Trojúhelníky #2

Termín odevzdání:	**03.11.2019 23:59:59**	1324688.768 sec
Pozdní odevzdání s penalizací:	**06.01.2020 23:59:59** (Penále za pozdní odevzdání: 100.0000 %)	
Hodnocení:	**0.0000**	
Max. hodnocení:	**5.0000** (bez bonusů)	
Odevzdaná řešení:	0 / 20 Volné pokusy + 10 Penalizované pokusy (-10 % penalizace za každé odevzdání)	
Nápovědy:	0 / 2 Volné nápovědy + 2 Penalizované nápovědy (-10 % penalizace za každou nápovědu)	

Úkolem je vytvořit program, který bude porovnávat dvojice trojúhelníků. Tato úloha je rozšířením jednodušší varianty zadání. Trojúhelníky v této úloze mohou být zadávané pomocí délek stran (SSS), dvojice stran a jimi sevřeného úhlu (SUS) a dvojice úhlů přilehlých k zadané straně (USU). Doporučujeme nejprve vyřešit jednodušší zadání a následně odladěné řešení rozšířit.

V rovině jsou zadané 2 trojúhelníky, každý trojúhelník je jednoznačně zadán jedním ze tří způsobů:

- SSS - délkami svých stran,
- SUS - délkami dvojice stran a velikostí úhlu jimi sevřeného (úhel je zadán ve stupních, pořadí na vstupu je `strana úhel strana`),
- USU - velikostí dvojice úhlů a délkou přilehlé strany (úhly jsou zadané ve stupních, pořadí na vstupu je `úhel strana úhel`).

Program tato čísla přečte ze svého vstupu a rozhodne se pro jednu z následujících variant:

- zda zadané vstupy tvoří trojúhelník,
- zda jsou zadané trojúhelníky shodné,
- zda jsou zadané trojúhelníky podobné, ale nejsou shodné, nebo
- zda jsou zadané trojúhelníky zcela odlišné.

Pokud je vstup neplatný, program to musí detekovat a zobrazit chybové hlášení. Chybové hlášení zobrazujte na standardní výstup (ne na chybový výstup). Za chybu považujte:

- na vstupu není žádný ze tří povolených způsobů zadání trojúhelníku (SSS/SUS/USU),
- nečíselné zadání délek stran nebo velikosti úhlů,
- délka strany je záporná nebo nulová,
- úhel je menší nebo roven 0 stupňů, nebo větší nebo roven 180 stupňů,
- chybějící vstupní údaj/údaje (strana/úhel).

Ukázka práce programu:

```
Trojuhelnik #1:
SSS 4 6.5 7
Trojuhelnik #2:
SSS 7 6.5 4
Trojuhelniky jsou shodne.
```

```
Trojuhelnik #1:
SSS 7 7 7
Trojuhelnik #2:
USU 60 13 60
Trojuhelniky nejsou shodne, ale jsou podobne.
```

```
Trojuhelnik #1:
  SSS
   4.5
 6       7
Trojuhelnik #2:
SUS 7 30 12
Trojuhelniky nejsou shodne ani podobne.
```

```
Trojuhelnik #1:
SSS 9.861 9.865 9.883
Trojuhelnik #2:
SSS 9861 9883 9865
Trojuhelniky nejsou shodne, ale jsou podobne.
```

Trojuhelnik #1:
USU 60 11 60
Trojuhelnik #2:
SUS 13 60 13
Trojuhelniky nejsou shodne, ale jsou podobne.

Trojuhelnik #1:
SUS 10 180 20
Nespravny vstup.

Trojuhelnik #1:
USU 120 20 90
Vstup netvori trojuhelnik.

Trojuhelnik #1:
S SS 20 30 30
Nespravny vstup.

Trojuhelnik #1:
SSS 20 30 50
Vstup netvori trojuhelnik.

Trojuhelnik #1:
USU -4 10 12
Nespravny vstup.

Trojuhelnik #1:
SUS 1 2 abcd
Nespravny vstup.

Poznámky:

- Ukázkové běhy zachycují očekávané výpisy Vašeho programu (tučné písmo) a vstupy zadané uživatelem (základní písmo). Zvýraznění tučným písmem je použité pouze zde na stránce zadání, aby byl výpis lépe čitelný. Váš program má za úkol pouze zobrazit text bez zvýrazňování (bez HTML markupu).
- Znak odřádkování (\n) je i za poslední řádkou výstupu (i za případným chybovým hlášením).
- Pro reprezentaci hodnot použijte desetinná čísla typu `double`. Nepoužívejte typ `float`, jeho přesnost nemusí být dostatečná.
- Úlohu lze vyřešit bez použití funkcí. Pokud ale správně použijete funkce, bude program přehlednější a bude se snáze ladit.
- Číselné vstupní hodnoty jsou zadávané tak, aby se vešly do rozsahu datového typu `double`. Referenční řešení si vystačí s číselnými typy `double` a `int`.
- Pro načítání vstupu se hodí funkce `scanf`.
- Úhly jsou zadávané ve stupních. Pokud budete používat goniometrické funkce ze standardní knihovny, musíte úhly převést na radiány. V matematické knihovně je dostupná konstanta `M_PI`, která aproximuje číslo π s dostatečnou přesností..
- Způsob zadání trojúhelníku (SSS/SUS/USU) lze přečíst jako řetězec nebo jako tři znaky. Práce s řetězci není v C jednoduchá, proto doporučujeme postup z trojicí znaků. Pokud se rozhodnete pro postup se znaky, podívejte se na popis formátovacího řetězce funkce `scanf` a najděte si rozdíl mezi konverzí `" %c"` a `"%c"` (s mezerou a bez mezery před vlastní konverzí). Pokud se rozhodnete pro postup s řetězcem, podívejte se na funkci `strcmp` a na postupy, kterými lze správně ošetřit nečekaně dlouhý řetězec na vstupu.
- Při programování si dejte pozor na přesnou podobu výpisů. Výstup Vašeho programu kontroluje stroj, který požaduje přesnou shodu výstupů Vašeho programu s výstupy referenčními. Za chybu je považováno, pokud se výpis liší. I chybějící nebo přebývající mezera/odřádkování je považováno za chybu. Abyste tyto problémy rychle vyloučili, použijte přiložený archiv se sadou vstupních a očekávaných výstupních dat. Podívejte se na videotutoriál (materiály -> cvičebnice -> video tutoriály), jak testovací data použít a jak testování zautomatizovat.

- Váš program bude spouštěn v omezeném testovacím prostředí. Je omezen dobou běhu (limit je vidět v logu referenčního řešení) a dále je omezena i velikost dostupné paměti (ale tato úloha by ani s jedním omezením neměla mít problém). Testovací prostředí dále zakazuje používat některé "nebezpečné funkce" -- funkce pro spouštění programu, pro práci se sítí, ... Pokud jsou tyto funkce použité, program se nespustí. Možná ve svém programu používáte volání:

```
int main ( int argc, char * argv [] )
{

  ...

  system ( "pause" ); /* aby se nezavrelo okno programu */
  return 0;
}
```

Toto nebude v testovacím prostředí fungovat - je zakázáno spouštění jiného programu. (I pokud by se program spustil, byl by odmítnut. Nebyl by totiž nikdo, kdo by pauzu "odmáčkl", program by čekal věčně a překročil by tak maximální dobu běhu.) Pokud tedy chcete zachovat pauzu pro testování na Vašem počítači a zároveň chcete mít jistotu, že program poběží správně, použijte následující trik:

```
int main ( int argc, char * argv [] )
{

  ...

#ifndef __PROGTEST__
  system ( "pause" ); /* toto progtest "nevidi" */
#endif /* __PROGTEST__ */
  return 0;
}
```

- Slovní popis struktury platných vstupních dat není zcela exaktní. Proto připojujeme i formální popis vstupního jazyka v EBNF:

```
input      ::= { whiteSpace } triang { whiteSpace } triang { whiteSpace }
whiteSpace ::= ' ' | '\t' | '\n' | '\r'
triang     ::= type { whiteSpace } decimal { whiteSpace } decimal { whiteSpace } decimal
type       ::= 'SSS' | 'SUS' | 'USU'
decimal    ::= [ '+' | '-' ] integer [ '.' integer [ ( 'e' | 'E' ) [ '+' | '-' ] integer ] ]  |
               [ '+' | '-'  ] '.' integer [ ( 'e' | 'E' ) [ '+' | '-' ] integer ]
integer    ::= digit { digit }
digit      ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

Vzorová data:		**Download**
Odevzdat:	Choose File	**Odevzdat**