sT: A Simple Turing Programming Language

Programming Assignment 2 Syntactic and Semantic Definitions

Symmetre and Semantic Benintions

Due Date: 1:20PM, Wednesday, May 25, 2023

Your assignment is to write an LALR(1) parser for the sT language. You will have to write the grammar and create a parser using **yacc**. Furthermore, you will do some simple checking of semantic correctness. Code generation will be performed in the third phase of the project.

1 Assignment

You must create an LALR(1) grammar using **yacc**. You need to write the grammar following the syntactic and semantic definitions in the following sections. Once the LALR(1) grammar is defined, you can then execute **yacc** to produce a C program called "**y.tab.c**", which contains the parsing function **yyparse**(). You must supply a main function to invoke **yyparse**(). The parsing function **yyparse**() calls **yylex**(). You will have to revise your scanner function **yylex**().

1.1 What to Submit

You should submit the following items:

- revised version of your lex scanner
- a file describing what changes you have to make to your scanner
- your **yacc** parser

 Note: comments must be added to describe statements in your program
- Makefile
- test programs

1.2 Implementation Notes

Since **yyparse()** wants tokens to be returned back to it from the scanner, you should modify the definitions of **token**, **tokenInteger**, **tokenString**. For example, the definition of **token** should be revised to:

```
#define token(s,t) {LIST; printf("<%s>\n",s); return(t);}
```

2 Syntactic Definitions

2.1 Data Types and Declarations

The predefined scalar data types are **bool**, **real**, **int**, and **string**. The only structured type is the *array*. There are two types of constants and variables in a program:

- global constants and variables declared outside functions and procedures
- local constants and variables declared inside functions, procedures, or blocks

Constants

A constant declaration has the form:

```
CONST ID con
```



```
const identifier \langle :type \rangle := constant\_exp
```

where $constant_exp$ is an expression of constant operands and $\langle \ \rangle$ denotes that type declaration is optional. Note that assignments to constants are not allowed after declaration. For example,

```
const s :string := "Hey There"
const i := -25
const f := 3.14
const b :bool := true
```

Variables

A variable declaration has the form:

```
var identifier \langle :type \rangle := constant_{exp}
```

where the type declaration could be optional. Another form of variable declaration has the format:

```
var identifier : type \langle := constant_exp\rangle
```

where the value initialization could be optional. For example,

```
var s :string
var i := 10
var d :float
var b :bool := false
```

Arrays

Arrays can be declared by

```
var identifier: array num... num of type

We will look back later.

How to put an array in symbol tuble?
```

For example,

2.2 Program Units

The two program units are the *program* and *functions*.

2.2.1 Program

A program has the form:

```
⟨ zero or more variable, constant, or function declarations ⟩
⟨ zero or more statements ⟩
```

where the item in the $\langle \rangle$ pair is optional.

2.2.2 Functions and Procedures

A function declaration has the following form:

```
function identifier \langle ( zero or more formal arguments separated by comma ) \rangle: type \langle zero or more variable, or constant declarations and statements \rangle end identifier
```

Parentheses are not required if no arguments are declared. No functions may be declared inside a function. The formal arguments are declared in a formal argument section, which is a list of declaration separated by comma. Each declaration has the form

```
identifier:type
```

Note that if arrays are to be passed as arguments, they must be fully declared. All arguments are passed by values.

Procedures are similar to functions, but return no value at all.

```
procedure identifier ( zero or more formal arguments separated by comma ) 

zero or more variable or constant declarations and statements 
end identifier
```

For example, following are valid function or procedure declaration headers:

```
function func1(x :int, y :int, z :string) :bool
function func2(a :bool):int
procedure func3
```

The name of every function must be unique. For example, the following is a program:

```
% Example
% constant and variable declaration
const a :int := 5
var c :int
% function decclaration
function add (a :int, b :int) :int
  result a+b
end add
```

```
/% main statements
c := add(a, 10)
if (c > 10) then
  put -c
else
  put c
end if
put ("Hello World")
```

2.3 Statements

There are several distinct types of statements.

2.3.1 blocks

A block consists of a sequence of declarations and statements delimited by the keywords begin and end.

```
begin⟨zero or more variable or constant declarations and statements⟩end
```

Note that variables and constants that are declared inside a block are local to the statements in the block and no longer exist after the block is exited.

2.3.2 simple

The simple statement has the form:

```
identifier := expression
or
put expression
or
get identifier
or expression
result identifier or return
or
exit \( \text{ when bool_expression } \)
or
skip
```

expressions

Arithmetic expressions are written in infix notation, using the following operators with the precedence:

```
(1) - (unary)

(2) * / mod

(3) + -

(4) < <= = => > not=

(5) not

(6) and

(7) or
```

Note that the – token can be either the binary subtraction operator, or the unary negation operator. All binary operators are left associative, except for the uniary operators in list (1). Parentheses may be used to group subexpressions to dictate a different precedence. Valid components of an expression include literal constants, variable names, function invocations, and array reference of the form

△ A [integer_expression]

function invocation

A function invocation has the following form:

```
identifier \langle ( expressions separated by zero or more comma ) \rangle
```

2.3.3 conditional

The conditional statement may appear in two forms:

```
if boolean_expr then
      \ zero or more variable or constant declarations and statements \
      \(\rm zero or more variable or constant declarations and statements\)
      end if
or
      if boolean_expr then
      \(\rangle\) zero or more variable or constant declarations and statements \(\rangle\)
      end if
```

2.3.4 loop

The loop statement has the form:

```
loop
       ⟨ zero or more variable or constant declarations and statements ⟩
       end loop
or
       for \( \) decreasing \( \) identifier : num .. num
      \(\rangle\) zero or more variable or constant declarations and statements \(\rangle\)
       end for
```

2.3.5 procedure invocation

A procedure is a function that has no return value. A procedure call is then an invocation of such a function. It has the following form:

```
identifier ( zero or more expressions separated by comma )
```

where the parentheses should not be included when there are no actual arguments.

3 Semantic Definition

The semantics of the constructs are the same as the corresponding Pascal and C constructs, with the following exceptions and notes:

- The parameter passing mechanism for procedures is call-by-value.
- Two arrays are considered to be the same type if they have the same number of elements and the types of elements are the same.
- Scope rules are similar to C.
- Types of the left-hand-side identifier and the right-hand-side expression of every assignment must be matched.
- Type declaration of a function must be matched with the type of its return value. Furthermore, the types of formal parameters must match the types of the actual parameters.

4 Example sT Program

```
{% Sigma.st
 % Compute sum = 1 + 2 + ... + n
% constant and variable declarations
const n :int := 10
var sum :int
var index :int
% for loop
sum := 0
for index: 1 .. n
 sum := sum + index
end for
put "The result is "
put sum
skip
% loop
sum := 0
index := 1
loop
 sum := sum + index
 index := index + 1
 exit when index = n
end loop
put "The result is "
put sum
```

5 yacc Template

```
응 {
#include <stdio.h>
               printf(t)
#define Trace(t)
응 }
/* tokens */
%token SEMICOLON
응응
program:
              identifier semi
              Trace("Reducing to program\n");
semi:
              SEMICOLON
              Trace("Reducing to semi\n");
응응
#include "lex.yy.c"
FILE *yyin; /* file descriptor of source program */
yyerror(msg)
char *msg;
   fprintf(stderr, "%s\n", msg);
main(int argc, char *argv[])
   /* open the source program file */
   if (argc != 2) {
      printf ("Usage: sc filename\n");
      exit(1);
   /* perform parsing */
   if (yyparse() == 1)
                                  /* parsing */
      yyerror("Parsing error !"); /* syntax error */
}
```

4	h 5	1.	lo	N .		r. <i>1</i>	- Fr																				
Ħ	f (MC	ano	وو	٦.	. ta	b _ V																					
	الساء ،																										
Ţp	*	Cons	st s	2 %		ssign	1,	retur	n e	XYBY	M	essay	e														
Ą	教	60	rahie	- 7	存	syn] تولم	table	2																		
				A	-Æ 1	nemo	y is																				
/	J		•					0																			
	,							Р.	11																		
						nst an	t, or	tenetta	n decle	wat ion	<i>></i>																
	(let	0 67	- ste	tomer	Œ/																						
	r	۔ ذعبہ	24	内	ス・	可定	*	ſ,	و دخت	74		经备	D YO	fw/		三年	Ī	St.	1 5								
	Tav.	1.		L.	2 -	JA	スス	שני האל	李紫	数	,	1~3)	/F '	,,,	41								
- I	21 OC	ron		. 7M	MICH		-		4 -7	-4/																	
		Const 如果 變數 Proje (zer (zer	Const a \$ \$ const \$ \$ project \$ (zero or (zero or	Const a Do R const s State of value Project R fa (zero or val (zero or se	Const a \$ 7 Const & 4 \$ \$ Const & 4 \$ \$ \$ Const & 4 \$ \$ \$ Const & 4 \$ \$ Const & 4 \$ \$ Const & 4 \$ Cons	Const a Do R const R 被 d Do R const R 被 d Do R const R 被 d A const R 和 d Czero or vapable, ro (zero or setement)	Const a Do not	Do R Court & M assign, State of value 不存在symbol A Exemory if Project 是有 I dimension as (zero or variable, constant, or (zero or setement)	Const a To R const R it assign, return to Resident Resident and table for memory te. Project R to I dimension array. (zero or restance) Fartion A FIRE has	Const a To Reconst & the assign, return a second table for memory the. Project & To I d'impossion aroy. (zero or varable, constant, or function decle (zero or sistemost) Favorion N 7 TR & function	Const a Do R const R 被 assign, return error Do R const R 被 assign, return error Do R const R 在 symbol table A R memory 但. Project R 有 I d'imension arroy. (zero or retenent) Cero or retenent T 可尼義 function	Const a Do R const & AR assign, return error m Response to the assign error error m Response to the assign error	Const a To & court & the assign, return error messay to be some of the symbol table The memory the. Project & to I dimension arroy. (zero or rationers) (zero or retoners)	Const a Do R const & 被 assign, return error message Do R const & project & 有 I d'imension array. (zero or rationale) Cero or setonale)	Const a To & const & AR assign, return error message Extra 65 value 7-74 symbol table Fix memory 12. Project & A I d'imansion arroy. (zero or variable, constant, or function declaration) (zero or statement) Fixerion M 7-92 & function.	Const a Do R const R 株 assign, return error message Do R const R 株 assign, return error message Do R const R to assign, return error message De R to a remove the total and the tot	Const a To R const & lett assign, return error message Lett show had table Fire nemony te. Project & to I dimension array. (zero or retement) (zero or retement) To R & Lynchron & Return to Function declaration.	Const a Do R const & 被 assign, return error message Do R const & 在 assign, return error message Do R const & 在 fat gm bol table A memory 证. Project & 有 I dimension arroy. (zero or retonent) (zero or retonent)	Const a Do R const x 被 assign, return error message Do R const x 神 assign, return error message Do R const x 神 assign, return error message De R const x 有 I d'imension arroy. (zero or rotante, constant, or function declaration) (zero or sutante) To return 本 不可定義 function. (De return 要 標明祭	Const a Do R const & 被 assign, return error message State of value 不存在 symbol table 存在 memory 证. Project 不有 I dismonston arroy. (zero or variable, constant, or function declaration) (zero or statement) Function	Const a Do R const & ik assign, return error message who so the 7-17在 symbol table fine nemory the. Project & 有 I dimension arroy. (zero or retained) (zero or retained) Function to 不可定義 function. (20 return 表	Const a Do R const x 被 assign, return error message De Dob value 不存在 gun bol table Fix memory 证. Project x 有 I d'imension arroy. (zero or variable, constant, or function declaration) (zero or stronger) Fix metion 所 不可定義 function. (De return 要 存納 多数	Const a Do R const 又被 assign, return error message Do R to the 不存在 gm bol table Fix memory 证. Project 只有 I d'imension appy. (zero or variable, ronstant, or function declaration) (zero or statement) Fix motion 所 不可定義 function. 後面 return 要 標明學教	Const a \$ \$ \$ const & the assign, return error message \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$	Const a \$ \$ \$ const & the assign, return error message \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$	Const a Do a Const x 被 assign, return error message Do a const x 被 assign, return error message Do a const x 在 spendol table Fix memory it. (zero or variable, constant, or function delimation) (zero or retemore) Fix metion 所 不可定義 function. 後面 return 要 存 明然 数	Const a Do a Const x 被 assign, return error message Do a sold value 不存在 symbol table 存在 memory 证. Project x 有 I d'imension apoy. (zero or variable, constant, or function declaration) (zero or systement) Function 所 不可定義 function. 後面 return 要 標明然数