

Csci 5161 - Spring 2003

C-- Language Description

C-- is a subset of the ANSI C language. In this course, we are going to implement a compiler that translates C-- programs into MIPS code. We do away with some of the more difficult aspects in ANSI C such as the pointer types. The set of data types supported in C-- is also restricted to a subset of what C traditionally provides. The data structures and control constructs supported in C-- appear in most programming languages.

The following sections define the form and content of what might constitute a C-- program.

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 would look like following:

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

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

<type> function_3 (<parameter list>)
{
    statement_block
}
.....

<type> function_4 (<parameter list>)
{
    statement_block
}
```

where one of the functions *must* necessarily be the main function.

You can assume that the main function returns an integer and does not take any arguments, so it would be written as

```
int main ( )
{
    .....
}
```

The `statement_block` 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 control structure or a loop structure (which can in turn contain statements).

A C-- program does not contain any macro control lines such as defines and includes. You may assume C-- programs have been preprocessed by a C macro preprocessor.

2. Expressions

The following operators are supported in C--

Arithmetic operators:

+, -, /, *

Relational operators

<, >, >=, <=, !=, ==

Logical operators

||, && and !

Assignment operator

=

The semantics for all these operations is the same as in the ANSI C language. The operands for the operators are integer numbers, floating point numbers and defined variables. Variables must start with a letter and may contain any sequence of digits, alphabets and the underscore character. Although there is no limit to the size of the name, you can assume some reasonable value.

3. Loop Structures

The C-- language supports **for** and **while** loops. The form of these structures is 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 (<conditional>)
{
    statement_block
}
```

4. Control Structures

The C-- language provides nested control structures in the form of if-then-else constructs. The form of the nested construct is

```
if ( <conditional> )
{
    statement_block
}
```

```
[ else (<conditional>)
  {
    statement_block
  } ]
```

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

5. Data Types and Declarations

The primitive data types the language supports are only **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.

In regard to the arrays, the size must always be explicitly specified in declarations. For example, arrays can be declared in the following forms:

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

The only exception to this rule being parameter lists in function declarations, where the name 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. Therefore, within the parameter list of a function, the *first row dimension* may remain empty. For example,

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

In regular variable declarations, the following are NOT allowed in C--:

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

In the first case, the dimension is undefined. In the second case, the dimension is not defined at compile time¹.

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

Structures and Unions are supported, and they can be nested. We do not support pointer arithmetic operations nor explicit pointer operations such as `*p`, `&a`, `c->d`.

Local declarations may contain `typedef` statements and variable declarations. A scalar variable may be initialized when it is declared, but a structure field may not be initialized. We do not allow an array to be initialized.

We assume a struct never has a tag. To declare struct variables, we always use `"struct {... fields} var_name;"` or we use `"typedef struct { ... fields} new_type_name"` to declare a new type name first and then do `"new_type_name var_name, var_name, ..;"`.

¹ C—does not support `#define` directives.

We do not support functions returning "struct" or taking "struct" as a parameter. We also do not support functions to return an array.

6. Library Functions

There are three library functions provided for input and output:

- 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).

An example of using read/write functions can be seen in sample programs listed in section 7.

7. Sample Programs

A. Factorial

This program computes the factorial of a number recursively. The factorial of a number is defined as $n! = n \times (n-1) \times \dots \times 1$

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

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);
}
```

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);
}
```

8. Appendix

List of reserved words (you can build these into the lexical analyzer)

```
return
typedef
if
else
int
float
for
fread
read
struct
union
void
while
write
```