CIS*4650 (Winter 2020) Compilers

Specification for the C- Language

C- is a simplified C programming language with some advanced features removed so that we can build a working compiler within one semester time. In the following, we describe the lexical conventions, the syntactic rules, and the semantic requirements of the C- language. We also provide several sample C- programs that can be used for testing and development.

Lexical Conventions

(1) The C- language contains the following keywords: **`else`**, **`if`**, **`int`**, **`return`**, **`void`**, **`while`**. All of them are reserved and must be written in lowercase.

(2) Special symbols are the following:

```
+  -  *  /  <  <=  >  >=  ==  !=  =  ;  ,  (  )  [  ]  {  }
```

Other tokens are **`ID`** and **`NUM`** defined by the following regular expressions:

```
ID = [_a-zA-Z][_a-zA-Z0-9]*

NUM = [0-9]+
```

(3) White space consists of blanks, tabs, and newlines. White space is ignored expect that it must be used to separate **`ID`**'s, **`NUM`**'s, and keywords.

(4) Comments are surrounded by the usual C style notations **`/*`** … **`*/`**. They can be placed anywhere white space can appear (e.g., comments cannot be placed within tokens) and may include more than one line. Comments may not be nested.

Syntactic Rules in BNF

```
(1) <program> -> <declaration-list>
(2) <declaration-list> -> <declaration-list> <declaration>
                        | <declaration>
(3) <declaration> -> <var-declaration> | <fun-declaration>
(4) <var-declaration> -> <type-specifier> ID ;
                       |  <type-specifier> ID [ NUM ] ;
(5) <type-specifier> -> int | void
(6) <fun-declaration> -> <type-specifier> ID ( <params> ) <compound-
    stmt>
(7) <params> -> <param-list> | void
```

```
(8) <param-list> -> <param-list> , <param> | <param>
(9) <param> -> <type-specifier> ID | <type-specifier> ID [ ]
(10) <compound-stmt> -> { <local-declarations> <statement-list> }
(11) <local-declarations> -> <local-declarations> <var-declaration>
                           |
(12) <statement-list> -> <statement-list> <statement> |
(13) <statement> -> <expression-stmt> | <compound-stmt>
                  | <selection-stmt> | <iteration-stmt>
                  | <return-stmt>
(14) <expression-stmt> -> <expression> ; | ;
(15) <selection-stmt> -> if ( <expression> ) <statement>
                   |  if ( <expression> ) <statement> else <statement>
(16) <iteration-stmt> -> while ( <expression> ) <statement>
(17) <return-stmt> -> return <expression> ; | return ;
(18) <expression> -> <var> = <expression> | <simple-expression>
(19) <var> -> ID | ID [ <expression> ]
(20) <simple-expression> -> <additive-expression> <relop> <additive-
   expression> | <additive-expression>
(21) <relop> -> <= | < | > | >= | == | !=
(22) <additive-expression> -> <additive-expression> <addop> <term>
                      | <term>
(23) <addop> -> + | -
(24) <term> -> <term> <mulop> <factor> | <factor>
(25) <mulop> -> * | /
(26) <factor> -> ( <expression> ) | <var> | <call> | NUM
(27) <call> -> ID ( <args> )
(28) <args> -> <arg-list> |
(29) <arg-list> -> <arg-list> , <expression> | <expression>
```

Semantic Requirements

During the semantic analysis, we should enforce that all variables and functions are defined before they are used and the last declaration should be "void main(void)" function. For parameter passing, we will do pass-by-value for integer parameters and pass-by-reference for array parameters. Functions can be recursive. For input and output purposes, we assume two predefined functions in the global environment with the following interfaces: "**`int input(void) { ...}`**" and "**`void output(int x) { … }`**".

Sample C- Programs

(1)
```
/* A program to compute the factorial value of an integer */
void main(void) {
  int x; int fac;
  x = input();
  fac  = 1;
  while (x > 1) {
     fac = fac * x;
     x = x - 1;
  }
  output(fac);
}
```

(2)
```
/* A program to perform Euclid's algorithm to
   compute the greatest common divisor */

int gcd(int u, int v) {
   if (v == 0) return u;
   else return gcd(v, u - u/v*v);
   /* note that u - u/v*v = u mod v */
}

void main(void) {
   int x; int y;
   x = input(); y = input();
   output(gcd(x, y));
}
```

(3)
```
/* A program to perform selection sort on an array of
   10 integers */

int x[10];

int minloc(int a[], int low, int high) {
   int i; int x; int k;
   k = low;
   x = a[low];
   i = low + 1;
   while (i < high) {
      if (a[i] < x) {
         x = a[i];
         k = i;
      }
      i = i + 1;
```

```
   }
   return k;
}
void sort(int a[], int low, int high) {
   int i; int k;
   i = low;
   while (i < high - 1) {
      int t;
      k = minloc(a, i, high);
      t = a[k];
      a[k] = a[i];
      a[i] = t;
      i = i + 1;
   }
}

void main(void) {
   int i;
   i = 0;
   while (i < 10) {
      x[i] = input();
      i = i + 1;
   }
   sort(x, 0, 10);
   i = 0;
   while (i < 10) {
      output(x[i]);
      i = i + 1;
   }
}
```