

CS F363 Compiler Construction

Assignment-2

1 Language Specification

Consider a programming language, basic PASCAL, which supports integer constants, keywords, variables, expressions, assignments, conditional statements, loops, arrays, single line comments (using `//`) and read and write statements:

1. **Keywords:** program, int, real, boolean, char, var, for, do, while, array, and, or, not, begin, end, read, and write. The keywords are not case-sensitive.
2. **Variables:** The name of a variable can be composed of letters, digits, and the underscore character. It must begin with a letter. The variable names are not case-sensitive, so uppercase and lowercase letters mean same here. However, the keywords are not allowed to use as a variable names.

All variables must be declared before we use them in the program. All variable declarations are followed by the `var` keyword. A declaration specifies a list of variables, followed by a colon (`:`) and the type. Syntax of variable declaration is

```
var
    variable_list : type;
```

Here, *variable_list* is a list of variables separated by a comma (,) and *type* from the list `{Character, Integer, real, boolean}`.

Example:

```
var
    age, weekdays : integer;
    taxrate, net_income: real;
    choice, isready: boolean;
    initials, grade: char;
```

Note that at the variable declaration, assigning a value to one or more variables is not allowed.

3. **Operators:** An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. We allow the following types of operators:
 - Arithmetic operators:
 - + (*addition*) ,
 - − (*subtraction*),
 - * (*multiplication*),
 - / (*division*, return real value),
 - % (*remainder*, returns integer type)
 - Relational operators:
 - = (equals, comparison operator)
 - <> (not equal to)
 - <, >, <=, >= (these operators have usual meaning)
 - Boolean operators:
 - and** : boolean AND operator, if both the operands are true, then condition becomes true.
 - or** : boolean OR Operator. If any of the two operands is true, then condition becomes true.
 - not**: boolean NOT Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

4. **Assignment statement:** To assign a value to a variable, follow this syntax:

```
variable_name := expression;
```

here the *expression* is a single value or a variable, or an arithmetic expression over constants/variables with arithmetic operators mentioned above.

5. **Block of statements:** A set of one or more statements is consider as a block and each block starts with **begin** and ends with **end;**.

```
begin
    statement_1;
    ....
    .....
    statement_k;
end;
```

6. **Conditional statements:** We allow if-then, if-then-else, and nested if statements as in PASCAL language.
7. **Looping statements:** We allow while-do and for-do loops as in PASCAL.
8. **Arrays:** we consider only one-dimensional arrays as in PASCAL.
9. **Read and Write statements:**

- **write:** it prints the text or values of variables on the screen. The syntax is given below:

```
write("text");    // prints the text on the screen.
write(variable_list) //prints the values of the variables on the screen.
```

example:

```
write("Welcome to CS F363")
write(day, age, cost) //day, age, cost are variables and the type of these variables
need not be the same.
```

- **read:** takes the value from the user as an input and the syntax is given below:

```
read(id) //where id is a variable in the program
```

10. **Program structure:** the program starts with keyword “program” followed by the name of the program and terminates with ; (semicolon). Next is the variable declaration section, then followed by main program block. The main program block starts with keyword “begin” and ends with keyword “end” followed by a period (.).

```
program name_of_the_program;

var
    variabl_list:type;    //declaration of variables

begin // main program block starts

end. // the end of main program block
```

Here, *name_of_the_program* follows the rules of variables.

Example:

```
program AddTwoNumbers;  
  
var  
  num1, num2, sum: Integer;  
  
begin  
  Write('Enter the first number: ');  
  read(num1);  
  
  Write('Enter the second number: ');  
  read(num2);  
  
  // Perform addition  
  sum := num1 + num2;  
  
  // Display the result  
  Write("The sum is ");  
  write(sum)  
end.
```

2 Tasks

1. **Lexical Analysis [6 marks]** - For this phase, write a LEX program that breaks the input program into tokens (identifiers, operators, numbers, keywords, punctuators, etc.). A symbol table is created with the list of tokens obtained from the input.
2. **Syntax Analysis [6 marks]**- For this phase, write a YACC program that takes tokens generated after lexical analysis and checks the given input program is syntactically correct or not.
3. **Semantic Analysis [8 marks]**- Extend your YACC program written in the last phase to examine semantic errors and the abstract syntax tree is printed.
Semantic errors like type checking, undeclared variables, multiple declarations of variables are verified.
4. **Code Generation [8 marks]** - Extend your YACC program in the last phase to generate the intermediate code in three address code format.
5. **Final stage: [8 marks]** Finally, extend your YACC program to print the output of the given input program.

3 Instructions

1. You work on the assignment with your groups members and strongly discouraged to discuss with other group members.
2. You can refer internet/web resources only to understand the syntax of PASCAL language.
3. Taking code from internet/web is strictly prohibited. If you do so, it will lead to severe penalty.
4. Submit the assignment in two parts:
 - (a) **Part-1:** Submit the tasks 1, 2, and 3 (Dead line: 20 April 2024 11:59 PM)
 - (b) **Part-2:** Submit the tasks 4 and 5 (Dead line: 30 April 2024 11:59 PM)
5. Late submissions will be allowed up to maximum 24 hrs after the deadline with penalty of 2% per each hour.

4 Submission guidelines

1. Only one submission per group.
2. Place the LEX and YACC programs (name the programs with your group submission code) for each task in a separate folder and name the folder with the task name.
3. Create a *readme.txt* file that contains the information of the group members and mention how to compile the programs for each task.
4. Finally, place all the folders and readme.txt inside a new folder (name it with your group submission code) and zip the folder and submit it.
5. Your programs must execute on Ubuntu 23.04.

5 Input and Output formats

1. Input will be given in a file and the name of the file will be given at the runtime.
2. You print the output on the terminal and the format for each task is given below.
3. **Lexical Analysis:** print the list of valid tokens in the given input program in the following format

line number	lexeme	token type
-------------	--------	------------

4. **Syntax Analysis:** if the input program has any syntax error print “syntax error” message, otherwise print “valid input”. (No need to print the type of the syntax error.)
5. **Semantic Analysis:** Print the abstract syntax tree (if the input program has no syntax errors). For the structure and further details refer the web link <https://www.epaperpress.com/lexandyacc/calcg.html>
If a variable is used before declaration, print an error message “undeclared variable” along with the name of the variable.
If a variable is declared more than once, print an error message “multiple declarations of a variable” along with the name of the variable.
Further, print if there are any type mismatches in the given input program like, a real value is assigned to an integer variable (similar cases), a character type is added to integer type, etc.
6. **Code Generation:** print the three-address code generated in the form of **Quadruple** (see lab sheet 10 for the exact structure (will upload in the coming weeks).)
7. **Final stage:** print the output of the given input and print the symbol table in the following format:

Variable	Type	Value
----------	------	-------

-----All the best-----