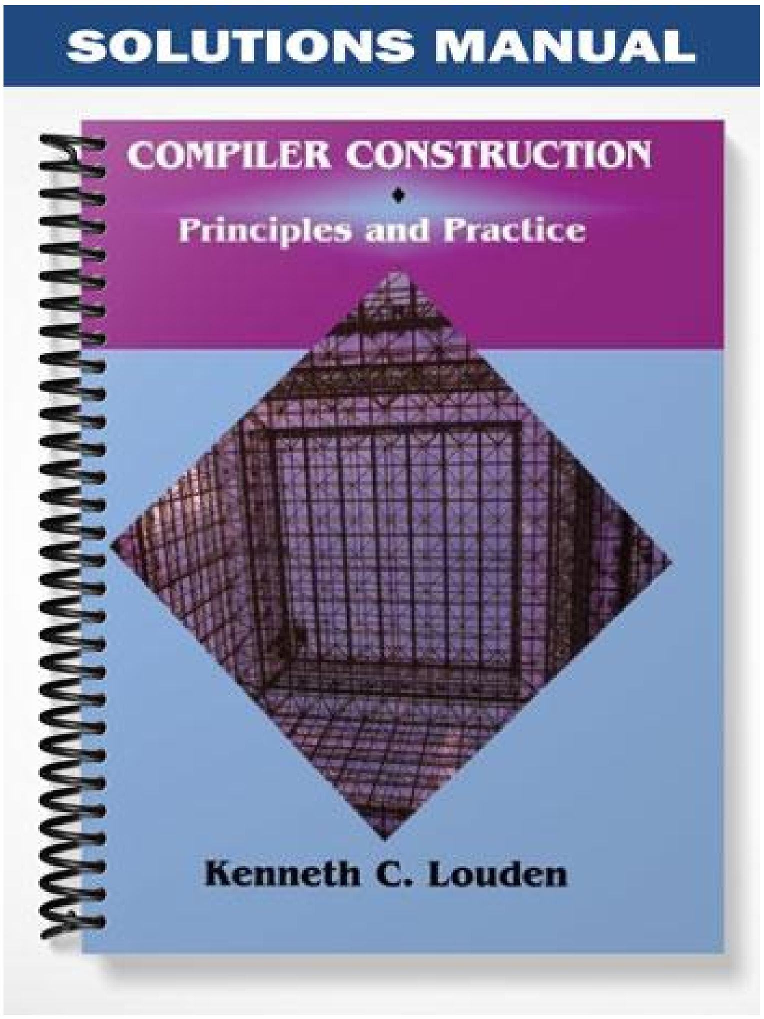


SOLUTIONS MANUAL

COMPILER CONSTRUCTION

♦

Principles and Practice

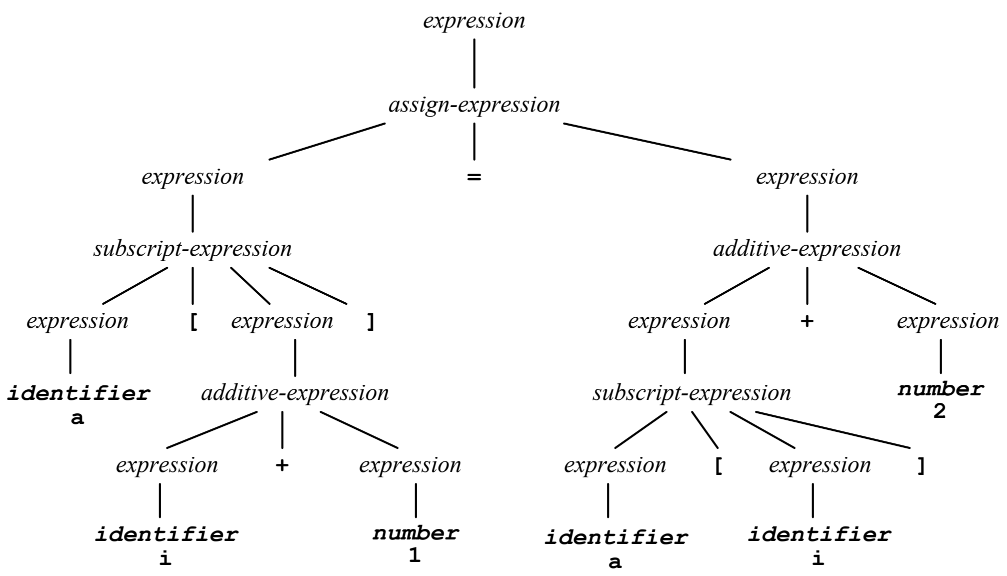
Kenneth C. Louden

Compiler Construction: Principles and Practice
by Kenneth C. Louden

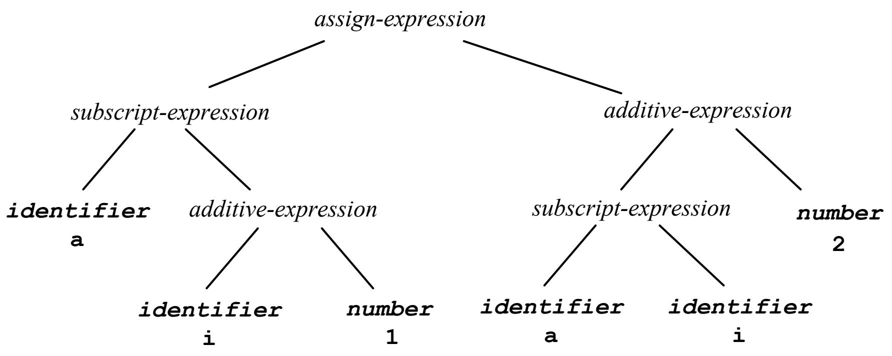
Chapter 1 Exercise Answers

<u>Exercise 1.2</u>

The parse tree is:



The syntax tree is:



<u>Exercise 1.4</u>

(a) and (b). The following table shows the assembly output of three compilers (slightly edited, with optimizations turned on) for four similar short functions in C.

C Source Code	Borland 3.0 (PC)	Sun C (Sparc)	Gnu C (Sun Sparc)

int f(void)
{ int x = 2 + 3;
  return x;
}
	_f proc near
	   mov ax,5
	   ret
	_f endp
		_f:
		  retl
		  add %g0,5,%o0
			_f:
			  save %sp,-112,%sp
			  mov 5,%i0
			  ret
			  restore

int f(void)
{const int x = 2;
 const int y = 3;
 int z = x + y;
 return z;
}
	_f proc near
	   push bp
	   mov bp,sp
	   sub sp,2
	   mov ax,2
	   add ax,3
	   mov word ptr
	          [bp-2],ax
	   mov sp,bp
	   pop bp
	   ret
	_f endp
		_f:
		  retl
		  add %g0,5,%o0
			_f:
			  save %sp,-112,%sp
			  mov 5,%i0
			  ret
			  restore

int f(void)
{ int x = 2;
  int y = 3;
  int z = x + y;
  return z;
}
	_f proc near
	   push bp
	   mov bp,sp
	   sub sp,2
	   mov ax,2
	   add ax,3
	   mov word ptr
	          [bp-2],ax
	   mov sp,bp
	   pop bp
	   ret
	_f endp
		_f:
		  retl
		  add %g0,5,%o0
			_f:
			  save %sp,-112,%sp
			  mov 5,%i0
			  ret
			  restore

int f(void)
{ int x = 2;
  int y = 3;
  int z;
  if (z) z = x+y;
  else z = x-y;
  return z;
}
	_f proc near
	   push bp
	   mov bp,sp
	   sub sp,4
	   mov word ptr
	           [bp-2],2
	   mov word ptr
	           [bp-4],3
	   or dx,dx
	   je short @1@86
	   mov dx,2
	   add dx,3
	   jmp short
	             @1@114
	@1@86:
	   mov dx,word ptr
	             [bp-2]
	   sub dx,word ptr
	             [bp-4]
	@1@114:
	   mov ax,dx
	   mov sp,bp
	   pop bp
	   ret
	_f endp
		_f:
		  save %sp,-72,%sp
		  ld [%fp-4],%o0
		  tst %o0
		  be,a L77005
		  mov -1,%i5
		  mov 5,%i5
		L77005:
		  ret
		  restore
		         %g0,%i5,%o0
			_f:
			  save %sp,-112,%sp
			  mov 2,%o2
			  mov 3,%o1
			  tst %o0
			  be L2
			  nop
			  b L3
			  mov 5,%o0
			L2:
			  sub %o2,%o1,%o0
			L3:
			  mov %o0,%i0
			  ret
			  restore

The table shows that all three compilers perform constant folding, and that the Gnu C compiler performs constant propagation except where control flow is present. The Sun C compiler is the only one that performs propagation in all cases.

(c) Constant propagation is more difficult than constant folding because the value of a variable must be tracked through a number of statements to determine that it is constant, while in constant folding the constants are right in the statement itself (they are literals or "manifest" constants). Tracking constant values of variables is particularly difficult in the presence of control flow, as the fourth example in the table shows.

(d) Note that the use of `const` declarations had no effect on the code the above compilers produce. This does not mean that the compilers could not have used the information provided by such a declaration. In fact, a `const` variable could be of two different kinds: its value could be determinable at compile time (such as in the code example above), or its value might not be determinable until execution time, such as in a declaration

```
    const char c = getchar();
```

Clearly only in the former case can optimizations be peformed.

Exercise 1.6

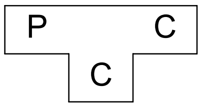
(a) A language preprocessor is a program that performs simple manipulations on a source program file before it is sent to the compiler proper. Preprocessors perform such manipulations as textual replacement, inclusion of additional files, conditional inclusion/exclusion of blocks of code, etc. A standard example is the C preprocessor, which reads all lines beginning with the # character, and performs textual replacements from a symbol table on remaining lines. Preprcessors generally do not perform extensive compiler-like analysis, but only provide simple macro-like operations.

(b) A pretty-printer is a program that will format a source file in a way that is especially readable. Usually this includes adjusting indentations, printing keywords in boldface, and possibly comments in italics. Pretty-printers must perform a full parse of the source text in order to know when to indent, and to identify keywords and comments.

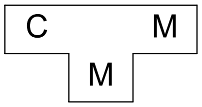
(c) A text formatter is a program that adjusts the spacing of text on a page to make it print in a pleasing form. Usually this includes justification of text and font manipulations such as kerning (variable spacing). Such a program is not all all concerned with the internal structure of source code, since it treats all text alike. However, it geneally needs to be able to accept formatting codes, and occasionally must perform certain automatic operations (like adjusting type size during subscripting), and this involves some parsing and analysis similar to that peformed by a compiler.

Exercise 1.7

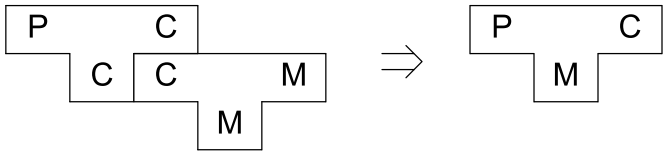
Let us represent Pascal programs with the letter P, C programs with the letter C, and programs that run on the machine we are using by the letter M. Then the Pascal-to-C translator written in C can be represented as



and the working C compiler as



We form a working Pascal compiler in two steps. First, we compile the Pascal-to-C translator using the working C compiler to get a working Pascal-to-C translator:



Then we link the working Pascal-to-C translator with the working C compiler to form a working Pascal compiler (this operation can be viewed as a pipe from the first translator to the second):