From www.gnu.org/software/autoconf/manual:

4.8.1 Preset Output Variables

Some output variables are preset by the Autoconf macros. Some of the Autoconf macros set additional output variables, which are mentioned in the descriptions for those macros. See Output Variable Index, for a complete list of output variables. See Installation Directory Variables, for the list of the preset ones related to installation directories. Below are listed the other preset ones, many of which are precious variables (see Setting Output Variables, AC_ARG_VAR).

The preset variables which are available during `config.status` (see Configuration Actions) may also be used during `configure` tests. For example, it is permissible to reference '`$srcdir`' when constructing a list of directories to pass via option `-I` during a compiler feature check. When used in this manner, coupled with the fact that `configure` is always run from the top build directory, it is sufficient to use just '`$srcdir`' instead of '`$top_srcdir`'.

— Variable: **CFLAGS**

Debugging and optimization options for the C compiler. If it is not set in the environment when `configure` runs, the default value is set when you call AC_PROG_CC (or empty if you don't). `configure` uses this variable when compiling or linking programs to test for C features.

If a compiler option affects only the behavior of the preprocessor (e.g., `-D` *name*), it should be put into CPPFLAGS instead. If it affects only the linker (e.g., `-L` *directory*), it should be put into LDFLAGS instead. If it affects only the compiler proper, CFLAGS is the natural home for it. If an option affects multiple phases of the compiler, though, matters get tricky. One approach to put such options directly into CC, e.g., `CC='gcc -m64'`. Another is to put them into both CPPFLAGS and LDFLAGS, but not into CFLAGS.

However, remember that some `Makefile` variables are reserved by the GNU Coding Standards for the use of the "user"—the person building the package. For instance, CFLAGS is one such variable.

Sometimes package developers are tempted to set user variables such as CFLAGS because it appears to make their job easier. However, the package itself should never set a user variable, particularly not to include switches that are required

for proper compilation of the package. Since these variables are documented as being for the package builder, that person rightfully expects to be able to override any of these variables at build time. If the package developer needs to add switches without interfering with the user, the proper way to do that is to introduce an additional variable. Automake makes this easy by introducing`AM_CFLAGS` (see Flag Variables Ordering), but the concept is the same even if Automake is not used.

— Variable: **configure_input**

A comment saying that the file was generated automatically by `configure` and giving the name of the input file. `AC_OUTPUT` adds a comment line containing this variable to the top of every makefile it creates. For other files, you should reference this variable in a comment at the top of each input file. For example, an input shell script should begin like this:

```
#!/bin/sh
# @configure_input@
```

The presence of that line also reminds people editing the file that it needs to be processed by `configure` in order to be used.

— Variable: **CPPFLAGS**

Preprocessor options for the C, C++, Objective C, and Objective C++ preprocessors and compilers. If it is not set in the environment when `configure` runs, the default value is empty. `configure` uses this variable when preprocessing or compiling programs to test for C, C++, Objective C, and Objective C++ features.

This variable's contents should contain options like `-I`, `-D`, and `-U` that affect only the behavior of the preprocessor. Please see the explanation of `CFLAGS` for what you can do if an option affects other phases of the compiler as well.

Currently, `configure` always links as part of a single invocation of the compiler that also preprocesses and compiles, so it uses this variable also when linking programs. However, it is unwise to depend on this behavior because the GNU Coding Standards do not require it and many packages do not use `CPPFLAGS` when linking programs.

See Special Chars in Variables, for limitations that `CPPFLAGS` might run into.

— Variable: **CXXFLAGS**

Debugging and optimization options for the C++ compiler. It acts like `CFLAGS`, but for C++ instead of C.

— Variable: **DEFS**

`-D` options to pass to the C compiler. If `AC_CONFIG_HEADERS` is called, `configure` replaces '`@DEFS@`' with `-DHAVE_CONFIG_H` instead (see Configuration Headers). This variable is not defined while `configure` is performing its tests, only when creating the output files. See Setting Output Variables, for how to check the results of previous tests.

— Variable: **ECHO_C**
— Variable: **ECHO_N**
— Variable: **ECHO_T**

How does one suppress the trailing newline from `echo` for question-answer message pairs? These variables provide a way:

```
echo $ECHO_N "And the winner is... $ECHO_C"
sleep 100000000000
echo "${ECHO_T}dead."
```

Some old and uncommon `echo` implementations offer no means to achieve this, in which case `ECHO_T` is set to tab. You might not want to use it.

— Variable: **ERLCFLAGS**

Debugging and optimization options for the Erlang compiler. If it is not set in the environment when `configure` runs, the default value is empty. `configure` uses this variable when compiling programs to test for Erlang features.

— Variable: **FCFLAGS**

Debugging and optimization options for the Fortran compiler. If it is not set in the environment when `configure` runs, the default value is set when you call `AC_PROG_FC` (or empty if you don't). `configure` uses this variable when compiling or linking programs to test for Fortran features.

— Variable: **FFLAGS**

Debugging and optimization options for the Fortran 77 compiler. If it is not set in the environment when `configure` runs, the default value is set when you

call AC_PROG_F77 (or empty if you don't). configureuses this variable when compiling or linking programs to test for Fortran 77 features.

— Variable: **LDFLAGS**

Options for the linker. If it is not set in the environment when configure runs, the default value is empty. configure uses this variable when linking programs to test for C, C++, Objective C, Objective C++, and Fortran features.

This variable's contents should contain options like -s and -L that affect only the behavior of the linker. Please see the explanation of CFLAGS for what you can do if an option also affects other phases of the compiler.

Don't use this variable to pass library names (-l) to the linker; use LIBS instead.

— Variable: **LIBS**

-l options to pass to the linker. The default value is empty, but some Autoconf macros may prepend extra libraries to this variable if those libraries are found and provide necessary functions, see Libraries.configure uses this variable when linking programs to test for C, C++, Objective C, Objective C++, and Fortran features.

— Variable: **OBJCFLAGS**

Debugging and optimization options for the Objective C compiler. It acts like CFLAGS, but for Objective C instead of C.

— Variable: **OBJCXXFLAGS**

Debugging and optimization options for the Objective C++ compiler. It acts like CXXFLAGS, but for Objective C++ instead of C++.

— Variable: **builddir**

Rigorously equal to '.'. Added for symmetry only.

— Variable: **abs_builddir**

Absolute name of builddir.

— Variable: **top_builddir**

The relative name of the top level of the current build tree. In the top-level directory, this is the same as `builddir`.

— Variable: **top_build_prefix**

The relative name of the top level of the current build tree with final slash if nonemtpy. This is the same as `top_builddir`, except that it contains zero or more runs of `../`, so it should not be appended with a slash for concatenation. This helps for `make` implementations that otherwise do not treat `./file` and `file` as equal in the toplevel build directory.

— Variable: **abs_top_builddir**

Absolute name of `top_builddir`.

— Variable: **srcdir**

The name of the directory that contains the source code for that makefile.

— Variable: **abs_srcdir**

Absolute name of `srcdir`.

— Variable: **top_srcdir**

The name of the top-level source code directory for the package. In the top-level directory, this is the same as `srcdir`.

— Variable: **abs_top_srcdir**

Absolute name of `top_srcdir`.