Akademia Podlaska
w Siedlcach

Siedlce, dn. 20.V.2006r.

Wydział Nauk Ścisłych

Instytut Informatyki

„Interpreter - EasyCompile”

Projekt z SISW

Wykonali:
Paweł Sawczuk
Edwin Tomczuk
Krzysztof Pietraszek
I Inf. Mgr uzup. (gr. 3)

Prowadzący ćwiczenia:
dr inż. Krzysztof Trojanowski

Spis treści:

1. Ogólny opis projektu

Celem niniejszego projektu grupowego z przedmiotu "*Sieci i systemy wirtualne*" było opracowanie interpretera dowolnego (własnego) języka. Projekt był realizowany w trzech etapach:

1. etap I – Skaner,
2. etap II – Parser,
3. etap III – Linker.

Po trzecim etapie otrzymano gotowy interpreter-kompilator poprawnie interpretujący składnię założonego języka (notacja BNF), wyświetlający wyniki plików z programami i informujący użytkownika o ewentualnych błędach. Cały projekt (wszystkie jego pliki) zostały zaimplementowane w języku JAVA.

2. Notacja BNF

Notacja „BNF"dla kompilatora ***EasyCompile***:

```
<program>::= program
                        <sekcja_deklaracji>
                        begin
                        <sekwencja_wyrazen>
                        end.

<sekcja_deklaracji>::= var {<zmienna>:<type>;}

< sekwencja_wyrazen>::= {<wyrazenie_if> | <przypisanie> | <petla_while> |
<wyrazenie_we> | <wyrażenie_wy>}

<wyrazenie_if>::= if(<wyrażenie_logiczne>)
                                <sekwencja_wyrazen>
                               [else
                               <sekwencja_wyrazen>]
                                endif;

<przypisanie>::= <zmienna>:=<liczba> |
<zmienna>:=<zmienna>|<zmienna>:=<zmienna><dzialanie><zmienna> |
<zmienna>:=<liczba><dzialanie<zmienna> | <zmienna>:=<zmienna><dzialanie><liczba>
;

<liczba>::= <calkowita> | <zmiennoprzecinkowa>
```

```
<calkowita>::= <cyfra>{<cyfra>}

<cyfra>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<zmiennoprzecinkowa>::= <cyfra>{<cyfra>}.<cyfra>{<cyfra>}

<type>::= int | float

<zmienna>::= <litera>{<litera | literaDuza> }

<litera>::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q |
r | s | t | u | v | w | x | y | z|

<znak_specjalny> ::= | |\

<literaDuza>::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
Q | R | S | T | U | V | W | X | Y | Z

<string>::= ‘{<litera> | <literaDuza> | <cyfra> | <znak_specjalny>}’

<operator>::= < | > | <= | >= | = | <>

<dzialanie>::= - | + | * | /

<wyrazenie_logiczne>::= <zmienna><operator><zmienna> |
<liczba><operator><zmienna> | <zmienna><operator><liczba>

<petla_while>::= while(<wyrazenie_logiczne>)
                            <sekwencja_wyrazen>
                            endwhile;

<wyrazenie_we>::= read(<zmienna>);

<wyrazenie_wy>::= write(<string> | <zmienna> );
```

3. Zasada działania i funkcjonalność

Zasada działania programu polega na uruchomieniu w wierszu poleceń pliku programu wraz z parametrem, którym jest plik z kodem do kompilacji i wykonania. Wygląda to tak:

`java interpreter` ***`<nazwa_pliku>`*** ,

gdzie *nazwa_pliku* to ścieżka do pliku z kodem programu, np.: c:\stud\program1.txt.
W wyniku wykonania programu zostaną zwrócone (wyświetlone) jego wyniki – komunikaty, liczby, wyniki obliczeń, itp. lub zostanie zwrócony komunikat błędu z numerem linii, w której jest ten błąd. Oprócz tego, gdy nie ma błędów zostaną wyświetlone komunikaty:

```
Proces skanowania zakończył się sukcesem
```

`Proces parsowania zakończył się pomyślnie`

Każdy poprawny kod programu w języku rozpoznawanym przez *EasyCompile* musi mieć następujące elementy, w następującej kolejności:

```
program
var [deklaracje zmiennych;]
begin
     [instrukcje;]
end.
```

Bez któregokolwiek z tych słów kluczowych lub innej kolejności tych słów wystąpi błąd. Jak wiadomo każdy kod programu, w jakimkolwiek języku posiada sekcję deklaracji zmiennych i sekcję instrukcji (tzw. ciało programu). We wspomnianym języku, zbliżonym do Pascala, przykładowa deklaracja zmiennych wygląda następująco:

```
var liczba_calk : int;
    liczba_rzecz: float;
    ...
    [kolejne zmienne]
```

Jak widać, nasz język rozpoznaje dwa typy liczbowe **`int`** i **`float`**. Każda instrukcja i deklaracja zmiennej musi być zakończona znakiem średnika (**`;`**). Każda zmienna (i jej typ) musi być oddzielnie deklarowana. Każda deklaracja zmiennej innego typu niż wymienione, lub przypisanie do zmiennej znaku lub ciągu znaków, nie będących liczbą całkowitą ani liczbą rzeczywistą kończy się błędem. Język rozpoznaje **liczby ujemne**.

Istniej jedna instrukcja wejścia `'read'` i jedna instrukcja wyjścia `'write'`. Przykład ich zastosowań:

`read(zmienna);`

`write('ciąg znaków');` lub `write(zmienna);`

jeśli chcemy po wyświetleniu ciągu znaków lub zmiennej przejść do nowej linii to wyświetlamy **`\n`** np.:

`write('ciąg znaków\nTo jest nowa linia');` lub samo `write('\n');`

Instrukcja przypisania pozwala na złożone działania i rozpoznaje kolejność działań arytmetycznych, np.:

```
j:=n+1/3-5*2;
```

Niestety nie jest dozwolone zastosowanie nawiasów i grupowanie podwyrażeń, np.:

```
j:=(n+1/3-5)-2*(n/j-1);   //błąd!!!
```

Konstrukcja instrukcji warunkowej jest następująca:

```
if(warunek)
     instrukcje;
[ else
     instrukcje; ]
endif;
```

Jak widać możliwe jest wykonanie instrukcji nie spełniających *warunku*. Każdy `'if'` musi mieć swój `'endif;'`.

Konstrukcja jedynej w tym języku pętli (*while*) jest następująca:

```
while(warunek)
     instrukcje;
endwhile;
```

Każde `'while'` musi mieć swój `'endwhile;'`.

Zarówno przy `'if'` jak i `'while'` jest sprawdzany wynik porównania: *zmienna – liczba* lub *liczba – zmienna* lub *zmienna – zmienna*. Gdy warunek jest prawdziwy zostają wykonane odpowiednie instrukcje, a gdy fałszywy wykona się sekcja `'else'` (w przypadku instrukcji `'if'`) lub nic się nie wykona (w przypadku instrukcji `'while'`).

Instrukcje `'if'` i `'while'` mogą być dowolnie (na przemian) i dowolną ilość razy zagnieżdżane, z zachowaniem **kolejności** ich otwarcia i zamknięcia!

4. Anomalie funkcjonalne - ograniczenia

Interpreter *EasyCompile* w wersji obecnej jest kompilatorem bardzo prostego języka, zbliżonego składnią do języka PASCAL. Spełnia on jednak postawioną mu minimalną funkcjonalność! Jeżeli zaś chodzi o porównanie do innych języków np.: wspomnianego PASCAL'a to możemy wyróżnić następujące ograniczenia-anomalie:

1. po słowie kluczowym `'program'` nie może wystąpić żadna nazwa, a tylko słowo

kluczowe `'var'` rozpoczynające sekcję deklaracji zmiennych,

2. każda zmienna musi być zadeklarowana oddzielnie, tzn. błędem jest np.:

```
var liczba_calk, suma : int;
```

3. istnieją tylko dwa typy `'int'` i `'float'`- nie ma typów boolowskich, znakowych ani innych liczbowych,
4. brak jest operatorów logicznych,
5. każda instrukcja warunkowa i pętla musi być zakończona odpowiednim słowem kluczowym, mimo że może posiadać tylko jedną inną instrukcję np.: przypisania,
6. warunek instrukcji `'if'` i `'while'` musi być prosty (bez operatorów logicznych), tzn. tylko dwa argumenty: *zmienna – liczba, liczba – zmienna, zmienna – zmienna*,
7. warunek instrukcji `'if'` i `'while'` musi być podany w nawiasach zwykłych,
8. istnieje tylko jedna instrukcja wejścia `'read'`, która wczytuje wartość podaną z klawiatury do zmiennej i przechodzi do następnej linii
9. istnieje tylko jedna instrukcja wyjścia `'write'`, która wypisuje ciąg znaków lub wartość liczbową
10. jedna instrukcja `'read'` wczytuje na raz jedną wartość do jednej zmiennej,
11. jedna instrukcja `'write'` wypisuje na raz **albo** ciąg znaków, **albo** wartość zmiennej,
12. w instrukcji przypisania nie mogą się pojawić nawiasy, tzn. nie ma możliwości budowy bardziej złożonych wyrażeń arytmetycznych,
13. nie można dodawać zmiennych i liczb ze znakiem minusa i odwrotnie, np: `j:=j+-5;`
14. interpteter nie rozróżnia dużych i małych liter – wszystkie traktowane jednakowo.

5. Wymagania, instalacja i pliki projektu

Aby można było skompilować i uruchomić projekt należy mieć zainstalowaną wirtualną maszynę Javy – środowisko *JRE* (ang. *Java Runtime Environment*), najlepiej w wersji 1.5.0 (w tej wersji był testowany) lub późniejszej. Przy wersjach wcześniejszych program też powinien działać bez zarzutu.

Dalej, należy dodać wpis do zmiennej środowiskowej PATH naszego systemu

operacyjnego, mówiący o lokalizacji środowiska Javy, np.:

C:\Program Files\Java\jdk1.5.0_05\bin,

aby można było uruchomić nasz program z dowolnej lokalizacji w systemie katalogów.

Pliki programu można pobrać ze strony http://easycompile.sourceforge.net/, zakładka **Interpreter-Binary** (pliki źródłowe *.*java*) lub **Interpreter-Source** (pliki wykonywalne*.*class*). Zarówno w jednym jak i drugim przypadku są to archiwa *.*zip*, które należy po ściągnięciu ze strony rozpakować.

Pliki potrzebne do **kompilacji** programu:

1. Scanner.java,
2. Parser.java,
3. interpreter.java.

Pliki potrzebne do **uruchomienia** programu:

4. Scanner.class,
5. Parser.class,
6. interpreter.class.

Dodatkowo, w archiwum ***Interpreter-bin.zip*** (zakładka **Interpreter-Binary**) zostały umieszczone przykładowe pliki z kodem programów w naszym języku.