Compiler Construction

Specification File v/s Final Code Comparison

Name-

2018A7PS0165U-Ayushi Agarwal

2018A7PS0274U-Anshumaan Chauhan

* 1. Starting constructs in ***C*** language

#include <stdio.h>

int main ()

{

statements;

}

**We have successfully implemented this initial phase in our program. For this we had to replace the class classname { with an equivalent part which is mentioned above.**

* 1. Terminator symbol

Every statement must have an end of line character, which gives special meaning to the parser. Here ‘;’ semi-colon symbol will be used for marking the end of line. It is like both what is used in C as well as in Java.

**As our proposed language and C both use the same terminal symbol therefore it was easy to implement this part of the translation. No replacement or deletion was required we just needed to print ; whenever we found its occurrence.**

* 1. Data Types

In ***C*** we are going to use 5 different data types as follows:

1) integer

2) float

3) character

4) double

// we are not introducing String because in the end we will have to convert it into a character array in C.

**We successfully implemented all these data types, along with this we also coded the parser in such a way that it won’t allow any statement that has type mismatch. And will stop the translation part just after the line in which there was error, so that we can know which line/grammar rule to change in the parser.**

* 1. Declaration statements

1. datatype var\_name1, var\_name2, . . ., var\_namen; datatype can be any one of the above 5 specified. And var\_name1,…., var\_namen are the names of variables.
2. For arrays syntax will be a bit different as follows:
3. Datatype var\_name[row\_size]; this will declare 1D array variable of size row\_size of type ‘Datatype.’

Datatype var\_name[row\_size] [col\_size]; this will declare 2D array variable of size row\_size\*col\_size of type ‘Datatype.’

Similarly, datatype var\_name[x1][x2][x3][x4][x5]……[xn]; for declaring a n dimensional array.

In ***C*** language no default initialization feature will be provided, therefore, we must give them default values explicitly and after that we can perform some operation on them.

**The implementation of declaration of arrays along with the type checking was implemented successfully. Parser was successfully able to parse and translate only those statements that have same dimensions of the array and were of the same datatype.**

* 1. Operators

1. Equal will be used in place of = for the assignment operation
2. And will be used for the logical and operation (&&)
3. Or is used for logical or operation (||)
4. Not is used as !.
5. Rest everything including math operations will be same as C.

+ is for addition.

* Is for subtraction.

% is for getting the remainder

^ is for to the power, or exponent

\*Is for multiplication.

/ is for getting the quotient.

**All basic mathematical operations such as specified above are completely allowed and parsed by the parser. One condition is implemented with the use of mod operator, that is, it can only be used with integers. Therefore we had implemented a small code which checks the datatype of operands, and if integer only then allows to parse.**

**All logical operators can be used such as >,<,<=,>=,!= and ==. Along with these, we have also implemented operators such as &&,|| and ! in order to combine one or more logical expressions, which can be used in the looping and conditional constructs.**

* 1. Conditional Statements

1. If constructor

If there is only 1 condition to be compared then we will use the following format:

if (condition) //logical expression

{statements; //executed if the result of logical expression is true.

}

else

{statements; //executed if the result of logical expression is false.

}

In case we have many independent conditions to be compared then we use:

if(condition)

{

}

else

{if(condition)

{statements;

}

else

{statements;

}

}

1. Conditional operator

It is kind of a construct where we use comparison as well as assignment operation in a single statement.

(logical expression)? {statement to be executed if logical expression is true}:{ statement to be executed if logical expression is false }

***C***also supports operations such as break and continue like C language. Alongside we will try to incorporate the goto operator. Correspondingly, we could have labels in our program (LABEL followed by ’:’ operator)

1. Switch-case :

***C*** also supports the switch case, where we can execute a set of statements depending on the value of the variable. If the value of variable does not match to any of the cases then in that case, default case will be executed. The restriction for switch is that the variable should be of integer or character datatype only. The structure of the construct will be as follows:

switch(variable) :

case ‘x’ : statement Ax;

.

.

.

break;

case ‘y’ : statement Ay;

.

.

.

break;

.

.

.

.

.

default : statements;

**All the conditional constructs can be parsed by the parser. Nested if conditions were also validated by the parser. Ternary operator has a limitation of having a single program statement in the success or failure part of the conditional statement. Switch case was implemented successfully with the constraint that break is compulsory before ending of each case. Which also handles the case of fall through. Default case is also compulsory for executing a switch case, if there is no default case, parser will show an syntax error.**

* 1. Loop Structure

1. While loop is a construct which keeps on running until the condition inside the brackets is true. This construct will be implemented in the same manner as that in C, that is as follows:

While(condition)

{

Statements;

}

1. Do-while is a looping structure which by default runs 1 time before even checking whether the condition in the while is true or not. This construct is implemented in simple terms as follows:

repeat

{

Statements;

}while(condition);

1. The most used looping structured of all time, for loop will be implemented in ***C***  in a slightly different manner and will involve a bit of pseudocode.

for (i=initial value ***to*** final value ; increment or decrement step(example x=x+1

{ statements;

}

//can also declare variable i inside loop definition.

Remember that increment or decrement step is not optional, it is compulsory to write it, it does not have a default increment or decrement value.

* 1. Input and Output

1. Read(var1, var2, . . . , varn); //In contrast to C, the function Read() does not bother about

verifying the data types associated with variables.

2. Write(var1, var2, . . . , varn); // We will support any kind of interactive messages in our language. Example Write(‘Input value’);

Here we are following the same thing mentioned in the template provided in the classroom.

**They were successfully translated into printf and scanf instructions of C language. Along with it, we also added the appropriate data type keywords, i.e. if want to print x which is integer, then we use write(x) simply, but in C it gets converted to printf(“%d”,x), same thing was implemented for scanf instruction.**

* 1. main function

function int main() // see section 1.12 for function definitions.

{

statement1;

statement2;

.

.

.

statementn;

}

**Main is a common function for both our language and C language, therefore it was a simple translation based on the occurrence position of the main.**

* 1. User defined datatypes

We will also try to allow user to create his own user defined datatypes, like struct in C.

Uddcreate

{ datatype1 var1;

datatype2= var2;

}datatype\_name;

Now later user can create variable of type ‘datatype\_name’, in the same format in which variables are created for inbuilt datatypes. These datatypes should be created at the start of the main function, else they will give a syntax error during the parsing.

**We successfully implemented the declaration of user defined data types as well the declaration of the variables of user defined datatype. We were not able to implement the program statements involving variables of user defined datatype, because it was hard to store the values of internal variables of the different user defined datatype variables and then to refer them if we had more than 1 variable of each user defined datatype.**

* 1. Function Definitions

Functions can be defined using the keyword ‘func’ followed by the return type of that function and the name. Example:

func int sum (int a, int b)

{

Return a+b;

}

Compiler will check whether the return type is in accordance with the value that is being returned by the function. ‘return’ keyword will be used to pass the value back to the calling function, i.e., the main function ,as we are not implementing non leaf procedures here. Only valid return types will be what are already defined, therefore void return type will not be supported in the language.

Always remember to make a default constructor of the class, our language does not support the functionality where, a default constructor is created while compiling if not present already.

**Function defining is successfully implemented. Return type of the function and the datatype of variable which will store the returned value is also cross checked, which enables a error free function calls later also. We had to replace the func keyword, rest of the syntax of both C and proposed language is same.**

* 1. Class and Object

In ***C*** language every program should begin with a keyword “class” followed by class name. class name cannot be a keyword and should follow the same properties that identifier follows, with an exception that it cannot contain a ‘\_’ symbol in it.

***C*** has the concepts of objects, it is an instance of a class, which should follow the format:

Classname objectname=new classname();

We can only call functions created in the start of the program using the object.

**As C don’t use the concept of classes and object, although we have created them in our proposed language they are simply ignore in our translational part. But they are important as in the proposed language function calls will be allowed only when the object of that class is used.**

* 1. Function Calling

We can functions, with the help of objects.

The format for calling functions is as follows:

Var\_name = objectname.functioname(parameters);

Key point here to remember is that, the var\_name should be of same datatype which is returned by the function. As we are not allowing any functions that have no parameters to be declared therefore, parameters passed are also typed checked by the parser.

**Function calls have been successfully implemented along with the type checking of the passed parameters and the required parameters of the function.**

* 1. Keywords

Keywords are special sequence of characters that give a special meaning to the compiler. In our language, keywords cannot be used as class name, variable name, or function name. Therefore, this is one of the restriction or constraint that needs to be followed while writing the program in ***C***. Example of keywords are: if, switch, while, etc.

**Lex program has its rules accordingly placed so as to handle this issue, that is, keywords are initialized early, so lex returns that particular keyword to parser and if it doesn’t find any proper grammar rule for the statement error will be thrown.**

**Additionally, we also tried to implement the pointers, they are accepted by the programming language. And the parser also successfully parses them. But the translation of only declaration of the pointers was successfully implemented. And for the other operations, parser will parse them successfully, but the output will not have translated code for those statements.**