信息科学与技术丛书 移动与嵌入式开发系列

吴士力 刘奇 朱兰 编著

嵌入式Linux应用开发全程解析与实战

- 基于友善之臂 Mini2440 开发板与红旗 Linux 6.0
- Linux 2.6 设备驱动程序开发
- 嵌入式 MiniGUI 开发
- 嵌入式数据库 SQLite3 开发
- 嵌入式 Linux 开发环境的搭建
- 电子菜单的设计与实现

机械工业出版社
CHINA MACHINE PRESS

信息科学与技术丛书·移动与嵌入式开发系列

嵌入式 Linux 应用开发全程解析与实战

吴士力　刘奇　朱兰　编著

机械工业出版社

本书详细介绍了在红旗 Linux 6.0 平台上开发嵌入式 Linux 应用程序的基本原理和过程。主要内容包括嵌入式软硬件平台的介绍、Linux 2.6 内核的移植、Linux 驱动程序的原理、Bootloader 的原理、嵌入式 Linux 开发环境的搭建、嵌入式数据库 SQLite3 和嵌入式 GUI 系统 MiniGUI 的移植和编程等。最后通过电子菜单实例系统介绍了在 Mini2440 开发板上开发基于 Linux 2.6.29 内核的 GUI 应用程序的全过程。书中使用的嵌入式 Linux 应用开发软件平台均为开源软件，具有较高的市场占有率。

本书适合嵌入式 Linux 应用开发的初学者，或计算机、电子专业的大中专高年级学生和本科生。本书也可作为嵌入式 Linux 应用开发技术人员的参考书。

图书在版编目（CIP）数据

嵌入式 Linux 应用开发全程解析与实战 / 吴士力，刘奇，朱兰编著. —北京：机械工业出版社，2009.11
（信息科学与技术丛书 • 移动与嵌入式开发系列）
ISBN 978-7-111-28631-8

Ⅰ. 嵌… Ⅱ. ①吴…②刘…③朱… Ⅲ. Linux 操作系统－程序设计 Ⅳ. TP316.89

中国版本图书馆 CIP 数据核字（2009）第 196298 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）
策划编辑：车 忱
责任编辑：车 忱
责任印制：洪汉军

三河市国英印务有限公司印刷

2010 年 1 月第 1 版 • 第 1 次印刷
184mm×260mm • 21.75 印张 • 538 千字
0001—4000 册
标准书号：ISBN 978-7-111-28631-8
ISBN 978-7-89451-289-5（光盘）
定价：45.00 元（含 1DVD）

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务
社服务中心：（010）88361066
销 售 一 部：（010）68326294
销 售 二 部：（010）88379649
读者服务部：（010）68323821

网络服务
门户网：http://www.cmpbook.com
教材网：http://www.cmpedu.com

封面无防伪标均为盗版

出 版 说 明

随着信息科学与技术的迅速发展，人类每时每刻都会面对层出不穷的新技术、新概念。毫无疑问，在节奏越来越快的工作和生活中，人们需要通过阅读和学习大量信息丰富、具备实践指导意义的图书，来获取新知识和新技能，从而不断提高自身素质，紧跟信息化时代发展的步伐。

众所周知，在计算机硬件方面，高性价比的解决方案和新技术的应用一直备受青睐；在软件技术方面，随着计算机软件的规模和复杂性与日俱增，软件技术受到不断挑战，人们一直在为寻求更先进的软件技术而奋斗不止。目前，计算机在社会生活中日益普及，随着因特网延伸到人类世界的层层面面，掌握计算机网络技术和理论已成为大众的文化需求。由于信息科学与技术在电工、电子、通信、工业控制、智能建筑、工业产品设计与制造等专业领域中已经得到充分、广泛的应用，所以这些专业领域中的研究人员和工程技术人员越来越迫切需要汲取自身领域信息化所带来的新理念和新方法。

针对人们对了解和掌握新知识、新技能的热切期待，以及由此促成的人们对语言简洁、内容充实、融合实践经验的图书迫切需要的现状，机械工业出版社适时推出了“信息科学与技术丛书”。这套丛书涉及计算机软件、硬件、网络、工程应用等内容，注重理论与实践相结合，内容实用，层次分明，语言流畅，是信息科学与技术领域专业人员不可或缺的图书。

现今，信息科学与技术的发展可谓一日千里，机械工业出版社欢迎从事信息技术方面工作的科研人员、工程技术人员积极参与我们的工作，为推进我国的信息化建设作出贡献。

机械工业出版社

前　言

自从第一台电子计算机诞生以来，人们的生活方式发生了巨大的改变。如今，计算机已经渗入到各行各业中。计算机之所以能够获得广泛应用，原因就在于计算机可以模拟世间万物的各种变化过程，并计算出各种变化过程的结果。随着计算机运算速度的提高，人们可以通过计算机来进行绘制 DNA 图、天气预报、模拟核爆炸、人工智能求解或证明数学定理等高度复杂的计算过程。计算机科学的主要研究方向就是如何让计算机以更快的速度解决更复杂的计算问题。

说到这里，大家不要以为计算机的任务只是解决复杂的计算问题。因为在我们的生活中还大量存在着很多相对简单的计算。例如车床控制、数字电视信号处理、实时监控等，它们的计算量不是很大，对计算速度也没有很高的要求。于是，出于实用性、易用性、成本等因素的考虑，人们在传统计算机的基础上发明了针对专业用途的计算机系统，称为嵌入式计算系统。目前常见的手机、智能家电、MP3 播放器、远程网络监控、电子导航仪等都属于嵌入式系统的范畴。

在专业领域，嵌入式系统具有比传统通用计算机更高的可靠性、实用性和更低的成本。所以，嵌入式系统正在以比通用计算机更高的发展速度走进人们的生活中。如今，可以说每个人一般都最少拥有十几个嵌入式产品，例如家电、数码相机、MP3、手机等。在我国，嵌入式产业的大发展才刚刚开始，属于真正的朝阳产业。嵌入式开发技术人员的薪酬在 IT 行业中也处于较高的水平。本书正是为准备进入或刚刚进入嵌入式开发领域的读者编写的。

目前在嵌入式技术领域，还没有哪种软硬件技术平台具有绝对的垄断地位。在硬件方面，ARM 处理器以强大的功能和突出的性能占据了 70%~80%的嵌入式处理器市场。在软件方面，嵌入式 Linux 操作系统正在快速发展，目前已占据了 30%左右的市场。由于 Linux 内核的开源性和开放式开发方式，在全世界拥有大量的 Linux 系统编程技术人员，这为嵌入式 Linux 系统的发展提供了坚实的基础和平台。

本书以基于 ARM9 处理器和嵌入式 Linux 的嵌入式开发技术为介绍对象。其中，嵌入式处理器采用的是基于 ARM9 的 S3C2440 处理器，开发板平台是广州友善之臂公司生产的 Mini2440。Mini2440 具有稳定的性能和丰富的接口，使用方便，可操作性强，是理想的嵌入式系统开发平台。软件平台采用了 Linux 2.6.29 内核、BusyBox、SQLite3、MiniGUI、红旗 Linux 6.0、Eclipse 等，涵盖了嵌入式操作系统、嵌入式文件系统、嵌入式数据库、嵌入式 GUI、宿主机系统平台和嵌入式应用程序开发环境等方面。

本书以 Linux 2.6 内核为核心，详细讲解了基于 ARM 和 Linux 的嵌入式应用开发技术原理和开发方法，使读者能够掌握嵌入式 Linux 应用开发技术的基本原理和一般方法，并初步具备在各种嵌入式平台上进行 Linux 应用开发的能力。

本书的配套光盘提供了书中第 8、10、11、13 章的示例程序源代码以及相关的嵌入式 Linux 应用开发软件。

本书在编写过程中，参考了大量的开源技术资料。在此向这些为开源技术做出贡献的公司和各界人士表示衷心的感谢！

由于编者水平有限、时间仓促，书中难免有疏漏和不足之处，恳请广大读者批评指正。

目　录

出版说明
前言
第 1 章　C 语言编程技术要点 ……1
1.1　数据类型 ……1
1.1.1　C 语言基本数据类型的实现 ……1
1.1.2　C 语言基本数据类型的转换 ……3
1.1.3　C 语言复合数据类型的实现 ……4
1.1.4　C 语言的数据管理 ……9
1.2　函数 …… 12
1.3　C 语言预处理 …… 13
1.3.1　宏定义 …… 13
1.3.2　条件编译 …… 14
1.3.3　头文件 …… 15
1.4　C 语言编程规范 …… 17
1.4.1　命名规则 …… 17
1.4.2　程序版式 …… 17
第 2 章　计算机硬件平台技术要点 …… 19
2.1　CPU 的结构 …… 19
2.2　CPU 指令 …… 20
2.2.1　CPU 指令格式 …… 21
2.2.2　寻址 …… 22
2.2.3　x86 CPU 指令系统 …… 24
2.3　实模式和保护模式 …… 26
第 3 章　Linux 操作系统原理与使用 … 28
3.1　操作系统原理概述 …… 28
3.1.1　进程管理 …… 29
3.1.2　内存管理 …… 31
3.1.3　文件管理 …… 37
3.1.4　设备管理 …… 39
3.2　Linux 内核 …… 40
3.2.1　Linux 内核结构 …… 41
3.2.2　Linux 进程管理 …… 43
3.2.3　内存管理 …… 46
3.2.4　文件管理 …… 48
3.2.5　设备管理 …… 51
3.3　Linux 操作系统的使用 …… 52
3.3.1　安装红旗 Linux 6.0 …… 52
3.3.2　红旗 Linux 6.0 的使用 …… 56
第 4 章　Linux 应用程序开发 …… 70
4.1　GCC …… 70
4.1.1　GCC 工作流程 …… 71
4.1.2　Glibc …… 74
4.2　工程管理器 …… 76
4.2.1　Makefile …… 77
4.2.2　Makefile 特性介绍 …… 79
4.3　Makefile 的自动生成 …… 89
4.4　GDB 调试器 ……103
4.5　Eclipse 程序开发 ……107
4.5.1　Eclipse 环境安装 ……107
4.5.2　Eclipse C 程序开发 ……109
第 5 章　嵌入式系统基础 ……116
5.1　嵌入式系统概述 ……116
5.1.1　嵌入式系统简介 ……116
5.1.2　嵌入式系统的结构 ……117
5.2　嵌入式处理器 ……117
5.2.1　ARM 处理器介绍 ……117
5.2.2　ARM 处理器指令 ……121
5.2.3　S3C2410/ S3C2440 介绍 ……125
5.3　嵌入式操作系统 ……132
第 6 章　嵌入式 Linux C 应用开发 ……134
6.1　嵌入式 Linux 开发环境 ……134
6.1.1　交叉编译器 ……134
6.1.2　交叉编译器的使用 ……135
6.2　Eclipse 交叉编译 ……145
第 7 章　嵌入式 Linux 系统开发 ……148
7.1　嵌入式 Linux 内核 ……148
7.1.1　内核的定制原理 ……148
7.1.2　内核的配置 ……162
7.1.3　编译内核 ……166

7.1.4 内核启动过程 ……168
7.2 根文件系统 ……174
7.2.1 根文件系统的结构 ……174
7.2.2 init 进程与配置文件 ……176
7.2.3 BusyBox ……180
7.2.4 文件系统类型 ……185
7.3 SkyEye 模拟运行嵌入式 Linux 内核 ……190
7.3.1 SkyEye 介绍 ……190
7.3.2 SkyEye 运行嵌入式 Linux 系统的方法 ……194
第 8 章 Linux 设备驱动程序开发 ……200
8.1 Linux 模块 ……200
8.1.1 Linux 模块介绍 ……200
8.1.2 Linux 模块编程 ……201
8.2 设备驱动程序原理 ……203
8.2.1 字符设备驱动概述 ……205
8.2.2 并发控制 ……209
8.2.3 同步控制 ……215
8.2.4 中断处理 ……219
第 9 章 Bootloader ……222
9.1 Bootloader 的工作原理 ……222
9.1.1 Bootloader 概述 ……222
9.1.2 Bootloader 的工作过程分析 ……223
9.2 ViVi ……224
9.2.1 ViVi 的源代码结构 ……225
9.2.2 ViVi 的启动过程 ……225
9.2.3 ViVi 的基本命令 ……231
9.3 U-Boot ……233
9.3.1 U-Boot 的源代码结构 ……234
9.3.2 U-Boot 的启动过程 ……234
9.3.3 U-Boot 命令 ……236
第 10 章 嵌入式 GUI 开发 ……240
10.1 MiniGUI 系统介绍 ……240
10.1.1 嵌入式 GUI 系统介绍 ……240
10.1.2 MiniGUI 介绍 ……241
10.2 MiniGUI 使用介绍 ……242
10.2.1 MiniGUI 的安装与使用 ……242
10.2.2 MiniGUI 的定制 ……244
10.3 MiniGUI 编程 ……251
10.3.1 MiniGUI 的窗口和消息 ……251
10.3.2 MiniGUI 实例分析 ……255
第 11 章 嵌入式数据库 ……261
11.1 嵌入式数据库概述 ……261
11.2 SQLite3 的使用 ……262
11.2.1 SQLite3 的命令 ……262
11.2.2 SQLite3 的 C 接口 ……264
第 12 章 嵌入式 Linux 开发环境的搭建 ……271
12.1 Minicom 的使用 ……271
12.2 Bootloader 的烧写 ……273
12.3 使用 Supervivi 搭建开发环境 ……275
12.3.1 Supervivi 使用介绍 ……275
12.3.2 分区 ……277
12.3.3 烧写镜像 ……280
12.3.4 调试内核 ……282
12.4 NFS ……284
12.5 Tftp ……286
第 13 章 电子菜单的设计与实现 ……288
13.1 电子菜单介绍 ……288
13.2 电子菜单程序的设计与实现 ……290
13.2.1 源文件结构 ……290
13.2.2 源代码分析 ……290
13.2.3 Makefile 设计 ……314
13.3 电子菜单结构的设计与实现 ……316
13.3.1 Bootloader 镜像 ……316
13.3.2 内核镜像 ……318
13.3.3 文件系统设计 ……326
13.3.4 文件系统镜像的设计与制作 ……337
参考文献 ……341

第1章 C语言编程技术要点

在自然界中不论多复杂的问题都是由两部分组成的，一部分是问题处理的对象，另一部分是处理问题的具体方法。例如用布匹做衣服的问题，衣服就是问题的对象，具体的剪裁工艺就是做衣服的方法。由于现实生活中的许多问题需要计算，所以人们发明了计算工具来帮助处理问题。例如古老的沙漏、算盘等。但直到电子计算机的出现，人们才真正从繁重的计算任务中解脱出来。

电子计算机之所以具有强大的计算能力，除了运算速度快，根本原因在于计算机具有自动运行程序的能力。因此，计算机能否正确、高效地处理问题取决于程序能否客观、正确地描述问题。而程序要把问题客观、正确地描述清楚，最基本的要求是使用具有一套正确、合理的语法机制的编程语言。这也就说明,学习编程语言的内容之一就是学习其语法规则和运行机制。

在众多的编程语言中，C 语言是一门历史悠久但生命力很强的高级语言。据最新的调查数据显示，目前 C 语言的使用率依旧保持在 30%以上。C 语言之所以能够长盛不衰，主要原因有以下几点：

（1）C 语言具有出色的可移植性，能在多种体系结构的软硬件平台上运行。

（2）C 语言具有简洁紧凑、使用灵活的语法机制，并能直接访问硬件。

（3）C 语言具有很高的运行效率。

鉴于以上原因，很多操作系统的内核、系统软件等都是使用 C 语言编写的。在嵌入式 Linux 开发领域，C 语言同样是使用最广泛的语言之一。下面从语法角度介绍 C 语言编程的基本知识。

1.1 数据类型

1.1.1 C 语言基本数据类型的实现

在 C 语言中，描述问题对象的基本语法单元是数据类型。然而，自然界中的数据类型是无穷无尽的。那么如何用“有限”的语言来描述“无限”的数据类型呢？最简单也是最有效的方法就是创建一些基本的简单数据类型，然后通过这些基本数据类型之间的组合来描述其他复杂的数据类型。例如，整数通过正负号、0 和自然数的组合来表示；复数通过两个实数的组合来表示；每个人的基本信息通过姓名、性别、年龄、籍贯、政治面貌等数据的组合来表示。

对于编程语言来说，基本数据类型的设计是非常有讲究的。如果基本数据类型过少，就会加大数据类型的实现复杂度，进而加重编程的负担。但如果基本数据类型过多，又会增加

语法复杂度，进而影响程序的运行效率。

为了保证可移植性，基本数据类型的实现方式必须考虑到计算机软硬件系统的体系结构。因此在不同的平台中，C 语言基本数据类型的实现方式也是有所不同的（本章以 IBM 兼容机和 GCC 平台为准）。这一点在嵌入式系统开发中需要特别注意。为了提高程序的运行效率，基本数据类型的最大长度单位一般和 CPU 的字长相同。

在形形色色的数据中，数字是一种最基本的数据。早在两千多年前，古希腊著名的毕达哥拉斯学派就提出了“万物皆是数”的观点。所以几乎所有的编程语言都定义了用于描述数的基本数据类型。在 C 语言中，数的基本数据类型有整型和浮点型两大类。

1．整型

整型是用于描述整数的基本数据类型。C 语言把整型分为基本型、短整型、长整型和无符号型四种。四种整型的描述范围见表 1-1。

表 1-1 整型数据表

类型	标识符	范围	长度
基本型	int	-2147483648~2147483647	32
短整型	short	-32768~32767	16
长整型	long	-2147483648~2147483647	32
无符号型	unsigned	0~4294967295	32

从表中可以看到，不同类型的数据占用的内存空间是不同的，所以描述的数值范围也是不同的。

在编程语言中，整型数据除了十进制外还常用八进制、十六进制来表示。为了便于区别，八进制数以 0 开头，十进制数不能以 0 开头，十六进制数以 0x 开头。例如整数 10 可以表示为：

012（八进制）

10（十进制）

0xA（十六进制）

2．浮点型

浮点数是指小数点在逻辑上不固定的数，一般表示成 $m \times b^e$。其中，b 表示基数；m 表示尾数，显示 m 中每个数的值都在 0 和 $b-1$ 之间，e 表示指数。例如十进制数 0.562 可以表示成 5.62×10^{-1}，98 000 可以表示成 9.8×10^4。为了记述方便，0.562 还可以表示成 5.62E-1，98 000 可以表示成 9.8E4。

在 C 语言中，浮点数是通过浮点型来描述的。浮点型分为单精度型和双精度型两种，其描述范围见表 1-2。

表 1-2 浮点型数据表

类型	标识符	范围	有效数字	长度
单精度型	float	E-38~E38	7	32
双精度型	double	E-308~E308	15~16	64

和整型不同的是，浮点型还存在有效数字的限制，以此来控制小数的精度。例如：

```
float a;
a = 1.2345678;
printf("%f", a);
```

结果为：

1.234568，只有 7 位有效数字。

3．字符型

在自然界中，有很多问题是无法只使用数字就能直观描述的。为此，C 语言定义了用于描述字符数据的字符类型。由于计算机本身只能处理数字，所以要让计算机能够识别字符就必须先对字符进行编码，也就是在字符与数字之间建立一种对应关系。在 20 世纪 60 年代，美国制定了标准信息交换码（ASCII 码），这是目前应用最广泛的一套编码规则。如字母 a 的 ASCII 码是 97，字母 A 的 ASCII 码是 65 等。由于常用的基本字符不会超过 128 个（常见的 PC 标准键盘上大约有 100 个按键），所以在 ASCII 码中，每个字符使用 7 个二进制位来表示。但为了提高运行效率，C 语言（其实是 C 语言的编译系统，但为了方便，全部统称为 C 语言）会为每个字符分配一个字节的空间。对于剩下的一个二进制位，一般会用于校验等用途。

在 C 语言中，一个字符型数据使用一对单引号来标注。如'1'就是一个字符型数据，而不加单引号的 1 就是整型数据。字符型数据的标识符是 char，每个字符型数据会占用一个字节的空间。由于字符型数据是以 ASCII 码的形式存在的，所以一个字符型数据既能以字符形式表示，也能以整数形式表示。例如：

```
char x,y;
x = 97;
y = 65;
printf("x = %c, y = %c\n", x, y);
x ='a';
y ='A';
printf("x = %d, y = %d\n", x, y);
```

结果为：

```
x = a, y = A
x = 97, y = 65
```

显然，单个字符的描述能力也是非常有限的。如果把多个字符组合在一起形成一个字符串，这样可以大大丰富描述的对象。

在 C 语言中，字符串使用一对双引号来标注，如"hello world"。需要注意的是，字符'a'和字符串"a"虽然内容相同，但存储的方式却是不同的。C 语言在存储字符串时，会自动在字符串的末尾加上一个字符'\0'，以此来标识一个字符串的末尾。所以，字符串"a"实际上会占用 2B 的空间，而字符'a'只占用 1B 的空间。

1.1.2　C 语言基本数据类型的转换

在实际的计算过程中，往往需要把不同类型的数据放在一起运算，如“3 + 5.5/2”。在对数据进行混合运算时，编程语言必须把不同类型的数据转换成同一种类型的数据。C 语言的

转换规则如下所示：

```
字符型→整型（短整型→基本型→无符号型→长整型）→浮点型（单精度型→双精度型）
```

以上规则说明了数据类型可以由低精度向高精度转换，并且这个转换过程是由 C 语言自动完成的。例如整型和单精度型相加，C 语言会自动把整型转换成单精度型，再把两个单精度型的数据相加。

如果必须把高精度型数据转换为低精度型数据，则可以使用强制类型转换。在 C 语言中，强制类型转换的定义方式为：

```
（类型）数据表达式
```

例如“（float）a”表示把 a 转换成单精度型。但高精度向低精度的强制类型转换会造成有效数字的丢失。例如：

```
int a;
float b;
a = 1;
b = 1.8;
b = a + b;
printf("%f\n", b);
b = 1.8
a = (int) b;
printf("%d\n", a);
```

结果为：

```
2.800000
1
```

可以看到，把浮点型变量 b 强制转换成整型后，丢失了有效数字 8。在有些情况下，这种数据丢失会严重影响计算的精度。

1.1.3 C 语言复合数据类型的实现

C 语言在整型、浮点型、字符型等基本数据类型的基础上定义了复合数据类型。基本方法就是按照一定的逻辑关系对基本数据类型进行组合，以此来形成复杂数据类型。在 C 语言中，常用的复杂数据类型有数组、结构体和共用体等。

1. 数组

在 19 世纪末，德国数学家康托发明了集合论，为现代科学奠定了坚实的理论基础。

集合就是把具有相同性质的对象集中起来组成的整体。例如水果类是所有水果的集合；班级是班中同学的集合；太阳系是太阳和九大行星的集合等。集合的结构虽然简单，但应用极其广泛。

在 C 语言中，描述集合的数据类型称为数组。根据元素之间的逻辑层次关系可以把数组分为一维数组和多维数组。

（1）一维数组。一维数组的定义方式为：

```
类型 数组名 [数组长度]
```

例如“int a[10]”表示定义了一个有 10 个整型元素的数组，数组名为 a。

C 语言规定，数组的下标默认从 0 开始，以便内存寻址。在定义数组的时候必须给出数组元素的个数，而且不允许动态改变。所以在定义数组时必须考虑实际的应用需求。数组过大，会造成内存浪费，过小又会造成越界。

在为一维数组分配内存空间时，数组中的每个元素根据下标顺序排列。“int a[6]”在内存中的分布如图 1-1 所示。

只要给出数组第一个元素的地址，C 语言就可以根据数组下标随机访问任意一个数组元素。所以，在 C 程序中可以把存储第一个数组元素地址的数组名作为参数进行传递。

（2）二维数组。二维数组的定义方式为：

```
类型 数组名 [数组长度] [数组长度]
```

例如“char a[8][10]”表示定义了一个有 8×10 个字符型元素的二维数组。和一维数组一样，定义二维数组的时候也必须给出数组元素的个数。“int a[2][3]”在内存中的分布如图 1-2 所示。

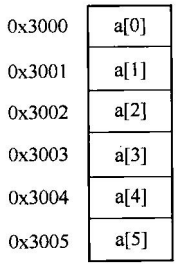

图 1-1　一维数组 a 分布图

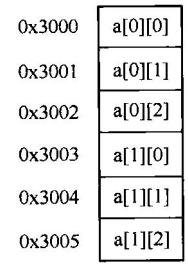

图 1-2　二维数组 a 分布图

可以看到，二维数组在内存中是以一维数组的方式进行存储的。依此类推，很容易得到三维、四维等多维数组的存储方式。由于 C 语言没有直接提供字符串类型数据，所以字符串是以字符型数组的方式进行存储的。

2．结构体

在数组中，所有的元素都具有相同的数据类型。而在结构体中，允许元素具有不同的数据类型。在 C 语言中，结构体的定义方式为：

```
struct 结构体名
{
    类型 成员名;
}
```

结构体可以描述的对象显然要比数组丰富得多。例如学生的信息可以由字符型的姓名、整型的年龄和浮点型的身高组成。这里以学生信息为例，定义结构体 student。

例如：

```
struct student
{
    char name[5];
    int age;
    float height;
}
```

在 C 语言中，结构体可以嵌套定义，也就是说结构体中的成员还可以是结构体。在实现方式上，C 语言会依次为结构体中的每个成员分配各自的内存空间，结构体 student 在内存中的分布如图 1-3 所示。

有一些编程语言为了提高 CPU 内存寻址和取值的速度，会对结构体中成员的存储地址进行对齐处理。对齐处理是根据对齐模数来实现的。如果结构体中某个基本数据类型成员占用的内存空间大于其他基本数据类型成员占用的内存空间，那么就称这个成员的空间大小为对齐模数。例如结构体 student 中，成员 name 是字符型数组，每个数组元素的空间大小是 1B；成员 age 的内存空间是 2B；成员 height 的内存空间是 4B。所以 4 就是结构体 student 的对齐模数。

对齐的基本思想就是结构体中每个成员所占的内存大小必须是对齐模数的整数倍，如某个成员的空间小于对齐模数的整数倍，则由编译系统自动填充剩下的空间。结构体 student 对齐后在内存中的分布如图 1-4 所示。

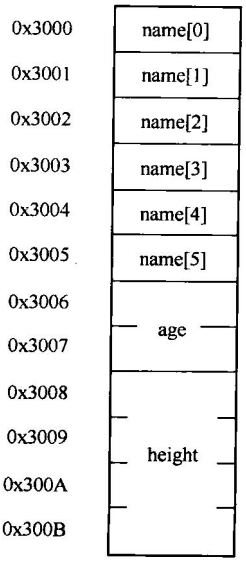

图 1-3 结构体 student 分布图

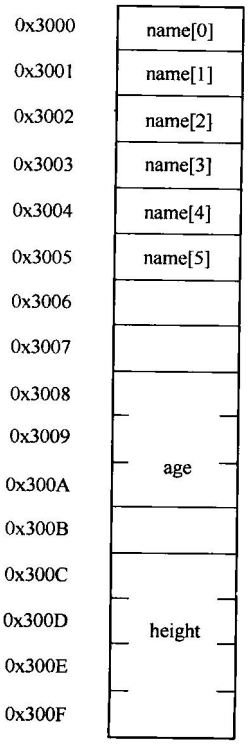

图 1-4 结构体 student 对齐分布图

结构体中成员的访问格式为：

```
结构体名.成员名
```

例如访问结构体 student 中成员 age 的格式为 student.age。

3．共用体

共用体在逻辑结构上和结构体是相同的。区别在于结构体所占内存的空间是所有成员空间的总和，而共用体把所有成员重叠在同一片内存空间中，使得共用体所占的内存空间等于某个所占空间最大的成员的空间。例如：

```
union student
{
    char name[5];
    int age;
    float height;
}
```

其中所占空间最大的成员是“char name[5]”，共占用 5B 的内存，所以共用体 student 的内存空间就是 5B。

4．指针

指针是 C 语言特有的一种基本数据类型，它使 C 语言具备了直接访问硬件的能力。因为指针的值是某个具体的内存地址，所以指针可以看成常量。但为了增加指针的使用灵活度，C 语言允许使用变量来存放指针值。用于存放指针值的变量称为指针变量。

在 C 语言中，通过指针的地址值可以直接访问存放在对应地址中的数据，这个过程称为指针指向数据。

指针的定义方式为：

```
类型 *变量名
```

其中符号“*”表示定义的变量为指针变量，“*变量名”表示指针变量所指向的数据，“类型”为指针指向的数据的类型。例如“int *p”表示 p 是一个指针变量，*p 表示一个指针 p 所指向的整型量。C 语言规定，一个指针变量只能指向一种数据类型的数据，不能指向其他类型的数据。

由于指针变量值是一个地址，所以为指针赋值的方式和整型、字符型有所不同，下面是两种常用的指针赋值方法。

```
int a;
int *p = &a;
```

或

```
int a;
int *p;
ptr = &a;
```

指针变量本身也是一种数据类型，所以也可以参与运算。指针变量的运算方式有以下几种：

（1）运算符。取址符&：通过取址符&可以得到指针的地址值。取值符*：通过取值符*可以得到指针指向的变量值。

（2）加减运算。为指针加上或减去一个整数值，表示把指针从当前指向的位置向前或后移动一个整数值的位置。

（3）指针间的相减运算。两个指针相减得到的差表示指针之间相隔的元素个数。

注意，指针的加减运算以及指针间的相减运算只对数组有意义，因为数组中的元素是顺序存储的。例如：

```
int x;
scanf("%d\n", &x);
printf("输入的是：%d\n", x);

int *p;
int *q;
int a[5] = {1,2,3,4,5};
p = q = a;
printf("%f\n", &p);
printf("%d\n", *p);

p++;
printf("%d\n", *p);
p--;
printf("%d\n", *p);

int i;
for(i=0; i<3; i++)
{
   q++;
};
printf("%d\n", q-p);
}
```

结果为：

```
100
输入的是：100
1
2
1
3
```

通过指针可以直接访问存储器中的各种数据，所以在对程序性能和灵活性有较高要求的嵌入式领域，指针的应用非常广泛。

1.1.4　C 语言的数据管理

在程序的运行过程中，往往会产生大量的数据，每个数据所起的作用是不一样的。为了提高程序的运行效率，必须对每个数据进行统一的安排和管理。就像要提高一支部队的战斗力，必须根据每个士兵的作战能力进行统一的安排和管理。

在 C 语言中，分别从数据的值和数据的作用范围两个角度去考察数据在程序运行时的作用。

1．常量和变量

在自然界中，有些数据的值会不断地发生变化，但有些数据的值始终不会发生变化或在一个阶段内不发生变化。例如，每天的气温都是变化的；在常压下水沸腾的温度恒定在 100℃等。在程序的运行过程中，值不会发生变化的数据称为常量，值会发生变化的数据称为变量。

2．全局变量和局部变量

在程序的运行过程中，有些变量会被多个函数所使用，这样的变量称为全局变量。有些变量只会被某个函数所使用，这样的变量称为局部变量。在 C 语言中，局部变量定义在函数体内，全局变量定义在函数体外。

根据全局变量和局部变量的特点，C 语言严格定义了这两种变量的作用域（作用范围）规则。

（1）全局变量。全局变量在被定义后的整个程序源文件范围内有效。也就是说，在全局变量之后定义的所有函数都可以直接访问该全局变量。

（2）局部变量。局部变量只能在被定义的函数内有效。一旦函数运行结束后，局部变量占用的内存空间就会被编译系统释放。例如：

```
int x = 1;   /* x 为全局变量 */
void add(void)
{
   extern y;
   int c;        /* c 为局部变量 */
   c = x + y;
   printf("%d", c);
}

int y = 1;   /* y 为全局变量 */

void p(void)
{
   printf("%d", c);
}
```

结果为：

```
2     /* 函数 add 的执行结果 */
In function 'p':
error: 'c' undeclared (first use in this function)
```

（3）外部变量。除了全局和局部变量，还有一种是程序中多个文件都需要使用的变量，称为外部变量。外部变量通过关键字 extern 进行标识。外部变量可以避免在多个文件中重复定义同一个变量。例如：

a.c：

```
int e = 98;
int main()
{
    b();
    return 0;
}
```

b.c：

```
#inlude <stdio.h>
extern int e;
void b()
{
    printf( "%d" , e);
}
```

结果为：

```
98
```

作用域使得变量一旦超出其作用范围就会失效。这样既能节省内存，又可以降低程序在逻辑上出错的概率。

无论是常量与变量，还是全局变量与局部变量，都是从数据的逻辑作用角度出发的。在物理实现上，编译系统会对内存进行分区，每个分区用于存储一类具有相同逻辑作用的数据。C 语言的分区结构如图 1-5 所示。

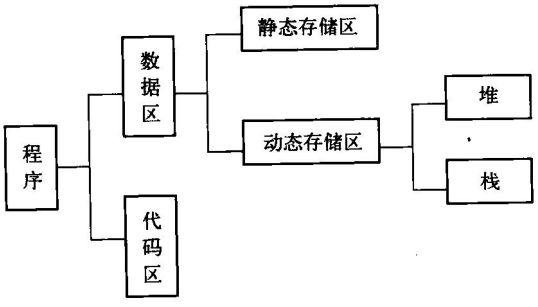

图 1-5　C 语言内存分区图

（1）代码区。代码区用于存放程序中的代码。

（2）数据区。数据区用于存放程序中的数据。

（3）静态存储区。静态存储区是在程序开始运行时就已经分配好的内存空间。在整个程