14.3 Arrays, pointers, and subscripting

Every time an identifier of array type appears in an expression, it is converted into a pointer to the first member of the array. Because of this conversion, arrays are not lvalues. By definition, the subscript operator `[]` is interpreted in such a way that `E1[E2]` is identical to `*((E1)+(E2))`. Because of the conversion rules which apply to `+`, if `E1` is an array and `E2` an integer, then `E1[E2]` refers to the `E2`-th member of `E1`. Therefore, despite its asymmetric appearance, subscripting is a commutative operation.

A consistent rule is followed in the case of multi-dimensional arrays. If `E` is an n-dimensional array of rank $i \times j \times \cdots \times k$, then `E` appearing in an expression is converted to a pointer to an $(n-1)$-dimensional array with rank $j \times \cdots \times k$. If the `*` operator, either explicitly or implicitly as a result of subscripting, is applied to this pointer, the result is the pointed-to $(n-1)$-dimensional array, which itself is immediately converted into a pointer.

For example, consider

```
        int x[3][5];
```

Here `x` is a 3×5 array of integers. When `x` appears in an expression, it is converted to a pointer to (the first of three) 5-membered arrays of integers. In the expression `x[i]`, which is equivalent to `*(x+i)`, `x` is first converted to a pointer as described; then `i` is converted to the type of `x`, which involves multiplying `i` by the length the object to which the pointer points, namely 5 integer objects. The results are added and indirection applied to yield an array (of 5 integers) which in turn is converted to a pointer to the first of the integers. If there is another subscript the same argument applies again; this time the result is an integer.

It follows from all this that arrays in C are stored row-wise (last subscript varies fastest) and that the first subscript in the declaration helps determine the amount of storage consumed by an array but plays no other part in subscript calculations.

14.4 Explicit pointer conversions

Certain conversions involving pointers are permitted but have implementation-dependent aspects. They are all specified by means of an explicit type-conversion operator, §§7.2 and 8.7.

A pointer may be converted to any of the integral types large enough to hold it. Whether an `int` or `long` is required is machine dependent. The mapping function is also machine dependent, but is intended to be unsurprising to those who know the addressing structure of the machine. Details for some particular machines are given below.

An object of integral type may be explicitly converted to a pointer. The mapping always carries an integer converted from a pointer back to the same pointer, but is otherwise machine dependent.

A pointer to one type may be converted to a pointer to another type. The resulting pointer may cause addressing exceptions upon use if the subject pointer does not refer to an object suitably aligned in storage. It is guaranteed that a pointer to an object of a given size may be converted to a pointer to an object of a smaller size and back again without change.

For example, a storage-allocation routine might accept a size (in bytes) of an object to allocate, and return a `char` pointer; it might be used in this way.

```
        extern char *alloc();
        double *dp;

        dp = (double *) alloc(sizeof(double));
        *dp = 22.0 / 7.0;
```

`alloc` must ensure (in a machine-dependent way) that its return value is suitable for conversion to a pointer to `double`; then the *use* of the function is portable.

The pointer representation on the PDP-11 corresponds to a 16-bit integer and is measured in bytes. `char`s have no alignment requirements; everything else must have an even address.

On the Honeywell 6000, a pointer corresponds to a 36-bit integer; the word part is in the left 18 bits, and the two bits that select the character in a word just to their right. Thus `char` pointers are measured in units of 2^{16} bytes; everything else is measured in units of 2^{18} machine words. `double` quantities and aggregates containing them must lie on an even word address (0 mod 2^{19}).

The IBM 370 and the Interdata 8/32 are similar. On both, addresses are measured in bytes; elementary objects must be aligned on a boundary equal to their length, so pointers to `short` must be 0 mod 2, to `int` and `float` 0 mod 4, and to `double` 0 mod 8. Aggregates are aligned on the strictest boundary required by any of their constituents.

15. Constant expressions

In several places C requires expressions which evaluate to a constant: after `case`, as array bounds, and in initializers. In the first two cases, the expression can involve only integer constants, character constants, and `sizeof` expressions, possibly connected by the binary operators

```
+  -  *  /  %  &  |  ^  <<  >>  ==  !=  <  >  <=  >=
```

or by the unary operators

```
-  ~
```

or by the ternary operator

```
?:
```

Parentheses can be used for grouping, but not for function calls.

More latitude is permitted for initializers; besides constant expressions as discussed above, one can also apply the unary `&` operator to external or static objects, and to external or static arrays subscripted with a constant expression. The unary `&` can also be applied implicitly by appearance of unsubscripted arrays and functions. The basic rule is that initializers must evaluate either to a constant or to the address of a previously declared external or static object plus or minus a constant.

16. Portability considerations

Certain parts of C are inherently machine dependent. The following list of potential trouble spots is not meant to be all-inclusive, but to point out the main ones.

Purely hardware issues like word size and the properties of floating point arithmetic and integer division have proven in practice to be not much of a problem. Other facets of the hardware are reflected in differing implementations. Some of these, particularly sign extension (converting a negative character into a negative integer) and the order in which bytes are placed in a word, are a nuisance that must be carefully watched. Most of the others are only minor problems.

The number of `register` variables that can actually be placed in registers varies from machine to machine, as does the set of valid types. Nonetheless, the compilers all do things properly for their own machine; excess or invalid `register` declarations are ignored.

Some difficulties arise only when dubious coding practices are used. It is exceedingly unwise to write programs that depend on any of these properties.

The order of evaluation of function arguments is not specified by the language. It is right to left on the PDP-11, left to right on the others. The order in which side effects take place is also unspecified.

Since character constants are really objects of type `int`, multi-character character constants are permitted. The specific implementation is very machine dependent, however, because the order in which characters are assigned to a word varies from one machine to another.

Fields are assigned to words and characters to integers right-to-left on the PDP-11 and left-to-right on other machines. These differences are invisible to isolated programs which do not indulge in type punning (for example, by converting an `int` pointer to a `char` pointer and inspecting the pointed-to storage), but must be accounted for when conforming to externally-imposed storage layouts.

The language accepted by the various compilers differs in minor details. Most notably, the current PDP-11 compiler will not initialize structures containing bit-fields, and does not accept a few assignment operators in certain contexts where the value of the assignment is used.

17. Anachronisms

Since C is an evolving language, certain obsolete constructions may be found in older programs. Although most versions of the compiler support such anachronisms, ultimately they will disappear, leaving only a portability problem behind.

Earlier versions of C used the form =*op* instead of *op*= for assignment operators. This leads to ambiguities, typified by

```
x=-1
```

which actually decrements `x` since the = and the - are adjacent, but which might easily be intended to assign -1 to `x`.

The syntax of initializers has changed: previously, the equals sign that introduces an initializer was not present, so instead of

```
int  x    = 1;
```

one used

```
int  x    1;
```

The change was made because the initialization

```
    int  f    (1+2)
```

resembles a function declaration closely enough to confuse the compilers.

18. Syntax Summary

This summary of C syntax is intended more for aiding comprehension than as an exact statement of the language.

18.1 Expressions

The basic expressions are:

```
expression:
        primary
        * expression
        & expression
        - expression
        ! expression
        ~ expression
        ++ lvalue
        -- lvalue
        lvalue ++
        lvalue --
        sizeof expression
        ( type-name ) expression
        expression binop expression
        expression ? expression : expression
        lvalue asgnop expression
        expression , expression

primary:
        identifier
        constant
        string
        ( expression )
        primary ( expression-list_opt )
        primary [ expression ]
        lvalue . identifier
        primary -> identifier

lvalue:
        identifier
        primary [ expression ]
        lvalue . identifier
        primary -> identifier
        * expression
        ( lvalue )
```

The primary-expression operators

```
() [] . ->
```

have highest priority and group left-to-right. The unary operators

```
* & - ! ~ ++ -- sizeof ( type-name )
```

have priority below the primary operators but higher than any binary operator, and group right-to-left. Binary operators and the conditional operator all group left-to-right, and have priority decreasing as indicated:

binop:

```
    *   /   %
    +   -
    >>  <<
    <   >   <=  >=
    ==  !=
    &
    ^
    |
    &&
    ||
    ?:
```

Assignment operators all have the same priority, and all group right-to-left.

asgnop:

```
    =  +=  -=  *=  /=  %=  >>=  <<=  &=  ^=  |=
```

The comma operator has the lowest priority, and groups left-to-right.

18.2 Declarations

declaration:
  decl-specifiers init-declarator-list$_{opt}$ **;**

decl-specifiers:
  type-specifier decl-specifiers$_{opt}$
  sc-specifier decl-specifiers$_{opt}$

sc-specifier:

```
    auto
    static
    extern
    register
    typedef
```

type-specifier:
  `char`
  `short`
  `int`
  `long`
  `unsigned`
  `float`
  `double`
  struct-or-union-specifier
  typedef-name

init-declarator-list:
  init-declarator
  init-declarator , *init-declarator-list*

init-declarator:
  declarator initializer$_{opt}$

declarator:
  identifier
  (*declarator*)
  `*` *declarator*
  declarator ()
  declarator [*constant-expression*$_{opt}$]

struct-or-union-specifier:
  `struct` { *struct-decl-list* }
  `struct` *identifier* { *struct-decl-list* }
  `struct` *identifier*
  `union` { *struct-decl-list* }
  `union` *identifier* { *struct-decl-list* }
  `union` *identifier*

struct-decl-list:
  struct-declaration
  struct-declaration struct-decl-list

struct-declaration:
  type-specifier struct-declarator-list ;

struct-declarator-list:
  struct-declarator
  struct-declarator , *struct-declarator-list*

struct-declarator:
    declarator
    declarator : *constant-expression*
    : *constant-expression*

initializer:
    = *expression*
    = { *initializer-list* }
    = { *initializer-list* , }

initializer-list:
    expression
    initializer-list , *initializer-list*
    { *initializer-list* }

type-name:
    type-specifier abstract-declarator

abstract-declarator:
    empty
    (*abstract-declarator*)
    * *abstract-declarator*
    abstract-declarator ( )
    abstract-declarator [*constant-expression*$_{opt}$]

typedef-name:
    identifier

18.3 Statements

compound-statement:
    { *declaration-list*$_{opt}$ *statement-list*$_{opt}$ }

declaration-list:
    declaration
    declaration declaration-list

statement-list:
    statement
    statement statement-list

statement:
    compound-statement
    expression `;`
    `if (` *expression* `)` *statement*
    `if (` *expression* `)` *statement* `else` *statement*
    `while (` *expression* `)` *statement*
    `do` *statement* `while (` *expression* `) ;`
    `for (` *expression-1*$_{opt}$ `;` *expression-2*$_{opt}$ `;` *expression-3*$_{opt}$ `)` *statement*
    `switch (` *expression* `)` *statement*
    `case` *constant-expression* `:` *statement*
    `default :` *statement*
    `break ;`
    `continue ;`
    `return ;`
    `return` *expression* `;`
    `goto` *identifier* `;`
    identifier `:` *statement*
    `;`

18.4 External definitions

program:
    external-definition
    external-definition program

external-definition:
    function-definition
    data-definition

function-definition:
    type-specifier$_{opt}$ *function-declarator function-body*

function-declarator:
    declarator `(` *parameter-list*$_{opt}$ `)`

parameter-list:
    identifier
    identifier `,` *parameter-list*

function-body:
    type-decl-list function-statement

function-statement:
    `{` *declaration-list*$_{opt}$ *statement-list* `}`

data-definition:

`extern`$_{opt}$ *type-specifier*$_{opt}$ *init-declarator-list*$_{opt}$;

`static`$_{opt}$ *type-specifier*$_{opt}$ *init-declarator-list*$_{opt}$;

18.5 Preprocessor

`#define` *identifier token-string*
`#define` *identifier*(*identifier* , ... , *identifier*) *token-string*
`#undef` *identifier*
`#include "`*filename*`"`
`#include <`*filename*`>`
`#if` *constant-expression*
`#ifdef` *identifier*
`#ifndef` *identifier*
`#else`
`#endif`
`#line` *constant identifier*

INDEX

+ addition operator 37, 188
& address operator 89, 187
\ backslash character 7, 35, 180
& bitwise AND operator 44, 190
^ bitwise exclusive OR operator 44, 190
| bitwise inclusive OR operator 44, 190
?: conditional expression 47, 191
-- decrement operator 16, 42, 102, 187
/ division operator 10, 37, 188
== equality operator 17, 38, 190
\ escape character 7, 35, 180
> greater than operator 38, 189
>= greater than or equal to operator 38, 189
++ increment operator 16, 42, 102, 187
* indirection operator 89, 187
!= inequality operator 14, 38, 190
<< left shift operator 45, 189
< less than operator 38, 189
<= less than or equal to operator 38, 189
&& logical AND operator 19, 38, 45, 190
! logical negation operator 38, 187
|| logical OR operator 19, 38, 45, 191
% modulus operator 37, 188
* multiplication operator 37, 188
~ one's complement operator 45, 187
>> right shift operator 45, 189
-> structure pointer operator 122, 185
- subtraction operator 37, 188
- unary minus operator 37, 187
\0 null character 27, 35
abstract declarator 199
access mode, file 152, 162
addition operator, + 37, 188
additive operators 188
address arithmetic 3, 96
address of variable 24, 89
address operator, & 89, 187
aggregate 198
alignment by `union` 174
alignment, field 138, 196
alignment restriction 130, 133, 140, 157, 173, 210
`alloc` function 97, 175
allocator, storage 96, 133, 136, 210
ambiguity, `if-else` 52, 201
anachronisms 212
a.out 6, 68
`argc` argument count 110
argument conversion 42
argument, function 6, 23, 186
argument list 6
argument, pointer 95
argument, subarray 95
arguments, command-line 110
arguments, `#define` 87
arguments, variable number of 71
`argv` argument vector 110
arithmetic conversions 39, 184
arithmetic operators 37, 188
arithmetic, pointer 90, 94, 96, 113, 129, 188
arithmetic types 183
array, character 18, 25, 99
array declaration 20, 195
array, explanation of subscripting 93, 210
array initialization 83, 109, 198
array, initialization of two-dimensional 105
array, multi-dimensional 103, 210
array name argument 3, 24, 71, 95, 105, 205
array name, conversion of 94, 185
array of pointers 105
array size, default 84, 109, 124, 195
array, storage order of 104, 210
array subscripts, two-dimensional 104
array, two-dimensional 103
arrays and pointers 93, 110
arrays of structures 123
ASCII character set 17, 35, 40, 127
`asm` keyword 180

assignment, conversion by 41, 191
assignment expression 15, 19, 47, 191
assignment, multiple 19
assignment operators 38, 46, 191
assignment statement, nested 15, 19
assignment suppression, `scanf` 148
associativity of operators 48, 185
`atof` function 69
`atoi` function 39, 58
`auto` storage class 23, 28, 192
automatic variable 23, 28, 72
automatics, initialization of 28, 37, 82, 198
automatics, scope of 76
`\b` backspace character 7
backslash character, `\` 7, 35, 180
backspace character, `\b` 7
`binary` search function 54, 125, 129
binary tree 130
bit fields 136, 196
bit manipulation idioms 137
`bitcount` function 47
bitwise AND operator, `&` 44, 190
bitwise exclusive OR operator, `^` 44, 190
bitwise inclusive OR operator, `|` 44, 190
bitwise operators 44, 190
block 51, 81, 201
block structure 3, 51, 81, 201
blocks, initialization in 201
boundary condition 17, 61, 109
braces 6, 9, 51
braces, position of 10
`break` statement 56, 61, 203
buffered `getchar` 162
buffered input 160
byte 126, 188
`%c` conversion 11
C preprocessor 86
C program verifier (*lint*) 3
calculator program 70, 72, 75, 150
call by reference 3, 24, 91
call by value 3, 24, 71, 91, 186
`calloc` storage allocator 157
`case` prefix 55, 202
cast operator 42, 133, 157, 187, 199
cat program 151, 153, 154
cc command 6, 68, 144
`cfree` 157
`char` type 9, 34, 182, 193
character, `\` escape 35
character array 18, 25, 99
character array initialization 84, 199
character constant 17, 35, 180
character constant, octal 35
character count program 16
character input/output 13
character, negative 183
character pointer 99
character set, ASCII 17, 35, 40, 127
character set, EBCDIC 35, 40
character, signed 40, 182
character string 7, 36, 80, 181
character testing functions 156
character-integer conversion 21, 39, 183
`close` system call 163
coercion operator 42
comma operator 58, 192
command-line arguments 110
comment 8, 179
commutative operators 37, 49, 185
comparison, pointer 98, 189
compilation, separate 3, 65, 76, 205
compiler control lines 207
compiling a C program 6, 23
compiling multiple files 68
compound statement 51, 81, 201
conditional compilation 208
conditional expression, `?:` 47, 191
constant, character 17, 35, 180
constant, `double` 181
constant expression 35, 55, 211
constant, floating 10, 181
constant, hexadecimal 35, 180
constant, integer 10, 180
constant, `long` 35, 180
constant, octal 35, 180
constant, symbolic 12, 14, 18, 86, 127
constants 180
`continue` statement 62, 203
conversion by assignment 41, 191
conversion by `return` 70, 203
conversion, character-integer 21, 39, 183
conversion, `double-float` 42, 184
conversion, `float-double` 41, 69, 184
conversion, floating-integer 42, 184
conversion, format 10
conversion, function argument 42, 186
conversion, input 92
conversion, integer-character 41, 183
conversion, integer-floating 10, 184
conversion, integer-`long` 183
conversion, integer-pointer 188, 210
conversion, integer-`unsigned` 184
conversion, `long`-integer 183
conversion of array name 94, 185
conversion of function 185
conversion operator, explicit 187, 210
conversion, pointer 133, 188, 210
conversion, pointer-integer 102, 184, 188, 210
conversion rules 41
conversion, truth value 41
conversion, `unsigned`-integer 184

conversions, arithmetic 39, 184
copy function 26
copy program 15
cp program 163
creat system call 162
%d conversion 11
data definitions 205
data structure, recursive 131
date conversion 103, 119, 121
day_of_year function 104, 122
declaration, array 20, 195
declaration, extern 30
declaration, field 137, 196
declaration, FILE 151
declaration, implicit 208
declaration, mandatory 8, 36, 76
declaration of external variable 28
declaration of function 194
declaration of function, implicit 68, 186, 209
declaration of pointer 90, 95, 194
declaration, register 81
declaration, static 80
declaration, storage class 192
declaration, structure 119, 196
declaration, syntax of 36
declaration, type 194
declaration, typedef 140, 192, 200
declaration, union 138, 196
declaration vs. definition 76
declarations 192
declarator 193
declarator, abstract 199
decrement operator, -- 16, 42, 102, 187
default array size 84, 109, 124, 195
default initialization 84, 198
default prefix 55, 203
defensive programming 53, 56
#define 12, 86, 207
#define arguments 87
#define, too-long 86
definition, external 28, 204
definition, function 23, 67, 204
definition of external variable 30
dependency, machine 40, 138, 139, 141
derived types 3, 9, 183
descriptor, file 159
directory 169
dir.h directory structure 169
division, integer 10, 37, 212
division operator, / 10, 37, 188
DO loop idiom 57
do statement 3, 59, 202
do-nothing function 67
double constant 181
double quote character 7, 18, 36
double type 9, 16, 34, 193
double-float conversion 42, 184
E notation 34
EBCDIC character set 35, 40
echo program 111
efficiency 47, 65, 81, 85, 108, 127, 133, 143, 160, 176
#else 208
else statement 19
else-if 21, 53
empty function 67
empty statement 16, 204
end of file 14, 144
end of string 27, 35
#endif 208
entry 180
EOF, value of 14, 40, 144
equality operator, == 17, 38, 190
equality operators 38, 190
escape character, \ 7, 35, 180
escape sequence 7, 17, 36, 180
evaluation, order of 19, 37, 45, 49, 50, 59, 75, 87, 91, 185, 212
exit, _exit 154
explicit conversion operator 187, 210
explicit type conversion 42, 133, 157
exponentiation 22
expression 185
expression, assignment 15, 19, 47, 191
expression, constant 35, 55, 211
expression, parenthesized 186
expression, primary 185
expression statement 51, 53, 201
expression, subscript 20
expression, type of 12
expression, unary 187
expressions, order of evaluation of 49, 185
extern declaration 30
extern storage class 192
external data definitions 205
external definition 28, 204
external names, length of 33, 179
external static variables 80
external variable 28, 72
external variable, declaration of 28
external variable, definition of 30
externals, initialization of 37, 72, 77, 82
externals, scope of 76, 206
%f conversion 11, 16
fclose 153
fgets 155
field alignment 138, 196
field declaration 137, 196
field width 137
fields, restrictions on 138, 197
file access 151, 159

file access mode 152, 162
file appending 152, 164
file concatenation 151
file copy program 14, 161
file creation 152, 159
FILE declaration 151
file descriptor 159
file, end of 14, 144
file inclusion 86, 207
file opening 151, 159
file pointer 151, 165
file protection mode 162
file, rewind 164
_fillbuf function 168
float type 9, 34, 193
float-double conversion 41, 69, 184
floating constant 10, 181
floating types 183
floating-integer conversion 42, 184
fopen 151
fopen function 167
for statement 3, 11, 16, 56, 202
formal parameter 23, 81, 95, 204
format conversion 10
formatted input 11, 147
formatted output 10, 145
fortran keyword 180
fprintf 152
fputs 155
free function 97, 177
free-form input 92
fscanf 152
fsize program 171
function argument 6, 23, 186
function argument conversion 42, 186
function call semantics 186
function call syntax 186
function, conversion of 185
function, declaration of 194
function definition 23, 67, 204
function, implicit declaration of 68, 186, 209
function names, length of 33, 179
function, pointer to 114, 141, 209
function return value 23
functions, character testing 156
functions, operations permitted on 209
fundamental types 3, 9, 182
getbits function 45
getc 152
getc macro 166
getch function 79
getchar 13, 144
getchar, buffered 162
getchar macro 152
getchar, unbuffered 161
getchar, value of 40
getint function 93
getline function 26, 67
getop function 78
getword function 127
goto statement 62, 204
greater than operator, > 38, 189
greater than or equal to operator, >= 38, 189
grep program 65, 112
hash function 135
hash table 134
hexadecimal constant 35, 180
#if 208
if statement 17
#ifdef 208
if-else ambiguity 52, 201
if-else statement 3, 19, 51, 201
#ifndef 208
illegal pointer arithmetic 98, 99
implicit declaration 208
implicit declaration of function 68, 186, 209
#include 86, 143, 207
inconsistent type declaration 70
increment operator, ++ 16, 42, 102, 187
indentation 10, 16, 52
index function 67
indirection operator, * 89, 187
inequality operator, != 14, 38, 190
infinite loop 57
infix notation 72
information hiding 65, 66, 73, 75, 81
initialization 37, 198
initialization, array 83, 109, 198
initialization, character array 84, 199
initialization, default 84, 198
initialization in blocks 201
initialization of automatics 28, 37, 82, 198
initialization of externals 37, 72, 77, 82
initialization of registers 82
initialization of statics 37, 82, 198
initialization of structure arrays 124
initialization of two-dimensional array 105
initialization, pointer 97, 129
initialization, structure 120, 121, 198
initializer, permitted form of 211
inode entry 169
input and output, terminal 13, 144, 160
input, buffered 160
input, character 13
input conversion 92
input, formatted 11, 147
input, pushback of 79
input redirection 144
input, unbuffered 160
install function 136

int type 9, 34, 193
integer constant 10, 180
integer-character conversion 41, 183
integer-floating conversion 10, 184
integer-long conversion 183
integer-pointer conversion 188, 210
integer-unsigned conversion 184
integral types 183
Interdata UNIX 2
internal names, length of 33
internal static variables 80
I/O redirection 144, 152, 160
isalpha macro 127, 156
isdigit macro 127, 156
islower macro 156
isspace macro 156
isupper macro 145, 156
itoa function 60
keyword count program 125
keywords, list of 180
label 62, 204
labeled statement 204
%ld conversion 16
leap year computation 37
left shift operator, << 45, 189
length of names 33, 179
length of string 27, 36, 99
less than operator, < 38, 189
less than or equal to operator, <= 38, 189
lexical conventions 179
lexical scope 205
lexicographic sorting 114
library function 6, 7, 65, 76
library, standard 4, 92, 96, 127, 143, 159, 165
#line 208
line count program 17
lint program verifier 3, 50, 68, 103, 141
list directory program 169
list of keywords 180
loading from standard library 144
loading multiple files 68
local variable 28
logical AND operator, && 19, 38, 45, 190
logical expression, value of 41
logical negation operator, ! 38, 187
logical operators 38
logical OR operator, || 19, 38, 45, 191
long constant 35, 180
long type 9, 16, 34, 182, 193
longest line program 26
long-integer conversion 183
lookup function 135
loop, for 11
loop, infinite 57
loop, test at the bottom 59
loop, test at the top 17, 59, 108
loop, while 9
lower case program 145
lower function 40
ls command 169
lseek system call 164
lvalue 183
machine dependency 38, 40, 45, 81, 98, 103, 130, 133, 138, 139, 141, 166, 173, 212
macro preprocessor 86, 207
macros with arguments 87
magic number 12
main function 6
mandatory declaration 8, 36, 76
member name, structure 120, 197
missing storage class specifier 193
missing type specifier 193
modularization 22, 25, 30, 72, 73, 106
modulus operator, % 37, 188
month_day function 104, 123
month_name function 109
morecore function 176
multi-dimensional array 103, 210
multiple assignment 19
multiple files, compiling 68
multiplication operator, * 37, 188
multiplicative operators 188
multi-way decision 21, 53
\n newline character 7, 17
name, symbolic 12
names, length of 33, 179
negative character 40, 183
nested assignment statement 15, 19
nested structure 120
newline character, \n 7, 17
null character, \0 27, 35
NULL pointer 97, 192
null statement 16, 204
null string 36
number of arguments, variable 71
numbers, size of 9, 16, 34, 182
numcmp function 117
numeric sorting 114
%o conversion 11
octal character constant 35
octal constant 35, 180
one's complement operator, ~ 45, 187
open system call 162
operations permitted on functions 209
operations permitted on pointers 99
operations permitted on structures 209
operations permitted on unions 209
operators, additive 188
operators, arithmetic 37, 188
operators, assignment 38, 46, 191

operators, associativity of 48, 185
operators, bitwise 44, 190
operators, commutative 37, 49, 185
operators, equality 38, 190
operators, logical 38
operators, multiplicative 188
operators, precedence of 15, 48, 91, 122, 123, 185, 215
operators, relational 14, 38, 189
operators, shift 44, 189
order of evaluation 19, 37, 45, 49, 50, 59, 75, 87, 91, 185, 212
order of evaluation of expressions 49, 185
origin, subscript 20
output, character 13
output, formatted 10, 145
output redirection 144
overflow 38, 185
parameter, formal 23, 81, 95, 204
parenthesized expression 186
pattern finding program 66, 112, 113
PDP-11 UNIX 2
permitted form of initializer 211
permitted on functions, operations 209
permitted on structures, operations 209
permitted on unions, operations 209
✓ pipe 144, 160
pointer argument 95
pointer arithmetic 90, 94, 96, 113, 129, 188
pointer arithmetic, illegal 98, 99
pointer arithmetic, scaling in 94, 98
pointer, character 99
pointer comparison 98, 189
pointer conversion 133, 188, 210
pointer, declaration of 90, 95, 194
pointer, file 151, 165
pointer initialization 97, 129
pointer, `NULL` 97, 192
pointer subtraction 98
pointer to function 114, 141, 209
pointer to structure 128
pointer-integer conversion 102, 184, 188, 210
pointers 3, 24
pointers and arrays 93, 110
pointers and subscripts 93
pointers, array of 105
pointers, operations permitted on 99
Polish notation 72
`pop` function 75
portability 2, 3, 35, 40, 45, 71, 103, 139, 141, 143, 145, 166, 173, 212
position of braces 10
postfix ++ and -- 42, 101, 187
`power` function 23, 24
precedence of operators 15, 48, 91, 122, 123, 185, 215
prefix ++ and -- 42, 102, 187
preprocessor, macro 86, 207
primary expression 185
`printd` function 85
`printf` 7, 11, 145
program arguments 110
program, calculator 70, 72, 75, 150
program, *cat* 151, 153, 154
program, character count 16
program, copy 15
program, *cp* 163
program, `echo` 111
program, file copy 14, 161
program format 10, 16, 130, 179
program, *fsize* 171
program, *grep* 65, 112
program, keyword count 125
program, line count 17
program, list directory 169
program, longest line 26
program, lower case 145
program, pattern finding 66, 112, 113
program readability 19, 39, 57, 60, 61, 85
program, sorting 106, 115
program, table lookup 134
program, temperature conversion 8
program, trailing blanks 61
program verifier, *lint* 3, 50, 68, 103, 141
program, white space count 20, 55
program, word count 18, 130
protection mode, file 162
`push` function 75
pushback of input 79
`putc` 152
`putc` macro 166
`putchar` 13, 144
`putchar` macro 152
`read` system call 160
readability, program 10, 15, 19, 33, 39, 47, 52, 57, 60, 61, 83, 85, 141
`readlines` function 107
recursion 3, 84, 131, 132, 186
recursive data structure 131
✓ redirection of I/O 144
`register` declaration 81
`register` storage class 81, 192
registers, initialization of 82
relational expression, value of 41
relational operators 14, 38, 189
reserved words 33, 180
restrictions on fields 138, 197
restrictions on registers 212
restrictions on structures 121, 209
restrictions on types 195
restrictions on unions 140, 209
`return`, conversion by 70

return statement 23, 68, 70, 203
return, type conversion by 70, 203
reverse function 59
reverse Polish notation 72
rewind file 164
right shift operator, >> 45, 189
rules, scope 76, 205
rules, type conversion 41, 184
%s conversion 11, 27
sbrk system call 175
scaling in pointer arithmetic 94, 98
scanf 147
scanf assignment suppression 148
scientific notation 34
scope of automatics 76
scope of externals 76, 206
scope rules 76, 205
seek system call 164
self-referential structure 131, 197
semantics, structure reference 186
semicolon 9, 13, 16, 51, 53
separate compilation 3, 65, 76, 205
sequencing of statements 201
Shell sort 58, 108
shift operators 44, 189
short type 9, 34, 182, 193
side effects 50, 87, 185
sign extension 40, 41, 166, 212
signed character 40, 182
single quote character 17
size of numbers 9, 16, 34, 182
sizeof operator 126, 187
sort function 58, 108, 116
sorting, lexicographic 114
sorting, numeric 114
sorting program 106, 115
sorting text lines 105
specifier, missing storage class 193
specifier, missing type 193
specifier, storage class 192
specifier, type 193
sprintf 150
squeeze function 43
squirrel 131
sscanf 150
standard error 160
standard input 144, 160
standard library 4, 92, 96, 127, 143, 159, 165
standard library, loading from 144
standard output 144, 160
stat definitions 171
stat structure 170
stat system call 170
statement grouping 3, 51
statement terminator 9, 51
statements 201
statements, sequencing of 201
stat.h 170
static declaration 80
static storage class 28, 80, 192
static variables, external 80
static variables, internal 80
statics, initialization of 37, 82, 198
stderr 152, 154
stdin 152
stdio.h contents 166
stdio.h header file 143
stdout 152
storage allocator 96, 133, 136, 210
storage allocator, calloc 157
storage class, auto 23, 28, 192
storage class declaration 192
storage class, extern 192
storage class, register 81, 192
storage class specifier 192
storage class specifier, missing 193
storage class, static 28, 80, 192
storage order of array 104, 210
strcat function 44
strcmp function 101
strcpy function 100
string constant 7, 18, 35, 99, 181
string, end of 27, 35
string, length of 27, 36, 99
string, type of 186
strlen function 36, 95, 98
strsave function 103
structure arrays, initialization of 124
structure declaration 119, 196
structure initialization 120, 121, 198
structure member name 120, 197
structure member operator, . 120, 185
structure, nested 120
structure pointer operator, -> 122, 185
structure, pointer to 128
structure reference semantics 186
structure reference syntax 186
structure, self-referential 131, 197
structure tag 119, 197
structures, arrays of 123
structures, operations permitted on 209
structures, restrictions on 121, 209
subarray argument 95
subscript expression 20
subscript origin 20
subscripting, explanation of 93, 210
subscripts and pointers 93
subtraction operator, - 37, 188
subtraction, pointer 98
swap function 92, 117
switch statement 3, 22, 54, 73, 202

symbolic constant 12, 14, 18, 86, 127
symbolic constants, length of 33
syntax notation 182
syntax of declaration 36
syntax of variable names 33
syntax, structure reference 186
syntax summary 214
system 157
tab character 7
table lookup program 134
tag, structure 119, 197
tag, union 197
temperature conversion program 8
terminal input and output 13, 144, 160
termination, program 153, 154
test at the bottom loop 59
test at the top loop 17, 59, 108
text lines, sorting 105
token replacement 207
tolower macro 145, 156
too-long **#define** 86
toupper macro 156
trailing blanks program 61
tree, binary 130
tree function 132
treeprint function 133
truncation 10, 37, 70
truth value conversion 41
two-dimensional array 103
two-dimensional array, initialization of 105
two-dimensional array subscripts 104
type, **char** 9, 34, 182, 193
type checking 3, 133
type conversion by **return** 70, 203
type conversion, explicit 42, 133, 157
type conversion operator 42
type conversion rules 41, 184
type conversions 39
type declaration 194
type declaration, inconsistent 70
type, **double** 9, 16, 34, 193
type, **float** 9, 34, 193
type function 127
type, **int** 9, 34, 193
type, **long** 9, 16, 34, 182, 193
type mismatch 70
type names 199
type of a variable 8
type of expression 12
type of string 186
type, **short** 9, 34, 182, 193
type specifier 193
type specifier, missing 193
type, **unsigned** 34, 45, 183, 193
typedef declaration 140, 192, 200
typedef, use of 173
typeless language 2
types, arithmetic 183
types, derived 3, 9, 183
types, floating 183
types, fundamental 3, 9, 182
types, integral 183
types, restrictions on 195
types.h 170
unary expression 187
unary minus operator, - 37, 187
unbuffered **getchar** 161
unbuffered input 160
#undef 207
underflow 38
underscore character 33, 179
ungetc 156
ungetch function 79
union, alignment by 174
union declaration 138, 196
union tag 197
unions, operations permitted on 209
unions, restrictions on 140, 209
UNIX file system 159, 169
UNIX, Interdata 2
UNIX operating system 2, 4
UNIX, PDP-11 2
unlink system call 163
unsigned type 34, 45, 183, 193
unsigned-integer conversion 184
use of **typedef** 173
value of **EOF** 14, 40, 144
value of **getchar** 40
value of logical expression 41
value of relational expression 41
variable, address of 24, 89
variable, automatic 23, 28, 72
variable, external 28, 72
variable names, length of 33
variable names, syntax of 33
variable number of arguments 71
while statement 3, 9, 56, 202
white space count program 20, 55
word count program 18, 130
write system call 160
writelines function 107, 108
%x conversion 11
zero, omitted test against 52, 99, 101

547261