WordLang (compiles to C or C++)

Builtin Data-types:

char - Is a single ASCII character. It can be Empty (no character)

int - Like int in C/C++

word - Is single word, i.e. 0 or more characters with no white-space

sentence - Is a sentence, containing 0 or more (blank-seperated) words possibly with whitespaces and terminated with a newline ('\n');

Constants/Literals:

char - 'x' (as in C/C++)

Empty: '\0'

word - "xxx.." (as a string literal is in C/C++).

Empty: ""

sentence - ^xxxx xxxx xxxx^ (the \n is implied)

Empty: ^^ (only has \n)

int - integer number (as in C/C++)

Variable definition:

<type> <variable name list>; for example: word w12;

sentence s, s1;

int i, j;

Variable Naming:

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

Operators (in increasing precedence order):

Operator	Function	Priority (low to High)
=	Assignment: assign to a variable (as in C/C++)	4, right-left
-	decrement: remove character from word or sentence or remove word from sentence. Subtract for int	3, left-right
+	Addition for int	
#	Concatenate: concatenate characters to form a word; concatenate words to form a sentence	3, left-right

:	Indexing: extract a letter from a word; extract a word from a sentence	2, left-right
(…)	Parenthesis (as in C/C++)	1

Expressions:

	Evaluation
<lval> # <rval>	Concatenate <lval> and <rval>. Result in the higher level
<lval> - <rval>	Remove first instance of <rval> from <lval>, if possible. Result in <lval>'s level
<val>:<int>	Element at position <int>. <val> is a <char> - Only index 0 exists. Result is <char> <val> is <word> - <char> at index <int>. Result is <char> <val> is <sentence> - <word> at index <int>. Results is <word> If <int> is negative, it refers to a measure from the end of <val>. I.e., <val>:-1 is the last element in <val> If <int> larger than largest element, return Empty-value
(<expression>)	Calculate <expression> first.
Any combination/repeat of elements above, as in C/C++ For example: char c1; word w1; sentence s1; s1 # w1 – w1:2 # c1	

Conditions

<val>	True if <val> not Empty. False if <val> is Empty. False if int val is 0, true otherwise.
<lval> { < | >| <= | >= } <rval>	Lexicographic compare between <lval> and <rval>
<lval> == <rval>	True if <lval> equals <rval> exactly. False otherwise.
! <Condition>	Negates <Condition>

Special

Comment	//***<Any characters>***\n

Commands and Blocks

<expression>;	Sentence

{
<expression>;
<expression>;
:
<expression>;
}
	Sentence Block

Control

if (<condition>) <Sentence> | <Block>	Execute <Sentence> or <Block> if <condition> is true.
if (<condition>) <Sentence$_t$> | <Block$_t$> else <Sentence$_f$> | <Block$_f$>	Execute <Sentence$_t$> or <Block$_t$> if <condition> is true. Otherwise, execute <Sentence$_f$> or <Block$_f$> instead
while (<condition) <Sentence> | <Block>	Execute <Sentence> or <Block> repeatedly, while <condition> is true.
loop (<int>) <Sentence> | <Block>	Execute <Sentence> or <Block> repeatedly, <int> times

Input/Output

input <prompt-value> <var>;	Output the <prompt-value> and input reply into variable <var>
output <expression>;	Evaluate and Output <expression>

Examples:

“Hello”#”World” ➔ “HelloWorld”

^Hello^#”World” ➔ ^Hello World^

“Hello”#”World”:1 ➔ “Helloo”

(“Hello”#”World”):7 ➔ ‘r’

(“Hello”#”World”):-1 ➔ ‘d’

(^Hello^#”World”):1 ➔ “World”

“Hello”#”World”-‘l’ ➔ “HeloWorld”

“Hello”#(”World”-‘l’) ➔ “HelloWord”

sentence s1;

s1 = ^We live in a^;

output s1#^World^ ➔ Outputs: We live in a World

s1 = s1#”wonderful”#”world”#’!’

output s1 ➔ Outputs: We live in a wonderful world!

output s1-“wonderful” ➔ Outputs: We live in a world!

output s1-‘w’ ➔ Outputs: We live in a onderful world!

Example Program

```
word w1;
sentence s1, adverb;
loop (3)
   adverb = adverb # “very”;
w1 = “interesting”;
s1 = ^Compilers are a^ # adverb # w1;
output s1 # “topic”;                    //Outputs: Compilers are a very very very interesting topic

word object, article;
char notVowel;
input ^What are you holding?^  object;
notVowel = object:0;
notVowel = notVowel – ‘a’ – ‘e’ – ‘i’ – ‘o’ – ‘u’;
s1 = ^You are holding^;
if (notVowel) article=”a”;
else article=”an”;
output s1 # article # object;       //Outputs: You are holding {a | an} <object>
                                                //for example:  You are holding an apple
                                                //                    You are holding a hammer

word last;          //Flip Sentence (works only if words are distinct)
last = s1:-1;
while (s1:0 != last) {
   w1 = s1:0;
   s1 = (s1 - w1) # w1;
}
output s1;                              //Outputs: <object> {a | an}  holding are You

char b, e, palindrome;                  //Find Palindrome (works only if no more than 2 equal chars)
word savWord;
palindrome = ‘Y’;
input ^Enter a word:^ w1;           //For example: racecar
savWord = word;
while (w1) {
   b = w1:0;
   e = w1:-1;
   if (palindrome == ‘Y’)
      if (e != b)
         palindrome = ‘N’;
   w1 = w1 – b – e;
}
If (palindrome == ‘Y’)
   output savWord # ^is a palindrome word^;     //In example: racecar is a palindrome word
else
   output savWord # ^is not a palindrome^;
```

```
// Sort words in s1. For example: ^apple an holding are you^
int i, j, count;
count=0;
while (s1:count)
        count = count + 1;
sentence s;
while (count>1) {
  i = 0;
  loop (count-1) {
    if (s1:i > s1:(i+1)) {
      s = ^^;
      j=0;
      loop(i) {
        s = s # s1:j;
        j = j + 1;
      }
      s = s # s1:(i+1);
      s = s # s1:i;
      j = j + 2;
      loop (count-j) {
        s = s # s1:j;
        j = j + 1;
      }
      s1 = s;
    }
    i = i +1;
  }
  count = count-1;
}
output s1;        //In example: ^an apple are holding You^
```

Implement:

A compiler from “WordLang” 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. The example source file in “WordLang” Language
6. The results C/C++ file after compiling the example file
7. The executable after compiling the C/C++ of the example file compilation
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