Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Katedra softwarových technologií

Objektově orientované programování - IOOP

Zadání semestrální práce

Lexikální analyzátor

Autor zadání:
Karel Šimerda

Garant předmětu:
Karel Šimerda

Verze 1.0
20.9.2017

1 Úvod

1.1 Cíl semestrální práce

Cílem semestrální práce je osvěžení znalostí získané v předmětech IZAPR a IPALP z předchozího semestru. Dále bude cílem zjistit do jaké míry studenti mají představu o objektově orientovaného programování. To vše bude ověřeno na malém projektu, který studenti vypracují do značné míry samostatně. Až budou probrány složitější partie programovacího jazyka JAVA, bude požadováno tento projekt přepracovat v duchu nových poznatků .

1.2 Účel dokumentu

Tento dokument je určen pro vyučující a studenty, jako zadání první semestrální práce na cvičení předmětu „Objektově orientované programování“ (IOOP).
Dokument je členěn do několika části:
Nejdříve dokument obsahuje vizi, která je neformální popis požadované aplikace. Vize slouží k přiblížené problematiky lexikální analýzy. Vizi lze zpracovat pomocí techniky analýzy podstatných jmen a sloves, z kterou byli studenti seznámeni v prvních ročníku, k nalezení tříd a jejich odpovědností vyvíjené aplikace.
Za vizí následují požadavky. Ty jsou rozděleny do dvou základních kategorií a to do funkčních a nefunkčních požadavků. Funkční požadavky popisují budoucí hlavní vlastnosti aplikace. Nefunkční požadavky předepisují co se může použít, jak musí ošetřit chyby a jaké jsou výkonnostní omezení.
Teprve splněním všech požadavků, funkčních a nefunkčních, může být semestrální práce přijata. Do te doby bude studentů práce vrácená k dopracování.

2 Vize

Program „**Lexikální analyzátor**“ bude umožňovat zpracování textu ze souboru a zobrazení výsledků v podobě seznamu tokenů na monitoru.

Proč takové zadání?

Pro zpracování textu a převod do dat v počítači byla v minulém semestru využívána třída Scanner. Její instance umožňovala bohaté zpracování textu ze vstupního zařízení. Její nevýhodou bylo, že se muselo předpokládat jaký druh informace v textu bude. Když se očekávalo celé číslo musela se použít metoda `nextInt()` a když reálné číslo tak `nextDouble()` nebo `nextfloat()`. Před tím se mohlo zeptat, zda takový formát dat je v textu připraven ke zpracování. K tomu sloužily metody `hasNextInt()`, `hasNextDouble()`, `hasNextFloat()` a tak podobně. To je někdy nepraktické a v případě jednoduché rozboru vstupního textu je výhodnější si naprogramovat vlastní řešení. K takovému řešení se obvykle říká lexikální analyzátor (anglicky scanner).

Lexikální analýza, kterou provádí lexikální analyzátor, rozděluje posloupnost znaků na lexikální elementy (například identifikátory, čísla, klíčová slova, operátory, a pod.). Tímto způsobem překladače zpracovávají naše zdrojové kódy do tzv. tokenů a ty se dál poskytují k dalšímu zpracování v syntaktickém analyzátoru. S tímto se někteří studenti setkají na navazujícím magisterském studiu v předmětu kompilátory nebo v teorii jazyků.

My samozřejmě nebudeme realizovat kompilátor, ale pokusíme se o naprogramování jednoduché lexikální analýzy v podobě malého lexikálního analyzátoru.

Náš program rozebere vstupní text na výstupní tokeny v podobě několika klíčových slov, čísel v desítkové a hexadecimální soustavě a na zbylé texty, které bude považovat za identifikátory. Za oddělovače budou použity bílé znaky, jako jsou mezera, tabelátor, konec řádku a dále znaky rovnítka,

čárky a středníku. Mezera, tabelátor a znaky konce řádku budou oddělovat jednotlivé lexikální elementy. Čárka, rovnítko a středník budou plnit funkci oddělovačů a zároveň budou tokeny. Výstupem programu bude výpis charakteristik všech tokenů.

Příklad rozboru

Vstupní text:

```
begin

a,b=100;

c=0x10;

end;
```

Výpis tokenů

```
KeyToken{klicoveSlovo=KeyWord{key=begin}}
SeparatorToken{Separator{bílý znak}}
IdentifierToken{a}
SeparatorToken{Separator{čárka}}
IdentifierToken{b}
SeparatorToken{Separator{rovná se}}
NumberToken{value=100}
SeparatorToken{Separator{středník}}
SeparatorToken{Separator{bílý znak}}
IdentifierToken{c}
SeparatorToken{Separator{rovná se}}
NumberToken{value=16}
SeparatorToken{Separator{středník}}
SeparatorToken{Separator{bílý znak}}
KeyToken{klicoveSlovo=KeyWord{key=end}}
SeparatorToken{Separator{středník}}
SeparatorToken{Separator{bílý znak}}
```

3 Požadavky

3.1 Funkční požadavky

FR1 Zobrazovaní seznamu: Program bude vypisovat výsledek lexikální analýzy vstupního textového souboru v podobě charakteristiky tokenů, tak je naznačeno v kapitole 2.Vize

FR2 Spouštěcí příkaz: Program se bude spouštět z příkazového řádku svým názvem,kdy jeho prvním parametrem bude jméno souboru, případně s cestou k němu.

FR3 Volba klíčových slov Požaduje se, aby každý student si zvolil svoji sadu klíčových slov.

FR4 Číselné literály Program bude rozlišovat desítková a hexadecimální čísla. U hexadecimální čísel se požaduje prefix `0x`

FR5 Velikost písmen Program bude zpracovávat pouze malá písmena.

FR6 Opakování bílých znaků Požaduje, při výskytu více bílých znaků za sebou, aby byl vypsán pouze jeden token.

3.2 Nefunkční požadavky

NR1 Kolekce: Je dovolenou použít knihovní kolekce Javy, jako je například rozhraní `List` a třída `ArrayList`.

NR2 Ošetření chyb: Požaduje se, aby chyby ve vykonávání metod byly hlášeny pouze pomocí výjimek.

NR3 Výčtové hodnoty: Požaduje se, aby místo konstant byly důsledně používány výčty.

NR4 Jména výčtových hodnot: Výčty musí umožňovat uložení přirozeného českého jména hodnoty a to malými písmeny, které so potom použijí při výpisech charakteristik tokenů.

NR5 Omezení délek identifikátorů a klíčových slov: Požaduje se aby, délka identifikátorů a klíčových slov nepřesáhla délku 32 znaků.

NR6 Hledání klíčových slov: Na vyhledaní klíčových slov se požaduje využití výčtu, v kterém jsou klíčová slova definována.

NR7 Struktura tokenů: Požaduje se, aby tokeny byly uspořádány do hierarchického stromu dědičnosti.

NR8 Ověření: Pro ověření aplikace si každý student vytvoří alespoň jeden zkušební textový soubor, kterým bude moci přezkoušet plnění požadavků.