The University of Newcastle

School of Electrical Engineering and Computing

COMP3290 Compiler Design

Semester 2, 2018

The CD18 Programming Language
Version 1.0 – 2018-08-01

Introduction:

CD18 is a language with reminders of Ada, Pascal and C, designed to make use of features of the SM18 computer.

Lexical Structure:

CD18 is a free format language, whitespace terminates all tokens, except that string constants may contain space characters. There are no special character codes for tab, newline, etc.

Comments start with **/--** and are terminated by the end of the current line of source code. Key words are reserved words in CD18 (they cannot be used as identifiers). They are: **CD18 constants types is arrays main begin end array of func void const integer real boolean for repeat until if else input print printline return not and or xor true false.**

While identifiers are CaSe SeNSitivE in CD18, the keywords are not case sensitive, so *begin*, *Begin*, *BegiN*, *bEGIN* and *BEGIN* are all the same keyword (*begin*). By convention the CD18 keyword is upper case, the others are lower case – breaching this convention is considered bad programming practice but does not concern the compiler.

Identifiers follow standard rules – they must start with a letter, which can then be followed by any number of letters or digits (there is an effective limit on the length of an identifier as the length of a source code line).

Integer literals are a sequence of digits, which is of any length. There is an effective limit on an integer literal size of about 9×10^{18} (64 bits worth). Integer literals cannot be negative, so a sequence such as -4 is two tokens, *minus* and *4*.

String literals are any length and are delimited by "…" and may not go across a source line and may not contain " characters.

Other significant lexical items in CD18 are: semicolon (**;**) leftbracket (**[**) rightbracket (**]**) comma (**,**) leftparen (**(**) rightparen (**)**) equals (=) plus (+) minus (-) star (*****) slash (/) percent (**%**) carat (^) less (<) greater (>) exclamation (**!**) quote (**“**) colon (**:**) dot (**.**).

Some of these are combined to form operators such as: <=, >=, **!**=, ==, +=, -=, ***=,** /=. These composite operators may NOT contain embedded whitespace characters.

CD18 Language Syntax – Version 1.0

<program> ::= CD18 <id> <consts> <types> <arrays> <funcs> <mainbody>
<consts> ::= constants <initlist> | ε
<initlist> ::= <init> | <init> , <initlist>
<init> ::= <id> = <expr>
<types> ::= types <typelist> | ε
<arrays> ::= arrays <arrdecls> | ε
<funcs> ::= <func> <funcs> | ε
<mainbody> ::= main <slist> begin <stats> end CD18 <id>
<slist> ::= <sdecl> | <sdecl> , <slist>
<typelist> ::= <type> <typelist> | <type>
<type> ::= <structid> is <fields> end
<type> ::= <typeid> is array [<expr>] of <structid>
<fields> ::= <sdecl> | <sdecl> , <fields>
<sdecl> ::= <id> : <stype>
<arrdecls> ::= <arrdecl> | <arrdecl> , <arrdecls>
<arrdecl> ::= <id> : <typeid>
<func> ::= func <id> (<plist>) : <rtype> <funcbody>
<rtype> ::= <stype> | void
<plist> ::= <params> | ε
<params> ::= <param> , <params> | <param>
<param> ::= <sdecl> | <arrdecl> | const <arrdecl>
<funcbody> ::= <locals> begin <stats> end
<locals> ::= <dlist> | ε
<dlist> ::= <decl> | <decl> , <dlist>
<decl> ::= <sdecl> | <arrdecl>
<stype> ::= integer | real | boolean
<stats> ::= <stat> ; <stats> | <strstat> <stats> | <stat>; | <strstat>
<strstat> ::= <forstat> | <ifstat>
<stat> ::= <reptstat> | <asgnstat> | <iostat> | <callstat> | <returnstat>
<forstat> ::= for (<asgnlist> ; <bool>) <stats> end
<repstat> ::= repeat (<asgnlist>) <stats> until <bool>
<asgnlist> ::= <alist> | ε
<alist> ::= <asgnstat> | <asgnstat> , <alist>
<ifstat> ::= if (<bool>) <stats> end
<ifstat> ::= if (<bool>) <stats> else <stats> end
<asgnstat> ::= <var> <asgnop> <bool>
<asgnop> ::= = | += | -= | *= | /=
<iostat> ::= input <vlist> | print <prlist> | printline <prlist>
<callstat> ::= <id> (<elist>) | <id> ( )
<returnstat> ::= return | return <expr>
<vlist> ::= <var> , <vlist> | <var>
<var> ::= <id> | <id>[<expr>] . <id>
<elist> ::= <bool> , <elist> | <bool>
<bool> ::= <bool><logop> <rel> | <rel>
<rel> ::= not <expr> <relop> <expr> | <expr> <relop><expr> | <expr>
<logop> ::= and | or | xor
<relop> ::= == | != | > | <= | < | >=
<expr> ::= <expr> + <fact> | <expr> - <fact> | <fact>

<fact> ::= <fact> * <term> | <fact> / <term> | <fact> % <term> | <term>
<term> ::= <term> ^ <exponent> | <exponent>
<exponent> ::= <var> | <intlit> | <reallit> | <fncall> | true | false
<exponent> ::= (<bool>)
<fncall> ::= <id> (<elist>) | <id> ( )
<prlist> ::= <printitem> , <prlist> | <printitem>
<printitem> ::= <expr> | <string>

<id>, <structid>, <typeid> are all simply identifier tokens returned by the scanner.
<intlit>, <reallit> and <string> are also special tokens returned by the scanner.

Notes on CD18 Semantics

1. The identifiers at the beginning and end the program must match.
2. Any variable name must be declared before it is used, with the exception that function names may be used (called) before they are declared, thus allowing a group of functions to be mutually recursive.
3. Constant identifiers are of a type determined by the type of the expression used to set their values. It is common practice within CD18 for array type lengths to be declared as constant identifiers, but this is not necessary.
4. Constant identifiers are global definitions for the whole program, unless re-declared as variables in a function. Re-declaring constant identifier names within a function is not considered good programming practice, even though it is legal.
5. Type names for structures and array types are global to the whole program, unless re-declared within a procedure or function (which once again is considered bad practice).
6. Field names within structure types are completely separate from variable names in the program and so a field name may be used more than once in different structure types and may also be used as a variable name anywhere in the program. This is considered bad programming practice, but is allowed to facilitate re-use of legacy code.
7. Program level array identifiers are globally visible within functions unless they are re-declared within the parameter list or local variable list for the function.
8. A function call must have the same number of actual parameters as there are formal parameters in the function definition. The type of each actual parameter must match the type of its respective formal parameter in the function definition.
9. Identifier names may be re-declared in functions as either formal parameters or local variables. The original definition of the name disappears only during the definition of the function and then becomes visible again.
10. The environment for any function consists of all the actual parameters and all the local variables of that function, plus any program level arrays whose names have not been re-declared.
11. The environment for the main section of the program consists of the declared arrays, and the simple variables declared locally to the program main section.
12. Only functions that return *void* may be used as programming statements, all other function calls are evaluated within expressions.
13. Simple parameters are passed into functions by value, arrays are passed by reference. Array parameters declared *const* are passed by reference but are read-

only within the function, and may only be passed into another *const* parameter in a subsequent function call.

14. CD18 is a strongly typed language. A variable may only store a value with the same type as its declaration.
15. Expressions are also evaluated using strong typing, the one exception to this is the automatic type promotion of integer expressions to real expressions for numeric calculations.
16. CD18 directly reflects the SM18 architecture by adopting a special notion of *close enough to equals* for the relational testing of equality and inequality for real expressions. If two real expressions are within 0.000001 of each other then they are considered equal for the purpose of testing equality. In order to be considered unequal then they must be outside this range. Note that this does not carry over to the testing of <, >, <=, or >=, which are performed with exact expression values. This also follows the equivalent computations performed by the SM18 for these operations.
17. CD18 does not mandate that variables are initialized before they are used.
18. Whole of array copy is possible via a simple assignment statement, e.g. a2 = a1 provided a1 and a2 are of the same type (not simply the same structure).
19. As the CD18 language stands (version 1.0), whole of structure copying is not allowed, that is a2[2] = a1[1] is illegal. You might like to think about what would need to change (and where) in order to make operations like this legal, and also what scope these operations might have.

DB **v1.0 2018-08-02** (originally issued : *2018-08-02*)