Bell Laboratories 1467

subject: Changes to C and the Portable C Compiler -- Issue 1
File: 40125

date: October 2, 1979

from: B. R. Rowland
IH 3621
4F-431 x4140

3621-791002.01MF

ABSTRACT

This document is the first in what will become a series of notifications to implementors of a version of the Portable C Compiler in Bell Laboratories. The release of documents in this series will coincide with releases of the Portable C Compiler (PCC) machine independent parts (mip) and will serve to identify:

- C language changes
- PCC mip source changes

C language changes will be described to clarify PCC mip changes for the benefit of PCC implementors. This series of memoranda is not intended to supplant the timely release of documented C language changes that should interest a far greater audience. PCC changes will be described in terms of their impact on both programmers and compiler administrators.

Issue 1 changes include a major language change -- non-unique structure member names and fully qualified member references, a minor language change -- a new character literal and several error corrections and implementor feature enhancements.

Copy to
All Supervision Laboratory 362
All Members Languages and Compilers Group (3621)
A. D. Berenbaum
J. A. Deutsch
F. S. Dworak
J. W. Farrell
S. I. Feldman
J. F. Gimpel
Continued next page

THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION OF BELL TELEPHONE LABORATORIES AND IS NOT TO BE REPRODUCED OR PUBLISHED WITHOUT BELL LABORATORIES APPROVAL

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

Copy to cont'd
S. C. Johnson
M. J. Melchner
T. B. London
G. G. Riddle
D. M. Ritchie
G. Ronkin
L. Rosler
S. Rugaber
J. M. Scanlon
M. J. Y. Williams

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

Bell Laboratories

subject: Changes to C and the Portable C Compiler -- Issue 1
File: 40125

date: October 2, 1979

from: B. R. Rowland
IH 3621
4F-431 x4140

3621-791002.01MF

MEMORANDUM FOR FILE

1. INTRODUCTION

The Portable C Compiler [Joh 78] is a framework that can be used to implement a C compiler for wide class of target processors. While the resultant compiler is often used in a cross compilation environment on a PDP-11 with the UNIX* operating system, it has been successfully bootstrapped and installed on a target machine in a number of cases. The compiler expects source that is preprocessed by the C preprocessor (/lib/cpp), and it has been constructed to deliver assembly language as output. Supplied with the machine independent portions of the Portable C Compiler (PCC) is a lexical analyzer, parser, symbol table building routines, expression tree generator, and code generation mechanisms that run off machine dependent tables and routines.

The PCC and Lint [Joh 77] are being supported by Department 3621 subsequent to their development by S. C. Johnson in Center 127. An attempt is being made to keep PCC based compilers and Lint synchronized with the UNIX C compiler (cc) on the PDP-11 implemented by D. M. Ritchie. For the most part, this synchronization can be accomplished in the machine independent part of PCC.

To simplify the distribution of C language changes and PCC improvements, official release points will be established for the machine independent parts (mip) of the PCC. Each release of the mip will be accompanied by documentation describing changes to PCC that result from C language changes, PCC error fixes, and PCC functional enhancements. This memorandum documents the first such release of the PCC by Department 3621. The release is named PCC/362r0. For comparison purposes we will term the final two pass compiler release by S. C. Johnson as PCC/127r0 and his final one pass release as PCC/127r1.

Changes are described in terms of both compiler user impact and compiler implementor impact. Compiler implementor impact

* UNIX is a trademark of Bell Laboratories.

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

addresses work that must be performed by those maintaining a PCC instance to incorporate a new feature or to integrate an algorithm or data structure change.

2. C LANGUAGE CHANGES

Several C language changes have occurred that necessitated Portable C Compiler changes. This section describes briefly the language changes and the PCC mip changes to implement them. A more detailed discussion of these changes to the C language is contained in [Row 79].

2.1 Non-unique Structure Member Names

Structure and union member names are now permitted to be non-unique. This change allows member names to be redeclared at the same or different block levels with different offsets and types. When member names are truly non-unique (not just same offset and type in different structures), their usage will be type checked by PCC compilers. If a member of a structure or union is referenced from a structure or union of which it is not a member, a fatal diagnostic is given. Only warning diagnostics will be given for the misuse of unique member names so that old C programs will continue to compile.

The majority of the changes for this language feature occur in the function defid() in pftn.c and buildtree() in trees.c. A new flag has been added to the flag member of a symbol table entry, SNONUNIQ. This flag is set in any member name (already marked with SMOS) that is non-unique.

2.2 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.

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. At the present time, incomplete qualifications will be flagged with user warning messages. Union members that are structures must be named so that complete qualifications can be constructed.

This change has been implemented in PCC simply by removing a recursive call in chkstr() in trees.c that allowed one to drop a

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

zero offset member name in a qualified reference.

2.3 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, three name pools now exist in the PCC:

I. structure, union, and enumeration tag names

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

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

A new symbol table flag, STAG, and symbol table wart, TAGNAME, have been added to the PCC for use in symbol table lookup routines. These flags affect routines lookup(), hide(), unhide(), and defid() in pftn.c and routines lxres() and yylex() in scan.c as well as two actions in cgram.y.

2.4 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"). The change was implemented in yylex() of scan.c.

3. PORTABLE C COMPILER CHANGES

A number of PCC changes have occurred for either of two reasons:

a. a PCC error was corrected, or

b. an improved capability has been constructed for PCC implementors.

The new facilities added to PCC are aimed at improving access to symbolic information, enhancing code generation tables and the

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

code generation algorithm, simplifying targeting for certain classes of architectures, and improving compile-time performance.

3.1 Symbol Table Hooks for Symbolic Debugging

In past releases of PCC, it was difficult to retrieve information from the symbol table at the right time to emit certain information into the assembly file being produced. In partular, symbolic information about structures and function arguments for symbolic debugging tools was not available from dependent routines at the end of a structure declaration and as arguments were being processed.

Macro invocations have been added to pftn.c in dclargs() and dclstruct() so that machine dependent functions can be called to pass adequate symbolic debugging information to an assembler. The macros are FIXARG(p) which is invoked once for each formal argument to a function in the order they are found in the argument list (not specification order) and FIXSTRUCT(szindex,oparam) which is invoked at the end of a structure template declaration. If these hooks are of no interest to a particular compiler implementation, no action on the part of the implementor is required.

3.2 Multi-level Tree/Template Matching

The algorithm for code generation in PCC is based on matching expression trees generated during parsing to code generation templates. When a match of a template is made, the code associated with the template is expanded and emitted based on characteristics of the portion of the expression tree that the template matched. The templates in previous releases of PCC were limited to being simply two-level (one operator, two leaf operands) unless special code was written in the machine dependent parts to extend the scope of an operand beyond that of simple leaf. An optional facility has been added to PCC to describe multi-level tree templates in terms of data tables. These multi-level templates can be used to extend in a simple fashion the code generation templates and to describe tree patterns that can be matched without adding new code and functions to the dependent portion of a compiler.

A function mlmatch() has been added to match.c that is capable of matching trees of arbitrary depth to tree descriptions contained in array mltree[]. The matching can take place completely from the root of the expression tree to all of its leaves or from the root to any depth along its branches according to specifications in the multi-level template. The intent of this facility is not to generate code according to tree matches but to extend the power of shape descriptors in the existing code generation templates. This same facility can be used in dependent file routines such as setbin(), setasg(), setstr() to recognize particular tree forms to be handled in special ways and routines like mycanon() and special() to discover new addressing modes and

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

machine dependent doubly indexed OREG nodes.

The function mlmatch() is called with three parameters:

p — A pointer to a NODE which is the tree to be compared to the multi-level templates in mltree[].

target — An integer code indicating the desired class of multi-level templates to be compared to p.

subtarget — An integer code indicating the desired subclass of target trees to be matched. If subtarget is zero, only the target is used.

Multi-level templates are described in mltree[] by specifying triples of {op, shape, type} descriptors for each node in the tree to be matched. The op field can be either an operator (eg.: UNARY MUL, PLUS) or an operator class (eg.: OPLTYPE, OPCOMM). The operator class OPANY matches any operator but causes the matching routine to treat the matched node as a leaf. The shape field can be any mip defined shape descriptor (eg.: STARREG, SNAME) or an implementor defined special shape. The SANY shape again matches any node. The type field can be any of the mip defined type descriptors (eg.: TUCHAR, TPOINT, TPTRTO|TSTRUCT). TANY matches the type of any node. The combination of multiple mip shape and type descriptors using the bitwise or (eg.:

```
SNAME|SOREG|SCON|SAREG|STAREG
```

or

```
TUSHORT|TSHORT )
```

can be done as usual. (Special shapes, of course, cannot be combined with either other specials or mip defined shapes.) The multi-level template descriptors are interpreted in the same fashion that trees were read from the front end of a PCC/127r0 compiler to the back end. A BITYPE node is anticipated to have two offspring, a UTYPE node to have a single offspring, and an LTYPE node (and OPANY) to have no offspring. (OPLOG is considered to be BITYPE.) A header precedes each template description and identifies the target and subtarget codes. The third field in the header is left as an internal link to be inserted by the routine mlinit(). This initialization routine also checks the integrity of the templates to the degree possible.

Templates are constructed in an array that is initialized in a machine dependent back end file with the following declaration:

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

```
struct ml_node mltree[] = {
        ...
        /* templates */
        ...
        0,0,0
        };
```

Examples of multi-level templates that match assignments of a structure returned from a function to a structure variable follow:

```
STCALLASG,      0,      0,
        STASG,  SANY,   TANY,
                OPANY,  SANY,   TSTRUCT,
                STCALL, SANY,   TANY,
                        OPANY,  SANY,   TANY,
                        OPANY,  SANY,   TANY,

STCALLASG,      0,      0,
        STASG,  SANY,   TANY,
                OPANY,  SANY,   TSTRUCT,
                UNARY STCALL,   SANY,   TANY,
                        OPANY,  SANY,   TANY,
```

Following a template's header triple, tree nodes are listed in preorder. The indentation suggested in the example (the left and right subtrees of a node indented under the node) improves the readability of each template. An example of a tree template that might be useful for three address instructions is:

```
THREEADDR,      ASSIGN, 0,
        ASSIGN, SANY,   TANY,
                AWD,    SANY,   TANY,
                OPSIMP, SANY,   TANY,
                        AWD,    SANY,   TANY,
                        AWD,    SANY,   TANY,
```

The templates can be used in a recursive manner as well. The following pair of templates can be used to match towers of integer additions that lean to the right:

```
ADDTOWER,       0,      0,
        PLUS,   SANY,   TINT,
                OPLTYPE,        SANY,   TINT,
                OPANY, SPECIAL|MULTILEVEL|ADDTOWER, TINT,

ADDTOWER,       0,      0,
        OPLTYPE,        SANY,   TINT,
```

As suggested by the previous example, the tshape() function in match.c has been extended to call mlmatch() directly when implementor defined shapes have the SPECIAL and MULTILEVEL bits set.

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

5-1328 (11-72)

Inclusion of the multilevel template matching facilities in a PCC instance is controlled by the definition of a macro name MULTILEVEL. In order to include multi-level tree matching in a PCC instance and for tshape() to work correctly, the following definition should be placed in mac2defs:

```
# define MULTILEVEL     040000
```

The multi-level match routines (mlmatch(), mlinit()) require approximately 1500 bytes of text space on a PDP/11. The advantages offered by the facility include the replacement of C text by data. The table of templates is simpler to read and modify than C source if statements needed to recognize an equivalent set of trees. Profiling a PCC compiler that used this facility in place of explicit C source showed no significant overhead in execution time to compile identical code. The use of this general purpose mechanism should simplify the generation of new PCC instances.

3.3 Creating OREGs in clocal()

A significant amount of the compiler's time is spent converting automatic and formal parameter NAME nodes into indirect register and offset additions that are later recognized and converted to OREG nodes, often an addressing mode on many machines.. One can now construct a OREG node in clocal() of local.c in the front of the compiler. A local routine (addroreg) must be supplied by a PCC implementor choosing to build an OREG in the front to unwrap the node when the address operator is applied to it. This routine is enabled by # defining the macro ADDROREG in macdefs. The function addroreg() takes one parameter, a NODE pointer pointing to the OREG to be unwrapped, and returns a pointer to the NODE after unwrapping. Compilation speed improvements on the order of 5% have been observed with this improvement.

3.4 Shape Recognition by Order Routine

There were certain cases in the previous release of PCC in which the code generator was not aware that a particular expression subtree was already suitable as an addressing mode and proceeded to generate code to load the subtree into another addressing mode, for instance, a register. Code generation routines have been enhanced to avoid overprocessing subtrees in this fashion.

The purpose of the mip function order() in reader.c is to generate code for its first argument, an expression tree, such that it satisfies one of the cookies specified in its second argument. (A cookie is a code generation goal or subgoal.) A call on tshape() exists in the cleanup portion of the order() function to check that the remaining tree structure after code generation matches at least one of the cookies order() was called with. Tshape() in match.c will verify that a tree matches either a cookie or a shape due to their similarity in bit encodings.

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

However, it is sometimes too late in the cleanup section of order() to check that the input tree has been modified to a reasonable shape. A call on tshape() has been placed in the main loop of order() so that after calls to the setbin(), setasg(), etc. functions in order.c code generation can be stopped as soon as an acceptable shape has been reached. For instance, if

```
order( p->right, INAREG|INTAREG|SOREG|STARREG )
```

was called with p pointing to the following tree:

```
ASSIGN
        REG
        UNARY MUL
                PLUS
                        [arbitrary expression]
                        ICON
```

will now be capable of leaving p as follows:

```
ASSIGN
        REG
        OREG
```

With this capability in this example, a load of the OREG node into a temporary REG node was avoided. This order() modification will simplify the coding of the setbin(), setasg(), etc. functions, improve code generation where calls on order() into registers can be expanded into order() calls that specify many allowable addressing modes, and offer an opportunity to generate code into implementor defined shapes that can be recognized through special() or mlmatch().

3.5 Overriding Result Type Conversions

In the C language, the result of an arithmetic expression is either unsigned, int, or double depending on the types of the operands of the expression. For instance, the sum of two shorts is an int, and the product of a float and an int is a double. On some processors, notably MAC-8, it may be inefficient and sometimes impossible to make such type conversions. A facility has been added to the expand() and reclaim() functions in match.c and allo.c respectively to allow a PCC implementor to override this automatic type conversion. When a register is being rewritten as the result of a matching a tree with a code generation template that has a code expansion character string that contains the letter T, its type will not be converted. Consider the following templates:

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

```
ASG PLUS,       INAREG|INTAREG,
        SAREG,  TCHAR|TSHORT,
        AWD,    TCHAR|TSHORT,
                0,      RLEFT,
                "       add     TAL,AR\n",

UNARY MINUS,    INAREG|INTAREG,
        AWD,    TCHAR|TSHORT,
        SANY,   TANY,
                NAREG,  RESC1,
                "       movneg  TA1,AL\n",
```

When either template is matched, the rewrite of the matched tree is typed according to the type of the operator as constructed in buildtree() of trees.c, not as an integer.

3.6 Incompatible Assignments, Casts, and Returns

A PCC error allowed the assignment of a structure to a pointer. This has been corrected so that such an attempt will be flagged with a user error causing a halt in code generation but a continuation of syntax analysis. The correction was made in the independent file trees.c, function opact().

3.7 Addressable Values as a Left Operand to DOT

The PCC previously permitted some non-addressable primary expressions to precede the structure reference operator (.). Only "lvalues" are permitted before this operator. The following references are illegal in C:

```
struct x {int a,b;} f();

....f().a ... /* call to f is not addressable */
... f.a ...   /* function name is not an lvalue */
```

To correct this error, a check has been inserted into the file cgram.y and a suitable user error message will be issued when such a violation of C occurs in a source file.

3.8 Legitimate Lvalues

Appropriate checks have been added in trees.c to the notlval() function and buildtree() function to disallow structure assignments and function calls that return structures to be addressable. The following cases are now correctly flagged as user errors by PCC:

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

```
struct x {..} f(), xa, xb, *xp;
...
f() = xa; /* cannot assign to a returned structure */
(xa=xb) = f(); /* cannot assign to an assignment */
xp = &(f()); /* cannot take address of returned struct */
```

3.9 Function Arguments

Previously in the PCC, formal parameters that were declared to be functions (an impossibility in C) were silently converted to function pointers. A warning has been added in trees.c to illuminate this misuse of the compiler. The function parameter is still converted to a function pointer. An example of affected C source is:

```
f(x) int x(); {...}

/* which is treated as if it were */

f(x) int (*x)(); {...}
```

3.10 Check for Offset Calculation Overflow

Several host machine independent software checks have been added in the file pftn.c to detect when either a structure or a stack frame has been declared too large for the compiler to maintain the correct bit offset of its members (structures) or its automatic variables (stack frame). For instance, with a target processor that has eight bits to the byte and a host with sixteen bit integers, the largest offset that can be maintained is 2**15-1 (the largest signed integer) bits or 4095 bytes. The number of bits in an integer on the host machine is computed in mainp1() of scan.c. A compiler error is now issued when overflow is detected.

3.11 Assembly Escape Use Detection

In cases where a compiler command must know whether it is safe to use a post compiler optimizer or it is necessary to invoke an assembly language macro processor before sending the compiler-generated assembly code to an assembler, it must be capable of determining whether the assembly language escape (asm("<asm string>")) has been exercised by a C programmer. A global variable `asm_esc' has been added to scan.c which is set to 1 if an assembly language escape has been encountered in the C source.

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

4. PCC MIP SOURCE AVAILABILITY

The machine independent parts of the Portable C Compiler are available from the Murray Hill stockroom on the USG VAX machine (mhtsa).

5. SUMMARY

A first release of the Portable C Compiler machine independent parts is being made by the Languages and Compilers group of Department 3621. This release contains several C language changes (most notably: non-unique structure member names), error corrections, and improved compiler generation facilities. Questions regarding the PCC changes and distribution of the mip source code should be directed to the author.

IH-3621-BRR-mm

B. R. Rowland

Att.
References

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)

REFERENCES

[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.

[Row 79] B. R. Rowland. "Recent Changes to C," Technical Memorandum, TM-77-3621-2, 3621-791002.02MF (October 2, 1979).

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328 (11-72)