Compilers Project

Team Members

- Abdullah Zaher Abu Sedo
- Bahaa Eldeen Mohamed
- Ebrahim Gomaa
- Tarek Samy

Project overview

Phase 1:

we designed an interpreter with
1- variable and constant declaration
2 – mathematical and logical expressions
3 – assignment statement
4 – if then else
5 – while loops
6 – for loops
7 – repeat until loops
8 – switch statements
9 – break and continue
10 – print and toggle debug

we support 4 main datatype
1 – int
2 – bool
3 – char
4 – float
we support implicit type casting

we interpreted commands using abstract syntax tree approach

Phase 2

we converted interpreter to a complete compiler that generates quadruples
We added a GUI to enter the code and get:

1. Quadruples
2. Symbol table
3. errors

removed instructions

1. break and continue
2. print and toggle debug
3. drop support for strings

Tokens

Token	description	regex
Integer_	Integer numbers	({ZERO}| {DIGIT_NO_ZERO} {DIGIT}*)
float_	Floating numbers	({ZERO}| {DIGIT_NO_ZERO} {DIGIT}*\.{DIGIT}*)
char_	Single character	'.'
bool_	boolean	"true"|"false"
(){};	open_bracket, close bracket, open curly braces, close curly braces, semi column	
+-*/%	Mathematical operations	
|| && ! >= <= < > != ==	Logical operations	
	If, else , while, for, repeat, until, break, case, continue, switch,...etc	Just check for constant work
var_name_	Variable names, must start with a-z,A-Z and can contains alphabetic and digits, case sensitive	{CHAR}({DIGIT}|{CHAR})*

Quadruples

Quadruple general formula

OP , arg1, arg2, result

Quadruple	Description	operation
Assign, arg1, null, result	Assign operation	Result = arg1
PLUS, arg1,arg2,result	+	Result = arg1 + arg2
MINUS, arg1,arg2,result	-	Result = arg1 - arg2
MUL, arg1,arg2,result	*	Result = arg1 * arg2
DIV, arg1,arg2,result	/	Result = arg1 / arg2
MOD, arg1,arg2,result	%	Result = arg1 % arg2
GT, arg1,arg2,result	>	Result = arg1 > arg2
GTE, arg1,arg2,result	>=	Result = arg1 >= arg2
LT, arg1,arg2,result	<	Result = arg1 < arg2
LTE, arg1,arg2,result	<=	Result = arg1 <= arg2
EQ, arg1,arg2,result		Result = arg1 == arg2
NOTEQ, arg1,arg2,result	!=	Result = arg1 != arg2
AND, arg1,arg2,result	&&	Result = arg1 && arg2
OR, arg1,arg2,result	||	Result = arg1 || arg2
NOT, arg1,null,result	!	Result = !arg1
UMINUS,,arg1,null,result	- (unary minus)	Result = -arg1
JT arg1,null,null	Jump to arg1 => label, if result of previous boolean expression is true	JT arg1=>label
JNT arg1,null,null	Jump to arg1 => label, if result of previous boolean expression is false	JNT arg1=>label
JMP arg1,null,null	Jump to arg1 => label unconditionally	JMP arg1 => label
Label arg1,null,null	Define a label with name arg1	Label label1

Errors detected in semantic analysis

1 – multiple variable declaration
2 – syntax error => parser phase
3 – using undefined variable
4 – assigning value to constant variable after declaration
5 – perform mathematical operation on 2 incompatible datatype (in phase 2 this cant happen as all the types can implicitly be converted to one another)
6 – perform mod on 2 non integers

Symbol Table

1. id of variable
2. variable name
3. variable type
4. variable data
5. is constant or not

Tools and Technologies used

1. C to write all the logic GCC
2. Flex for the lexical analyzer
3. Bison for the parser generation
4. PySimpleGUI to create the GUI