The University of Newcastle
School of Information and Physical Sciences
COMP3290 Compiler Design
Semester 2, 2022

The CD22 Programming Language
Version 1.00-2022-07-18

CD22 PL Syntax – Version1.00 – with Syntax Tree Node Values

NPROG	<program>	::=	CD22 <id> <globals> <funcs> <mainbody>
NGLOB	<globals>	::=	<consts> <types> <arrays>
Special	<consts>	::=	constants <initlist> | ε
NILIST	<initlist>	::=	<init> , <initlist>
Special	<initlist>	::=	<init>
NINIT	<init>	::=	<id> = <expr>
Special	<types>	::=	types <typelist> | ε
Special	<arrays>	::=	arrays <arrdecls> | ε
NFUNCS	<funcs>	::=	<func> <funcs>
Special	<funcs>	::=	ε
NMAIN	<mainbody>	::=	main <slist> begin <stats> end CD22 <id>
NSDLST	<slist>	::=	<sdecl> , <slist>
Special	<slist>	::=	<sdecl>
NTYPEL	<typelist>	::=	<type> <typelist>
Special	<typelist>	::=	<type>
NRTYPE	<type>	::=	<structid> def <fields> end
NATYPE	<type>	::=	<typeid> def array [<expr>] of <structid> end
NFLIST	<fields>	::=	<sdecl> , <fields>
Special	<fields>	::=	<sdecl>
NSDECL	<sdecl>	::=	<id> : <stype>
NTDECL	<sdecl>	::=	<id> : <structid>
NALIST	<arrdecls>	::=	<arrdecl> , <arrdecls>
Special	<arrdecls>	::=	<arrdecl>
NARRD	<arrdecl>	::=	<id> : <typeid>
NFUND	<func>	::=	func <id> (<plist>) : <rtype> <funcbody>
Special	<rtype>	::=	<stype> | void
Special	<plist>	::=	<params> | ε
NPLIST	<params>	::=	<param> , <params>
Special	<params>	::=	<param>
NSIMP	<param>	::=	<sdecl>
NARRP	<param>	::=	<arrdecl>
NARRC	<param>	::=	const <arrdecl>
Special	<funcbody>	::=	<locals> begin <stats> end
Special	<locals>	::=	<dlist> | ε
NDLIST	<dlist>	::=	<decl> , <dlist>
Special	<dlist>	::=	<decl>
Special	<decl>	::=	<sdecl> | <arrdecl>
Special	<stype>	::=	int | float | bool
NSTATS	<stats>	::=	<stat> ; <stats> | <strstat> <stats>

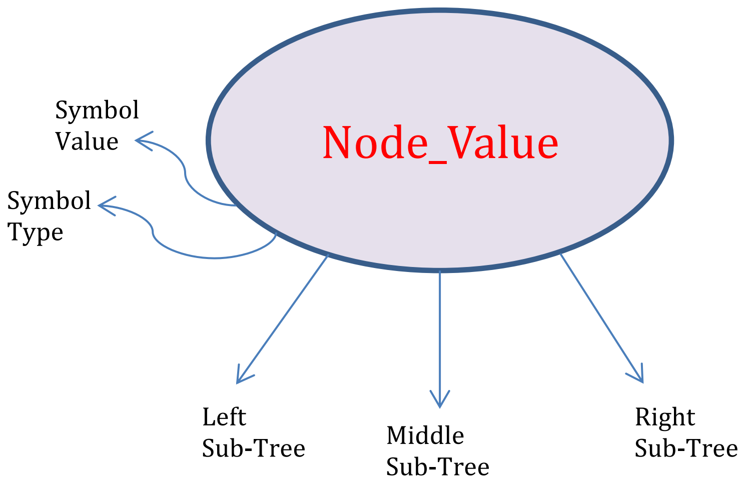
Special	<stats>	::= <stat>; | <strstat>
Special	<strstat>	::= <forstat> | <ifstat>
Special	<stat>	::= <reptstat> | <asgnstat> | <iostat>
Special	<stat>	::= <callstat> | <returnstat>
NFORL	<forstat>	::= for (<asgnlist> ; <bool>) <stats> end
NREPT	<repstat>	::= repeat (<asgnlist>) <stats> until <bool>
Special	<asgnlist>	::= <alist> | ε
NASGNS	<alist>	::= <asgnstat> , <alist>
Special	<alist>	::= <asgnstat>
NIFTH	<ifstat>	::= if (<bool>) <stats> end
NIFTE	<ifstat>	::= if (<bool>) <stats> else <stats> end
NIFEF	<ifstat>	::= if (<bool>) <stats> elif (<bool>) <stats> end
Special	<asgnstat>	::= <var> <asgnop> <bool>
NASGN	<asgnop>	::= =
NPLEQ	<asgnop>	::= +=
NMNEQ	<asgnop>	::= -=
NSTEA	<asgnop>	::= *=
NDVEQ	<asgnop>	::= /=
NINPUT	<iostat>	::= input <vlist>
NPRINT	<iostat>	::= print <prlist>
NPRLN	<iostat>	::= printline <prlist>
NCALL	<callstat>	::= <id> (<elist>) | <id> ( )
NRETN	<returnstat>	::= return void | return <expr>
NVLIST	<vlist>	::= <var> , <vlist>
Special	<vlist>	::= <var>
NSIMV	<var>	::= <id>
NARRV	<var>	::= <id>[<expr>] . <id>
NAELT	<var>	::= <id>[<expr>]
NEXPL	<elist>	::= <bool> , <elist>
Special	<elist>	::= <bool>
NBOOL	<bool>	::= <bool><logop> <rel>
Special	<bool>	::= <rel>
NNOT	<rel>	::= not <expr> <relop> <expr>
Special	<rel>	::= <expr> <relop><expr>
Special	<rel>	::= <expr>
NAND	<logop>	::= and
NOR	<logop>	::= or
NXOR	<logop>	::= xor
NEQL	<relop>	::= ==
NNEQ	<relop>	::= !=
NGRT	<relop>	::= >
NLSS	<relop>	::= <
NLEQ	<relop>	::= <=
NGEQ	<relop>	::= >=
NADD	<expr>	::= <expr> + <term>
NSUB	<expr>	::= <expr> - <term>
Special	<expr>	::= <term>
NMUL	<term>	::= <term> * <fact>
NDIV	<term>	::= <term> / <fact>
NMOD	<term>	::= <term> % <fact>

Special	<term>	::=	<fact>
NPOW	<fact>	::=	<fact> ^ <exponent>
Special	<fact>	::=	<exponent>
Special	<exponent>	::=	<var>
NILIT	<exponent>	::=	<intlit>
NFLIT	<exponent>	::=	<reallit>
Special	<exponent>	::=	<fncall>
NTRUE	<exponent>	::=	true
NFALS	<exponent>	::=	false
Special	<exponent>	::=	(<bool>)
NFCALL	<fncall>	::=	<id> (<elist>) | <id> ( )
NPRLST	<prlist>	::=	<printitem> , <prlist>
Special	<prlist>	::=	<printitem>
Special	<printitem>	::=	<expr>
NSTRG	<printitem>	::=	<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.

The Syntax Tree

The basic design of a Syntax Tree Node is to have a node that holds a node value (Nxxxx), and that this will also contain references to other (child) nodes in the syntax tree (say 3 of these) and a couple of references to Symbol Table records as follows:



It is possible to design polymorphic variants for each of the different Node values that can occur in the syntax tree. However, it can be just as easy to use a single Node type and leave references as null if they are not needed for a particular case. There will be significant discussion in class on many issues pertinent to the syntax tree.

Child Node assignments will be in the order as presented in the previously listed rules. If a rule contains *three children*, then work left to right; if a rule contains *two children*, use left and right, if a rule contains *one child*, use the left.

For example:

NTRAF	<trafficlight> ::= <id><red><yellow><green>
NTROG	<trogdor> ::= he was a <dragon><man>;
NSCHM	<schmoo> ::= <value> schmoo

```
     NTRAF(id)         NTROG            NSCHM
    /  |  \           /     \          /
   /   |   \         /       \        /
  r    y    g     dragon     man    value
```

The Parser

Version 1.00 of the grammar, as it stands, is not sufficient to write a Recursive Descent Parser for CD22, and it is certainly not LL(1). There are a number of places where rules need left-factoring and other places where left-recursion needs to be eliminated.

Version 1.00 18-July-2022.