```
format PE console
entry start

include 'include\win32a.inc'

; Создает координату вектора по двум точкам. op3 сохраняет результат
;op3 = op2 - op1
macro make_vector op1, op2, op3
{
	push	eax

	mov		eax, [op2]
	sub		eax, [op1]
	mov		[op3], eax

	pop		eax
}

;res = vec[ind] - vec[ind + 1];
; Создает вектор из двух точек прямой из массива
macro make_vector2 vec, ind, res
{
	push	ecx
	push	eax

	mov	ecx, [ind]
	mov		eax, [vec + 4 * ecx]
	inc		ecx
	sub		eax, [vec + 4 * ecx]
	mov		[res], eax

	pop		eax
	pop		ecx
}

; Перемножает два числа и возвращает результат в res
;res = a * b
macro multiply_nums a, b, res
{
	push	eax
	push	edx

	mov		eax, [a]
	cdq
	imul	eax, [b]
	mov		[res], eax

	pop	edx
	pop	eax
}

; Использует псевдосколярное произведение для получения знака поворота
macro GetTurnSign ax1, ax2, ay1, ay2, bx1, by1
{
```

```
        ;Вектор AB
        ;x1 = ax2 - ax1;
        ;y1_ = ay2 - ay1;
        make_vector ax1, ax2, x1
        make_vector ay1, ay2, y1

        ;Вектор AC
        ;x2 = bx1 - ax1;
        ;y2 = by1 - ay1;
        make_vector ax1, bx1, x2
        make_vector    ay1, by1, y2

        ;return (x1 * y2 - x2 * y1);
        multiply_nums x1, y2, tmp1
        multiply_nums x2, y1, tmp2
        mov         eax, [tmp1]
        sub         eax, [tmp2]
}

macro IsIntersect ax1, ax2, ay1, ay2, bx1, bx2, by1, by2
{
        ;Получаем знак поворота AB, AC
        GetTurnSign ax1, ax2, ay1, ay2, bx1, by1
        mov         [sign1], eax
        ;Получаем знак поворота AB, AC
        GetTurnSign ax1, ax2, ay1, ay2, bx2, by2
        mov         [sign2], eax
        ;Проверяем равенство знака
        multiply_nums   sign1, sign2, tmp1
        mov         eax, [tmp1]
        cmp         eax, 0
        jl          .secondStep

        jmp         .endIntersect
.secondStep:
        ;Получаем знак поворота CD, CA
        GetTurnSign bx1, bx2, by1, by2, ax1, ay1
        mov         [sign1], eax
        ;Получаем знак поворота CD, CB
        GetTurnSign bx1, bx2, by1, by2, ax2, ay2
        mov         [sign2], eax
        ;Проверяем равенство знака
        multiply_nums   sign1, sign2, tmp1
        mov         eax, [tmp1]
        cmp         eax, 0
        jl          .thirdStep

        jmp         .endIntersect

.thirdStep:
    pusha
    invoke  printf, outputres, [ax1], [ay1], [ax2], [ay2], [bx1], [by1], [bx2], [by2]
    popa
    inc         [count]
```

```
.endIntersect:
}

section '.idata' import data readable writeable
      library kernel32, 'kernel32.dll',\
                  msvcrt, 'msvcrt.dll'

      import kernel32,\
            GetStdHandle, 'GetStdHandle',\
            WriteConsole, 'WriteConsoleA',\
            ReadConsole, 'ReadConsoleA',\
            ExitProcess, 'ExitProcess'

      import msvcrt,\
            printf, 'printf',\
            scanf,  'scanf',\
            getch,  '_getch'

section '.data' data readable writeable
            inputText1 db 'Input five lines in format: x1 y1 x2 y2', 0dh, 0ah, 0
            inputText2 db 'Example: We have line at point (1, 0) and point (0, 1)', 0dh, 0ah, 0
            inputText3 db 'As a result, we will input: 1 0 0 1', 0dh, 0ah, 0
            input   db     '%d'
            outputres db 'Line1: ({%d, %d}, {%d, %d}); Line2: ({%d, %d}, {%d, %d})', 0dh, 0ah, 0
            notResult db 'No perpendicular intersections found', 0dh, 0ah, 0
            endprog db     'Program is end', 0
            px          dd     10     dup(0) ; Массив координат X у точек. Создаются прямые из 5 пар
точек i и i + 1
            py          dd     10     dup(0) ; Массив координат Y у точек
            x1          dd     ?
            x2          dd     ?
            y1          dd     ?
            y2          dd     ?
            tmp1    dd     ?
            tmp2    dd     ?
            sign1   dd     ?
            sign2   dd     ?
            index   dd     ?
            jndex   dd     ?
            count   dd     0

section '.code' code readable executable
start:

.input: ; Объявление цикла ввода данных
            invoke  printf, inputText1
            invoke  printf, inputText2
            invoke  printf, inputText3
            xor          ecx, ecx
            mov           eax, dword 1
            mov          [index], ecx
.input_loop: ; Тело цикла ввода данных
            invoke  scanf, input, x1
            invoke  scanf, input, y1
```

```
        mov         ecx, [index] ;Записываем текущий индекс в регистр
        mov         eax, [x1]
        mov         [px + 4 * ecx], eax ;Записываем значение в список из X
        mov         eax, [y1]
        mov         [py + 4 * ecx], eax ;Записываем значение в список из Y

        inc         [index]

        cmp         [index], 10
        jne         .input_loop
.end_input_loop:

; Поиск всех перпендикулярных пар прямых

.loop1: ;for (int i = 0; i < 9; i += 2)
        xor         eax, eax
        mov         [index], eax
.loop1_start:
        .loop2: ;for (int j = i + 2; j < 9; j += 2)
                mov         eax, [index]
                add         eax, 2
                mov         [jndex], eax
        .loop2_start:
                ;Проверка является ли векторы двух прямых перпендикулярными

                ;x1 = x[i + 1] - x[i];
                ;y1_ = y[i + 1] - y[i];
                make_vector2 px, index, x1
                make_vector2 py, index, y1

                ;x2 = x[j + 1] - x[j];
                ;y2 = y[j + 1] - y[j];
                make_vector2 px, jndex, x2
                make_vector2 py, jndex, y2

                ;x1 * x2
                ;y1 * y2
                multiply_nums x1, x2, tmp1
                multiply_nums y1, y2, tmp2

                ;invoke printf, outind, [jndex]
                ;if x1 * x2 + y1 * y2 == 0
                mov         eax, [tmp1]
                add         eax, [tmp2]
                cmp         eax, 0
                jne         .it_not_perp

                ; Если вектора перпендикулярны, то
                mov         ecx, [index]
                mov         edx, [jndex]
                ;IsIntersect(x[i], x[i + 1], y[i], y[i + 1], x[j], x[j + 1], y[j], y[j + 1]))
                IsIntersect (px + 4 * ecx), (px + 4 * (ecx + 1)),\
                                (py + 4 * ecx), (py + 4 * (ecx + 1)),\
```

```
                                        (px + 4 * edx), (px + 4 * (edx + 1)),\
                                        (py + 4 * edx), (py + 4 * (edx + 1))

        .it_not_perp: ; Если вектора не перпендикулярны, прыгаем сюда

                        inc         [jndex]
                        inc         [jndex]

                        cmp          [jndex], 8
                        jle         .loop2_start
        .loop2_end:

        inc          [index]
        inc          [index]

        cmp           [index], 6
        jle          .loop1_start
.loop1_end:

    ;if(count == 0)
    mov          eax, [count]
    cmp          eax, 0
    jne         exit

    invoke  printf, notResult

exit:
        invoke  printf, endprog
        invoke  getch
        invoke  ExitProcess, 0
```