Appendix 1

A Quick Introduction to C

C is a popular Algol-like language with features that support systems programming. Although it cannot be explained or learned in ten minutes, a few pointers and some simple examples are usually sufficient to provide experienced programmers with a reading knowledge sufficient to understand the code in this book. A chart that explains the basics of C syntax and semantics in terms of Pascal follows a short list of the highlights of the language.†

1. Types are not as strict as in Pascal. Although later versions of the language are more strongly typed than early versions, compilers usually allow data types to be interchanged freely. One often sees arithmetic operations on characters, because declaring something to be a character is an easy (although nonportable) way of declaring a one-byte object.
2. There is no type *Boolean*. A nonzero integer is equivalent to *true*; zero is equivalent to *false*.
3. Like Pascal and PL/I, C has aggregate types *array* and *structure* (called *record* in Pascal). Unlike many languages, C has no notion of files.
4. C procedure declarations may not be nested, so it is not block-structured in the Algol, Pascal, or PL/I sense.
5. C does not distinguish *procedures* from *functions*. All procedures return a value; if the programmer does not specify one, the value returned will be nonsense. The calling program can ignore the returned value by invoking a procedure in a single statement or can use the returned value by invoking the procedure in an expression.
6. C supports recursive procedure calls.
7. The C syntax is concise. C uses left and right braces in place of PL/I's *do end* or Pascal's *begin end*.
8. Brackets [ ] denote subscripting, and parentheses ( ) denote procedure calls. If x is the name of an array, then x[i] refers to element i, but the name x used without subscripts refers to the address of the first element (i.e., the address

†For more details about C consult Kernighan and Ritchie [1978].

of x[0]). Similarly, a procedure name that appears without parentheses following refers to the address of the code for the procedure. Subscripts for an array of *N* elements range from 0 through *N*–1.

9. C has an extensive and powerful set of operators including such functions as bit-wise *shift*, *and*, *or*, *complement*, and *exclusive or*. Operators are often denoted by one or two special characters. Even operators that have side-effects return functional values. For example, the assignment operator assigns its right-hand operand to its left-hand operand and returns the value assigned as in APL. Some surprises: = denotes assignment, == denotes test for equality, != denotes test for inequality, *x denotes *whatever x points to*, x++ has the same value as x, but it has the side-effect of incrementing x after it has been accessed, ++x means *first increment x and then return its new value*.
10. C allows arithmetic on pointers, where the type of the pointer determines how the new address is computed. If x points to an object of size y bytes, then x+1 points to the ''next'' occurrence of the object (i.e., produces an address that is equivalent to adding y to the integer value of x).
11. Pointer arithmetic and array subscripting in C are curiously related. By definition, x[a] means ''add'' a to x, where addition is performed according to the rules of pointer arithmetic. The definition happens to make a[x] equivalent to x[a] if a is an integer and x is an array. It also means that if p points to x[5] then p +1 points to x[6] (provided p is declared to be a pointer to the type of object in x). Unfortunately, many C programs contain cryptic code that ''walks'' through an array by assigning a pointer the starting address and incrementing the pointer with ++.
12. Semicolons are statement terminators; they follow every statement except for compound statements. Like PL/I, but unlike Pascal, semicolons in C precede the keyword *else*. A semicolon by itself is a null statement.
13. The C macro preprocessor handles parameterized macros, allowing in-line expansion of code and substitution of symbolic constants in the program. The macro facility, which also provides for source file inclusion, is less powerful than that of PL/I, but more powerful than the *const* facility in Pascal. By convention, the names of symbolic constants are written in upper case; other names are written in lower case. Constants and data used by more than one source program are usually declared in a separate file and included in the source program at compile time. By convention, the names of included files end in ''.h'' (for *header*).
14. C supports separate compilation of procedures. In declarations, the keyword *static* limits the visibility of the declared name to the currently compiled file, providing a way to group procedures and data so that only selected names are visible in the rest of the program. Whenever a procedure is loaded, the loader includes all procedures and data that were compiled with it.
15. The keyword *extern* makes a declaration refer to an external object; all references to a given external name refer to the same object independent of the

file in which the declaration occurred (like external variables in PL/I or named common in FORTRAN). Declaring an object to be external merely instructs the compiler to generate code to reference it as such; the actual definition of storage for an external object is made by writing a declaration outside of the scope of a procedure. External objects must be defined once no matter how many times they are declared.

16. In C, data can be initialized at compile time; by default, the initial value of an external data object is zero.
17. Unlike the Pascal *case* statement that selects one out of n statements (i.e., is an n-way conditional), the C *switch* statement is merely an n-way branch. The switch expression is evaluated and control transfers to one of n branch points (called *cases*) as expected; but after the transfer, execution continues falling though case after case until the program explicitly branches out of the switch statement. The usual way to branch out is with a *break* statement, which causes flow of control to pass to the statement following the switch. *Switch* statements can include a *default* label, equivalent to *otherwise* in some dialects of Pascal. Control passes to the *default* if no other case is true.
18. Comments start with ‘‘/*’’ and end with the next ‘‘*/’’ as in PL/I.
19. Identifiers can contain letters, digits, or underscores; they must begin with a letter or an underscore.

The following chart explains specific constructs in C by giving their equivalent in Pascal. Notice, in particular, the powerful *for* statement, the array subscripting (which extends from 0 through *N*-1, not 1 through *N*), and the unusual declaration syntax.

C construct	explanation	Pascal equivalent
int a; char b; char *c; int (*x)();	declarations: integer character pointer to character pointer to procedure that returns integer	var a: integer; var b: char; var c: ↑char; -none-
char d[10]; char e[10][12]; char *f[5] char **g;	array of characters 2-dimensional array array of pointers pointer to pointer	var d: array[0..9] of char; var e: array[0..9,0..11] of char; var f: array[0..4] of ↑char; var g: ↑↑char;
´a´ ´\014´	character constant character constant with value 014 (octal)	´a´ -none-
"abc"	string constant Array of contiguous characters terminated by a null byte (i.e., ’a’, ’b’, ’c’, ’\0’). Newline and tab denoted by \n and \t. Value is the the address of the first character	-none- (some Pascal compilers use ’...’)
123 0123 0x123	decimal constant octal constant (has leading zero) hexadecimal constant	123 -none- (decimal value is 83) -none- (decimal value is 291)

struct x {
int f1;
char f2;
}
	structure (record)
	declaration with
	fields f1 and f2
		x = record
		f1: integer;
		f2: char
		end

struct x y[2];	y is array of struct x	var y: array[0..1] of x;
#define A v	symbolic constant	const A = v;
#define A(x) v(x)	parameterized macro	-none-
#ifdef A X #endif	conditional compilation code X is compiled only if symbol A defined	-none-
#ifndef A X #endif	negative conditional compilation; X compiled only if A not defined	-none-

C Construct	explanation	Pascal equivalent
#include *src*	source file inclusion (if *src* is *"path"* then path is relative to current directory; if <*path*> then relative to system directory)	-none-
=	assignment operator	:=
+ - *	arithmetic operators	+ - *
/	division (C does integer division on integers)	/ div
%	modulus or remainder	-none-
var op= exp v += 9	operation and assignment example: add 9 to variable v	var := var op exp v := v + 9
== != > < <= >=	test equality test inequality test greater than test less than test less than or equal test greater than or equal	= <> > < <= >=
*x	pointer dereference (whatever x points to)	x↑
sizeof(x)	size of data object x in bytes	-none-
&x	address of object x (when used as a unary operator)	-none-
&	bitwise and (when used as binary operator)	-none-
|	bitwise or	-none-
~	bitwise (1's) complement	-none-
&& || !	Boolean and Boolean or Boolean not (&& and || are evaluated left-to-right with early termination)	and or not (Pascal does not use early termination)
x[i]	array reference	x[i]

C construct	explanation	Pascal equivalent
s.f p–>f	reference to field f in structure s reference to field f in structure pointed to by p	s.f p↑.f
e ? a : b	conditional expression (if e is nonzero, value is a else value is b)	-none-
++x x++ – –x x– –	preincrement postincrement predecrement post decrement (when used in an expression ++x refers to the value of x after incrementing; x++ refers to the value before incrementing)	x := x + 1 x := x + 1 x := x – 1 x := x – 1
p(e1,e2,...,en)	procedure invocation	p(e1,e2,...,en)
while (exp) S;	indefinite iteration	while exp <> 0 do S
if (exp) S;	conditional	if exp <> 0 then S
if (exp) S1; else S2;	2-way conditional	if exp <> 0 then S1 else S2
{S1 ; S2 ;...; Sn ; }	compound statement (note semicolons)	begin S1 ; S2 ;...; Sn end

for(S1 ; exp ; S2)
S3;
	indefinite iteration with initialization and reinitialization (S1, exp and S2 are optional – if exp is omitted, infinite loop results)	S1;
		while exp <> 0 do
		begin
		S3;
		S2
		end

return	procedure return	finish executing procedure
return(exp)	return exp to caller as function value	*function name* := exp; finish executing function

name(formals)
declaration of formals
{
declaration of
local variables;
statements
}
	procedure declaration	procedure name(formals);
		declaration of locals;
		begin
		
		statements
		
		end;

C construct	explanation	Pascal equivalent

type name(formals)
declaration of formals
{
  declaration of
  local variables;
  statements
}
	function declaration	function name(formals) : *type*;
		declaration of locals;
		begin
		
		  statements
		
		end;

for(i=0 ; i<N ; i++)
  ...x[i]...
	typical loop to search array x, assuming x has size N	for i := 1 to N do
		  ...x[i]...

*x++	idiomatic expression for pointer x: its value is whatever x points to; x is incremented after the reference according to pointer arithmetic	-none-
(type)exp 1 + (int) &x	type casting the type of expression exp is changed. Example: make the address of x an integer before adding 1 to it (i.e., use integer, not pointer, arithmetic)	-none-
/*...*/	Comment	(*...*) or {...}