C Compiler User's Guide

Updated for the new millenium

C Compiler User's Guide: Updated for the new millenium

Revision History

Revision C June 1983

Table of Contents

Acknowledgements

Thw OS-9 C Compiler was written by James McCosh with OS-9 implementation assistance from Terry Crane and Kim Kempf. The Relocatable Assembler, Linker, and Profiler was edited by Wes Camden and Ken Kaplan.

Differences between Versions 1.1 and 1.0

This package contains the OS-9 C Compiler Version 1.1. Many improvements and bug fixes have been incorporated since the V1.0 release. If you are upgrading from V1.0, be *absolutely sure* to all *all* the files from the V1.1 disks. None of the compiler sections or the library is compatible with V1.0 files. Any ".r" or ".a" files produced by the V1.0 compiler should not be assembled or linked with any ".a" or ".r" files produced by the V1.1 compiler. To be safe, recompile/reassemble *all* ".a" and ".r" files with V.1.1.

This update include appendices for the C Compiler User's Guide describing the compiler error messages, compiler phase command lines, interfacing C functions to BASIC09, and an overview of the relocating macro assembler.

The remainder of this notice describes the changes made since V1.0.

General:

The compiler code generator and c.opt have been improved to produce even smaller object code. This, and improved source coding, has resulted in a 1 page decrease in the size of c.comp and a 4 page decrease in c.pass1.

Chapter 1. The C Compiler System

1.1. Introduction

The "C" programming language is rapidly growing in popularity and seems destined to become one of the most popular programming languages used for microcomputers. The rapid rise in the use of C is not surprising. C is an incredibly versatile and efficient language that can handle tasks that previously would have required complex assembly language programming.

1.2. The Language Implementation

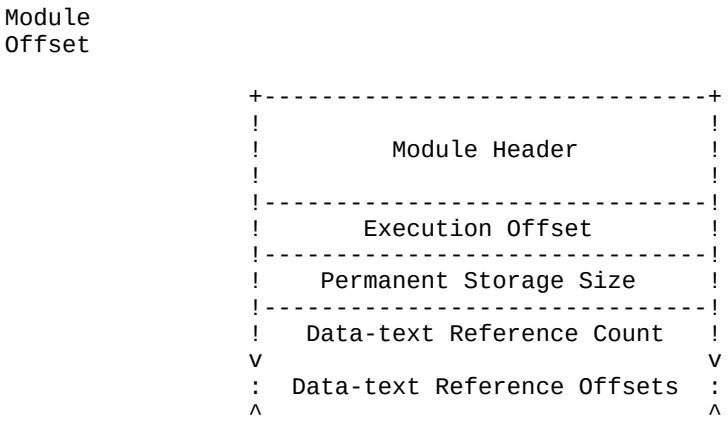
OS-9 C is implemented almost exactly as described in 'The C Programming Language' by Kernighan and Ritchie (hereafter referred to as K&R).

Allthough this version of C follows the specification faithfully, there are some differences. The differences mostly reflect parts of C that are obsolete or the constraints imposed by memory size limitations.

Chapter 2. Characteristics of Compiled Programs

2.1. The Object Code Module

The compiler produces position-independent, reentrant 6809 code in a standard OS-9 memory module format. The format of an executable program module is shown below. Detailed descriptions of each section of the module are given on following pages.



2.1.1. Module Header

This is a standard module header with the type/language byte set to $11 (Program + 6809 Object Code), and the attribute/revision byte set to $81 (Reentrant + 1).

Chapter 3. C System Calls

This section of the C compiler manual is a guide to the system calls available from C programs.

It is NOT intended as a definitive description of OS-9 service requests as these are described in the OS-9 SYSTEM PROGRAMMER'S manual. However, for most calls, enough information is available here to enable the programmer to write systems calls into programs without looking further.

The names used for the system calls are chosen so that programs transported from other machines or operating systems should compile and run with as little modification as possible. However, care should be taken as the parameters and returned values of some calls may not be compatible with those on other systems. Programmers that are already familiar with OS-9 names and values should take particular care. Some calls do not share the same names as the OS-9 assembly language equivalents. The assembly language equivalent call is shown, where there is one, on the relevant page of the C call description, and a cross-reference list is provided for those already familiar with OS-9 calls.

The normal error indication on return from a system call is a returned value of -1. The relevant error will be found in the predefined int "errno". Errno always contains the error from the last erroneous system call. Definitions for the errors for inclusion in the programs are in "<errno.h>".

In the "SEE ALSO" sections on the following pages, unless otherwise stated, the references are to other system calls.

Where "#include" files are shown, it is not mandatory to include them, but it might be convenient to use the manifest constants defined in them rather than integers; it certainly makes for more readable programs.

Abort

Name

`Abort` — stop the program and produce a core dump

Synopsis

```
abort(void);
```

Description

This call causes a memory image to be written out to the file "core" in the current directory, and then the program exits with a status of 1.

Abs

Name

`Abs` — Absolute value

Synopsis

```
abs(int i);
```

Description

Placeholder

Access

Name

`Access` — give file accessibility

Synopsis

```
access(char *name, int perm);
```

Description

Placeholder

Chain

Name

`Chain` — load and execute a new program

Synopsis

```
chain(type arg1);
```

Description

Placeholder

Chdir

Name

`Chdir, Chxdir` — change directory

Synopsis

```
chdir(type arg1);
```

Description

Placeholder

Chmod

Name

`Chmod` — change access permissions of a file

Synopsis

```
#include <modes.h>
chmod(type arg1);
```

Description

Placeholder

Chown

Name

`Chown` — change the ownership of a file

Synopsis

```
chown(type arg1);
```

Description

Placeholder

Close

Name

`Close` — close a file

Synopsis

```
close(type arg1);
```

Description

Placeholder

Crc

Name

`Crc` — compute a cyclic redundancy count

Synopsis

```
crc(type arg1);
```

Description

Placeholder

Creat

Name

`Creat` — create a file

Synopsis

```
#include <modes.h>
creat(type arg1);
```

Description

Placeholder

Defdrive

Name

`Defdrive` — get default system drive

Synopsis

```
defdrive(type arg1);
```

Description

Placeholder

Dup

Name

`Dup` — duplicate an open path number

Synopsis

```
dup(type arg1);
```

Description

Placeholder

Exit

Name

`Exit, _Exit` — task termination

Synopsis

```
exit(type arg1);
```

Description

Placeholder

Getpid

Name

`Getpid` — get the task id

Synopsis

```
getpid(type arg1);
```

Description

Placeholder

Getstat

Name

`Getstat` — get file status

Synopsis

```
#include <sgstat.h>
getstat(type arg1);
```

Description

Placeholder

Getuid

Name

`Getuid` — return user id

Synopsis

```
getuid(type arg1);
```

Description

Placeholder

Intercept

Name

`Intercept` — set function for interrupt processing

Synopsis

```
intercept(type arg1);
```

Description

Placeholder

Kill

Name

`Kill` — send an interrupt to a task

Synopsis

```
#include <signal.h>
kill(type arg1);
```

Description

Placeholder

Lseek

Name

`Lseek` — position in file

Synopsis

```
lseek(type arg1);
```

Description

Placeholder

Mknod

Name

`Mknod` — create a directory

Synopsis

```
#include <modes.h>
mknod(type arg1);
```

Description

Placeholder

Modload

Name

`Modload` — return a pointer to a module structure

Synopsis

```
#include <module.h>
modload(type arg1);
```

Description

Placeholder

Munlink

Name

`Munlink` — unlink a module

Synopsis

```
#include <module.h>
munlink(type arg1);
```

Description

Placeholder

_os9

Name

`_os9` — system call interface from C programs

Synopsis

```
#include <os9.h>
_os9(type arg1);
```

Description

Placeholder

Open

Name

`Open` — open a file for read/write access

Synopsis

```
open(type arg1);
```

Description

Placeholder

Os9fork

Name

`Os9fork` — create a process

Synopsis

```
os9fork(type arg1);
```

Description

Placeholder

Pause

Name

`Pause` — halt and wait for interrupt

Synopsis

```
pause(type arg1);
```

Description

Placeholder

Prerr

Name

`Prerr` — print error message

Synopsis

```
prerr(type arg1);
```

Description

Placeholder

Read

Name

`Read` — read from a file

Synopsis

```
read(type arg1);
```

Description

Placeholder

Sbrk

Name

`Sbrk` — request additional working memory

Synopsis

```
sbrk(type arg1);
```

Description

Placeholder

Setpr

Name

`Setpr` — set process priority

Synopsis

```
setpr(type arg1);
```

Description

Placeholder

Setime

Name

`Setime, Getime` — set and get system time

Synopsis

```
#include <time.h>
setime(type arg1);
```

Description

Placeholder

Setuid

Name

`Setuid` — set user id

Synopsis

```
setuid(type arg1);
```

Description

Placeholder

Setstat

Name

`Setstat` — set file status

Synopsis

```
#include <sgstat.h>
setstat(type arg1);
```

Description

Placeholder

Signal

Name

`Signal` — catch or ignore interrupts

Synopsis

```
#include <signal.h>
signal(type arg1);
```

Description

Placeholder

Stacksize

Name

`Stacksize, Freemem` — obtain stack reservation size

Synopsis

```
stacksize(void);
freemem(void);
```

Description

Placeholder

Strass

Name

`Strass` — byte by byte copy

Synopsis

```
_strass(type arg1);
```

Description

Placeholder

Tsleep

Name

`Tsleep` — put process to sleep

Synopsis

```
tsleep(type arg1);
```

Description

Placeholder

Unlink

Name

`Unlink` — remove directory entry

Synopsis

```
unlink(type arg1);
```

Description

Placeholder

Wait

Name

`Wait` — wait for task termination

Synopsis

```
wait(type arg1);
```

Description

Placeholder

Write

Name

`Write, Writeln` — write to a file or device

Synopsis

```
write(type arg1);
```

Description

Placeholder

Chapter 4. C Standard Library

The Standard Library contains functions which fall into two classes: high-level I/O and convenience.

The high-level I/O functions provide facilities that are normally considered part of the definition of other languages; for example, the FORMAT "statement" of Fortran. In addition, automatic buffering of I/O channels improves the speed of file access because fewer system calls are necessary.

atof

Name

`atof` — Placeholder

Synopsis

```
atof(type arg1);
```

Description

Placeholder

fflush

Name

`fflush` — Placeholder

Synopsis

```
#include <stdio.h>
fflush(type arg1);
```

Description

Placeholder

feof

Name

`feof` — Placeholder

Synopsis

```
#include <stdio.h>
feof(type arg1);
```

Description

Placeholder

findstr

Name

`findstr` — Placeholder

Synopsis

```
findstr(type arg1);
```

Description

Placeholder

fopen

Name

`fopen` — Placeholder

Synopsis

```
#include <stdio.h>
fopen(type arg1);
```

Description

Placeholder

fread

Name

`fread` — Placeholder

Synopsis

```
#include <stdio.h>
fread(type arg1);
```

Description

Placeholder

fseek

Name

`fseek` — Placeholder

Synopsis

```
#include <stdio.h>
fseek(type arg1);
```

Description

Placeholder

getc

Name

`getc` — Placeholder

Synopsis

```
#include <stdio.h>
getc(type arg1);
```

Description

Placeholder

gets

Name

`gets` — Placeholder

Synopsis

```
#include <stdio.h>
gets(type arg1);
```

Description

Placeholder

isalpha

Name

`isalpha` — Placeholder

Synopsis

```
#include <ctype.h>
isalpha(type arg1);
```

Description

Placeholder

l3tol

Name

`l3tol` — Placeholder

Synopsis

```
l3tol(type arg1);
```

Description

Placeholder

longjmp

Name

`longjmp` — Placeholder

Synopsis

```
#include <setjmp.h>
longjmp(type arg1);
```

Description

Placeholder

malloc

Name

`malloc` — Placeholder

Synopsis

```
malloc(type arg1);
```

Description

Placeholder

mktemp

Name

`mktemp` — Placeholder

Synopsis

```
mktemp(type arg1);
```

Description

Placeholder

putc

Name

`putc` — Placeholder

Synopsis

```
#include <stdio.h>
putc(type arg1);
```

Description

Placeholder

puts

Name

`puts` — Placeholder

Synopsis

```
#include <stdio.h>
puts(type arg1);
```

Description

Placeholder

qsort

Name

`qsort` — Placeholder

Synopsis

```
qsort(type arg1);
```

Description

Placeholder

scanf

Name

`scanf` — Placeholder

Synopsis

```
#include <stdio.h>
scanf(type arg1);
```

Description

Placeholder

setbuf

Name

`setbuf` — Placeholder

Synopsis

```
#include <stdio.h>
setbuf(type arg1);
```

Description

Placeholder

sleep

Name

`sleep` — Placeholder

Synopsis

```
sleep(type arg1);
```

Description

Placeholder

strcat

Name

`strcat` — Placeholder

Synopsis

```
strcat(type arg1);
```

Description

Placeholder

system

Name

`system` — Placeholder

Synopsis

```
system(type arg1);
```

Description

Placeholder

toupper

Name

`toupper` — Placeholder

Synopsis

```
#include <ctype.h>
toupper(type arg1);
```

Description

Placeholder

ungetc

Name

`ungetc` — Placeholder

Synopsis

```
#include <stdio.h>
ungetc(type arg1);
```

Description

Placeholder

Appendix A. Compiler Generated Error Messages

The error codes are shown in both hexadecimal (first column) and decimal (second column). Error codes other than those listed are generated by programming languages or user programs.

already a local variable

Variable has already been declared at the current block level. (8.1, 9.2)

argument : <text>

Error from preprocessor. Self-explanatory. Most common cause of this error is not being able to find an include file.

Appendix B. Compiler Phase Command Lines

This appendix decsribes the command lines and options for the individual compiler phases. Each phase of the compiler may be executed separately. The following information describes the options available to each phase.

cc1 & cc2 (C executives)

cc [options] file {file} [options]

Appendix C. Interfacing to Basic09

The object code generated by the Microware C Compiler can be made callable from the BASIC09 "RUN" statement. Certain portions of a BASIC09 program written in C can have a drmatic effect on execution speed. To effectively utilize this feature, one must be familiar with both C and BASIC09 internal data representation and procedure calling protocol.

Appendix D. Relocating Macro Assembler Reference

This appendix gives a summary of the operation of the "Relocating Macro Assembler" (named c.asm as distributed with the C Compiler). This appendix and the example assembly source files supplied with the C compiler should provide the basic information on how to use the "Relocating Macro Assembler" to create relocatable-object format files (ROF). It is further assumed that you are familiar with the 6809 instruction set and mnemonics. See the Microware Relocating Assembler Manual for a more detailed description. The main function of this appendix is to enable the reader to understand the output produced by c.asm. The Relocating Macro Assembler allows programs to be compiled separately and then linked together, and it also allows macros to be defined within programs.

Differences between the Relocating Macro Assembler (RMA) and the Microware Interactive Assembler (MIA):

RMA does not have an interactive mode. Only a disk file is allowed as input.

RMA output is an ROF file. The ROF file must be processed by the linker to produce an executable OS9 memory module. The layout of the ROF file is described later.

RMA has a number of new directives to control the placement of code and data in the executable module. Since RMA does not produce memory modules, the MIA directives "mod" and "emod" are not present. Instead, new directives PSECT and VSECT control the allocation of code and data areas by the linker.

RMA has no equivalent to the MIA "setdp" directive. Data (and DP) allocation is handled by the linker.

D.1. Symbolic Names

A symbolic name is valid if it consists of from one to nine uppercase or lowercase characters, decimal digits or the characters "$", "_", "." or "@". RMA does not fold lowercase letters to uppercase. The names "Hi.you" and "HI.YOU" are distinct names.