粤嵌教育

V 3.0

粤嵌教育嵌入式学院

LINUX C编程秘籍

1

内容提要

粤嵌教育

2

1

粤嵌教育

第一节

一、LINUX C编程前奏
二、典型C程序实例概览
三、数据类型

3

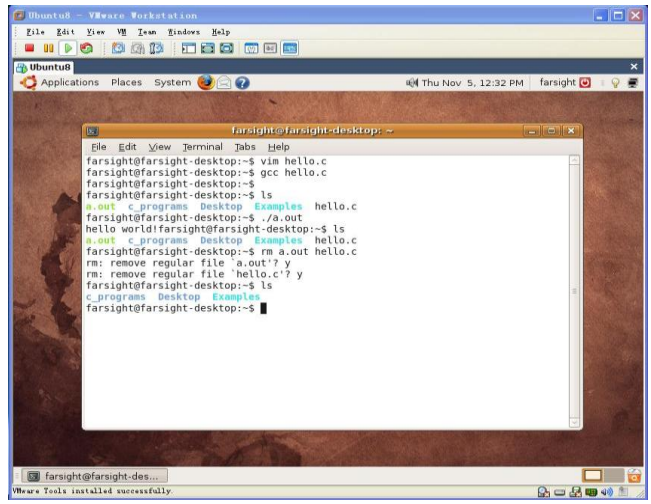
粤嵌教育

一、LINUX C编程前奏

4

一 Linux C编程前奏

虚拟机 VMware



5

一 Linux C编程前奏

自由的ubuntu

- 什么是Ubuntu?
  - Ubuntu(乌帮图)是一个非洲词汇，它的意思是“人性对待他人”或“群在故我在”。Ubuntu发行版将Ubuntu精神带到软件世界之中。
  - 目前已有大量各种各样基于GNU/Linux的操作系统，例如：Debian, SuSE, Gentoo, RedHat 和 Mandriva，当然还有Ubuntu。Debian 是一个广受称道、技术先进且有着良好支持的发行版，Ubuntu 正是基于 Debian 之上，旨在创建一个可以为桌面和服务器提供一个最新且一贯的 Linux 系统。
  - Ubuntu项目完全遵从开源软件开发原则，并鼓励人们使用、完善和传播开源软件。也就是说Ubuntu目前是并将永远是免费的。

6

一 Linux C编程前奏

自由的ubuntu

- GNU/Linux系统简介
  - 桌面系统Gnome
    - 成熟的linux桌面系统有Gnome和KDE，其中Gnome用C编写，而KDE则是用C++编写的。
  - 终端与shell
    - 打开一个终端，即运行一个shell程序。shell是一个命令行解释器，它使得用户能够与操作系统进行交互。
  - APT软件包管理
    - 常用命令集有：apt-get, apt-cache等。
  - Linux分区与目录结构
    - Linux中分区从属于目录，Windows中目录从属于分区。

7

一 Linux C编程前奏

shell基础

- shell命令提示符
  - Shell提示符标识了命令行的开始，通常Shell命令提示符采用以下的格式：
  - username@hostname:pathname$
  - 其中：
    - username → 当前登陆的用户名
    - @ → 用户名与主机名的分隔符
    - hostname → 当前登陆的主机名
    - : → 主机名与路径名的分隔符
    - pathname → 当前路径名
    - $ → 普通用户命令开始提示符(超级用户是#)

8

一 Linux C编程前奏

shell基础

粤嵌教育

- shell命令格式
  - $command [-options] argument1 argument2 …
  - 其中：
    - command → 命令名称
    - options → 选项（一般由连字符-引导）
    - argument → 参数
  - 命令的三要素之间要用空格隔开。
  - 一条命令要书写多行，用反斜杠 \ 表明未结束。
  - 多条命令同时写在一行，用分号 ; 隔开。

9

一 Linux C编程前奏

shell基础

粤嵌教育

- shell常用的重要命令
  - ls：列出文件名。
    - 例如：ls ~/dir(列出目录dir下的所有文件名，若dir是一普通文件，则仅列出dir本身)
  - rm：删除文件。
    - 例如：rm file(删除文件file，若file是一目录，则需加上选项-r，即 rm file -r)
  - cp：复制文件。
    - 例如：cp file1 file2(把文件file1复制一份，并命名为file2)
  - mv：移动或重命名文件。
    - 例如：mv file ./dir(把文件file移动到./dir目录下)

10

一 Linux C编程前奏

shell基础

粤嵌教育

- shell常用的重要命令
  - mkdir：创建目录。
    - 例如：mkdir dir(在当前目录中创建一个目录dir)
  - cd：转换当前目录。
    - 例如：cd ~/dir(把当前目录转换为主目录下的dir目录)
  - file：查看文件属性。
    - 例如：file helloworld.c(查看文件helloworld.c的属性)
  - man：获取帮助。
    - 例如：man ls(查看ls的帮助文档)
  - pwd：显示当前路径。

11

一 Linux C编程前奏

字符编程利器 vim

粤嵌教育

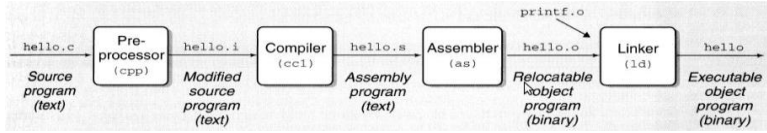
- 编辑模式
  - 进入编辑模式按i(在光标当前处插入)、a (在光标之后插入) 、o (在光标当前行的下一行插入)等。
- 命令模式
  - 按ESC进入命令模式。
- 剪切当前行dd、复制当前行yy、粘贴p
  - 剪切单词dw，剪切字符x，复制单词yw
- 保存:w，退出:q，强制退出:q!
- 查找/或?，替换:%s/old_string/new_string

12

一 Linux C编程前奏

GNU编译器 gcc

- 预处理：gcc hello.c -E -o hello.i
  - 调用预处理器cpp，完成诸如宏展开、处理条件编译、删除注释等工作。一般生成.i文件。
- 编译：gcc hello.c -S -o hello.s
  - 调用编译器cc1，把源程序翻译成对应于目标系统的汇编文件.s。
- 汇编：gcc hello.c -c -o hello.o
  - 调用汇编器as，将汇编指令翻译成机器指令，生成可重定位目标文件。
- 链接：gcc hello.c -o hello
  - 调用链接器ld，将生成的可重定位文件与相关库文件链接，生成可执行目标文件。



13

一 Linux C编程前奏

你需要知道的C编程常识

- C与UNIX
  - 1972年，伟大的D.M.Ritchie在B语言的基础上设计出来C语言，其目的是为了描述和实现UNIX操作系统。
- ANSI C
  - 1983年，美国国家标准协会(ANSI)为C制订的工业标准，称之为ANSI C。
- ISO C89和ISO C99
  - 1990年和1999年，国际标准化组织(ISO)两次为C语言制订的标准。
- POSIX标准
  - UNIX世界中最流行的API是基于POSIX标准的，POSIX是LINUX前进的灯塔，glibc库是遵循POSIX标准的典型代表。
- GNU C
  - LINUX内核开发者使用的C语言涵盖了ISO C99标准和GNU C扩展特性，GNU C扩展特性包括内联函数、内联汇编、分支声明等。
- 标准C库
  - 库函数是对系统调用的进一步封装，旨在为应用层程序提供一致的编程接口，也就是通常所说的API。

14

一 Linux C编程前奏

语言标准

- 目前有许多C实现可用，理想情况下，编写C程序时，假如改程序未使用机器特定的编程技术，则它在任何实现方式中的运行应该是相同的。要在实践中做到这一点，不同的实现方式需要遵守一个公认的标准。
- Brian Kernighan和Dennis Ritchie编写的《*The C Programming Language*》早已成为大家接受的标准，通常称为K&R C或经典C。而C语言比大多数其他语言更加依赖库，所以还需要一个库标准。因为缺乏任何官方标准，所以提供UNIX实现的库成为事实上的标准。
- 随着C的发展和更加广泛地用于更多种类的系统上，使用C的群体意识到它需要一个更加全面新颖和严格的标准。于是，ANSI在1983年设立了一个委员会以发展新的标准，并于1989年正式采用。这个新标准（ANSI C）定义了语言和一个标准C库。ISO于1990年采纳了它，因此ANSI/ISO C被称为C89或C90。

15

一 Linux C编程前奏

语言标准

- 随着时间的推移，C语言标准也在其原有的基础上做了一些修订，1994年修订工作开始了，这一努力的结果是产生了C99标准。ANSI/ISO C委员会保持C语言的短小而简单，他们的意图不是为语言添加新的特性，而是为了满足新的目标。新目标包括国际化、修正其不足和改进计算的实用性。
- 这三点是主要的面向改变的目标。形成的关于更改的计划在性质上更加保守，例如让与C90和C++的不兼容性达到最小，让语言在概念上保持简单。
- 结果是C99的修改保持了C的本质特性，C继续是一种简短、清楚、高效的语言。

16

粤嵌教育

二、典型C程序实例概览

17

二 典型C程序实例概览

粤嵌教育

一个简单的Linux C程序实例分析

```
#include <stdio.h>

/* 一个简单的Linux  C程序 */
int main(void)  //main函数的返回类型为int型
{                    //函数体左右花括号各独占一行，函数体注意缩进
    int num;    //定义一个int型变量
    num = 1;  //对变量赋值
    char *pstring = “hello world”;  //定义一个char *变量并初始化之

    for(num=0; num<7; num++){  //for循环语句
        printf(“%s\n”,pstring); // 代码块统一缩进
    }
    return 0;
}
```

18

二 典型C程序实例概览

一个简单的Linux C程序实例分析

粤嵌教育

- LINUX C编程风格
  - 1、适当的空行和空格，空行一般可以出现在逻辑块之间。
  - 2、一定要有缩进，代码块中的代码一定要缩进，无论是函数体、循环体、switch多路分支和普通复合语句。（注意，C语言是一种“自由体”语言，程序中的所有空白符都与逻辑无关，只起排版作用，因此空行缩进等格式对语法来说不是必须的）
  - 3、适当的注释，程序具有一定逻辑和算法复杂度的时候，在关键的地方做好注释。写注释是一种修养。（注意，注释要写代码的功能，而不是其原理）
  - 4、LINUX C风格中的函数名和变量名一般用小写字母加下划线，比如变量apple_tree，而不是像在其他平台中会用到诸如AppleTtree这样的变量名或者函数名。（当然常量通常用大写字母表示）
  - 5、最好按照规范写代码，代码不仅自己将来要看，而且经常是要给别人看的，不要自成一体。

19

二 典型C程序实例概览

一个简单的Linux C程序实例分析

粤嵌教育

- #include <stdio.h> → 包含另一个文件
  - 该行告诉编译器包含文件stdio.h中的全部信息。文件stdio.h是所有C语言编译包的一个标准部分。这这个文件对关键字输入和显示输出提供支持。
- /* 一个简单的Linux C程序 */ → 注释
  - 符号/*和*/中包含有助于使程序更清晰的注释性内容，它们只是为了帮助读者理解，在编译时将被编译器忽略。
- { → 函数体的开始
  - 这个开始花括号标志着组成函数的语句的开始。而结束花括号(})则标志着函数的结束。所有的代码块都必须用花括号括起来！
- int num; → 声明语句
  - 这个语句表明您将使用num这个变量，并且它是int(整数)类型的。
- num = 1; → 赋值语句
  - 该语句表明把值1赋给num这个变量。

20

二 典型C程序实例概览

一个简单的Linux C程序实例分析

粤嵌教育

- char *pstring = “hello world”; → 定义字符指针变量
  - 在定义变量的同时，我们可以对其进行赋初值，这样的赋值叫做初始化。对于静态存储的变量，初始化必须用常量而不能用变量。
- for(num=0; num<7; num++) → for循环语句
  - 控制其花括号中语句的执行次数。
- printf(“%s\n”, pstring); → 调用库函数printf()
  - printf函数负责把变量pstring按照你指定的格式“%s”输出到标准输出设备上（也就是屏幕）。
- return 0; → 返回语句
  - C函数可以给他的使用者提供或返回一个数值。现在可以暂时认为这一行用来满足ISO/ANSI C对正确书写mani()函数所做的要求。
- } → 结束
  - 显然，一个程序必须以一个右花括号中止。

21

二 典型C程序实例概览

main函数

粤嵌教育

- int main(void){…} → good
  - 这行代码声明了一个main函数，一个C程序（我们不考虑一些例外的情况）总是从被称为main( )的函数开始执行的。
  - int指明了main( )函数的返回类型，这意味着它的返回值类型是整型，这个返回值返回给操作系统。
  - 圆括号中写了void，表明这个简单的例子不需要外界传递任何信息给它，相反，如果main( )函数需要接收参数，则可以写成：
    - int main(int argc, char *argv[]) { … }
  - 其中argc是参数个数，argv是一个指针数组，这是main函数的标准声明形式
- main( ) { … } → bad
  - 你可能会看到这种老版本的C代码，C90勉强允许这种形式，但是C99标准不允许，因此即使你现在用的编译器允许这种形式，你也最好别这么写。
- void main( ){…} → bad
  - 有些编译器允许这种形式，但是还没有任何标准考虑接受它。因而编译器不必接受这种形式，并且许多编译器也不这样做。总之，请使用标准形式，那样当你把程序从一个编译器移到另一个编译器时也不会有问题。

22

二 典型C程序实例概览

声明

- **int num;**
  - 程序中的这一行叫做声明语句(declaration statement)。这种语句是C语言中最重要的功能之一。这个句话做了两件事情：1 在函数中声明了一个叫做i的变量，在恰当的位置我们就可以使用它；2 int说明了这个变量是一个整数。编译器使用这个信息为变量i在内存中分配一个合适的存储空间。
  - 在C语言中所有的变量都必须在使用之前定义。
  - num是该变量的名字，变量的要符合一定的规范：例如不能与关键字重名，只能以字母或下划线开头等。右边的表格显示了那些命名是正确的，而那些命名是非法的。

正确的名字	错误的名字
wiggles	$Z]**
cat2	2cat
Hot_Tub	Hot-Tub
taxRate	tax rate
_kcab	don't

23

二 典型C程序实例概览

赋值和printf函数

- **num = 1;**
  - 这行程序是一个赋值语句(assignment statement)。赋值语句是C语言的基本操作之一。这个例子的意思是把值1赋给变量num。前面的 int num; 语句在计算机内存中为变量num分配了空间，该赋值语句在哪个地方为变量存储了一个值。如果你愿意，你以后还可以给num赋另一个值。正是这个原因，我们才把num称之为变量。
  - 注意，赋值语句的复制顺序是从右到左的。
- **printf("%s\n", pstring);**
  - printf是C语言的一个标准函数，其后面紧跟着的一对圆括号()表明printf是一个函数名，括号里的内容是从main函数传到函数printf的信息，这些信息称之为参数(arguments)。
  - printf是一个变参函数，其参数的个数和类型体现在双引号中的“格式控制串”中，例如在这个例子中是%s，表示printf接受一个字符指针参数。
  - 换行符‘\n’是转义字符(escape sequence)的一个例子，转义字符通常用于代表难于表达的或是无法键入的字符。比如\t表示制表符，\b表示退格键等等。

24

二 典型C程序实例概览

return语句

粤嵌教育

- **return 0;**
  - return语句是程序的最后一个语句。在int main(void)中int表示main函数返回值应该是一个整数。C标准要求main函数这么做。
  - 带有返回值的C语言函数要使用一个return语句，该语句包括关键字return，后面紧跟着要返回的值，然后是一个分号。
  - 对于main函数来说，如果你漏掉了return语句，大多数编译器会给出一个警告，但仍将编译改程序。此时，你可以暂时把main函数中的return语句看做是保持逻辑连贯性所需的内容。
  - 注意：return语句出现在普通函数中时，表示返回其调用者处，return语句出现在main函数中时，表示退出整个进程。

25

粤嵌教育

三、数据类型

26

三 数据类型

常量数据和变量

粤嵌教育

- 在程序的指示下，计算机可以做很多事情，比如数值计算，名字排序，执行语音或视频命令，计算彗星轨道，准备邮寄列表等等，要完成这些任务，需要用到数据，即承载信息的数字与字符。
- 有些数据可以在程序使用之前预先设定并在整个运行过程中没有变化，这称之为常量。另外的数据在程序运行过程中可能变化或者被赋值，这称之为变量。
- 变量和常量的区别在于，变量的值可以在程序执行过程中变化或者指定，而常量则不可以。

27

三 数据类型

基本数据类型

粤嵌教育

- 常量
  - 整型常量和浮点型常量
    - 整型： 123，0123，0x123ff
    - 浮点型： 3.14，3.14e3，3.14E-2
  - 字符型常量和字符串型常量
    - 字符’a’，’\0’，’\n’，’\027’，’\x1ff’；字符串”a”，”hello world!”
    - 表示不可见字符有三种方法：
      - 1：使用ASCII码，例如：char beep = 7;
      - 2：使用转义序列，例如：char beep = ‘\a’;
      - 3：使用转义字符，例如：char beep = ‘\007’; //或char beep = ‘\x07’

28

三 数据类型

基本数据类型

粤嵌教育

- 我们有时候需要在程序中使用常量，例如要计算一个圆的周长：
  - circumference = 3.14159 * diameter;
- 这里的3.14159代表了圆周率常量，有足够充足的理由让我们用一个常量符号来替代这种直接用数字表示的方法，比如写成：
  - circumference = PI * diameter;
- 这样做的理由包括：
  - 1、使用符号常量通常比一个数字更容易读懂。
  - 2、假设在多个地方使用了同一个常量，那修改起来将会很方便。
- C的预处理器允许定义常量，方法如下：
  - #define PI 3.14159

29

三 数据类型

基本数据类型

粤嵌教育

- 请注意格式，首先是#define其次是常量的符号名，接着是常量的值。其一般形式为：
  - #define NAME value
- C中的常量一般用大写字母表示。大写常量使程序更容易阅读。另外，符号常量也跟变量一样，一定要符合命名规则。
- const修饰符
  - 这里要特别强调的是，用const修饰的标识符是只读变量不是常量，例如：const int i = 2; 该语句声明了一个只读变量i且其初值为2。
- 系统定义的明显常量
  - C头文件limits.h跟整数大小限制相关的详细信息，文件中定义了一系列应用于你的实现的明显常量。

30

三 数据类型

常量数据和变量

粤嵌教育

- 除了变量和常量的区别，各种数据类型间也有不同。一些数据是数字，而另一些则是字符（包括可见字符和不可见字符），而单就数字而言，我们又可以细分为正整数、整数、实数等等。C通过识别一些基本数据类型来区分不同类型的数据。

原来的K&R关键字	C90关键字	C99关键字
int	signed	_Bool
long	void	_Complex
short		_Imaginary
unsigned		
char		
float		
double		

31

三 数据类型

基本数据类型

粤嵌教育

- 整型 int
  - 声明 int 变量
    - 之前的典型C实例中已经看到，int关键字用于声明基本的整数变量。书写格式为先写“int”后加变量名，再加一个分号。比如：
    - int erns;
    - int apples, cows, goats;
  - 初始化 int 变量
    - 我们可以对定义了的变量赋值，也可以用scanf函数来为变量赋值，也可以用第三种办法：初始化。如下所示：
    - int apples = 1;
    - 初始化的特点就是在定义变量的同时给它赋一个初值。简而言之，声明语句为变量创建、标定存储空间并为其指定初始值。

32

三 数据类型

基本数据类型

粤嵌教育

- int 类型常量
  - 1，21，94等都是整数常量。C把不含小数点和指数的数当做整数，比如22和-44是整数常量，而22.0和2.2e1则不是。
  - C把大多数整数常量看做 int 类型。
- 八进制和十六进制
  - 一般地，C假设整数常量为十进制数，或者称为以10为基数的数。然而，很多程序员十分熟悉八进制和十六进制。因为8和16是2的幂，所以这些数制可以更加方便地表示与计算机相关的值。
  - C语言中用0开头来表示八进制数，用0x开头来表示十六进制数。例如：
    - 123 ← 这是一个10进制数
    - 0123 ← 这是一个8进制数
    - 0X123 ← 这是一个16进制数
  - 注意：这种使用不同数值系统的选择是为了方便而提供的，它们并不影响数字在计算机内部的真实存储。

33

三 数据类型

基本数据类型

粤嵌教育

- 初学语言的时候，int类型会满足你的大多数需求。我们可以用int来定义整形数据，用来表示一个整数。
- C提供3个附属关键字来修饰基本的整数类型：short、long和unsigned。需要注意以下几点：
  - short int(或简写为short)类型，用于仅需小数值的场合以节省空间。同int一样，short是一种有符号类型。
  - long int(或简写为long)类型，用于使用大数值的场合，long也是一种有符号的类型。
  - unsigned int(或简写为unsigned)类型，用于只使用非负值场合。
  - 关键字signed可以和任何有符号类型一起使用，它是可选的，使得数据类型更加明确。例如short，short int， signed short以及signed short int都表示一种类型，他们是等价的。

34

三 数据类型

基本数据类型

粤嵌教育

- 字符类型 char
  - 字符类型用于表示字母和标点符号之列的数据，但是从技术上来说实际上char型是整型，char型存储的是整数而不是字符。为了处理字符，计算机使用一种数字编码，用特定的整数表示特定的字符。目前我们最常用的编码是美国的ASCII码。
  - 标准的ASCII码值从0到127，只需7位即可表示。而char类型通常定义为使用8位内存单元，这表示标准的ASCII码绰绰有余。许多系统提供扩展ASCII编码，从更普遍的角度来看，C保证char类型足够大，以存储其实现所在的系统上的基本字符集。
  - 字符常量及其初始化：
    - char ch; /* 声明了一个char变量 */
    - ch = 'A'; /* 可以，编译器会把字符常量A对应的ASCII码65赋给变量ch */
    - ch = A; /* 不可以！编译器会把A是一个变量！ */
    - ch = "A"; /* 不可以！编译器会把A看做一个字符串！ */

35

三 数据类型

基本数据类型

粤嵌教育

- 布尔类型 _Bool、bool
  - _Bool类型由C99引入，用于表示布尔值，即逻辑值true（真）和false（假）。因为C用值1表示true，用值0表示false，所以_Bool类型实际上也是一种整数类型。只是原则上它仅仅需要1位来进行存储。
  - 要使用bool类型和true和false关键字，必须包含stdbool.h头文件。
  - 程序使用布尔值来选择执行哪个代码分支，比如循环或者跳转控制语句。

36

三 数据类型

基本数据类型

粤嵌教育

- 浮点型 float、double、long double
  - 浮点数(floating-point)差不多可以和数学中的实数相对应。实数包含了整数之间的那些书。2.75、3.16E7和2e-8都是浮点数。加了小数点就是浮点型数据，因此整数7是整型数据，而整数7.00则是一个浮点数据。
  - 最重要的一点是浮点数与整数的存储方案不同。浮点数表示法将一个数分为小数部分和指数部分分开存储，所以也许int型7和float型7.00占用的内存块大小一样，数值也一样，但是它们的内部表示是完全不同的。
- void型
  - 关键字void用来修饰指针变量、函数返回类型和函数参数列表。

37

三 数据类型

基本数据类型

粤嵌教育

- 下面的表格列出了典型系统的浮点数情况

类型	Macintosh	Linux	Windows	ANSI C规定的最小值
float	6位	6位	6位	6位
	-37到38	-37到38	-37到38	-37到37
double	18位	15位	15位	10位
	-4931到4932	-307到308	-307到308	-37到37
long double	18位	18位	18位	10位
	-4931到4932	-4931到4932	-4931到4932	-37到37

对于每种类型，上面的行代表有效数字位数，下面的行代表指数的范围

38

三 数据类型

基本数据类型

粤嵌教育

- 数据类型转换
  - 隐式类型转换
    - int → long → unsigned long → long long → unsigned long long → float → double → long double
    - 记住：隐式转换都是“小”类型向“大”类型转换的。
  - 强制类型转换
    - 语法格式： (数据类型)表达式;
  - 需要注意的问题：
    - 隐式类型转换都是低精度数据类型向高精度类型转换的，因此不会产生丢失信息的危险，相反，强制类型转换一般是要把高精度类型转换成低精度类型，存在丢失数据信息的危险。
    - 无符号型比有符号型占用存储空间要大，因此在类型转换中它们之间的转换方向是：有符号类型向无符号类型转换。

39

三 数据类型

可移植数据类型

粤嵌教育

- 你可能认为自己已经接触到了足够多的名字了，可是这些基本的名字不够明确，比如，知道一个变量是int型，可是你不知道它有多少位。
- 解决这个问题，C99提供了一个可选的名字集合，以确切地描述有关信息。例如：
  - int16_t表示一个16位的有符号整数类型，uint32_t表示一个32位无符号整数类型。
- 要使这些名字有效，应当在程序中包含inttypes.h头文件。

40

第一节强化训练

粤嵌教育

- 1、指出下列常量的类型和意义（如果有的话）：
  - a) '\b'
  - b) 1066
  - c) 99.44
  - d) 0XAA
  - e) 2.0e30
- 2、编写一个程序，实现如下功能：用户输入一个ASCII码值(如66)，程序输出相应的字符。

41

第一节强化训练

粤嵌教育

- 3、Mr. Bing写了下面这个程序，请指出你认为不妥的地方：
  - include "stdio.h"
  - main{ }
  - (
    - float g; h;
    - float tax, rate;
    - g = e21;
    - tax = rate * g;
    - printf("%f\n", tax);
  -)

42

第一节强化训练

粤嵌教育

- 4、一年大约有3.1536*10^7s。编写一个程序，要求输入你的年龄，然后显示该年龄等于多少秒。
- 5、一个水分子的质量大约为3.0*10^{-23}g，1夸脱水大约有950g。编写一个程序，要求输入水的夸脱数，然后显示这么多水中包含多少个水分子。
- 6、假设c为char类型变量。使用转义序列、十进制值、八进制字符常量以及十六进制字符常量等方法将其赋值为回车符（使用ASCII码）。

43

第一节强化训练

粤嵌教育

- 7、说说'A'与"A"有什么区别？
- 8、有时候我们需要使用uint32_t类型变量代替unsigned int类型变量的原因是什么？

44

粤嵌教育

第二节

四、字符串和格式化IO
五、运算符，表达式和语句

45

粤嵌教育

四、字符串和格式化IO

46

四 字符串和格式化IO

字符串简介

- 字符串(character string)就是一个或多个字符的序列。下面是一个字符串的例子：
  - “helloworld”
- 双引号不是字符串的一部分，它们只是通知编译器其中包含了一个字符串，正如单引号标识着一个字符一样。
- C没有专门的字符串类型，而是把它存储在char型数组当中。字符串的每个字符存放在相邻的存储单元中。如下图：

h	e	l	l	o	w	o	r	l	d	\0

- 请注意，数组中的最后一个位置显示字符\0，这个字符就是空字符(null character)，C用它来标记字符串的结束。

47

四 字符串和格式化IO

字符串简介

- 字符串和字符
  - 必须知道，字符常量’x’和字符串常量”x”是完全不同的。其中一个区别是前者是基本类型，后者则是派生类型。第二个区别是”x”实际上由两个字符组成。
- strlen()函数
  - 与sizeof运算符不同，strlen()函数以字符为单位给出字符串的长度，其计算的长度的方法是遇到字符串结束符’\0’为止。
  - 要在程序中使用strlen()函数，则需要包含头文件string.h

48

四 字符串和格式化IO

格式化IO函数

粤嵌教育

- printf()和scanf()函数用于使你能够和程序通信。它们被称为输入输出函数，简称IO函数。
- printf()函数和scanf()虽然用途不同，但它们的工作几乎相同，都是用了格式控制串和参数列表。
- 使用printf打印变量的指令取决于变量的类型，例如打印整数时用%d，打印字符时用%c等，这些符号被称为转换说明(conversion specification)，是它们指定了改如何把数据转换成可现实的形式。

四 字符串和格式化IO

转换说明中的转换类型

粤嵌教育

转换类型	说明
%d	有符号十进制整数
%o	无符号八进制整数
%u	无符号十进制整数
%x	无符号十六进制整数
%c	一个字符
%s	字符串
%f	十进制计数法的浮点数
%e	E-计数法的浮点数
%p	指针

四 字符串和格式化IO

格式化IO函数

- 注意事项：
  - 1、不要忘记给格式控制串后面的列表中的每个项目都是用一个转换说明。例如不要写这样的代码：
    - printf(“apple tree:%d, %d\n”, num);
  - 2、这里没有值使用第二个%d。这种错误的结果取决于具体系统，最好的结果是输出无意义的值。
  - 3、如果只打印一个语句，那么不需要任何转换说明。
  - 4、printf()函数打印列表中使用的是值，而不管是变量常量还是表达式。例如:
  - 5、如果要打印%本身，则需要用%%代替。

51

四 字符串和格式化IO

printf修饰符

- 可以在%和定义转换字符之间通过插入修饰符对基本的转换说明加以修改。下表列出了可以插入的合法字符。

修饰符	意义
标志	-：左对齐，示例 “%20s” +：显示正负号，示例 “%+6.2f” 0：填充0，示例 “%010d”
digit(s)	字段宽度最小值，示例 “%4d”
.digit(s)	精度，示例 “%5.2f”
h	和整数转换说明符一起使用，表示一个短整型，示例 “%hu”，”%6.4hd”
l	和整数转换说明符一起使用，表示一个长整型，示例 “%ld”，”%8lu”
ll	和整数转换说明符一起使用，表示一个long long型或unsigned long long型，示例 “%lld”，”%8llu”

52

四 字符串和格式化IO

格式化IO函数

- 一、格式化输出
  - 执行格式化输出处理的是4个printf函数：
    - int printf(char *format, …);
    - int fprintf(FILE *fp, char *format, …);
    - int sprintf(char *buf, char *format, …);
    - int snprintf(char buf, size_t n, char *format);
- 二、格式化输入
  - 执行格式化输入处理的是3个scanf函数：
    - int scanf(char *format, …);
    - int fscanf(FILE *fp, char *format, …);
    - int sscanf(char *buf, char *format, …);

53

四 字符串和格式化IO

格式化IO函数

- 一、printf将格式化数据写到标准输出，fprintf写至指定的流，sprintf将格式化的字符送入数组buf中。sprintf在该数组的尾端自动加一个null字节，但不包含在返回值中。snprintf的引入是为了解决sprintf函数缓冲区溢出问题。
- 二、scanf函数族用于分析输入字符串，并将字符序列转换成指定类型的变量。格式之后的个参数包含了变量的地址，以用转换结果初始化这些变量。

54

四 字符串和格式化IO

格式化IO函数

粤嵌教育

- 使用scanf()函数注意事项：
  - 1、scanf()函数使用的转换说明符跟printf()函数几乎相同，唯一区别的是前者用%lf来说明double型数据，而不是跟float型数据一样都是用%f.
  - 2、scanf()函数使用“空白符”来决定怎样把输入分成几个字段。它依次把转换说明与字段相匹配，并跳过它们之间的空格(当格式控制符为%c时例外)。
  - 3、当scanf()遇到不匹配数据类型的数据时，ANSI C要求函数在第一个出错的地方停止读取输入。
  - 3、如果该函数使用%s转换说明符，那么除“空白符”以外的字符都是可以接受的，换句话说%s用于读取单个单词。

55

粤嵌教育

五、运算符，表达式和语句

56

五 运算符，表达式和语句

运算符

- C语言为我们提供了丰富多彩的运算符，这些运算符包括：
  - 算术运算符：比如+、-等
  - 关系运算符：比如>、<等
  - 逻辑运算符：比如&&、||等
  - 位运算符：比如&、|等
  - 其他运算符：比如sizeof、=等

五 运算符，表达式和语句

运算符

- 赋值运算符
  - 在C里，“=”不表示“相等”，而是一个赋值运算符，例如：
    - bmw = 2002;
  - 符号=的左边是一个变量名，右边是赋给该变量的值。符号=被称为赋值运算符(assingment operator)。再强调一次：不要把上面的语句读成“bmw等于2002”，而要读成“把值2002赋给变量bmw”。赋值运算的执行顺序是从右到左的。
  - 像下面的语句：
    - 2002 = bmw;
  - 从数学角度上看也许没什么奇怪，但C不允许这样的语法，原因是2002是一个常量，我们是不能对一个常量进行赋值。

五 运算符，表达式和语句

运算符

- 几个术语
  - 数据对象
    - “数据对象”(data object)是泛指数据存储区的术语，数据存储区能用于保存值。例如，用于保存变量或者数组的数据存储区是一个数据对象。
  - 左值和右值
    - 左值(lvalue)指用于标识一个特定的数据对象的名字或表达式。例如，变量的名字是一个左值。所以对象指的是实际的数据存储，但是左值是用于识别或定位那个存储的标识符。
    - 术语右值(rvalue)指的是能赋给可修改左值的量。例如，考虑下面这个语句：
      - `bmw = 2002;`
    - 这里bmw是一个可修改的左值，2002是一个右值。右值可以是常量、变量或者任何可以产生一个值的表达式。
  - 操作数
    - 操作数即运算符操作的对象。

59

五 运算符，表达式和语句

运算符

- 1) 算术运算符

运算符	功能说明	举例
+	加法，一目取正	a+b
-	减法，一目取负	a-b
*	乘法	a*b
/	除法	a/b
%	取模（求余）	a%b
++	自加1	a++, ++b
--	自减1	a--, --b

60

五 运算符，表达式和语句

运算符

粤嵌教育

- 2) 关系运算符

运算符	功能说明	举例
>	大于	a>b
>=	大于等于	a>=5
<	小于	3<x
<=	小于等于	x<=y+1
==	等于	x+1==0
!=	不等于	c!='\0'

61

五 运算符，表达式和语句

运算符

粤嵌教育

- 3) 逻辑运算符

运算符	功能说明	举例
!	逻辑反	!(x==0)
&&	逻辑与	x>0 && x<10
||	逻辑或	y<10 || x>10

62

五 运算符，表达式和语句

运算符

粤嵌教育

- 4) 位运算符

运算符	功能说明	举例
~	位逻辑反	~a
&	位逻辑与	a&b
|	位逻辑或	a|b
^	位逻辑异或	a^b
>>	右移	a>>1
<<	左移	b<<4

63

五 运算符，表达式和语句

运算符

粤嵌教育

- 5) 其他运算符
  - 赋值运算符：=
  - 复合赋值运算符：<操作符>=
    - += -= *= /= %= >>= <<= &= ^= |=
  - 条件运算符： ?=
    - 格式：<表达式1>?<表达式2>:<表达式3>
  - sizeof运算符：sizeof
    - 格式：sizeof (操作数)
  - 逗号运算符： ,
    - 由逗号运算符构成的语句的值由最右边的表达式决定

64

4、运算符，表达式和语句

运算符的优先级及其结合性

优先级	运算符及其含义	结合性
1	[ ] ( ) . -> 后缀++ 后缀--	从左向右
2	前缀++ 前缀-- sizeof & ^ + - ~ !	从右向左
3	强制类型转换	从右向左
4	* / % (算术乘除)	从左向右
5	+ - (算术加减)	从左向右
6	<< >> (位移位)	从左向右
7	< <= > >=	从左向右
8	== !=	从左向右
9	& (位逻辑与)	从左向右
10	^ (位逻辑异或)	从左向右
11	| (位逻辑或)	从左向右
12	&&	从左向右
13	||	从左向右
14	?:	从右向左
15	= ^= /= %= += -= <<= >>= &= ^= |=	从右向左
16	,	从左向右

65

五 运算符，表达式和语句

表达式和语句

- 表达式(expression)
  - 所谓表达式就是所谓表达式是指由运算符、运算量和标点符号组成的有效序列，其目的是用来说明一个计算过程。例如：
    - 1
    - i+100
    - x = ++q%3
  - C的一个重要的属性是每一个C表达式都有一个值。为了得到这个值，你可以按照运算符优先级描述的顺序来完成运算。下面是一些表达式和它们的值。

表达式	值
-4+6	2
c=3+8	11
5>3	1
6+(c=3+8)	17

66

五 运算符，表达式和语句

表达式和语句

粤嵌教育

- 语句(statement)
  - 表达式可以独立形成语句，称为表达式语句。大多数语句由表达式构造而成，在表达式的后面加一个分号标识就构成语句。一个语句是一条完整的计算机指令。例如：
    - const = 1 //这是一个表达式
    - const = 1; //这是一个语句(表达式语句)
  - 尽管一个语句(或者至少是一个有作用的语句)是一条完整的指令，但不是所有的完整的指令都是语句。考虑下面的语句：
    - x = 6+(y=5);
  - 在这个语句中，字表达式 y=5 是一个完整的指令，但是他只是一个语句的一部分。因为一条完全的指令不鄙视一个语句，所以分号被用来识别确实是语句的指令。

67

第二节强化训练

粤嵌教育

- 1、编写一个程序，实现以下功能：用户输入一个分钟数，程序将其转换成以小时和分钟表示的时间并输出到屏幕上。（使用#define来定义一个代表60的符号常量）
- 2、编写一个程序，此程序要求输入一个整数，然后打印出从输入的值(含)到比输入的值大10(含)的所有整数值(比如输入5，则输出5到15)。要求在各个输出值之间用空格、制表符或者换行符分开。

68

第二节强化训练

粤嵌教育

- 3 、写出下面表达式运算后a的值，设原来a=12。设a和n已定义为整型变量。
  - (1) a += a
  - (2) a -= 2
  - (3) a *= 2+3
  - (4) a /= a+a
  - (5) a %= (n%2)，n的值为5
  - (6) a += a -= a *= a

69

第二节强化训练

粤嵌教育

- 4、编写一个程序，该程序要求输入一个float型数并打印概述的立方值。使用你自己设计的函数来计算该值的立方并且将它的立方打印出来。main函数负责把输入的值传递给该函数。

- 5、编写一个程序，此程序要求输入天数，然后将该值转换为星期数和天数。例如输入18，则要求输出：
  - 18 days are 2 weeks, 4days.

70

第二节强化训练

粤嵌教育

- 6、分析并解释以下程序的执行结果。
  - #include <stdio.h>
  - int main(void)
  - {
    - int a,b,c,d;
    - a=10;
    - b=a++;
    - c=++a;
    - d=10*a++;
    - printf(“b，c，d：%d，%d，%d”，b，c，d);
    - return 0;
  - }

71

第二节强化训练

粤嵌教育

- 7、分析并解释以下两个程序片段的执行结果。

片段A：

```
unsigned short i;
unsigned short index=0;
for(i = 0; i <index-1; i++)
{
    …… 
}
```

片段B：

```
unsigned short i;
unsigned long index=0;
for(i=0; i<index-1; i++)
{
    …… 
}
```

72

粤嵌教育

第三节

六、控制流
七、字符IO和输入确认

73

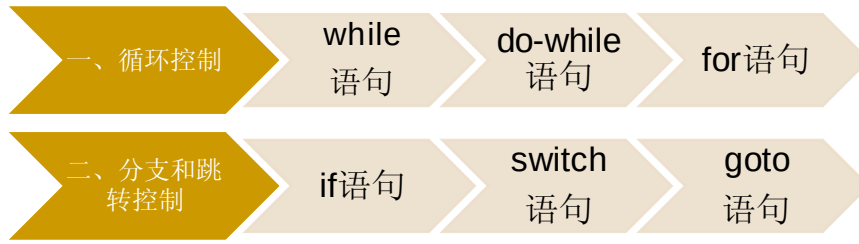
粤嵌教育

六、控制流

74

六 控制流

粤嵌教育



75

六 控制流

while循环

粤嵌教育

- while循环：先判断再执行
  - 一般形式：
    - while(expression)
      - statement
  - 例如：
    - while(i<1000){
      - i--;
    - }
  - 判断表达式是否为真，是则进入循环体，否则退出

76

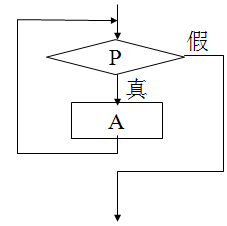
六 控制流

while循环

粤嵌教育

- while循环被称为“入口条件循环”，即在每次检查判断条件之后才执行循环。通常关系表达式会出现在一般形式中的expression中，但事实上可以是任意表达式，其结果判断都是一样的，即：如果expression为真(即表达式的值非零)，则执行一次statement，然后再次判断expression。
- 在expression变为假(零)之前要重复这个判断和执行的循环。每次循环都被称为一次迭代。

语法要点：只有位于判断条件之后的单个语句才是循环的部分。缩进是为了帮助读者而不是计算机。



77

六 控制流

do…while循环

粤嵌教育

- do-while循环：先执行再判断
  - 一般形式：
    - do
      - statement
    - while(expression);
  - 例如：
    - do{
      - i--;
    - } while(i > 0);
  - 判断表达式是否为真，是则返回循环体，否则退出

78

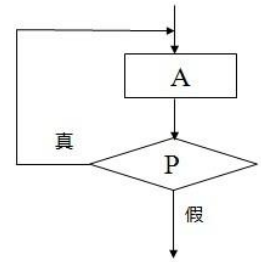
六 控制流

do…while循环

粤嵌教育

- do…while循环被称为“退出条件循环”，即在每次执行循环之后检查判断条件，这样循环中的语句就至少必须被执行一次。跟while循环一样，该形式的statement部分可以是一个简单语句，或者一个复合语句。
- 在expression变为真(非零)之前要重复这个执行和判断的循环。

语法要点：do…while循环本身是一个语句，因此它需要一个分号来结束。



79

六 控制流

for循环

粤嵌教育

- for循环：最灵活的循环控制流
  - 一般形式：
    - for(initialize; test; update)
      - statement
  - 例如：
    - for(i=0; i<1000; i++){
      - printf(“hello world!\n”); //打印1000行helloworld!
    - }
  - 判断表达式是否为真，是则返回循环体，否则退出

80

六 控制流

for循环

粤嵌教育

- for循环执行顺序：
  - 1、先计算initialize表达式，即“初始化语句”，一般用来负责初始化控制循环的变量的。
  - 2、其次计算test表达式，即“测试语句”，如果条件成立(该表达式的值为非零)，则进入第3步。如果条件不成立(该表达式的值为零)，则退出循环。
  - 3、执行statement。
- 语法要点：
  - 1、initialize表达式只执行一次。
  - 2、各个表达式之间要用分号隔开，且各个表达式可以省略，但分号不能省。

81

六 控制流

循环

粤嵌教育

- 选择哪种循环
  - 没有好坏之分，看具体应用场合。但通常说来while比do…while用得更多一些。
  - 对于while和for而言，通常它们是可以互换的，比如可以用for(;test;)来替代while(test)。再如：
    - for(initialize; test; update) ⇔
    - initialize; while(test){body; update}
- 嵌套循环
  - 嵌套循环(nested loop)指的是在一个循环体中包含另一个循环体。例如我们在按行按列处理数据的时候能用到这种技术。
  - 嵌套循环不宜太深，否则会令逻辑难以理解。

82

六 控制流

if条件跳转

粤嵌教育

- if语句被称为分支语句(branching statement)或者选择语句(selection statement)，因为它提供了一个交汇点，在此处程序需要选择两条分支中的一条前进。一般形式如下：
  - if(expression)
    - statement
- 其逻辑为：如果expression求得的值为真(非零)，就执行statement，否则，跳过该语句。

83

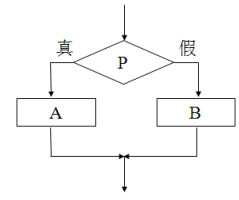
六 控制流

if条件跳转

粤嵌教育

- if语句可以跟else语句配对，其通用形式为：
  - if(expression)
    - statement_A
  - else
    - statement_B

if…else…语句执行逻辑 ➔



- 语法要点：
  - C不要求缩进排版，但以上是标准风格。缩进是的语句依赖于判断而执行这一事实显得一目了然。
  - 关键字else一定要跟if配对，否则编译出错。另外，else总是跟最近的if配对。

84

六 控制流

if条件跳转

粤嵌教育

- 多重选择else if
  - 日常生活通常会给我们多于两个以上的选择，此时我们可以用 else if 扩展 if else 结构来适应这种情况。
  - 下面是使用 else if 结构的例子：
    - if(expression_A)
      - statement_A
    - else if(expression_B)
      - statement_B
    - else if(expression_C)
      - statement_C
    - … …
    - else
      - statement_X
  - 这种结构也叫阶梯型 if 语句。

85

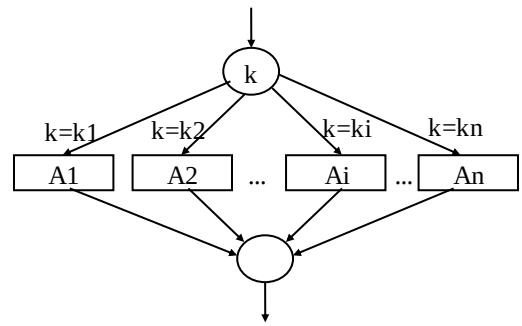
六 控制流

switch多路分支

粤嵌教育

- 使用条件运算符和 if else 结构可以很容易地编写从两个选择中进行选择的程序。然而，又是程序需要从多个选择中选择一个。可以利用 if else if…else 来这样做，但多数情况下，使用C的switch语句更加方便。

- 其一般形式为：
  - switch(expression){
    - case constant1:
      - statement_1
    - case constant2:
      - statement_2
    - ……
    - default:
      - statement_n
    - }



86

六 控制流

switch多路分支

语法要点：

- 1、switch语句中的expression表达式的值为整型。
- 2、case之后的constant为整型常量。
- 3、关键字case只起到标号的作用。
- 4、关键字default是可选的。一般情况下我们都会把default语句放到最后，但这不是必须的。

87

六 控制流

continue语句

- continue语句
  - continue结束本次循环，进入下一次循环。例如：
    - int i=0;
    - while(i < 100){
      - i++;
      - if(i%2 == 0)
        - continue; //结束本次循环
      - printf("%d\n", i);
    - }
  - 该示例将打印出100以内所有的奇数。
- 注意：continue只跟循环结构匹配。

88

六 控制流

break语句

- break语句
  - 关键字break跳出当层循环。例如：
    - int i=0;
    - while(1){
      - i++;
      - if(i > 100){
        - break;
      - }
      - printf(“%d\n”, i);
    - }
  - 该示例将打印出100以内所有的整数。
- 注意：break跟switch和循环结构匹配。

六 控制流

goto语句

- goto语句
  - 使用格式：
    - label:
      - statement
    - goto label;
  - label遵循变量命名规则，并拥有自己的命名空间。
  - label可以出现在goto之前或者之后。
  - continue和break是goto的特殊情况。
- 避免使用goto
  - 原则上，C程序根本不需要使用goto语句。因为所有使用goto的地方都可以用C语言的循环或者分支控制流来代替。

粤嵌教育

七、字符IO和输入确认

七 字符IO和输入确认

粤嵌教育

单字符IO

- IO函数将信息传输至拟定程序并从你的程序中传出信息：printf()、scanf()、getchar()、putchar()就是这样的例子。
- 下面的程序是一个非常简单的例子，该程序完成的任务就是获取从键盘输入的字符并将其发送至屏幕。

```
/* echo.c */
#include <stdio.h>
int main(void)
{
    char  ch;
    while((ch = getchar())  !=  '#')
        putchar(ch);
    return 0;
}
```

七 字符IO和输入确认

缓冲区

- 执行上面那个例子，我们输入的字符并不会立即显示在屏幕上，而是在我们输入了完整了一行以后才回显至屏幕。这是由于单字符IO函数使用了缓冲区的缘故。
- 使用缓冲区的原因：
  - 1、将若干个字符作为一个块传输比逐个发送这些字符耗费的时间少。
  - 2、如果输入有误，可以使用键盘的更正功能来修正错误。
- 另一方面，一些程序需要非缓冲输入。例如在游戏中，我们希望按一下键就执行某个动作，而不需要延迟。

93

七 字符IO和输入确认

文件和流

- 文件(file)是一块存储信息的存储器的区域。通常，文件被保存在某种永久存储器上。比如硬盘或磁带。
- C对文件的操作包含打开、读写和关闭文件等，在一个级别上，它可以使用宿主OS的基本文件工具来处理文件。这被称为低级IO。实际上C还可以以第二个级别处理文件，成为标准IO包(standard IO package)。
- 从概念上讲，C处理的是一个流而不是直接处理文件。流(stream)是一个理想化的数据流，实际输入或者输出映射到这个数据流。

94

七 字符IO和输入确认

输入确认

- 在实际情况中，程序的用户并不总是遵循指令，在程序所期望的输入与其实际获得的输入之间可能存在不匹配。这种情况能导致程序运行失败。
- 一个可能有输入错误的例子是，你写了一个处理非负整数的程序，但用户的输入不符合程序对数据的要求：
  - int n;
  - scanf(“%d”, &n);
  - while(n>=0){
    - //对n的处理
    - scanf(“%d”, &n);
  - }
- while语句中的判断条件避免了用户输入负整数的潜在错误。

95

七 字符IO和输入确认

输入确认

- 另一种情况是，用户输入了类型不匹配的数据，例如用户输入一个字符’x’而不是一个数字。
- 检测这一错误的一种方式就是检查scanf()函数的返回值。对于上面那个例子，仅当用户输入一个整数时下列表达式为真：
  - scanf(“%d”, &n) == 1
- 据此，我们改进后的代码如下：
  - int n;
  - while((scanf(“%d”, &n) == 1) && (n>=0)){
    - //对n的处理
  - }

96

七 字符IO和输入确认

输入确认

- 再者，如果用户输入了非法的字符，这些字符是不会被丢弃的，它们依然残存在缓冲区里面，我们可以用以下的语句来清空缓冲区：
  - while(getchar() != '\n');
- 据此，我们再次改进后的代码如下：
  - int n;
  - while((scanf("%d", &n) != 1) && (n>=0)){
    - //对n的处理
  - }

97

第三节强化训练

- 1、编写一个程序，要求用相应的控制流语句往屏幕打印26个小写字母。
- 2、使用嵌套循环产生下列图案：
  - $
  - $$
  - $$$
  - $$$$
  - $$$$$

98

第三节强化训练

- 3、编写一个程序，用户输入某个大写字母，产生一个金字塔图案。例如用户输入字母E，则产生如下图案：
  - A
  - ABA
  - ABCBA
  - ABCDCBA
  - ABCDEDCBA

第三节强化训练

- 4、编写一个程序，该程序读取输入直到遇到#字符，然后报告读取的空格数目、读取的换行符数目以及读取的所有其他字符数目。
- 5、编写一个程序，接受一个整数输入，然后显示所有小于或等于该数的素数。
- 6、输入一个华氏温度，要求输出摄氏温度。要求结果保留2位小数。
  - 转换公式为：c = 5(F-32)/9

第三节强化训练

粤嵌教育

- 7、编写一个程序，用户输入一个整数，要求：1)求出它是几位数；2)分别打印出每一位数字；3)按逆序打印出各位数字。
- 8、打印如下图案：

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

101

粤嵌教育

第四节

八、函数

九、字符串和字符串函数

102

粤嵌教育

八、函数

八 函数

函数概览

粤嵌教育

- 函数(function)是用于完成特定任务的程序代码的“黑盒子”。尽管C中的函数和其他语言中的函数、子程序或者子过程有相同的作用，但是在细节上会有所不同。
- 为什么使用函数？
  - 第一，函数的使用可以省去重复代码的编写。
  - 第二，即使某种功能在程序中只是用一次，将其以函数的形式实现也是有必要的，因为函数是的程序更加模块化，从而有利于程序的阅读修改和完善。
- 把函数看成黑盒子时，我们关心的是函数的功能而不是具体实现，这样就有利于把精力投入到程序的整体设计而不是其实现细节。因此，编写函数代码之前首先需要考虑的是函数的功能以及函数和程序整体上的关系。

八 函数

函数定义

- 定义一个函数的语法如下：
  - <storage type> <data tpye> <function>(parameters)
  - {
    - statement
    - return expression;
  - }
- storage type描述该函数本身的链接类型或者其返回值的易变性。
- data type描述该函数的返回值类型。
- function为函数名，parameters是函数接受的参数列表。
- statement是函数主体，return语句返回函数值。

105

八 函数

基本概念

- 1、函数的命名
  - 命名规则、命名空间
  - Linux C编程规范
- 2、返回值类型
- 3、函数原型声明
  - 声明与定义
- 4、参数列表
  - 实参和形参

```
int max(int x, inty)
{
        int max = (x>y)?x:y;
        return max;
}
```

106

八 函数

基本概念

粤嵌教育

- 函数的调用
  - 实现了函数的具体定义，我们就可以在程序中调用函数，调用函数的方法非常简单，如下：
    - int a=1, b=2, max_value;
    - max_value = max(a, b); // 调用max函数
  - 注意：C语言中对变量和函数要求必须先声明后引用，因此，以上调用max函数必须之前先对max声明：
    - max(int x, inty); // 声明max函数。变量名x和y是可选的。

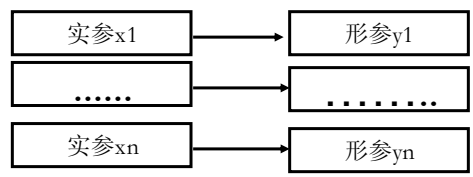
107

八 函数

传参

粤嵌教育

- 函数参数传递方式：
  - 1、赋值传递方式。
    - 赋值传递方式是函数间传递数据常用的式。简而言之就是用实参的值的一份拷贝来初始化形参，以便函数使用。
    - 赋值传递方式的示意图如下：



    - 箭头是指将实参的数据拷贝给了形参变量，实参和形参是具有相同数据类型但存储空间是不同的两组空间。

108

八 函数

传参

粤嵌教育

- 使用赋值传递方式试图交换数据的例子。
  - #include <stdio.h>
  - void swap(double x, double y) ; // swap函数声明
  - int main(void)
  - { double x1, x2;
    - x1=21.56; x2=65.12;
    - swap(x1, x2); // 调用swap函数，试图交换x1和x2的值
    - printf("x1=%.2f,x2=%.2f\n",x1,x2);
  - }
  - void swap(double x, double y) //x和y分别是x1和x2的副本
  - { double temp;
    - temp= x; x = y; y=temp;
  - }
- 运行结果(交换数据失败)：
  - x1=21.56 x2=65.12↙

八 函数

传参

粤嵌教育

- 2、地址传递方式。
  - 地址传递方式和赋值传递方式不同，这种方式是将实参的地址传给被调用函数。因此，被调用函数中对形参的操作，将直接改变实参的值。
  - 调用函数将实参的地址传送给被调用函数，被调用函数对该地址的目标操作，相当于对实参本身的操作。按地址传递,实参为变量的地址，而形参为同类型的指针。

八 函数

传参

- 使用地址传递方式的数据交换例子。
  - #include <stdio.h>
  - void swap(double *x, double *y) ; // swap函数声明
  - int main(void)
  - { double x1, x2;
    - x1=21.56; x2=65.12;
    - swap(&x1, &x2); // 调用swap函数，交换x1和x2的值
    - printf("x1=%.2f,x2=%.2f\n",x1,x2);
  - }
  - void Swap(double *x, double *y) //实现x和y指向的目标
  - { double temp;
    - temp= *x; *x = *y; *y=temp;
  - }
- 运行结果：
  - x1=65.12 x2=21.56↙

九、字符串和字符串函数

九 字符串和字符串函数

字符串

- 字符串是C里面用得最多、最重要的数据类型之一。当然，对你而言最基本的知识你已经知道了，那就是：字符串(character string)是以空字符(\0)结尾的char型数组。因此，我们可以把数组和指针的知识用到字符串上。
- 字符串常量(字符串文字)
  - 指的是位于一对双引号中的任何字符。双引号里面的字符加上编译器自动提供的字符串结束符‘\0’作为一个字符串被存储到内存中。注意，我们还可以用宏来定义字符串常量。

113

九 字符串和字符串函数

字符串

- 如果字符串常量中间没有间隔或者间隔的是空格符，ANSI C会将其串联起来：
  - char str[30] = “abc” “def” “ghijk”;
  - char str[30] = “abcdefghijk”;
- 以上两个赋值语句是等价的。

114

九 字符串和字符串函数

字符串

- 字符数组及其初始化
  - 定义一个字符数组时，你必须让编译器知道它需要多大空间。一个办法就是制定一个足够大的数组来容纳字符串，例如：
    - const char at[40] = “apple tree”;
  - 这种初始化和下面所示的标准数组初始化相比是很简洁的：
    - const char at[40] = {‘a’, ‘p’, ’p’, ’l’ ,’e’ ,’ ’, ’t’ ,’r’ ,’e’ ,’e’ , ‘\0’};
  - 注意：指定数组大小时，一定要确保数组元素数比字符串长度至少多1

115

九 字符串和字符串函数

字符串

- 字符串数组
  - 有一个字符串数组是很方便的，这样就可以使用下标来访问多个不同的字符串。例如：
    - const char *mytal[5] = {“China”, “America”, “Japan”,
    - “Russia”, “France”};
  - 这里实际上mytal是一个指针数组，这个数组存放了5个指针，他们分别是： mytal[0]、 mytal[1]…等，这些指针顺序指向了初始化列表中的各个字符串。

116

九 字符串和字符串函数

字符串

- 字符串函数
  - C库提供了许多处理字符串的函数，ANSI C用头文件string.h给出这些函数的原型。常用的字符串函数有：
    - strlen(const char *s) ➔ 计算字符串s的长度
    - strcat(char *restrict s1, const char *restrict s2)
    - ➔ 将字符串s2合并到字符串s1中
    - strcmp(const char *s1, const char *s2)
    - ➔ 比较字符串s1和s2
    - strcpy(char *restrict s1, const char *restrict s2)
    - ➔ 将字符串s2复制到s1指向的位置。

117

第四节强化训练

- 1、说明函数传参的方式和异同。
- 2、写出下面所描述的各个函数的ANSI函数头。注意：只写出函数头即可，不需要实现。
  - a) donut()接受一个int类型的参数，然后输出若干个0，输出0的数目等于参数的值。
  - b) gear()接受两个int类型的参数并返回int类型的值。
  - c) stuff_it()的参数包括一个double类型的值以及一个double类型变量的地址，功能是把第一个数值存放到指定的地址中。
- 3、编写一个函数，使其返回3个整型参数中的最大值。

118

第四节强化训练

- 4、编写一个函数my_power，用循环的方法实现返回一个float类型数的某个整数次幂(保留6位小数)。如调用my_power(3.14, -2)返回0.101424
- 5、编写一个函数，判断一个整数是否为素数。
- 6、编写一个程序，将两个字符串连接起来，不要用strcat或strncat函数。

119

第四节强化训练

- 7、编写一个函数Fibonacci()，要求程序输出第n项斐波那契数，n由用户输入。
  - 斐波那契数列：1, 1, 2, 3, 5, 8, 13, 21, …….
- 8、编写一个函数，求两个整数的最大公约数，用主函数调用这个函数并输出结果，两个整数由键盘输入。（提示：用辗转相除法求最大公约数）
- 9、编写一个程序，清除用户输入字符串中的空格符并将之输出。（例如用户输入”a b”，输出”ab”）

120

粤嵌教育

第五节

十、数组与指针（1）

121

粤嵌教育

十、数组与指针（1）

122

十 数组与指针（1）

基本概念

- 概念：数组(array)是由一系列相同类型的元素构成的复合数据类型。
- 定义一个数组的语法如下：
  - <存储类型> <数据类型> 数组名[元素个数];
  - static int array[10]; /* array是一个一维数组 */
- 注意：
  - 1、存储和数据类型说明的都是元素的属性。
  - 2、数组名遵循变量命名规则

123

十 数组与指针（1）

一维数组

- 一维数组
  - 所谓一维数组是指只有一个下标的数组。它在计算机的内存中是连续存储的。
- 例如：
  - char a[10];
  - 此语句声明了一个拥有10个元素的数组a，并且数组里面的每个元素a[0], a[1], a[2], … , a[9]都是char型的数据。
  - 在内存中它们是连续存放的，下标从0开始。

124

十 数组与指针（1）

一维数组

- 一维数组的初始化
  - 1、对所有元素赋值，例如：
    - int i[3] = {32, 1, 1024};
    - int j[ ] = {1, 1, 2, 3, 5, 8, 13};
    - char a[10] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’};
    - char a[10] = “hello”;
  - 2、对一部分元素赋值，例如：
    - int i[10] = {32, 1, 1024};
    - char a[10] = {‘A’, ‘B’, ‘C’, ‘D’, ‘E’};

125

十 数组与指针（1）

一维数组

- 一维数组的初始化
  - C99增加了一种新特性：指定初始化项目(designated initializer)。此特性允许选择对某些元素初始化。C99规定，在初始化列表中使用带有方括号的元素下标可以指定某个特定的元素，例如：
    - int arr[10] = {[3]=100, [9]=200};
  - 其他没有被初始化的元素，其具体的值与其存储类型相关，具体来说是，如果是该数组是静态数组，则其余元素将被初始化为0，如果是局部数组，则其余元素的值将取决于当其时系统的栈空间数据。

126

十 数组与指针（1）

一维数组

粤嵌教育

- 一维数组的初始化
  - 另一个复杂一点儿的例子：
    - #define MONTHS 12
    - int days[MONTHS] = {31, 28, [4]=31,30,31, [1]=29};
  - 如果编译器支持C99特性，将此数组的个元素打印出来的结果是：31,29,0,0,31,30,31,0,0,0,0,0
  - 从这个例子可知：
    - 1、如果在一个指定初始化项目后跟有不值一个值，则这些值将用来对后续的数组元素初始化。
    - 2、如果多次对一个元素进行初始化，则最后一个有效。

127

十 数组与指针（1）

一维数组

粤嵌教育

- 数组的赋值
  - 声明完数组之后，可以借助数组的索引(即下标)对数组成员进行赋值。例如：
    - for(i=0; i<MONTHS; ++i)
      - days[i] = 100;
  - 注意，这种赋值的方式是使用循环对元素逐个赋值，C不支持把数组作为一个整体来进行赋值，也不指出用花括号括起来的列表形式进行赋值(初始化的时候除外)。

128

十 数组与指针（1）

一维数组

- 数组的边界
  - 使用数组的时候，需要注意数组的索引不能超过数组的边界。也就是说，数组索引应该具有对于数组来说有效的值。假定你有这样的声明：
    - int doofi[20];
  - 那么你在使用数组索引的时候，要确保它的范围在0到19之间，因为编译器不会为你检查出这种错误。换句话说，对数组边界的确定，这个责任落程序编写者自己的肩上，如果不注意，将导致运行时错误。

129

十 数组与指针（1）

一维数组

- 数组的边界
  - 编译器不检查索引的合法性，因此如果在程序中如果真的那么做了，那么程序的执行结果是不可知的。也就是说，程序也许能够运行，但是运行的结果可能很奇怪，也可能会异常中断程序的执行。
  - C之所以会允许这种事情的发生，是出于C信任程序员的原则。不检查边界能够让C程序的运行速度更快。在程序运行之前，索引的值有可能尚未确定下来，所以编译器此时不能找出所有的索引错误。
  - 一件需要记住的简单的事情是：一个具有N个元素的数组的索引从0开始，到N-1为止。

130

十 数组与指针（1）

多维数组

- 多维数组
  - 概念：具有两个或两个以上下标的数组称为多维数组。
  - 定义一个多维数组的语法如下：
    - <存储类型> <数据类型> 数组名[表达式1][表达式2]…[表达式n];
    - 例如：
    - int i[3][5]; /* 定义了一个二维整型数组 */
    - double d[1][2][3]; /* 定义了一个三维浮点型数组 */

131

十 数组与指针（1）

多维数组

- 多维数组的初始化
  - 与一维数组类似，可以对多维数组的所有元素进行初始化，也可以对部分元素初始化。
    - int i[2][3] = {{1, 2, 3}, {4, 5, 6}};
    - int a[][3]= {{1,2,3}, {3,4}, {1,2,3}};
  - 对于多维数组，第一个方括号内的元素个数可以省略，但是子数组的元素个数不能省略(例如上面的例子中a[ ][3]不能写成a[ ][ ])。未被赋值的元素被初始化为0.

132

十 数组与指针（1）

指针基本概念

粤嵌教育

- 指针变量
  - 概念：储存内容为地址的变量，称之为指针变量，简称指针。
  - 定义指针变量的语法如下：
    - <存储类型> <数据类型> *指针变量名;
    - 例如：
    - int *pi;
    - static char *pc=NULL;
    - char *pstr[10];

133

十 数组与指针（1）

指针基本概念

粤嵌教育

- 指针指向的内存区域中的数据称为指针的目标。如果它指向的区域是程序中的一个变量的内存空间， 则这个变量称为指针的目标变量。 指针的目标变量简称为指针的目标。

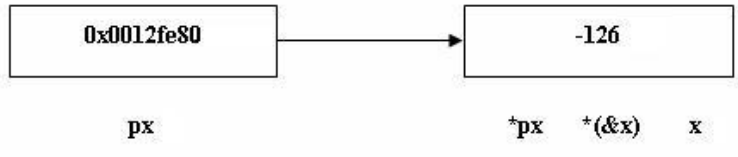


图 4.2 px 的目标变量等价 x

134

十 数组与指针（1）

指针运算

- 指针运算操作
  - int i=1, j=2, *pi, *pj;
  - 赋值(assignment)： pi = &i; pj = &j;
  - 取值(dereferencing)： i = *pj;
  - 取指针地址： &p;
  - 加法： p+1; p++; pi + pj;
  - 减法： p-1; p--; pi – pj;
  - 比较： if(pi > pj)

135

十 数组与指针（1）

const型指针

- const型指针
  - const int *p; 或者 int const *p;
    - p为指向整型变量指针，且该变量相对于p而言只读。
  - int *const p;
    - p为指向整型变量指针，且该指针p本身只读。
  - const int *const p; 或者 int const *const p;
    - p为指向整型变量只读指针，且该变量相对于p而言也是只读的。

136

十 数组与指针（1）

各种与数组和指针相关的类型

- 数组指针和指针数组
  - 数组指针：指向数组的指针，例如：int (*p)[10];
  - 指针数组：包含指针的数组，例如：int *p[10];
- 函数指针和指针函数
  - 函数指针：指向函数的指针，例如：int (*p)(int);
  - 指针函数：返回指针的函数，例如：int *p(int);
- 函数指针数组
  - 由函数指针构成的数组，例如：int (*p[10])(int);

137

十 数组与指针（1）

最后的叮嘱

- 使用指针，有一个规则需要特别注意：不能对未初始化的指针取值。
  - int *pt; // 未初始化的指针
  - *pt = 5; //一个可怕的错误！
- 为什么这样的代码危害极大？这段程序的第二行表示把数值5存储在pt所指向的地址。但是由于pt没有被初始化，因此它的值是随机的，不知道5会被存储到什么位置。这个位置也许对系统危害不大，但也许会覆盖程序数据或者代码，甚至导致程序的崩溃。
- 切记：当创建一个指针时，系统只分配了用来存储指针本身的内存空间，并不分配用来存储数据的内容空间。因此在使用指针之前比如给他赋予一个已分配的合法内存地址。

138

十 数组与指针（1）

最后的叮嘱

- 给定下面的声明：
  - int urn[3];
  - int *ptr1, *ptr2;
- 下表是一些合法的和非法的语句：

合 法	非 法
ptr1++;	urn++;
ptr2 = ptr1 + 2;	ptr2 = ptr2 + ptr1;
ptr2 = urn + 1;	ptr2 = urn * ptr1;

139

第五节强化训练

- 1、假如有如下定义：
  - int grid[30][100];
  - a. 用1种方法表示grid[22][56]的地址。
  - b. 用2种方法表示grid[22][0]的地址。
  - c. 用3种方法表示grid[0][0]的地址。
- 2、编写一个函数，返回一个double型数组中最大和最小值的差值，并在一个简单的程序中测试这个函数。

140

第五节强化训练

- 3、用变量a给出下面的定义
  - a) 一个整型数
  - b) 一个指向整型数的指针
  - c) 一个指向指针的的指针，它指向的指针是指向一个整型
  - d) 一个有10个整型数的数组
  - e) 一个有10个指针的数组，该指针是指向一个整型数的
  - f) 一个指向有10个整型数数组的指针
  - g) 一个指向函数的指针，该函数有一个整型参数并返回一个整型数
  - h) 一个有10个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数

141

第五节强化训练

- 4、下面的程序将打印出什么？解释原因
- #include <stdio.h>
- int main(void)
- {
  - int ref[ ] = {8, 4, 0, 2};
  - int *ptr;
  - int index;
  - for(index = 0, ptr = ref; index<4; index++, ptr++)
    - printf(“%d %d\n”, ref[index], *ptr);
  - return 0;
- }

142

第五节强化训练

- 5、在第4题中，ref是哪些数据的地址？ref+1呢？++ref指向什么？
- 6、下面每种情况中*ptr和*(ptr+2)的值分别是什么？
  - a)
    - int *ptr;
    - int torf[2][2] = {12, 14, 16};
    - ptr = torf[0];
  - b)
    - int *ptr;
    - int fort[2][2] = {{12}, {14, 16}};
    - ptr = fort[0];

143

第五节强化训练

- 7、编写一个函数，把两个数组内的相应元素相加，结果存储到第3个数组内。也就是说，如果数组1具有值2, 4, 6, 8，数组2具有值1, 0, 3, 6，则函数对数组3赋值为3, 4, 9, 14。（该函数的参数包括3个数组名和数组大小）
- 8、假设有如下声明：
  - float apple[10], apple_tree[10][5], *pf, weight = 2.2;
  - int i = 3;

144

第五节强化训练

- 则下列语句中那些是正确的，哪些是错误的？
  - a. apple[2] = weight;
  - b. scanf(“%f”, &apple);
  - c. apple = weight;
  - d. printf(“%f”, apple[3]);
  - e. apple_tree[4][4] = apple[3];
  - f. apple_tree[5] = apple;
  - g. pf = weight;
  - h. pf = apple;

145

第六节

十一、数组与指针（2）

146

十一、数组与指针（2）

147

十一 数组与指针（2）

函数、数组和指针

- 假设有一个数组int ar[10]，要定义一个处理该数组的函数，用来计算数组元素之和，则一个合乎情理的调用如下：
  - total = sum(ar);
- 那这个函数该怎么定义呢？由于数组就是首元素的地址，因此有以下两种等价的方法：
  - int sum(int ar[10]){ … … }
  - int sum(int *ar){ … … }
- 注意：当且仅当在函数参数列表中时，数组和指针两者才等价。

148

十一 数组与指针（2）

变长数组VLA

- C99标准引入了变长数组(VLA)，它允许使用变量定义数组各维。例如你可以使用下面的声明：
  - int quarters = 4;
  - int regions = 5;
  - double sales[quarters][regions]; // 一个二维VLA
- VLA有一些限制，它必须是自动存储类的，这意味着他们必须在函数内部或者作为函数参量声明，而且声明是不可以进行初始化。

149

十一 数组与指针（2）

变长数组VLA

- 下面是一个使用VLA的简单示例：
  - int thing[10][6];
  - twoset(10, 6, thing);
  - … …
  - void twoset(int n, int m, int ar[n][m])
  - {
    - int temp[n][m]; temp[0][0] = 2; ar[0][0] = 2;
  - }
- VLA允许动态分配存储单元。这表示可以在程序运行时指定数组的大小。常规的C数组时静态存储分配的，也就是说数组大小在编译时已经确定。这是因为数组大小是常量，所以编译器可以得到这些信息。

150

十一 数组与指针（2）

多级指针

- 多级指针
  - 如果一个指针的内容是另一个指针的地址，则该指针为多级指针，例如：
    - int i = 100; int *pi = &i; int **ppi = π
  - 在这里，pi是一个一级指针，它的内容就是变量 i 的地址，而ppi是一个二级指针，它的内容是变量pi的地址。我们可以通过不同的方法来访问变量i：
    - i = 200;
    - *pi = 200;
    - **ppi = 200;

151

十一 数组与指针（2）

指针与数组

- 数组与指针的关系
  - a、&a和&a[0]的异同
    - 数组名称的本质 ➔ 内存的别名
  - 数组不能整体操作
    - 变量赋值的本质 ➔ 左值和右值
  - 数组下标操作符的内部实现机制
    - a[2] 相当于 *(a+2)
  - 作为函数参数传递时的数组和指针
    - int main(int argc, char **argv);

152

十一 数组与指针（2）

指针与数组

- 直接引用和间接引用
  - 考虑以下两个声明语句：
    - extern int *p;
    - extern int p[ ];
  - 这两个声明语句完全不同，前者声明了一个指针变量，后者声明了一个int型数组，长度尚未确定(不完整的类型)，其存储在别处定义。
  - 在一个文件里面定义一个数组或指针，不能指望在另一个文件中将其声明为指针或数组来使用它，因为指针时间接引用，而数组时直接引用。

153

十一 数组与指针（2）

指针与数组

- 什么时候数组和指针是不同的

指针	数组
保存数据的地址	保存数据
间接访问数据，首先取得指针的内容，把它作为地址，然后从这个地址提取数据。如果指针有一个下标[i]，就把指针的内容加上i作为地址，从中提取数据。	直接访问数据，a[i]只是简单地以a+i为地址取得数据。
通常用于动态数据结构	通常用于存储固定数目且数据类型相同的元素
相关的函数为malloc()，free()	隐式分配和删除
通常指向匿名数据	自身即为数据名

154

十一 数组与指针（2）

指针与数组

- 什么时候数组和指针是相同的
  - 规则1：“表达式中的数组名”就是指针
  - 规则2：C语言把数组下标作为指针的偏移量
  - 规则3：“作为函数参数的数组名”等同于指针
- 简而言之，数组和指针的关系颇有点儿像诗和词的关系：它们都是文学形式之一，有不少共同之处，但在实际的表现手法上又各有特色。

155

第六节强化训练

- 1、以下代码中的两个sizeof用法有问题吗？

```
void upper_case(char str[ ])
{
    int i;
    for(i = 0; i < sizeof(str) / sizeof(str[0]); i++){
        if(str[i] > ‘a’ && str[i] < ‘z’) str[i] -= (‘a’ - ‘A’);
    }
}
int main(void)
{
    char str[ ] = “aBcDe”;
    printf(“length of the string: %d\n”, sizeof(str) / sizeof(str[0]));
    upper_case(str);
}
```

156

第六节强化训练

- 2、在x86平台下，分析以下代码的输出结果：
  - #include <stdio.h>
  - int main(void)
  - {
    - int a[4] = {1, 2, 3, 4};
    - int *ptr1=(int *)(&a+1);
    - int *ptr2=(int *)((int)a+1);
    - printf("%x, %x\n", ptr1[-1], *ptr2);
    - return 0;
  - }

157

第六节强化训练

- 3、声明一个二维int型数组arr，再声明另一个一维数组指针数组，使该数组的每一个指针分别指向二维数组中的每一个元素(即每一个一维数组)，然后利用数组p2arr计算数组arr的和。

- 4、已知等待接合线程的函数原型如下：
  - int pthread_join(pthread_t tid, void **value_ptr);
- 试解释如下函数调用：
  - void *ret;
  - pthread_join(tid, &ret);
  - printf("ret: %d\n", *((int *)&ret));

158

第六节强化训练

- 5、一个有N个元素的整型数组，求该数组的各个子数组中，子数组之和的最大值是多少？
  - 例如数组a[6] = {-2, 5, 3, -6, 4, -8, 6};
  - 则子数组之和的最大值是8 (即a[1] + a[2])。
- 6、编写一个程序，初始化一个3x5的二维double型数组，并利用一个基于变长数组的函数把该函数赋值到另一个二维数组，另外再写一个基于变长数组的函数来显示两个数组的内容。这两个函数应该能够处理任意的NxM数组。

159

第六节强化训练

- 7、编写一个程序，去掉给定字符串中重复的字符。例如给定”google”，输出”gole”。（华为笔试题）

160

粤嵌教育

第七节

十二、存储类，链接和内存管理
十三、LINUX C内存映像

161

粤嵌教育

十二、存储类，链接和内存管理

162

十二 存储类，链接和内存管理

基本概念

- 存储类
  - C为变量提供了5种不同的存储模型，或者称为存储类，分别是：
    - 自动存储： auto
    - 静态存储 ： static
    - 寄存器存储 ： register
    - 外部存储： extern
    - 关键字： typedef
  - 各种存储类可以从不同的角度来描述，分别是：
    - 作用域
    - 链接类型
    - 存储期

十二 存储类，链接和内存管理

作用域

- 作用域(scope)
  - 概念：作用域描述了程序中合法访问一个标识符的区域。
  - 一个C变量的作用域可以是：
    - 代码块作用域(block scope)
    - 函数原型作用域(function prototype scope)
    - 文件作用域(file scope)

十二 存储类，链接和内存管理

链接

- 链接(linkage)
  - 概念：跟作用域类似，变量的链接是一个空间概念，描述了程序中合法访问一个标识符的区域。
  - 一个C变量的链接类型可以是：
    - 外部链接(external linkage)
    - 内部链接(internal linkage)
    - 空链接(no linkage)

165

十二 存储类，链接和内存管理

存储期

- 存储期(storage duration)
  - 概念：变量的声明周期，描述了一个C变量在程序执行期间的存在时间。
  - 一个C变量的存储期可以是：
    - 静态存储期(static storage duration)
    - ➔全局变量、static型局部变量
    - 自动存储期(automatic storage duration)
    - ➔局部变量（包括函数形参）
    - 动态存储器(dynamic storage duration)
    - ➔存储于由malloc等函数分配的堆内存里的变量

166

十二 存储类，链接和内存管理

- 存储类说明符：
  - 1、auto：声明一个自动变量
  - 2、static：声明一个静态变量，或声明一个内部链接函数和全局变量
  - 3、register：声明一个寄存器存储类变量
  - 4、extern：声明一个外部存储变量
  - 5、typedef：语法意义上的存储类，与实际存储类型无关。

十二 存储类，链接和内存管理

- C语言用作用域、链接和存储期来定义5种存储类：

存储类	时期	作用域	链接	声明方式
自动	自动	代码块	空	代码块内
寄存器	自动	代码块	空	代码块内 Register
具有外部链接的静态	静态	文件	外部	所有函数之外
具有内部链接的静态	静态	文件	内部	所有函数之外 static
空链接的静态	静态	代码块	空	代码块之内 static

十二 存储类，链接和内存管理

粤嵌教育

- 各种存储类型和特性总结如下：

	局部变量			全局变量	
存储类别	auto	register	局部static	外部static	外部
存储方式	动态		静态		
存储区	动态区	寄存器	静态存储区		
生存期	函数调用开始至结束		程序整个运行期间		
作用域	定义变量的函数或复合语句内			本文件	其他文件
赋初值	每次函数调用时		编译时赋初值，只赋一次		
未赋初值	不确定		自动赋初值为零值或NULL		

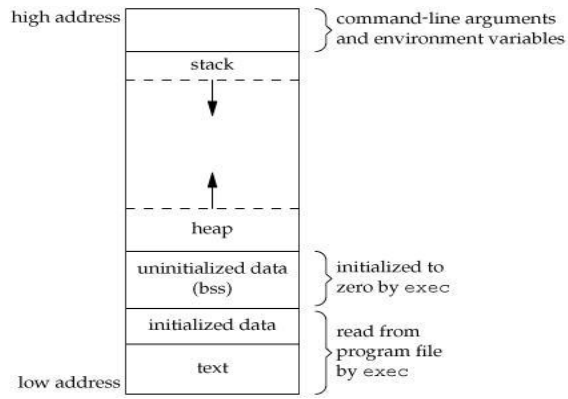
169

十二 存储类，链接和内存管理

内存管理

粤嵌教育

- 进程内存映像



170

十二 存储类，链接和内存管理

内存管理

- 静态存储分配
  - 通常定义变量，编译器在编译时都可以根据该变量的类型知道所需内存空间的大小，从而系统在适当的时候为他们分配确定的存储空间。
  - 在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。

171

十二 存储类，链接和内存管理

内存管理

- 动态存储分配
  - 有些操作对象只有在程序运行时才能确定，这样编译器在编译时就无法为他们预定存储空间，只能在程序运行时，系统根据运行时的要求进行内存分配，这种方法称为动态存储分配。
  - 所有动态存储分配都在堆区中进行。
  - 从堆上分配，亦称动态内存分配。程序在运行的时候用malloc或new申请任意多少的内存，程序员自己负责在何时用free或delete释放内存。动态内存的生存期由我们决定，使用非常灵活，但问题也最多。

172

十二 存储类，链接和内存管理

内存管理

- 堆内存的分配与释放
  - 当程序运行到需要一个动态分配的变量或对象时，必须向系统申请取得堆中的一块所需大小的存贮空间，用于存贮该变量或对象。当不再使用该变量或对象时，也就是它的声明结束时，要显式释放它所占用的存贮空间，这样系统就能对该堆空间进行再次分配，做到重复使用有限的资源。
  - 堆区是不会自动在分配时做初始化的（包括清零），所以必须用初始化式(initializer)来显式初始化。

十二 存储类，链接和内存管理

内存管理

- malloc和free函数原型如下：
  - void *malloc(size_t num);
  - void free(void *p);
- malloc函数本身并不识别要申请的内存是什么类型，它只关心内存的总字节数。
- malloc申请到的是一块连续的内存，有时可能会比所申请的空间大。其有时会申请不到内存，返回NULL。
- malloc返回值的类型是void*，所以在调用malloc时要显式地进行类型转换，将void*转换成所需要的指针类型。
- 如果 free的参数是NULL的话，没有任何效果。
- 如果释放一块内存中的一部分是不被允许的。

十二 存储类，链接和内存管理

内存管理

- malloc函数族的其他成员：
  - #include <stdlib.h>
  - void *calloc(size_t nelem, size_t elsize);
  - void *realloc(void *ptr, size_t size);
- calloc函数为进程动态分配一块内存，nelem为指定对象数量，elsize为对象的指定大小，并把该空间中的每一位都初始化为零。
- realloc函数用以更改以前分配区的长度。当增加长度时，可能须将以前分配区的内容移到另一个足够大的区域。
- 这三个分配函数所返回的指针一定是适当对齐的，使其可用于任何数据对象。

175

十二 存储类，链接和内存管理

内存管理

- 注意事项：
  - 函数free究竟干了什么？
  - malloc究竟返回什么？
  - 防止内存泄露(memory leakage)。
    - 动态分配的内存的声明期。
  - 防止悬垂指针(dangling pointer)。产生的原因有：
    - 1、没有初始化。
    - 2、释放了对应的内存。
    - 3、指针操作超出了合法范围。

176

粤嵌教育

十三、 LINUX C内存映像

十三 Linux C内存映像

内存布局

粤嵌教育

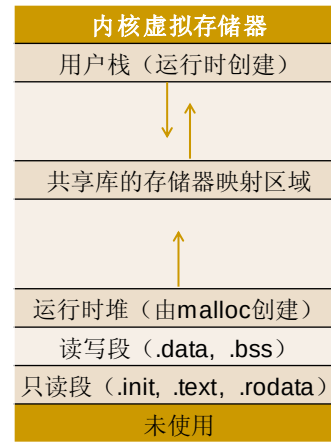
- 每个Linux程序都有一个运行时存储器映像。代码段总是从地址0x08048000开始，而数据段在接下来的下一个4K对齐的地址上，运行时堆在接下来的读写段之后的第一个4K对齐的地址处，并通过调用malloc函数族向上增长。
- 用户栈总是从地址0xbfffffff处开始向下增长，从栈的上部开始于地址0xc0000000处的段是为OS驻留存储器部分的代码和数据保留的。

十三 Linux C内存映像

内存布局

粤嵌教育

- 当加载器运行时，它创建如右图所示的存储器映像。其中的数据段和代码段从可执行文件中加载。
- 堆和栈是在程序运行时创建的。.data段和.bss段分别存放已初始化和未初始化的静态变量，.init段是系统启动代码，.text是程序的可执行代码，.rodata存放程序的常量。

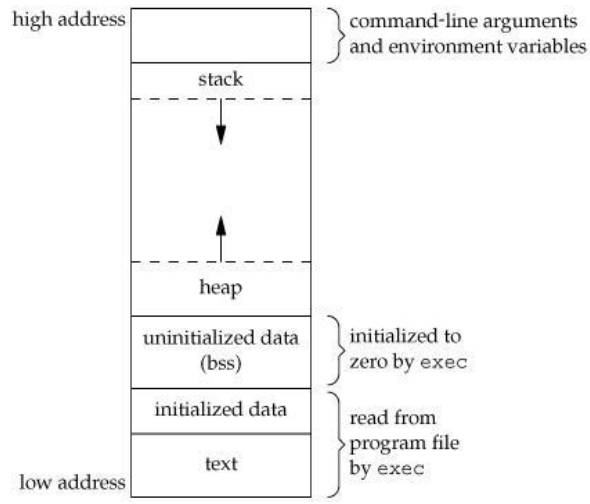


179

十三 Linux C内存映像

内存布局

粤嵌教育



180

第七节强化训练

- 1、哪一存储类的变量在包含他们的程序运行时期内一直存在？哪一存储类的变量可以在多个文件中使用？哪一存储类的变量只限于在一个文件中使用？
- 2、代码块作用域变量具有哪种链接类型？
- 3、说出C程序中所有不同的存储类变量在内存中的详细分布情况。
- 4、编写一个函数，它返回函数自身被调用的次数，并在一个循环中测试之。

181

第七节强化训练

- 5、分析以下代码的输出结果并解释原因。

```
char *get_memory(void)
{
    char p[ ] = "hello world";
    return p;
}
void Test(void)
{
    char *str = NULL;
    str = get_memory();
    printf("%s\n", str);
}
```

182

第七节强化训练

- 6、指出以下代码第二次输出结果，解释原因。
  - int main(void)
  - {

```
extern  int  a; int  b=0; static int  c;
a += 3; other();   b += 3; other();
```

  - }
  - int a=5;
  - void other(void)
  - {

```
int  b=3;static  int  c=2;
a += 5; b += 5; c += 5;
printf("%d, %d, %d\n", a, b, c);   c=b;
```

  - }

183

第八节

十四、复杂声明

十五、结构体、共用体和枚举

184

粤嵌教育

十四、 复杂声明

十四 复杂声明

粤嵌教育

基本概念

- 声明(declaration)与定义(definition)
  - 为了使不同的文件都可以访问同一个变量，C会区分变量的声明和定义。
  - 变量的定义会为这个变量分配存储空间，并且可能会为其指定一个初始化的值，一个变量的定义有且仅有一处。定义实际上是一种特殊的声明。
  - 变量的声明会将变量的类型和名称传达给程序。当然。我们可以通过使用“extern”关键字来声明一个变量，而不用定义它。声明的形式就是在变量的名字和类型前面，加上关键字“extern”。

十四 复杂声明

typedef

粤嵌教育

- typedef关键字可以为一种类型引入新的名字，而不是为变量分配空间，如：
  - typedef int *point2int;
  - 以上语句为整型指针引入了一个别名point2int，可以用它来定义新的变量。
- 一般情况下，typedef用于简洁地表示指向其他东西的指针，signal函数是典型的例子。
- signal函数的POSIX.1原型声明：
  - typedef void (*sighandler_t)(int);
  - sighandler_t signal(int sig, sighandler_t handler);

187

十四 复杂声明

typedef

粤嵌教育

- typedef int *apple 与 #define peach int *
  - typedef是一种彻底的封装类型——在声明它之后不能再往里面增加别的东西。它和宏的区别体现在两个方面：
  - 1、不能用其他类型说明符进行扩展。
    - unsigend apple j; //这是非法的！
    - unsigned peach i; // 没问题，i是指向无符号整型变量的指针
  - 2、在连续几个变量的 声明中，#define定义的类型不能保证所有的变量为同种类型。
    - apple a, b; // a和b都是指向整型变量的指针
    - peach c, d; // c是整型变量指针，d是整型变量

188

粤嵌教育

十五、结构体，共用体和枚举类型

189

十五 结构体、共用体和枚举

粤嵌教育

结构体

- 概念：结构体(structure)是由各种类型组成的的复合数据类型。
- 定义一个结构体类型的语法如下：
  - struct <结构体标签名称> {成员列表};
- 定义一个结构体变量的3种形式：
  - 常规形式：
    - struct <结构体标签名称> {成员列表};
    - struct <结构体标签名称> <结构体变量名称>;
  - 与类型定义同时进行：
    - struct <结构体标签名称> {成员列表}<结构体变量名>;
  - 直接定义：
    - struct {成员列表}<结构体变量名>;

190

十五 结构体、共用体和枚举

结构体

- 先定义一个结构体类型：
  - struct student
  - {
    - char name[32];
    - int age;
    - char gender;
    - double score;
  - };
- 再定义一个结构体变量：
  - struct student Bill, Michael;

191

十五 结构体、共用体和枚举

结构体

- 同时定义一个结构体类型和该类型的结构体变量：
  - struct student
  - {
    - char name[32];
    - int age;
    - char gender;
    - double score;
  - }Phoebe, Michael;

192

十五 结构体、共用体和枚举

结构体

- 直接定义一个结构体类型和该类型的结构体变量：
  - struct //没有结构体标签
  - {
    - char name[32];
    - int age;
    - char gender;
    - float score;
  - }Phoebe, Michael;

193

十五 结构体、共用体和枚举

结构体

- 结构体的初始化
  - 1、与普通的变量一样，在定义结构体变量的时候对其进行初始化：
    - struct student Phoebe={“Phoebe”, 17, ‘F’, 92.5};
  - 上述初始化方式与以下方式等价：
    - struct student Phoebe;
    - strcpy(Phoebe.name, “Phoebe”);
    - Phoebe.age = 17;
    - Phoebe.gender = ‘F’;
    - Phoebe.score = 92.5;

194

十五 结构体、共用体和枚举

结构体

- 结构体的初始化
  - 2、在定义结构体类型的同时进行结构体变量初始化：
    - struct student
    - {
      - char name[32];
      - int age;
      - char gender;
      - double score;
    - }Phoebe = {“Phoebe”, 17, ‘F’, 92.5};

十五 结构体、共用体和枚举

结构体

- 结构体的初始化
  - 3、结构体变量部分初始化：
    - struct student
    - {
      - char name[32];
      - int age;
      - char gender;
      - double score;
    - }Phoebe = {.name=“Phoebe”, .age=17};
    - struct student Michael = {.age=25, .score=88.5};

十五 结构体、共用体和枚举

结构体

- 结构体数组
  - 一个班里的所有学生，可以用之前的student结构体来构成一个结构体数组：
    - struct student this_class[30]; //30个student结构体
  - 用点运算符引用成员变量：this_class[0].age = 23;
- 结构体指针
  - 结构体指针也就是指向结构体的指针，例如：
    - struct student *pstudent = &Bill; //指向结构体Bill的指针
  - 用点运算符引用成员变量： (*pstudent).age = 27;
  - 用箭头引用成员变量：pstudent->age = 27;

197

十五 结构体、共用体和枚举

共用体

- 共用体(union)是一个能在同一个存储空间里(但不同时)存储不同类型数据的复合数据类型。
- 定义一个共用体类型的语法如下：
  - union <共用体标签名字> {成员列表};
  - 定义类型和变量示例如下：
    - union foo /* 定义一个共用体类型foo */
    - {
      - int digit;
      - double bigfl[10];
      - char letter;
    - }baz; /* 定义一个example类型的共用体变量baz */

198

十五 结构体、共用体和枚举

共用体

- 跟结构体一样，定义一个共用体变量也有如下3种形式：
  - 常规形式：
    - union <共用体标签名称>{成员列表};
    - union <共用体标签名称><共用体变量名称>;
  - 与类型定义同时进行：
    - union <共用体标签名称>{成员列表}<共用体变量名>;
  - 直接定义：
    - union {成员列表}<共用体变量名>;

199

十五 结构体、共用体和枚举

枚举类型

- 枚举类型(enumerated type)可以用来声明代表整数常量的符号名称。
- 定义一个枚举类型的语法如下：
  - enum <枚举标签名字>{枚举常量};
  - 定义类型和变量示例如下：
    - enum spectrum{ /* 定义一个枚举类型spectrum */
      - red=100,
      - blue,
      - green=99,
      - yellow
    - }color; /* 定义一个spectrum类型的枚举变量color */

200

十五 结构体、共用体和枚举

枚举类型

- 枚举常量
  - 枚举常量是int型的常量，在使用int类型的任何地方都可以使用它。
- 默认值
  - 没有特定指出常量值时，枚举列表中的常量被指定为整数值0、1、2等，依次递增。
- 指定值
  - 可以选择常量具有的整数值，后面的常量会被赋予后续的值。

201

第八节强化训练

- 1、指出以下代码片段的不妥之处？
  - structure{
    - char itable;
    - int num[20];
    - char * togs
  - }
- 2、设计一个结构模板，保存一个月份名、一个3个字母的该月份的缩写、该月的天数，以及月份号。

202

第八节强化训练

- 3、分析以下结构所占的存储空间大小：
  - struct animals{
    - char dog;
    - unsigned long cat;
    - unsigned short pig;
    - char fox;
  - };
- 4、定义一个结构体变量(包括年月日)。计算该日在本年中是第几天？注意闰年问题。

203

第八节强化训练

- 5、假设有以下结构：
  - struct gas {
    - float distance;
    - float gals;
    - float mpg; // mpg = distance * gals
  - };
- a) 设计一个函数，它接受一个struct gas参数。假定传递进来的结构包括distance和gals信息。函数为mpg成员正确计算初值并返回这个完整的结构。
- b)设计一个函数，它接受一个struct gas参数的地址。假定传递进来的结构包括distance和gals信息。函数为mpg成员正确计算初值并把它赋给恰当的成员。

204

第八节强化训练

- 6、声明一个枚举类型，使用choices作为标记，将枚举常量no、yes和maybe分别设置为0、1和2。
- 7、声明4个函数，并把一个指针数组初始化为指向它们。每个函数接受两个double参数并返回一个double值。

205

第八节强化训练

- 8、假设有以下说明和定义：
  - typedef union
  - {long i; int k[5]; char c;
  - } fruit;
  - struct creature
  - {int cat; fruit apple; double dog;
  - };
  - fruit berry;
- 则语句
  - printf("%d", sizeof(struct creature)+sizeof(berry));
- 的执行结果是？

206

第八节强化训练

- 9、编写一个transform()函数，它接受4个参数：包含double类型数据的源数组名，double类型的目标数组名，表示数组元素个数的int变量以及一个函数名(或者等价的指向函数的指针)。transform()函数把指定的函数作用于源数组的每个元素，并将返回值放到目标数组中。例如：
  - transform(source, target, 100, sin);
- 这个函数调用sin(source[0])赋给target[0]，等等。共有100个元素。在一个程序中测试该函数，调用4次transform()，分别使用math.h函数库中的两个函数以及自己设计的两个适合的函数作为参数。

207

第九节

十六、高级议题

208

粤嵌教育

十六、高级议题

209

十六 高级议题

预处理（Preprocessing）

粤嵌教育

- 所谓的预处理就是进行编译的第一遍扫描(词法扫描和语法分析)之前所做的工作。
- C提供的预处理功能有：
  - 宏定义：#define
  - 文件包含：#include
  - 条件编译：#if、#ifdef 、#ifndef 、#else、#endif

210

十六 高级议题

预处理（Preprocessing）

- 宏定义
  - 1、预定义符号
    - __FILE__ 正在编译的文件名
    - __LINE__ 文件当前的行号
    - __FUNCTION__ 当前所在的函数名
    - __DATE__ 预编译文件的日期
    - __TIME__ 预编译文件的时间
    - __STDC__ 判断编译器是否遵循ANSI C，是则为1
  - 其中， __LINE__和__STDC__是整型常量，其余为字符串常量。

211

十六 高级议题

预处理（Preprocessing）

- 宏定义
  - 2、不带参数的宏定义。其定义的一般形式是：
    - #define 标识符 字符串
    - 其中：
    - #是预处理命令标记。
    - define是宏定义命令。
    - 标识符即该宏定义的宏名。
    - 字符串可以是常量、表达式、格式串等。
  - 宏定义只是简单的替换。
  - 宏定义不是声明或语句，行末不必加分号。

212

十六 高级议题

预处理（Preprocessing）

- 宏定义
  - 3、带参数的宏定义。其定义的一般形式是：
    - #define 宏名(参数表) 字符串
    - 例如：
    - #define MAX(a, b) ((a>b)?(a):(b))
  - 宏名与参数表之间没有空格。
  - 形参不分配内存单元。
  - 宏定义的形参可以是标识符，但宏调用的实参可以是表达式。
  - 字符串中的参数应该用括号括起来。

213

十六 高级议题

预处理（Preprocessing）

- 文件包含
  - 文件包含是C语言预处理程序的另一个重要功能，其命令一般形式为：
  - #include <文件名> 或 #include “文件名”
  - 例如：#include <stdio.h>
  - #include “someheader.h”
  - 尖括号表示在系统头文件目录中查找，双引号表示先在当前目录中查找，若找不到再到系统头文件目录中找。

214

十六 高级议题

预处理（Preprocessing）

- 条件编译
  - 预处理中的条件编译功能，使我们能按不同的条件编译出不同的代码。
  - 条件编译的三种形式，第一种形式如下：
    - #ifdef 标识符
    - 程序段1
    - #else
    - 程序段2
    - #endif
  - 功能是：如果标识符已经被#define语句定义过，则编译程序段1，否则编译程序段2

215

十六 高级议题

预处理（Preprocessing）

- 条件编译
  - 第二种形式如下：
    - #ifndef 标识符
    - 程序段1
    - #else
    - 程序段2
    - #endif
  - 与第一种形式类似，区别是使用了预处理命令ifndef，其功能变成：如果标识符没有被定义则编译程序段1，否则编译程序段2.

216

十六 高级议题

预处理（Preprocessing）

- 条件编译
  - 第三种形式如下：
    - #if 常量表达式
    - 程序段1
    - #else
    - 程序段2
    - #endif
  - 它的功能是：如果常量表达式的值为真(即非0)则编译程序段1，否则编译程序段2.

217

十六 高级议题

可移植性

- 字长和数据类型
  - 不透明类型
    - 不透明数据类型隐藏了它们内部格式或结构，作为替代，开发者利用typedef声明一个类型，即不透明类型。例如pid_t，atomic_t等。
  - 指定数据类型
    - 内核中有一些数据虽然无需用不透明类型，但它们已被指定了数据类型，比如jiffy数目用的是unsigned long型，视其为unsigned是常见错误。
  - 长度明确的类型
    - 有时我们需要在程序中使用长度确定的数据，比如一个网络数据报
  - char型的符号问题

218

十六 高级议题

可移植性

- 地址对齐
  - 避免对齐引发的问题。
  - 非标准类型的对齐。
    - 对于数组，按照基本数据类型对齐即可。
    - 对于结构和共用，使它们的成员中长度最大的数据类型对齐即可。
  - 结构体填补。
    - 为了结构体中的每一个成员都能够自然对齐，结构体要被填补。

219

十六 高级议题

可移植性

- 字节序
  - 概念：字节序指的是处理器在对字取值时，解释其中各个字节的顺序。
  - 大端序(big-endian)：
    - 最高有效位所在的字节放在最高字节位置上，其他字节依次放在低字节位置上，则该字节序称为高位优先(即大端序)。
  - 小端序(little-endian)：
    - 最低有效位所在的字节放在最高字节位置上，其他字节依次放在低字节位置上，则该字节序称为低位优先(即小端序)。

220

十六 高级议题

递归函数

- 递归
  - C函数允许嵌套调用，如果嵌套的函数刚好又是该函数本身，那么这个嵌套调用过程称之为递归。
  - 例如，经典的阶乘递归算法：
    - int fac(int i)
    - {
      - int j;
      - if (i==0 || i==1) return 1;
      - else j=fac(j-1)*i;
      - return j;
    - }

221

十六 高级议题

变参函数

- C语言的函数虽然不具备C++的多态性，但也可以接受参数不确定的情况，当然，C语言中的变参函数实际在功能上是受限的。
- 变参函数的典型代表：
  - int printf(const char *format, …);
- 编写变参函数涉及到的几个宏定义是：
  - va_start(ap, A): 初始化参数列表
  - va_arg(ap, T): 获得当前ap指向的变参值并使ap指向下一个参数。
  - va_end(ap): 提示参数提取结束

222

十六 高级议题

变参函数示例


```
void simple_va_fun(int i, ...){
    va_list arg_ptr;  //定义一个指向函数变参列表的指针arg_ptr
    int j;
    va_start(arg_ptr, i);  //使arg_ptr指向第一个可变参数
    j = va_arg(arg_ptr, int);  /* 取得arg_ptr当前所指向的参数的值，
                                  并使arg_ptr指向下一个参数  */
    va_end(arg_ptr);  //指示提取参数结束
    printf("%d %d\n", i, j);     return;
}
int main(void){
    simple_va_fun(3, 4);      return 0;
}
```

223

十六 高级议题

回调函数

- 所谓的回调函数，指的是不直接在程序中显式地调用，而是通过调用其他函数返过来调用的函数。
- 看一个典型的回调函数的例子：
  - `void (*signal(int sig, void (*func)(int)))(int);`
- 其中，func是一个函数指针，这个函数指针将来就指向了用户自定义的信号处理函数，signal函数会找到这个函数并且调用之。

224

十六 高级议题

内联函数

- 通常函数调用需要一定的时间开销。这意味着执行调用时花费了时间用于角暗里调用、传递参数、跳转到函数代码段并返回。使用类函数宏的一个原因就是可减少执行时间。
- C99为我们提供了另一种方法：内联函数，通过把一个函数声明为内联类型，将建议编译器尽可能快速地调用该函数。上述建议的效果由实现来定义。
- 定义内联函数的方法是在函数声明中使用函数说明符inline。例如：
  - inline int max(int i, int j){ return (i>j)?i:j; }
- 因为内联函数没有预留给他的单独代码块，所以无法获得内联函数的地址。

第九节强化训练

- 1、写一个带参数的宏MIN(x, y)，这个宏输入两个参数并返回较小的一个。
- 2、用预处理指令#define 声明一个常数，用以表明1年中有多少秒（忽略闰年问题）。
- 3、某头文件中有以下语句，解释其作用：
  - #ifndef SOME_HEADER_H_
  - #define SOME_HEADER_H_
    - … …
  - #endif

第九节强化训练

粤嵌教育

- 4、设计一个C函数，若处理器是大端序的则返回0，若处理器是小端序的则返回1。
- 5、编写一个函数，计算1+2+3+4+…+n的值。
- 6、用递归的思想重做第四节的第4道必做题。

227

第九节强化训练

粤嵌教育

- 7、下面函数实现数组元素的逆转，请填写空白处使其完整。

```
void recur(int a[ ], int k)
{
    int tmp;
    if( _____ )
    {
        recur( _____, _____ );
        tmp = a[0];
        a[0] = a[k-1];
        a[k-1] = tmp;
    }
}
```

228

第十节

十七、复习与测验

229