AKADEMIA PODLASKA

WYDZIAŁ NAUK ŚCISŁYCH

INSTYTUT INFORMATYKI

Projekt zaliczeniowy przedmiotu Sieci i Systemy Wirtualne

„APInterPas”

Wykonali:
Łukasz Miłobędzki
Radek Radzikowski
Irena Markiewicz

SIEDLCE 2005

Spis treści:

1. Cel projektu

Celem projektu było zaprojektowanie i zaprogramowanie interpretera języka Pascal przy wykorzystaniu dowolnego języka programowania. Wykonany projekt sprawdza poprawność kodu i wyświetla wyniki lub ewentualne błędy w kodzie testowego programu.

Interpreter jest translatorem, który za każdym razem od nowa analizuje kod źródłowy programu i przekłada instrukcje programu na kod pośredni, w celu przetłumaczenia poleceń, które się w nim znajdują na formę zrozumiałą dla maszyny.

Zakres specyfikacji projektu:

- Deklarowanie zmiennych,
- Typy danych
  - Integer,
  - Real,
  - Boolean,
- Operacja przypisania: :=,
- Operacje działania matematycznego: +,-,/,*,
- Operacje porównania: <>,<,<=,>,>=,=,
- Operacje wejścia/wyjścia:
  read, readln, write, writeln,
- Początek/ koniec bloku: begin, end,
- Operatory arytmetyczne:

  - Dodawania: +,
  - Odejmowania: -,
  - Mnożenia: *,
  - Dzielenia: /,

- Operatory relacyjne
  - Mniejsze: <,
  - Większe: >,
  - Mniejsze równe: <=,
  - Większe równe: >=,
  - Różne: <>,
  - Równe: =,
- Operacja pętli:
  for <identyfikator><operator_przypisania><wartosc>to<wartosc>do
- Operacja warunku: if <warunek> then <instrukcje> oraz
  if <warunek> then <instrukcje> else <instrukcje>.

Projekt „APInterPas” został zaprojektowany w języku JAVA. Nasza aplikacja oferuje przyjazny interfejs graficzny, w tym odczyt oraz zapis pliku, również zawiera funkcje sprawdzenia składniowego oraz leksykalnego kodu.

2. **Notacja BNF (Bockusa-Naura)**

Notacja Backusa-Naura (ang. Backus Naur Form (BNF)) została napisana w kryteriach wymogów projektu w celu przedstawienia tekstowej definicji podzbioru języka Pascal. Przyjęto następujące symbole metajęzykowe (alfabetu pomocniczego):

<> - służące do nazywania jednostek składniowych definiowanego języka umieszcza się w nawiasach kątowych;

= - czytany “jest równe z definicji”, łączy strony produkcji. Po lewej tronie symbolu podaje się nazwę definiowanego pojęcia, po prawej - definicję;

[] – selekcja. Oznacza opcjonalne, czyli nieobowiązkowe wystąpienie;

{} – iteracja. Oznacza dowolną liczbę powtórzeń (także zerową) elementów definicji objętych nawiasami;

| - rozdzielanie alternatyw na liście. Oznacza w produkcjach spójnik logiczny “albo” (alternatywę) i pozwala na zmniejszenie ich liczby. Po obu stronach tego symbolu występują różne, ale równoprawne człony definicji, których jednocześnie można wybrać tylko jeden;

() – opcjonalność. Oznacza jeden z alternatywnych zapisów alternatywy albo alternatyw objętych tymi nawiasami;

. - oznacza koniec definicji;

... - komentarz;

@ - oznaczenie elementu identyfikującego;

- **Słowa kluczowe**

<Begin>::=“BEGIN”,
<End>::=”END”,
<Var>::=”VAR”,
<Integer>::=”INTEGER”,
<real>::=”REAL”,
<Boolean>::=”BOOLEAN”,
<For>::=”FOR”,
<to>::=”TO”,
<downto>::=”DOWNTO”,
<do>::=”DO”,
<if>::=”IF”,
<then>::=”THEN”,
<else>::=”ELSE”,
<write>::=”WRITE”,
<writeln>::=”WRITELN”,
<read>::=”READ”,
<readln>::=”READLN”.

- **Operatory, typy, zmienne**

<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|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,

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

<slowo_kluczowe> ::= begin|end|var|integer|real|for|to|do|if|then|Ele

<separator> ::= ;

<przecinek> ::= ,

<kropka> ::= .

<dwukropek> ::=:

<operator_przypisania> ::= :=

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

<opareatory_relacyjne> ::= = | <> | < | <= | > | >=

<nawias_lewy> ::= (

<nawias_prawy> ::=)

<cudzysłów>::=’

<tekst>:=<cudzysłów>{<litera>|<cyfra>|<znak_specjalny>}<cudzysłów>

<liczba_calkowita > ::= [-]<cyfra>{<cyfra>}

<liczba_rzeczywista>::=[-]<cyfra>{<cyfra>}[<kropka>{<liczba_calkowita>}]

<wartosc_boolean_> ::=true | false

<wartosc> ::= <liczba_calkowita>|<liczba_rzeczywista>|<wartosc_boolean>

< nazwa_zmiennej > ::= <litera>{<litera>}

- **Deklaracje i operacje**

<deklaracja_zmiennej> ::= <identyfikator><dwukropek><typ>< separator >

<deklaracja_zmiennych> ::=
<slowo_klucz_deklaracja_zmiennych><deklaracja_zmiennej> |
<deklaracja_zmiennych><deklaracja_zmiennej>

<deklaracja_stalej> ::= <identyfikator><operator_ini_stalej><wartosc>
<separator><identyfikator><dwukropek><typ><operator_ini_stalej><wartosc>
< separator>

<deklaracja_stalych> ::= <slowo_klucz_deklaracja_stalych><deklaracja_stalej>
| <deklaracja_stalych><deklaracja_stalej>

<operacja_arytmetyczna> ::= <wartosc><operator_arytmetyczny><wartosc>
<operacja_relacyjna>::=<wartosc><operator_porownania><wartosc>
<operacja_przypisania> ::= <identyfikator><operator_przypisania><wartosc> |
<identyfikator><operator_przypisania><operacja_binarna>

<operacja_wyjscia> ::=
<rodzaj_operacji_wyjscia><nawias_otwarty><wartosc><nawias_zamkniety><
separator >

<operacja_wejscia> ::=
<rodzaj_operacji_wejscia><nawias_otwarty><identyfikator><nawias_zamkniet
y>< separator >

<operacja_wejscia_wyjscia> ::= <operacja_ wejscia > | <operacja_ wyjscia >

<tresc_bloku> ::= <polecenie> | <tresc_bloku><polecenie>

<blok> ::= <Begin><tresc_bloku><End>

<polecenie> ::= <operacja> | <instrukcja_warunkowa> | <petla_for> |
< operacja_wejscia_wyjscia >

<instrucja_warunkowa> := <jezeli><operacja_relacji><wtedy><polecenie> |
<jezeli><operacja_relacji><wtedy><blok> |
<jezeli><operacja_relacji><wtedy><polecenie><albo><polecenie>
<jezeli><operacja_relacji><wtedy><blok><albo><polecenie> |
<jezeli><operacja_relacji><wtedy><polecenie><albo><blok> |
<jezeli><operacja_relacji><wtedy><blok><albo><blok>

<petla_for> ::=
<dla><identyfikator><operator_przypisania><wartosc><do><wartosc><wykon
uj><polecenie> |
<dla><identyfikator><operator_przypisania><wartosc><do><wartosc><wykon
uj><blok>

<instrukcja_zlozona>::=<begin>{<instrukcja>|<instrucja_warunkowa>}*<end>
<separator>.

3. Wymagania dotyczące projektu

Wymagania sprzętowe:

- Procesor INTEL Pentium 100 MHz lub ekwiwalent,
- Pamięć RAM 128 MB,

- ü Przestrzeń dysku twardego około 1GB,
- ü Karta graficzna z minimalną paletą kolorów 65000.

Wymagania programowe:

- ü J2SDK v1.3 – 1.4.2_02,
- ü Windows 98/2000/XP,
- ü Biblioteka SWT.

4. Założenia i uwagi dotyczące projektu

Przebieg procesu kompilacji:

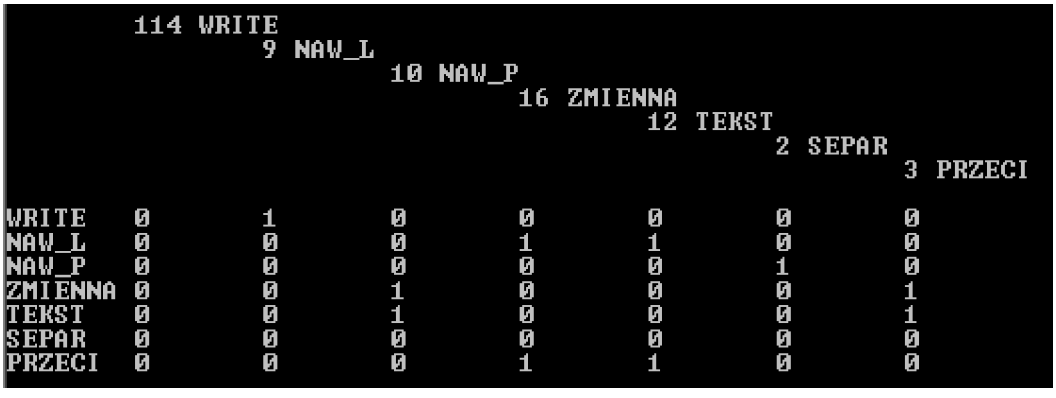
- analiza kodu, której wynikiem jest lista słów (klasa Analizator)
- rozpoznanie typów słów (klasa PascalParser)
- walidacja (klasa Validator)
- konwarsja kodu pascalowego na kod wykonywalny (klasa CodeConverter)

Przebieg procesu walidacji:

Proces walidacji skladni polega na sprawdzeniu kolejnosci wystepowania wyrazów w programie. Walidator pobiera dwa kolejne wyrazy, a nastepnie sprawdza czy pierwszy z nich jest wejsciem do walidacji jakiegos fragmentu programu (for, if, itd.). Jesli tak to rozpoczyna sie walidacja w oparciu o odpowiednia macierz przejsc (klasa ValidationMatrix).

Przykladowa macierz pokazano na rysunku 1. Wiersze zawieraja typy słów wejsciowych a kolumny wyjsciowych. Jedynka na przecieciu oznacza, ze dwa typy moga wystapic po sobie. Na przyklad po zmiennej moze wystapic nawias prawy badz przecinek

.



	114 WRITE	9 NAW_L	10 NAW_P	16 ZMIENNA	12 TEKST	2 SEPAR	3 PRZECI
WRITE	0	1	0	0	0	0	0
NAW_L	0	0	0	1	1	0	0
NAW_P	0	0	0	0	0	1	0
ZMIENNA	0	0	1	0	0	0	1
TEKST	0	0	1	0	0	0	1
SEPAR	0	0	0	0	0	0	0
PRZECI	0	0	0	1	1	0	0

rysunek 1

Wyjście z walidacji następuje w momencie natrafienia na słowo wyjściowe. W przeciwnym wypadku sprawdzane są kolejne wejścia do macierzy. Przy sprawdzaniu składni dodatkowo sprawdzane są inne zależności (np. przy sprawdzaniu wyrazeń trzeba sprawdzic zgdnosc typów oraz deklaracje zmiennych). Jesli wystepuja bledy zglaszany jest wyjatek.

Przebieg procesu konwersji kodu pascalowego na wykonywalny:

Celem konwersji kodu jest:
- uzyskanie prostych instukcji, które beda latwe do zinterpretowania i wykonania przez klase Executor,
- uwzglednienie zagniezdzen przez wyliczanie adresów skoków
- utworzenie plilku, który moze byc od razu wykonany (aktualnie przez serializacje kodu)

Dostepne instrukcje:
- SET - ustalanie wartosci zmiennej, parametry: zmienna, :=, wyrazenie
- BLOCK_OPEN - otwarcie bloku - wykorzystywane tylko przy konwersji kodu
- BLOCK_CLOSE - zamkniecie bloku - wykorzystywane tylko przy konwersji kodu
- READ - odczyt z klawiatury, parametry: lista zmiennych
- READLN - odczyt z klawiatury, parametry: lista zmiennych
- WRITE - wyswietlanie na konsoli, parametry: zmienne, tekst
- WRITELN - wyswietlanie na konsoli, parametry: zmienne, tekst
- JUMP - skok bezwarunkwy
- COMPARE_VALUE - wyrazenie do porównania, parametry: :=, wyrazenie
- JUMPG, JUMPS, JUMPE, JUMPNE, JUMPEG, JUMPES - skoki warunkowe, parametry: adres, zmienna lub wartosc
- INC - inkrementacja zmiennych, parametry: lista zmiennych
- DEC - dekrementacja zmiennych, parametry: lista zmiennych
- JUMP_MODIFY - modyfikacja skoku, wykorzystywane tylko przy konwersji kodu, parametry: adres skoku do modyfikacji, :=, wyrazenie do sprawdzenia

Proces konwersji polega na:
- wybieraniu z kodu potrzebnych informacji
- zamianie instrukcji pascalowych na odpowiadajace im instrukcje
- wyliczeniu adresów skoków - przy uzyciu stosu (CodeStack)

Obsluga stosu na kod (klasa CodeStack):

Wkładanie linii kodu na stos odbywa sie przy petlach i warunkach.
Zdejmowanie przy instrukcjach i na koncach bloków.

Przykład 1.

Kod wejsciowy:

```
var
x: integer;
begin
for x:=1 to 10 do
writeln('loop');
writeln('end');
end.
```

Kod wynikowy:

```
0: SET X,:=,1
1: COMPARE_VALUE :=,10
2: JUMPG 6,:=,X
3: WRITELN 'loop'
4: INC X
5: JUMP 1
6: DEC X
7: WRITELN 'end'
8: HALT
```

Proces konwersji:

- znaleziono petle FOR
- ustalenie wartosci poczatkowej SET X,:=,1
- ustalenie wartosci do porównania COMPARE_VALUE :=,10
- zapisanie skoku warunkowego JUMPG - bez parametrów
- zapisanie na stosie DEC X - korekcja zmiennej, po wykonaniu petli x = 10
- zapisanie na stosie JUMP_MODIFY 2,:=,X - dla modyfikacjia skoku JUMPG
- zapisanie na stosie JUMP 1 - skok do poczatku petli COMPARE_VALUE
- zapisanie na stosie INC X - modyfikacja zmiennej petli
- dodanie do kodu WRITELN 'loop'
- sprawdzenie stosu, jesli nie jest pusty i wyraz na stosie nie jest END tzn, ze po petli nie wystapil blok, wiec trzeba zdjac kod ze stosu do napotkania END,

- zdjęcie kodu ze stosu, jesli natrafimy na JUMP_MODIFY modyfikujemy wszystkie paametry JUMPG

Wykonywanie programu przez Executor:

Executor pobiera kolejne linie kodu wskazywane przez programCounter

Dzialania podejmowane przez Executor:
- SET - oblicza wyrazenie (klasa ObliczWyraz) i ustala wartosc zmiennej
- READ - wywluje metode read, która jest nadpisywana przy tworzeniu instancji executora
READLN - wywluje metode read, która jest nadpisywana przy tworzeniu instancji executora, wyswietla wynik, ustawia wartosc zmiennej
- WRITE - wywoluje metode display(str), która jest nadpisywana przy tworzeniu instancji executora
- WRITELN - wywoluje metode display(str), która jest nadpisywana przy tworzeniu instancji executora
- JUMP - ustawia programCounter i blokuje jego modyfiakacje
- COMPARE_VALUE - oblicza wyrazenie (klasa ObliczWyraz) i ustala wartosc zmiennej do porównania
- JUMPG, JUMPS, JUMPE, JUMPNE, JUMPEG, JUMPES - oblicza wyrazenie (klasa ObliczWyraz), porównuje COMPARE_VALUE, jesli warunek skoku jest spelniony modyfikuje programCounter i blokuje jego modyfiakacje
- INC - pobiera zmienne i zmienia ich wartosc
- DEC - pobiera zmienne i zmienia ich wartosc

Dodatkowa funkcjonalnosc Executora (moze byc wykrzystana w kolejnych wersjach):

- dostepna modyfikacja wskaznika programu - dzieki temu mozemy sterowac wykonaniem programu
- dostepne wykonanie 1 linii kodu, wskazywanej przez wskaznik programu (metoda runStep())

Uruchamianie programu z wiersza poleceń:

Kompilacja
java -classpath ApInterPas.jar Compiler plik_wejsciowy.pas plik_wynikowy.run

Uruchomienie skompilowanego programu w konsoli
java -classpath ApInterPas.jar Executor plik_wynikowy.run

Uruchomienie skompilowanego programu z interfejsem graficznym
java -classpath ApInterPas.jar RunningFrame plik_wynikowy.run

Uruchomienie skompilowanego programu z interfejsem graficznym z opcja debug
java -classpath ApInterPas.jar RunningFrame plik_wynikowy.run debug

Uruchamianie projektu
javaw -jar ApInterPas

5. Podsumowanie

Podczas realizacji projektu zdobyliśmy doświadczenie w projektowaniu Interpretera dla języka Pascal na przykładzie prostego Interpretera, którego zaprojektowaliśmy w języku Java. Podczas realizacji zapoznaliśmy się z metodami implementacji kompilatorów oraz z witryną SOURCEFORGE.

Wynikiem zadania postawionego przed nami powstał projekt „APInterPas". W projekcie zostały opracowane implementacja oraz sposób działania prostego interpretera języka programowania Pascal. Nie jest to profesjonalne rozwiązanie tego problemu, lecz z pewnością można powiedzieć, że główny cel został osiągnięty, a funkcjonalność aplikacji jest zadowalająca. Przyczyną tak prostego rozwiązania były ograniczenia czasowe.

W rezultacie projektu powstała aplikacja o przejrzystym interfejsie graficznym wspólnie z proponowanymi funkcjami. Oczywiście to nie jest końcowa wersja programu. Projekt zostanie umieszczony na naszej witrynie WWW oraz na witrynie SourceForge.net, w celu rozwijania aplikacji można będzie dodawać nowe moduły oraz dołączać nowe komponenty.