"Another" Language (Compiles to C or C++)

Builtin Data types:

int Same definition as ***int*** in C/C++

arr Array of zero or more ***int***'s. The size of the arr is dynamic and increases as needed.

Constants/Literals:

int	-- as for ***int*** in C	Example: 4, 1055, -23, 0
arr	-- [int_1, int_2, …, int_n]	Example: [1, 2, 3], [-5, 0, 4, -1], [2], []

Variable definition:

int i; -- An int variable (just like in C/C++)

arr a; -- An arr variable (initially it is an empty array, having 0 elements)

Variable Naming:

A variable name must begin with a letter or '_' optionally followed by alphanumeric (letters or digits), up to 32 characters.

Operators (in decreasing precedence order):

(…) - Parenthesis (as in C)

@ - Dot-product

+ | **-** | ***** | **/** | - Function/Associativity/Precedence as in C

: - Indexing

= - Assignment (as in C)

Expressions:

int – Same as in C/C++ for ***int*** (but without **++** / **--** and without shortcuts **+=** ***=** etc.)

arr expressions:

<arr variable> = <arr constant>	Assign r-value values to l-value variable, erases previous value
<arr variable> = <arr variable>	Assign r-value arr to l-value variable, erases previous values
<arr variable> = <int>	Creates an array with a single value (the r-value), erases previous value
<arr varliable> = <expression>	Assign expression result to l-value variable, erases previous value
$<arr>^L$ +|-|*|/ $<arr>^R$	Combines 2 arr (variable or constant) to a result arr, by carrying out the operation on every arr index: $<arr>^E{}_i$ = $<arr>^L{}_i$ +|-|*|/ $<arr>^R{}_i$ A missing index is regarded 0
$<arr>^L$ @ $<arr>^R$	Result is an <int> dot-product of the 2 arr elements: <int>^{E} = $\sum_i < arr >^l{}_i * < arr >^r{}_i$
<arr>:<expression>	Resolves to the arr element at the index given by the <int expression>.

Pseudo definition of the language grammar:

<program> ➔ <block>

<block> ➔ **begin** <statement-list> **end**

<statement-list> ➔ one or more <statement>

<statement> ➔ <declarator>; | <assignment>; | <conditional> | <loop> | <print>;

<declarator> ➔ <type> <variable-list>

<assignment> ➔ <variable> = <expression>

<conditional> ➔ **if** (<cond>) **then** <block>

<loop> ➔ **while** (<cond>) **do** <block>

<print> ➔ **print** <expression -list>

<variable-list> ➔ one or more <variable> (comma separated)

<expression-list> ➔ one or more <expression> (comma separated)

<variable> ➔ <identifier>

<expression> ➔ mathematical expression between one or more <variable> <constant> <op>:

| <expression> <ops> <expression>

| <arr variable> @ <arr variable>

| (<expression>)

| <variable>

| <number>

<ops> ➔ '**+**' | '**-**' | '***' | '**/**' (same meaning as in C)

<cond> ➔ <expression> <rel-ops> <expression>

<rel-ops> ➔ '**>**' | '**<**' | '**>=**' | '**<=**' | '**!=**' | '**==**'

<identifier> ➔ <letter> followed by zero or more <letter> | <digit> (like in C)

<number> ➔ integer number

| arr constant (i.e., [int_1, int_2, …, int_n])

<type> ➔ int | arr

Example Program:

```
Begin
    int i, fAvg;
    arr fib;

    fib:0 = 0;
    fib:1 = 1;
    i=2;
    print 0, 0                          //🡐print Fibonacci 0
    print 1, 1                          //🡐print Fibonacci 1
    while (i<16) do
        begin
            fib:i = fib:i-2 + fib:i-1
            print i, fib:i              //🡐print Fibonacci numbers at 2..15
            i=i+1
        end
    print fib@[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]      //🡐print Fibonacci sum of indexes 0..15
    fAvg = fib@[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] /16
    print fAvg                                                       //🡐print Fibonacci average of first 16 values

    i=2;
    while (i<16) do
        begin
            if (fib:I > fAvg) then begin
                print i, fib:i
            end
            i=i+1
        end
end
```

Expected Output

```
0, 0
1, 1
2, 1
3, 2
4, 3
5, 5
6, 8
7, 13
8, 21
9, 34
10, 55
11, 89
12, 144
13, 233
14, 377
15, 610
1596
99
12, 144
13, 233
14, 377
15, 610
```

Implement:

A compiler from “Another” Language to ‘C’ or ‘C++’ language

Submit Materials:

1. LEX file
2. YACC file
3. Make to generate compiler (Specify for what environment: PC/MAC/Linux)
4. The compiler executable
5. An example source file in “Another” Language
6. The results C/C++ file after compiling the example file
7. The executable after compiling the C/C++ of the example file
8. A short video showing the make process of the compiler, compiling the example to ‘C’ source, compiling the ‘C’ source to executable and running the executable.

Zip all files in an archive and upload to moodle