COMPILER DESIGN GRAMMAR PRE.

What is Compiler?

A compiler translates the code written in one language to some other language without changing the meaning of the program. It is also expected that a compiler should make the target code efficient and optimized in terms of time and space.

WHAT IS COMPILER DESIGN?

Compiler design principles provide an in-depth view of translation and optimization process. Compiler design covers basic translation mechanism and error detection & recovery. It includes lexical, syntax, and semantic analysis as front end, and code generation and optimization as back-end.

Little C

OUR PROJECT

WHAT WE GONNA DO

The grammar that we have designed systematically describes the execution flow from the import statement of the library to the end of the execution of the program.

SPECIAL HIGHLIGHTS

- Import Library
- $<Comments>$
- Special Structural Types(Stack, Queue, and Array)
- Letter = a | ... | z | A | ... | Z
- Digit = 0 | ... | 9
- Logical Operator = and | or
- Conditional Operators < | > | <= | >= | != | ==
- Arithmetic Operators = + | - | * | / | %
- Functions Declarations

THE GRAMMAR

- Program → Import [standard Library] Variable_declaration_ list Funtion_list
- Library → Math Library | String Library | Math | String | Ɛ
- Variable_Declaration_ List → Variable ; Variable_Declaration_List | Ɛ
- Funtion_List → Function;Function_List | Ɛ
- Function → Type id (ParameterList) Block | sType id(ParameterList) Block
- sType → Array | Stack | Queue
- Variable → Type id | Type sType id[Num]
- Type → Int | Float | Char | Bool | Void | Date_Time | String
- Id → Letters Alpha
- Alpha → Letters Alpha | Digit Alpha | _Alpha | Ɛ
- Parameter_List → Parameter Parameter_List | Ɛ
- Parameter → Type id | Type sType id | Ɛ
- Block → {(Variable_Declaration_List | Statement_List)*}
- Statement_List → Statement Statement_List | Ɛ
- Statement → Expression | Loop_Statement | Conditional_Statement | Function_Call_Statement | Return Expression | Return Id | break | Read(id); | Write(Text);

...CONTINUED

- Loop_Statement → for(id=expression; id Conditional_Operator expression; id=id Arithmetic_Operator Num) Block | while(expression) Block | do Block while(expression)
- Expression → Num | Expression Arithmetic_Operator Expression | Expression Conditional_Operator_Expression | Expression Logical_Operator Expression | {Expression} | [Expression] | (Expression) | id | id = Expression | Text | Ɛ
- Conditional_Statement → if(Expression) Block else Block | if(Expression) Block
- Function_Call_Statement → Function_Name (List_Of_Identifiers)|id=Function_Name(List_Of_Identifiers)
- Text → [a-zA-Z0-9\t\n ./<>?;:"'`!@#$%^&*()\[\]{}_+=|\\-]
- List_Of_Identifiers → id | id, List_Of_Identifier | Ɛ
- Conditional_Operator → < | > | != | >= | <= | ==
- Arithmetic_Operator → + | - | * | /
- Logical_Operator → OR | AND
- Num → [digit]+
- Letter → [a-z|A-Z]
- Digit →[0-9]
- Num →[+|-]?Digit[.]Digit

AN EXAMPLE

```
Import [Standard Math]
        int ab;
        int c;
        char d;
int main( ) {
        float z;
        write("Hello World");
        write("Enter Value");
        read("Number");
        for(i=1; i<= number; i=i+2) {
                write(i);
        }
}
```

REFERENCES

- Compiler principles, techniques and tools by Ravi sat, Jeffery d.ulman
- torialspoint.com/compiler_design

OUR TEAM

- Ravi Shankar Kiradoo(201451007)
- Saurabh Nitnaware(201451017)
- Rahul Chaurasia(201451039)
- Chirag Garg(201451052)
- Mukesh Kumar Sain(201451060)