普通高等学校计算机教育规划教材

C语言程序设计题解与实验辅导

王秀贵　编著

The Answer and Practice of The C Programming Language

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

内 容 简 介

本书是主教材《C 语言程序设计教程》（王秀贵、曹江莲编著，中国铁道出版社出版）的配套教材。其中，思考题和习题给出了详细的解释和代码示例说明；对于学习中的重点、难点和容易出现的错误都给出了简明的归纳和必要的提示；每一章就所涉及的知识点精心设计了 3～6 个上机实验，并为每个实验提供了一个样例；此外，附录 A 还提供了多套测试试题，附录 B 给出这些试题的参考答案。

本书适合作为高校计算机专业及相关专业的 C 语言程序设计教程，也可作为 C 语言程序设计编程爱好者的理想参考书。

图书在版编目（CIP）数据

C 语言程序设计题解与实验辅导/王秀贵编著. —
北京：中国铁道出版社，2010.1
普通高等学校计算机教育规划教材
ISBN 978-7-113-11027-7

Ⅰ.①C… Ⅱ.①王… Ⅲ.①
C 语言－程序设计－高等学校－教学参考资料 Ⅳ.
①TP312

中国版本图书馆 CIP 数据核字（2010）第 018347 号

书　　名：C 语言程序设计题解与实验辅导
作　　者：王秀贵　编著

策划编辑：秦绪好　周海燕
责任编辑：秦绪好　　**编辑部电话**：（010）63560056
编辑助理：张　丹
封面设计：付　巍　　**封面制作**：白　雪
责任印制：李　佳

出版发行：中国铁道出版社（北京市宣武区右安门西街 8 号　　邮政编码：100054）
印　　刷：河北省遵化市胶印厂
版　　次：2010 年 3 月第 1 版　　2010 年 3 月第 1 次印刷
开　　本：787mm×1092mm　1/16　**印张**：12.25　**字数**：314 千
印　　数：4 000 册
书　　号：ISBN 978-7-113-11027-7
定　　价：20.00 元

FOREWORD

前言

本书是主教材《C 语言程序设计教程》（王秀贵、曹江莲编著，中国铁道出版社出版）的配套教材。因此，本书在结构上完全按主教材的章节展开，每一章包括以下几个方面的内容：

（1）概述。因为读者阅读“概述”的目的是加深对重要术语、概念和方法的学习，所以这部分内容侧重于对知识点的整理、分析和归纳；解答疑难；辨析异同；消除误解。

（2）常见的错误。结合每一章的内容，指明初学者在学习和编码中常犯的一些错误。因为有些错误可能会给读者在调试程序时造成困惑以及带来意想不到的后果，尤其是初学者在这方面务必要引起注意。

（3）上机实验。每一章都精心安排了 3～6 个上机实验题，这些实验题是针对本章的主要知识点而设计的，难易有别。编者希望读者通过这些实验，能够进一步加深对 C 语言语法规则的理解、纠正或消除对某些概念可能存在的误解，以及提高掌握编码技术和方法的能力。

（4）教材思考题与习题解析。本书对主教材中（除第 11、12 章中几道题外）的思考题和习题给出了详细解答，部分习题还给出了多个解决方案。思考题中细致入微的解答有助于读者在概念或 C 语言语法细节方面的深入理解；通过学习书中提供的代码，对提高读者的编程能力有很大的帮助。

（5）实验题解析。之所以给出实验题的一个参考解答，是想给读者一个参考对照。就编码类实验而言，编者所提供的解答可能并不可取，读者应相信自己能给出更好的解答。

为了检验读者的学习情况，附录 A 提供了四套测试试题，每套试题建议用时 120 分钟，读者可以进行自我测试。附录 B 提供了参考答案，供读者自我评判。当然，仅凭几套试题并不一定能完整地反映读者的学习情况，但它能告诉读者哪些知识点在认识和理解上还存在不足，以及是否具备了基本的阅读和编写程序的能力。

编者长期在大学讲授“程序设计语言”课程，从汇编语言到高级语言，从面向过程的语言到面向对象的语言，C 语言是作者感兴趣和自认为有些心得的语言之一。因此，编者在课堂上往往能够把自己对 C 语言认识的思路讲授出来。

感谢湘潭大学教务处和信息工程学院的领导和同事们，感谢中国铁道出版社的编辑们，他们为本书的编写和出版做了大量的工作。

由于作者水平所限，书中难免有错误和不足之处，恳请读者批评指正。

欢迎使用本书，期待您的建议。

编　者

2010 年 1 月

CONTENTS

目录

第 1 章 基本概念

1.1 概　　述

内存储器经常被称为“内存”或“主存”，逻辑上看是一个线性存储单元序列，在不断电的情况下，它能保存被加工的数据和数据加工以后的结果。

不论何时把某个值存放（又称“读入”）到某个存储单元中，该值将会覆盖这个存储单元中原有的值。所以，把数据读入存储单元的过程称为“破坏性读入”。

不论何时从某个存储单元中取出（又称“读出”）数据，该存储单元中仍保留着该数据不改变。所以，从内存读取数据的过程又称为“非破坏性读出”。

ANSI C 是 1989 年推出的标准化 C 版本。它既是美国国家标准协会（ANSI）公布的美国标准，亦是国际标准化组织（ISO）公布的世界标准。

严格地说，C 语言程序从开发到运行通常要经过六个步骤，即编辑、预处理、编译、连接、加载和运行。

编辑：程序员用编辑器输入并在需要的时候修改程序。

预处理：C 预处理器根据预处理命令完成包括把其他文件包含到要编译的文件中和利用程序文本替换专门的符号等操作。

编译：编译器把 C 语言程序翻译成机器语言代码（即目标代码）。

连接：连接程序把目标代码与标准库函数或程序公用库函数的代码连接起来，并产生可执行文件。

加载：加载程序从磁盘中取出可执行文件并把它装入内存。

在 VC 开发系统中，由编译器在编译前自动调用预处理器完成预处理，并在编译完成后自动调用连接装配程序完成连接和装配。

编译时有可能产生两类错误，即致命性错误（error）和警告错误（Warning error）。前者是由于程序中存在的语法错误而产生的，必须更正；后者提醒用户程序可能存在隐患，不更正程序也能运行，但可能会产生不可预料的结果。

程序在运行时也可能产生错误。例如，被零除或者程序试图访问其无权访问的内存单元。

结构化程序设计是编写清晰、正确和易于修改的程序的严格方法，其核心思想：自顶向下，逐步求精。

程序的编写应尽可能做到“可读性第一，效率第二”的准则。

在程序中加上必要的注释可以提高程序的可读性。ANSI C 规定：可以用“/*...*/”的形式为 C 语言程序的任何一部分作注释，在“/*”开始后，一直到“*/”为止中间的任何内容都被认为是注释。编译器编译时忽略注释。

C 语言程序由若干函数组成，其中一个函数必须是 main()，称其为主函数。每一个 C 语言程序都是从 main()函数开始执行，并终止于 main()函数，它与 main()函数所处的位置无关。

预处理命令是由预处理器解释执行的，一行只能书写一个预处理命令。

预处理命令#include<stdio.h>通知编译器把输入/输出头文件 stdio.h 包含到程序中，该头文件中包含了编译器用来校验对输入/输出函数（如 printf()和 scanf()）调用是否正确的相关信息。

在 C 语言中，分号“;”是语句的一部分，而且是一个语句的终结符。也就是说，分号不是语句之间的“分隔符”。

1.2 常见的错误

其实，就程序设计而言，初学者最常见的问题就是编写程序少，上机实验少，因此就会觉得编写程序太难了。

1.3 上 机 实 验

实验目的：

1. 初步掌握使用 VC++ 6.0 开发平台调试 C 语言程序的基本方法；
2. 了解如何在该系统上编辑、编译、连接和运行一个 C 语言程序；
3. 通过运行简单的 C 语言程序，初步了解 C 语言源程序的特点。

实验内容：

实验一：创建一个新的 C 语言的工程，修改新工程的配置。

实验二：添加一个文件到一个空的工程中。

实验三：输入如下程序，然后编译并运行该程序。

```
#include <stdio.h>
int main()
{
  int i,j;
  printf("\1\1\n");
  for(i=1;i<11;i++)
  {
    for(j=1;j<=i;j++)
      printf("%c%c",219,219);
    printf("\n");
  }
  return 0;
}
```

运行以上程序，看看输出什么结果。注意，程序中的“219”是 ASCII 码，查看 ASCII 码表，

找出它代表哪个字符。然后进行如下操作：

（1）更换 ASCII 码，再运行程序查看效果。

（2）将程序第四行中的“int”去掉，再查看编译结果。

实验四：初步掌握调试程序的一种方法——设置断点。

实验五：初步掌握调试程序的一种方法——动态查看变量的值。

1.4 教材思考题与习题解析

1.1 内存的存储特性是什么？

答：最典型的存储特性是数据一次存入，可多次取出，即取出的是它的副本，如果数据被覆盖或者关闭计算机，就会导致内存中的数据完全丢失。这个特性使得内存适合存储一些临时性数据，因为一旦计算机重新启动，这些数据就会自动消失。也正是这个特性，使得内存不适合存储重要的数据，因为一旦计算机死机，这些东西就再也找不回来了。

1.2 自顶向下和逐步求精的具体含义是什么？

答：自顶向下、逐步求精是结构化程序设计的基本准则，它们分别从两个不同的方面概括了结构化程序设计的方法。自顶向下指的是程序功能模块的划分方法，强调“自顶向下”分解功能模块，即由上到下、由抽象到具体将程序功能逐层分解；逐步求精指的是编码应遵循的方法，由粗到细，逐步降低程序的抽象级。

1.3 在编写程序时，为什么要尽可能地利用标准库函数而不是自己去实现它？

答：因为标准库函数通常是开发商经过反复验证并具有准确性、高效性和可移植性的函数，因此使用标准库函数既不会出错，又可以提高开发效率，降低开发成本，还可以使程序的结构更加清晰、易读。

1.4 一个 C 语言程序往往不是由一个而是由多个函数构成。为什么不写成一个函数呢？

答：一个程序如果是由一个函数构成，其中至少有两个重大缺陷。首先，程序的结构很混乱，易读性极差；其次，调试和验证程序的正确性很困难甚至不可能进行验证。

1.5 “在 C 语言程序中，语句之间必须用分号分隔”这个命题成立吗？

答：不成立。分号不是 C 语言语句的分隔符，而是语句的终结符。

1.6 C 语言程序的文本外观格式非常自由，几乎无规则可言。但主教材例 1-1 似乎遵循了某种空白使用规则。你对此有何想法？

答：至少视觉效果不错，程序结构很清晰，给读懂程序带来很大帮助。

1.7 什么是编译时错误？什么是运行时错误？请分别举出两种不同的错误例子。

答：这两种错误分别指的是程序在编译期间出现的错误和运行期间出现的错误。将关键字定义为变量或语句没有用分号结束都会导致编译时错误；被 0 除或数值溢出会导致程序运行时异常终止。

1.8 预处理命令（preprocessor directives）是怎么处理的？

答：预处理命令是由预处理器解释执行的。解释执行过程可以简单描述为：预处理器首先读入源程序文件，然后根据预处理命令修改源程序（比如文件包含命令 include，它将指定的头文件中的内容替换这条命令），最后将修改后的源程序交编译器处理。

1.9 下面的程序有何错误?

```
#include <stdio.h>
int main()
{                                   /* 本程序打印 "Hello, world! " */:
  printf("Helloworld\n");
}
```

答：在注释行的最后多了一个冒号，这将导致编译错误。

1.10 下面的程序有何错误?

```
#include <stdio.h>;
int main()
{                                   /* 本程序打印 "n=5" */
  n=0;
  printf("n=3+2\n")
}
```

答：本程序存在三个方面的错误。首先，存在两个致命性错误，它们是变量 n 未定义就被引用；printf 作为一个独立的语句在最后缺少一个分号。这两个致命性错误导致程序不能产生目标代码。其次，是未实现程序的要求，应输出 "n=5" 而实际输出的是 "n=3+2"，这个错误既不是编译错误也不是运行错误，而是属于算法错误。最后是存在两个警告错误：预处理命令行后面多了一个分号，尽管这个分号不会影响生成目标代码；另一个警告错误是编译器提醒用户：变量 n 没有被引用。

1.11 改写主教材例 1-1，要求是去掉求最大公约数的那个函数。

答：下面是它的一种改写方案。

```
#include <stdio.h>                  /* 预处理命令 */
#include <stdlib.h>                 /* 预处理命令 */
int lcm(int,int);                   /* 求最小公倍数函数原型声明 */
int main(void)                      /* 以下是主函数模块 */
{
   int a,b;                         /* 定义两个整型变量 a 和 b */
   printf("Enter 2 integers:a and b.(a>0,b>0): ");      /* 输出提示行 */
   scanf("%d%d",&a,&b);             /* 输入 a 和 b */
   printf("lcm(%d, %d) = %d \n",a,b,lcm(a,b));          /* 输出结果 */
   return EXIT_SUCCESS;             /* 程序正常结束返回 */
}
int lcm(int m,int n)                /* 以下是求最小公倍数函数模块 */
{
  int a,b,r;
  a=m;
  b=n;
  while(b!=0)
  {
    r=a%b;
    a=b;
    b=r;
  }
  return m*n/a;                     /* 返回函数计算结果 */
}
```

1.5 实验题解析

实验三：在这里，我们以注释的形式对程序中的语句行作简单解释。

```
#include <stdio.h>                /* 预处理命令,它支持程序中的输入和输出 */
int main()                        /* main()为主函数名,C语言程序不可缺少的函数 */
{
  int i,j;                        /* 定义了两个变量,取名为i和j */
  printf("\1\1\n");               /* 输出两个笑脸符.\1中的"1"是笑脸符的ASCII码 */
  for(i=1;i<11;i++)               /* 循环 */
  {
    for(j=1;j<=i;j++)
    printf("%c%c",219,219);       /* 219在这里是一个ASCII码 */
    printf("\n");
  }
  return 0;
}
```

第 2 章 数据描述与数据类型

2.1 概　　述

在 C 语言中，数据类型可以归纳为四种——整型、浮点型、指针和构造类型，所有其他的数据类型都可以从这四种的组合派生而来。

26 个英文字母在 C 语言中是区分大小写的，因此 a 与 A 是两个不同的字母。

保留字的含义是系统预定义好的，程序员无权更改它的固有含义。

标识符的首字符必须是字母或下画线，它通常用来表示变量名、函数名和常量名。通常，标识符的长度不要超过 31 个字符，便于提高程序的可移植性。

常量是不允许改变的量。

整型常量的缺省类型是 int，但可以通过加后缀的方法更改缺省规则。例如，12L 将被解释为 long int 型值。整型常量除了十进制表示方式外，还可以用八进制和十六进制表示。在实际应用中，究竟采用哪种表示形式取决于程序中应用的需要。例如，对于十进制数 61440 的十六进制表示是 0xF000。显然，如果需要知道某个数的哪几位是 1 或 0，则把整型常量写成十六进制形式更为清晰。

字符常量的类型是 int 型，即它以整数的形式（该字符的 ASCII 码）存储，但不能像整型常量那样通过加后缀的方法来改变它的缺省规则。

实型常量只有十进制一种表示形式，必须有一个小数点或一个指数，或者两者都有。实型常量的缺省类型是 double，除非它的后面跟一个 L 或 l 表示它是一个 long double 类型的值，或者跟一个 F 或 f 表示它是一个 float 类型的值。

可以给一个常量命名一个符号名，即所谓符号常量。符号常量的意义在于可以提高程序的可读性和可维护性。#define 命令是命名符号常量的一种常用方法。

一个变量名（简称为变量）本质上是内存某存储区的一个抽象。任何一个变量，必须先定义才能被引用。如果有必要，在定义一个变量的同时可以对其初始化。

应该注意的是，尽管 C 语言引入 char 类型的目的是为了存储字符型值，但字符本质上是整型值，即字符总是以整数形式存储。因此，一个 char 型的值要么是 signed char，要么是 unsigned char，这取决于编译器。这个事实意味着不同系统上的 char 可能拥有不同范围的值。但只要使用 ASCII 码字符集中的字符，程序总是可移植的。

一个未初始化的变量，其值可能是不确定的（更详细的介绍见教材 8.15 节）。直接引用未初始化的变量可能是一个错误。

C 语言允许程序员为各种数据类型定义新的类型名。命名新类型名的最常用的方法是使用保留字 typedef。使用 typedef 声明类型的好处是可以使得类型声明变得简单明了，同时可提高程序的可维护性。在教材第 9 章和第 10 章有很多这方面的介绍及应用。

值得注意的是，#define 命令也可以命名新的类型名。读者最好使用 typedef 而不要使用#define 来创建新的类型名，因为后者无法正确地处理指针类型。

指针类型是一个重要的数据类型。在本章，读者应牢牢地记住指针变量（简称指针）的值是一个地址值，并明确“空”指针和“悬挂”指针的区别。另外，读者也应该掌握将一个指针指向一个具体目标的方法，并通过这个指针去访问它所指向的目标。

还要提及的一点是，当定义一个变量并初始化后，如果这个变量的值不允许再被更改，则应当在定义中加上关键字 const，这种方法不仅使用户的意图在该程序的阅读者面前得到清晰的展现，而且当这个变量的值被意外更新时，编译器能捕获这个错误。

2.2　常见的错误

1. 大小写字母不加区分的使用。
2. 对各种类型常量的缺省类型不重视。
3. 标识符的首字符不是字母或下画线。
4. 直接引用一个未定义的变量或一个未初始化的变量。
5. 预处理命令不以#开始，却又错误地用分号结束。

2.3　上 机 实 验

实验目的：

1. 熟悉各种常量的缺省规则，牢记字符常量本质上是整型的；
2. 掌握定义整型、字符型和实型以及指针变量及其初始化的方法；
3. 理解#define 命令和修饰符 const 的意义及其使用方法；
4. 进一步熟悉 VC++开发工具。

实验内容：

实验一：下面的程序将小写字母转换为大写字母，运行下面程序并分析结果。

```
#include <stdio.h>
int main()
{
  char c1='a',c2='b';                    /* 定义字符型变量并初始化 */
  c1=c1-32;                              /* 小写转换为大写,明白其中原因吗*/
  c2=c2-32;
  printf("%c %c\n",c1,c2);               /* %c 意为要求按字符形式输出 */
  printf("%d %d\n",c1,c2);               /* %d 要求按十进制整数形式输出.为什么 */
}
```

编译然后运行，分析程序输出结果。然后再将程序中的第四行即：

```
char c1 = 'a',c2 = 'b';
```

修改为

```
char c1 = 97,c2 = 98;
```

再编译并运行，要理解为什么会得到这样的结果。

实验二：下面这个程序输出结果“a = 10”。编译运行过程中如有错，请更正。

```
#include <stdio.h>
int main(void)
{
  int a=5;
  printf("%d",A)
  return 0;
}
```

实验三：调试下面的程序，更正其中的错误。

```
#include <stdio.h>
#define 20 MAX
int main(void)
{
  Int *p,a=5;
  int const elem;
  p=a;
  elem=10;
  printf("a = %d, elem = %d, elem+a = %d\n",*p,elem,elem+a);
  return 0;
}
```

2.4 教材思考题与习题解析

2.1 一定要将常量定义成符号常量吗？

答：不一定。但这种做法是提倡的，其好处：语义更明确；提高程序的维护性；便于保持数据的一致性。

2.2 为什么说对字符串常量的修改是极其危险的？如果必须要修改，那么该怎么办？

答：ANSI C 认为对字符串常量的修改，其结果未定义。这是因为，不同的字符串可能相邻地存储在一片连续的空间；相同的字符串可能共用同一个空间，这就使得对字符串 A 的修改可能涉及字符串 B。一个字符串要被修改，采用字符数组存储是安全的。

2.3 用#define 定义的符号常量和用 const 定义的变量有什么异同？为什么要用 const 修饰变量？

答：两者都可以当做常量使用。但是，符号常量只能在语法上允许常量出现的地方使用；const 常量只能用于允许使用变量的地方。定义 const 变量的目的是明确地告诉程序阅读者，这个变量的值是不能被修改的。如果试图修改它，编译器会报告错误。符号常量能提高程序的可读性和维护性，const 常量可增强数据的安全性。

2.4 用#define 定义符号常量有什么好处？

答：至少有两个明显的好处，易读性更好；维护更方便且不易出错。

2.5　变量定义的作用是什么?

答：它告诉编译器所定义的变量名和类型，要求按指定的类型分配空间。

2.6　既然枚举型变量事实上就是整型变量，那么在应用中是否可以将枚举型变量当做整型变量来使用。

答：从语法的角度看是合法的。但如果将枚举型变量无差别地混同于整型变量，枚举类型的定义就变得毫无意义了。例如：

```
enum Color{ RED, BLUE, GREEN, WHITE, BLACK } mycolor;
mycolor = RED;
```

如果将 mycolor 当做整型变量输出，其结果为 0。颜色为 0，它是什么颜色呢?

2.7　什么是溢出？整数溢出与浮点数溢出有什么区别?

答：当发生整数溢出时，这个变量会被“反卷”成负数，产生错误的结果；当发生浮点数溢出时，这个变量的值会被设为代表无穷大的常数 inf。

2.8　如果整数类型在正常情况下是带符号的，那么关键字 signed 的目的是什么?

答：由于字符在本质上是（小）整型值，但 ANSI C 标准并没有规定缺省的 char 究竟是 signed char 还是 unsigned char。这意味着不同系统上的 char 可能拥有不同范围的值，所以只有当程序中的 char 变量的值位于它们二者的交集中，这个程序才是可移植的。

显然，在这种情况下，如果把 char 定义为 signed 或 unsigned，可提高这类程序的可移植性，能确保在不同系统的机器中在字符是否为有符号值方面保持一致。

2.9　数值溢出错误是编译错误还是运行错误？为什么说这种错误更严重?

答：这是一种在运行时出现的错误。其严重性在于编译器不能捕获它，调试者也往往难以发现和定位问题所在，甚至程序崩溃了，还不知问题出在哪里。

2.10　下列代码有什么错误?

```
enum fruit {apple,orange,lemon,tomato};
enum vegetable {potato,carrot,tomato};
```

答：第二个 enum 试图重新定义常量 tomato。

2.11　以下两条语句有何不同：

```
char ch = 'A';
char ch = 65;
```

答：没有什么不同。

2.12　指出下列内容哪些是 C 语言的整型常量，哪些是实型常量，哪些两者都不是。

1E-4	A423	-E-31	0xABCL	.32E31	087
0xL	0x12.5	11E	056L	0123.	.0

答：整型常量：1E-4　087　056L

实型常量：.32E31　0123.　.0

2.13　在计算机上运行下面的程序，分析它的输出结果，解析其原因。

```
#include <stdio.h>
int main()
{
  int n=10;
  while(n>0)                    /* 当 n>0 时输出 n 的值 */
```

```
    printf("n=%d\n",n=n*n);
}
```

答：在 Turbo C 下运行后输出结果是：

```
n = 100
n = 10000
n = -7936
```

可见，由于程序中用 n*n 不断地改变整型变量 n 的值，当 n 的值超出了 n 的表示范围时，其值被“反卷”为负数，从而结束程序。由此可见，整型数超出表示范围时，该编译器并不视为错误。

2.14 在计算机上运行下面的程序，分析它的输出结果，解释其原因。

```
#include <stdio.h>
int main()
{
  float x=10.0;
  while(x>0)
    printf("x = %g\n",x=x*x);
}
```

答：在 Turbo C 下运行后输出结果是：

```
x = 100
x = 10000
x = 1e+08
x = 1e+16
x = 1e+32
Floating point error: Overflow.
```

可见，由于程序中用 x*x 不断地改变实型变量 x 的值，使得当 x 的值超出 x 的表示范围时，报告错误，从而结束程序。由此可见，实型数超出表示范围时，该编译器视为错误，并且非正常终止程序运行。

2.15 分析下列程序的输出结果。

```
#include <stdio.h>
int main()
{
  char ch='A';
  printf("char(%c) = %c, A = %c\n",ch,ch,'A');
  printf("int(%c) = %d, A = %d\n",ch,ch,'A');
}
```

答：在 Turbo C 下运行后输出结果是：

```
char(A) = A, A = A
int(A) = 65, A = 65
```

输出说明 char 型变量可以与 int 型变量一起参加运算。

2.16 分析下列程序的输出结果，解释其原因。

```
#include <stdio.h>
int main()
{
  int a=10;
  int *iptr=&a;
  char *sptr="String";
```

```
    int *ptr=0;
    printf("a = %d\t*iptr = %d\n",a,*iptr);
    printf("s = %s\n", sptr);
    printf("*ptr = %d\n",*ptr);
}
```

答：在 Turbo C 下运行后输出结果是：

```
a = 10, *iptr = 10
s = String
*ptr = 283
```

因为指针 iptr 和 sptr 分别指向变量 a 和字符串“String”，所以程序输出的前两行是正确的。由于指针 ptr 被初始化为空指针，它没有指向一个确定的目标，所以程序输出的第三行是没有意义的。可见，该编译器对空指针的直接引用并不认为是一种错误，这显然潜伏着危险。

2.17　仿照习题 2.15 编写并运行一个程序，打印元音字母字符 A、B 和 Z 以及数字字符 0、1 和 9 的 ASCII 码。

答：可编写如下程序。

```
#include <stdio.h>
int main()
{
  printf("char(A) = %d, B = %d, Z = %d\n",'A','B','Z');
  printf("digit(0) = %d, 1 = %d, 9 = %d\n",'0','1','9');
}
```

2.5　实验题解析

实验一：要理解如下两点。

1. 在 C 语言中，字符本质上是一个整数。字符常量的缺省类型是 int 型，这意味着用一对单引号括起来的一个字符实际上是书写整数的另一种方法。因此，在一个 ASCII 码实现中，'a'和 97（十进制）或 0141（八进制）、0x61（十六进制）相同。当需要它以字符形式输出时，即是字符；当需要它以整数形式输出时，即是一个整数。

2. 'a'的 ASCII 码是十进制 97，'A'的 ASCII 码是十进制 65。

实验二：原程序存在三个错误。

1. 由于输入不用心，在 printf()函数中将变量 a 误写为 A。这会导致一个错误：编译器显示变量 A 没有被定义。

2. 在函数 printf()的最后少输入一个分号，这也是一个编译错误。从语法上讲，printf()是一个函数表达式，它本身并不要求以分号结束。但在这里，printf()是作为一个独立的语句出现的，C 语言规定，一个语句必须以分号结束。

3. 输出的结果没有达到程序的要求。编译器并不能捕获这个错误（我们称这一类错误为算法错误）。要达到程序要求的输出，应该将 printf()改为：

```
printf("a = %d",a+5);
```

实验三：下面这个程序存在三个编译错误，下面以注释的形式标示。

```
#include <stdio.h>
#define 20 MAX                          /* 定义中的顺序反了 */
```