C-- Language Description

C-- is a subset of the ANSI C language. In this course, we are going to implement a C-- compiler that generates ARM/V8 code. Although C-- small language, the data structures and control constructs supported in it appear in many widely used programming languages. Since C -- is a subset of ANSI C, we do not give a complete definition of it, instead, students are referred to the standard C reference manual (e.g. "C – A Reference Manual", by Samuel P. Harbison and Guy L. Steele JR., published by Prentice Hall). In this document, we only give the selected features supported in C --.

0. Tokens and Comments

C-- programs contain the following tokens:

1) **Reserved words**:
return
typedef
if
else
int
float
for
void
while

2) **Constants**:
   a. **Integer literals:** Integer literals consist of one or more digits.
   b. **Floating point literals:** The same as defined in ANSI C.[1]
   c. **String literals:** String literal consist of a sequence of characters between double quotes (i.e. " and ").
3) **Identifiers:** Identifiers are strings of one or more letters, digits and underscore characters. They must begin with a letter.
4) **Symbols and operators:**
   Arithmetic operators:
   +, -, /, *

   Relational operators
   <, >, >=, <=, !=, ==

   Logical operators
   ||, && and !

   Assignment operator
   =
5) **Separators:**
   { left brace
   } right brace
   [left bracket

[1] The floating point literal definition can include integer literals. I suggest that you define the integer literal RE before the floating point literal RE so that integer numbers will be correctly recognized.

```
]     right bracket
(     left parenthesis
)     right parenthesis
;     semicolon
,     comma
.     period or dot
```

C-- supports standard C comments. Anything between /* and */ is a comment and should be ignored by the compiler. C-- does not support ***nested comments***.

1. Program Structures

A C-- program has a single main function and a number of (possibly zero) functions that are referenced from within the main program. A generic C-- program looks like the following:

```
<type> function_1 (<parameter list>)
{
        statement_blocks
}

<type> function_2 (<parameter list>)
{
        statement_blocks
}

.......

<type> function_n (<parameter list>)
{
        statement_blocks
}
```

One of the functions *must* be the main function.

The statement_blocks consists of a number of statements delimited by the "***;***" character and enclosed within the braces "***{***" "***}***". You may have declarations for local variables and executable statements in every statement_block. Within any statement_block, variable declarations precede any executable statements in that block. A statement can be an assignment statement, a function call, a control structure or a loop structure (which can in turn contain statements).

No macro control supports such as ***defines*** and ***includes*** in C--. You may assume C-- programs have already been preprocessed by a C macro preprocessor.

2. Expressions

The semantics for all operations are identical to that in the ANSI C language.

3. Control Statements

3.a Conditional Statements

The form of the If-then-Else construct is

```
if ( <expression> )
       statement_block
[ else
       statement_block ]
```

The square brackets "[" and "]" indicate that the else part is optional. The statement_block may again contain an if-then-else construct.

3.b Iterative Statements

The forms of **for** and **while** structures are given below:

```
for (<expr1>; <expr2> ; <expr3>)
 statement_block
```

Here, expr1, expr2 and expr3 have the same semantics as in the C language and are all optional.

```
while (<expression>)
{
 statement_block
}
```

3.c Other Statements

You are not required to support other C statements such as do-while, switch, break/continue, and goto's.

4. Data Types and Declarations

The primitive data types supported in C-- are **int** and **float**. High level types are arrays and structures. These can be extended in a limited sense by using the **typedef** construct. Enumeration types are not supported.

Any time when the storage for an array is allocated, the size of the array must be known. Hence the memory size of an array must be specified in declarations. For example, arrays can be declared in the following forms:

```
a[5], b[5][10], d1[100][100][1000]
```

However, it is possible to omit the size when dealing with a ***formal parameter*** to a function. This is because the name of the array parameter actually references the address of the array being passed. Since the formal parameter declaration does not cause any memory to be allocated, the size specification is not needed (Also because subscripts are not checked to lie within declared bounds). Within the parameter list of a function, the *first row dimension* may remain empty. For example,

```
Funtion_A (int a[], float b[][100])
```

Notice that in variable declarations, the following are NOT allowed:

```
int arr[];
int abc[size];
```

In the first case, the dimension is undefined. In the second case, the dimension is not defined at compile time[2].

Declarations might have either local or global scope. Declarations in a statement_block have scope limited to that block. Global declarations have scope throughout the entire program and are declared outside of any function (including the main function).

Structures and Unions are not required to be implemented in this semester. Each semester, we make some changes of the requirements. C— does not supports pointer arithmetic operations and explicit pointer operations such as *p, &a, or c->d.

Local declarations may contain **typedef** statements and variable declarations.

Only scalar variables may be initialized when they are declared. We do not support array initializations. We also do not support functions to return an array.

5. Library Functions

There are three library functions provided for handling I/O:

```
read()  – reads and returns an integer number;
fread() – reads and returns a floating point number;
write() – prints out an integer, a float or a character string
          (Characters enclosed in double-quotes).
```

These are system library functions. At the code generation phase, you simply generate such calls, and we will provide the libraries for you to link with.

An example of using read/write functions can be seen in the sample programs listed in section 6.

6. Sample C-- Programs

6.a A factorial function

This program computes the factorial of a number recursively. The factorial of a number is defined as *n! = n x(n-1) x .. x 1*

```
int fact (int n)
{
 if (n == 1 )
  {
   return n;
  }
  else
  {
   return (n*fact(n-1));
  }
```

[2] C—does not support #define directives.

```
}

int main()
{
 int n, result;
 write("Enter a number");
 n = read();

 if (n > 1)
 {
  result = fact(n);
 }
 else
 {
  result = 1;
 }

 write("The factorial is");
 write(result);
}
```

6.b Sum of 1..n

This illustrates the usage of a **for** construct

```
int main()
{
 int n, sum;
 int loopvar;
 write("What is n:");
 n = read();
 sum = 0;

 for (loopvar = 1; loopvar <= n; loopvar = loopvar + 1)
 {
  sum = sum + loopvar;
 }

 write("The sum is :");
 write(sum);
}
```