Faculty of Engineering
Cairo University

Big Data Project Phase 2

Team 7

Name	SEC	BN
كريم علاء عبد الله	2	3
أحمد أسامة فوزي زهران	1	6
محمود أسامة محمود خطاب	2	15
محمد عادل محمد عز الدين	2	9

Under Supervision of:
Eng. Abdelrahman Kaseb

Table of Contents

Overview

- **YAPL** _ Yet Another Programming Language _ is a programming language designed to be c-like syntax, the language supports a variety of operations and data types that allow the developer to perform some simple tasks.

- The given compiler accepts a file with. yapl extension that contains your code and checks for any syntax and semantic errors “as much as possible” and inform the user with errors and dump it to a text file – if exists – or else it will run the program generating the output stream written the the language, generate a comma separated files (Symbol Table, Quadruples, Optimized Quadruples).

- Attached with the language a PyQt5 GUI that provides a simple code editor that allows you to do some functionalities:
    - Write code in a text editor
    - Showing the output files in a user-friendly manner
    - Highlighting erroneous lines
    - Upload text files
    - Save current file
    - Run the code

Tools And Technologies

- Flex
- Bison
- CPP
- PyQt5
- Python

YAPL Tokens

Target Description	Regex (simplified)	Examples
Keywords	return | switch | case | default | continue | break | const | true | false | if | else | while | do | for | print	-
Special Functions	pow | sqrt | print	-
Data Types	Int | float | char | string | bool | void	-
Integer Values	0+ | -?[0-9]+[0-9]*	000000 10 -10 -1
Float Values	0+\.0* | [-+]?[0-9]*\.[0-9]*	000000. 0.0 -0.10 +.10
Character Value	' . '	‘a’ ‘.’ ‘0’
String Value	\"[^"]*\"	“xyz” “123” “av12”
Comment	\/\/.*\n?	// this is a comment
Operators	[+ - * / %] << | >> && | || == | != ++ | -- += | -= | *= | /= | %= | ^= | &= | |= | <= | >=	-

Language Syntax

1- Variable Declaration

What Does This Node Consist Of

- The name of the variable being declared
- The data type of this variable and whether it is constant
- The initialization value of this definition, will be null if it's a definition without assignment
- The next variable that will be defined with the same type as the current definition

How Does Production Rule Generated

```
VariableDeclaration:
      Type IDENTIFIER
    | Type IDENTIFIER '=' Operation
    | VariableDeclaration ',' IDENTIFIER
    | VariableDeclaration ',' IDENTIFIER '=' Operation
    | {$$ = nullptr;};
```

How Does the Quadruples Look Like

- The variable definition itself consists of one record

Operation	Operand1	Operand2	Result
Declare	Data Type	-	Variable Name

- In case there is some initializing value it should be like the following

Operation	Operand1	Operand2	Result
Declare	Data Type	-	Variable Name
The Generated Quadruples for The RHS Value			
=	Result Variable	-	Variable Name
The Generated Quadruples for The Next Define Node Connected To The Current Node			

What Kinds of Errors We Handle

- If the variable being defined was defined before in the same scope, a "Variable already defined will be logged
- If the LHS type doesn't match the RHS, a "Type Mismatch" error will be logged
- When the LHS cannot be resolved, the expression itself will log the cause of the error

Examples

Const int x = 10;	int x;
float y = 10.2;	float y;
Const bool z = false;	Const bool z;
Bool x, y = false;	Const Bool x = true, z = false;
String w = "hello", z = "there";	Char c, d , e;

2- Variable Assignment

What Does This Node Consist Of

- The name of the variable being assigned
- The value of the LHS being assigned
- The next assignment node that will be executed after the current assignment node

How Does Production Rule Generated

```
VariableAssignment:
    IDENTIFIER '=' Operation    { $$ = new AssignNode($1.line, $1.str, $3); }

  | Operation {
      $$ = new AssignNode(lineNumber, "", $1);
    }
```

- This rule Is extended furthermore for symbols like (+=, -+, *=, /=, %=, &=, |=) but was trimmed for the sake of the report cleanliness
- It can be a “floating Operation” so that we can call functions or perform some sentences like (x++, x--)

How Does the Quadruples Look Like

Operation	**Operand1**	**Operand2**	**Result**
The generated Quadruples for The RHS Expression -> RHS target			
=	RHS target	-	DEST
The Generated Quadruples for The Next Assign Node Connected to The Current Node			

What Kinds of Errors We Handle

- When the variable being assigned was not defined, a “Variable Not Exist” error will be printed in the text file
- When the variable being assigned type doesn’t match the LHS type, a “type mismatch” error will be printed
- When the LHS cannot be resolved, the expression itself will log the cause of the error

Examples

x = 10;	z = false;
y = 10.2;	c = ‘a’;

3- Function Definition

What Does This Node Consist Of

- The name of the function being defined
- Function parameters
- The body of the function
- Return type of the function

How Does Production Rule Generated

```
FunctionDefinition:
      Type IDENTIFIER '(' FunctionParams ')' Scope {
        $$ = new FunctionDefintionNode($2.line, $1, $2.str, $4, dynamic_cast<ScopeNode*>($6));
      }


FunctionParams:
      Type IDENTIFIER ',' FunctionParams
    | FunctionDefaulValueParams
    | Type IDENTIFIER
    |

FunctionDefaulValueParams:
      Type IDENTIFIER '=' FunctionParamOperand ',' FunctionDefaulValueParams
    | Type IDENTIFIER '=' StringOperand ',' FunctionDefaulValueParams
    | Type IDENTIFIER '=' FunctionParamOperand
    | Type IDENTIFIER '=' StringOperand
```

How Does the Quadruples Look Like

- Note that we didn't add the function parameters, because we don't support function redefinition with different signature, so the function body in assembly should be knowing that the first N elements on the stack are its arguments

Operation	Operand1	Operand2	Result
FUNC_START	Function name	-	-
Generated Quadruples for the Function Body			
FUNC_END	Function name		

What Kinds of Errors We Handle

- Function Redefinition will give a "function already declared" error
- Declaring default value parameters before required parameters will cause a syntax error
- Invalid return type for the function

Examples

Int f(){ return 0; }	Float f(int y = 0) { return y; }
Void f() { print("hello"); }	Float f(int x, char c = 'a') { return x + c; }
Void f() { print("hello"); return; }	bool f(int x) { return x > 4; }

4- Function Call

What Does This Node Consist Of

- The name of the function being called
- The passed parameter

How Does Production Rule Generated

```
FunctionCall:
        IDENTIFIER '(' FunctionCallParams ')'
    |   IDENTIFIER '(' ')'

FunctionCallParams:
        FunctionCallParams ',' Operation
    |   Operation
```

How Does the Quadruples Look Like

- It's not necessary for the function parameters to be equal to the whole set of function arguments
- In case of a function call not passing the whole parameters, it will be replaced with the given default value which was assigned and stored in the function definition node.

Operation	Operand1	Operand2	Result
ARG	Parameter 1 value Or Default value 1	-	-
ARG	Parameter 2 value Or Default value 2	-	-
ARG	Parameter N value Or Default value N	-	-
Call	Function Name	Number Of Arguments	tx

What Kinds of Errors We Handle

- Calling Non-Existent Function, will log "function x is not defined"
- Passing parameters more than the total parameters needed by the functions
- Passing parameters less than the required parameters by the function
- Type mismatch between the passed parameters and the defined ones

Examples

F();	Int x = F();
F(0);	Float x = F(0, 1);
F(0, 1, 4);	Char c = F(1, 1, 1);

5- If Sentence

What Does This Node Consist Of

- The condition on which the branching decision will be mane
- The acceptance scope or program node
- The rejection scope or program node

How Does Production Rule Generated

```
IFSentence:
        IF '(' VariableAssignment ')' IfExtension

      | IF '(' VariableAssignment ')' IfExtension ELSE IfExtension


IfExtension:
        Sentence                                                    {
      | Scope
```

How Does the Quadruples Look Like

Operation	Operand1	Operand2	Result
Generate The Quadruples Of The Condition Expression -> Tx			
Jump If False	Tx	-	Rejection Label
Generate Acceptance Node Quadruples			
JMP	-	-	Skip Label
Label	-	-	Rejection Label
Generate Acceptance Node Quadruples			
Label	-	-	Skip Label

What Kinds of Errors We Handle

- Calling Non-Existent Function, will log “function x is not defined”
- Passing parameters more than the total parameters needed by the functions
- Passing parameters less than the required parameters by the function
- Type mismatch between the passed parameters and the defined ones

Examples

If (condition) {} else {}	If (){ if() {} else do_something() }
If (condition) do_somthing();	If() {} else if (){} else {}

6- Switch Sentence

What Does This Node Consist Of

- The operation on which we will branch i.e switch (x), switch(x + 3) ,...
- Switch Body, which consists of a bunch of 'Case' Nodes that is followed by bunch of statements that will be executed given that the case condition was satisfied
- Default case condition is nullptr, meaning that it is always satisfied

How Does Production Rule Generated

```
SwitchSentence:
    SWITCH '(' Operation ')' SCOPE_BEGIN SwitchBody SCOPE_END { $$ = new SwitchNode($1.line, $3, $6); }
  | SWITCH '(' Operation ')' SCOPE_BEGIN SCOPE_END            { $$ = new SwitchNode($1.line, $3, nullptr); }

SwitchBody:
    Case Sentences                {
      $$ = new SwitchBody(dynamic_cast<CaseNode*>($1)->assignBody($2));
    }
  | SwitchBody Case Sentences     {
      $$ = dynamic_cast<SwitchBody*>($1)->addCase(dynamic_cast<CaseNode*>($2)->assignBody($3));
    }

Case:
    CASE Operation ':'            { $$ = new CaseNode($1.line, $2); }
  | DEFAULT ':'                   { $$ = new CaseNode($1.line);};
```

How Does the Quadruples Look Like

- In case of encountering a break inside any case, it will jump to the Exit Switch Label

Operation	Operand1	Operand2	Result
Generate The Quadruples Of The Switch Variable -> Tx			
Label	-	-	Case "I" condition Label
==	Tx	Case "I" Condition Value	Ty
Label	-	-	Case "I" Skip Condition Label
Jump If False	Ty	-	Case "I + 1" condition value
Generate Case Body			
JMP	-	-	Case "i+1" Skip Condition Label
Generate Next case body (starting from the second line but for I + 1)			
Label	-	-	Exit Switch Label

What Kinds of Errors We Handle

- Type mismatch between the passed parameters and the defined ones

7- For Sentence

What Does This Node Consist Of

- The sentences that are meant to be executed before starting the loop (optional)
- The condition (optional but will give you a warning)
- The sentences that are meant to be execute after finishing one iteration (optional but gives a warning)
- The loop body

How Does Production Rule Generated

```
ForSentence: FOR '(' VariableDeclaration ';' ForCondition ';' ForPostSentence ')' LoopScope {
    $$ = new ForNode($1.line, $3, $5, dynamic_cast<AssignNode*>($7), dynamic_cast<ScopeNode*>($9));
}
| FOR '(' VariableAssignment ';' ForCondition ';' ForPostSentence ')' LoopScope {
    $$ = new ForNode($1.line, $3, $5, dynamic_cast<AssignNode*>($7), dynamic_cast<ScopeNode*>($9));
}

ForCondition: Operation |  {$$ = nullptr;};

ForPostSentence:
      VariableAssignment                        { $$ = $1;  }
    | ForPostSentence ',' VariableAssignment    { $$ = dynamic_cast<AssignNode*>($1)->setNextAssignment($3); };
    | {$$ = nullptr;};
```

How Does the Quadruples Look Like

Operation	Operand1	Operand2	Result
Generate The Quadruples Of The Pre Loop Nodes			
JMP	-	-	Loop Condition Label
Label	-	-	Loop Body
Generate Quadruples for The Loop Body			
Generate Quadruples for The Post Loop Nodes			
Label	-	-	Loop Condition Label
Generate Quadruples for The Condition -> Tx			
Jump If True	Tx	-	Loop Body

What Kinds of Errors We Handle

- Nothing new, the same for the type mismatching and accessing variables that doesn't exist

Examples

For(;;){ do_something(); }	For(int I = 0, j =1, I < 0 && j > 1; i++, j--) { do_something() }
For(int I = 0; ; i++){}	For(;;i++){}

8- While Sentence

What Does This Node Consist Of

- The condition
- The loop body

How Does Production Rule Generated

```
WhileSentence: WHILE '(' VariableAssignment ')' LoopScope {
    $$ = new WhileNode($1.line, dynamic_cast<AssignNode*>($3), dynamic_cast<ScopeNode*>($5));
}
```

How Does the Quadruples Look Like

Operation	Operand1	Operand2	Result
JMP	-	-	Loop Condition Label
Label	-	-	Loop Body
Generate Quadruples for The Loop Body			
Label	-	-	Loop Condition Label
Generate Quadruples for The Condition -> Tx			
Jump If True	Tx	-	Loop Body

What Kinds of Errors We Handle

- Nothing new, the same for the type mismatching and accessing variables that doesn't exist

Examples

while(true){ do_something(); }	While(x > 0) {}
While(1){do_something();}	While(isReady()) {}
While(x++){}	While(--x);

9- Do While Sentence

What Does This Node Consist Of

- The condition
- The loop body

How Does Production Rule Generated

```
WhileSentence: WHILE '(' VariableAssignment ')' LoopScope {
    $$ = new WhileNode($1.line, dynamic_cast<AssignNode*>($3), dynamic_cast<ScopeNode*>($5));
}
```

How Does the Quadruples Look Like

Operation	Operand1	Operand2	Result
Label	-	-	Loop Body
Generate Quadruples for The Loop Body			
Generate Quadruples for The Condition -> Tx			
Jump If True	Tx	-	Loop Body

What Kinds of Errors We Handle

- Nothing new, the same for the type mismatching and accessing variables that doesn't exist

Examples

Do {do_something(); } while(true);	Do {} while(x > 0);
Do {do_something();}While(1);	Do {} While(isReady());
Do {} While(x++);	Do {} While(--x);

10- Print Sentence

What Does This Node Consist Of

- The expression to be printed

How Does Production Rule Generated

```
PrintSentence:
    PRINT '(' Operation ')' ';'      { $$ = new PrintNode($1.line, $3);  }
```

How Does the Quadruples Look Like

Operation	Operand1	Operand2	Result
Generate Quadruples for The Expression			
Print	Tx	-	-

What Kinds of Errors We Handle

- Nothing new, the same for the type mismatching and accessing variables that doesn't exist

Examples

Print(x + y);	Print("hello");
Print("x = " + x);	Print(x + " = X");
Print("");	Print(x);

Quadruples Shortcuts

Symbol	Description	Operations
ASSIGN	Assign the first argument to the result	Dest = arg1
DECLARE	Declare a variable with type as arg1, and variable name as result	-
EQ NEQ LT LTE GT GTE AND OR NOT XOR SHL SHR	Logical Operations	Dest = arg1 == arg2 Dest = arg1 != arg2 Dest = arg1 < arg2 Dest = arg1 <= arg2 Dest = arg1 > arg2 Dest = arg1 >= arg2 Dest = arg1 && arg2 Dest = arg1 || arg2 Dest = !arg1 Dest = arg1 ^ arg2 Dest = arg1 << arg2 Dest = arg1 >> arg2
ADD	Adds the value in arg1 to the value in arg2 and store it in result	Dest = arg1 + arg2
SUB	Subtracts the value in arg1 to the value in arg2 and store it in result	Dest = arg1 – arg2
MUL	Multiplicate the value in arg1 to the value in arg2 and store it in result	Dest = arg1 * arg2
DIV	Divides the value in arg1 to the value in arg2 and store it in result	Dest = arg1 / arg2
MOD	Takes the modular of arg1 when divided by arg2 and store it in result	Dest = arg1 % arg2
NEG	Negates the value of Arg1 and store it in result	Dest = -1 * arg1
JMP	Unconditional jump to destination label	-
JNZ	Jumps to destination label if the value of Arg1 is not zero	-
JZ	Jumps to destination label if the value of Arg1 is zero	-
Print	Prints the value in Arg1	-
LABEL	Defines A Label For Jump Instructions	-
FUNC_START – FUNC_END	Defines [begin - end] of a function	-
ARG	Defines a parameter to be passed to a function	-
CALL	Calls a function with name stored in arg1 and with number of parameters	Dest = Function_Name()

	stored in arg2 and save the return value in result	
RET	Return the value in arg1 and exit the current function	Return arg1

What's More

- We have implemented a simple module that will try to optimize the result quadruples by removing as much as unnecessary records as we could implement.

- The code can run, so enjoy it.

- Expression is handled intensively such that it can accommodate any type casting, and in case of failure it will inform the compiler organizer that there is an error and running will not be possible

- We are operating on the AST (abstract syntax tree) so we done some optimization for the expression calculation, where we try to reduce the expression to one value if it is possible to do so, like if we have int x = 10 + 20, when we make this "Define Node" we can just reduce this 10 + 20 to 30, and this will make some optimizations for the memory usage of the compiler and more compact quadruples

THANK YOU