CSCE 425/825 - Compiler Construction

StaticJava Project

(пересказ с английского)

Задача – построить синтаксический анализатор для ограниченного подмножества языка Java:

- Только два типа данных: boolean and int
- Нет объектов (следовательно, нет и обработки исключений), массивов и параллельных потоков
- Программа состоит только из одного класса с методом main [для случая трансляции в трёхадресный код этот метод – единственный в классе]
- Некоторые ограничения в синтаксисе выражений и операторов цикла (см. ниже)
- Весь код в одном файле

Синтаксис

<*program*> ::= <*class-declaration*>

<*class-declaration*> ::= "**public**" "**class**" *ID* "**{**" <*main-method-declaration*> <*field-or-method-declaration*>* "**}**"

<*main-method-declaration*> ::= "**public**" "**static**" "**void**" "**main**" "(" "**String**" "**[**" "**]**" *ID* ")" "**{**" <*method-body*> "**}**"

<*field-or-method-declaration*> ::= <*field-declaration*> | <*method-declaration*>

<*field-declaration*> ::= "**static**" <*type*> *ID* "**;**"

<*method-declaration*> ::= "**static**" <*return-type*> *ID* "(" <*params*>? ")" "**{**" <*method-body*> "**}**"

<*type*> ::= "**boolean**" | "**int**"

<*return-type*> ::= <*type*> | "**void**"

<*params*> ::= <*param*> ("**,**" <*param*>)*

<*param*> ::= <*type*> *ID*

<*method-body*> ::= <*local-declaration*>* <*statement*>*

<*local-declaration*> ::= <*type*> *ID* "**;**"

<*statement*> ::= <*assign-statement*> | <*if-statement*> | <*while-statement*> | <*invoke-exp-statement*> | <*return-statement*>

<*assign-statement*> ::= *ID* "**=**" <*exp*> "**;**"

<*if-statement*> ::= "**if**" "**(**" <*exp*> "**)**" "**{**" <*statement*>* "**}**" ("**else**" "**{**" <*statement*>* "**}**")?

<*while-statement*> ::= "**while**" "**(**" <*exp*> "**)**" "**{**" <*statement*>* "**}**"

<*invoke-exp-statement*> ::= <*invoke-exp*> "**;**"

<*return-statement*> ::= "**return**" <*exp*> "**;**"

<*exp*> ::= <*literal-exp*> | <*unary-exp*> | <*binary-exp*> | <*paren-exp*> | <*invoke-exp*> | <*var-ref*>

<*literal-exp*> ::= <*boolean-literal*> | *NUM*

<*boolean-literal*> ::= "**true**" | "**false**"

<*unary-exp*> ::= <*unary-op*> <*exp*>

<*unary-op*> ::= "+" | "-" | "**!**"

<*binary-exp*> ::= <*exp*> <*binary-op*> <*exp*>

<*binary-op*> ::= "+" | "-" | "*****" | "/" | "**%**" | ">" | ">=" | "==" | "<" | "<=" | "**!**=" | "**&&**" | "||"

<*paren-exp*> ::= "**(**" <*exp*> "**)**"

<*invoke-exp*> ::= (*ID* "**.**")? *ID* "**(**" <*args*>? "**)**"

<*args*> ::= <*exp*> ("**,**" <*exp*>)*

<*var-ref*> ::= *ID*

ID = ('a'..'z' | 'A'..'Z' | '_' | '$') ('a'..'z' | 'A'..'Z' | '_' | '0'..'9' | '$')*

NUM = '0' | ('1'..'9') ('0'..'9')*

Примеры

```
public class Factorial
{
	public static void main(String[] args)
	{
		StaticJavaLib.println(
			factorial(StaticJavaLib.getIntArgument(args, 0)));
	}

	static int factorial(int n)
	{
		int result;
		int i;

		StaticJavaLib.assertTrue(n >= 1);
		result = 1;
		i = 2;
		while (i <= n)
		{
			result = result * i;
			i = i + 1;
		}
		return result;
	}
}
```