CIS 4650 Winter 2026

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. 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 to test its implementation process.

Lexical Conventions

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

(2) Special symbols are the following:

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

Other tokens are `ID, NUM,` and `TRUTH`, defined by the following regular expressions

```
   ID = [_a-zA-Z][_a-zA-Z0-9]*
  NUM = [0-9]+
TRUTH = false | true
```

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

(4) The only comments are multiline comments, as they appear in C. In other words, comments look like `/* ... */`. They can be placed anywhere white space can appear; comments cannot be placed within tokens. Comments also may not be nested, i.e., you may not have `*/` in a comment.

Syntactic Rules in BNF

Listed below are the recommended syntactic rules for the C- language. Some rules as in (12) and (13) contain an empty clause, which corresponds to an ϵ-production in the grammar. In addition, some clauses are shown in red, indicating that instances of **NilExp** are likely needed to build the corresponding nodes of their syntax trees in the implementation of a parser for our compiler project.

$$
\begin{aligned}
\mathit{program} &\rightarrow \mathit{declaration\text{-}list} && (1)\\
\mathit{declaration\text{-}list} &\rightarrow \mathit{declaration\text{-}list}\ \ \mathit{declaration} \mid \mathit{declaration} && (2)\\
\mathit{declaration} &\rightarrow \mathit{var\text{-}declaration} \mid \mathit{fun\text{-}declaration} \mid \mathit{fun\text{-}prototype} && (3)\\
\mathit{var\text{-}declaration} &\rightarrow \mathit{type}\ \texttt{ID}\ \texttt{;} \mid \mathit{type}\ \texttt{ID}\ \texttt{[}\ \texttt{NUM}\ \texttt{]}\ \texttt{;} && (4)\\
\mathit{type} &\rightarrow \texttt{bool} \mid \texttt{int} \mid \texttt{void} && (5)\\
\mathit{fun\text{-}declaration} &\rightarrow \mathit{type}\ \texttt{ID}\ \texttt{(}\ \mathit{params}\ \texttt{)}\ \mathit{compound\text{-}stmt} && (6)\\
\mathit{fun\text{-}prototype} &\rightarrow \mathit{type}\ \texttt{ID}\ \texttt{(}\ \mathit{params}\ \texttt{)}\ \texttt{;} && (7)\\
\mathit{params} &\rightarrow \mathit{param\text{-}list} \mid \texttt{void} && (8)\\
\mathit{param\text{-}list} &\rightarrow \mathit{param\text{-}list}\ \texttt{,}\ \mathit{param} \mid \mathit{param} && (9)\\
\mathit{param} &\rightarrow \mathit{type}\ \texttt{ID} \mid \mathit{type}\ \texttt{ID}\ \texttt{[}\ \texttt{]} && (10)\\
\mathit{compound\text{-}stmt} &\rightarrow \texttt{\{}\ \mathit{local\text{-}declarations}\ \ \mathit{stmt\text{-}list}\ \texttt{\}} && (11)\\
\mathit{local\text{-}declarations} &\rightarrow \mathit{local\text{-}declarations}\ \ \mathit{var\text{-}declaration} \mid && (12)\\
\mathit{stmt\text{-}list} &\rightarrow \mathit{stmt\text{-}list}\ \ \mathit{stmt} \mid && (13)\\
\mathit{stmt} &\rightarrow \mathit{expression\text{-}stmt} \mid \mathit{compound\text{-}stmt} \mid \mathit{selection\text{-}stmt} \mid && (14)\\
&\quad \mathit{iteration\text{-}stmt} \mid \mathit{return\text{-}stmt}\\
\mathit{expression\text{-}stmt} &\rightarrow \mathit{expression}\ \texttt{;} \mid \texttt{;} && (15)\\
\mathit{selection\text{-}stmt} &\rightarrow \texttt{if}\ \texttt{(}\ \mathit{expression}\ \texttt{)}\ \mathit{stmt} \mid \texttt{if}\ \texttt{(}\ \mathit{expression}\ \texttt{)}\ \mathit{stmt}\ \texttt{else}\ \mathit{stmt} && (16)\\
\mathit{iteration\text{-}stmt} &\rightarrow \texttt{while}\ \texttt{(}\ \mathit{expression}\ \texttt{)}\ \mathit{stmt} && (17)\\
\mathit{return\text{-}stmt} &\rightarrow \texttt{return}\ \mathit{expression}\ \texttt{;} \mid \texttt{return}\ \texttt{;} && (18)\\
\mathit{expression} &\rightarrow \mathit{var}\ \texttt{=}\ \mathit{expression} \mid \mathit{obool\text{-}expression} && (19)\\
\mathit{var} &\rightarrow \texttt{ID} \mid \texttt{ID}\ \texttt{[}\ \mathit{expression}\ \texttt{]} && (20)\\
\mathit{obool\text{-}expression} &\rightarrow \mathit{obool\text{-}expression}\ \texttt{||}\ \mathit{abool\text{-}expression} \mid \mathit{abool\text{-}expression} && (21)\\
\mathit{abool\text{-}expression} &\rightarrow \mathit{abool\text{-}expression}\ \texttt{\&\&}\ \mathit{ubool\text{-}expression} \mid \mathit{ubool\text{-}expression} && (22)\\
\mathit{ubool\text{-}expression} &\rightarrow \texttt{!}\ \ \mathit{ubool\text{-}expression} \mid \mathit{simple\text{-}expression} && (23)\\
\mathit{simple\text{-}expression} &\rightarrow \mathit{additive\text{-}expression}\ \ \mathit{relop}\ \ \mathit{additive\text{-}expression} \mid && (24)\\
&\quad \mathit{additive\text{-}expression}\\
\mathit{relop} &\rightarrow \texttt{<=} \mid \texttt{<} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{==} \mid \texttt{!=} && (25)\\
\mathit{additive\text{-}expression} &\rightarrow \mathit{additive\text{-}expression}\ \ \mathit{addop}\ \ \mathit{term} \mid \mathit{term} && (26)\\
\mathit{addop} &\rightarrow \texttt{+} \mid \texttt{-} && (27)\\
\mathit{term} &\rightarrow \mathit{term}\ \ \mathit{mulop}\ \ \mathit{signed\text{-}factor} \mid \mathit{signed\text{-}factor} && (28)\\
\mathit{mulop} &\rightarrow \texttt{*} \mid \texttt{/} && (29)\\
\mathit{signed\text{-}factor} &\rightarrow \texttt{-}\ \mathit{signed\text{-}factor} \mid \mathit{factor} && (30)\\
\mathit{factor} &\rightarrow \texttt{(}\ \mathit{expression}\ \texttt{)} \mid \mathit{var} \mid \mathit{call} \mid \texttt{NUM} \mid \texttt{TRUTH} && (31)\\
\mathit{call} &\rightarrow \texttt{ID}\ \texttt{(}\ \mathit{args}\ \texttt{)} && (32)\\
\mathit{args} &\rightarrow \mathit{args\text{-}list} \mid && (33)\\
\mathit{args\text{-}list} &\rightarrow \mathit{args\text{-}list}\ \ \texttt{,}\ \ \mathit{expression} \mid \mathit{expression} && (34)
\end{aligned}
$$

Semantic Requirements

During the semantic analysis, we should enforce that all variables and functions are defined before they are used in a program and the last declaration should be the "`void main(void)`" function. For parameter passing, we will do pass-by-value for integer and boolean parameters and pass-by-reference for array parameters. Functions can be recursive, including mutually 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) {...}`". Note that "`void`" can't be used to declare variables such as "`void x`" and "`void y[10]`" since it is only used to specify the return or parameter type of a function, as shown in the interfaces for "`input`" and "`output`" above. Also note that we will follow the common scoping rules for variable definitions and accesses in that an inner scope can see the definitions in the outer scopes in the order of inside-out direction, but not the other way around. Finally, for simplicity, functions are distinguished by their names only so that if "`void foo()`" is already defined, "`void foo(int x)`" is considered as a redefinition.

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 factorial program that uses a Boolean variable */
void main(void) {
    int x;
    int fac;
    bool test;
    x = input();
    fac = 1;
    test = x > 1;
    while (test) {
        fac = fac * x;
        x = x - 1;
        test = x > 1;
    }
    output(fac);
}
```

(3)
```
/* 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));
}
```

(4)
```
/* 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;
    }
}
```

(5)
```
/* A program that contain mutual recursions */
int x;
int y;
void g(int n); /* prototype for a function */
void f(int n) {
    g(n);
    y = y - 1;
}
void g(int m) {
    m = m - 1;
    if (m > 0) {
        f(m);
        y = y - 1;
        g(m);
    }
}
int main(void) {
    x = input();
    y = input();
    g(x);
    output(x);
    output(y);
}
```