The University of Newcastle
School of Information and Physical Sciences
COMP3290 Compiler Design
Semester 2, 2024

The CD24 Programming Language
Version 1.02-2024-07-29

Introduction

CD24 is a procedural language drawing on aspects from languages such as Ada, Pascal and C, and designed to make use of features of the SM24 computer – the SM24 computer will be provided as a simulator (see sm24_user_guide.pdf).

IMPORTANT: Note that there is a very good chance that this grammar will change as we discover edge cases, conflicts and ambiguities (there may even be an intentional ambiguity in there, to satisfy some of the learning outcomes), and thus this document will be clearly marked with a version number and a date. Changes away from v1.00 will be highlighted for clarity.

This document is v1.02-2024-07-29

Lexical Structure

1. CD24 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.
2. Keywords are reserved words in CD24 (i.e. they cannot be used as identifiers). They are: **`CD24 constants typedef def arraydef main begin end array of func void const int float bool for repeat until do while if else elif switch case default break input print printline return not and or xor true false.`**
3. Single-line comments start with **`/--`** and are terminated by the end of the current line of source code. Multi-line comments start with **`/**`** and are terminated by **`**/`** or the end of file.
4. While identifiers are CaSe SeNSitivE in CD24, the keywords are not case sensitive, so **`begin, Begin, BegiN, bEGIN`** and **`BEGIN`** are all the same keyword (**`begin`**). By convention keywords are lower case, except for **`CD24`** – breaching these conventions is considered bad programming practice but does not concern the compiler (thus **`cd24`** is valid).
5. 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).
6. 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 and real literals cannot be negative, so an input sequence such as **`-4`** is considered as two separate lexemes, *minus* and *4*.

7. String literals are any length and are delimited by `"…"` and may not go across a source line and may not contain `"` characters.
8. Other *significant lexical* items in CD24 are: comma (`,`) leftbracket (`[`) rightbracket (`]`) leftparen (`(`) rightparen (`)`) equals (`=`) plus (`+`) minus (`-`) star (`*`) slash (`/`) percent (`%`) carat (`^`) less (`<`) greater (`>`) colon (`:`) semicolon (`;`) dot (`.`).
9. Some of these are combined to form operators such as: `<=, >=, !=, ==, +=, -=, *=, /=`. These composite operators may NOT contain embedded whitespace characters.

CD24 Language Syntax – Version 1.02:

<program> ::= CD24 <id> <globals> <funcs> <mainbody>

<globals> ::= <consts> <types> <arrays>
<consts> ::= constants <initlist> | ε
<initlist> ::= <init> | <init> , <initlist>
<init> ::= <id> = <expr>

<types> ::= typedef <typelist> | ε
<typelist> ::= <type> <typelist> | <type>
<type> ::= <structid> def <fields> end
<type> ::= <typeid> def array [<expr>] of <structid> end
<fields> ::= <sdecl> | <sdecl> , <fields>

<arrays> ::= arraydef <arrdecls> | ε
<arrdecls> ::= <arrdecl> | <arrdecl> , <arrdecls>
<arrdecl> ::= <id> : <typeid>

<funcs> ::= <func> <funcs> | ε
<func> ::= func <id> (<plist>) : <rtype> <funcbody>
<rtype> ::= <stype> | void
<plist> ::= <params> | ε
<params> ::= <param> | <param> , <params>
<param> ::= <sdecl> | <arrdecl> | const <arrdecl>
<funcbody> ::= <locals> begin <stats> end
<locals> ::= <dlist> | ε
<dlist> ::= <decl> | <decl> , <dlist>
<decl> ::= <sdecl> | <arrdecl>

<mainbody> ::= main <slist> begin <stats> end CD24 <id>
<slist> ::= <sdecl> | <sdecl> , <slist>
<sdecl> ::= <id> : <stype> | <id> : <structid>

<stype> ::= int | float | bool

<stats> ::= <stat> ; <stats> | <strstat> <stats> | <stat>; | <strstat>
<strstat> ::= <forstat> | <ifstat> | <switchstat> | <dostat>
<stat> ::= <repstat> | <asgnstat> | <iostat> | <callstat> | <returnstat>

<forstat> ::= for (<asgnlist> ; <bool>) <stats> end
<repstat> ::= repeat (<asgnlist>) <stats> until <bool>
<dostat> ::= do <stats> while (<bool>) end
<asgnlist> ::= <alist> | ε
<alist> ::= <asgnstat> | <asgnstat> , <alist>

<ifstat> ::= if (<bool>) <stats> end
<ifstat> ::= if (<bool>) <stats> else <stats> end
<ifstat> ::= if (<bool>) <stats> elif (<bool>) <stats> end

<switchstat> ::= switch (<expr>) begin <caselist> end
<caselist> ::= case <expr> : <stats> break ; <caselist> | default : <stats>

<asgnstat> ::= <var> <asgnop> <bool>
<asgnop> :: == | += | -= | *= | /=

<iostat> ::= input <vlist> | print <prlist> | printline <prlist>

<callstat> ::= <id> (<elist>) | <id> ( )

<returnstat> ::= return void | return <expr>

<vlist> ::= <var> , <vlist> | <var>
<var> ::= <id> | <id>[<expr>] | <id>[<expr>] . <id>

<elist> ::= <bool> , <elist> | <bool>
<bool> ::= not <bool> | <bool><logop> <rel> | <rel>
<rel> ::= <expr> <relop><expr> | <expr>
<logop> ::= and | or | xor
<relop> ::= == | != | > | <= | < | >=

<expr> ::= <expr> + <term> | <expr> - <term> | <term>
<term> ::= <term> * <fact> | <term> / <fact> | <term> % <fact> | <fact>
<fact> ::= <fact> ^ <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 CD24 Syntax

1. A structure type exists for use as an independent variable or an array element type in an array type declaration.
2. An array may only be declared as an array of a particular structure, so arrays of simple types such as **int**, **float**, or **bool** are not allowed.
3. A structure type can have only primitive data type (int, float and bool) as fields. Any nested structure should be reported as a syntax error.

Notes on CD24 Semantics

1. The identifiers at the beginning and end of the program must match.
2. CD24 is a strongly typed language; a variable may only store a value with the same type as its declaration.
3. Any variable name must be declared before it is used, with the exception that function names may be used (i.e. *called*) before they are declared, thus allowing a group of functions to be mutually recursive.
4. Numeric variables are to be initialised to zero when they are declared.
5. Constant identifiers are of a type determined by the type of the expression used to set their values. It is common practice within CD24 for array type lengths to be declared as constant identifiers, but this is not necessary.
6. 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.
7. 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). Any type name cannot be re-used as a global array name.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. The environment for the main section of the program consists of the declared program level arrays, and the simple variables declared locally to the program main section.
14. Only functions that return *void* may be used as programming statements, all other function calls are evaluated within expressions.
15. 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.

16. 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.
17. CD24 directly reflects the SM24 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. These also follow the equivalent computations performed by the SM24 for these operations.
18. Whole of array copy is allowed via a simple assignment statement, e.g. *a2 = a1* provided a1 and a2 are of the same type (not simply the same structure). Whole of structure copying is also allowed, that is *a4[i] = a3[j]* is legal, but only if the two *arrays* are of the same type. One assignment operation may therefore perform a large data copy.
19. Whole of array and whole of structure output and input is also possible. So it is possible to perform a very large amount of data input or output via seemingly simple statements such as *input a1;* or *printline a2[2];*
20. Expression of Array size in array declaration: Can be an expression of only constants or integer literals: e.g. 2 * SIZE. Not meant to be a complete expression which could have variable or function call in it.

V 1.02 19-Aug-2024

Typo corrected in the following rule:

<stat> ::= <repstat> | <asgnstat> | <iostat> | <callstat> | <returnstat>

<reptstat> → <repstat>

V 1.01 29-Jul-2024.

V 1.01 29-Jul-2024 Changes:

'break' is added as a new keyword in Lexical structure Section 2.

The CD24 grammar rule for <caselist> is modified by adding the 'break' keyword.