Finanční analýza

Termín odevzdání:	**03.12.2017 23:59:59**
Hodnocení:	**3.0000**
Max. hodnocení:	**3.0000** (bez bonusů)
Odevzdaná řešení:	1 / 20 Volné pokusy + 10 Penalizované pokusy (-10 % penalizace za každé odevzdání)
Nápovědy:	1 / 2 Volné nápovědy + 2 Penalizované nápovědy (-10 % penalizace za každou nápovědu)

Úkolem je vytvořit program, který bude filtrovat finanční data, aby "lépe vypadala".

Předpokládáme, že máme informace o stavu účtu v čase. Z těchto dat je potřeba vybrat vhodný úsek do prezentace. V prezentaci chceme dokumentovat dobré hospodaření s prostředky v dlouhodobém horizontu. Tedy chceme vybrat co nejdelší spojitý úsek dat (to je primární kritérium), ale zároveň požadujeme, aby stav účtu na konci vybraného úseku byl vyšší nebo roven stavu účtu na začátku vybraného úseku dat. Fakticky nás tedy zajímá pouze hodnota na začátku a na konci vybraného úseku dat, hodnoty uvnitř vybraného úseku dat nejsou důležité, při výběru nehrají žádnou roli.

Pokud je na vstupu posloupnost hodnot např.: `115 111 57 56 103 96 -1000 100 83 45`, bude výsledkem úsek od hodnoty `57` do hodnoty `83` (je nejdelší možný a 83 ≥ 57).

Vstupem programu je posloupnost celých čísel, která udává stav účtu v čase. Délka posloupnosti je omezena nejvýše na 250000 hodnot. Zadávání hodnot končí v okamžiku, kdy je detekován EOF (konec vstupu).

Výstupem programu je nalezený úsek maximální délky takový, že stav účtu na konci úseku je vyšší nebo rovný stavu účtu na začátku úseku. Program zobrazí délku úseku a indexy začátku a konce (indexy počítané od 0). Stejně dlouhých úseků může existovat více než jeden, pak program v libovolném pořadí zobrazí všechny možnosti. Může se stát, že vstupní posloupnost neobsahuje žádný vhodný úsek, v tomto případě je zobrazeno speciální hlášení (viz ukázka).

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:

- vstup není celé číslo,
- na vstupu nejsou alespoň dvě hodnoty,
- vstup je delší než 250000 hodnot.

Ukázka práce programu:

```
Hodnoty:
115 111 57 56 103 96 -1000 100 83 45
7: 2 - 8
Moznosti: 1
```

```
Hodnoty:
5 9 11 8
   4 16 -8


3 1 3
6: 0 - 5
Moznosti: 1
```

```
Hodnoty:
23 112 5 11 8 4 16 -8 3 0 1 3
5: 7 - 11
5: 2 - 6
Moznosti: 2
```

```
Hodnoty:
9 7 3 1 -8 -1000
Nelze najit.
```

```
Hodnoty:
1 2 3 4 hello
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 postačuje datový typ `int`.
- Vstupní hodnoty je potřeba ukládat do pole, výsledek nelze určit on-line. Vstupní posloupnost má délku nejvýše 250000, postačuje tedy staticky alokované pole s touto velikostí. Nealokujte pole dynamicky, zbytečně si tím úlohu zkomplikujete.
- Úloha nabízí bonus, kterého lze dosáhnout, pokud je řešení časově efektivní. Časové efektivity lze dosáhnout zejména tím, že zvolíte vhodný algoritmus zpracování a rozumně efektivně jej implementujete:
  - Test rychlosti 1 lze zvládnout naivním (kvadratickým) algoritmem, pokud zvolíte vhodnou implementaci. Pokud Vaše implementace tímto testem neprojde, nedosáhnete plných 100% bodů.
  - Test rychlosti 2 lze zvládnout pouze pokud je použitý algoritmus efektivnější než kvadratický. Pokud tento test zvládnete, získáte hodnocení vyšší než 100%.
- Pokud má úloha více řešení, musí je program zobrazit všechna. Jejich pořadí ve výpisu ale není podstatné, testovací prostředí si pořadí nalezených úseků před porovnáním uspořádá.
- Slovní popis struktury platných vstupních dat není zcela exaktní. Proto připojujeme i formální popis vstupního jazyka v EBNF:

```
input      ::= { whiteSpace } number { whiteSpace } number { whiteSpace } { number { whiteSpace } }
whiteSpace ::= ' ' | '\t' | '\n' | '\r'
number     ::= [ '+' | '-' ] digit { digit }
digit      ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```