국민대학교
전자정보통신대학
컴퓨터공학부

컴파일러 과목 설계 프로젝트 중간보고서

프로젝트명	*rusper*
학 번	
성 명	
담당교수	
제출일	

SW 명칭	RUSPER **VERSION 0.1**
Date	

과제 명칭 변경 내역

제안서 과제명	rusper(RUby Scheme PERl) : 테스트케이스 작성에 특화된 언어
중간보고서 과제명	상 동
최종보고서 과제명	상 동

과제 명칭 변경 사유

중간보고서 과제명 변경 사유	해당 사항 없음
최종보고서 과제명 변경 사유	해당 사항 없음

목 차

1 목표 및 필요성

프로그래밍을 하는 경우, 실제 문제를 해결하기 위한 프로그램을 작성하는 경우도 많지만, 해당 프로그램이 실제로 해당 문제를 정확히 해결하는지 테스트 하는 프로그램을 작성하는 경우도 많이 있다. 특히 테스트를 위한 테스트케이스를 사람이 직접 작성하기도 하지만 보다 다양한 테스트를 하기 위해서 자동화된 테스트케이스를 작성하는 프로그램을 작성하기도 한다. 이때 일반적으로 테스트케이스는 단순한 입력 순서의 나열이거나 간단한 알고리즘으로 생성할 수 있다. 하지만 기존의 언어를 활용하는 경우 I/O 관련된 처리가 언어마다 다르고 복잡하다. 테스트케이스를 작성하는데 필요한 내용이나 알고리즘에 비해 I/O처리가 더 까다롭거나 복잡한 경우가 대부분이다. 따라서 이러한 I/O 관련된 처리를 보다 쉽게 하면서 원래 하려고 했던 간단한 테스트케이스를 작성하는데 집중할 수 있게 도와주는 언어를 설계하게 되었다.

rusper는 복잡한 I/O 관련 출력을 대괄호([,])를 통해 간단하게 추상화하고, 연산자는 전위표현을 사용하여 우선순위에 의한 미묘한 문제를 피하고, 중첩된 표현식(expression)을 사용하여 여러 개의 연산자를 적용하는데도 문제가 없다.

2 테스트케이스 작성에 특화된 언어 설계

2.1 스크립트 작성에 특화된 언어의 특징

2.1.1 Ruby의 출력 형식

Ruby에서는 STDOUT의 출력을 $stdout 변수의 값을 변경함으로써 STDOUT에서 임의의 파일로 출력을 redirection을 할 수 있다. 또한 일반적인 String Literal에서 내부에 #{}를 이용하면 #{} 내부의 표현식(expression)을 평가(evaluate)하여 String Literal을 구성할 수 있다.

2.1.2 Scheme의 연산자

Scheme에서는 모든 연산자가 가장 왼쪽에 오는 전위표현식(prefix experession)을 사용한다. 또한 피연산자(operand)의 위치에 표현식(expression)이 올 수 있으므로 중첩하여 표현식을 표현할 수 있다.

2.1.3 Perl의 $ 연산자

Perl에서 $로 시작하는 변수는 특별한 의미를 가지고 문맥에 따라 다양하게 해석될 수 있다. 이러한 $를 이용하면 보다 쉽게 다양한 표현식(expression)을 간결하게 작성할 수 있다.

2.2 테스트케이스 작성에 특화된 언어 rusper(RUby Scheme PERl) 설계

Ruby의 $stdout 변수의 값을 변경하는 방법을 도입하여 OUTPUT 키워드를 이용하여 redirection을 할 수 있다. 또한 String Literal 내부에 기술하는 #{}를 도입하여 중괄호({, })를 이용하여 표현식(expression)의 평가된(evaluated) 값을 출력할 수 있다. 출력의 경우에는 대괄호([,])로 감싼 내용은 STDOUT으로 출력하거나 OUTPUT 키워드를 이용하여 redirection을 한 경우에는 redirection한 파일로 출력된다. Scheme의 전위표현식(prefix expression)을 도입하여 모든 표현식은 연산자가 가장 왼쪽에 오고 피연산자는 소괄호((,))로 둘러싼다. 연산자는 기본적으로 1개에서 2개까지 가능하다. 만약에 여러 개의 연산자를 중첩하는 경우에는 피연산자에 표현식을 중첩하는 방식으로 기술하면 모든 표현식을 기술할 수 있다. 반복문의 경우에는 FOR, DO, LOOP 이렇게 3가지 방식을 지원한다. FOR의 경우에는 매개변수가 2개의 정수이고, DO는 1개의 정수, LOOP는 매개변수가 없다. FOR는 구간을, DO는 0부터 주어진 정수 – 1 까지(즉 주어진 정수만큼 반복), LOOP는 무한반복을 나타낸다. 반복문의 내부에서 BREAK와 $를 사용할 수 있다. BREAK의 경우 무조건적으로 반복문을 빠져나오며, Perl에서 도입한 $의 경우(단, Perl의 용법과는 조금 다르다) 현재 반복횟수를 나타낸다. IF, FOR, DO, LOOP는 복합문을 가져야 하고, 해당 복합문은 콜론(:)으로 둘러싼다.

2.3 rusper의 구문 구조 (EBNF로 기술)

```
<program> ::=           <file>
<file> ::=              <statement_list>
<statement_list> ::=    <statement>
                        | <statement> <statement_list>
<statement> ::=         <declaration_statement>
                        | <expression_statement>
                        | <selection_statement>
                        | <break_statement>
                        | <for_statement>
                        | <do_statement>
                        | <loop_statement>
                        | <print_statement>
                        | <output_statement>
                        | <comment_statement>
<declaration_statement> ::=     "VAR" <identifier>
<expression_statement> ::=      <expression>
<selection_statement> ::=       "IF" "(" <expression> ")" ":" <statement_list> ":"
<break_statement> ::=           "BREAK"
<for_statement> ::=             "FOR" "(" <integer_constant> "," <integer_constant> ")"
                                ":" <statement_list> ":"
<do_statement> ::=              "DO" "(" <integer_constant> ")" ":" <statement_list> ":"
<loop_statement> ::=            "LOOP" ":" <statement_list> ":"
<print_statement> ::=           "[" <print_list> "]"
<print_list> ::=                <print>
                                | <print> <print_list>
<print> ::=                     <string_literal>
                                | "{" expression "}
<output_statement> ::=          "OUTPUT" "(" <string_literal> ")"
<comment_statement> ::=         ";" { <character_constant> }1+
<expression> ::=        <assignment_expression>
                        | <arithmetic_expression>
                        | <equality_expression>
                        | <relational_expression>
                        | <logical_expression>
                        | <random_expression>
```

```
                        | <iterator_expression>
                        | <string_literal>
                        | <integer_constant>
                        | <identifier>
<assignment_expression> ::=     "=" "(" <lvalue> "," <expression> ")"
<arithmetic_expression> ::=     "+" "(" <expression> "," <expression> ")"
                        | "-" "(" <expression> "," <expression> ")"
                        | "*" "(" <expression> "," <expression> ")"
                        | "/" "(" <expression> "," <expression> ")"
                        | "%" "(" <expression> "," <expression> ")"
<equality_expression> ::=       "==" "(" <expression> "," <expression> ")"
                        | "!=" "(" <expression> "," <expression> ")"
<relational_expression> ::=     "!" "(" <expression> ")"
                        | "&&" "(" <expression> "," <expression> ")"
                        | "||" "(" <expression> "," <expression> ")"
<logical_expression> ::=        "<" "(" <expression> "," <expression> ")"
                        | ">" "(" <expression> "," <expression> ")"
                        | "<=" "(" <expression> "," <expression> ")"
                        | ">=" "(" <expression> "," <expression> ")"
<random_expression> ::= "RANDOM" "(" <integer_constant> "," <integer_constant> ")"
<iterator_expression> ::=       "$"
<string_literal> ::=            """ { <character_constant> }1+ """
<character_constant> ::=        "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l"
                        | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "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" | "W" | "X" | "Y" | "Z" | "_" | " "
<integer_constant> ::=          <decimal_constant>
                        | "+" <decimal_constant>
                        | "-" <decimal_constant>
<decimal_constant> ::=          "0" | <nonzero_digit> <digit_sequence>
<nonzero_digit> ::=             "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<digit_sequence> ::=            { <digit> }1+
<digit> ::=                     "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<lvalue> ::=                    <identifier>
<identifier> :=                 { <character_constant> }1+
```

3 rusper의 어휘분석

3.1 rusper에 대한 Lex 입력

rusper.l

```
%{

/*
 * Kookmin University
 * 컴퓨터공학부
 * 3학년
 * 정태성
 *
 * rusper.l
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"

%}
letter      [a-zA-Z_]
nonzero_digit      [1-9]
digit       [0-9]
%%
"VAR"               return T_VAR;
"IF"                return T_IF;
"("                 return T_LPAREN;
")"                 return T_RPAREN;
":"                 return T_COLON;
"BREAK"             return T_BREAK;
"FOR"               return T_FOR;
","                 return T_SEPARATOR;
"DO"                return T_DO;
"LOOP"              return T_LOOP;
"["                 return T_LBRACKET;
"]"                 return T_RBRACKET;
"{"                 return T_LCURLYBRACE;
"}"                 return T_RCURLYBRACE;
"="                 return T_ASSIGNMENT;
"+"                 return T_PLUS;
"-"                 return T_MINUS;
"*"                 return T_MUL;
"/"                 return T_DIV;
"%"                 return T_REMAINDER;
"=="                return T_EQUAL;
"!="                return T_NOT_EQUAL;
```

```
"!"                 return T_NOT;
"&&"                return T_AND;
"||"                return T_OR;
"<"                 return T_LT;
">"                 return T_GT;
"<="                return T_LT_EQUAL;
">="                return T_GT_EQUAL;
"RANDOM"            return T_RANDOM;
"$"                 return T_ITERATOR;
"OUTPUT"            return T_OUTPUT;
";".*               return T_COMMENT;
₩".*₩"              return T_STRING_LITERAL;

{letter}({letter}|{digit})*  return T_IDENTIFER;
[-+]?{digit}+                return T_INTEGER;

[ ₩t₩n₩r]           ;
.                   return T_ERROR;

%%

// /*
int main(void)
{
        int token_number;

        printf("Start of rusper₩n");
        while(0 != (token_number = yylex())) {
                printf("<%d, %s>₩n", token_number, yytext);
        }

        printf("End of rusper₩n");

        return 0;
}
// */

int yywrap(void)
{
        return 1;
}
```

3.2 어휘분석기 실험 결과

3.2.1 gcd.rus

Source	; 최대공약수 구하기 VAR a VAR b VAR r ; initialize a =(a, 64) ; initialize b =(b, 12) LOOP : =(r, a) =(a, b) =(b, %(r, b)) IF(<=(b, 0)) : BREAK : : ; print gcd ["gcd is " {a}]
실험 결과	Start of rusper <293, ; 최대공약수 구하기> <259, VAR> <291, a> <259, VAR> <291, b> <259, VAR> <291, r> <293, ; initialize a> <273, => <261, (> <291, a> <266, ,> <292, 64> <262,)> <293, ; initialize b> <273, => <261, (>

```
<291, b>
<266, ,>
<292, 12>
<262, )>
<268, LOOP>
<263, :>
<273, =>
<261, (>
<291, r>
<266, ,>
<291, a>
<262, )>
<273, =>
<261, (>
<291, a>
<266, ,>
<291, b>
<262, )>
<273, =>
<261, (>
<291, b>
<266, ,>
<278, %>
<261, (>
<291, r>
<266, ,>
<291, b>
<262, )>
<262, )>
<260, IF>
<261, (>
<286, <=>
<261, (>
<291, b>
<266, ,>
<292, 0>
<262, )>
<262, )>
<263, :>
<264, BREAK>
<263, :>
<263, :>
<293, ; print gcd>
<269, [>
<294, "gcd is ">
<271, {>
<291, a>
<272, }>
<270, ]>
```

	End of rusper

3.2.2 prime.rus

Source

```
; 소수 판별 프로그램

VAR prime
VAR i
VAR flag

OUTPUT("prime_result.txt")

; initialize prime
=(prime, 131)

; initialize flag
=(flag, 0)

IF(&&(<(2, prime), !=(0, %(prime, 2))))
:
        ; initialize i
        =(i, 3)

        LOOP
        :
                IF(<=(prime, i))
                :
                        =(flag, 1)
                        BREAK
                :

                IF(==(0, %(prime, i)))
                :
                        BREAK
                :

                =(i, +(i, 2))
        :
:

IF(==(1, flag))
:
        [{prime} "is prime."]
:

IF(==(0, flag))
:
        [{prime} "is not prime."]
:
```

	; end of source
실험 결과	Start of rusper <293, ; 소수 판별 프로그램> <259, VAR> <291, prime> <259, VAR> <291, i> <259, VAR> <291, flag> <290, OUTPUT> <261, (> <294, "prime_result.txt"> <262,)> <293, ; initialize prime> <273, => <261, (> <291, prime> <266, ,> <292, 131> <262,)> <293, ; initialize flag> <273, => <261, (> <291, flag> <266, ,> <292, 0> <262,)> <260, IF> <261, (> <282, &&> <261, (> <284, <> <261, (> <292, 2> <266, ,> <291, prime> <262,)> <266, ,> <280, !=> <261, (> <292, 0> <266, ,> <278, %> <261, (> <291, prime> <266, ,> <292, 2> <262,)>

```
<262, )>
<262, )>
<262, )>
<263, :>
<293, ; initialize i>
<273, =>
<261, (>
<291, i>
<266, ,>
<292, 3>
<262, )>
<268, LOOP>
<263, :>
<260, IF>
<261, (>
<286, <=>
<261, (>
<291, prime>
<266, ,>
<291, i>
<262, )>
<262, )>
<263, :>
<273, =>
<261, (>
<291, flag>
<266, ,>
<292, 1>
<262, )>
<264, BREAK>
<263, :>
<260, IF>
<261, (>
<279, ==>
<261, (>
<292, 0>
<266, ,>
<278, %>
<261, (>
<291, prime>
<266, ,>
<291, i>
<262, )>
<262, )>
<262, )>
<263, :>
<264, BREAK>
<263, :>
<273, =>
```

```
<261, (>
<291, i>
<266, ,>
<274, +>
<261, (>
<291, i>
<266, ,>
<292, 2>
<262, )>
<262, )>
<263, :>
<263, :>
<260, IF>
<261, (>
<279, ==>
<261, (>
<292, 1>
<266, ,>
<291, flag>
<262, )>
<262, )>
<263, :>
<269, [>
<271, {>
<291, prime>
<272, }>
<294, "is prime.">
<270, ]>
<263, :>
<260, IF>
<261, (>
<279, ==>
<261, (>
<292, 0>
<266, ,>
<291, flag>
<262, )>
<262, )>
<263, :>
<269, [>
<271, {>
<291, prime>
<272, }>
<294, "is not prime.">
<270, ]>
<263, :>
<293, ; end of source>
End of rusper
```

3.2.3 random.rus

Source	; random list generator DO(10) : ; print current index [{$}] ; generate random number 1 ~ 3 [{RANDOM(1, 10)}] :
실험 결과	Start of rusper <293, ; random list generator> <267, DO> <261, (> <292, 10> <262,)> <263, :> <293, ; print current index> <269, [> <271, {> <289, $> <272, }> <270,]> <293, ; generate random number 1 ~ 3> <269, [> <271, {> <288, RANDOM> <261, (> <292, 1> <266, ,> <292, 10> <262,)> <272, }> <270,]> <263, :> End of rusper

3.2.4 sum.rus

Source	; 1 부터 10까지 합 VAR sum ; initialize sum =(sum, 0)

	; repeat 10 times DO(10) : =(sum, +(sum, +($, 1))) ; print sum [{sum}] : ; end of source
실험 결과	Start of rusper <293, ; 1 부터 10까지 합> <259, VAR> <291, sum> <293, ; initialize sum> <273, => <261, (> <291, sum> <266, ,> <292, 0> <262,)> <293, ; repeat 10 times> <267, DO> <261, (> <292, 10> <262,)> <263, :> <273, => <261, (> <291, sum> <266, ,> <274, +> <261, (> <291, sum> <266, ,> <274, +> <261, (> <289, $> <266, ,> <292, 1> <262,)> <262,)> <262,)> <293, ; print sum> <269, [> <271, {> <291, sum> <272, }>

	<270,]> <263, :> <293, ; end of source> End of rusper

4 향후 추진 계획

어휘분석을 바탕으로 구문분석을 하고 목적코드를 생성한다.

4.1 향후 계획의 세부 내용

세부내용	11/23	11/30	12/07	12/14
rusper 언어 Lex 입력 파일 작성 및 수정				
rusper 언어로 작성된 샘플 소스 작성 및 수정				
rusper 언어 Yacc 입력 파일 작성 및 수정				
구문분석 테스트 및 결과 정리				
목적코드 생성기 작성				
rusper 언어에 대한 최종보고서 작성				
rusper 언어에 대한 자체평가 보고서 작성				
rusper 언어로 작성된 소스 파싱결과를 트리형태로 출력해주는 프로그램 작성 (예정)				

5 과제 수행 문제점 및 건의 사항

기본적으로 새로운 언어를 설계하는 것은 매우 어렵다. 기존 언어들의 특징과 장단점을 잘 이해하고, 또한 최근 각광받고 있는 객체지향(OOP) 언어 또는 함수형(FP) 언어를 모두 고려하여 언어를 설계하려고 한다면 제한된 시간에 달성할 수 있는 목표는 많지 않다. 언어에 대한 경험이 없어 언어 설계에 대한 경험적인 측면에서는 매우 유용하나, 실제로 사용 가능한 언어를 설계하는 데는 한계가 있다. 또한 해당 언어로 작성된 소스를 목적코드로 변환하는 과정은 언어에 대한 내용뿐만 아니라 하드웨어와 운영체제에 관련된 내용을 모두 포함하는 매우 포괄적이고 어려운 내용이다. 따라서 목적코드에 대한 내용 자체만으로도 상당히 큰 내용이므로 목적코드에 관련된 내용을 간소화하고 언어 설계에 대해 집중한다면 더 높은 완성도의 언어 설계를 달성할 수 있을 것이다.