Linux C编程一站式学习

> http://akaedu.github.io/book/

Part1 C语言入门

1 程序的基本概念

- 解释型语言, 编译型语言
- 自然语言和形式语言
- 程序的调试:编译时错误, 运行时错误, 逻辑错误和语义错误
- 第一个程序

```
$ gcc main.c
$ gcc main.c -o main
$ ./main
$ gcc -Wall main.c //gcc提示所有告警信息
```

2 常量、变量和表达式

- C标准规定的转义字符:

转义字符	含义
'	单引号'（Single Quote或Apostrophe）
"	双引号"
?	问号?（Question Mark）
\	反斜线\（Backslash）
\a	响铃（Alert或Bell）
\b	退格（Backspace）
\f	分页符（Form Feed）
\n	换行（Line Feed）
\r	回车（Carriage Return）
\t	水平制表符（Horizontal Tab）
\v	垂直制表符（Vertical Tab）

> C语言规定了几个控制字符，不能用键盘直接输入，因此采用\加字母的转义序列表示。现在Windows上的文本文件用\r\n做行分隔符，许多应用层网络协议（如HTTP）也用\r\n做行分隔符，而Linux和各种UNIX上的文本文件只用\n做行分隔符。

- 常量:常量（Constant）是程序中最基本的元素，有字符（Character）常量、整数（Integer）常量、浮点数（Floating Point）常量和枚举常量。

```
printf("character: %c\ninteger: %d\nfloating point: %f\n", '}', 34, 3.14);
```

- 应避免使用以下划线开头的标识符，以下划线开头的标识符只要不 和C语言关键字冲突的都是合法的，但是往往被编译器用作一些功能扩展。
- 运算符和表达式:
  - 由运算符和操作数所组成的算式称为表达式，任何表达式都有值和类型两个基本属性；
  - 等号运算符还有一个和+ - * /不同的特性，如果一个表达式中出现多个等号，不是从左到右计算而是从右到左计算;
  - 向下取整的运算称为Floor，向上取整的运算称为Ceiling;
  - 在C语言中整数除法总是把小数部分截掉，在数轴上向零的方向取整.
- 字符类型和字符编码
  - ASCII码：字符'a'~'z'、'A'~'Z'、'0'~'9'的ASCII码都是连续的

3 简单函数

- 数学函数
  - 头文件通常位于/usr/include目录下，使用math.h等头文件需要注意gcc命令要加-lm选项，因为数学函数位于libm.s库文件中(通常位于/lib文件夹下)
  - c标准库和glibc：c标准主要有两部分组成。一部分描述c语法，另一部分描述c标准库。Linux平台上使用最广泛的c函数库是glibc，glibc提供一组头文件和库文件，最基本最常用的标准库函数和系统函数在libc.so库文件中，有些数学计算依赖于libm.so，有些多线程的c程序依赖于libpthread.so。
- 自定义函数
  - $?是Shell中的一个特殊变量，表示上一条命令的退出状态,`echo $?`
  - 操作系统在调用main时是传参数的，main的标准形式应该是`int main(int argc, char *argv[])`，c标准也允许int main(void)这种写法。除了这两种形式外，**定义main函数的其它写法都是错误的或不可移植的**。
  - 函数声明、函数原型(Prototype)、函数定义：在代码中可以单独写一个函数原型，后面加;结束，而不写函数体，这种写法叫函数声明而不叫函数定义，类似于变量，只有分配存储空间的变量声明才是变量定义(带函数体的函数声明才是函数定义);
  - **先声明，后使用**;
  - 隐式声明:隐式声明的函数返回值类型都是int.
- 形参和实参

  - 使用命令`man 3 printf`可以查看printf的声明:`int printf(const char*format, ...);`，第一个参数是char*类型的，后面的...可以代表任意多个参数，这些参数的类型也是不确定的，成为可变参数。

  - Man Page：Man Page是Linux开发最常用的参考手册，由很多页面组成，每个页 面描述一个主题，这些页面被组织成若干个Section：

Section	描述
1	用户命令，如ls(1)

Section	描述
2	系统调用，如_exit(2)
3	库函数，如printf(3)
4	特殊文件，例如null(4)描述了设备文 件/dev/null、/dev/zero的作用
5	系统配置文件的格式，例如passwd(5)描述了 系统配置文件/etc/passwd的格式
6	游戏
7	其它杂项，例如bash-builtins(7)描述 了bash的各种内建命令
8	系统管理命令，例如ifconfig(8)

Man Page中有些页面有重名，比如敲man printf命令看到的并不 是C函数printf，而是位于第1个Section的系统命令printf，要查看 位于第3个Section的printf函数应该敲man 3 printf，也可以敲man -k printf命令搜索哪些页面的主题包含printf关键字.

- 局部变量和全局变量:局部变量可以用类型相符的任意表达式来初始化，而全局变量只能用常量表达式.
- 程序开始运行时要用适当的值来初始化全局变量，所以初始值必须保存在编译生成的可执行文件中.如下面的全局变量初始化不合法:

```
int minute = 360;
int hour = minute / 60;//illegal
```

4 分支语句

- if语句
- if/else语句
- 布尔代数
  - 逻辑非运算符只有一个操作数，称为**单目运算符（Unary Operator）**，以前讲过的加减乘 除、赋值、相等性、关系、逻辑与、逻辑或运算符都有两个操作数，称为**双目运算符（Binary Operator）**。
- switch语句
  - 格式:

```
switch (控制表达式) {
case 常量表达式:  语句列表
case 常量表达式:  语句列表
...
default:  语句列表 }
```

  - case后面跟表达式的必须是常量表达式;
  - 浮点型不适合做精确比较，所以C语言规定case后面跟的必须是整型 常量表达式;
  - 进入case后如果没有遇到break语句就会一直往下执行.

5 深入理解函数

- return语句
- 增量式开发：解决问题的过程是把大的问题分成小的问题，小的问题再分成更小的问题，这个过程在代码中的体 现就是函数的分层设计（Stratify）。
- 递归
  - 随着函数调用的层层深入，存储空间的一端逐渐增长，然后 随着函数调用的层层返回，存储空间的这一端又逐渐缩短；具有这种性 质的数据结构称为堆栈或栈（Stack）;
  - 写递归函数时一定要记得写Base Case，否则即使递推关系正确，整个函数也不正确，函数就会永远调用下去，直到操作系统为程序预留的栈空间耗尽程序崩溃（段错误）为 止，这称为无穷递归（Infinite recursion）;
  - 递归和循环是等价的，用循环能做 的事用递归都能做。

6 循环语句

- while循环
- do/while循环:while语句先测试控制表达式的值再执行循环体，而do/while语句先执行循环体再测试控制表达式的值
- for语句
- break语句和continue语句
- goto语句:滥用goto语句会使程序的控制流程非常复杂，可读性很差.通常goto语句只用于这种场合，一个函数中任何地方出现了错误条件都可以立即跳转到函数末尾做 出错处理（例如释放先前分配的资源、恢复先前改动过的全局变量等），处理完之后函数返回.

7 结构体

- 复合类型和结构体
  - 基本类型、复合类型
  - struct定义

```
struct complex{
    double x,y;
} z1, z2; //同时定义变量z1, z2
struct complex z3;
```

  - 访问struct成员，`z1.x`
  - struct在定义时初始化

```
struct complex{
    double x,y;
} z1, z2; //同时定义变量z1, z2
struct complex z3 = {3.0, 4.0};
```

    - 有时候结构体或数组中只有某一个或某几个成员需要初始化，其它成员都用0初始化即可，用Designated Initializer语法可以针对每个成员做初始化（Memberwise Initialization）

```
struct complex z3 = {.y = 4.0}; //z3.x = 0.0, z3.y = 4.0
```

- 数据抽象
- 数据类型标志
    - 通过数据类型标志和两个浮点数适配直角坐标和极坐标

```
enum coordinate_type { RECTANGULAR, POLAR }; //枚举类型
struct complex_struct {
enum coordinate_type t; //注意声明的方式
double a, b;
};
```

- 结构体嵌套

8 数组

- 数组基本概念
    - 后缀运算符：后缀++、后缀--、结构体取成员.、数组取下标[]、函数调用();单目运算符(或者叫前缀运算符)：前缀++、前缀--、正号+、负号-、逻辑非!;C语言中后缀运算符的优先级最高，单目运算符的优先级仅次于后缀运算符，比其它运算符 的优先级都高;
    - **C编译器并不检查数组访问越界错误，需要尤其注意，事后依靠调试来解决问题的成本是很高的;**
    - 数组也可以像结构体一样初始化，未赋初值的元素也是用0来初始化`int a[3] = {5,2};`,利用C99的新特性也可以做Memberwise Initialization:`int a[3] = {[2] = 2]};`
- 数组应用：统计随机数
    - C标准库中生成伪随机数的是rand函数，使用这个函数需要包含头文件stdlib.h，它没有参数，返回值是一个介于0和RAND_MAX之间的接近均匀分布的整数。RAND_MAX是该头文件中定义的一个常量，通常我们用到的随机数是限定在某个范围之中的，例如0~9:`int x = rand() % 10;`
    - **用gcc的-E选项可以看到预处理之后、编译之前的程序(展开头文件，替换#define)**
    - C标准库允许我们自己指定一个初值，然后在此基础上生成伪随机数，这个初值称为Seed，可以用srand函数指定Seed。通常我们通过别的途径得到一个不确定的数作为Seed，例如调用time函数得到当前系统时间距1970年1月1日00:00:00[18]的秒数，然后传给srand:`srand(time(NULL));//#include<time.h>`
    - 1970.1.1:各种派生自UNIX的系统都把这个时刻称为Epoch，因为UNIX系统最早发明于1969年
- 字符串
    - 。如果要用一个字符串字面值准确地初始化一个字符数组，最好的办法是不指定数组的长度，让编译器自己计算：`char str[] = "hello world.";//或 char* str = "hello world.";`
- 多维数组

9 编码风格

Linux内核的[CodingStyle]

- 缩进和空白
  - 关键字if、while、for与其后的控制表达式的(括号之间插入一个空格分隔，但括号内的表达式 应紧贴括号;
  - 双目运算符的两侧各插入一个空格分隔，单目运算符和操作数之间不加空格;
  - 后缀运算符和操作数之间不加空格;
  - 由于UNIX系统标准的字符终端是24行80列的，接近或大于80个字符的较长语句要折行写，折行后用空格和上面的表达式或参数对齐

```
if (sqrt(x*x + y*y) > 5.0
    && x < 0.0
    && y > 0.0)
```

  - 较长的字符串可以断成多个字符串然后分行书写

```
printf("This is such a long sentence that "
       "it cannot be held within a line\n");
```

  - if/else、while、do/while、for、switch这些可以带语句块的语句，语句块的{或}应该和关键 字写在同一行，用空格隔开，而不是单独占一行

```
if (...) {
→语句列表 }
else if (...) {
→语句列表
}
```

  - 函数定义的{和}单独占一行，这一点和语句块的规定不同

```
int foo(int a, int b)
{
    ...
}
```

  - switch和语句块里的case、default对齐写
  - 代码中每个逻辑段落之间应该用一个空行分隔开
- 注释
  - 单行注释应采用/* comment */的形式
  - 多行注释最常见的是这种形式:

```
/*
 * Multi-line
```

```
 *
 * comment
 */
```

  - 整个源文件的顶部注释。说明此模块的相关信息，例如文件名、作者和版本历史等，顶头写不缩进;
  - 函数注释。说明此函数的功能、参数、返回值、错误码等，写在函数定义上侧，和此函数定义 之间不留空行，顶头写不缩进;
  - 相对独立的语句组注释。对这一组语句做特别说明，写在语句组上侧，和此语句组之间不留空行，与当前语句组的缩进一致
  - 代码行右侧的简短注释。对当前代码行做特别说明，一般为单行注释
  - 复杂的结构体定义比函数更需要注释
  - 复杂的宏定义和变量声明也需要注释

- 标识符命名
  - 标识符命名要清晰明了，可以使用完整的单词和易于理解的缩写,例如count写成cnt，block写成blk，length写成len
  - 内核编码风格规定变量、函数和类型采用全小写加下划线的方式命名，常量（比如**宏定义和枚举常量**）采用全大写加下划线的方式命名
  - 全局变量和全局函数的命名一定要详细，不惜多用几个单词多写几个下划线
- 函数
  - 实现一个函数只是为了做好一件事情，不要把函数设计成用途广泛、面面俱到的，这样的函数肯定会超长，而且往往不可重用，维护困难
  - 函数内部的缩进层次不宜过多，一般以少于4层为宜。如果缩进层次太多就说明设计得太复杂了，应考虑分割成更小的函数（Helper Function）来调用
  - 函数不要写得太长，建议在24行的标准终端上不超过两屏，太长会造成阅读困难
  - 执行函数就是执行一个动作，函数名通常应包含动词，例如get_current、radix_tree_insert
  - 比较重要的函数定义上侧必须加注释，说明此函数的功能、参数、返回值、错误码等
  - 另一种度量函数复杂度的办法是看有多少个局部变量，5到10个局部变量已经很多了，再多就很难维护了，应该考虑分割成多个函数
- indent工具
  - indent工具可以把代码格式化成某种风格

```
indent -kr -i8 main.c
```

-kr选项表示K&R风格，-i8表示缩进8个空格的长度，基本上-kr -i8这两个参数就够用了

10 gdb

- 单步执行和跟踪函数调用
  - 在编译时要加上-g选项，生成的可执行文件才能用gdb进行源码级调试

```

```

```
$ gcc -g main.c -o main
$ gdb main
```

- 断点
- 观察点
- 段错误

11 排序与查找

- 算法的概念
- 插入排序
- 算法复杂度分析
- **归并排序**
- 线性查找
- 折半查找

12 栈与队列

- 数据结构的概念
- 堆栈
  - 操作:push、pop
- 深度优先搜索
- 队列与广度优先搜索
  - 队列(FIFO)操作: enqueue、dequeue
- 环形队列