PROJECT : PROGRAMMING LANGUAGES AND COMPILERS

NWirth
Compiler
User Manual

Submitted by:
Kiran Dhakal
st121108
CS

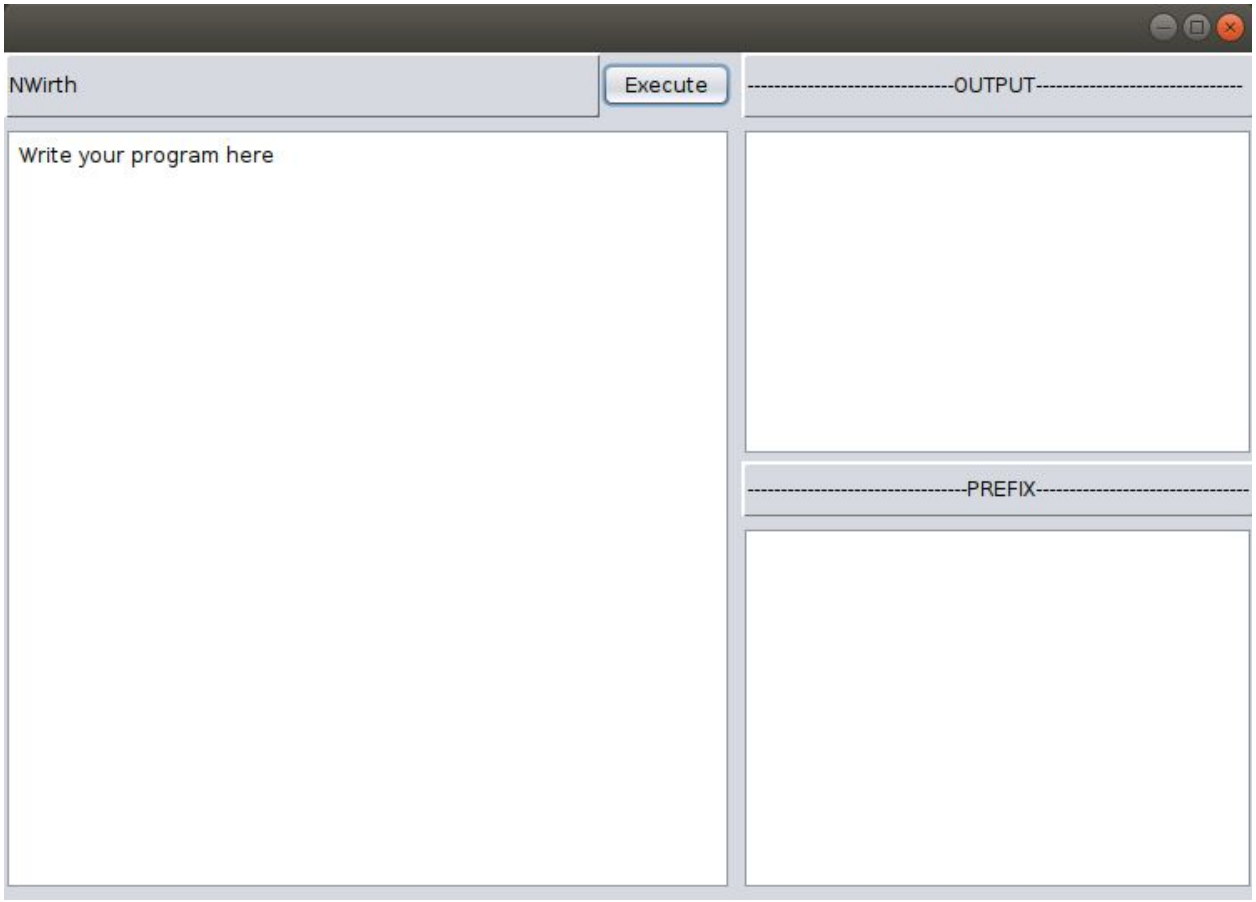
TABLE OF CONTENT

Introduction

This compiler for the language named “NWirth” after Niklaus Wirth, the creator of Pascal language, is created as an individual project requirement of Programming Languages and Compilers course in Asian Institute of Technology. The compiler is implemented in Java using JFlex and CUP.

Note For Evaluators:

The compiler is in a java project form. The project is CUP->ex6. The project can be compiled by using the build.xml file. Then, NWirthcompiler.java should be run to get a UI where the program can be written. And if the Execute button is clicked, the output is shown in the Output console, and the prefix of the program is shown in the Prefix console.



Data Types

Five different data types are supported by the compiler. They are:

1. Integer
2. Floating Point
3. Character
4. String
5. Boolean

Integer

Integer is defined as a number that is either 0 or that starts from any digit except 0. Negative integers are also supported. The INT keyword is used to declare and define the integers.
Examples:
Valid Integers : -1090988, -100, -1, 0, 1, 100, 1098734
Invalid Integers : -0000212, -01, 00, 00000, 01, 02, 000002, 00111123, 1.0

Floating Points

Floating Points are real numbers with a dot (.) symbol. They cannot start with multiple 0's. Negative floating points are also supported. The FLOAT keyword is used to declare and define the floating points.
Examples:
Valid floating points: - 1235.2398, -100.0, -1.111, 0.0, 1.23234, 100.12323, 12389834.1231932
Invalid floating points: -00001231.123, -01.0, 00.00, 00000, 01.0001, 00021312343.1, 1, -1

Character

Character is a single 16-bit character. The compiler supports any single alphabet or integer enclosed inside a single quotation (‘’).
Examples:
Valid characters: ‘0’, ‘A’, ‘z’
Invalid characters: “A”, ‘ab’, ‘01’, “0”, “-1”

String

Strings are data types used to store a sequence of characters enclosed inside double quotations (“”). String accepts any combination of alphabets (small or capital), digits or special symbols.

Example:
Valid Strings: “a”. “Aazzhshai”, “09876543”, “0”, “!%$”
Invalid Strings: ‘a’, Aazzhshai, ‘0’, ‘!%$’

Boolean

The boolean data type represents one bit of information. There are only two possible values: true and false. The default value of boolean is false.

Operators

Arithmetic Operators

- +, -, *, /, %, exp are supported for integers and float types separately
- + is also supported for string and character types separately and together and the result is always the string type.

Comparison Operators

- Both the expressions should be of the same type
- The values of the expressions are compared and not the memory location eg. two strings are equal if the values of the strings are equal, unlike in java
- Use of parentheses is mandatory when two comparisons are made and joined with a logical operator
- Output : boolean

<	:	supports integer, float, char
>	:	supports integer, float, char
<=	:	supports integer, float, char
>=	:	supports integer, float, char
==	:	supports integer, float, char, string, boolean
!=	:	supports integer, float, char, string, boolean

Examples

```
PRINT 1<2;
```
Output : true
```
PRINT true AND true;

PRINT 2<1;
```
Output : false

```
PRINT (1<2) AND (2<3);
```
Output : true

Logical Operators

- Supports AND and OR operators
- Both the expressions should be of the boolean type
- Supports multiple expressions eg. (true AND true) AND (true AND false) (and so on)

- In case of multiple expressions, precedence must be defined by using the parentheses ()
- Output : boolean

Examples of logical operators

```
PRINT true AND false;
```
Output : false

```
PRINT false AND true OR true;
```
Output : true

Examples of using comparison operators and logical operators together

```
PRINT 1<2 AND 2<3 AND 3<4;
```
Output : true

```
PRINT 1<2 AND 3<2 AND 3<4;
```
Output : false

Statements

A statement always ends with a semicolon. It can be of the following types:

- Print statement
- Declaration statement
- Definition statement
- If then else statement
- Assignment statement
- Do While statement
- Do Until statement

The statements are executed in order from top to bottom. Different statements have expressions as their properties. For instance, the PRINT statement has an expression that is evaluated before printing it to the output console, the IF statement has an expression that is evaluated before executing the statements inside it's block.

Declaration

Variables can be declared at any point of the program but should be declared before they are used or assigned. The compiler does not allow declaration of more than one variable with the same name. The compiler stores a default value of the variable when they are declared according to their data type.

Default Values

Integer: 0
Float: 0.0
Char: 'a'
String: " "
Boolean: false

Syntax:

<Data Type> <identifier>;

Examples

```
INT a;
FLOAT b;
CHAR char1;
STRING my_string2;
BOOLEAN abc;
```

Limitation

Does not support “INT a, b, c;” and similar syntax.

Initialization

A new variable can be initialized at the time of variable declaration. The program accepts variables of any data type supported by the programming language.

Syntax:

<Data Type> <identifier> = <value>;

Examples

```
INT a = 1;
FLOAT b = 20.0;
CHAR char1 = ‘c’;
STRING my_string2 = “Hello Welcome”;
BOOLEAN abc = true;
```

Limitation

Compiler does not support “INT a=1, b=2;” and similar syntax.

Assignment

The compiler supports assignment of any pre-existing variables. Assignment can be done anywhere in the program.

Syntax

```
<variable declaration>;
<identifier> = <value>;
```

Example

```
INT a;
INT b=2;
a= 1;
b= b+a;
```

Print

The print statement is used to display the output in the display area.

Syntax

PRINT <expression>

- Prints an expression to console
- Always ends with a semicolon

Examples

```
PRINT 1;
Output : 1

PRINT 1+2+3/3*2;
Output : 5

PRINT 1.2;
Output : 1.2

PRINT 3.0/2.0;
Output : 1.5

PRINT ‘a’;
Output : a

PRINT “Hello Welcome”;
Output : Hello Welcome

PRINT true;
Output : true

PRINT 1<2;
Output : true

PRINT (1<2) AND (2<3);
Output : true
```

IF ELSE Statement

IF ELSE statements can take the following forms:

- Syntax : IF <expression> {<multiple statements>};
- Syntax : IF <expression> {<multiple statements>} ELSEIF <expression> {<multiple statements>};
- Syntax : IF expression {<multiple statements>} ELSE {<multiple statements>};
- Syntax : IF expression {<multiple statements>} ELSEIF <expression> {<multiple statements>} ELSE {<multiple statements>};

All kinds of IF statements should end with a semicolon. All the blocks (IF, ELSEIF, ELSE) should be inside braces/curly brackets. All the expressions can be within or without the parentheses. Nested if else statements are also supported following the same rules.

Example

A nested if else statement, which shows various kinds of if else statements

```
IF true {
    PRINT "INSIDE IF";

    IF false {
        PRINT "INSIDE NESTED  IF IF";
    }

    ELSEIF true {
        PRINT "INSIDE NESTED  IF IFELSE";

        IF false {
            PRINT "INSIDE NESTED  IF ELSEIF IF";
        }

        ELSEIF true {
            PRINT "INSIDE NESTED  IF ELSEIF ELSEIF";
        };
    };
}

ELSEIF true{
    PRINT "INSIDE ELSEIF";
}
```

```
ELSE{
    PRINT "HELLO FROM THE ELSE WORLD";
};
```

The above program will give the following output:

```
INSIDE IF
INSIDE NESTED  IF IFELSE
INSIDE NESTED  IF ELSEIF ELSEIF
```

WHILE Statement

The WHILE statement checks the value of the expression and if it is true, executes all the statements inside the braces/curly brackets. Once it executes the statements once, it again checks the value of the expression and executes the statements once again and so on.

Syntax

```
WHILE <expr> {
     <Statement list>
};
```

Example

```
     Int a = 0;
     WHILE a<5 {
          PRINT a;
          a = a +1;
     };
```

Output:

```
0
1
2
3
4
```

DO WHILE Statement

The DO WHILE first executes the statements in the braces/curly brackets once. Then it checks the value of the expression and if it is true, executes all the statements inside the braces/curly brackets again. Then it again checks the value of the expression and if true executes the statements once again and so on.

Thus, the statements inside the braces are executed at least once.

Syntax

```
DO {
      <Statement list>
} WHILE <expr>;

Examples of DO WHILE statement:
INT c = 0;
DO {
      PRINT c;
      c = c+1;
} WHILE c < 2;
```

Output:
0
1
2

```
INT c = 0;
DO {
      PRINT c;
} WHILE c!=0;
```

Output:
0

Conversion to Prefix

The compiler also translates the program written in the above given syntax, which is in the infix form, to the prefix format.
The compiler writes the prefix form of the program in the ---PREFIX--- section of the output console. The prefix of the program is the collection of the prefix form of the statements in the program. If any statement has an expression, the expression is also converted to the prefix form.
The prefix output of each statement is shown in a new line. The prefix output is indented except for the nested statements.

Example

Infix : `PRINT 1 + 2;`
Prefix : `PRINT + 1 2;`

Infix : `PRINT 1 + 2 + 3 / 4 * 5 + 6 - 7;`
Prefix : `PRINT - + + + 1 2 * / 3 4 5 6 7;`

Infix : `PRINT 1 + 2 + 3 / 4 * (5 + 6) - 7;`
Prefix : `PRINT - + + 1 2 * / 3 4 + 5 6 7;`

Infix : `PRINT "Hello" + " " +"Welcome";`
Prefix : `PRINT + + "Hello" " " "Welcome";`

A more complex example of the conversion from infix to prefix:

Infix:

```
INT a = 1;
FLOAT b = 2.3;
CHAR c = 'c';
STRING d = "Hello Everyone";
BOOLEAN e = true;
a =2;

IF true {
    INT f = 2;
    FLOAT g = 2.0;
    PRINT "Hello everyone from the IF statement";
 }
ELSEIF true {
```

```
    INT h = 3;
    FLOAT i = 2.1;
}
ELSE {
    CHAR j = 'j';
    PRINT "HELLO everyone from the ELSE statement";
};

WHILE a<5{
    PRINT "INSIDE WHILE STATEMENT";
    PRINT a;
    a = a+1;
};

DO {
    PRINT "INSIDE DO WHILE STATEMENT";
    PRINT a;
    a = a+2;
}WHILE a<10;
```

Prefix:

```
= INT a 1;
= FLOAT b 2.3;
= CHAR c 'c';
= STRING d "Hello Everyone";
= BOOLEAN e true;
= a 2;
IF true {
     = INT f 2;
     = FLOAT g 2.0;
     PRINT "Hello everyone from the IF statement";
}true {
     = INT h 3;
     = FLOAT i 2.1;
}{
     = CHAR j 'j';
     PRINT "HELLO everyone from the ELSE statement";
};
WHILE < a 5 {
     PRINT "INSIDE WHILE STATEMENT";
     PRINT a;
     = a + a 1;
}
```

```
DO WHILE < a 10 {
     PRINT "INSIDE DO WHILE STATEMENT";
     PRINT a;
     = a + a 2;
}
```