Dokumentace k úlohám na zkoušku z Úvodu do programování (ZS 2023/2024)
Vypracoval: Bc. Jakub Kraft

Úloha 74 – Převod čísla na římské číslice a zpět (4)

1. Zadání úlohy

Vstupní hodnotou je celé číslo v desítkové soustavě zadané uživatelem. Nalezněte jeho reprezentaci římskými číslicemi tak, aby respektovala pravidla použitá pro zápis v této nepoziční soustavě. Zápis římskými číslicemi následně převeďte zpět do desítkové soustavy.

2. Rozbor problému

Účelem programu je převod čísel desítkové soustavy na římské číslice a zpět. Podstata problému v tomto převodu spočívá v tom, že římské číslice mají zcela odlišnou strukturu než desítková soustava. Zatímco desítková soustava je poziční a využívá deset základních číslic (0-9) (Wikipedie 2022), římské číslice jsou založeny na kombinaci písmen (I, V, X, L, C, D, M) pro reprezentaci různých hodnot, jedná se tak o nepoziční číselnou soustavu (Wikipedie 2023). V římských číslicích se používají některá speciální pravidla pro zápis čísel, příkladem může být “IV“ pro 4 místo “IIII“. To poměrně značně komplikuje sestavování správného algoritmu.

3. Struktura a popis programu

Byl zvolen algoritmus, který pro převod využívá dvě slovníkové struktury. Pro převod z desítkové soustavy na římské číslice program postupně odečítá největší možné hodnoty a přidává odpovídající římské znaky do výstupního řetězce. Pro opačný převod program čte římské znaky nebo jejich kombinace (páry) a přičítá jejich numerické hodnoty. Výsledkem je identické číslo desítkové soustavy, které uživatel zadal.

Třída **`RomanConverter`** obsahuje metody pro převod čísel z desítkové soustavy na římské číslice.

```
class RomanConverter:
    def __init__(self):
        self.roman_to_num = {
            'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M':
1000,
            'IV': 4, 'IX': 9, 'XL': 40, 'XC': 90, 'CD': 400, 'CM': 900
        }
        self.num_to_roman = {
            1: 'I', 4: 'IV', 5: 'V', 9: 'IX', 10: 'X', 40: 'XL', 50: 'L',
            90: 'XC', 100: 'C', 400: 'CD', 500: 'D', 900: 'CM', 1000: 'M'
        }
```

Prvním krokem bylo vytvoření implicitního inicializátoru **`def __init__(self`**) a následných dvou datových položek instance. V obou případech se jedná o slovníkové struktury. **`self.roman_to_num`** je slovník vytvořený pro převod z římských číslic na čísla desítkové soustavy. Obsahuje jak jednotlivé znaky (například “I“, “X“), tak i speciální kombinace dvou znaků (například “IV“, “IX“). Druhý slovník **`self.num_to_roman`** naopak dále v programu slouží k převodu z čísel desítkové soustavy na římské číslice a obsahuje stejné hodnoty, akorát jejich pozice jsou obrácené (**`'I': 1`** a **`1: 'I'`**).

```
def to_roman(self, num):
    roman = ''
    for value in sorted(self.num_to_roman.keys(), reverse=True):
        while num >= value:
            roman += self.num_to_roman[value]
            num -= value
    return roman
```

Metoda **`def to_roman(self, num)`** obecně převádí čísla z desítkové soustavy na římské číslice.

`for value in sorted(self.num_to_roman.keys(), reverse=True)`

Tento řádek iteruje a třídí klíče v sestupném pořadí. Sestupné pořadí je nutné, protože v římských číslicích se obvykle nejdříve zapisují vyšší hodnoty. **`while`** cyklus pokračuje, dokud je číslo **`num`**, které má být převedeno na římskou číslici, větší nebo rovno aktuální hodnotě **`value`**. V cyklu se přidává do řetězce **`roman`** ekvivalent římské číslice aktuální hodnoty **`value`**. Po přidání římské číslice se v dalším řádku odečte **`value`** od **`num`**. Výsledek cyklu by mělo být kompletní a správně napsané římské číslo odpovídající zadané hodnotě.

```
def from_roman(self, roman):
    num = 0
    i = 0
    while i < len(roman):
        if i+1 < len(roman) and roman[i:i+2] in self.roman_to_num:
            num += self.roman_to_num[roman[i:i+2]]
            i += 2
        else:
            num += self.roman_to_num[roman[i]]
            i += 1
    return num
```

Druhá metoda **`def from_roman(self, roman)`** naopak konvertované římské číslice převede zpět do desítkové soustavy. Hodnota se akumuluje v proměnné **`num`** zatímco proměnná **`i`** funguje jako počáteční index pro průchod řetězcem římského čísla. Cyklus **`while`** zde běží dokud index **`i`** nedosáhne délky řetězce.

`if i+1 < len(roman) and roman[i:i+2] in self.roman_to_num`

Tato podmínka kontroluje, zda existuje platný pár znaků (například “IV“, “IX“) začínající na aktuálním indexu. Nejprve je potřeba se ujistit, že index dalšího znaku (**`i+1`**) je uvnitř hranic řetězce. Když je první část podmínky splněna, tak program kontroluje, zda řetězec **`roman[i:i+2]`** tvoří pár ve slovníku **`roman_to_num`**. Pokud je podmínka splněna, tak se do proměnné **`num`** přidává číselná hodnota dvouznakové římské číslice. Tím se zvýší index **`i`** o 2, aby se přeskočily právě zpracované dva znaky. Pokud podmínka splněna nebyla, znamená to, že aktuální a následující znak netvoří platný pár. V tomto případě se provede blok stejným způsobem, akorát program pracuje s jednotlivou římskou číslicí.

```
def main():
    converter = RomanConverter()
    try:
        user_input = int(input("Zadejte celé číslo v desítkové soustavě:
"))
        roman_numeral = converter.to_roman(user_input)
        converted_back = converter.from_roman(roman_numeral)

        print(f"Zde je jeho reprezentace římskými číslicemi:
{roman_numeral}")
```

```
        print(f"Číslo zpět převedeno do desítkové soustavy:
{converted_back}")
    except ValueError:
        print("Nezadali jste celé číslo v desítkové soustavě.")

main()
```

Tato část kódu definuje hlavní funkci **`main`**, která slouží jako vstupní bod pro interakci s uživatelem. Vytvoří se objekt třídy **`RomanConverter`**, který má přístup k metodám **`to_roman`** a **`from_roman`**. Je zde použit blok **`try-except`** pro zachycení situací, kdy uživatel zadá neplatný vstup. V takovém případě se vyvolá **`ValueError`** a program vypíše chybovou zprávu. **`user_input`** tedy po uživateli požaduje, aby zadal celé číslo v desítkové soustavě.

`roman_numeral = converter.to_roman(user_input)`

Tato část kódu volá metodu **`to_roman`** v objektu **`converter`** a předává jí číslo zadané uživatelem.

`converted_back = converter.from_roman(roman_numeral)`

Pomocí tohoto řádku je předané římské číslo znovu převedeno do desítkové soustavy. Pak už následují výpisy výsledků.

4. Popis vstupních a výstupních dat

Vstupními daty jsou celá čísla v desítkové soustavě zadané uživatelem. Výstupní hodnoty jsou dvě. První je převedené číslo do zápisu římskými číslicemi. Tento zápis je následně znovu převeden do desítkové soustavy, jedná se tak o identické číslo, které bylo zadané uživatelem.

Příklad vstupních a výstupních dat z programu:

Zadejte celé číslo v desítkové soustavě: 2024

Zde je jeho reprezentace římskými číslicemi: MMXXIV

Číslo zpět převedeno do desítkové soustavy: 2024

5. Problematická místa a možná vylepšení

Největší obtíží v převodu mezi číselnými soustavami bylo vyřešit problematiku párů znaků římských číslic, které ve společné kombinaci nereprezentují hodnotu, kterou by měly jednotlivě (“IV“, “IX“ apod.). Snahou bylo najít co nejlepší možné řešení, ale je nutné si přiznat, že může existovat řešení efektivnější.

Slabou stránkou zápisu římskými číslicemi je operace s velkými čísly, protože největší hodnota reprezentované písmenem “M“ je ekvivalentem hodnoty 1000 v desítkové soustavě. Takže při zadání relativně většího čísla desítkové soustavy není pak výsledek zapsaný římskými číslicemi přehledný. To však není chyba programu, ale nevýhoda vyjádření hodnoty římskými číslicemi.

Možným vylepšením je vytvoření grafického uživatelského rozhraní, které by program učinilo přístupnějším a uživatelsky přívětivějším. K tomu by se mohla použít knihovna TkInter (Python Wiki 2024).

6. Zdroje

WIKIPEDIE (2022): Desítková soustava. Dostupné z: https://cs.wikipedia.org/wiki/Des%C3%ADtkov%C3%A1_soustava [cit. 24. 1. 2024].

WIKIPEDIE (2023): Římské číslice. Dostupné z: https://cs.wikipedia.org/wiki/%C5%98%C3%ADmsk%C3%A9_%C4%8D%C3%ADslice [cit. 24. 1. 2024].

PYTHON WIKI (2024): GUI Programming in Python. Dostupné z: https://wiki.python.org/moin/GuiProgramming [cit. 25. 1. 2024].