

6.3 MVS-specific C Definitions

The following sections describe aspects of Waterloo C that are specific to the OS, MVS and MVS/XA implementation of the C language.

6.3.1 Basic Arithmetic Types

The following definitions are used for arithmetic types in Waterloo C.

Type	Size	Range
signed char	8 bits	-128..127
char	8 bits	0..255
unsigned char	8 bits	0..255
short int	16 bits	-32,768..32,767
unsigned short int	16 bits	0..65,535
int	32 bits	-2,147,483,648..2,147,483,647
unsigned int	32 bits	0..4,294,967,295
long int	32 bits	-2,147,483,648..2,147,483,647
unsigned long int	32 bits	0..4,294,967,295
float	32 bits	fraction 6 digits (base 16) exponent -64..+63 (base 16)
double	64 bits	fraction 14 digits (base 16) exponent -64..+63 (base 16)

The library header files `<limits.h>` and `<float.h>` contain several constant definitions that can be used in a program to determine the characteristics of the basic C types.

6.3.2 Filename Format for the #include Directive

A source file that is included into another source file with the #include preprocessor directive (usually called a **header** file) can be specified as follows:

```
#include "membername.ddname"
```

ddname must be allocated to the partitioned data set (PDS) that contains membername. An alternate form

```
#include <name.h>
```

is reserved for the inclusion of C library header files, such as <stdio.h>. In either case, the compiler translates the file specifier to upper case.

6.3.3 Special Characters

Some of the characters that are used in the C language are not available on a number of terminals, such as the IBM 3278 and 3279. As described in the previous section, an ANSI C trigraph sequence can be used in place of any of the characters for which a sequence is defined. Alternatively, the following operator character substitutions are offered.

- The C **exclusive-or** operator, usually the ASCII circumflex '^', can be replaced by either the EBCDIC turnstile '⋈' or the EBCDIC cent-sign '¢'.
- **Array subscripting** is usually indicated by left and right square brackets '[' and ']'. For Waterloo C, the two-character sequences "(:" and "):" can be used instead.
- The C **inclusive-or** operator can be either the solid vertical bar '|' or the broken vertical bar '¦'.

For the specific character code values, see the table of ASCII and EBCDIC codes in the *Waterloo C Run-time Library Reference*.

6.3.4 Programming Restrictions

The following restrictions apply to programs that are compiled with the Waterloo C compiler:

- A function that is defined as taking a variable number of arguments is restricted to a maximum of 31 unspecified parameters.
- Waterloo C generates object and assembler output that contains global symbols that are the first eight characters of the corresponding C extern symbol, translated to upper-case, with underscore characters ('_') replaced with dollar sign characters ('\$'). Therefore, extern symbols must be unique within the first eight characters from all other extern symbols in a program.

6.4 Compiler Command Format and Options

The Waterloo C compiler can be used in a batch (JCL) environment, or interactively with TSO, to compile C programs into object or assembler output.

6.4.1 Input/Output Data Definitions

The following data definitions (DD statements/TSO allocations) can be specified when programs are compiled with the Waterloo C compiler:

✓ SYSIN	C source code to be compiled
✓ OBJ1	code and read-only data object output; this DD is not required when the PPC option is specified
OBJ2	read/write data object output; this DD is used only when the SPLIT or RENT option is specified
ASM1	code and read-only data assembler output; this DD is used only when the ASM or ASMSRC option is specified
ASM2	read/write data assembler output; this DD is used only when the ASM or ASMSRC option, and the SPLIT or RENT option is specified
DBGINFO	debugger symbolic information output; this DD is used only when the DBG option is specified
SYSLIB	default C source code include (header) library; alternate DD's of the form TYPE may be provided if a file is included into the C source code with the following:

```
#include "name.type"
```

SYSLIB will be searched if the member NAME is not found in the partitioned data set TYPE.

✓ SYSPRINT	preprocessor output; this DD is used only when the PPC option is specified
✓ SYSLIST	compiler diagnostic output

6.4.2 Compiling from JCL

The JCL procedure CW is provided with the system and may be installed in a system procedure library. To use the compiler with this procedure definition, the following job step can be used:

```
//CWSTEP EXEC CW,
//      PARS='option1 option2...',
//      INCLIB='include-library-dsn',
//      OUTC=sysout-parm,
//      REG='nnnK'
```

All of the CW keyword parameters are optional and the following default values are used when a parameter is not specified:

PARMS	all compiler options assume their default values, as defined below
INCLIB	SYSLIB is allocated to the standard include library 'WCOS.CLIB.H'
OUTC	SYSLIST is defined as SYSOUT=A
REG	the compiler is run with a region size of 1000K

Because the CW procedure contains DD statements to define SYSLIB and SYSLIST, these DD names should not be defined in JCL that uses CW. A DD statement for the source code input SYSIN must always be specified. Other DD statements may be required, depending on the compiler options that are specified.

The following options can be specified for the C compiler from JCL. Each option is described in detail following the list of options.

ALIGN	enables address alignment of integer and float type values
ASCIIOUT	converts all string and character constants to equivalent ASCII values in the generated code
ASM	writes assembler code to the DD ASM1; if the RENT or SPLIT option is also specified, assembler code for read/write data is written to the DD ASM2

Reference

ASMSRC	same as ASM, except that the C source lines appear as comments in the assembler output
AUX name	uses the auxiliary storage allocation information file name to define offsets for read/write data from a global base register; name can be a DD name or a fully-qualified data set name
COL column	input from the source file (and include files that are delimited with a double quote (") on the #include directive) is read, beginning in the specified column of each line
CSECT name	sets the code section (CSECT) definition in the object output to @name
DBG	generates symbolic information for use by the Waterloo C Debugger and writes the output to the DD DBGINFO
OPT	enables compiler optimization (the default)
NOOPT	disables compiler optimization
OS	generates code which uses an OS style register linkage convention with a reserved run-time stack pointer register
OSENTRY	generates code which uses an OS style register linkage convention with no run-time stack
OSFUNC	generates code which uses an OS style register linkage convention where a special call is generated during the function prologue to obtain a stack pointer value
PPC	writes preprocessor output (without compiling it) to the DD SYSPRINT
PRM id token	simulates the addition of the preprocessor directive #define id token

at the top of the source file; both `id` and `token` must be specified; to define more than one preprocessor identifier, specify the option once for each definition

RENT	generates reentrant code using register based data references; assembler and object code for executable code and read-only data, and read/write data is written to the DD's ASM1, ASM2, OBJ1 and OBJ2, subject to the ASM, ASMSRC and NOTEXT options
SPLIT	partitions executable code and read-only data, and read/write data into separate assembler and object output DD's ASM1, ASM2, OBJ1 and OBJ2, subject to the ASM, ASMSRC and NOTEXT options; note that the RENT and SPLIT options cannot be specified together
STACKCHK	generates code which checks for a run-time stack overflow condition at the start of every function
NOTEXT	suppresses the generation of object output
TRUNC column	characters after the specified column in the source file (and include files that are delimited with a double quote (") on the <code>#include</code> directive) are ignored
WNG	generates warning messages for any questionable statements
NOWNG	suppresses the display of all warning messages
NOZERO	disables the initialization of otherwise uninitialized static variables to zero values

The number of errors that are diagnosed is returned as the completion code for the step.

For example, consider the following sample JCL:

Reference

```
//HELLOCW JOB 'ACCOUNTING INFORMATION'
//*
//COMPILE EXEC CW,
// PARM='ASMSRC CSECT HELLO DBG PRM FIVE 5 PRM Six 6'
//SYSIN DD DSN=WCOS.EXAMPLE.C(HELLO),DISP=SHR
//ASML DD SYSOUT=A,DCB=(LRECL=80,BLKSIZE=80,RECFM=FB)
//DBGINFO DD DSN=WCOS.EXAMPLE.DBGINFO(HELLO),DISP=SHR
//OBJ1 DD DSN=WCOS.EXAMPLE.OBJ(HELLO),DISP=SHR
```

In this example, the source code in 'WCOS.EXAMPLE.C(HELLO)' is compiled with the following options in effect:

```
ASMSRC
CSECT HELLO
DBG
PRM FIVE 5
PRM Six 6
```

The generated assembler code is written to SYSOUT class A, and the object code and debugger symbolic information are written to permanent data sets.

6.4.3 Compiling from TSO

The TSO CLIST CW is provided with the system and may be installed in a system command library. When the compiler is used with this CLIST, one of the following command formats can be used:

```
CW name(member) { option1 option2... }

CW 'src-dsname{(member)}' OBJ(obj-dsname{(member)}) +
  { option1 option2... }
```

In the first format, where name is not a fully qualified data set name, the CLIST allocates the source code input DD SYSIN to 'user.name.C(member)', where user is the usual TSO user prefix. The object code DD OBJ1 is allocated to 'user.name.OBJ(member)'. If the DBG option is specified, the debugger symbolic information output DD DBGINFO is allocated to 'user.name.DBGINFO(member)'.

If the second command format is used, where 'src-dsname' or 'src-dsname(member)' is a fully qualified data set name, an object output data set must be specified with the OBJ option. Note that when no member name is

specified, 'src-dsname' must be a sequential data set.

For both command formats, compiler messages are written to the terminal. In addition, the ERRS option can be used to specify a data set or PDS member where messages should be written. Also, the CLIST allocates the default include library DD SYSLIB to the standard include library 'WCOS.CLIB.H'.

The following options can be specified for the C compiler from the CW command. Each option is described in detail following the list of options.

ALIGN	enables address alignment of integer and float type values
ASCIIOUT	converts all string and character constants to equivalent ASCII values in the generated code
ASM(dsname)	writes assembler code to the data set dsname, which is allocated to DD ASM1; if the RENT or SPLIT option is also specified, assembler code for read/write data is written to the DD ASM2
ASMSRC(dsname)	same as ASM, except that the C source lines appear as comments in the assembler output
AUX(dsname)	uses the auxiliary storage allocation information file dsname to define offsets for read/write data from a global base register
COL(column)	input from the source file (and include files that are delimited with a double quote (") on the #include directive) is read, beginning in the specified column of each line
CSECT(name)	sets the code section (CSECT) definition in the object output to @name
DBG	generates symbolic information for use by the Waterloo C Debugger and writes the output to the DD DBGINFO; if the command line did not specify a fully qualified data set name for the source code input, DBGINFO is allocated to 'user.name.DBGINFO(member)'

Reference

ERRS (dsname)	allocates the diagnostic output DD SYSLIST to dsname
OBJ (dsname)	allocates the object output DD OBJ1 to dsname
OPT	enables compiler optimization (the default)
NOOPT	disables compiler optimization
OS	generates code which uses an OS style register linkage convention with a reserved run-time stack pointer register
OSENTRY	generates code which uses an OS style register linkage convention with no run-time stack
OSFUNC	generates code which uses an OS style register linkage convention where a special call is generated during the function prologue to obtain a stack pointer value
PPC(dsname)	writes preprocessor output (without compiling it) to the data set dsname
PRMS('PRM id1 token1 PRM id2 token2...')	<p>simulates the addition of the preprocessor directives</p> <pre>#define id1 token1 #define id2 token2 . . .</pre> <p>at the top of the source file; PRM, id and token must be specified for each preprocessor identifier that is to be defined</p>
RENT	generates reentrant code using register based data references; assembler and object code for executable code and read-only data, and read/write data is written to the DD's ASM1, ASM2, OBJ1 and OBJ2, subject to the ASM, ASMSRC and NOTEXT options
SPLIT	partitions executable code and read-only data, and read/write data into separate assembler and object output

DD's ASM1, ASM2, OBJ1 and OBJ2, subject to the ASM, ASMSRC and NOTEXT options; note that the RENT and SPLIT options cannot be specified together

STACKCHK	generates code which checks for a run-time stack overflow condition at the start of every function
NOTEXT	suppresses the generation of object output
TRACE	traces the execution of the CLIST statements
TRUNC (column)	characters after the specified column in the source file (and include files that are delimited with a double quote (") on the #include directive) are ignored
WNG	generates warning messages for any questionable statements
NOWNG	suppresses the display of all warning messages
NOZERO	disables the initialization of otherwise uninitialized static variables to zero values

The number of errors that are diagnosed is returned as the command return code.

For example, consider the following CW command, assuming that the TSO user prefix is WCOS:

```
CW EXAMPLE(HELLO) ASMSRC(EXAMPLE.ASM(HELLO)) +
  CSECT(HELLO) DBG PRMS('PRM FIVE 5 PRM Six 6')
```

With this command, the source code in 'WCOS.EXAMPLE.C(HELLO)' is compiled with the following options in effect:

```
ASMSRC
CSECT HELLO
DBG
PRM FIVE 5
PRM Six 6
```

The following are other examples of CW commands:

Reference

```
CW EXAMPLE (HELLO)
```

```
CW EXAMPLE (HELLO) DBG
```

```
CW EXAMPLE (HELLO) OBJ (''WCOS.EXAMPLE.OBJ (HELLO) '')
```

6.4.4 The ALIGN Option

The ALIGN option ensures that objects (such as structure elements) of type short int or short unsigned int are stored on a machine halfword (2-byte) boundary, that objects of type int, float, unsigned int, long int and unsigned long int are stored on a machine fullword (4-byte) boundary and that objects of type double are stored on a machine doubleword (8-byte) boundary.

If the ALIGN option is not specified, only function parameters are aligned. As part of the run-time conventions, the run-time stack pointer is maintained to a machine doubleword alignment. On some System/370 architecture processors, object alignment can significantly improve the performance of a program.

6.4.5 The ASCIIOUT Option

The ASCIIOUT option converts all string and character constants in the generated object code from the EBCDIC character set to an equivalent ASCII character, if such a character exists. Explicitly defined character constants in the program, such as `\xff`, and characters for which no ASCII equivalent exists, are left unchanged.

Although MVS is fundamentally an EBCDIC system, C programs can process ASCII data. See the descriptions of the `atoe` and `etoe` functions and the table of ASCII and EBCDIC character codes in the *Waterloo C Run-time Library Reference* for more information.

6.4.6 The ASM and ASMSRC Options

The ASM option causes the compiler to create output that contains System/370 assembler code. Output is written to the DD ASM1 and, if one of the RENT or SPLIT options is also specified, assembler code for read/write data definitions is written to the DD ASM2. The ASMSRC option performs the same function, with the

additional feature that the output that is created contains the C source lines as comments. For both ASM and ASMSRC, the compiler creates the output with a fixed record format and a logical record length of 80 so that it can be assembled using the standard System/370 assembler. There are no assembler macros in the generated output. If the ALIGN option (see above) is not specified with the ASM option, it may be necessary to specify the NOALIGN option when the assembler is run.

6.4.7 The AUX Option

This option provides the compiler with the name of an **auxiliary storage allocation information** file for use in the definition of register-based global read/write data. For example,

```
CW EXAMPLE (TEST) AUX (EXAMPLE.AUX (MYFILE) )
```

uses the storage allocation definitions in 'user.EXAMPLE.AUX (MYFILE) '. The section *Register-based Global Data Using AUX Data Sets* contains information about the use of this facility.

6.4.8 The COL and TRUNC Options

The COL and TRUNC options can be used to restrict the compiler to processing C source lines within specified columns. It is particularly useful if the source file contains sequence numbers.

6.4.9 The DBG Option

The DBG option is specified when the file that is being compiled is to be used with the Waterloo C Debugger. Output containing symbolic information for use by the debugger is created by the compiler and written to the DD DBGINFO.

6.4.10 The OPT and NOOPT Options

The Waterloo C compiler normally applies several code-improvement techniques to increase the efficiency of the generated code. If NOOPT is specified, the optimization phase of the compiler is disabled. For some C files, this may reduce the

Reference

time required to compile the file; however, the generated code will usually be less efficient.

6.4.11 The OS, OSENTRY and OSFUNC Options

When any of these options is specified, the generated code will use an OS style linkage convention instead of the normal Waterloo C function linkage. The following is a summary of the three varieties of OS linkage that are available to C programmers.

OS	generates code that assumes that register 12 (R12) is reserved for use as a run-time stack register and that the function base address is contained in R15 when the function is called
OSENTRY	generates code that requires no run-time stack register and that makes no assumption about the function base address; each function defines a static save area
OSFUNC	generates code that contains a call to a routine (\$GETSPTR) to retrieve a run-time stack pointer value; the function base address is assumed to be contained in R15

The *OS-style Register Conventions* section of this manual contains more information about the use of each of these options.

6.4.12 The PPC Option

The PPC option causes the compiler to perform only the preprocessing phase of compilation. Preprocessor output is written to the DD SYSPRINT. This output can in turn be compiled by specifying it as input to a subsequent compile step.

6.4.13 The PRM Option

The PRM option allows the user to specify preprocessor macro definitions when the program is compiled. For each occurrence of the PRM option in the compiler option string, a preprocessor token definition is performed. For example,

```
CW EXAMPLE(TEST) PRMS('PRM FIVE 5 PRM Six 6')
```

is equivalent to inserting the following macro definitions before the first line of the source file `EXAMPLE.C (TEST)`:

```
#define FIVE 5
#define Six 6
```

6.4.14 The SPLIT and RENT Options

The `SPLIT` option partitions modifiable C variables, for which a fixed storage location is allocated by the compiler, into a separate code section (CSECT) and output DD from the read-only data and executable code. Object and assembler code for the read-only portion is written to the usual output DD's `OBJ1` and `ASM1` and code for read/write data definitions is written to the output DD's `OBJ2` and `ASM2`, subject to whether the `ASM`, `ASMSRC` or `NOTEXT` options are specified.

The `RENT` option partitions modifiable data and constant code and data as with the `SPLIT` option, but generates special, register-based references for the modifiable, relocatable data. This option is useful in a program that is to be run in the MVS link pack area (LPA).

Only one of the `RENT` or `SPLIT` options should be specified on the `CW` command line. See the *Separation of Global Data* and *Register-based Global Data* sections of this manual for more information about the use of these options.

6.4.15 The STACKCHK Option

The `STACKCHK` option causes code to be generated at the start of every function to verify that sufficient space on the run-time stack is available for the function to run correctly. A function call that is equivalent to

```
gsignal( _STACK_OVER );
```

is executed if insufficient stack space is available. See the *Run-time Stack Overflow Checking* section of this manual for more information about the use of this option.

6.4.16 The NOTEXT Option

By default, the Waterloo C compiler writes object output to the DD OBJ1. The NOTEXT option suppresses the generation of this output.

6.4.17 The WNG and NOWNG Options

The Waterloo C compiler generates two classes of warning messages, called **conditional** warnings and **unconditional** warnings, neither of which causes the termination of compilation. When neither NOWNG nor WNG is specified on the command line, unconditional warnings are written to the DD's STDOUT and SYSLIST; conditional warnings are suppressed. The NOWNG option suppresses the printing of both conditional and unconditional warnings. The WNG option causes both conditional and unconditional warnings to be written.

6.4.18 The NOZERO Option

All modifiable C variables for which a fixed storage location is allocated by the compiler (subject to the RENT option) are normally initialized to some value: either to 0 by default, or to the values explicitly specified in a C initializer. The default initialization to 0 is suppressed when the NOZERO option is specified.

For a C source file with large array definitions that are initialized at run-time, the size of the generated object code can be substantially reduced by the use of this option. If the ASM or ASMSRC option is specified, the normal **DC** (define constant) assembler directive in the generated assembler output is replaced with a **DS** (define storage) directive.