

# Zkouška z úvodu do programování

## Úloha č. 5 (Operace s řetězci)

### 1. Zadání

Nalezení samohlásek v zadaném textu a jejich zvýraznění umístěním do závorek.

Cílem této úlohy bylo ve vstupním textovém souboru vyhledat samohlásky (bez ohledu na velká a malá písmena), uzavřít je do závorek a takto upravený text uložit do nově vytvořeného textového souboru. Rozšířeným zadáním této úlohy bylo rozeznat, jestli je více samohlásek po sobě bezprostředně následujících a v takovém případě je neuzavírat do samostatných závorek ale pouze do jedné společné.

### 2. Popis a rozbor problému, vzorce

Program načte vstupní soubor s textem, kde je potřeba uzávorkovat samohlásky. Nejprve je potřeba rozlišit samohlásky od jiných znaků, k čemuž slouží metoda *“isVowel”*, která kontroluje načtený znak hledá v zadaném seznamu samohlásek a pokud ho tam najde, vrátí hodnotu *“True”*. Při základní verzi programu by při vrácení *“True”* uzávorkovala daný znak a zapsala ho do výstupního souboru a při vrácení *“False”* zapsala znak do výstupního souboru v nezměněné podobě. Tato verze by ale neřešila uzávorkování více samohlásek jdoucích po sobě, proto je řešení složitější. Řešení tohoto problému je blíže popsáno v bodě č. 4: *Problematické situace a jejich rozbor*.

### 3. Popisy algoritmů formálním jazykem

- 1: Opakování pro  $\forall$  znak v textu ( $ch_i, ch_{i+1}$ ):
  - 2:     Rozeznání, jestli je  $ch_i$  samohláska pomocí metody *“isVowel”*.
  - 3:     if  $ch_i \in$  seznamu samohlásek *“vowels”* a předchozí znak v textu nebyla samohláska,  
          vypíše se *“(ch<sub>i</sub>”*  
          elif  $ch_i \in$  seznamu samohlásek *“vowels”* a předchozí znak v textu byla samohláska,  
          vypíše se *“ch<sub>i</sub>”*  
          elif  $ch_i \notin$  seznamu samohlásek *“vowels”* a předchozí znak v textu byla samohláska,  
          vypíše se *“)ch<sub>i</sub>”*  
          else:  
              vypíše se *“ch<sub>i</sub>”*
  - 4: if poslední znak *“z”*  $\in$  *“vowels”*,  
      vypíše *“)”*

### 4. Problematické situace a jejich rozbor

Nejobtížnějším problémem při tomto úkolu bylo zjistit, jestli se v textu nachází dvě nebo více samohlásek za sebou, které by se měly uzávorkovat společně. Tento problém byl vyřešen pomocí *for cyklu*. Před samotným *for cyklem* byla zvolena proměnná *“previous\_vowel”*, jejíž hodnota se měnila na *“True”*, pokud poslední načtený znak byla samohláska, jinak nabývala hodnoty *“False”*.

Poté byly *for cyklem* procházeny jednotlivé znaky, u kterých bylo pomocí metody třídy *“Character”* *“isVowel”* rozhodnuto, jestli je daný znak samohláska a poté zjištěno pomocí proměnné

*"previous\_vowel"*, jestli předchozí znak byla také samohláska. Podle kombinace těchto dvou aspektů byla zvolena jedna z čtyř možných situací, které mohly nastat a podle ní byly k danému znaku umístěny závorky. Dále pokud byla posledním znakem samohláska, musela být přidána ještě jedna závorka na konec.

## 5. Vstupní data

Text, jenž obsahuje samohlásky k uzávorkování se nachází v souboru pojmenovaném *"5\_vstupni\_text.txt"*, který je načítán pomocí funkce *"open()"* a jeho obsah je uložen do proměnné *"text"*.

## 6. Výstupní data

Výstupní data, tedy text se samohláskami uzavřenými do závorek, jsou uloženy do textového souboru *"5\_vystupni\_text.txt"*.

## 7. Dokumentace

Program nahraje vstupní data v textovém souboru pojmenovaném *"5\_vstupni\_text.txt"*, který obsahuje text určený ke zpracování. Pokud soubor, který se uživatel snaží otevřít neexistuje nebo je prázdný, program na to uživatele upozorní a skončí.

Program po spuštění vybere samohlásky, které zvýrazní jejich umístěním do závorek a takto upravený text uloží do souboru s názvem *"5\_vystupni\_text.txt"*.

## 8. Závěr, náměty na vylepšení, neřešené problémy

Program byl vyzkoušen na několika vstupních souborech, na kterých řešil problém úspěšně. Za vylepšení programu mohu považovat uzavírání po sobě jdoucích samohlásek do společných závorek.

## 9. Seznam literatury

Při řešení tohoto problému nebyla použita žádná doplňující literatura, pouze znalost z přednášek a cvičení.

## Úloha č. 67 (Počítačová geometrie)

### 1. Zadání

Test, zda bod leží uvnitř konvexního mnohoúhelníku.

Cílem této úlohy bylo načíst z textového souboru souřadnice bodů, které tvoří vrcholy konvexního mnohoúhelníku a poté z téhož souboru načíst zadaný bod a vybranou metodou zjistit jeho polohu vůči mnohoúhelníku – jestli leží vně, uvnitř nebo na jeho hraně a také graficky znázornit danou situaci pomocí vhodného grafického rozhraní.

### 2. Popis a rozbor problému, vzorce

K řešení problému byl využit algoritmus založený na half plane testu, který je popsán v následujícím bodě č. 3: *Popisy algoritmů formálním jazykem*.

Pro řešení úlohy byly využity tři třídy “Point”, “Line” a “Polygon”.

Každý objekt třídy “Point” je charakterizován souřadnicemi  $x$  a  $y$ . Objektem třídy “Point” je například analyzovaný bod. Třída “Point” obsahuje metodu “getx()”, která ze zadaného objektu třídy “Point” vrátí souřadnici  $x$  a metodu “gety()”, která vrátí souřadnici  $y$ .

Objekty třídy “Line” představují přímku, která vznikne prodloužením hrany mnohoúhelníku. Jsou tedy charakterizovány dvěma body, tvořícími hranu.

Třída “Line” obsahuje metodu “location()”, která pro každou hranu mnohoúhelníku vypočítá souřadnice vektorů “ $v$ ” a “ $u$ ”. Vektor “ $u$ ” je dán dvěma vrcholy mnohoúhelníku, které tvoří jednu hranu. Vektor “ $v$ ” je dán počátečním bodem vektoru “ $u$ ” a analyzovaným bodem, jehož polohu chceme zjistit. Pomocí výpočtu determinantu matice, kterou tvoří oba vektory, zjistí orientaci bodu vůči hraně – může být tedy záporná, kladná nebo nulová.

Poslední třída “Polygon” je charakterizována vrcholy, které jsou načteny ze vstupního souboru a musí být minimálně tři. Ve třídě “Polygon” je také vytvořen seznam hran.

Třída “Polygon” také obsahuje metodu “location\_of\_point()”, která prochází hrany mnohoúhelníku a použije metodu třídy “Line” “location()” a pro každou hranu polygonu určí orientaci bodu vzhledem k hraně. Pokud je orientace bodu u všech stran stejná (u všech kladná nebo u všech záporná), znamená to, že bod leží uvnitř konvexního mnohoúhelníku. Pokud je orientace rovná nule, znamená to, že bod leží na hraně mnohoúhelníku.

### 3. Popisy algoritmů formálním jazykem

Algoritmus užitý v programu je založený na half plane testu.

Metoda half plane test je založená na tom, že hledáme, ve které polorovině zadaný bod leží. Polorovina “ $P$ ” je dána dvěma vektory – “ $v$ ” a “ $u$ ”. Vektor “ $u$ ” je dán dvěma vrcholy mnohoúhelníku, které tvoří jednu hranu. Vektor “ $v$ ” je dán počátečním bodem vektoru “ $u$ ” a analyzovaným bodem “ $Q$ ”, jehož polohu chceme zjistit. Pomocí výpočtu determinantu matice danou oběma vektory, zjistíme orientaci bodu vzhledem k hraně. Pokud vyjde orientace u všech hran větší než 0 nebo u všech hran menší než 0, znamená to, že bod je součástí konvexního mnohoúhelníku.

*Test polohy bodu  $Q$  vzhledem k polorovině  $P$ :*

1: Opakování pro  $\forall$  stranu  $(h_i, h_{i+1})$ :

- 2: Určení orientace  $o_i$  bodu  $Q$  ke straně  $(h_i, h_{i+1})$ .
- 3: if  $Q \in (h_i, h_{i+1})$ , bod  $Q \in \partial P$ .
- 4: if  $o_i \neq o_{i-1}$ , bod  $Q \notin P$ .
- 5: Bod  $Q \in P$

*Určení orientace  $o_i$  bodu  $Q$  od hrany  $h_i$*

- 1: Opakování pro  $\forall$  stranu  $(h_i, h_{i+1})$ :
- 2: Vektor  $u = (\text{souřadnice } x_B \text{ koncového bodu hrany} - \text{souřadnice } x_A \text{ počátečního bodu hrany}, \text{souřadnice } y_B \text{ koncového bodu hrany} - \text{souřadnice } y_A \text{ počátečního bodu hrany}) = (u_x, u_y)$
- 3: Vektor  $v = (\text{souřadnice } x_Q \text{ bodu } Q - \text{souřadnice } x_A \text{ počátečního bodu hrany}, \text{souřadnice } y_Q \text{ bodu } Q - \text{souřadnice } y_A \text{ počátečního bodu hrany}) = (v_x, v_y)$
- 4: výpočet orientace  $o_i = u_x * v_y - u_y * v_x$
- 5: if  $o_i > 0$ , bod  $Q \in$  levé polorovině  
elif  $o_i == 0$ , bod  $Q \in h_i$   
else: bod  $Q \in$  pravé polorovině

#### 4. Problematické situace a jejich rozbor

Nejvíce problematické pro mě bylo odlišit od ostatních dvou situací bod ležící na hraně mnohoúhelníku. Bylo nutné omezit přímkou danou dvěma body pouze na úsečku.

#### 5. Vstupní data

Vstupní data se načítají z textového souboru "67\_vrcholy\_a\_bod.txt", který obsahuje na každém novém řádku souřadnice vrcholu mnohoúhelníku (souřadnice  $x$  a  $y$  je oddělena znakem ","). Vrcholy jsou seřazeny tak, jak se nachází popořadě po obvodu mnohoúhelníku. Bod na posledním řádku není vrchol mnohoúhelníku, ale jedná se o bod, jehož polohu vůči mnohoúhelníku chceme zjistit.

#### 6. Výstupní data

Do terminálu je vypsáno, jestli bod leží uvnitř, vně nebo na hraně mnohoúhelníku a v novém okně je situace znázorněna graficky pomocí modulu "turtle" a je tedy vykreslen mnohoúhelník a bod, který je analyzován.

#### 7. Dokumentace

Program nahraje data ze vstupního souboru pojmenovaného "67\_vrcholy\_a\_bod.txt", který musí obsahovat na každém řádku jeden vrchol mnohoúhelníku – tedy jeho souřadnice oddělené znakem ",". Tyto vrcholy musí být seřazeny popořadě, tak jak jsou seřazeny po obvodu mnohoúhelníku. Na posledním řádku jsou souřadnice bodu, jehož polohu vůči mnohoúhelníku chceme zjistit (též oddělené znakem ",").

Pokud je načten neexistující nebo prázdný soubor, program zahlásí odpovídající chybovou hlášku a skončí. Program také rozpozná, pokud je načteno méně než 3 vrcholy, protože mnohoúhelník musí být tvořen minimálně třemi body. Program nerozpozná, jestli zadané body tvoří konvexní nebo nekonvexní mnohoúhelník.

Po spuštění program zjistí polohu zadaného bodu vůči konvexnímu mnohoúhelníku a výsledek vypíše do terminálu.

Mnohoúhelník, poloha bodu i těžiště je vykreslena pomocí modulu "turtle" v novém okně. Těžiště je od analyzovaného bodu odlišeno červenou barvou. Okno s grafickým znázorněním se zavře pomocí kliknutí myši.

#### 8. Závěr, náměty na vylepšení, neřešené problémy

Při testování programu na různých datech řešil zadání správně. Věcí, která by se dala vylepšit je určitě zpřehlednění programu a také přidání výjimky, která by ověřila, zdali zadané body náhodou netvoří nekonvexní mnohoúhelník.

Také si myslím, že by bylo možné elegantněji vyřešit grafické znázornění například přidáním nějaké metody.

#### 9. Seznam literatury

BAYER, Tomáš. *Úvod do výpočetní geometrie. Základní vztahy* [online].

Dostupné z: <https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk2.pdf>

BAYER, Tomáš. *Point Location Problem* [online].

Dostupné z: [http://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk3\\_new.pdf](http://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk3_new.pdf)