Eva Lemberková
MOBIBO, 2. ročník

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 prochází znaky textu a aktuálně načtený znak hledá v zadaném seznamu samohlásek a pokud ho tam najde, vrací hodnotu "*True*". Při základní verzi programu by při vracení "*True*" uzávorkovala daný znak a zapsala ho do výstupního souboru a při vracení "*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

Při řešení tohoto problému nebyl použit žádný specifický algoritmus.

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 za 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.

Eva Lemberková
MOBIBO, 2. ročník

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 popsaný 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. Objekty třídy "*Point*" jsou například analyzovaný bod nebo těžiště. Třída "*Point*" obsahuje metodu "*getx()*", která ze zadaného objektu třídy "*Point*" vrací souřadnici *x* a metodu "*gety()*", která vrací souřadnici *y*.

Objekty třídy "*Line*" představují přímku, která vznikne prodloužením hrany mnohoúhelníku. Jsou tedy charakterizovány dvěma body, tvořícími hranu. Ze souřadnic těchto bodů jsou spočítány koeficienty *a, b* a *c,* které nám pomohou danou přímku vyjádřit obecnou rovnicí.

Třída "*Line*" obsahuje metodu "*poloha_bodu_k_hrane()*", která do obecné rovnice přímky tvořené hranou dosadí nejprve souřadnice *x* a *y* těžiště a poté analyzovaného bodu. Výsledek porovnává s nulou a podle toho následně rozliší, jestli bod leží ve stejné polorovině jako těžiště (tedy uvnitř mnohoúhelníku) nebo ne.

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 a vypočítáno těžiště.

Těžiště je vypočítáno pomocí metody "*najdi_teziste()*", která je blíže popsána v bodě 3.

Třída "*Polygon*" také obsahuje metodu "*poloha_bodu()*", která použije metodu třídy "*Line*" "*poloha_bodu_k_hrane()*" pro každou hranu polygonu a podle kombinace zjištěných výsledků určí polohu analyzovaného bodu vůči 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 je dána přímkou a bodem. Pokud tedy chceme zjistit, jestli bod leží uvnitř mnohoúhelníku, můžeme mnohoúhelník považovat za průnik všech polorovin daných hranou mnohoúhelníku a bodem, který leží vždy uvnitř mnohoúhelníku. Takovým bodem je u konvexního mnohoúhelníku těžiště.

Pro výpočet souřadnic těžiště byla použita metoda třídy "*Polygon*" "*najdi_teziste()*", která načetla souřadnice vrcholů mnohoúhelníku a vypočítala z nich souřadnice těžiště pomocí vzorce:

Souřadnice *x* těžiště = (součet souřadnic *x* vrcholů mnohoúhelníku)/počet hran mnohoúhelníku
Souřadnice *y* těžiště = (součet souřadnic *y* vrcholů mnohoúhelníku)/počet hran mnohoúhelníku

Poté byly inicializovány hrany mnohoúhelníku pomocí obecné rovnice přímky, jejíž koeficienty *a, b* a *c* se vypočítaly pomocí dosazení souřadnic bodů, jimiž byla hrana definovaná, do vzorce pro obecnou rovnici přímky:

$$ax + by + c = 0$$

Dále byla využita funkce "*poloha_bodu_k_hrane()*", která si ukládá hodnotu obecné rovnice poloroviny po dosazení těžiště a poté analyzovaného bodu. Obecná rovnice poloroviny připomíná rovnici přímky, pouze je konečné rovná se nahrazeno znaménkem nerovnosti.

Do rovnice byl tedy nejprve za *x* a *y* dosazena souřadnice těžiště a poté souřadnice analyzovaného bodu. Pokud obě nerovnosti byly obě menší než 0 nebo obě větší než 0, znamenalo to, že oba body leží ve stejné polorovině. Pokud nerovnosti platily u všech polorovin daných hranami mnohoúhelníku, pak analyzovaný bod ležel uvnitř daného mnohoúhelníku. Pokud se jedna rovnice poloroviny odvozená od analyzovaného bodu rovnala nule a alespoň jedna rovnice odpovídala tomu, že analyzovaný bod leží ve stejné polorovině jako těžiště, pak analyzovaný bod ležel na hraně mnohoúhelníku. V ostatních případech ležel bod vně mnohoúhelníku.

4. Problematické situace a jejich rozbor

Nejvíce problematické pro mě bylo odlišit od ostatních dvou situací bod ležící na hraně mnohoúhelníku. Bylo nutné si uvědomit, že jedna rovnice hrany musí být rovna nule, ale minimálně jedna rovnice musí odpovídat tomu, že bod leží ve stejné polorovině jako těžiště.

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, jeho těžiště (červeně) a kde se nachází analyzovaný bod (černě).

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 myší.

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, na který by program nefungoval, protože těžiště nekonvexního mnohoúhelníku může ležet i vně samotného útvaru.

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