UNPL 1498

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: Recent Changes to C

Date: October 2, 1979

Other Keywords: programming languages
UNIX(TM)
C

TM: 79-3621-2

Author(s)	Location	Extension
B. R. Rowland	IH	4140

Charging Case: 49461-60
Filing Case: 40125

ABSTRACT

The C programming language is currently in widespread use across Bell Laboratories. It is the primary programming language (along with FORTRAN 77) on computers using the UNIX(TM) operating system and is available on many other general purpose computers such as the IBM System/370 with TSS and OS, the Intel 8086, and Honeywell HIS-6080 with GCOS. C is implemented for several of the processors produced by the Bell System including MAC-8 and 3B-20. C is a language with a flexible variety of both control and data structures as well as low level data access primitives. Recently C has evolved to meet new Bell Laboratory needs.

This memo describes recent enhancements to the C language that are not currently documented. These include:

- structure assignment
- structure valued functions
- structure valued parameters
- enumerations
- non-unique structure and union members
- fully qualified structure and union references

Examples of all the above are given.

Pages Text: 8 Other: 1 Total: 9

No. Figures: 0 No. Tables: 0 No. Refs.: 8

BELL TELEPHONE LABORATORIES, INC. TM-79-3621-2

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES
OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

AAGESEN,JOHN
ABATE,JOSEPH
AITCHESON,E J
ALBAGLI,V E
ANDREWS,W J
APPELBAUM,MATTHEW A
ARCHER,RUSSELL E,JR
ARMAN,THOMAS D
ARVIDSON,W P
BABU,RAJESH RATLAL
BAKER,MITCHELL B
BALENSON,CHRISTINE M
BALLANCE,ROBERT A
BAROPSKY,ALLEN
BARON,ROBERT V
BARR,DAVID L
BARTON,M E
BAUER,BARBARA T
BAYER,D L
BECKETT,J T
BERNHARDT,RICHARD C
BIREN,IRMA B
BITTRICH,MARY E
BLAZIER,S D
BLECHMAN,RONALD I
BLEIER,JOSEF
BLINN,J C
BLUM,MARION
BOEHM,EARL W
BOEHM,KIM E
BOGART,THOMAS G
BOUMA,HERMAN J
BOURNE,STEPHEN R
BRADLEY,M HELEN
BRAUN,DAVID A
BRENSKI,EDWIN F
BRITT,WARREN D
BROSS,JEFFREY D
BROWN,ELLINGTON L
BROWN,W R
BURNEI,ROSE M
BUTLETT,DARRELL L
BYORICK,ROBERT S
CAMPBELL,JERRY H
CANADAY,RUDD H
CARRAN,JOHN H
CARR,DAVID C
CHAFFEE,N F

COMPLETE MEMORANDUM TO

CHANG,HERBERT Y
CHANG,JO-MEI
CHARLES,JOYCE
CHELLIS,ALICIA L
CHEN,ROBERT
CHODROW,M M
CHOU,PAO-LO P
CHRISTENSON,DENNIS A
CICHINSKI,STEVEN
CLARK,C A
CLIFFORD,COURTENAY B
CLINE,LAUREL M I
CLINE,TERRY W
COHEN,DAVID
COHEN,HARVEY S
COLEMAN,ELANE
CONKLIN,DANIEL L
COOK,DIANA
COOK,T J
COWELL,ARLINE C
CRISTOFOR,EUGENE
CROWE,MARGARET M
CSASZAR,MARYANN
CSURI,JOHN O
DAVIDSON,CHARLES LEWIS
DAVIDSON,ROBERT F
DAVIS,D R
DAVIS,E DREW
DE GRAAF,D A
DE JAGER,D S
DE TREVILLE,JOHN D
DEAN,JEFFREY S
DENNY,MICHAEL S
DIE,GILBERT
DICKMAN,BERNARD N
DIESEL,MICHAEL E
DIXON,DAVID A
DOEDLINE,BARBARA ANN
DOLOTTA,T A
DONOHOE,D C
DOWDEN,DOUGLAS C
DOWD,PATRICK G
DUBMAN,M R
DUCHARME,ROBERT LAWRENCE
DUNLOP,ALFRED E
DWORAK,F S
DWORAK,MARY F
DWYER,T J
EISEN,STEVEN R
ELDUMIATI,I I
EPLEY,ROBERT V
ERWIN,W J
EVENSON,E K
FARLOW,COLON W
FAVIN,D L
FEAY,MARY R
FEINBERG;HENRY R
FENG,FRANK H
FISCHER,HERBERT B
FISHMAN,DANIEL H

COMPLETE MEMORANDUM TO

FITTON,MICHAEL J
FLAMHOLZ,JACK
FLANDRENA,B
FLEISLEBER,RAYMOND C
FLEMING,JAMES R
FONG,K T
FOO,YEOW PIN
FORTNEY,V J
FOUGHT,B T
FOY,J C
FREEMAN,K G
FREEMAN,MARTIN
GALE,WILLIAM A
GALLANT,R J
GANGAWARE,BERNICE C
GEORGEN,MICHAEL R
GEPNER,JAMES R
GIBSON,H T,JR
GILBERT,DAVID G
GLASSER,ALAN L
GLAZER,STANLEY
GOLDSTEIN,A JAY
GOODNOW,JAMES E,II
GORDON,MOSHE B
GROSS,ARTHUR G
GUERCIO,A M
GUIDI,PIER V
HAIGHT,R C
HALLER,N M
HALLIN,THOMAS G
HALPIN,T
HAMES,ROSALYN
HAMM,DEBORAH ANN
HANSEN,R C
HANSON,BRUCE L
HARKNESS,CAROL J
HARRISON,JAMES FRANCIS
HARR,JOHN A
HARTOIN,ROBERT E
HARTWELL,STEVEN
HAYES,MARILYN E
HAYWARD,HENRIETTA M
HECHT,MATTHEW S
HEINY,WILLIAM C
HESSELGRAVE,MARY E
HINES,JOHN NED
HOERL,DAVID F
HO,DON T
HO,TIEN-LIN
HSU,TAU
HUBER,RICHARD V
HWANG,HENRY
IMAGNA,C P
ISMAN,MARSHALL A
JABLONSKI,GRAZYNA C
JENKINS,JERRY
+JOHNSON,STEPHEN C
KAHN,MORRIS S
KANE,J RICHARD
KANODIA,RAJENDRA K

COMPLETE MEMORANDUM TO

KAPLAN,ROBERT S
KATZ,MARILYN
KAUFMAN,ANN S
KAYEL,R G
>KERNIGHAN,BRIAN W
KEVORKIAN,D E
KILGORE,D
KINCAID,B M
KIRCHHOFF,LELAND W
KLAPMAN,RICHARD N
KOENIG,ANDREW R
KOLOR,RICHARD W
KOWALSKI,THADDEUS J
KOZAR,MARY ANN
KOZLOW,JAN D
KURAS,JOHN E
KU,VICKI P
KWONG,TONY C W,JR
LAMB,J ELI
LAUR,J E
LAUTZENBACH,DEBORAH A
LAU,EDWIN J
LAWRENCE,WATSON A,JR
LAYTON,H J,JR
LEDFORD,RANDALL D
LEEPER,DAVID G
LEEPER,EVELYN C
LEEPER,MARK R
LEONARDO,MILDRED
LESCINSKY,F W
LESSEK,P V
LETH,JAMES W
LEVENBERG,R A
LEVIE,IRMELIN G
LEVIE,STERLING L,JR
LIAW,Y S
LIEBERMAN,ARTHUR Z
LIEN,Y EDMUND
LINDSEY,CHRIS P
LIND,R O
LOOTS,ROGER W
LOUDEN,T MICHAEL
LOW,GEORGE
L'HOMMEDIEU,CARMELA
LUKACS,M E
LUND,JOHN C
LUTZ,KENNETH J
MAC DONALD,ANTONINA H
MACCHIONE,WILLIAM
MADOR,H P
MARANTZ,LYN
MARANZANO,J F
MARSH,ROBERT L
MARTELLOTTO,N A
MARTIN,JOHN L
MARTIN,PATRICIA D
MAST,C A
MC CABE,P S
MC CARTHY,JOAN T
MC DONALD,H S

COMPLETE MEMORANDUM TO

MC ELROY,J D
MCCONNELL,RONALD C
MCLEAN,RICHARD H
MEE,C,III
MERRICK,THOMAS B
MESSINGER,ARMIDA J
MEYER,BRIAN C
MICHAUD,D A
MILLER,JO ANNE H
MILLER,L D
MILLS,ARLINE D
MILNE,D C
MITCHELL,WILLIAM J
>MITZE,ROBERT W
MONTEMAYOR,ROGER
MORGAN,DENNIS J
MROZ,WAYNE F
MUELLER,MARK B
MUHA,RALPH
MULLER,R ALLAN
MURPHY,LAWRENCE E
MUSEN,ROBERT M
NEY,LESLIE
NIEDFELDT,B G
NIPPERT,JAMES W
NOLAN,JULIANNE
NOWITZ,D A
O KANE,J J
O SHEA,W T
OH,RICHARD YOUNG
ORSHAN,SCOTT
O'BRYAN,H M,JR
O'CONNOR,JOHN J
OTT,PETER A
PAO,T W
PAPATHOMAS,THOMAS V
PASCHBURG,RICHARD H
PAULE,W JOSEPH
PEARSON,MARK
PECK,W DOUGLAS
PELLEGRIN,J F
PETERSON,RALPH W
PHALEN,GEORGE B
PHILLIPS,JOANNA
PIERCE,BETSY L
PILLA,M A
PLADEK,ROBERT B
POLLI,PHILIP V
POND,E W
PRINS,G C
PUCCI,M F
PUERLING,BRUCE W
PURZYCKI,MICHAEL J
PUTTRESS,JOHN J
RAMSEY,DAVID A
RATTI,P A
+REED,R A
REGELSON,KENNETH
REICHERT,W G,JR
REILLY,J W
...
395 TOTAL

+ NAMED BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION.....

COMPLETE MEMO TO:
3641-SUP 3642-SUP 3643-SUP 3644-SUP 3645-SUP 3646-SUP

UNPLCL = C LANGUAGE

COVER SHEET TO:
COPLGF = COMPUTING/PROGRAMMING LANGUAGES/GENERAL PURPOSE
UNGI# = GENERAL INFORMATION (DESCRIPTION,PROPOSALS,BEGINNERS INFORMATION,

---

NO CORRESPONDENCE FILES TM-79-3621-2
NO [illegible] 14127 TOTAL PAGES 10

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.
4. INDICATE WHETHER MICROFICHE OR PAPER IS DESIRED.

PLEASE SEND A COMPLETE

( ) MICROFICHE COPY ( ) PAPER COPY

TO THE ADDRESS SHOWN ON THE OTHER SIDE.

Bell Laboratories

subject: Recent Changes to C
Case: 49461-60
File: 40125

date: October 2, 1979

from: B. R. Rowland
IH 3621
x4140

3621-791002.02TM

TM: 79-3621-2

1. INTRODUCTION

The C programming language has been successfully used in systems programming as well as general purpose programming environments across Bell Laboratories on a wide variety of computers and stored program processors. Among these machines are the PDP-11 series, IBM/360-370 series, VAX 11/780, UNIVAC 1100 series, Honeywell HIS-6080, Interdata 832, Intel 8086, 3B Model 20, and MAC-8. This success stems partly from its flexible control and data structuring and low level data accessing primitives but is also due to a large measure because C compilers have proved relatively easy to port to new machines and the C language is fairly simple to learn.

The definition of the C language has evolved as C is becoming more widely used in Bell Laboratories. In response to changing needs and requirements of C programmers (as summarized in [Row 79b]), a few extensions have been made to the C language beyond what is described in the reference document ("The C Programming Language," Kernighan and Ritchie, Prentice-Hall, 1978). The extensions consist primarily of changes in the use of structures and unions, and the introduction of a new data type, enumeration. Along with descriptions of the syntactic and semantic changes to the C language, examples of the new features are illustrated.

As the C language changes, the compilers that implement the language have been tracking the changes. For the most part this involves the UNIX* system C compiler for the PDP-11 maintained by D. M. Ritchie and the Portable C Compiler [Joh 78] and Lint [Joh 77] developed by S. C. Johnson for which design control has been transferred to Department 3621. The changes described in this memorandum have been incorporated into these compilers. Other existing C compilers should follow suit by picking up the changes from the C compiler each is based upon.

---

* UNIX is a trademark of Bell Laboratories.

2. NEW FEATURES

2.1 Structure assignment

Structure assignment has been added to the C language to simplify both the source and object code associated with transferring the value of one structure instance to another and to allow functions to return aggregate values when invoked. Since many processors now contain some type of `move block' instruction, structure assignment will permit more efficient use of many machines. It also makes source programs more readable.

Structures may be assigned, passed as arguments to functions, and returned by functions. The types of structure operands taking part must be the same. Other plausible operators, such as equality comparison and structure casts, are not being implemented due to the difficulties associated with "holes" in structures caused by alignment restrictions.

The following code demonstrates the new structure assignment features.

```
struct clock{
        int hour, minute, second;
        };
struct date{
        int year, month, day;
        struct clock time;
        };
struct clock now={13,2,36};
extern struct date spring();
struct date today, tomorrow;

struct date nextday( day ) struct date day;{
        struct date tempday;
        ...
        return tempday;
        }

main(){
        today = spring();
        tomorrow = nextday( today );
        tomorrow.time = now;
        ...
        }
```

There is a subtle defect in the PDP-11 and VAX 11/780 implementations of functions that return structures: if an interrupt occurs during the return sequence, and the same function is called reentrantly during the interrupt, the value returned from the first call may be corrupted. The problem can occur only in the presence of true interrupts, as in an operating system or a user program that makes significant use of signals; ordinary

recursive calls are quite safe. This same defect is not present in the Basic-16 [Hei 79], IBM 370 [Row 79a] or 3B C compiler [Mit 78] implementations.

2.2 Enumeration Type

There is a new C data type analogous to the scalar types of Pascal [Wir 71]. Enumerations are unique types with named constants. They serve to replace in part the use of #define'd constants in C, but they offer the additional advantage of scoped constant names and strong type checking in the use of such names.

To the type-specifiers in the syntax on p. 193 of the C book add

enum-specifier

with syntax

enum-specifier:
enum { enum-list }
enum identifier { enum-list }
enum identifier

enum-list:
enumerator
enum-list , enumerator

enumerator:
identifier
identifier = constant-expression

The role of the identifier in the enum-specifier is entirely analogous to that of the structure tag in a struct-specifier; it names a particular enumeration. For example,

```
enum color { chartreuse, burgundy, claret, winedark };
...
enum color *cp, col;
...
col = claret;
cp = & col;
...
if( *cp == burgundy )...
```

makes `color' the enumeration-tag of a type describing various colors, and then declares `cp' as a pointer to an object of that type, and `col' as an object of that type.

The identifiers in the enum-list are declared as constants, and may appear wherever constants are required. If no enumerators with "=" appear, then the values of the constants begin at 0 and increase by 1 as the declaration is read from left to right. An enumerator with "=" gives the associated identifier the value

indicated; subsequent identifiers continue the progression from the assigned value.

```
enum interrupt{
        halt = 0,
        bad_instr = 01001,
        mem_fault,
        div_zero = 02001,
        overflow,
        underflow
        } icode;
...
if( (int)icode & 02000 )/* arithmetic fault */
...
```

The previous example illustrates specific enumeration value specification. In particular, the symbol `overflow' has internal value 02002.

Enumeration constants must all be distinct, and, unlike structure members, are drawn from the same set as ordinary identifiers.

Objects of a given enumeration type are regarded as having a type distinct from objects of all other types, and Lint flags type mismatches. In the PDP-11 implementation, all enumeration variables are treated as if they were int . Portable C Compiler implementations map enumerations into a convenient storage unit (char, short, or int) depending on the values associated with the enumeration constants.

2.3 Non-Unique Structure Member Names

The C language has been changed in a nearly upwards compatible fashion to allow more flexibility in the reuse of structure member and structure field names. The obscure case in which upwards compatibility is not maintained is explained in detail at the end of this section. This enhancement permits more natural structure and union member naming conventions in C programs and results in stronger type checking of both structure and union member references.

2.3.1 Former member name restrictions. Prior to this change, there were only two ways in which structure member names could be reused.

A. Member names of two distinct structures declared at any block levels (including different block levels) that represented the same member type and offset could be identical. For example, the name `xyz' is used in both of the following two structures:

```
struct s1{
        long abc;
        char xyz;
        float def;
        };
struct s2{
        long abc;
        char xyz;
        short jkl;
        };
```

With such a construction, the structure member name `xyz' could be referenced from any structure variable or any pointer without ambiguity.

B. Member names could be reused within a new name scoping (block) level. In the following code section, the member name `f_one' is reused:

```
struct outer{
        int f_zero:2,
            f_one:4,
            f_two:10;
        struct outer *next;
        };

function(){
        struct inner{
                int f_one, g_one, h_one;
                };
        ...
        }
```

When member names are redeclared at different block levels, the innermost declaration serves to block the outer declarations of the same name within the inner scope. In the previous example, the four-bit field `f_one' could not be referenced (even from structures that are explicitly declared to be type `outer') within the function `function'.

2.3.2 New flexibility for member names. The language change for structure member names allows the reuse or redeclaration of structure member or field names with only a single restriction:

1. A particular name may not be used for two distinct members within the same structure. (However, a name may be reused within nested structures.)

The impact of this change is stronger type checking for structures and unions. Call a structure (or union) member unique if it is declared only once, or if all its declarations conform to the requirements of case A above. If a uniquely-named member is mentioned in a structure reference in which it is not a member of

the structure, a warning diagnostic is issued. This allows old C programs that violate the language rules to continue to compile. However, if a member that is not uniquely named is used in a structure reference in which it is not a member of the structure, a fatal diagnostic is issued.

The case in which upwards compatibility is not maintained involves structure member name redeclarations of type (B) described above.

```
struct x{
        int a,b;
        } x_obj;

main(){
        int *ip;
        struct {
                int b,a;
                } y_obj;

        ... ip->a ...
        ... y_obj.a ...
        ... x_obj.a ...
        }
```

In the example above, prior to the language change, each of the references `ip->a', `y_obj.a', and `x_obj.a' were considered legitimate, and an offset of two bytes (on a sixteen-bit processor, such as the PDP-11) for the integer referenced by `a' was used. With non-unique structure members, the integer referenced by `a' in `x_obj.a' would have an offset of zero bytes from the address of `x_obj'. The reference `ip->a' could either be considered a user error by a particular compiler or a warning could be issued and the innermost declaration of `a' could be used to resolve the reference. Because of the lack of existing code with such potential ambiguities for most PCC compiler instances, a fatal diagnostic will be issued by the PCC for `ip->a'.

2.4 Complete Structure/Union Member Reference Qualifications

In past C compiler instances, a reference to a structure or a union member could be abbreviated in some cases. A structure or union member reference is a chain of member references (qualifications) that are prefixed by either a pointer to a structure or union or a structure or union proper. Since each qualification implies the addition of an offset within an address computation, it was possible in the past to omit those qualifications that had an offset of zero. Zero offsets occur in the first member of a structure and in all members of unions. With the two following declarations:

```
struct xx{
        struct yy{
                int y1; char y2;
                } ym;
        ...
        } *xp;

union u{
        struct a{
                int a1,a2,a3;
                } mema;
        struct b{
                char b1,b2,b3;
                } memb;
        } *up;
```

the following references were allowed:

```
xp->y2      /* same as */      xp->ym.y2

up->b2      /* same as */      up->memb.b2
```

Due to the ambiguities that can arise with incomplete qualifications and non-unique structure and union member names, complete qualifications are now required for structure and union member references in the C language. This change also serves to enforce stronger type checking of structure and structure pointer use within C. At the present time, incomplete qualifications will be flagged with user warning messages. Lint, run in its heuristic mode, will suggest how to complete an incomplete qualification. Union members that are structures must be named, so that complete qualifications can be constructed.

Of the references in the previous example, only the following structure and union member references are now legitimate:

```
xp->ym.y2

up->memb.b2
```

2.5 Tag Names

Structure, union, and enumeration tag names are the names associated with a declared type and always appear after the keywords struct, union, and enum, as in the following examples:

```
typedef enum bool {false, true} bool;
struct list *head;
union cell {unsigned word; char byte[2];};
```

Previous implementations of C required that all structure and union tag names be distinct from structure and union member

names. This restriction has been removed from the C language. As a result, four name pools now exist:

I. # define'd macro names
(Processed separately by /lib/cpp.)

II. structure, union, and enumeration tag names

III. structure and union members
(These may be non-unique.)

IV. all other names
(Includes: typedef names; array, structure instance, and variable names; and enumeration constant names.)

2.6 Vertical Tab Character Literal

A new character literal has been added to the C language. The vertical tab character (VT, octal 013 in ASCII and EBCDIC) can now be represented as '\v' in addition to '\013'. This character can also be used within character string literals (eg.: "Upper left\t\t\t\v\vLower right\n"). Vertical tab is now included in the definition of white space and thus can be used to delimit tokens in a C source file.

3. SUMMARY

A significant number of changes to the C language have occurred since the last release of the C reference manual [KR 77]. The changes affect mainly the use of structures and unions and the naming restrictions in the language. Cooperative efforts among C compiler and Lint implementors are leading to coordinated releases of these compilation tools with new language features.

4. ACKNOWLEGEMENTS

The language changes described in this memorandum are a result of language design work performed by Dennis Ritchie and Steve Johnson. A nontrivial effort was involved in reviewing a host of requested language enhancements and selecting and refining those that were compatible with the nature of the C language, implementable in existing tools, and compatible with nearly all existing code written in C.

IH-3621-BRR-mm

B. R. Rowland

Att.
References

REFERENCES

[Hei 79] W. C. Heiny. "Basic 16 C Compiler Implementation," Memorandum for File, 3243-790521-01MF, (May 24, 1979).

[Joh 77] S. C. Johnson. "Lint, a C Program Checker," Technical Memorandum, TM-77-1273-14 (September 16,1977).

[Joh 78] S. C. Johnson. "A Portable Compiler: Theory and Practice," Conference Record of the Fifth Annual ACM Conference on Principles of Programming Languages, Tucson, AZ (January 23, 1978) 97-104.

[KR 78] B. W. Kernighan and D. M. Ritchie. The C Programming Language, Prentice-Hall, Inc., Englewood Cliffs, NJ (1978).

[Mit 79] R. W. Mitze. "An Overview of C Compilation of UNIX User Processes on the 3B," Memorandum for File, 5521-780329.02MF (March 29, 1978).

[Row 79a] B. R. Rowland. "Status Report for IBM 370 C Compiler and its Indian Hill TSS Implementation," Memorandum for File, 2521-790201.01MF (February 1, 1979).

[Row 79b] B. R. Rowland. "C Language Enhancements: Laboratory 252 Recommendations," Memorandum for File, 2521-790305.02MF (March 5, 1979).

[Wir 71] N. Wirth. "The Programming Language PASCAL," Acta Informatica 1, 1 (1971) 35-63.