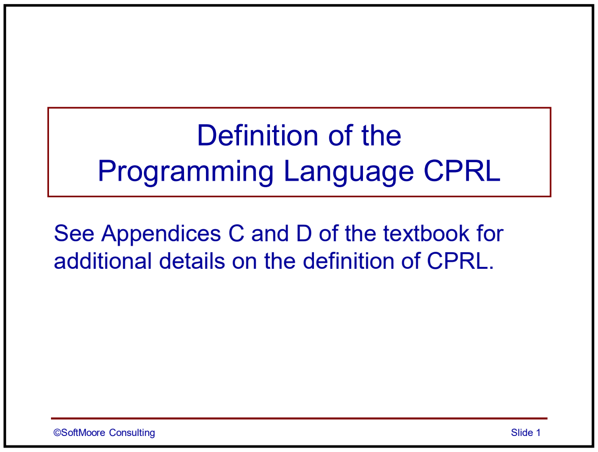
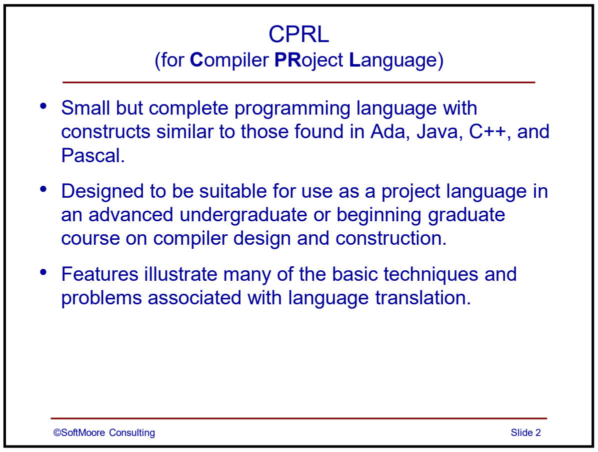


Definition of the Programming Language CPRL

See Appendices C and D of the textbook for additional details on the definition of CPRL.

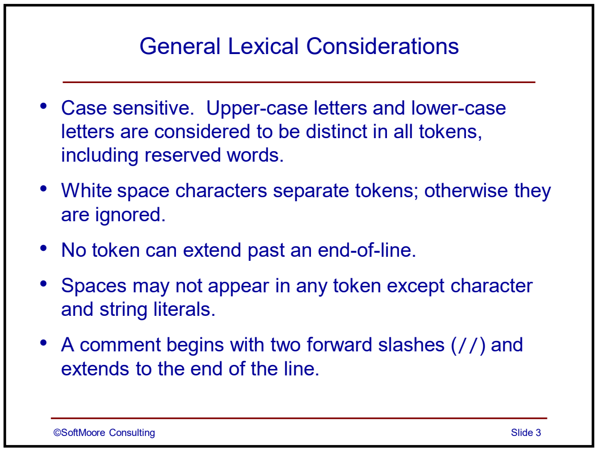
1



CPRL
(for **C**ompiler **PR**oject **L**anguage)

- Small but complete programming language with constructs similar to those found in Ada, Java, C++, and Pascal.
- Designed to be suitable for use as a project language in an advanced undergraduate or beginning graduate course on compiler design and construction.
- Features illustrate many of the basic techniques and problems associated with language translation.

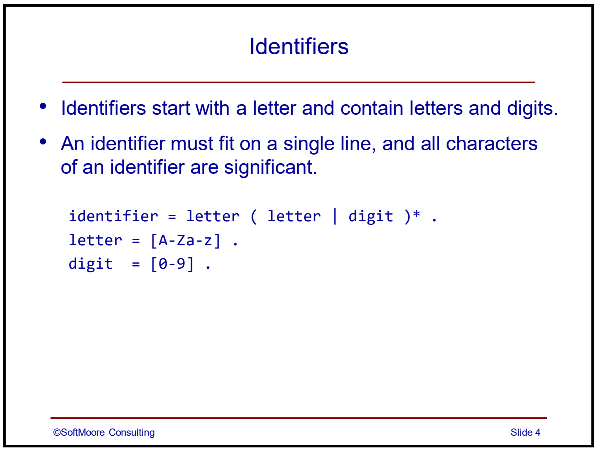
2



General Lexical Considerations

- Case sensitive. Upper-case letters and lower-case letters are considered to be distinct in all tokens, including reserved words.
- White space characters separate tokens; otherwise they are ignored.
- No token can extend past an end-of-line.
- Spaces may not appear in any token except character and string literals.
- A comment begins with two forward slashes (`//`) and extends to the end of the line.

3

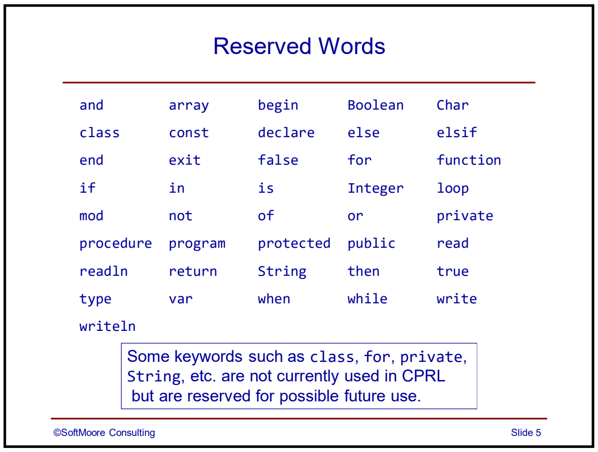


Identifiers

- Identifiers start with a letter and contain letters and digits.
- An identifier must fit on a single line, and all characters of an identifier are significant.

```
identifier = letter ( letter | digit )* .
letter = [A-Za-z] .
digit  = [0-9] .
```

4

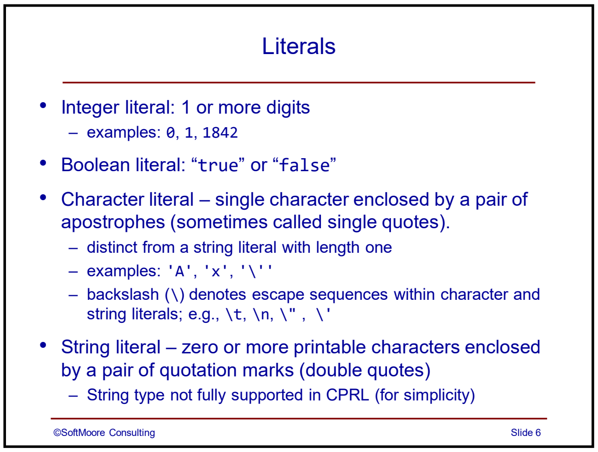


Reserved Words

and	array	begin	Boolean	Char
class	const	declare	else	elsif
end	exit	false	for	function
if	in	is	Integer	loop
mod	not	of	or	private
procedure	program	protected	public	read
readln	return	String	then	true
type	var	when	while	write
writeln				

Some keywords such as `class`, `for`, `private`, `String`, etc. are not currently used in CPRL but are reserved for possible future use.

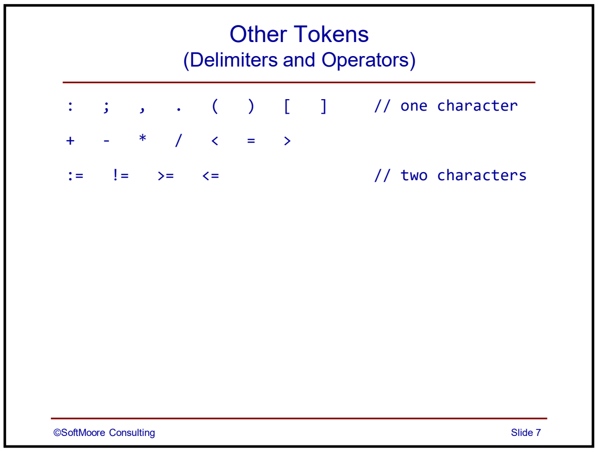
5



Literals

- Integer literal: 1 or more digits
  - examples: `0`, `1`, `1842`
- Boolean literal: "`true`" or "`false`"
- Character literal – single character enclosed by a pair of apostrophes (sometimes called single quotes).
  - distinct from a string literal with length one
  - examples: `'A'`, `'x'`, `'\''`
  - backslash (`\`) denotes escape sequences within character and string literals; e.g., `\t`, `\n`, `\"` , `\'`
- String literal – zero or more printable characters enclosed by a pair of quotation marks (double quotes)
  - String type not fully supported in CPRL (for simplicity)

6

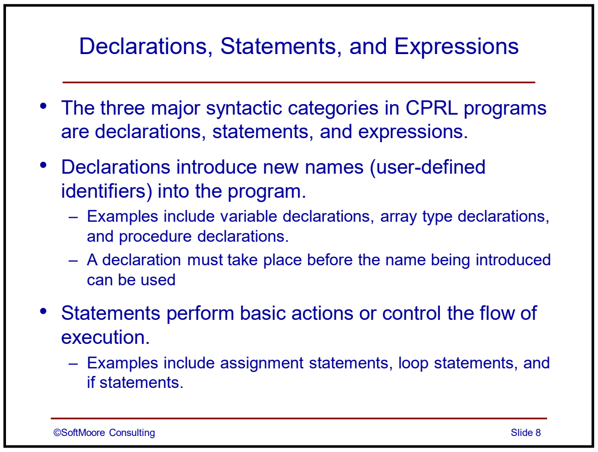


Other Tokens
(Delimiters and Operators)

```
:   ;   ,   .   (   )   [   ]     // one character
+   -   *   /   <   =   >
:=   !=   >=   <=                  // two characters
```

©SoftMoore Consulting Slide 7

7

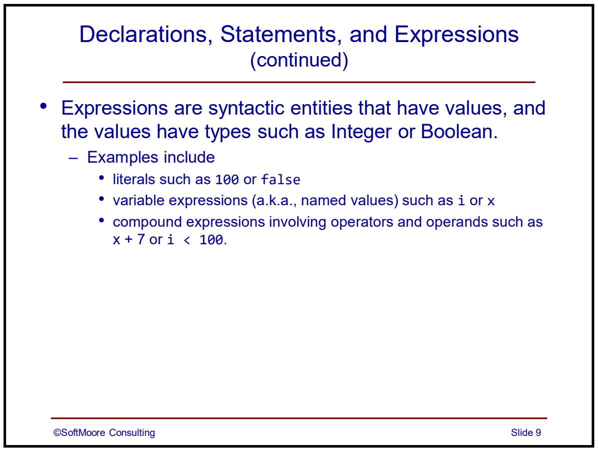


Declarations, Statements, and Expressions

- The three major syntactic categories in CPRL programs are declarations, statements, and expressions.
- Declarations introduce new names (user-defined identifiers) into the program.
  - Examples include variable declarations, array type declarations, and procedure declarations.
  - A declaration must take place before the name being introduced can be used
- Statements perform basic actions or control the flow of execution.
  - Examples include assignment statements, loop statements, and if statements.

©SoftMoore Consulting Slide 8

8

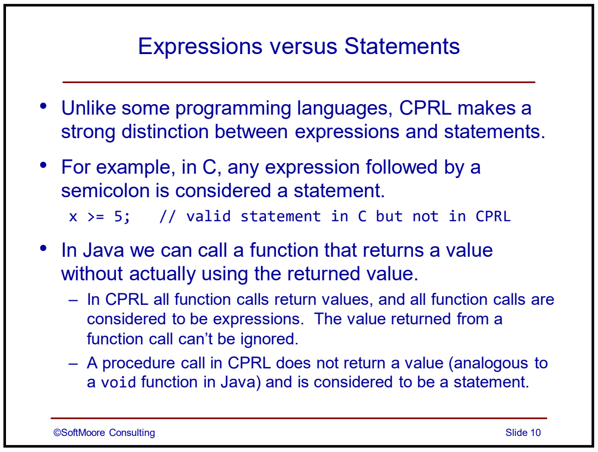


Declarations, Statements, and Expressions
(continued)

- Expressions are syntactic entities that have values, and the values have types such as Integer or Boolean.
  - Examples include
    - literals such as `100` or `false`
    - variable expressions (a.k.a., named values) such as `i` or `x`
    - compound expressions involving operators and operands such as `x` + `7` or `i < 100`.

©SoftMoore Consulting Slide 9

9



Expressions versus Statements

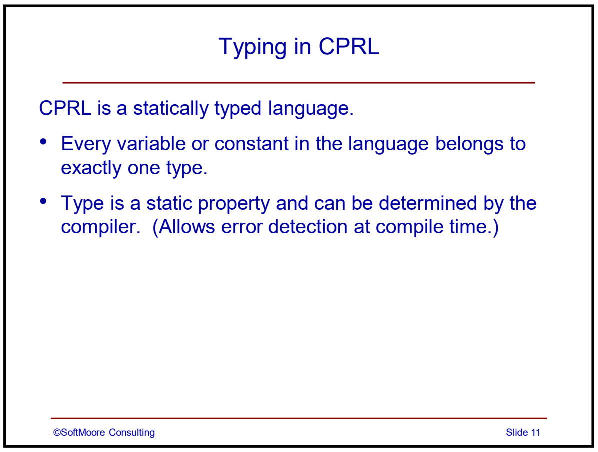
- Unlike some programming languages, CPRL makes a strong distinction between expressions and statements.
- For example, in C, any expression followed by a semicolon is considered a statement.

```
x >= 5;   // valid statement in C but not in CPRL
```

- In Java we can call a function that returns a value without actually using the returned value.
  - In CPRL all function calls return values, and all function calls are considered to be expressions. The value returned from a function call can't be ignored.
  - A procedure call in CPRL does not return a value (analogous to a `void` function in Java) and is considered to be a statement.

©SoftMoore Consulting Slide 10

10



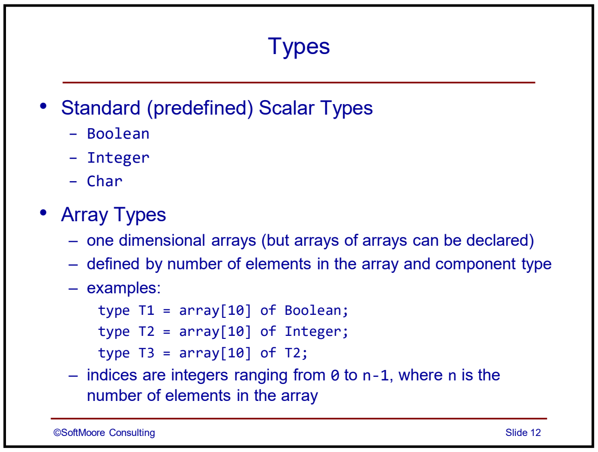
Typing in CPRL

CPRL is a statically typed language.

- Every variable or constant in the language belongs to exactly one type.
- Type is a static property and can be determined by the compiler. (Allows error detection at compile time.)

©SoftMoore Consulting Slide 11

11



Types

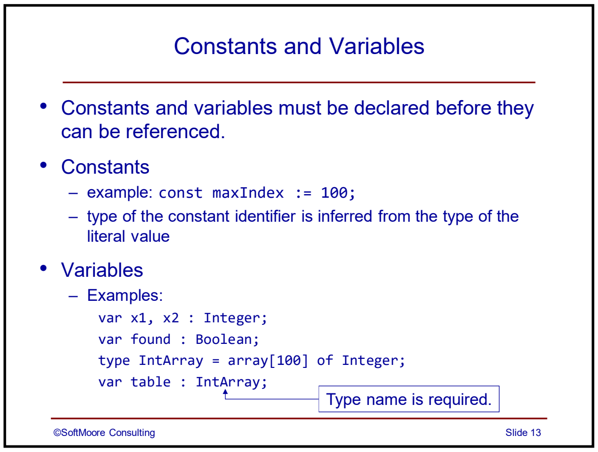
- Standard (predefined) Scalar Types
  - `Boolean`
  - `Integer`
  - `Char`
- Array Types
  - one dimensional arrays (but arrays of arrays can be declared)
  - defined by number of elements in the array and component type
  - examples:

```
type T1 = array[10] of Boolean;
type T2 = array[10] of Integer;
type T3 = array[10] of T2;
```

  - indices are integers ranging from `0` to `n-1`, where `n` is the number of elements in the array

©SoftMoore Consulting Slide 12

12



Constants and Variables

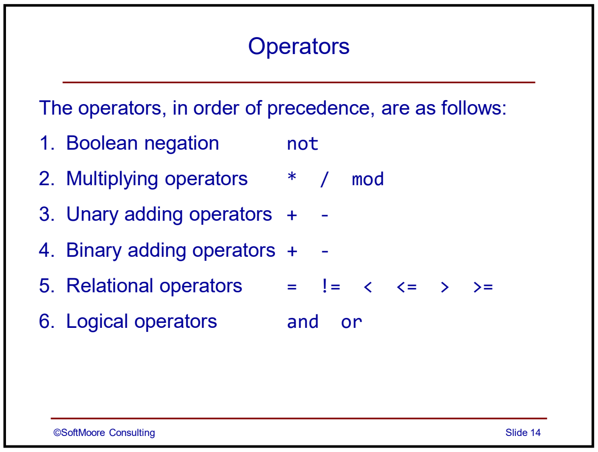
- Constants and variables must be declared before they can be referenced.
- Constants
  - example: `const maxIndex := 100;`
  - type of the constant identifier is inferred from the type of the literal value
- Variables
  - Examples:

```
var x1, x2 : Integer;
var found : Boolean;
type IntArray = array[100] of Integer;
var table : IntArray;
```

Type name is required.

©SoftMoore Consulting Slide 13

13



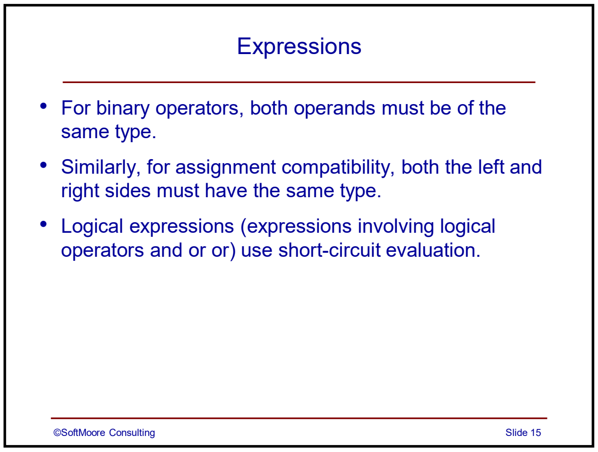
Operators

The operators, in order of precedence, are as follows:

1. Boolean negation `not`
2. Multiplying operators `*  /  mod`
3. Unary adding operators `+  -`
4. Binary adding operators `+  -`
5. Relational operators `=  !=  <  <=  >  >=`
6. Logical operators `and  or`

©SoftMoore Consulting Slide 14

14

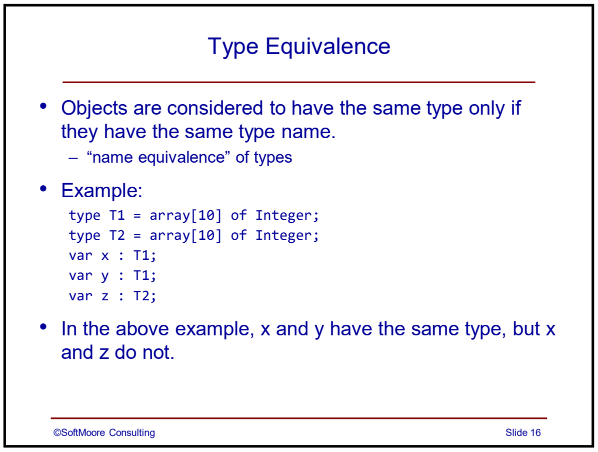


Expressions

- For binary operators, both operands must be of the same type.
- Similarly, for assignment compatibility, both the left and right sides must have the same type.
- Logical expressions (expressions involving logical operators and or or) use short-circuit evaluation.

©SoftMoore Consulting Slide 15

15



Type Equivalence

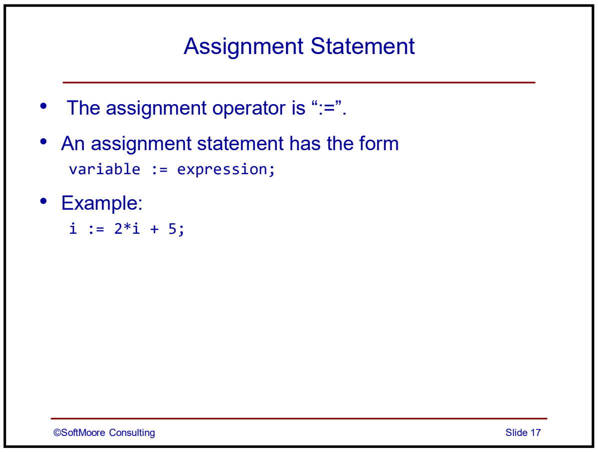
- Objects are considered to have the same type only if they have the same type name.
  - "name equivalence" of types
- Example:

```
type T1 = array[10] of Integer;
type T2 = array[10] of Integer;
var x : T1;
var y : T1;
var z : T2;
```

- In the above example, x and y have the same type, but x and z do not.

©SoftMoore Consulting Slide 16

16



Assignment Statement

- The assignment operator is ":=".
- An assignment statement has the form

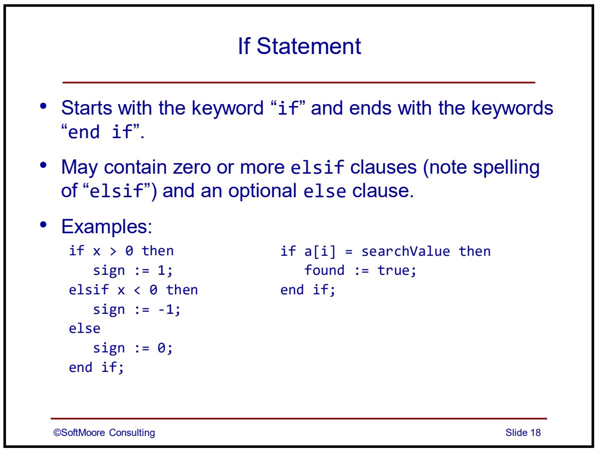
```
variable := expression;
```

- Example:

```
i := 2*i + 5;
```

©SoftMoore Consulting Slide 17

17



If Statement

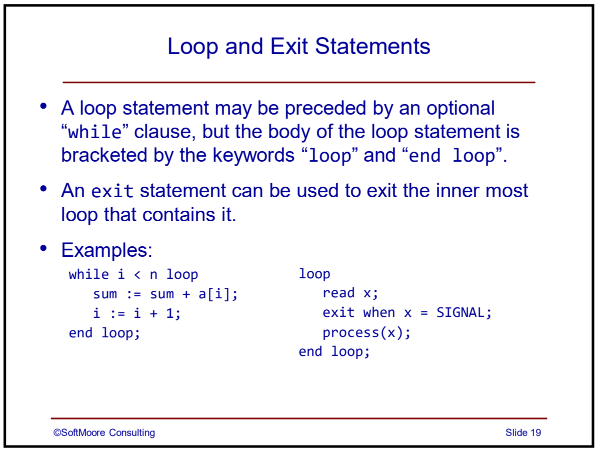
- Starts with the keyword "`if`" and ends with the keywords "`end if`".
- May contain zero or more `elsif` clauses (note spelling of "`elsif`") and an optional `else` clause.
- Examples:

```
if x > 0 then
   sign := 1;
elsif x < 0 then
   sign := -1;
else
   sign := 0;
end if;
```

```
if a[i] = searchValue then
   found := true;
end if;
```

©SoftMoore Consulting Slide 18

18



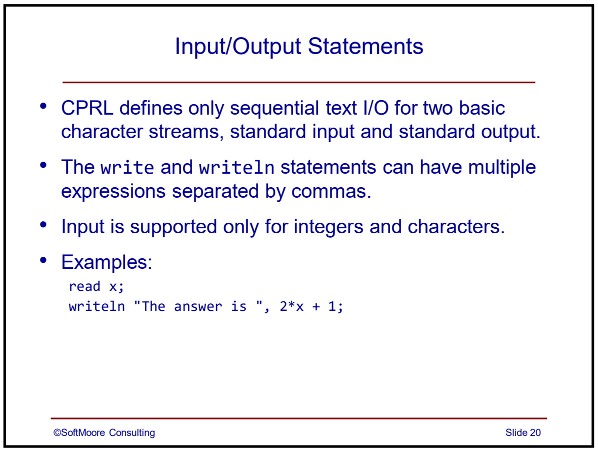
Loop and Exit Statements

- A loop statement may be preceded by an optional "`while`" clause, but the body of the loop statement is bracketed by the keywords "`loop`" and "`end loop`".
- An `exit` statement can be used to exit the inner most loop that contains it.
- Examples:

```
while i < n loop              loop
   sum := sum + a[i];            read x;
   i := i + 1;                   exit when x = SIGNAL;
end loop;                        process(x);
                              end loop;
```

©SoftMoore Consulting Slide 19

19



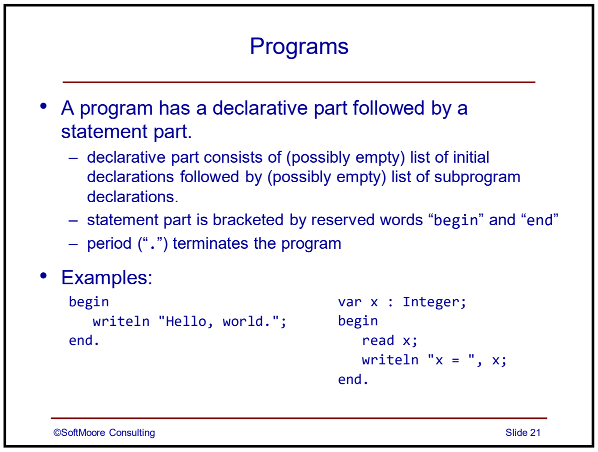
Input/Output Statements

- CPRL defines only sequential text I/O for two basic character streams, standard input and standard output.
- The `write` and `writeln` statements can have multiple expressions separated by commas.
- Input is supported only for integers and characters.
- Examples:

```
read x;
writeln "The answer is ", 2*x + 1;
```

©SoftMoore Consulting Slide 20

20



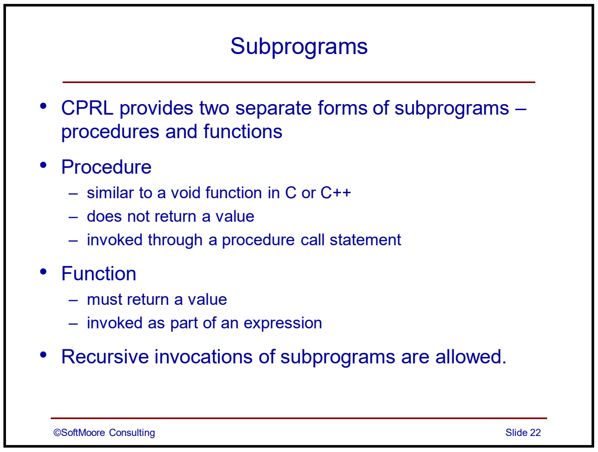
Programs

- A program has a declarative part followed by a statement part.
  - declarative part consists of (possibly empty) list of initial declarations followed by (possibly empty) list of subprogram declarations.
  - statement part is bracketed by reserved words "`begin`" and "`end`"
  - period ("`.`") terminates the program
- Examples:

```
begin                            var x : Integer;
   writeln "Hello, world.";      begin
end.                                read x;
                                    writeln "x = ", x;
                                 end.
```

©SoftMoore Consulting Slide 21

21

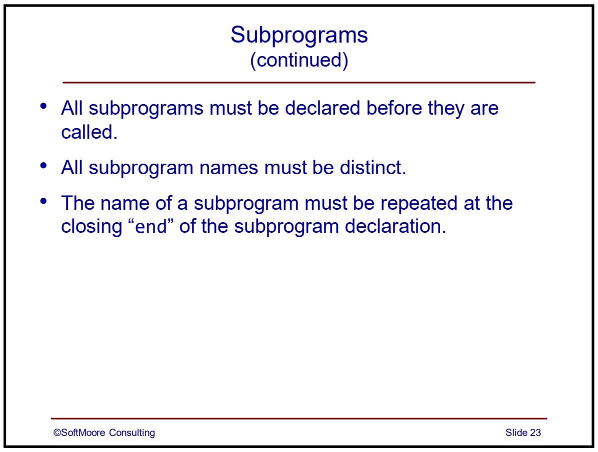


Subprograms

- CPRL provides two separate forms of subprograms – procedures and functions
- Procedure
  - similar to a void function in C or C++
  - does not return a value
  - invoked through a procedure call statement
- Function
  - must return a value
  - invoked as part of an expression
- Recursive invocations of subprograms are allowed.

©SoftMoore Consulting Slide 22

22

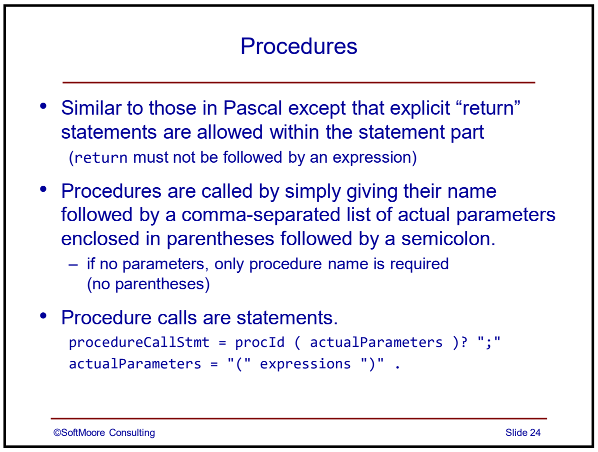


Subprograms
(continued)

- All subprograms must be declared before they are called.
- All subprogram names must be distinct.
- The name of a subprogram must be repeated at the closing "`end`" of the subprogram declaration.

©SoftMoore Consulting Slide 23

23



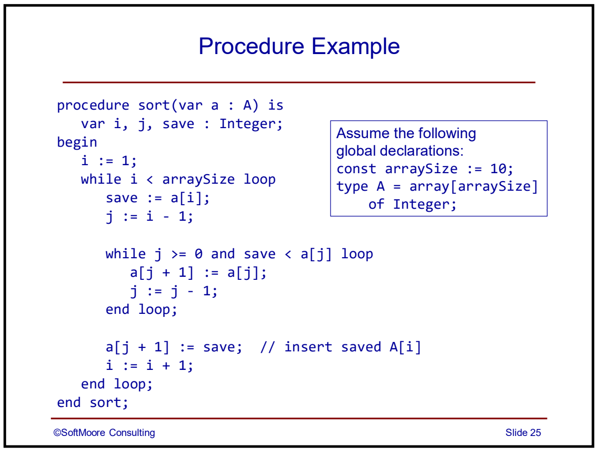
Procedures

- Similar to those in Pascal except that explicit "return" statements are allowed within the statement part
  (`return` must not be followed by an expression)
- Procedures are called by simply giving their name followed by a comma-separated list of actual parameters enclosed in parentheses followed by a semicolon.
  - if no parameters, only procedure name is required (no parentheses)
- Procedure calls are statements.

```
procedureCallStmt = procId ( actualParameters )? ";"
actualParameters = "(" expressions ")" .
```

©SoftMoore Consulting Slide 24

24



Procedure Example

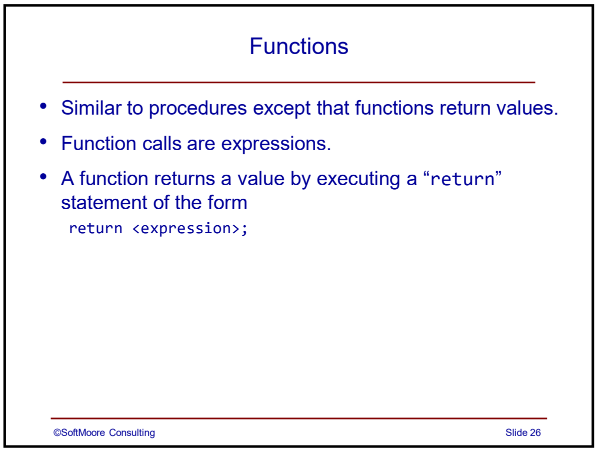
```
procedure sort(var a : A) is
   var i, j, save : Integer;
begin
   i := 1;
   while i < arraySize loop
      save := a[i];
      j := i - 1;

      while j >= 0 and save < a[j] loop
         a[j + 1] := a[j];
         j := j - 1;
      end loop;

      a[j + 1] := save;  // insert saved A[i]
      i := i + 1;
   end loop;
end sort;
```

Assume the following global declarations:

```
const arraySize := 10;
type A = array[arraySize]
    of Integer;
```

©SoftMoore Consulting Slide 25

25



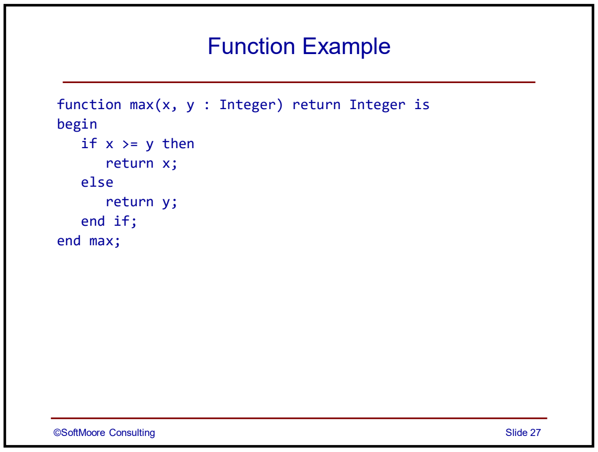
Functions

- Similar to procedures except that functions return values.
- Function calls are expressions.
- A function returns a value by executing a “`return`” statement of the form

```
return <expression>;
```

©SoftMoore Consulting Slide 26

26

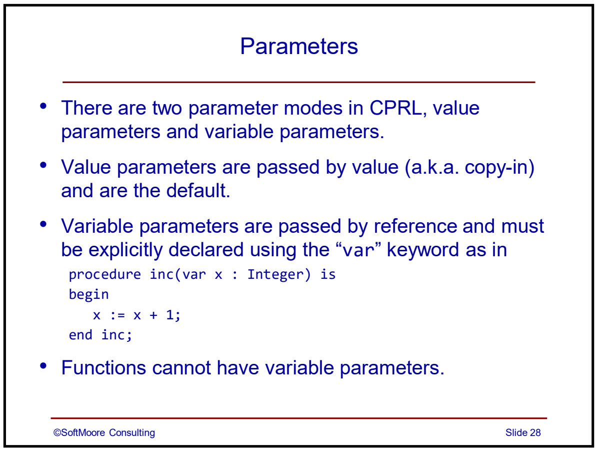


Function Example

```
function max(x, y : Integer) return Integer is
begin
   if x >= y then
      return x;
   else
      return y;
   end if;
end max;
```

©SoftMoore Consulting Slide 27

27



Parameters

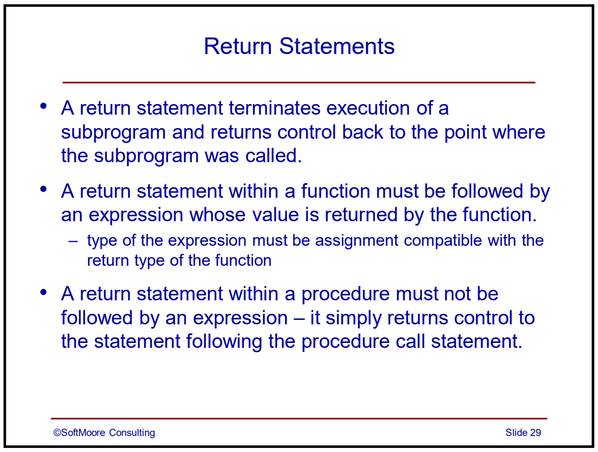
- There are two parameter modes in CPRL, value parameters and variable parameters.
- Value parameters are passed by value (a.k.a. copy-in) and are the default.
- Variable parameters are passed by reference and must be explicitly declared using the “`var`” keyword as in

```
procedure inc(var x : Integer) is
begin
   x := x + 1;
end inc;
```

- Functions cannot have variable parameters.

©SoftMoore Consulting Slide 28

28

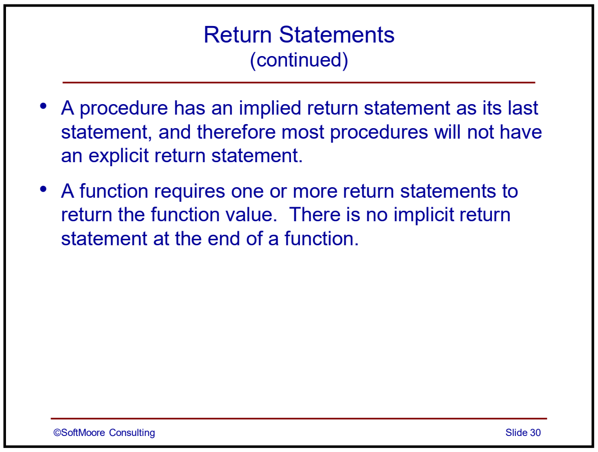


Return Statements

- A return statement terminates execution of a subprogram and returns control back to the point where the subprogram was called.
- A return statement within a function must be followed by an expression whose value is returned by the function.
  - type of the expression must be assignment compatible with the return type of the function
- A return statement within a procedure must not be followed by an expression – it simply returns control to the statement following the procedure call statement.

©SoftMoore Consulting Slide 29

29

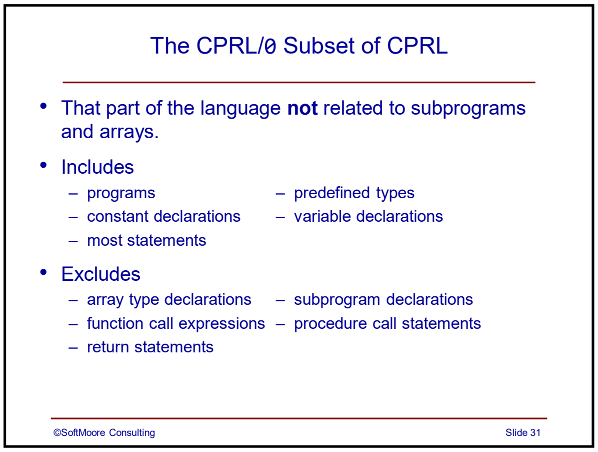


Return Statements
(continued)

- A procedure has an implied return statement as its last statement, and therefore most procedures will not have an explicit return statement.
- A function requires one or more return statements to return the function value. There is no implicit return statement at the end of a function.

©SoftMoore Consulting Slide 30

30



The CPRL/0 Subset of CPRL

- That part of the language **not** related to subprograms and arrays.
- Includes
  - programs
  - predefined types
  - constant declarations
  - variable declarations
  - most statements
- Excludes
  - array type declarations
  - subprogram declarations
  - function call expressions
  - procedure call statements
  - return statements

©SoftMoore Consulting

Slide 31

31