Рассматриваемое приложение было создано на практических занятиях по программированию на языке C++. Оно может рассчитать площадь треугольника, используя параметры его сторон, а также проверяет введённые значения на корректность.

Попробуем определить, каким параметрам должны соответствовать эти значения.

```
push    esi
push    offset aRus     ; "rus"
push    0               ; int
call    ds:setlocale
add     esp, 8
push    4E3h            ; wCodePageID
call    ds:SetConsoleCP
push    4E3h            ; wCodePageID
call    ds:SetConsoleOutputCP
mov     esi, ds:printf
xorps   xmm0, xmm0
push    offset Format   ; "\tВведите  стороны треугольника : "
movsd   [esp+44h+var_18], xmm0
movsd   [esp+44h+var_20], xmm0
movsd   [esp+44h+var_10], xmm0
call    esi ; printf
lea     eax, [esp+44h+var_10]
push    eax
lea     eax, [esp+48h+var_20]
push    eax
lea     eax, [esp+4Ch+var_18]
push    eax
push    offset aLfLfLf  ; "%lf%lf%lf"
call    ds:scanf
movsd   xmm1, [esp+54h+var_18]
xorps   xmm0, xmm0
add     esp, 14h
comisd  xmm0, xmm1
jnb     loc_401155
```

Здесь можем увидеть приглашение к вводу в текстовом виде.

После ввода первое введённое число из оперативной памяти помещается в регистр XMM1

Затем обнуляется значение в регистре XMM0

Команда comisd позволяет сравнивать скалярные величины с плавающей запятой.
В нашем случае это число 0 (XMM0) и первая введённая цифра (XMM1).

Если XMM0 (то есть число 0) не меньше первого введённого числа (т.е. первое число <=0), то следует переход на метку loc_401155. Что происходит по ней – рассмотрим далее.

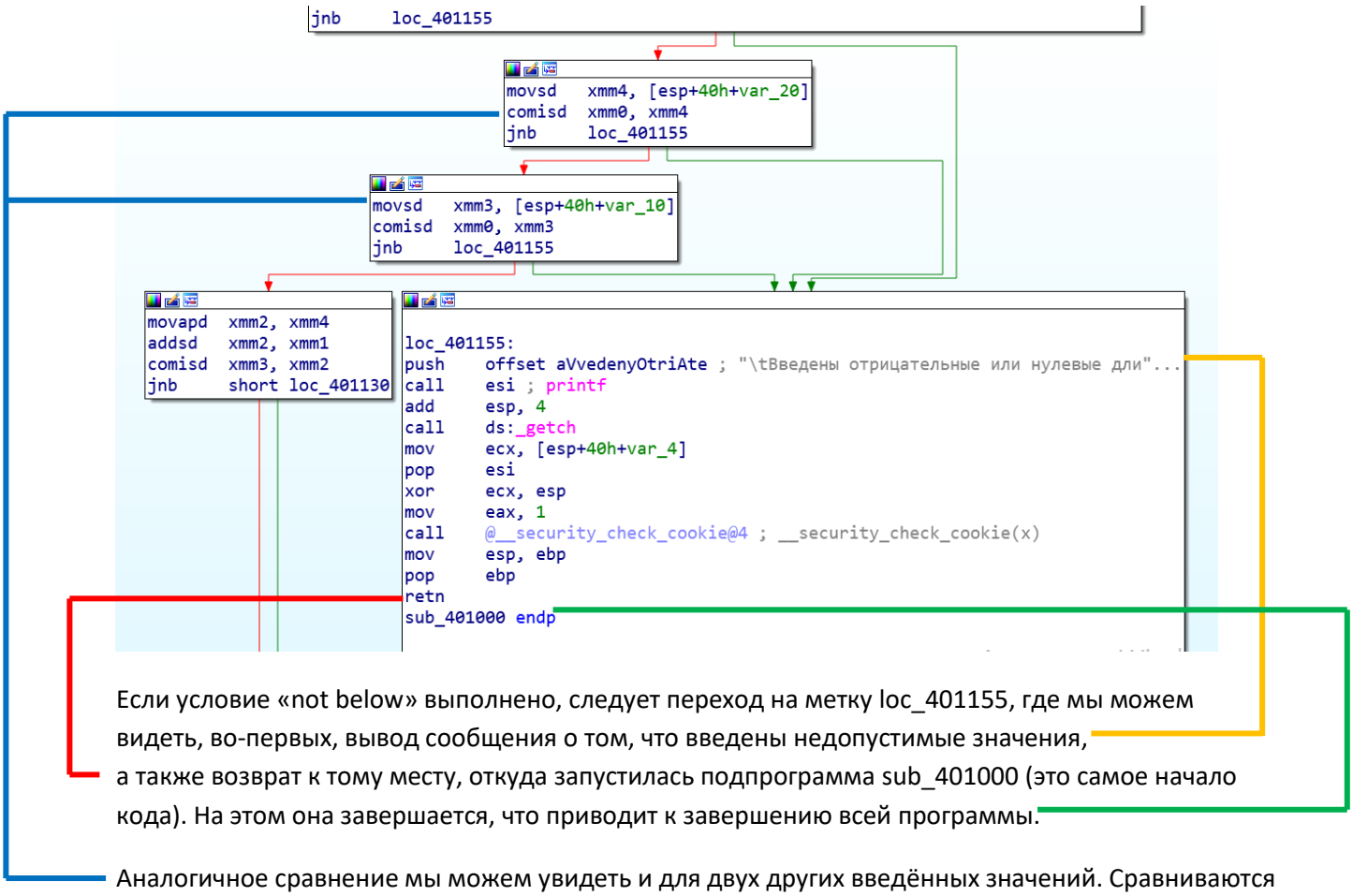
Также заодно обратим внимание на название подпрограммы в самом начале программы:

```
_text segment para public 'CODE' use32
assume cs:_text
;org 401000h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing


; Attributes: bp-based frame fuzzy-sp

sub_401000 proc near

var_48= qword ptr -48h
```



Если условие «not below» выполнено, следует переход на метку loc_401155, где мы можем видеть, во-первых, вывод сообщения о том, что введены недопустимые значения, а также возврат к тому месту, откуда запустилась подпрограмма sub_401000 (это самое начало кода). На этом она завершается, что приводит к завершению всей программы.

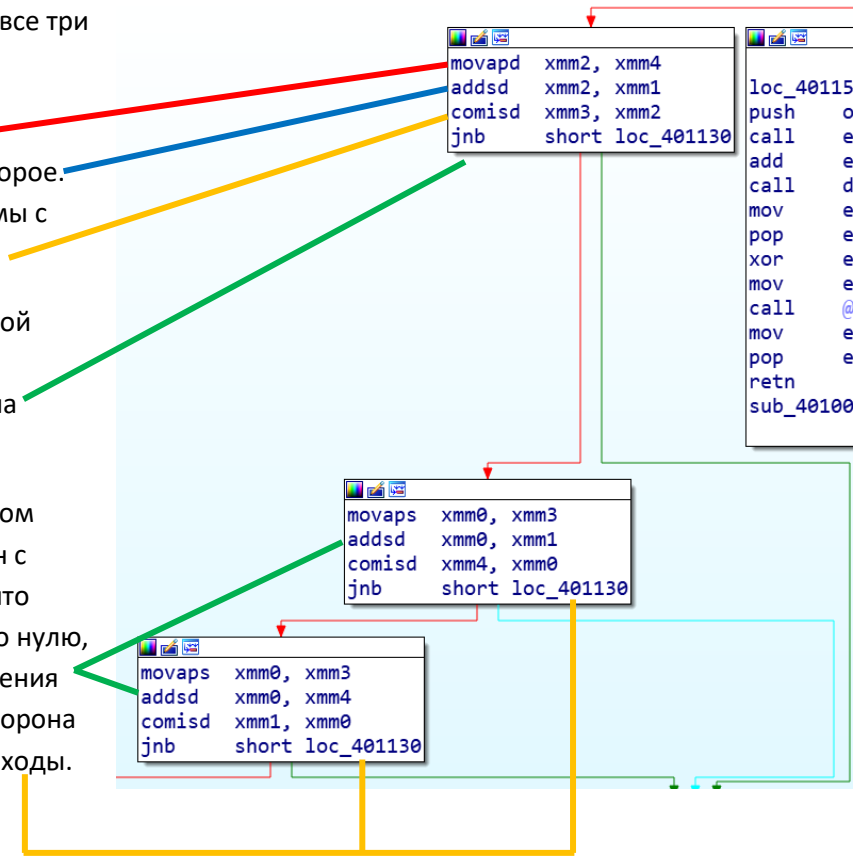
Аналогичное сравнение мы можем увидеть и для двух других введённых значений. Сравниваются они также с нулём, который по прежнему хранится в XMM0 и если нуль больше этих значений, опять же выводится сообщение и программа прекращает работу.

Посмотрим, что происходит, когда все три элемента прошли такую проверку.

В регистр XMM2 помещается одно значение и к нему прибавляется второе. Затем следует сравнение этой суммы с третьим значением.

Если третье значение не меньше этой суммы (что для треугольника невозможно), то следует переход на метку loc_401130.

Нетрудно увидеть, что таким образом сравниваются другие суммы сторон с одной стороной. Важно заметить, что здесь значение XMM0 уже не равно нулю, этот регистр используется для хранения сравниваемой суммы. Если одна сторона >= такой суммы, тоже следует переходы.



```
loc_401130:
push    offset aTreugolNikNeSu ; "\tТреугольник не существует"
call    esi ; printf
add     esp, 4
call    ds:_getch
mov     eax, 2
pop     esi
mov     ecx, [esp+3Ch+var_4]
xor     ecx, esp
call    @__security_check_cookie@4 ; __security_check_cookie(x)
mov     esp, ebp
pop     ebp
retn
```

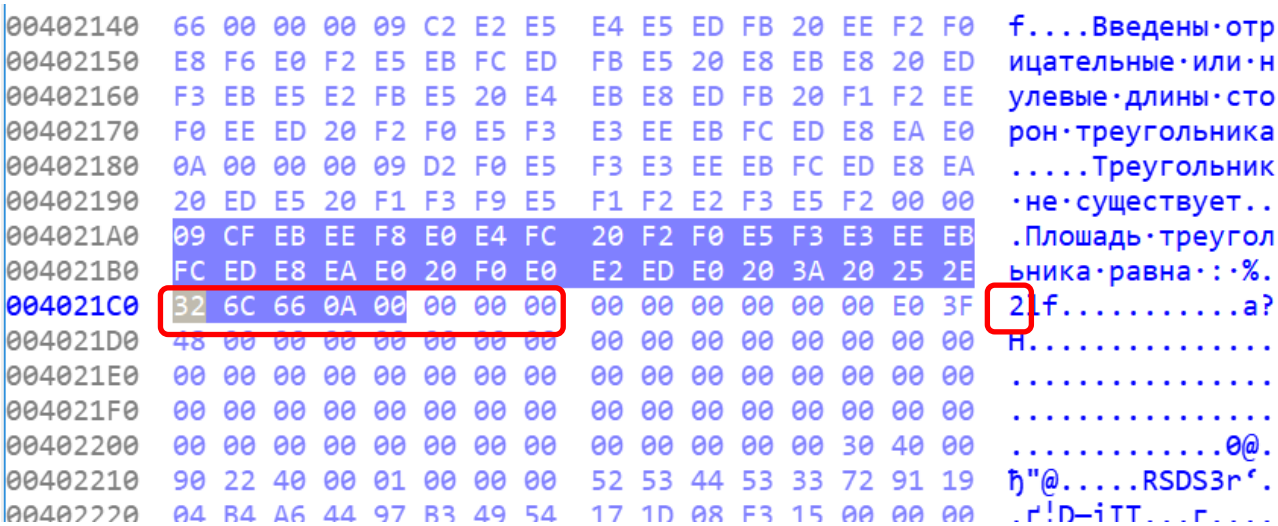
Все эти три сравнения при условии, что одна из сторон больше двух оставшихся, ведут к метке, которая содержит вывод сообщения о невозможности существования треугольника и возврату.

Проанализируем поведение программы, когда введённые значения корректны.

```
addsd   xmm2, xmm3
mulsd   xmm2, ds:qword_4021C8
movapd  xmm0, xmm2
subsd   xmm0, xmm1
movapd  xmm1, xmm2
subsd   xmm1, xmm4
mulsd   xmm0, xmm2
subsd   xmm2, xmm3
mulsd   xmm0, xmm1
mulsd   xmm0, xmm2
call    _libm_sse2_sqrt_precise
sub     esp, 8
movsd   [esp+48h+var_48], xmm0
push    offset aPloAdTreugolNi ; "\tПлощадь треугольника равна : %.2lf\n"
call    esi ; printf
add     esp, 0Ch
call    ds:_getch
xor     eax, eax
pop     esi
mov     ecx, [esp+3Ch+var_4]
xor     ecx, esp
call    @__security_check_cookie@4 ; __security_check_cookie(x)
mov     esp, ebp
pop     ebp
retn
```

Значение в регистре XMM2 не менялось, поэтому там по прежнему остаётся сумма чисел из регистров XMM4 и XMM1. К регистру XMM2 также прибавляется значение регистра XMM3, то есть теперь XMM2 содержит сумму всех введённых цифр.

Далее эта сумма (назовём её «периметр») умножается на число, которое находится в памяти по адресу DS:4021C8h; на него выделено 8 байт (qword). Посмотрим, что это за число.

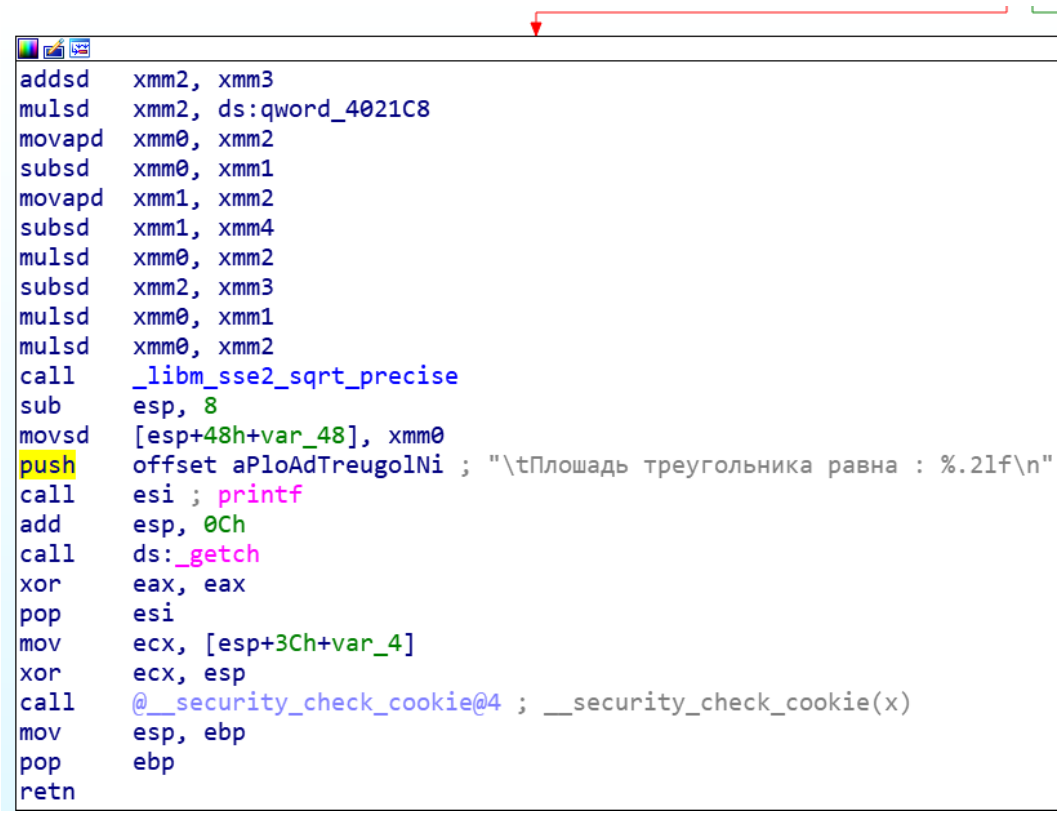


Очевидно, это число 2, которое компилятор разместил в памяти таким образом, выделив на него 8 байт. Итак, теперь в XMM2 у нас находится «периметр» * 2.

```
addsd   xmm2, xmm3
mulsd   xmm2, ds:qword_4021C8
movapd  xmm0, xmm2
subsd   xmm0, xmm1
movapd  xmm1, xmm2
subsd   xmm1, xmm4
mulsd   xmm0, xmm2
subsd   xmm2, xmm3
mulsd   xmm0, xmm1
mulsd   xmm0, xmm2
call    _libm_sse2_sqrt_precise
sub     esp, 8
movsd   [esp+48h+var_48], xmm0
push    offset aPloAdTreugolNi ; "\tПлощадь треугольника равна : %.2lf\n"
call    esi ; printf
add     esp, 0Ch
call    ds:_getch
xor     eax, eax
pop     esi
mov     ecx, [esp+3Ch+var_4]
xor     ecx, esp
call    @__security_check_cookie@4 ; __security_check_cookie(x)
mov     esp, ebp
pop     ebp
retn
```

Это же значение перемещается и в XMM0.

Затем из него вычитается XMM1 (одно введённое число).
В регистр, где это число содержалось, кладётся «периметр».
Из него вычитается второе введённое число.
Далее происходит ряд операция, заключающихся в умножении числа, равного периметру без значения одного числа
на число, равное периметру без значения второго числа
на число, равное периметру без значения третьего числа
на периметр.



```
addsd   xmm2, xmm3
mulsd   xmm2, ds:qword_4021C8
movapd  xmm0, xmm2
subsd   xmm0, xmm1
movapd  xmm1, xmm2
subsd   xmm1, xmm4
mulsd   xmm0, xmm2
subsd   xmm2, xmm3
mulsd   xmm0, xmm1
mulsd   xmm0, xmm2
call    _libm_sse2_sqrt_precise
sub     esp, 8
movsd   [esp+48h+var_48], xmm0
push    offset aPloAdTreugolNi ; "\tПлощадь треугольника равна : %.2lf\n"
call    esi ; printf
add     esp, 0Ch
call    ds:_getch
xor     eax, eax
pop     esi
mov     ecx, [esp+3Ch+var_4]
xor     ecx, esp
call    @__security_check_cookie@4 ; __security_check_cookie(x)
mov     esp, ebp
pop     ebp
retn
```

Далее вызывается имеющаяся в наборе инструкций процессора SSE подпрограмма (функция) _libm_sse2_sqrt_precise, осуществляющая вычисление корня.

После этого выводится соответствующее сообщение, вычисленное значение площади треугольника и программа завершает свою работу.