Linting c code

You can use the typtic program to make sure that programs C do not contain syntax errors and to verify that the programs do not contain data type errors. This section describes most of the audit operations performed by the hoarding: Program flow control Data type control Variable and function control Migration control Portability control Understanding a tbilit library For a complete list of the Tittic error messages Creation Tiftik options, see <a0><a1></a1></a0>. 6.1 Tbilisi Program Overview The Tbilisi program controls a program more carefully than some C compilers and displays messages that point to potential problems. Some of the messages require a hotfix in the source code; others are informational messages only and do not require corrections. The tymk command has the following syntax: the tym [options] [file [ ] options options for controlling tymtic checking operations. CC drive flags are available as -std, -std0 and -std1 tbilisi options. These flags affect the delocation of the resource and why you choose the titbook to use. Selecting the -std or -std1 flags opens the ANSI de-separation rules in the tbilisi. When you use the -MA windshich flag, -std1 C is used for the preprocessing phase and is defined by using the _ANSI_C_SOURCES -D preprocessor flag. The following table describes the action required for lint for each flag: Lint Preprocessor Lint Lint Library Option Key Parsing -MA -std1 and -D_ANSI_C_SOURCE ANSI llib-lansi.ln -std -std ANSI llib-lcstd.... ln -std1 -std1 ANSI llib-lcstd.ln -std0 -std0 EXTD llib-lc.ln Table Note: EXTD is an Extended C language, also known as K&amp;R C. To check the name of the C language source file for file Lint. The name must have one of the following sufficed: Suffration .c C source file .i File C preprocessor (cpp) .ln note that the hoarding library files produced by the tym library file were previously the result of a call with the -c or -o option. When the .c file is exported as input, they are similar to the .o files generated by the cc command. The ability to specifytific libraries as input to the hoarding program facilitates module interface control in large applications. Adding rules to their Makefiles that specify the creation of tbilisi libraries can make creating such applications more efficient. For a discussion about creating a tbilisi library, see Chapter 6.10. You can also specify a tymtic library entry in one of the system's default library search directories by using the -lx option. The library name must have the following form: llib-llibname.ln By default, the hoarding program adds the extended C (K&amp;R C) tym library (llib-lc.ln) to the list of files specified at the command line. If the -std or -std1 flag is used, you can use the standard C Adds. The following additional libraries are included in the system: Description Crses Controls specify library call syntax -lcrses m Controls math library call syntax -lm port Controls for portability with other systems -p (note -lport) ansi Enforces ANSI C standard rules -MA (not -lansi) If you do not specify flags at the command line, lint program checks the specified C source files and writes messages about the following encoding problems that it finds : Data types that are not normally entered and do not appear Variables that are not used correctly Variables Standard coding techniques that can cause problems if a program is moved to another system non-encoding applications and style differences that can cause problems Also check for syntax errors in statements in source programs. Syntax checking is always performed and is not affected by option flags. If it does not report any errors, the program has the correct syntax and will be compiled without errors. However, passing this test does not mean that the program will work correctly or that the logical design of the program is correct. For information about how to create your own tbilisi library, see Chapter 6.10. 6.2 Program Flow Control Controls the Dead code, that is, parts of a program that can never be executed because they cannot be accessed. Writes messages about expressions that do not have labels but immediately follow expressions such as goto, break, continue, and return that change the program flow. The tym program also detects and writes messages for the following conditions: Writes messages for the following conditions: A loop that cannot be exited at the bottom cannot be entered at the top like infinite loops: while(1) for (;;) Some programs that contain these cycle types may produce accurate results. However, such loops can cause problems. The tym program does not recognize the functions called, but can never return to the search program. For example, an output call can result in inaccessible code, but the hoard does not detect it. Programs created by Yacc and Lex can have hundreds of intermediate statements that cannot be accessed. The tym program normally writes an error message for each of these interrupt statements. Use the -O flag on the cc command when ingingering the program to eliminate the resulting object code inefficient, so that these additional expressions are not important. Use the -b flag with the tym program to prevent yacc and lex from writing these messages when checking the exit code. (For information about Yacc and lex, see programming support tools.) 6.3 The Data Type Control Tbilisi program enforces type control rules of language C more strictly than the compiler. In addition to the controls the compiler makes, it checks for possible data type errors in the following fields: Binary operators and axle assignment structures, and The function definition uses enumerators type control type dumps and details of each of these potentially problematic areas are provided in the following sections. 6.3.1 Binary Operators and Impered Assignments C language allows the following data types to be mixed in statements and does not show an error when the compiler is mixed: char short int long unsigned float automatically converts data types within this group to give the dual C language programmer more flexibility in programming. However, this flexibility means that data type mixing must be sure that it produces the desired result. You can mix these data types when using them in the following ways (in examples, alpha type char and num type int: Operands on both sides of an assignment operator, for example: alpha = num; /* alpha int is converted */ A conditional expression is also operands, for example: value=(alpha &lt; num) ? alpha : num; /* alpha converts operands on both sides of the relational operator to int */ Operands on both sides of a relational operator: if (alpha != num) /* alpha int */ The type of argument in the return statement is converted to the value that the function returns, for example: funct(x) /* a tams returns the month */ { return(alpha(} The data types of pointers must be fully accepted as long as you can mix the arrays of x's with the pointers of x's. 6.3.2 The Structures and Unions Tymtic program controls construction operations according to the following requirements: The left operand of the build pointer operator (-&gt;) must point to a structure. The build member operator (\.) must be a left operand structure. These operators must be a member of the same structure as the right operand. The tbilisi program makes similar audits for applications to trade unions. 6.3.3 Function Definition and Uses The Tymtic program enforces strict rules for function argument and return value matching. Arguments and check-in values can match type float arguments to double type arguments, except in the following exceptions. You can match arguments in the following types: char short int can match pointers with unconsens signatureed related arrays. 6.3.4 Enumerators The Tymtic program checks numbered data type variables to make sure they meet the following requirements: Enumerator variables or members of a numbered type are not confused with other types or other enumerator variables. Numbered data type variables are used only in the following fields: Assignment (=) Initialization Equivalence (==) Not equivalence (!=) Function arguments Return values 6.3.5 Type Dumps in C allow the program to process data of one type as if it were data of another type. The tbilisi program can check the writing transcripts and write a message if it finds one. For -wp and -h options the typtic command line checks the spelling of warning messages about listings. If none of these flags are used, the script generates warning messages about dumps that can cause portability issues. In migration control mode, -Qc suppresses dump warning messages (see <a0><a1></a1></a0>). Chapter 6.6). 6.4 The Variable and Function ControlTific program controls variables and functions that are declared but not used in a program. The typtic program checks for the following errors in the use of variables and functions: Functions that return functions that are defined as inconsistent but not used functions are used for functions that can return values that return programs that use the value of a function when the function does not return a value in each of these potential problem areas. 6.4.1 Inconsistent Function Check-in If a function returns a value under one set of conditions, but is not under another condition, you cannot predict the results of the program. The tymtic program controls functions for this type of behavior. For example, if both of the following statements are in the function definition, a program that calls the function may or may not receive a return value: return(expr); ... go back; These expressions cause the typtic program to type the following message to point to the potential problem: the function name is return(e); and the rotation Tymtik program also checks functions for check-ins caused by reaching the end of the function code (the im/ly check-in). For example, in the following part of a function, if a test is counted incorrectly, it calls payment correction\(ulit) and returns without a defined return value: payment (a) { if (a) return (3); fix\(ulit (); } } These expressions cause the typtic program to write the following message: the function is returned to payment(e); and return If fix\(ulit, exit, never returns, writes the message even though there are no problems. 6.4.2 Unused Function Values Controls situations where a function returns a value and the search program cannot use the value. If the value has never been used, the function definition may be inefficient and should be examined to determine whether it should be modified or eliminated. If the value is sometimes used, the function returns an error code that the search program does not check. 6.4.3 Disable Function Control To prevent The Tbilisi from checking for problems with functions, specify one or more of the following flags in the tiftik command: -x Do not check variables declared in an extern statement but never used. -v Do not check arguments for functions that are not used except those declared as record arguments. -u Do not check used and unused functions and external variables or defined and unused. Use this flag to eliminate useless messages when running a tbilisi on a subset of a larger program's files. (When using files that work with some, but not all, of the scripts, many of the functions and variables defined in these files may not be used. In addition, many functions and variables defined elsewhere can be used.) You can also place instructions in the program to control auditing: To avoid warning about unused function arguments, add the following directive to the program before the function definition: /*ARGSUSED*/ To prevent calls to a function from writing messages about variable variable numbers, Before the function definition, add the following directive: /*VARARGSn*/ To check the first few arguments and leave the next arguments unaltened, add one digit (n) to the end of the VARARGS directive to give the number of arguments that need to be checked: /*VARARGS2*/ When you read this directive, it checks only the first two arguments. To suppress complaints about unused functions and function arguments in the entire file, place the following directive at the beginning of the file: /*LINTLIBRARY*/ This is equivalent to using the -v and -x flags. Use the following guidelines to allow a standard prototype control library to be created from header files by making function prototype declarations appear as function definitions: /*LINTSTDLIB [_filename] */ /*LINTSTDLIB*/ directive indirectly enables the functions of directives /*NOTUSED*/ and /*LINTLIBRARY*/ to reduce warning noise levels. When a file is referenced (file name), only prototypes in that file are expanded. Multiple /*LINTSTDLIB_filename*/ statements are allowed. (See Section 6.10.1 /*LINTSTDLIB*/ for more information on the use of directives.) Then use the following directive to suppress warnings about all used but undefined external symbols and functions encountered in the file: /*NOTDEFINED*/ To suppress comments about inaccessible code, use the following instructions: /*NOTREACHED*/ Stops comments about unreachable code when placed at appropriate points in a program (usually following a check-in, interrupt, or continue statement), /*NOTREACH*/. Note that it doesn't recognize the tymtic output function and other functions that don't return. Then use the following instructions to suppress warnings about all unused external symbols, functions, and function parameters encountered in the file: /*NOTUSED*/ /*NOTUSED*/ directive /*LINTLIBRARY*/ is similar, but also applies to /*NOTUSED*/ external symbols. 6.5 Using Variables Before The Initials The Tym program provides a local variable (automatic and record storage classes) before a value is assigned Checks. Use a variable with automatic (automatic) or registration the class also includes taking the address of the variable. This is required because the program can always use the variable (through its address) after receiving the address of the variable. Therefore, if the program cannot assign a value to the variable before finding the address of the variable, it reports an error. Because the hoard only overstitude overses the physical order of variables and their use in the file, it can write messages about variables that are properly initial (in execution order). Recognizes and writes messages about the tym program: The initial automatic variables used in the expression that are first set and set variables that are never used note The operating system lowers static and extern variables to zero. Therefore, the tym rules that these variables are set to zero at the beginning of the program and does not check whether a value is assigned when they are used. When developing a program for a system that does not start this, make sure that the program adjusts the static and extern variables to the initial value. 6.6 Migration Control Use tbilisi to control all common programming techniques that can cause problems when switching programs from 32-bit operating systems to a 64-bit Digital UNIX operating system. The -Q option provides support for checking programs written for ULTRIX and DEC OSF/1 Version 1.0, where you switch to 64-bit systems. Because the -Q option disables checking for other programming problems, you should use this option only for migration control. Sub-options can be used to suppress specific control categories. For example, by entering -Qa, the pointer suppresses control of alignment issues. You can enter multiple sub-options with the -Q option to control pointer alignment issues, problem type dumps, and function prototype controls, respectively. For more information about migration control, see tym(1). 6.7 Increase Table Size The Tbilisi command provides the -N option and related submena options, which allow you to increase the size of various internal tables at run time if the default values are not sufficient for your program. These tables are: Symbol table Dimension table Local writing table Parse tree These tables are dynamically separated by the tym program. The -N option can be used in large source files to improve performance. 6.8 Portability Check Use lint to make sure you can compile and run C programs using different C language compilers and other systems. The following sections show the fields that were checked before compiling the program on another system. However, checking only these fields does not guarantee that the program will run on any system. Note that the llib-port.ln library is brought not by using the -lport flag, but by using the -p flag. 6.8.1 Character Uses Some Define characters in the C language program as signed quantities ranging from -128 to 127; other systems define characters as positive values. The tymtic program checks for non-portable character comparisons or assignments to other systems. For example, the following part can run on a single system, but may fail on systems where characters always receive positive values: char c; ... if(c = getchar()) &lt;0)\.\.\. This expression causes the typtic program to type the following message: non-portable character comparison To ensure that the program works on systems that use positive values for characters, getchar returns integer values because report c as an integer. 6.8.2 Bit Space Bit fields can also be problematic when a program is transferred to another system. Bit fields can be quantities signed in the new system. Therefore, when constant values are assigned to a bit field, the field value may be too small to hold. To have this assignment run on all systems, report the bit field unsigned before assigning a value. 6.8.3 External Name Size When switching from one system type to another, note the differences in information about external names during the installation process: The number of characters allowed for external names may vary. Some of the programs that the compiler calls and some of the functions that your programs are looking for may further limit the number of significant characters in identifiers. (In addition, the compiler adds a leading sub-score to all names and keeps uppercase and lowercase characters separate.) On some systems, uppercase or lowercase letters may not be important or may not be allowed. When transferring from one system to another, you should always take the following steps to avoid problems with installing the program: Review the requirements for each system. Run the tbilisi with the -p flag. The -p flag tells the feather to humiliate all external icons and limit them to six characters when checking input files. Generated messages show terms that need to be changed. 6.8.4 Multiple Uses and Side Effects Be careful when using complex expressions because of the following: The order in which complex expressions are evaluated differs in many C compilers. Function calls, which are arguments for other functions, may not be handled in the same way as ordinary arguments. Operators such as assignments, increments, and decrements can cause problems when used on different systems. The following situations show the types of problems that can be caused by these differences: If any variable is replaced by a side effect of one operator and used elsewhere in the same expression, the result remains undefined. The evaluation of variable years in the following printf statement is confusing because on some machines, increases after calling and calling functions on other machines: printf(%d%d, ++years, amort (interest, year); The tym program controls simple scaler variables that may be affected by evaluation sequence problems(for example, as in the following statement: a[i]=b[i++]; This expression causes the typtic program to type the following message: warning: i evaluation sequence undefined 6.9 Encoding Errors and Encoding Style Differences Use typtic to detect possible encoding errors and to detect differences from the encoding style that the typtic expects. Although the coding style is mainly a matter of individual taste, examine each difference to make sure the difference is both necessary and accurate. The following sections show the types of encoding and style issues that you can find in the tbilisi. 6.9.1 Assignments of Long Variables to Integer Variables If you assign type variables that are long to the variables of the Type type, the program may not function correctly. The long variable is truncated to fit in the integer field, and data can be lost. This type of error occurs frequently when a program that uses more than one typedef is converted to run on a different system. Use the -a flag to prevent long variables from writing int messages when they detect assignments to int variables. 6.9.2 Operator Priority The Typtic program detects possible or potential errors in operator priority. Without the same time to show the order in complex arrays, these errors can be difficult to find. For example, the following expressions are not clear: if(x&amp;077==0). . /* : if(x &amp; (077 ==0)) */ /* must be: if((x &amp; 077) == 0) */ x &lt;2+40 evaluated= as:= x=&gt;&lt;/2+40&gt; &lt;(2+40) */= should= be:=&gt;&lt;/(2+40)&gt; &lt;2) += 40= */= shift= x= left= 42= positions= */= use= parentheses= to= make= the= operation= more= clearly= understood.= if= you= do= not,= lint= writes= a= message.= 6.9.3 conflicting= declarations= the= lint= program= writes= messages= about= variants= that= are= declared= in= inner= blocks= in= ways= that= conflict= with= their= use= in= outer= blocks.= this= practice= is= allowed,= but= may= cause= problems= in= the= program.= use= the= -h= flag= with= the= lint= program= to= prevent= lint= from= checking= for= conflicting= declarations.= 6.10 creating= a= lint= library= for= programming= projects= that= define= additional= library= routines ,= you= can= create= an= additional= lint= library= to= check= the= the= syntax= of= programs.= using= this= library,= the= lint= program= can= check= the= new= functions= in= addition= to= the= standard= c= language= functions.= perform= the= following steps= to= create= a= = lint= library:= create= an= input= file= that= defines= the= new= functions.= process= the= input= file= to= create= the= lint= library= file.= run= lint= using= the= new= library.= the= following= sections= these describe= steps.= 6.10.1 creating= input= file= the= following= example= shows= an= input= file= that= defines= three= additional= functions= for= lint= to= check.= lintlibrary*/= = #include=&gt;&lt;/2)&gt; &lt;dms.h&gt; int dmsadd(&lt;/dms.h&gt; dmsadd(&lt;/dms.h&gt; recbuf, reclen) int rmsdes; char *recbuf; unconsens signature reclen; { return 0; } int dmsclos (rmsdes) int rmsdes; { return 0; } int dmscrea(path, mode, recfm, reclen) char *path; int mode; int recfm; unconsens signature reclen; { return 0; } An input file is a text file that you create with an editor. This occurs: A directive to tell the CPP program that the following information will be made into a serrated library of lint definitions to be made into a library: /*LINTLIBRARY*/ A set of function definitions that define: the type of function (int in the example) The name of the function returns the function alternatively to the types of parameters that the parameters expect in the types of function, you can create a lint file library from function prototypes. For example, suppose the dms.h file contains the following prototypes: int dmsadd(int, char*, un signed); int dmsclose(int); int dmscrea(char*, int, int, un signed); In this case, the input file contains the following: /*LINTSTDLIB*/ #include &lt;dms.h&gt; In cases where a header file may contain other tops, the LINTSTDLIB command may be limited to specific files: /*LINTSTDLIB_dms.h*/ In this case, only prototypes declared in dms.h will be expanded. More than one LINTSTDLIB command can be added. In any case, the name of the input file must be a preeki: llib-l. For example, the sample input file created in this section might be named llib-ldms. When selecting the name of the file, make sure that the existing files in the /usr/ccs/lib directory are not the same as any of them. 6.10.2 Creating the Lint Library File The following command creates a lint library file from the input file described in the previous section: % lint [options] -c llib_ldms.c This command tells you to create a lint library file using the llib-ldms.ln file. To use llib-ldms.ln as a system tbilisi library (that is, a library specified in the -lx option of the tym command), move it to /usr/ccs/lib. Use the -std or -std1 flag to use ANSI preprocessing rules to create the library. 6.10.3 Checking a Program with a New Library To control a program by using a new Library, use the lint command as: lint -lpgm filename.c represents the identifier of the variable pgm library, and variable file name.c represents the name of the file that contains the C language source code to be checked. If no other flags are specified, the lint program checks the C language source code against the standard lint library and compares it to the specified custom lint library. 6.11 Understanding Error Messages Although most error messages generated by the hoard are self-explanatory, some messages can be misleading without an additional explanation. Typically, once you understand what a message means, it is easy to correct the error. Below is a more ambiguous list of hymtics &lt;/dms.h&gt;NOT operatörü (!) ile bir sabit kullanılır. Bu yaygın bir kodlama pratice ve ileti genellikle bir sorun göstermez. Aşağıdaki program bu iletiyi oluşturabilecek kod türünü göstermektedir: % cat x.c #include &lt;stdio.h&gt;#define SUCCESS 0 main() { int value = ! BAŞARı; printf(değer = %d, değer); dönüş 0; } % tiftik -u x.c x.c, satır 7: uyarı: NOT % ./x değeri = 1 % Program beklendiği gibi çalışır, tiftik şikayet rağmen. Önerilen Eylem: -wC seçeneğini kullanarak bu tiftik uyarı iletilerini bastırın. koşullu bağlamda sabit Koşullu bir beklenen yerde sabit kullanılır. Bu sorun, makroların kodlanmış olması nedeniyle genellikle kaynak kodunda oluşur. Örneğin: typedef struct _dummy_q { int lock; struct _dummy_q *head, *tail; } DUMMY_Q; #define QWAIT 1 #define QNOWAIT 0 #define DEQUEUE(q, elt, bekleyin) [1] \ için (;;) { \ simple_lock(&amp;(q)-&gt;lock); \ if (queue_empty(&amp;amp;q)-&gt;head)) \ if (bekleyin) { [1] \ assert(q); \ simple_unlock(&amp;s)-&gt;lock); \ devam et; \ } else \ *(elt) = 0; \ else \ dequeue_head(&amp;&amp;gt;q)-&gt;head); \ simple_unlock(&amp;amp(&amp;amp)-&gt;lock); \ break; \ } int doit(DUMMY_Q *q, int *elt) { DEQUEUE(q, elt, QNOWAIT); } QWAIT veya QNOWAIT bayrağı üçüncü bağımsız değişken (bekleme) olarak geçirilir ve daha sonra if deyiminde kullanılır. Kod doğrudur, ancak bu şekilde kullanılan sabitler normalde gereksiz olduğundan ve genellikle savurgan veya gereksiz talimatlar oluşturduğundan uyarıyı yayınlar. [Örneğe dön] Önerilen Eylem: -wC seçeneğini kullanarak bu tiftik uyarı iletilerini bastırın. uzun dan dönüştürme doğruluk kaybedebilir A imzalı uzun daha küçük bir varlığa kopyalanır (örneğin, bir int). Bu ileti mutlaka yanıltıcı değildir, ancak sık sık oluşursa, aşağıdaki örnekte gösterildiği gibi bir kodlama sorunu gösterebilir veya göstermeyebilir. uzun BuffLim = 512; [1] void foo (tampon, boyut) char *tampon; int boyutu; { kayıt int sayısı; kayıt &lt; (int)BufLimit ? size : (int)BufLim; [1] The lint program reports the conversion error, even though the appropriate (int) cast exists. [Return to example] Recommended Action: Review code sections for which lint reports this message, or suppress the message by using the -wl option. declaration is missing declarator A line in the declaration section of the program contains just a semicolon (;). Although you would not deliberately write code like this, it is easy to inadvertantly generate such code by using a macro, followed by a semicolon. If, due to conditionalization, the macro is defined as empty, this message can result. Recommended Action: Remove the trailing semicolon. degenerate unsigned comparison An unsigned comparison is being performed against a signed value when the result is expected to be less than zero. The following program illustrates this situation: % (int)buflimit= size= := (int)buflim;= [1]= the= lint= program= reports= the= conversion= error,= even= the= appropriate= (int)= cast= exists.= [return= to= example]= recommended= action:= review= code= sections= for= which= lint= reports= this= message,= or= suppress= the= message= by= using= the= -wl= option.= declaration= is= missing= declarator= a= line= in= the= declaration= section= of= the= program= contains= just= a= semicolon= (;).= although= you= would= not= deliberately= write= code= like= this,= it= is= easy= to= inadvertantly= generate= such= code= by= using= a= macro,= followed= by= a= semicolon.= if,= due= to= conditionalization,= the= macro= is= defined= as= empty,= this= message= can= result.= recommended= action:= remove= the= trailing= semicolon.= degenerate= unsigned= comparison= an= unsigned= comparison= is= being= performed= against= a= signed= value= when= the= result= is= expected= to= be= less= than= zero.= the= following= program= illustrates= this= situation:= % =&gt;&lt;/ (int)BufLimit ? size : (int)BufLim; [1] The lint program reports the conversion error, even though the appropriate (int) cast exists. [Return to example] Recommended Action: Review code sections for which lint reports this message, or suppress the message by using the -wl option. declaration is missing declarator A line in the declaration section of the program contains just a semicolon (;). Although you would not deliberately write code like this, it is easy to inadvertantly generate such code by using a macro, followed by a semicolon. If, due to conditionalization, the macro is defined as empty, this message can result. Recommended Action: Remove the trailing semicolon. degenerate unsigned comparison An unsigned comparison is being performed against a signed value when the result is expected to be less than zero. The following program illustrates this situation: % &gt; int limit = boyutu&lt;/stdio.h&gt; boyutu&lt;/stdio.h&gt; #include &lt;stdio.h&gt; imzasız uzun ofset = -1; main() { if (ofset &lt; 0)= {= [1]= puts= (code= is= ok...); = return= 0;= }= else= {= puts= (unsigned= comparison= failed...); = return= 1;= }= }= % = cc -g -o x x.c= = % = lint x.c= = x.c line 7: warning: degenerate unsigned comparison= = % = ./x= = unsigned comparison failed...= = % = unsigned= comparisons= such= as= this= will= fail= if= the= unsigned= variable= contains= a= negative= value.= the= resulting= code= may= be= correct,= depending= upon= whether= the= programmer= intended= a= signed= comparison.= [return= to= example]= recommended= action:= you= can= fix= the= previous= example= in= two= ways:= add= a= (long)= cast= before= offset= in= the= if= comparison.= change= the= declaration= of= offset= from= unsigned= long= to= long.= in= certain= cases,= it= might= be= necessary= to= cast= the= signed= value= to= unsigned.= function= prototype= not= in= scope= this= error= is= not= strictly= related= to= function= prototypes,= as= the= message= implies.= actually,= this= error= occurs= from= invoking= any= function= that= has= not= been= previously= declared= or= defined.= recommended= action := add= the= function= prototype= declaration.= null= effect= the= lint= program= detected= a= cast= or= statement= that= does= nothing.= the= following= code= segments= illustrate= various= coding= practices= that= cause= lint= to= generate= this= message:= scsi_slot=device-&gt; ctlr_hd-&gt;slot,unit_str; [1] #define MCLUNREF(p) \ (MCLMAPPED(p) &amp;amp; --mclrefcnt[mtocl(p)] == 0) (geçersiz) MCLUNREF(m); [2] Önerilen Eylem: Gereksiz dökümleri veya ifadeleri kaldırın veya makroları güncelleştirin. olası işaretçi hizalama sorunu Bir işaretçi bir hizalama sorununa neden olabilecek bir şekilde kullanılır. Aşağıdaki kod kesimi bu iletiyi oluşturan kod türünü gösterir: read(p, args, retval) struct proc *p; void *args; uzun *retval; { kayıt struct args { uzun fdes; char *cbuf; imzasız uzun sayı; } *uap = (struct args *) args; [1] struct uio auio; struct iovec aiov; Satır *uap = (struct args *) args hatanın bildirilmesine neden olur. Bu yapı geçerli olduğundan ve çekirdek kaynağı boyunca oluştuğundan, bu ileti filtrelenir. [Örneğe dön] alan atamasında kaybolan kesinlik Alan değeri tutamayacak kadar küçük olduğunda bir bit alanına sabit bir değer atama girişiminde bulunuldu. Aşağıdaki kod segmenti bu sorunu göstermektedir: % cat x.c struct bitfield { imzasız int block_len : 4; } bt; void test() { bt.block_len = 0xff; } % tiftik -u x.c x.c, satır 8: uyarı: alan atamasında kaybedilen hassasiyet % cc -c -o x x.c % Gördüğünüz gibi, bu kod hatasız derler. Ancak, bit alanı sabiti tutamayacak kadar küçük olabileceğinden, sonuçlar programcının amaçladığı gibi olmayabilir ve çalışma zamanı hatası oluşabilir. Önerilen Eylem: Bit alanı boyutunu değiştirin veya farklı bir sabit değer atayın. 0 ile imzasız karşılaştırma unconfronted comparison against zero is performed when &lt;/stdio.h&gt; &lt;/stdio.h&gt; equal to or greater than zero. The following program shows this: % cat z.c #include &lt;stdio.h&gt; unconsens signature offset = -1; main() { if (offset &gt; 0) { [1] puts(0 Unsigned comparison with Failed); return 1; } else { puts(unsigned comparison 0 arrow); return 0; } } } } } } % cc -o z z.c % tiftik z.c z.c, line 7: warning: unconstructed comparison with 0? %./z fails if the unsigned variable contains a negative value, such as unsigned comparisons with 0 Failed % Unsigned comparisons. Depending on whether the programmer is making a signed comparison, the resulting code may not be correct. [Return to example] Suggested Action: You can correct the previous example in two ways: Add an (int) dump before offset in the If comparison. Int offset notification from unconsens signature. 6.12 Change the Notification to Use Alert Class Options to Suppress Lint Messages Several lint alert classes have been added to the lint program to allow messages associated with constants used in conditional, portability, and prototype controls to be suppressed. You can suppress messages in any of the alert classes by using the alert class option in the lint command. The alert class option has the following format: -wclass [class...] All alert classes are enabled by default, but can be disabled separately by adding the appropriate option as part of the class argument. Table 6-1 lists individual options. Note Several hoarding messages depend on more than one alert class. Therefore, you may need to specify several warning classes to suppress the message. Notes in Table 6-1 specify which messages can only be suppressed by specifying multiple warning classes. For example, you may decide to use the -wC option to suppress because the hoarding messages about constants in conditional statements do not necessarily indicate an encoding problem (as described in Section 6.11). The -wC option suppresses the following messages: because most of the messages associated with continuous argument portability controls that are not fixed in the conditional context are related to non-ANSI compilers, and you can use the -wp option to suppress constraints that are not in the C compiler for Digital UNIX. The -wp option suppresses the following messages: an ambiguous assignment for non-ansi compilers is a recommended encoding application (as described in Section 6.12.1), the use of function prototypes is a recommended encoding application, even though the field sensitivity lost in the field assignment (mark extended?) is lost in the field assignment, even if the illegal dump is long or key statement non-ansi compilers may have truncated the potential pointer alignment problem of non-portable character comparison. You can use the -wP option to suppress prototype controls. This The option suppresses the following messages: function prototype scope incompatible type function argument mix old and new style function declaration old style function declaration prototype table 6-1 in the presence of old style function definition usage: the old style warning class warning class description class non-ANSI properties. Prints: · Partially elided startup [Table Note 1] · Static function %s is not defined or used [Table Note 1] c Comparisons with unsigned values. Prints: · Comparison of unsigned with negative constant · Degenerated unconsens signature comparison · Un signed comparison with 0? d Notification consistency. Prints: · External symbol type conflict for %s · Illegal member usage: maybe %%s [Table Note 2] · Missing post for %s already completed · %s's re-declaration · Struct/union%s never defined [Table Note 2] · Rede defining %s hides a previous [Table Note 1] [Table Note 2] h Intuitive complaints. Prints: · Continuous argument for NOTE [Table Note 4] · Fixed in conditional context [Table Note 4] · Numbering type conflict, op %s · Illegal member usage: maybe %%s [Table Note 3] · Null effect [Table Note 6] · Possible pointer alignment problem, op %s [Table Note 5] · Priority confusion possible: the paratheses! [Table Note 7] · Struct/union%s never defined [Table Note 3] · Redefining %s hides an expected previous [Table Note 3] k K&amp;R type code. Prints: · Argument not used in function %s %s [Table Note 8] · Function prototype not covered [Table Note 8] · Partially elided startup [Table Note 8] · Static function %s is not defined or used [Table Note 8] · %s available before set [Table Note 2] [Table Note 3] · %s rede defining a previous hide [Table Note 2] [Table Note 3] · Set %s but function %s [Table Note 8] l Is not used to assign long values to non-long variables. Prints: · Long conversion may lose accuracy · A long conversion can sign and extend the wrong n Null-effect code. Prints: · Null effect [Table Note 2] o Unknown evaluation order. Prints: · Priority confusion possible: the paratheses! [Table Note 2] · %s evaluation order undefined p Miscellaneous portability concerns. Prints: · Ambiguous assignment for non-ansi compilers · Continuous expression illegal casting · Long case or key statement non-ansi compilers can be cut · Non-portable character comparison · Possible pointer alignment problem, op %s [Table Note 2] · (Signage expanded?) lost sensitivity in assigning to the field · Sensitivity lost in the field · Too many characters in the consistency of the character constant r Return statement. Prints: · Function %s return(e); and return; · Function %s must return a value · Main() randomly values the invocation environment S Storage capacity controls. Prints: · Array termination is not large enough to store null · Constant value (%0x x) (0x%x) u The correct usage of variables and functions is over. Prints: · Argument %s function %s [Table Note 1] is not used · Static function %s is not defined or used [Table Note 1] · Set %s but function %s [Table Note 1] is not used · Function %s %s [Table Note 8] A Enable all alerts. The default option in the script. Specifying another Class A allows all classes to pass the setting. C Conditionally occurring constants. Prints: · Continuous argument for NOTE [Table Note 2] · Fixed [Table Note 2] D External declarations are never used in conditional context. Prints: · Static %s %s are unused O Out of Old properties. Prints: · The storage class checks not the first type specifier P prototype. Prints: · Function prototype not covered [Table Note 1] · Unmatched type in function argument · Mix of old and new style function declaration · Old style argument declaration [Table Note 1] · Using the old-style function definition in the presence of prototype R Detection of unreachable code. Prints: · Statement that cannot be accessed table notes: You can also suppress this message by disabling the K alert class. To suppress this message, you must also disable the h alert class. To suppress this message, you must also disable the d warning class. You must also disable the C alert class to suppress this message. To suppress this message, you must also disable the p alert class. You must also disable the n warning class to suppress this message. To suppress this message, you must also disable that alert class. Other flags can also suppress these messages. 6.12.1 Creating Function Prototypes for Compile-Time Detection of Syntax Errors Digital recommends adding function prototypes to your program for both external and static functions. These declarations provide the compiler with the information it needs to control arguments and rotating values. The CC compiler provides an option that automatically generates prototype notifications. By selecting -proto[is] for the assembly, you can specify an output file (with the same name as the input file, but . H extension) function prototypes. Option I includes identifiers in prototype and s option prototypes for static functions. You can copy function prototypes from a \ . H file and source place in the appropriate locations and include the files. Files.