLen = dArrLengths[i];
        }
    }

    int resultIndex = 0;

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < maxLen; j++)
        {
            if (i - j < 0)
            {
                break;
            }
            if (j < dArrLengths[i - j])
            {
                result[resultIndex++] = dArr[i - j][j];
            }
        }
    }

    int i = size - 1;
    for (int j = 1; j < maxLen; j++)
    {
        for (int k = 0; k < maxLen; k++)
        {
            if (i - k >= 0 && j + k < dArrLengths[i - k])
            {
                result[resultIndex++] = dArr[i - k][j + k];
            }
        }
    }
}

int main()
{
```

```

int dArr[][5] = {{1, 2, 3, 4, 5}, {6, 7}, {8}, {9, 10, 11}, {12, 13, 14, 15, 16}},
dArrLengths[] = {5, 2, 1, 3, 5}, size = sizeof(dArrLengths) / sizeof(dArrLengths[0]);
const int TOTAL_ITEMS = 5 + 2 + 1 + 3 + 5;
int result[TOTAL_ITEMS] = {0};
findDiagonalOrder(result, dArr, dArrLengths, size);
for (size_t i = 0; i < TOTAL_ITEMS; i++)
{
std::cout << result[i] << ' ';
}
}

```

2)

```

#include <iostream>
#include <cstring>
#include <cassert>

int createNumber(const char *str, int &i)
{
int num = 0;
while (isdigit(str[i]))
{
num = num * 10 + str[i] - '0';
i++;
}
i--;
return num;
}

int digitSum(int x)
{
int sum = 0;
while (x)
{
sum += x % 10;
x /= 10;
}

return sum;
}

int cypherCost(const char *input, char *output)
{
int size = strlen(input), cost = 0, j = 0;
for (int i = 0; i < size; i++)
{
char c = input[i];
if (c == 'a' or c == 'o' or c == 'A' or c == 'O')
{
cost++;
continue;
}
if (isdigit(c))
{
int num = createNumber(input, i);

```

```

cost += digitSum(num);
output[j++] = 'X';
continue;
}
output[j++] = c;
}
output[j++] = '\\0';

return cost;
}

int main()
{
const int MAX_SIZE = 128;
// char input[MAX_SIZE];
// std::cin.getline(input, MAX_SIZE);

char output[MAX_SIZE];

assert(cypherCost("qwerty4asdfgh", output) == 5);
assert(strcmp(output, "qwertyXsdfgh") == 0);
assert(cypherCost("qwle2r3t1ly12ul3io", output) == 16);
assert(strcmp(output, "qwXeXrXtXyXuXi") == 0);
assert(cypherCost("asdlA2S3D123654", output) == 29);
assert(strcmp(output, "sdXXSXDX") == 0);
}

```

3)

```

#include <cstring>
#include <cassert>

bool consecutiveSymbols(char *input, char *&start, char *&end)
{
int size = strlen(input), occurrences = 1, max = 1, begining = 1;
for (int i = 1; i < size; i++)
{
if (input[i] == input[i - 1])
{
occurrences++;
continue;
}
if (occurrences > max)
{
max = occurrences;
start = input + begining;
end = input + i;
}
occurrences = 1;
begining = i;
}
if (occurrences > max)
{
max = occurrences;

```

```
start = input + begining;
end = input + size;
}
return max != 1;
}

int main()
{
char *start, *end;
char input[] = "pbfdaaabcssll";
assert(consecutiveSymbols(input, start, end) == true);
assert(start == input + 4);
assert(end == input + 7);

char input1[] = "asdf";
assert(consecutiveSymbols(input1, start, end) == false);

char input2[] = "asdffpp";
assert(consecutiveSymbols(input2, start, end) == true);
assert(start == input2 + 3);
assert(end == input2 + 5);

char input3[] = "asdffppp";
assert(consecutiveSymbols(input3, start, end) == true);
assert(start == input3 + 5);
assert(end == input3 + 8);
}
```