

VAX/VMS 培训教材

C 语言培训教材

2000 系列资料出版中心
沈阳计算所信息技术开发公司

前言

本手册将有关 VAX C 编程语言的参考信息及在 VAX/VMS 操作系统下开发和调试 VAX C 程序的有关信息综合起来。也包括有关从 VAX/VMS 中将 C 程序移植到其它操作系统中或自其它操作系统中向 VAX/VMS 移植 C 程序的有关信息, 同时还包括 VAX C 与其它语言工具的不同点。

读者对象

本手册的读者对象是需要学习 VAX C 的有经验的编程人员, 要了解 VAX C 与其它语言不同点的用户及需要参考有关信息的有经验的 VAX C 用户。读者应熟悉某一种高级语言。VAX C 的新用户应先读第一章, VAX C 语言起步, 读者应对 DIGITAL Command Language(DCL)有一定的了解, 那些还不太了解或要参考有关 DCL 信息的读者, 可参阅第 9 章, 程序开发导论。

本手册的结构

本手册有三个部分:

- 第一部分。VAX C 编程语言提供包括语言指导及 VAX C 词汇的 VAX C 结构的有关信息。
- 第二部分。使用 VAX/VMS 进行程序开发, 提供有关 DIGITAL Command Language 及其它语言解释器, 建立及维护源程序、调试程序, 使用 VAX/VMS 记录管理服务子程序(RMS), 及在混合语言环境中编程的信息。
- 第三部分。VAX C 运行时间库(RTL)提供有关在不同的操作系统间进行移植的信息, 以及函数和宏的类描述。

本手册还有 6 个附录:

- 附录 A 是 VAX C 运行时间库及其它 C 工具的运行时间库, 它提供了 VAX C RTL 函数及宏的构成以及其它 C 工具的相应函数。
- 附录 B 是 VAX C 的定义模块, 它提供了用于在源码中包含定义及 VAX C RTL 宏的 .H 文件的描述。
- 附录 C 是 VAX C 运行时间模块及入口点, 它提供了用于此工具的 VAX 运行时间模块及 VAX C 模块的描述。
- 附录 D 是 VAX C 编译器信息, 它提供一个由出错信息及为避免错误所采取的可能措施的字母顺序目录。
- 附录 E 是 VAX C 编译器的列表文件格式, 它提供了通过使用在第十一章—编译, 连接和运行 VAX C 程序—中介绍的命令行限定词而产生的列表文件的例子。
- 附录 F 是 VAX/VMS 符号调试器的命令句法, 它提供了所有调试命令及限定词的句法。

相关文献

在使用 VAX C 进行编程时，会发现下列文献会很有用：

- Installing VAX C —— 是为安装 VAX C 软件的系统程序员准备的。
- VAX C Language Summary —— 是为快速查找 VAX C 句法描述的程序员准备的。
- VAX / VMS Master Index —— 是为使用 VAX 机器体系及 VAX / VMS 系统服务子程序的程序员准备的。这个索引开列有关访问 VAX / VMS 的单个主题的手册名单。
- The C Programming Language —— 是为那些需要此第一章——开始使用 VAX C——更多更详细的参考资料的读者准备的。如在 The C Programming Language 中说明的那样，VAX C 具有许多特点，而且对 C 语言有许多改进的地方，因而，Programming in VAX C 应作为 VAX C 的全部描述的参考书使用。

目 录

第一部分 C 编程语言

第一章 VAX C 语言起步			
1.1 C 程序设计语言的背景	1	4.2.4 结构和联合的引用	35
1.2 VAX C 程序设计语言	2	4.3 VAX C 语言操作符讨论	35
1.3 编写第一个程序	3	4.4 一元表达式与操作符	38
1.4 输入/输出	4	4.4.1 负运算与逻辑表达式(!)	38
1.5 控制程序流	6	4.4.2 变量增1与减1(++ --)	38
1.5.1 if 语句	6	4.4.3 求地址和间接指针(&*)	39
1.5.2 开关(switch)语句	7	4.4.4 求反(~)	39
1.5.3 循环	8	4.4.5 强制转换成特定类型(cast 运算符)	39
1.6 值、地址和指针	10	4.4.6 计算变量与数据类型的大小 (sizeof)	40
1.7 集合	12	4.5 二元表达式与操作符	40
1.7.1 数组与字符串	12	4.5.1 加法运算符(+/-)	40
1.7.2 结构与联合	13	4.5.2 乘法运算符(* / %)	41
第二章 程序结构		4.5.3 相等运算符(== !=)	41
2.1 函数定义	17	4.5.4 关系操作符(> < <= >=)	41
2.1.1 主函数与函数标识符	20	4.5.5 逐位操作符(& ^)	41
2.1.2 参数与自变量	20	4.5.6 逻辑操作符(&&)	42
2.2 标识符	21	4.5.7 移位操作符(<< >>)	43
2.3 关键字	22	4.6 条件表达式与操作符(?:)	43
2.4 块	23	4.7 赋值表达式与操作符(= += -= * / =)	43
2.5 注释	24	4.8 逗号表达式与操作符(,)	45
第三章 语句		4.9 数据类型转换	45
3.1 控制流语句	25	4.9.1 操作对象的转换	45
3.1.1 空语句	25	4.9.2 函数自变量的转换	46
3.1.2 goto 语句	25	第五章 数据类型和说明	
3.1.3 带标号的语句	26	5.1 常量、定义和说明	47
3.2 表达式与块作为语句	26	5.1.1 变量说明的格式	48
3.2.1 表达式语句	26	5.2 函数返回值的数据类型	48
3.2.2 复合语句	26	5.2.1 空数据类型	49
3.3 条件语句	27	5.3 标量的说明和类型	49
3.3.1 if 语句	27	5.3.1 整型常量	49
3.3.2 开关(switch)语句	27	5.3.2 整型变量	50
3.3.2.1 开关语句内的说明	29	5.3.3 字符常量	50
3.4 循环语句	29	5.3.3.1 转义序列	51
3.4.1 for 语句	29	5.3.4 字符变量	51
3.4.2 while 语句	30	5.3.5 浮点常数	51
3.4.3 do 语句	31	5.3.6 浮点变量	52
3.5 中断(Interruting)语句	31	5.3.7 指针	52
3.5.1 break 语句	31	5.3.8 枚举类型	53
3.5.2 continue 语句	31	5.4 集合类型的定义和初始化	55
3.5.3 return 语句	32	5.4.1 数组	55
第四章 表达式与操作符		5.4.1.1 数组的初始化	56
4.1 左值与右值	33	5.4.2 字符串常量	57
4.2 初等表达式与操作符	34	5.4.3 字符串变量	58
4.2.1 带括号的表达式	34	5.4.4 结构和联合	58
4.2.2 函数调用	34		
4.2.3 数组引用[]	35		

5.4.4.1	结构和联合的说明	59	6.6.2	说明符 globalvalue	78
5.4.4.2	结构或联合成员的引用	60	6.6.3	全局枚举类型	79
5.4.4.3	结构的初始化	61	6.7	存储类修饰符	80
5.4.4.4	位字段	62	6.7.1	修饰符 noshare	80
5.5	解释说明语句	63	6.7.2	修饰符 readonly	81
5.6	类型定义(Type def)	65	第七章	预处理器控制行	
第六章	存储类与存储分配		7.1	标记替换	82
6.1	工作域	66	7.1.1	常量标识符	83
6.1.1	编译和连接处理	66	7.1.2	预处理器替换	84
6.1.2	说明语句的位置	67	7.1.3	宏替换	85
6.1.3	编译域和连接域	68	7.1.4	替换行的列表	86
6.1.4	程序例子	68	7.1.5	取消定义	86
6.2	存储分配	70	7.2	公共数据字典的抽取	87
6.2.1	程序段的属性	70	7.2.1	#dictionary 控制行	87
6.2.2	VAX C 语言产生的程序段	71	7.2.2	CDD 支持的数据类型	88
6.3	内部存储类	72	7.3	条件编译	90
6.3.1	说明符 auto	73	7.4	包含(Inclusion)文件	91
6.3.2	说明符 register	74	7.4.1	用角括号(< >)包含	91
6.4	静态存储类	74	7.4.2	用双引号(" ")包含	92
6.5	外部存储类	75	7.4.3	文本模块的包含	93
6.6	全局存储类	76	7.5	行号的说明	93
6.6.1	说明符 globaldef 和 globalref	76	7.6	模块名和标识的说明	94
6.6.1.1	全局与外部存储类的比较	78	第八章	VAX C 语言词汇表	

第二部分 使用 VAX/VMS 进行程序开发

第九章 程序开发导论

9.1	Digital 命令语言(DCL)	106
9.1.1	DCL 文件说明	108
9.1.1.1	缺省文件说明	108
9.1.1.2	暂时缺省	108
9.1.2	逻辑名	110
9.1.2.1	逻辑名翻译	112
9.1.3	改变当前目录	112
9.1.4	使用命令程序	112
9.1.5	库	114
9.1.5.1	库命令句法	115
9.1.5.2	文本库介绍	116
9.1.5.3	定义缺省文本库	118
9.1.5.4	VAX C 语言的缺省文本库	118
9.1.5.5	目标代码库介绍	119
9.1.5.6	缺省目标模块库	120
9.2	DEC/SHELL	121
9.2.1	起动 DEC/SHELL	122
9.2.2	DEC/SHELL 命令行句法	122
9.2.3	有效的 DEC/SHELL 文件说明	123
9.2.4	I/O 重定向	123
9.2.5	VAX C 语言与 DEC/SHELL	124

第十章 使用 EDT 编写源程序

10.1	编辑过程的控制	125
10.1.1	起动编辑器	126
10.1.2	初始化编辑过程	128
10.1.3	中断编辑过程	128
10.1.4	保护你的编辑	129
10.1.5	HELP EDT 命令	129
10.1.6	结束编辑过程	129
10.2	行编辑模式	130
10.2.1	在行编辑模式下建立文件	130
10.2.2	编辑正存在的文件	130
10.2.3	范围指定	131
10.2.4	文本缓冲区	132
10.2.5	文件中的操作	133
10.2.6	插入新文本	134
10.2.7	删除和替换文本	134
10.2.8	移动文本	135
10.2.9	替换文本	135
10.2.10	将文本移到另外一个文件或从另外的文件中将文本移入	136
10.3	屏幕编辑	137
10.3.1	进入屏幕模式和从屏幕模式退出	139
10.3.2	屏幕模式中的 HELP	139
10.3.3	光标操作	140

10.3.4	范围描述	141	12.4.1	调试器常驻符号	184
10.3.5	插入文本	141	12.4.2	全局符号的引用	184
10.3.6	删除和恢复文本	142	12.4.3	程序位置的引用	184
10.3.7	移动文本	142	12.4.4	程序变量的引用	185
10.4	日志文件选项	143	12.4.4.1	访问标量变量	185
10.5	EDT 对 VAX C 语言编程者的帮助	144	12.4.4.2	访问数组	186
10.5.1	结构化的制表符	144	12.4.4.3	访问字符串	187
10.5.2	特殊目的的键定义	144	12.4.4.4	访问结构与联合	188
10.5.3	起动命令文件	146	12.4.5	控制工作域及运行时间符号表	191
第十一章	编译、连接和运行 VAX C 语言程序		12.4.5.1	控制变量的工作域	193
11.1	程序的构成	148	12.5	调试器选项	195
11.2	CC 命令	149	12.5.1	小键盘模式	195
11.2.1	CC 命令格式	149	12.5.2	屏幕模式	196
11.2.2	文件说明及编译单元	150	12.6	对 VAX C 语言表达式的支持	199
11.2.3	指定库文件	150	第十三章	使用 VAX 记录管理服务(RMS)	
11.2.4	命令限定词	151	13.1	RMS 文件组织	202
11.2.5	编译器诊断信息及出错情况	157	13.1.1	顺序文件组织	203
11.3	Link 命令	159	13.1.2	相对文件组织	203
11.3.1	Link 命令格式	160	13.1.3	索引文件组织	203
11.3.2	连接器输入文件	161	13.2	记录访问模式	204
11.3.3	指定映象文件限定词	162	13.3	RMS 记录格式	204
11.3.4	指定调试程序限定词	163	13.4	RMS 函数	205
11.3.5	连接器信息	163	13.5	用 RMS 写 VAX C 语言程序	206
11.3.6	连接器输出文件	163	13.5.1	初始化文件访问块	207
11.4	RUN 命令	164	13.5.2	初始化记录访问块	208
11.4.1	用 RUN 命令执行映象	164	13.5.3	初始化扩展特征块	208
11.4.2	命令行参数	165	13.5.4	初始化名称块	209
11.4.3	映象退出	166	13.6	RMS 程序例子	209
11.4.4	运行时间错误	166	第十四章	混合语言编程	
11.4.5	中断程序	167	14.1	调用栈	222
11.4.6	返回 DCL 解释器的值	168	14.1.1	调用帧	222
第十二章	调试 VAX C 语言程序		14.1.2	参数表	223
12.1	VAX C 语言调试过程样本	170	14.2	由立即值来传递参数	224
12.2	控制调试过程	172	14.2.1	检查系统服务返回值	226
12.2.1	起动调试器	173	14.2.2	由立即值传递浮点参数	227
12.2.2	使用调试器命令程序	174	14.3	由引用传递参数	228
12.2.3	初始化调试过程	174	14.4	由描述符传递参数	230
12.2.4	在日志文件中记录调试过程	175	14.5	可变长度参数表	232
12.2.5	中断调试过程	175	14.6	返回状态值	234
12.2.6	HELP 命令	176	14.6.1	返回状态值格式	234
12.2.7	结束编辑过程	176	14.6.2	操纵返回状态值	235
12.3	控制程序执行	176	14.6.3	测试成功或者失败	236
12.3.1	调试器命令	176	14.6.4	测试特定的返回状态值	237
12.3.2	GO 命令	177	14.7	与用其它语言编写的过程共享数据	238
12.3.3	STEP 命令	177	14.7.1	与 FORTRAN 的 Common 块 语句共享的程序段	238
12.3.4	进入函数和自函数中返回	179	14.7.2	与 PL/I 外部变量共享程序段	238
12.3.5	断点	180	14.7.3	与 MACRO 程序共享程序段	240
12.3.6	跟踪点	182			
12.3.7	观察点	183			
12.4	运行时间符号表	183			

第三部分 VAX C 语言运行时间库

第十五章 VAX C 语言运行时间库

- 15.1 VAX C 语言运行时间库的实现 243
 - 15.1.1 把 VAX C 语言 RTL 当作可共享的映象使用 244
 - 15.1.2 宏 244
- 15.2 VAX C 语言 RTL 的函数和宏的句法 246
 - 15.2.1 DEC / Shell 文件说明 247
- 15.3 在 VAX / VMS 上的输入和输出 ... 248
 - 15.3.1 RMS 记录和文件的格式 250
 - 15.3.2 流式访问 RMS 记录文件 ... 251
- 15.4 可移植性涉及的具体问题 253

第十六章 标准输入 / 输出函数及宏指令

- 16.1 标准输入 / 输出及文件存取 257
- 16.2 转化规范 258
 - 16.2.1 输入信息的转化 258
 - 16.2.2 输出信息的转化 260
- 16.3 打开和关闭文件 261
 - 16.3.1 fclose 261
 - 16.3.2 fdopen 262
 - 16.3.3 fopen 262
 - 16.3.4 frcopen 263
- 16.4 从文件中读出数据 263
 - 16.4.1 getc, fgetc, getw 264
 - 16.4.2 getw 264
 - 16.4.3 getc 264
 - 16.4.4 fgetc 264
 - 16.4.5 frcad 265
 - 16.4.6 fscanf, sscanf 265
 - 16.4.7 sscanf 266
 - 16.4.8 ungetc 267
- 16.5 写入文件 267
 - 16.5.1 fprintf, sprintf 267
 - 16.5.2 fput 267
 - 16.5.3 fputs 267
 - 16.5.4 fwrite 268
 - 16.5.5 putc, fputc, putw 268
 - 16.5.6 putw 269
 - 16.5.7 sprintf 269
- 16.6 文件操作 269
 - 16.6.1 fflush 269
 - 16.6.2 fseek 269
 - 16.6.3 ftell 270
 - 16.6.4 rewind 270
- 16.7 附加的标准输入 / 输出函数与宏指令 271
 - 16.7.1 access 271
 - 16.7.2 clearerr 271
 - 16.7.3 delete 272
 - 16.7.4 feof 272
 - 16.7.5 ferror 272

- 16.7.6 fgetname 272
- 16.7.7 mktemp 273
- 16.7.8 setbuf 273
- 16.7.9 tmpfile 274
- 16.7.10 tmpnam 274
- 16.8 程序实例 274

第十七章 终端输入与输出函数

- 17.1 终端输入输出与定义的文件指针 ... 276
- 17.2 终端输入输出函数 276
 - 17.2.1 getchar 276
 - 17.2.2 gets 276
 - 17.2.3 printf 277
 - 17.2.4 putchar 278
 - 17.2.5 puts 278
 - 17.2.6 scanf 278
- 17.3 程序实例 279

第十八章 UNIX 输入 / 输出函数与宏指令

- 18.1 UNIX 输入 / 输出及文件描述 281
- 18.2 打开和关闭文件 282
 - 18.2.1 close 282
 - 18.2.2 create 282
 - 18.2.3 dup, dup2 284
 - 18.2.4 dup2 285
 - 18.2.5 open 285
- 18.3 读出与写入 286
 - 18.3.1 read 286
 - 18.3.2 write 287
- 18.4 文件中指针的定位 287
 - 18.4.1 lseek 287
- 18.5 附加的 UNIX 输入 / 输出函数及宏指令 288
 - 18.5.1 fileno 288
 - 18.5.2 fstat, stat 289
 - 18.5.3 getname 290
 - 18.5.4 isapipe 290
 - 18.5.5 isatty 291
 - 18.5.6 stat 291
 - 18.5.7 ttyname 291
- 18.6 程序实例 292

第十九章 字符处理函数及宏指令

- 19.1 字符归类宏指令 293
 - 19.1.1 isalnum 295
 - 19.1.2 isalpha 296
 - 19.1.3 isascii 296
 - 19.1.4 iscntrl 296
 - 19.1.6 isgraph 297
 - 19.1.7 islower 297
 - 19.1.8 isprint 297
 - 19.1.9 ispunct 297
 - 19.1.10 isspace 297

19.1.11	issupper	298	21.1.14	hypot, cabs	315
19.1.12	isxdigit	298	21.1.15	ldexp	315
19.2	字符转换函数及宏指令	298	21.1.16	log, log10	316
19.2.1	ecvt, fcvt, gcvt	298	21.1.17	log10	316
19.2.2	fcvt	299	21.1.18	modf	316
19.2.3	gcvt	299	21.1.19	pow	316
19.2.4	toascii	299	21.1.20	rand, srand	317
19.2.5	tolower, __tolower	299	21.1.21	sin	317
19.2.6	toupper, __toupper	300	21.1.22	sinh	317
19.2.7	__tolower	300	21.1.23	sqrt	317
19.2.8	__toupper	300	21.1.24	srand	318
19.3	程序实例	300	21.1.25	tan	318
第二十章	字符串及表格处理函数与宏指令		21.1.26	tanh	318
20.1	字符串操作	303	21.2	程序举例	319
20.1.1	atoi, atol	303	第二十二章	错误处理函数	
20.1.2	atoi	304	22.1	错误处理函数	320
20.1.3	atol	304	22.1.1	abort	321
20.1.4	strcat, strncat	304	22.1.2	exit, __exit	321
20.1.5	strchr, strrchr	304	22.1.3	perror	322
20.1.6	strcmp, strncmp	305	22.1.4	__exit	322
20.1.7	strcpy, strncpy	305	22.2	信号处理函数	322
20.1.8	strcspn	306	22.2.1	alarm	323
20.1.9	strlen	306	22.2.2	gsignal	324
20.1.10	strncat	306	22.2.3	kill	325
20.1.11	strncmp	306	22.2.4	longjmp, setjmp	325
20.1.12	strncpy	306	22.2.5	pause	326
20.1.13	strpbrk	306	22.2.6	setjmp	326
20.1.14	strrchr	307	22.2.7	sigblock	326
20.1.15	strspn	307	22.2.8	signal	327
20.2	访问可变长度参数表	307	22.2.9	sigpause	327
20.2.1	va_arg	308	22.2.10	sigsetmask	328
20.2.2	va_count	308	22.2.11	sigstack	328
20.2.3	va_end	309	22.2.12	sigvec	328
20.2.4	va_start, va_startl	309	22.2.13	sleep	329
20.2.5	va_start_l	309	22.2.14	ssignal	329
20.3	程序实例	310	22.2.15	VAX C \$ESTABLISH	330
第二十一章	数学函数		22.3	程序举例	330
21.1	数学函数和 errno 定义模块	312	第二十三章	内存分配函数	
21.1.1	abs, fabs	313	23.1	VAX C 语言与内存分配	332
21.1.2	acos	313	23.1.1	brk, sbrk	332
21.1.3	asin	313	23.1.2	calloc, malloc(内存分配)	333
21.1.4	atan	313	23.1.3	cfree, free(内存释放)	333
21.1.5	atan2	313	23.1.4	free	333
21.1.6	cabs	314	23.1.5	malloc	334
21.1.7	ceil	314	23.1.6	realloc(内存重分配)	334
21.1.8	cos	314	23.1.7	sbrk	334
21.1.9	cosh	314	23.2	程序举例	334
21.1.10	exp	314	第二十四章	子进程函数	
21.1.11	fabs	315	24.1	VAX C 语言与子进程	336
21.1.12	floor	315	24.2	VAX C 语言中子进程的实例	336
21.1.13	frexp	315	24.3	exec 函数	338

24.3.1	execl, execl, execl, execl, execl	338	26.5.7	[W]clrtoebot	366
24.3.2	Exec 处理	339	26.5.8	[W]clrtoeol	366
24.3.3	Exec 错误条件	339	26.5.9	[no]crmode	366
24.4	进程同步	340	26.5.10	[W]delch	367
24.4.1	wait	340	26.5.11	[w]deleteln	367
24.5	读写数据	340	26.5.12	delwin	367
24.5.1	pipe	340	26.5.13	[no]echo	367
24.6	程序举例	343	26.5.14	endwin	368
第二十五章 系统函数			26.5.15	[w]erase	368
25.1	VAX C 语言和系统程序设计	348	26.5.16	[w]getch	368
25.2	检索进程信息	348	26.5.17	[w]getstr	368
25.2.1	ctermid	348	26.5.18	getyx	369
25.2.2	cavseid	348	26.5.19	[w]inch	369
25.2.3	getegid, geteuid, getgid, getuid	349	26.5.20	initscr	369
25.2.4	getenv	349	26.5.21	[w]insch	369
25.2.5	geteuid	349	26.5.22	[w]insertln	370
25.2.6	getgid	350	26.5.23	[w]insstr	370
25.2.7	getpid	350	26.5.24	longname	370
25.2.8	getuid	350	26.5.25	leaveok	371
25.3	修改进程信息	350	26.5.26	[w]move	371
25.3.1	chdir	350	26.5.27	mv[w]addch	371
25.3.2	chmod	350	26.5.28	mv[w]addstr	372
25.3.3	chown	351	26.5.29	mvcu	372
25.3.4	mkdir	351	26.5.30	mv[w]delch	372
25.3.5	nice	352	26.5.31	mv[w]getch	372
25.3.6	setgid, setuid	352	26.5.32	mv[w]getstr	373
25.3.7	setuid	352	26.5.33	mv[w]inch	373
25.3.8	umask	353	26.5.34	mv[w]insch	373
25.4	检索时间信息	353	26.5.35	mv[w]insstr	373
25.4.1	ctime	353	26.5.36	mvwin	374
25.4.2	ftime	353	26.5.37	newwin	374
25.4.3	localtime	354	26.5.38	[no]nl	374
25.4.4	time	354	26.5.39	overlay	375
25.4.5	times	355	26.5.40	overwrite	375
25.5	程序举例	355	26.5.41	[w]printw	375
第二十六章 curses 屏幕管理函数			26.5.42	[no]raw	375
26.1	Curses 术语	359	26.5.43	[w]refresh	376
26.1.1	用户定义的窗口	359	26.5.44	[w]scanw	376
26.2	开始用 Curses	360	26.5.45	scroll	376
26.3	预定义变量和常数	362	26.5.46	scrollok	376
26.4	光标移动	363	26.5.47	[w]setattr	377
26.5	Curses 函数和宏	364	26.5.48	subwin	377
26.5.1	[W]addch	364	26.5.49	[w]standend	378
26.5.2	[W]addstr	364	26.5.50	[w]standout	378
26.5.3	box	365	26.5.51	touchwin	378
26.5.4	[W]clear	365	26.5.52	wrapok	378
26.5.5	clearok	365			
26.5.6	[W]clrattr	366			

26.6 程序举例	379	附录 D VAX C 语言编译器消息	405
附录 A VAX C 语言 RTL(运行时间库)与其它 C 语言工具的 RTL(运行时间库)	382	附录 E VAX C 编译程序的列表格式	425
附录 B VAX C 语言定义模块	393	附录 F VAX/VMS 符号调试器命令和限定 词	436
附录 C VAX C 运行时间模块和入口点	395		

第一部分 VAX C 编程语言

第一部论述了有关 VAX C 语言特点的参考资料。

第一章 VAX C 语言起步

本章是为希望了解一种不熟悉语言的基本特征的有经验的程序员设计的，此点是通过详细例子与简短的注解实现的。本章包括了整个手册所有章节要点的说明，第一部分各章描述了 VAX C 程序语言要点，第一部分还讨论了如下问题：

- 程序结构
- 语句
- 表达式与操作符
- 数据类型与说明
- 存贮类与分配
- 预处理器控制行
- VAX C 语言术语

熟悉 C 程序设计语言与 VAX/VMS 系统的程序员，可从第二章，程序结构开始学习。

1.1 C 程序设计语言的背景

C 语言由于它尺寸小而灵活，由于它提供了丰实的操作符而方便了使用，而且具有强功能的现代控制流与数据结构，所以 C 语言是一种容易掌握的通用的程序设计语言。最初 C 语言是在 PDP-11 机上 UNIX 操作系统下设计与实现的。C 语言的设计者就 C 语言功能指出：

“C 语言…不依赖于任何一种操作系统或计算机；尽管由于它在编写操作系统程序方面的实用性而被称为“系统程序设计语言”，但它同样也成功的被用于编写大量的数字，文本处理及数据库程序。”

象汇编语言一样，C 语言不是为适应任何一种特定的应用需求而设计的。C 语言处理和存贮数据与现代机器结构相同。然而 C 语言不象汇编语言那样复杂，不依赖于机器(C 语言具有极强的可移植性)。一个程序可在几种不同的机器上使用几种不同的编译程序进行编译，且能运行它的源程序，则称这个程序是可移植的。

尽管 C 程序语言没有 ANSI 或其它的工业标准，但在实现功能上保持一致。如果 C 语言是跨系统可移植的，就必须具备这种一致性。这是语言最希望有的特征之一。因此当用户编译且运行程序时，不仅 C 源程序必须可移植，而且由语言特征所决定在众多系统上必须产生同样效果。

因为 C 语言是在 UNIX 系统环境下研制的，最后被用于改写大部分 UNIX 系统程序，因此在 C 语言中许多“标准”操作方法与 UNIX 有关。例如，UNIX 系统通过一个数字文件描述符来访问文件，因此 C 语言工具通过文件描述符提供“访问”文件的功能。在 UNIX 系统环境下，用户通过一个简单的命令结构就可实现将一个程序或命令的输出做为另一个程序或命令的输入，并且能建立同步或异步子进程，无需许多限制和系统保护，就能使用操作系统的特征。

一些“标准”C 语言结构包括预处理器控制行和函数与宏指令的运行时间库。在 UNIX 系统环境下，一个预处理器在编译前对源代码中含有的预处理器控制行任务进行处理。

因为 C 语言无输入输出信息的手段，通常由运行时间库来提供这种服务。如果一个运行时间库函数产生了非 UNIX 系统环境下产生的副作用，则这一函数的可移植性就有问题了。

1.2 VAX C 程序设计语言

VAX C 程序设计语言合并了 C 语言基本特征和大多数 C 语言编译程序的特征。然而，VAX C 语言也提供了本身特有的特征，即只在 VAX/VMS 操作系统环境下，能直接而有效地工作。用户必须选择 VAX C 语言中对他编程最重要的特征：跨系统的可移植性或 VAX/VMS 操作系统特性的高效使用。选择的特征集必有优点与缺点。

如果用户选择 VAX C 语言程序，则他的源程序具有极强的跨系统的可移植性，但在某种程度上会降低效率。为使 VAX C 语言编译程序模仿 UNIX 系统特征，且 VAX C 语言必须保持满意的可移植性，必然会在处理 VAX/VMS 系统特征时使效率降低。例如，鉴于 UNIX 系统使用称为文件描述符的结构存取文件，VAX 记录管理服务(RMS)用各种控制结构存取文件。在 VAX C 语言中，input 与 output 函数以 UNIX 系统同样方式存取文件，但编译程序实际上处理 RMS 结构，好象用户在 UNIX 系统环境下处理一样。但在某些场合中，除了程序执行慢以外。大多数 VAX/VMS 操作对用户是透明的。

VAX C 语言与其它 C 语言实现的最大区别——妨碍源码可移植性区别——是由 VAX/VMS 与 UNIX 系统之间不同产生的。例如，当 VAX/VMS 系统不象在 UNIX 系统环境下那样将大量的控制权分配给用户，很难使 VAXC 建立一种环境允许程序员有很大的控制权。I/O 重定向不是 VAX/VMS 系统命令行语法部分；在 VAX/VMS 系统中建立子进程不如在 UNIX 系统那样有效；VAX/VMS 高级语言不用象在 UNIX 系统中的语言那样的方法来实现预处理。在手册内，正文部分与索引部分，标明了 VAX C 语言与其它 C 语言实现之间的差别，因此用户可广泛使用不可移植的 VAX C 语言结构。

如果用户选择用 VAX C 语言编程使其在 VAX/VMS 系统下以有效的方式运行，在某种程度上，降低了程序移植到其它系统或从其它系统移过来的可能性。例如，在 VAX C 语言程序内，调用 VAX/VMS 系统运行时间库例程，即有此效果。

然而，也具有可移植性与有效地存取功能强的 VAX/VMS 系统环境！

一种特殊选项是用 VAX C 语言特殊结构和 DIGITAL 命令语言(DCL)(象 VAX C 语言预处理器代换与 DCL 命令行限定 /STANDARD=PORTABLE)编写具有 VAX/VMS 系统环境优点的可移植代码。这些结构仅允许在 VAX/VMS 系统下运行时, 执行一些代码段, 而在非 VAX/VMS 系统下运行其它代码段。可移植性的完整讨论, 参见第 15 章 VAX C 语言运行时间库信息。有关预处理器控制行的更多信息, 参见第 7 章, 预处理器控制行。有关 DCL 命令行信息, 参见第 11 章编译, 连接与执行 VAX C 语言程序。

1.3 编写第一个程序

尽管概念简单, 但用一种不熟悉的语言编写第 1 个程序可能有失败的经历。由于计算机以有效的处理数字而著称, 因此在这里第 1 个程序是将二数相加且将总和存在一个变量中。例 1-1 显示了此程序是如何实现的。

例 1-1: 一个简单的 VAX C 语言加法程序

```
① /* This program adds two numbers and places the sum in
   * the variable total. */
3 main( ) /* The function name "main" */
{ /* Begins function body */
③ int total; /* Variable of type "int" */
/* Blank lines are allowed */
④ total = 2 + 2; /* Answer placed in "total" */
} /* Ends the function body */
```

注解:

- ① 字符(/ *)与(* /)之间的正文是注释。注释内不能再放注释(即注释不能“嵌套”), 但注释可放在任何空白处。空白是在源代码内分隔代码的空格与空白行之间的区域。在下面的章节中, 定义了 VAX C 语言结构的许可空白。
- ② VAX C 语言程序包括了用户定义的不可嵌套的外部函数。本例中, 定义的函数名为“main”。在 VAX C 语言中, 程序可从名为 main 函数开始执行, 也可从使用“main-program”选择项的函数开始执行, 或两者均可; 如果用户指明的 main 函数不存在, 在外部引用函数时, 程序流中的第一个函数是缺省 main 函数。main 选择项是 VAX C 语言特有的, 因而是不可移植的。有关 main-program 选择项的语法及使用的进一步信息, 参见第 2 章程序结构。
VAX C 语言函数有用参数与变元交换信息的方法。由于在主函数定义中, 空括号内无参数; 因此, 在上例中, 主函数不能通过使用参数来接收信息。
为在函数定义中指明参数, 在括号内列出了参数标识符且用逗号(,)分隔开。必须在函数体开始以前说明参数。如果在函数 main 内调用函数(通常不从程序其它部分调用主函数), 函数名放在括号内的变元表中且用逗号分开, 函数说明中变元的个数必须与参数相同。在上例中不存在函数调用。
函数完成由其函数体内语句所决定的任务, 也许将值返回调用的表达式。主函数体用花括号({}) 界定。这与 PL/I 中的 DO-END 或 Pascal 中的 BEGIN-END 类似。通常, 函数体含有一个或多个返回语句。一个返回语句规定了(如果有返回值的话)把什么返回给调用函数的表达式。依赖于函数的编写。可省略返回语句, 省略返回语句的函数返回值无定义。如果函数无返回值, 用户可将函数说明为具有 void 数据类型。有关函数的进一步信息, 参见第 2 章, 程序结构。
- ③ 本例中, 变量 total 在主函数中说明和定义。在程序内引用变量前必须先对其说明。说明以分号(;) 结束。在说明一个变量时, 要指定它的数据类型。数据类型指出了所需要的存储量与如何解释存储对象。例如, 变量 total 具有 int(整型)数据类型, 其对象需要 32 位存储空间(4 字节或 1 个长字), VAX C 语言把 int 类型变量解释为具有正或负号(或 0)的整数。

当定义一个变量时,需指明它的存储类,它的存储类影响它的位置,生命期与作用域。在函数内被说明为 int 类型的变量有一个 auto(自动)的缺省存储类。具有这种存储类类型的变量,当运行函数时获得存储区,且当控制返回到调用函数处时释放存储区。并非所有的存储类均都用缺省值说明。用户可指明 VAX C 语言的存储类并且在变量说明里将存储类关键字放在数据类型关键字前面或后面。

当确定一个变量的作用域时,数据类型与存储类是很重要的。有关数据类型的进一步信息,参见第 5 章,数据类型与说明。有关存储类的进一步信息,参见第 6 章,存储类与分配。

用于说明数据类型(如 int, double),存储类(如 auto, globalvalue),语句(如 if, goto),运算符(如 sizeof)的保留字称为关键字。关键字是预定义的标识符且不能再作说明。在用户程序中不能使用这些字来标识变量与函数。关键字必须用小写字母表示。有关 VAX C 语言关键字表,参见第 2 章,程序结构。

VAX C 语言是一种区分大小写字符的语言,说明变量(如 total)可用大写或小写字母混写。但如果在程序中引用变量 total,这种引用必须用小写字母。例如,如果企图以“Total”形式引用变量,则产生错误;编译程序由于该字开始使用大写字母而不识别这种变量。当引用 VAX C 语言所有标识符时,不必象上面 total 那样;语言区分大小写的所有例子在本手册中被标出来了。

- ④ 最后, 2+2 的和被存储在变量 total 中。这是通过使用一个有效的 VAX C 语句来实现的。任何以分号(;)结束的有效的表达式都可作为一个语句。标识符 total 是一种说明了的变量;等号(=)与加号(+)是有效的 VAX C 语言运算符;加法中的数字是有效的常数。有关各种 VAX C 语言语句的进一步信息,参见第 3 章,语句。有关 VAX C 运算符的进一步信息,参见第 4 章,表达式与运算符。

1.4 输入/输出

C 语言不包括处理输入与输出(I/O)的功能。然而,所有的语言工具必须具有程序与用户通信的方法。上例中缺少通信手段是很不方便的;使用户无法知道程序是否将正确的值 4 赋给变量 total。为了查看 total 变量中是否为 4,用户可使用 VAX C 语言运行时间库(RTL)函数将变量 total 值输出到终端。

所有 C 语言编译程序用时间库函数及宏指令完成输入,输出和与特定的操作环境有关的各种任务。VAX C 语言运行时间库提供了其他 C 语言包括的许多函数与 C 语言能实现的宏指令。另外,还存在在 VAX/VMS 系统环境下直接而有效地工作的函数。

VAX C 语言 RTL 函数当解决用户程序中的外部引用时,要“访问”目标码的程序段(运行时,为可共享的映像)。所有的 RTL 目标码模块装在 SYS\$LIBRARY:VAXCURSE.OLB, SYS\$LIBRARY:VAXCTRLG.OLB 与 SYS\$LIBRARY:VAXCRTL.OLB 库中。在用户执行本手册任何程序实例前,用户必须以恰当的顺序定义库,以便连接程序可查寻库,从而解决对这些 VAX C 语言 RTL 函数的引用。为了确定以哪种顺序定义这些库,参见第 15 章, VAX C 语言运行时间库信息。有关库的一般信息,参见第 9 章,对程序开发的介绍。

在程序源码中 VAX C 语言的 RTL 宏引用,看来就象函数引用一样。然而,编译程序在执行过程的早期阶段用 VAX C 语言源码来取代宏引用。编译程序将 VAX C 语言 RTL 宏指令源码放在 VAX C 语言提供的.H 的定义文件内。如果系统管理员在安装时对.H 文件进行挑选。可在 SYS\$LIBRARY 目录下访问所需文件,例如,在终端上可用如下命令显示 STDIO.H 文件:

```
$TYPE SYS$LIBRARY.STDIO.H RETURN
```

如果此命令有错, 向系统管理员了解在安装期间对.H 文件是如何摘选的。显示或打印所有的.H 文件, 看看 VAX C 语言提供的宏与定义是个好办法。

用户可将.H 定义文件放在文本库, VAX C DEF.TLB 中, 此文本库装在 SYS\$LIBRARY 目录下, 本手册把.H 文件看成定义模块, 因此它们可象这个文本库中的模块一样访问。

有关宏指令的进一步信息, 参见第 7 章, 预处理器控制行。有关访问 VAX C 语言 RTL 函数的各种方法的更多信息, 参见第 15 章, VAX C 语言运行时间库信息。

例 1-2 表明一个 VAX C 语言程序, 通过使用 VAX C 语言 RTL 的 printf 函数可在终端上显示信息。

例 1-2: 信息的输出

```
/* This program adds two numbers, assigns the value 4 to a
 * variable total, and then prints the answer on the
 * terminal screen. */

main( )
{
    int total;
    total = 2 + 2;

    ① printf("Here is the answer: ");
    printf("%d", total); /* Print the answer */
}
```

注解:

- ① VAX C 语言的 RTL 函数 printf 将信息写到标准输出设备上(终端显示屏幕)。第一次对 RTL printf 函数调用将一个字符串作为变元传递输出。第二次对 printf 调用将有特定格式的字符串与一个变量传递输出。在格式串内, 百分号(%)由 total 值取代, 减号(-)调整左对齐输出, 字母 d 迫使变元值以十进制数表示。句号(.)立即在 total 值后输出。

此程序输出形式如下:

Here is the answer:4

如果用户予将 total 值在另一行上显示, 那么串中必须有换行符。例 1-3 说明如何将信息输出到两行上。

例 1-3: 使用换行符输出的程序

```
/* This program adds two numbers, stores the sum in the
 * variable total, and then prints the answer on two
 * separate lines on the terminal screen. */

main( )
{
    int total;
    total = 2 + 2;

    printf("Here is the answer...\n"); /* Print intro string */
    printf("%d", total); /* Print the answer */
}
```

此程序输出如下:

Here is the answer...

4.

现在一个带有输出的程序已编完, 然后它必须用 DIGITAL 命令语言(DCL)进行编

译, 连接和运行, 最后看结果。编译一个程序就是将源码转换成目标码; 连接一个程序就是组织存贮区且解决外部引用(例如, 对 VAX C 语言 RTL 函数的引用); 运行一个程序就是执行映象。

在 VAX/VMS 系统环境下, 一个文件由文件名与文件扩展名来识别。用户可选自己比较容易识别的文件名。文件扩展的选择会影响文件的功能。例如, 文件名为 ADDITION.C 是 VAX C 语言源程序的一个文件名。文件扩展名.C 是 VAX C 语言编译程序缺省的文件扩展名。如果在编译 C 语言时写文件名 ADDITION, 则编译程序将寻找文件 ADDITON.C。

一旦用户程序已建立了且具有恰当的名字, 则程序可按如下方式编译, 连接且运行:

```
$ DEFINE LNK$LIBRARY SYS$LIBRARY:VAXRTL.OLB RETURN
$ CC ADDITION.C RETURN
$ LINK ADDITION.OBJ RETURN
$ RUN ADDITION.EXE RETURN
Here is the answer...
4.
```

.OBJ 与 .EXE 扩展分别指目标文件与映象文件的缺省文件扩展名。

用户为了在程序中使用 VAX C 语言 RTL 函数, 可为连接程序定义更多的库。上面的定义满足了执行本章内的所有例程。一旦用户定义了库, 在其余的终端对话期间不必再定义它们(直到退出为止)。有关对连接程序库规范的进一步信息, 参见第 9 章, 对程序开发的介绍。有关源码的建立的进一步信息, 参见第 10 章, 建立源程序。有关编译过程的进一步信息, 参见第 11 章, 编译, 连接与运行 VAX C 语言程序。

1.5 控制程序流

有些场合, 用户在已知某种条件下必须执行 VAX C 语言的一条或多条语句, 而在另一些场合, 用户必须重复地在循环体内执行一条或多条 VAX C 语言语句, 直到遇到某种条件为止。在 VAX C 语言中, 有几种语句用于完成这些任务。这些语句有 if 语句, switch 语句, do 语句与 for 语句。有关 while 语句(直到满足条件才停止的另一种循环语句)的信息, 参见第 3 章, 语句。

1.5.1 if 语句

在给定的条件下执行一条或多条 VAX C 语言语句时, 用户可使用 if 语句。例 1-4 表示了使用 if 语句的一个程序。

例 1-4: 使用 if 条件语句的程序例

```
/* This program asks the user to guess a letter. The
 * program tells whether the answer's correct or
 * incorrect. The program's hard coded to accept 'a' or
 * 'A' as the correct letter. */

main( )
{
```

```

char ch;                                /* Declare a character */
/* Ask the user to guess */
printf("Guess which letter I'm thinking of!\n");

① ch = getchar( );                      /* Get the character */

/* Correct = "a" or "A" */
② if (ch == 'a' || ch == 'A')           /* If correct guess */
    printf("You're right!");
else                                     /* If incorrect guess */
{
    printf("You're wrong.\n");
    printf("You'll have to try again!");
}

```

注解:

- ① VAX C语言RTL的getchar函数,从标准输出设备(终端)上取一个字符;程序暂停,等候用户键入一个字符后再按 RETURN 键。函数 getchar 只取第一个字符,而不理采其它键入的字符。
- ② 如果用户键入的字符是'a'或'A',那么输出信息指出选择是正确的。如果键入其它字符,那么输出信息指出选择是不正确的。等号(==)将变量ch与常量'a'与'A'比较。逻辑或运算符(||)表示测试条件。如果在测试条件下执行多个语句,那么必须用花括号({})将语句括起来。用花括号括起来的语句称为一个块或复合语句。块的概念对确定变量作用域很重要的。有关块的进一步信息,参见第2章,程序结构。

本程序的一个输出例如下:

```

$ RUN EXAMPLE4 RETURN
Guess which letter I'm thinking of!
B RETURN
You're wrong.
You'll have to try again!

```

1.5.2 开关(switch)语句

if 语句不是唯一的使指定的语句在给定的某种条件下执行的唯一方法。switch 语句可完成上例中 if 语句所完成的同样任务,但当需要测试许多条件时 switch 语句特别有用。switch 语句例如下:

例 1-5:使用 switch 语句有条件执行的程序例

```

/* This program plays the same guessing game as the
 * previous example except that it uses a switch
 * statement. */

① #include ctype                      /* Include proper module */

main( )
{
    char ch;

    printf("Guess what letter I'm thinking of!\n");
    ch = getchar( );
    ② _tolower(ch);                    /* Convert "ch": lowercase */
    switch(ch)                         /* Examine "ch" */
    {                                  /* Body of switch statement */

        case 'a':
            printf("You're right!");
            return;

        default:                       /* Any other answer */
            printf("You're wrong.\n");
    }
}

```