NAME

gmqcc — A Quake C compiler built from the NIH realm of sarcastic wit

SYNOPSIS

gmqcc [**options**] [*files...*]

DESCRIPTION

Traditionally, a QC compiler reads the file `progs.src` which in its first line contains the output filename, and the rest is a list of QC source files that are to be compiled in order. **gmqcc** optionally takes options to specify the output and input files on the commandline, and also accepts assembly files.

OPTIONS

gmqcc mostly tries to mimic gcc's commandline handling, though there are also traditional long-options available.

-h, **--help**
Show a usage message and exit.

-o, **--output=***filename*
Specify the output filename. Defaults to progs.dat. This will overwrite the output file listed in a `progs.src` file in case such a file is used.

-O*number*
Specify the optimization level

3	Highest optimization level
2	Default optimization level
1	Minimal optimization level
0	Disable optimization entirely

-O*name*, **-Ono-***name*
Enable or disable a specific optimization. Note that these options must be used after setting the optimization level, otherwise they'll be overwritten.

-Ohelp
List all possible optimizations and the optimization level they're activated at.

-q, **--quiet**
Be less verbose. In particular removes the messages about which files are being processed, and which compilation mode is being used, and some others. Warnings and errors will of course still be displayed.

-D*macroname*, **-D***macroname*=*value*
Predefine a macro, optionally with a optional value.

-E Run only the preprocessor as if **-fftepp** was used and print the preprocessed code to stdout.

-W*warning*, **-Wno-***warning*
Enable or disable a warning.

-Wall
Enable almost all warnings. Overrides preceding **-W** parameters.

The following warnings will *not* be enabled:

-Wuninitialized-global

-Werror, **-Wno-error**
Controls whether or not all warnings should be treated as errors.

-Werror-*warning*, **-Wno-error-***warning*
Controls whether a specific warning should be an error.

-Whelp
List all possible warn flags.

-f*flag*, **-fno-***flag*
Enable or disable a specific compile flag. See the list of flags below.

-fhelp
List all possible compile flags.

-nocolor
Disables colored output

-config=*file*
Use an ini file to read all the **-O**, **-W** and **-f** flag from. See the

-debug
Turn on some compiler debugging mechanisms.

-memchk
Turn on compiler mem-check. (Shows allocations and checks for leaks.)

--memdumpcols*columns*
Changes the number of columns to use for the debug memory dump, defaults to 16. **CONFIG** section about the file format.

-redirout=*file*
Redirects standard output to a `file`

-redirerr=*file*
Redirects standard error to a `file`

-std=*standard*
Use the specified standard for parsing QC code. The following standards are available: *gmqcc*, *qcc*, *fteqcc* Selecting a standard also implies some **-f** options and behaves as if those options have been written right after the **-std** option, meaning if you changed them before the **--std** option, you're now overwriting them.

-std=gmqcc includes:

```
-fadjust-vector-fields
-fcorrect-logic
-ftrue-empty-strings
-floop-labels
-finitialized-nonconstants
-ftranslatable-strings
-fno-false-empty-strings
-Winvalid-parameter-count
-Wmissing-returnvalues
-fcorrect-ternary (cannot be turned off)
```

-std=qcc includes:

-fassign-function-types
-fIno-adjust-vector-fields

-std=fteqcc includes:
-fftepp
-ftranslatable-strings
-fassign-function-types
-Wternary-precedence
-fno-adjust-vector-fields
-fno-correct-ternary

--add-info
Adds compiler information to the generated binary file. Currently this includes the following globals:
`reserved:version`
String containing the compiler version as printed by the --version parameter.

--correct, **--no-correct**
When enabled, errors about undefined values try to suggest an existing value via spell checking.

-dump
DEBUG OPTION. Print the code's intermediate representation before the optimization and finalization passes to stdout before generating the binary.

-dumpfin
DEBUG OPTION. Print the code's intermediate representation after the optimization and finalization passes to stdout before generating the binary. The instructions will be enumerated, and values will contain a list of liferanges.

-force-crc=*CRC*
Force the produced progs file to use the specified CRC.

-state-fps=*NUM*
Activate –femulate-state and set the emulated FPS to *NUM*.

COMPILE WARNINGS

-Wunused-variable
Generate a warning about variables which are declared but never used. This can be avoided by adding the `noref` keyword in front of the variable declaration. Additionally a complete section of unreferenced variables can be opened using `#pragma noref 1` and closed via `#pragma noref 0`.

-Wunused-component
Generate a warning about vector variables which are declared but not all their components are used.

-Wused-uninitialized
Generate a warning if it is possible that a variable can be used without prior initialization. Note that this warning is not necessarily reliable if the initialization happens only under certain conditions. The other way is *not* possible: that the warning is *not* generated when uninitialized use *is* possible.

-Wunknown-control-sequence
Generate an error when an unrecognized control sequence in a string is used. Meaning: when there's a character after a backslash in a string which has no known meaning.

-Wextensions
Warn when using special extensions which are not part of the selected standard.

-Wfield-redeclared
Generally QC compilers ignore redeclaration of fields. Here you can optionally enable a warning.

-Wmissing-return-values
Functions which aren't of type *void* will warn if it possible to reach the end without returning an actual value.

-Winvalid-parameter-count
Warn about a function call with an invalid number of parameters.

-Wlocal-shadows
Warn when a locally declared variable shadows variable.

-Wlocal-constants
Warn when the initialization of a local variable turns the variable into a constant. This is default behaviour unless **-finitialized-nonconstants** is used.

-Wvoid-variables
There are only 2 known global variables of type void: `end_sys_globals` and `end_sys_fields`. Any other void-variable will warn.

-Wimplicit-function-pointer
A global function which is not declared with the `var` keyword is expected to have an implementing body, or be a builtin. If neither is the case, it implicitly becomes a function pointer, and a warning is generated.

-Wvariadic-function
Currently there's no way for an in QC implemented function to access variadic parameters. If a function with variadic parameters has an implementing body, a warning will be generated.

-Wframe-macros
Generate warnings about `$frame` commands, for instance about duplicate frame definitions.

-Weffectless-statement
Warn about statements which have no effect. Any expression which does not call a function or assigns a variable.

-Wend-sys-fields
The `end_sys_fields` variable is supposed to be a global variable of type *void*. It is also recognized as a *field* but this will generate a warning.

-Wassign-function-types
Warn when assigning to a function pointer with an unmatching signature. This usually happens in cases like assigning the null function to an entity's .think function pointer.

-Wcpp
Show warnings created using the preprocessor's '#warning' directive.

-Wmultifile-if
Warn if there's a preprocessor *#if* spanning across several files.

-Wdouble-declaration
Warn about multiple declarations of globals. This seems pretty common in QC code so you probably do not want this unless you want to clean up your code.

-Wconst-var
The combination of *const* and *var* is not illegal, however different compilers may handle them differently. We were told, the intention is to create a function-pointer which is not assignable. This is exactly how we interpret it. However for this interpretation the `var` keyword is considered superflu-

ous (and philosophically wrong), so it is possible to generate a warning about this.

-Wmultibyte-character
Warn about multibyte character constants, they do not work right now.

-Wternary-precedence
Warn if a ternary expression which contains a comma operator is used without enclosing parenthesis, since this is most likely not what you actually want. We recommend the **-fcorrect-ternary** option.

-Wunknown-pragmas
Warn when encountering an unrecognized `#pragma` line.

-Wunreachable-code
Warn about unreachable code. That is: code after a return statement, or code after a call to a function marked as ’noreturn’.

-Wdebug
Enable some warnings added in order to help debugging in the compiler. You won’t need this.

-Wunknown-attribute
Warn on an unknown attribute. The warning will inlclude only the first token inside the enclosing attribute-brackets. This may change when the actual attribute syntax is better defined.

-Wreserved-names
Warn when using reserved names such as `nil`.

-Wuninitialized-constant
Warn about global constants (using the `const` keyword) with no assigned value.

-Wuninitialized-global
Warn about global variables with no initializing value. This is off by default, and is added mostly to help find null-values which are supposed to be replaced by the untyped ’nil’ constant.

-Wdifferent-qualifiers
Warn when a variables is redeclared with a different qualifier. For example when redeclaring a variable as ´var´ which was previously marked ´const´.

-Wdifferent-attributes
Similar to the above but for attributes like `[[noreturn]]`.

-Wdeprecated
Warn when a function is marked with the attribute "[[deprecated]]". This flag enables a warning on calls to functions marked as such.

-Wparenthesis
Warn about possible mistakes caused by missing or wrong parenthesis, like an assignment in an ’if’ condition when there’s no additional set of parens around the assignment.

-Wunsafe-types
When passing variadic parameters via **...**`(N)` it can happen that incompatible types are passed to functions. This enables several warnings when static typechecking cannot guarantee consistent behavior.

-Wbreakdef
When compiling original id1 QC there is a definition for ‘break‘ which conflicts with the ’break’ keyword in GMQCC. Enabling this will print a warning when the definition occurs. The definition is ignored for both cases.

-Wconst-overwrite

When compiling original QuakeWorld QC there are instances where code overwrites constants. This is considered an error, however for QuakeWorld to compile it needs to be treated as a warning instead, as such this warning only works when −std=qcc.

-Wdirective-inmacro

Warn about the use of preprocessor directives inside macros.

-Wbuiltins

When using a function that is not explicitly defined, the compiler will search its intrinsics table for something that matches that function name by appending "__builtin_" to it. This behaviour may be unexpected, so enabling this will produce a diagnostic when such a function is resolved to a builtin.

-Winexact-compares

When comparing an inexact value such as '1.0/3.0' the result is pathologically wrong. Enabling this will trigger a compiler warning on such expressions.

COMPILE FLAGS

-fdarkplaces-string-table-bug

Add some additional characters to the string table in order to compensate for a wrong boundcheck in some specific version of the darkplaces engine.

-fadjust-vector-fields

When assigning to field pointers of type *.vector* the common behaviour in compilers like *fteqcc* is to only assign the x-component of the pointer. This means that you can use the vector as such, but you cannot use its y and z components directly. This flag fixes this behaviour. Before using it make sure your code does not depend on the buggy behaviour.

-fftepp

Enable a partially fteqcc-compatible preprocessor. It supports all the features used in the Xonotic codebase. If you need more, write a ticket.

-fftepp-predefs

Enable some predefined macros. This only works in combination with ´−fftepp' and is currently not included by '−std=fteqcc'. The following macros will be added:

```
__LINE__
__FILE__
__COUNTER__
__COUNTER_LAST__
__RANDOM__
__RANDOM_LAST__
__DATE__
__TIME__
__FUNC__
```

Note that `__FUNC__` is not actually a preprocessor macro, but is recognized by the parser even with the preprocessor disabled.

Note that fteqcc also defines `__NULL__` which becomes the first global. Assigning it to a vector does not yield the same result as in gmqcc where `__NULL__` is defined to `nil` (See **-funtyped-nil**), which will cause the vector to be zero in all components. With fteqcc only the first component will be 0, while the other two will become the first to of the global return value. This behavior is odd and relying on it should be discouraged, and thus is not supported by gmqcc.

-fftepp-mathdefs
Enable math constant definitions. This only works in combination with ´−fftepp' and is currently not included by '−std=fteqcc'. The following macros will be added:

```
M_E
M_LOG2E
M_LOG10E
M_LN2
M_LN10
M_PI
M_PI_2
M_PI_4
M_1_PI
M_2_PI
M_2_SQRTPI
M_SQRT2
M_SQRT1_2
M_TAU
```

-fftepp-indirect-expansion
Enable indirect macro expansion. This only works in combination with '-fftepp' and is currently not included by '-std=fteqcc'. Enabling this behavior will allow the preprocessor to operate more like the standard C preprocessor in that it will allow arguments of macros which are macro-expanded to be substituted into the definition of the macro.

As an example:

```
#define STR1(x) #x
#define STR2(x) STR1(x)
#define THE_ANSWER 42
#define THE_ANSWER_STR STR2(THE_ANSWER) /* "42" */
```

With this enabled, an expansion of THE_ANSWER_STR will yield the string "42". With this disabled an expansion of THE_ANSWER_STR will yield "THE_ANSWER"

-frelaxed-switch
Allow switch cases to use non constant variables.

-fshort-logic
Perform early out in logical AND and OR expressions. The final result will be either a 0 or a 1, see the next flag for more possibilities.

-fperl-logic
In many languages, logical expressions perform early out in a special way: If the left operand of an AND yeilds true, or the one of an OR yields false, the complete expression evaluates to the right side. Thus `true && 5` evaluates to 5 rather than 1.

-ftranslatable-strings
Enable the underscore intrinsic: Using `_("A string constant")` will cause the string immediate to get a name with a "dotranslate_" prefix. The darkplaces engine recognizes these and translates them in a way similar to how gettext works.

-finitialized-nonconstants
Don't implicitly convert initialized variables to constants. With this flag, the *const* keyword is required to make a constant.

-fassign-function-types

If this flag is not set, (and it is set by default in the qcc and fteqcc standards), assigning function pointers of mismatching signatures will result in an error rather than a warning.

-flno

Produce a linenumber file along with the output .dat file.

-fcorrect-ternary

Use C's operator precedence for ternary expressions. Unless your code depends on fteqcc-compatible behaviour, you'll want to use thi soption.

-fsingle-vector-defs

Normally vectors generate 4 defs, once for the vector, and once for its components with _x, _y, _z suffixes. This option prevents components from being listed.

-fcorrect-logic

Most QC compilers translate `if(a_vector)` directly as an IF on the vector, which means only the x-component is checked. This option causes vectors to be cast to actual booleans via a NOT_V and, if necessary, a NOT_F chained to it.

```
if (a_vector) // becomes
if not(!a_vector)
// likewise
a = a_vector && a_float // becomes
a = !!a_vector && a_float
```

-ftrue-empty-strings

An empty string is considered to be true everywhere. The NOT_S instruction usually considers an empty string to be false, this option effectively causes the unary not in strings to use NOT_F instead.

-ffalse-empty-strings

An empty string is considered to be false everywhere. This means loops and if statements which depend on a string will perform a NOT_S instruction on the string before using it.

-futf8

Enable utf8 characters. This allows utf-8 encoded character constants, and escape sequence codepoints in the valid utf-8 range. Effectively enabling escape sequences like '\{x2211}'.

-fbail-on-werror

When a warning is treated as an error, and this option is set (which it is by default), it is like any other error and will cause compilation to stop. When disabling this flag by using –fno-bail-on-werror, compilation will continue until the end, but no output is generated. Instead the first such error message's context is shown.

-floop-labels

Allow loops to be labeled, and allow 'break' and 'continue' to take an optional label to decide which loop to actually jump out of or continue.

```
for :outer (i = 0; i < n; ++i) {
    while (inner) {
        ...;
        if (something)
            continue outer;
    }
}
```

-funtyped-nil

Adds a global named 'nil' which is of no type and can be assigned to anything. No typechecking will be performed on assignments. Assigning to it is forbidden, using it in any other kind of expression is also not allowed.

Note that this is different from fteqcc's __NULL__: In fteqcc, __NULL__ maps to the integer written as '0i'. It's can be assigned to function pointers and integers, but it'll error about invalid instructions when assigning it to floats without enabling the FTE instruction set. There's also a bug which allows it to be assigned to vectors, for which the source will be the global at offset 0, meaning the vector's y and z components will contain the OFS_RETURN x and y components.

In that gmqcc the nil global is an actual global filled with zeroes, and can be assigned to anything including fields, vectors or function pointers, and they end up becoming zeroed.

-fpermissive

Various effects, usually to weaken some conditions.

with **-funtyped-nil**

Allow local variables named `nil`. (This will not allow declaring a global of that name.)

-fvariadic-args

Allow variadic parameters to be accessed by QC code. This can be achieved via the '...' function, which takes a parameter index and a typename.

Example:

```
void vafunc(string...count) {
    float i;
    for (i = 0; i < count; ++i)
        print(...(i, string), "\n");
}
```

-flegacy-vector-maths

Most Quake VMs, including the one from FTEQW or up till recently Darkplaces, do not cope well with vector instructions with overlapping input and output. This option will avoid producing such code.

-fexpressions-for-builtins

Usually builtin-numbers are just immediate constants. With this flag expressions can be used, as long as they are compile-time constant.

Example:

```
void printA() = #1; // the usual way
void printB() = #2-1; // with a constant expression
```

-freturn-assignments

Enabiling this option will allow assigning values or expressions to the return keyword as if it were a local variable of the same type as the function's signature's return type.

Example:

```
float bar() { return 1024; }
float fun() {
    return = bar();
    return; // returns value of bar
}
```

`-funsafe-varargs`
When passing on varargs to a different functions, this turns some static error cases into warnings. Like when the caller’s varargs are restricted to a different type than the callee’s parameter. Or a list of unrestricted varargs is passed into restricted varargs.

`-ftypeless-stores`
Always use STORE_F, LOAD_F, STOREP_F when accessing scalar variables. This is somewhat incorrect assembly instruction use, but in all engines they do exactly the same. This makes disassembly output harder to read, breaks decompilers, but causes the output file to be better compressible.

`-fsort-operands`
In commutative instructions, always put the lower-numbered operand first. This shaves off 1 byte of entropy from all these instructions, reducing compressed size of the output file.

`-femulate-state`
Emulate OP_STATE operations in code rather than using the instruction. The desired fps can be set via -state-fps=NUM, defaults to 10. Specifying –state-fps implicitly sets this flag. Defaults to off in all standards.

`-farithmetic-exceptions`
Turn on arithmetic exception tests in the compiler. In constant expressions which trigger exceptions like division by zero, overflow, underflow, etc, the following flag will produce diagnostics for what triggered that exception.

`-fsplit-vector-parameters`
With this flag immediate vector literals which only ever appear as function parameters won’t be stored as vector immediates. Instead, the 3 floats making up the vector will be copied separately. Essentially this turns a vector-store instruction into 3 float-store instructions for such cases. This increases code size but can dramatically reduce the amount of vector globals, which is after all limited to 64k. There’s at least one known codebase where this lowers the number of globals from over 80k down to around 3k. In other code bases it doesn’t reduce the globals at all but only increases code size. Just try it and see whether it helps you.

`-fdefault-eraseable`
Force all expressions to be "eraseable" which permits the compiler to remove unused functions, variables and statements. This is equivlant to putting [[eraseable]] on all definitions. This is dangerous as it breaks auto cvars, definitions for functions the engine may be looking for and translatable strings. Instead, you can mark a definition with [[noerase]] to prevent this from happening.

OPTIMIZATIONS

`-Opeephole`
Some general peephole optimizations. For instance the code ‘a = b + c‘ typically generates 2 instructions, an ADD and a STORE. This optimization removes the STORE and lets the ADD write directly into A.

`-Otail-recursion`
Tail recursive function calls will be turned into loops to avoid the overhead of the CALL and RETURN instructions.

`-Ooverlap-locals`
Make all functions which use neither local arrays nor have locals which are seen as possibly uninitialized use the same local section. This should be pretty safe compared to other compilers which do not check for uninitialized values properly. The problem is that there’s QC code out there which really doesn’t initialize some values. This is fine as long as this kind of optimization isn’t used, but also, only as long as the functions cannot be called in a recursive manner. Since it’s hard to know

whether or not an array is actually fully initialized, especially when initializing it via a loop, we assume functions with arrays to be too dangerous for this optimization.

-Olocal-temps

This promotes locally declared variables to "temps". Meaning when a temporary result of an operation has to be stored somewhere, a local variable which is not 'alive' at that point can be used to keep the result. This can reduce the size of the global section. This will not have declared variables overlap, even if it was possible.

-Oglobal-temps

Causes temporary values which do not need to be backed up on a CALL to not be stored in the function's locals-area. With this, a CALL to a function may need to back up fewer values and thus execute faster.

-Ostrip-constant-names

Don't generate defs for immediate values or even declared constants. Meaning variables which are implicitly constant or qualified as such using the 'const' keyword.

-Ooverlap-strings

Aggressively reuse strings in the string section. When a string should be added which is the trailing substring of an already existing string, the existing string's tail will be returned instead of the new string being added.

For example the following code will only generate 1 string:

```
print("Hello you!\n");
print("you!\n"); // trailing substring of "Hello you!\n"
```

There's however one limitation. Strings are still processed in order, so if the above print statements were reversed, this optimization would not happen.

-Ocall-stores

By default, all parameters of a CALL are copied into the parameter-globals right before the CALL instructions. This is the easiest and safest way to translate calls, but also adds a lot of unnecessary copying and unnecessary temporary values. This optimization makes operations which are used as a parameter evaluate directly into the parameter-global if that is possible, which is when there's no other CALL instruction in between.

-Ovoid-return

Usually an empty RETURN instruction is added to the end of a void typed function. However, additionally after every function a DONE instruction is added for several reasons. (For example the qcvm's disassemble switch uses it to know when the function ends.). This optimization replaces that last RETURN with DONE rather than adding the DONE additionally.

-Ovector-components

Because traditional QC code doesn't allow you to access individual vector components of a computed vector without storing it in a local first, sometimes people multiply it by a constant like `'0 1 0'` to get, in this case, the y component of a vector. This optimization will turn such a multiplication into a direct component access. If the factor is anything other than 1, a float-multiplication will be added, which is still faster than a vector multiplication.

-Oconst-fold-dce

For constant expressions that result in dead code (such as a branch whos condition can be evaluated at compile-time), this will eliminate the branch and else body (if present) to produce more optimal code.

CONFIG

The configuration file is similar to regular .ini files. Comments start with hashtags or semicolons, sections are written in square brackets and in each section there can be arbitrary many key-value pairs.

There are 3 sections currently: `flags`, `warnings`, `optimizations`. They contain a list of boolean values of the form `VARNAME = true` or `VARNAME = false`. The variable names are the same as for the corresponding **`-W`**, **`-f`** or **`-O`** flag written with only capital letters and dashes replaced by underscores.

Here's an example:

```
# a GMQCC configuration file
[flags]
    FTEPP = true
    ADJUST_VECTOR_FIELDS = false
    LNO = true

[warnings]
    UNUSED_VARIABLE = false
    USED_UNINITIALIZED = true

[optimizations]
    PEEPHOLE = true
    TAIL_RECURSION = true
```

FILES

gmqcc.ini.example
: A documented example for a gmqcc.ini file.

SEE ALSO

`qcvm`(1)

AUTHOR

See <http://graphitemaster.github.com/gmqcc>.

BUGS

Currently the '−fftepp-predefs' flag is not included by '−std=fteqcc', partially because it is not entirely conformant to fteqcc.

Please report bugs on <http://github.com/graphitemaster/gmqcc/issues>, or see <http://graphitemaster.github.com/gmqcc> on how to contact us.