第一章 C 语法基础

1.1 C 语言基本数据类型

例题 1.1.1：以下为 Linux 下的 32 位 C 程序，请计算 sizeof 的值

1. char str[] = "Hello";
2. char *p = str;
3. int n = 10;

【答案】6,4,4

【来源公司】联发科（2017）

【考点】基本数据类型大小

【解题思路】str 为一个字符数组，被字符串"Hello"初始化，而字符串末尾自动添加‘\0’字符，因此大小为 6。p 为字符指针，指向字符数组 str，而在 32 位系统中，指针的大小始终为 4 个字节。n 为整型变量，在 32 位系统中，长度为 4 个字节。

例题 1.1.2：局部变量，函数返回值，动态申请的内存（malloc）分别存放于：

A. 堆，栈，堆

B. 堆，堆，堆

C. 栈，堆，堆

D. 栈，栈，堆

【答案】D

【来源公司】联发科（2017）

【考点】数据存储

【解题思路】进程空间对应的分区及对应的存储内容：

代码区：所有的可执行代码（程序代码指令、常量字符串等）
静态区：程序中所有的全局变量和静态变量
栈　　：所有的自动变量、函数形参
堆　　：malloc手动分配的变量

例题 1.1.3：

```
#include <stdio.h>
void print(unsigned char a[])
{
    int i = 0;

    while (i < 4)
    {
        printf("%d,  ", *(a + i));
        i++;
    }
}
int main()
{
    unsigned short a[4] = {256, 192, 128, 1};
    print(a);
    return 0;
}
```

以上程序运行的结果是：

【答案】0,1,192,0

【来源公司】联发科（2017）

【考点】数据存储及强制转换

【解题思路】本题的关键是注意到 main 函数中的 a 数组为 unsigned short 型数组，而 print 函数的形参为 unsigned char 型指针（函数形参为数组时，编译器将其会退化成相应类型的指针，因此 print 函数定义等价于 void print(unsigned char *a)），二者之间存在类型的不匹配。在内存中，数组 a 可以表示为（长度为 2*4=8 个字节）：

0x00	0x 01	0xC0	0x00	0x80	0x00	0x01	0x00
a[1]=256=0x0100		a[2]=192=0x00C0		a[3]=128=0x0080		a[4]=1=0x0001	
低位							高位

在 print 函数中，通过指针 a 访问该数组时，由于指针类型为 unsigned char*，指针运算时只能以字节为单位，因此结果应该为整个数组的前四个字节的内容，分别为

0x00	0x 01	0xC0	0x00

，即 0,1,192,0。

例题 1.1.4： 设有语句 char a = '\72';则关于变量 a 的说法正确的是？（）

A． 包含 2 个字符

B．说明不合法

C．包含 1 个字符

D． 包含 3 个字符

【答案】C

【来源公司】趋势科技（2017）

【考点】转义字符

【解题思路】

\为转义字符，\72 转义为一个八进制数 72，也就是十进制数的 58

赋值给 a，a 就是一个 ascii 码为 58 的字符。

例题 1.1.4：以下程序统计给定输入中每个大写字母的出现次数(不需要检查输入合法性)

```
1  void AlphabetCounting(char a[],int n){
2    int count[26]={},i,kind=0;
3    for(i=0;i<n;++i)  (1);
4    for(i=0;i<26;++i){
5      if(++kind>1) putchar(';');
6      printf("%c=%d",(2));
7     }
8  }
```

以下能补全程序中的（1）和（2）处，完成功能的选项是()

A． ++count[a[i]-'Z'];'Z'-i,count['Z'-i]

B． ++count['A'-a[i]];'A'+i,count[i]

C． ++count[i];i,count[i]

D． ++count['Z'-a[i]];'Z'-i,count[i]

E． ++count[a[i]];'A'+i,count[a[i]]

【答案】D

【来源公司】CVTE（2017）

【考点】字符处理

【解题思路】

题意为输入设定全部是大写（ASCII 码 A-Z 为 65-90，递增），所以有两种情况：

1、count[0;25]存储 A-Z 的个数，即 count[0]存储 A 的个数，于是(1)处为：++count[a[i]-'A']；(2)处为：'A'+i，count[i]；

2、count[0;25]存储 Z-A 的个数，即 count[0]存储 Z 的个数，于是(1)处为：++count['Z'-a[i]]；(2)处为：'Z'-i，count[i]。

答案为 D。

例题 1.1.5：小数值 1.5625 的二进制表示是？（）

A. 101.1001

B. 0.001

C. 101.111

D. 1.1001

【答案】D

【考点】进制转换

【解题思路】

本题可以采用排除法快速选择，小数点左边是 1，二进制仍然是 1，只有 D 选项符合。小数点右边采用乘 2 取整法。

0.5625 * 2 = 1.125　　取整为 1

0.125 * 2 = 0.25　　取整为 0

0.25 * 2 = 0.5　　取整为 0

0.5 * 2 = 1　　取整为 1

最终结果为 1.1001，选项 D 正确。

1.2 关键字及表达式

例题 1.2.1：C 语言关键字 volatile 修饰符变量时的作用？

A. 声明变量定义在外部文件

B. 优化该变量的存取

C. 尽可能将该变量的值缓存在寄存器中

D. 告诉编译器读取该变量时从变量地址取值

【答案】D

【来源公司】联发科（2017）

【考点】volatile

【解题思路】volatile 关键字修饰变量的目的是告诉编译器该变量的值随时可能发生变化，因此每次使用时都必须直接从原始内存地址进行存取。volatile 的重要性对于搞嵌入式的程序员来说是不言而喻的，对于 volatile 的了解程度常常被不少公司在招聘嵌入式编程人员面试的时候作为衡量一个应聘者是否合格的参考标准之一。一般说来，volatile 用在如下的几个地方：

1、中断服务程序中修改的供其它程序检测的变量需要加 volatile；

2、多任务环境下各任务间共享的标志应该加 volatile；

3、存储器映射的硬件寄存器通常也要加 volatile 说明，因为每次对它的读写都可能有不同意义。

例题 1.2.2：（多选题）以下合法的 C 语言表达式有：

A. 5.0 % 2

B. (5, 3, 1, 0)

C. x = y = z = 1;

D. 1 = 1 = 1 = 1;

E. ((a++)++) + b

F. 0 <= x <= 100

【答案】BCF

【来源公司】联发科（2017）

【考点】表达式

【解题思路】A 中取余表达式的操作数必须为整数，因此 A 非法；D 中由于常量不可以被赋值，因此 D 错误；E 中++操作数的左值必须为变量，而(a++)为表达式，值为常量，因此(a++)++表达式非法；另外，对于 BCF 选项，大家可以讨论一下其对应的表达式的值为多少呢？

例题 1.2.3：下面关于 auto、register、static、extern 变量描述错误的是

A. 一般情况下，不作专门说明的局部变量，均是自动变量（auto 变量）

B. register 变量存放在寄存器中，可以提高运算速率，它绝不会存在于内存，所以不能

用地址运算符&

C. static 局部变量始终存在着，也就是说它的生存期为整个程序

D. extern 变量表示变量定义在别的文件中，提示编译器遇到此变量或函数时，在其他模块寻找其定义，只是对变量的一次重复引用，不会产生新的变量

【答案】B

【来源公司】联发科（2017）

【考点】关键字

【解题思路】B 项错在一定二字。一般情况下，变量的值是存储在内存中的，CPU 每次使用数据都要从内存中读取。如果有一些变量使用非常频繁，从内存中读取就会消耗很多时间，为了解决这个问题，可以将使用频繁的变量放在 CPU 的通用寄存器中，这样使用该变量时就不必访问内存，直接从寄存器中读取，大大提高程序的运行效率。关于寄存器变量有以下事项需要注意：

1. 为寄存器变量分配寄存器是动态完成的，因此，只有局部变量和形式参数才能定义为寄存器变量。

2. 局部静态变量不能定义为寄存器变量，因为一个变量只能声明为一种存储类别。

3. 寄存器的长度一般和机器的字长一致，所以，只有较短的类型如 int、char、short 等才适合定义为寄存器变量，诸如 double 等较大的类型，不推荐将其定义为寄存器类型。

4. CPU 的寄存器数目有限，因此，即使定义了寄存器变量，编译器可能并不真正为其分配寄存器，而是将其当做普通的 auto 变量来对待，为其分配栈内存。当然，有些优秀的编译器，能自动识别使用频繁的变量，如循环控制变量等，在有可用的寄存器时，即使没有使用 register 关键字，也自动为其分配寄存器，无须由程序员来指定。

例题 1.2.4：解释以下几个的区别：const char *p,char const *p,char *const p

【答案】

const char *p 是指针指向的内容不能修改；

char *const p 是指针本身不能修改；

char const *p 和 char *const p 等价。

【来源公司】大华（2017）

【考点】表达式

【解题思路】本题目的关键是要会看 const 修饰的对象，理解 const 的作用。

例题 1.2.5：读下面程序，请给出 test（ ）函数的返回值。

```
1   int test( )
2   {
3       int k=0;
4       char c='A';
5       do{
```

```
6           switch (c++)
7           {
8               case 'A': k++; break;
9               case 'B': k--;
10              case 'C': k+=2; break;
11              case 'D': k=k%2; break;
12              case 'E': k=k*10; break;
13              default: k=k/3;
14          }
15          k++;
16      }while(c<'G');
17      return k;
18  }
```

【答案】8

【来源公司】趋势科技（2017）

【考点】表达式

【解题思路】本题目的关键是细心。

例题 1.2.6：以下表达式的结果（）？

int a = 0;

int b = (a=-1) ? 2:3;

int c = (a=0) ? 2:3;

A． b=2, c=2

B．b=3, c=3

C．b=2, c=3

D． b=3, c=2

【答案】C

【来源公司】趋势科技（2017）

【考点】赋值表达式，三目操作符

【解题思路】a=-1,表达式的值为-1，非零，为真，故 b=2; a=0,表达式的值为，为零，为假，故 c=3;

1.3 数组和指针

例题 1.3.1：有 c 语句如下:

int a[5]={2,3,1,6,7};int *p=a;

下列表达式中不能得到 1 的是：

A. a[2] B. a[1]-1 C. *(p+2) D. *p-1

【答案】B

【来源公司】烽火科技（2017）

【考点】数组访问

【解题思路】AC 选项等价，均访问数组的第三个元素，因此值为 1；D 选项中，*p 的值为 a[0]，因此表达式的值为 2-1=1；B 选项中 a[1]-1 为 3-1=2；因此选择 B。

例题 1.3.2：

若函数 fun 的函数头为：

int fun(int i, int j);

函数指针变量 p 指向函数 fun 的赋值语句为:

A. p = fun(i, j)

B. p = fun

C. p = * fun();

D. p =& fun

【答案】B

【来源公司】联发科（2017）

【考点】函数头的理解

【解题思路】

函数头中的 fun 为函数名。与数组名类似，函数名为该函数对应代码的起始地址，因此 fun 本身就是一个地址（指针）常量。顺便大家可以回忆一下，变量 p 应该如何正确定义呢？

例题 1.3.3：一个指向整型数组的指针可以定义为：

A. int(*ptr)[]

B. int *ptr[]

C. int*(ptr[])

D. int ptr[]

【答案】A

【来源公司】大华（2017）

【考点】数组指针

【解题思路】在理解 A 选项的同时，B 选择项则定义了一个指针数组，C 选项同 B 选项，D 选项为普通的整型数组。

例题 1.3.4：typedef const char* CHAR; CHAR 的类型是个常量指针还是个指向 char 类型常量的一个指针？

【答案】后者

【来源公司】大华（2017）

【考点】const

【解题思路】本题中由于 const 修饰的是 char，因此 CHAR 类型为指向常量 char 的指针。

例题 1.3.5：声明语句为 int a[3][4]; 下列表达式中与数组元素 a[2][1]等价的是？

A.*(a[2]+1)

B.a[9]

C.*(a[1]+2)

D.*(*(a+2))+1

【答案】A

【来源公司】趋势科技（2017）

【考点】数组名

【解题思路】a[2][1] 等价于 *(a[2] + 1) 或 *(*(a + 2) + 1) ，因此 A 正确，CD 错误。

例题 1.3.6：假如整型指针 p 已经指向某个整型变量 x，则(*p)++和下面哪一个等价?

A． p++

B． x++

C． *(p++)

D． &x++

【答案】B

【来源公司】趋势科技（2017）

【考点】指针运算

例题 1.3.7：对于 int *pa[5]; 的描述，正确的是？（）

A． pa 是一个具有 5 个元素的指针数组，每个元素是一个 int 类型的指针；

B．pa[5]表示某个数组的第 5 个元素的值；

C．pa 是一个指向数组的指针，所指向的数组是 5 个 int 类型的元素；

D． pa 是一个指向某个数组中第 5 个元素的指针，该元素是 int 类型的变量；

【答案】A

【来源公司】趋势科技（2017）

【考点】数组指针和指针数组

【解题思路】

int *pa[5]，由于[]优先级比*高，所以 pa 与[]先结合，pa[5]表明 pa 是一个数组，大小是 5，既然知道 pa 是数组了，接下来就是确认数组元素了，int*表明数组元素是指针，因此 int

*pa[5]为指针数组;

int（*p）[5]，首先()优先级比[]高，所以 pa 先与*结合，*pa 表明 pa 是一个指针，既然知道 pa 是指针，接下来确认指针指向的数据类型，int [5]表明指针指向大小为 5 的 int 型数组。

例题 1.3.8：char iArray[100] = {0};以下会有异常的是（）

A、memset(iArray[0],0,100);

B、memset(&iArray[0],0,100);

C、memset(iArray,0,100);

D、memset(&iArray,0,100);

【参考答案】A

【来源公司】CVTE（2017）

【考点】数组名, memset

【解题思路】

Memset 函数的第一个参数应该为地址，因此选项 A 错误。

例题 1.3.9：若定义：int a =10,*p=&a,**pp = &p;则表达式为真的有（）

A、a==*p;

B、&a==*pp；

C、p==*pp;

D、&p==pp;

【参考答案】ABCD

【来源公司】CVTE（2018）

【考点】二维指针

例题 1.3.10：16 位的嵌入式系统，一个指针占用几个字节（）

A、 8 个字节

B、4 个字节

C、1 个字节

D、2 个字节

【答案】D

【来源公司】CVTE（2017）

【考点】指针长度

【解题思路】指针的长度取决于地址线的宽度，由于为 16 位的嵌入式系统，那么长度为两个字节。

1.5 函数及预处理

例题 1.5.1：

int main(void)

{

printf("%5.3s\n", "printf");

return 0;

}以上程序输出结果为：

A. pri

B. printf

C. pri.ntf

D. ntf

【答案】A

【来源公司】联发科（2017）

【考点】格式化的字符串输出

【解题思路】

%m.ns：输出占 m 列，但只取字符串中左端 n 个字符。这 n 个字符输出在 m 列的右侧，左补空格。

%-m.ns：其中 m、n 含义同上，n 个字符输出在 m 列范围的左侧，右补空格。如果 n>m，则自动取 n 值，即保证 n 个字符正常输出。

例题 1.5.2：

```
#define cubc(x) (x*x*x)
int main()
{
    int a = 2;
    int b;
    b = cubc(a + 3);
    printf("%d\n", b);
}
```

程序输出为：

【答案】17

【来源公司】联发科（2017）

【考点】宏展开

【解题思路】

展开式为（2+3*2+3*2+3），结果应为 17。切记宏展开为机械的展开。

例题 1.5.3：以下程序

```
1   #include <stdio.h>
2   #define add(x,y) x+y
3   int c = 10;
4   int main()
5   {
6       void func(int c);
7       static int a = 5;
8       int b = 6;
9       int c = 2;
10
11      printf("a = %d, b = %d, c = %d\n", a, b, c);
12      func(c);
13      printf("a = %d, b = %d, c = %d\n", a, b, c);
14      func(c);
15  }
16  void func(int c)
17  {
18      static int a = 4;
19      int b = 10;
20      a += c*add(a, b);
21      c += 10;
22      b += c;
23      printf("a = %d, b = %d, c = %d\n", a, b, c);
24  }
```

的输出为：

【答案】

a = 5, b = 6, c = 2

a = 22, b = 22, c = 12

a = 5, b = 6, c = 2

a = 76, b = 22, c = 12

【来源公司】联发科（2017）

【考点】宏展开，静态变量，作用域

【解题思路】本题的主要注意点有：

1.第 11 行的输出考察考生对作用域的理解，此处的变量 c 应该为当前作用域（main 函

数中的）的变量 c=2，而非全局变量 c=10；

2.第 12 行调用函数 func，因此在 func 函数内部，c 为传参 c=2；同时注意 func 中的 a 为静态变量；func 函数中第 20 行为机械性的展开，可以以写成 a += c*a+b;

3.第 13 行同样考察作用域的理解，函数 func 的 abc 变量和 main 函数中 abc 变量无关，为两个不同作用域的值，因此第 13 行的输出与 11 行相同。

4.第 14 行调用 func 时，注意此时 func 函数中的变量 a 已经不为 4，应为上次的计算结果 22。

例题 1.5.4：以下程序的输出值是多少？

```
1   #include <stdio.h>
2   #define f(a,b) a+b
3   #define g(a,b) a*b
4   int main(int argc, char **argv)
5   {
6       int m;
7       m=2*f(3,g(4,5));
8       printf("\n m is %d\n",m);
9       return 0;
10  }
```

【答案】26

【来源公司】趋势科技（2017）

【考点】

【解题思路】#define 只是单纯的文本替换，文本替换过来则是: m=2*3+4*5。

例题 1.5.5：以下程序的输出值是多少？

```
1   #include<iostream>
2   using namespace std;
3   int nest(int i)
4   {
5       if (i < 0 )
6           return 0;
7       else if (i == 0)
8           return 1;
9       else
10          return nest(i-1) + nest(i-2) + i;
11  }
```

```
12  int main( )
13  {
14      cout << nest(7)<< endl;
15      return 1;
16  }
```

【答案】100

【来源公司】趋势科技（2017）

【考点】递归和递推

【解题思路】可以看出 nest 函数为递归函数，而且相邻项之间存在联系，因此可以采用从底向上的思路，由 nest(0)算起，能够更加方便得解题：

nest(0)=1

nest(1)=2

nest(2)=5

nest(3)=10

nest(4)=19

nest(5)=34

nest(6)=59

nest(7)=100

例题 1.5.6.在一个操作中，当一个函数调用执行时，以下信息中哪一个不存储在堆栈的活动记录中？

A.静态和动态地址链接

B.当前的递归深度

C.形式参数

D.执行函数调用后函数应该返回的位置

【答案】B

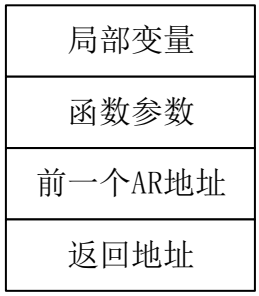
【来源公司】趋势科技（2018）

【考点】函数调用，活动记录

【解题思路】

每当有一个函数调用时，就会在堆栈(Stack)上准备一个被称为 AR(activation recored)的结构，抛开具体编译器实现细节的不同，这个 AR 基本结构如下所示：

因此，活动记录不包含递归深度。


局部变量
函数参数
前一个AR地址
返回地址

1.6 字符串

例题 1.6.1：执行以下语句标准输出的结果是什么？

char str[5] = {‘D’, ’A’, ‘H’, ‘U’, ‘A’};

printf(“length is %d\n”, strlen(str));

【答案】不确定

【来源公司】大华（2017）

【考点】strlen 函数

【解题思路】strlen 函数一般用于计算字符串的长度，当 strlen 碰到‘\0’时，结束判断。但是本题中不能确定字符数组 str 后面的‘\0’会出现在何处。

例题 1.6.2

const char *p = “abcd”;

const char *q = “ABCD”;

(1)p == q、0 == strcmp(p, q)的区别

(2)p > q、strcmp(p, q) > 0 的区别

【来源公司】大华（2017）

【考点】strcmp 函数

【解题思路】==、>操作符比较的是指针地址的大小，而 strcmp 比较的是指针指向的字符串的字典顺序。

例题 1.6.3

请写出函数 strncpy 的声明和实现。 (注: 不可以调用 C/C++的字符串库函数)

【答案】

```
1  char * strncpy(char *dest, const char *src, size_t n);
2  char * strncpy(char *dest, const char *src, size_t n)
3  {
4      size_t i;
5      for (i = 0; i < n && src[i] != '\0'; i++)
6          dest[i] = src[i];
7      for ( ; i < n; i++)
8          dest[i] = '\0';
9      return dest;
10 }
```

【来源公司】趋势科技（2017）

【考点】strncpy 函数

【解题思路】本题考查对 strncpy 函数理解和基本程序的编写，属于比较简单的类型。

例题 1.6.4

```
#include <stdio.h>
main()
{
  char a[10]={ '1','2','3','4','5','6','7','8','9',0},*p;
  int i;
  i=8;
  p=a+i;
  printf("%s\n",p-3);
}
```

以上程序的输出是？

A．6

B．6789

C．'6'

D．67890

【来源公司】趋势科技（2017）

【考点】字符串

【解题思路】p-3==a+i-3==a+5，故从 a[5]开始输出字符串直到结束，即输出 6789，答案选 B。

例题 1.6.5 下列对字符数组进行初始化的语句正确的是?

A． char a[] =“Hello”;

B． char a[][]={‘H’,‘e’,‘l’,‘l’,‘o’};

C． char a[5]= “Hello”;

D． char a[2][5]={ “Hello”,“World”};

【来源公司】趋势科技（2017）

【考点】字符数组初始化

【解题思路】B 中的二维数组最后一维必须有值，即多维数组只能省略第一维；CD 都超出长度，最后有"\0"

例题 1.6.6 请编写一个函数将字符串 s2 添加到字符串 s1 的末端，函数不受 s1、s2 空间大小的限制。可以利用常用字符串函数 strlen,strcpy,strcat,strcmp,strstr 实现。

常用字符串函数简单描述:

strlen(char *str)：求字符串长度。

strcpy(char *dest, char *src)：把 src 拷贝到 dest。

strcat(char *dest, char *src)：把 src 连接到 dest 后面。

strcmp(char *s1, char *s2)：按照各个字符（ascii）比较 s1 和 s2，相等则返回 0，否则返回 ascii 相减的结果。

strstr(char *s1, char *s2)：在 s1 中查找 s2，返回找到的位置，若找不到则返回 NULL。

【来源公司】趋势科技（2017）

【考点】字符串操作

【解题思路】

```
1   char* append(char*str1,char*str2)
2   {
3       char* temp;
4       //计算 str1 与 str2 总共的和
5       int length=strlen(str1)+strlen(str2);
6       //申请一段足够容下 str1 和 str2 的字符串空间
7       temp=(char *)malloc((length+1)*sizeof(char));
8       //先将 str1 复制到字符指针 temp 指向的内存的起始位置
9       strcpy(temp,str1);
10      //再将 str2 链接到 str1 的位置
11      strcat(temp,str2);
12      return temp;
13  }
```

本题应该重点注意的是第 7 行，必须使用堆上的空间来存储你的结果。如果使用的是普通的数组来存放最终的结果，那么函数返回以后数组内容会被释放，因此返回的 temp 指针也将变成野指针。

例题 1.6.7 以下的 C 标准库函数，哪些是不安全的，需要使用替代函数或者避免使用（）

A、strcpy

B、Gets

C、sprintf

D、strcat

【参考答案】ABCD

【来源公司】CVTE（2017）

【考点】C 标准库函数

【解题思路】以上四个函数均需要对被操作的缓冲区进行修改，而没有任何内存检查机制，因此很容易出现内存溢出的情况。

例题 1.6.8 旧键盘上坏了几个键，于是在敲一段文字的时候，对应的字符就不会出现。现在给出应该输入的一段文字、以及实际被输入的文字，请你列出肯定坏掉的那些键。

输入描述：

输入在 2 行中分别给出应该输入的文字、以及实际被输入的文字。每段文字是不超过

80 个字符的串，由字母 A-Z（包括大、小写）、数字 0-9、以及下划线“_”（代表空格）组成。题目保证 2 个字符串均非空。

输出描述：

按照发现顺序，在一行中输出坏掉的键。其中英文字母只输出大写，每个坏键只输出一次。题目保证至少有 1 个坏键。

示例：

输入	7_This_is_a_test _hs_s_a_es
输出	7TI

【考点】数组，字符串

【参考答案】

```
#include <stdio.h>
#include <wctype.h>
#include <string.h>
int main()
{
    char s1[1000]={0},s2[1000]={0};
    scanf("%s %s",s1,s2);
    unsigned char hash[256] = {0};
    int len_1 = strlen(s1),len_2 = strlen(s2);
    for(int i=0;i<len_2;++i)
    {
    //将 s2 中所有的小写字母转换为大写，其余字符不变
        s2[i] = towupper(s2[i]);
    //按照每个字符对应的 ASSCLL 码，将 hash 数组对应位置 1
        hash[s2[i]]=1;
    }
    for(int i=0;i<len_1;++i)
    {
    //将 s1 中所有的小写字母转换为大写，其余字符不变
        s1[i] = towupper(s1[i]);
    //如果对应 hash 数组的元素为 0，说明对应键丢失
        if(hash[s1[i]]==0)
```

```
            printf("%c",s1[i]);
        //要求只输出一次，因此将对应 hash 数组的元素置 1
            hash[s1[i]]=1;
        }
        return 0;
    }
```

【解题思路】

按照题意，本题首先应该将两个字符序列中的所有小写字母转换为大写字母，使用库函数 towupper 可以轻松做到这点，并且 towupper 对非小写字母不作处理。本题采用一个 hash 数组用于存放 s2 序列中已经出现过的字符的标志位，已经出现的字符对应的位置置 1。寻找丢失的键值时，只需要将 s1 在 hash 数组中遍历，对应 hash 数组的值若为 0，说明丢失。

例题 1.6.9 现在 IPV4 下用一个 32 位无符号整数来表示，一般用点分方式来显示，点将 IP 地址分成 4 个部分，每个部分为 8 位，表示成一个无符号整数（因此不需要用正号出现），如 10.137.17.1，是我们非常熟悉的 IP 地址，一个 IP 地址串中没有空格出现（因为要表示成一个 32 数字）。

现在需要你用程序来判断 IP 是否合法。

输入描述：

输入一个 ip 地址

输出描述：

返回判断的结果 YESorNO

示例：

输入	10.138.15.1
输出	YES

【考点】字符串处理，IP 地址

【参考答案】

```
#include <stdio.h>
int isIP(int ip)
{
    if(ip>=0&&ip<=255)
        return 1;
    else return 0;
}
int main()
```

```
{
    char ch;
    int a[4]={0};
    while(scanf("%d.%d.%d.%d",&a[0],&a[1],&a[2],&a[3])!=EOF)
    {
        if(isIP(a[0])&&isIP(a[1])&&isIP(a[2])&&isIP(a[3]))
            printf("YES\n");
        else
            printf("NO\n");
    }
    return 0;
}
```

例题 1.6.10 采用计算机完成一个表达式的运算，常常需要判别一个表达式中左、右括号是否配对出现，这时实现的算法采用()数据结构最佳

A. 线性表的顺序存储结构

B. 队列

C. 线性表的链式存储结构

D. 栈

【考点】栈

【参考答案】D

【解题思路】

我们都知道，栈（stack）具有后进先出的特点，所以在我们思考一个表达式中的左右括号是否匹配问题时，就自然会想到是不是可以利用栈的特点来判断左右括号是否匹配呢？其基本算法流程如下：

1.扫描整个表达式；

2.判断当前字符是否为括号（左右括号）

①如果不是，则继续扫描下一个字符；

②如果是，则判断当前操作符是否为左括号

若为左括号—>直接入栈。

如果不是左括号，则说明是右括号，这时应该判断栈是否为空。

若栈为空—> 说明此表达式右括号多于左括号。

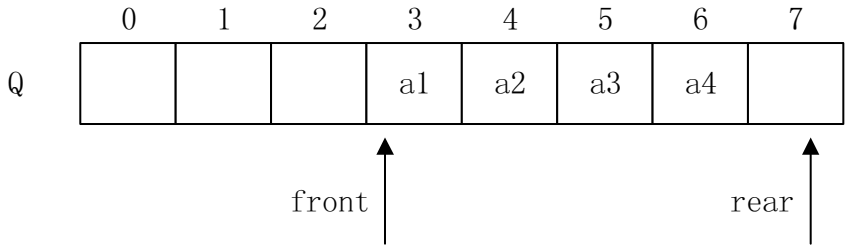
若栈不为空—>判断当前操作符是否和栈顶操作符匹配，若不匹配—->说明左右括号不匹配，若匹配— - >则继续判断下一个操作符。

3.最后，判断栈是否为空

①栈不为空—->说明左括号多于右括号

②栈为空—->说明括号匹配成功。

例题 1.6.11 设循环队列 Q[8]的当前状态如下：



其中 ai(1=<i=<4)为队元素，front,rear 分别为队头和队尾指针（其值为元素的下标）.

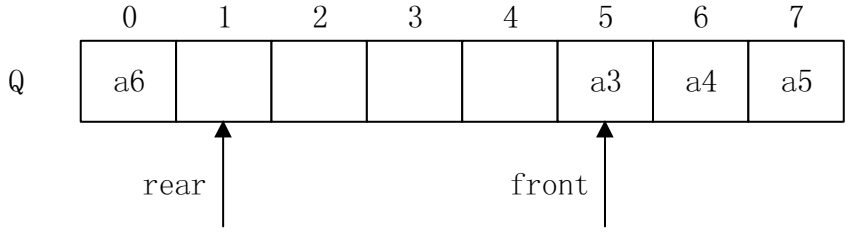
（1）请画出往队 Q 中插入元素 a5,a6,再删除元素 a1,a2 后的状态。

（2）写出队 Q 的队空，队满判定条件以及进队，出队操作 C 语言描述语句。

【考点】循环队列

【参考答案】

（1）



（2）

队空判定条件:front==rear;队满判定条件:(rear+1)%8==front。

进队操作:rear=(rear+1)%8;Q[rear]=x;（x 为进队元素）。

出队操作:front=(front+1)%8;x=Q[rear];（出队元素存入 x）。

【解题思路】

注意循环队列的 front 指针始终指向队头元素，队尾指针始终指向队尾元素的后面一个位置。

1.7 用户自定义数据结构

例题 1.7.1：在 32 位的 x86 机器上，以下结构体的大小为多少字节？

```
struct uname{
        char a[3];
        char *b;
        char c;
        char d;
    };
```

A.11

B.10

C.9

D.12

【答案】D

【来源公司】烽火科技（2017）

【考点】结构体内存对齐

【解题思路】结构体内存对齐遵循两条规则：

1、每个成员的偏移量都必须是当前成员所占内存大小的整数倍，如果不是编译器会在成员之间加上填充字节。

2、当所有成员大小计算完毕后，编译器判断当前结构体大小是否是结构体中最宽的成员变量大小的整数倍，如果不是会在最后一个成员后做字节填充。

所以，当计算该结构体的大小时，我们可以依次将每个成员放入内存中：

Step1:将数组 a 放入内存中；

a[0]	a[1]	a[2]									

Step2:将指针 b 放入内存中，在 32 位系统中，所有指针的长度都为 4；但是指针 b 可以直接紧接着 a[2]放置吗？仔细阅读规则 1，指针 b 的首地址在结构体中的偏移量必须为其本身大小的整数倍，而 a[2]后面的位置在结构体中的位置偏移量为 3，不是 4 的倍数，所以在 a[2]后需要空出一格，才能放置指针 b:

a[0]	a[1]	a[2]	空	b(4 字节)				

Step3:放置 c，c 的大小为 1，根据规则 1，c 可以直接在 b 后面放置:

a[0]	a[1]	a[2]	空	b(4 字节)	c			

Step4:放置 d，d 的大小为 1，根据规则 1，由于 c 元素后面的位置在结构体中的位置偏移量为 9，是 1 的倍数，因此直接放置 d:

a[0]	a[1]	a[2]	空	b(4 字节)	c	d		

Step5：注意，所有成员变量放置完成后，必须执行规则 2，看当前整个结构体的大小是

否为最长的成员的整数倍。当前结构体中最长的成员（数组成员除外）为指针 b，因此整个结构体的长度必须为 4 的倍数，所以需要在 d 后面再加两个字节的空位：

a[0]	a[1]	a[2]	空	b(4 字节)	c	d		

因此，答案为 D，12。

例题 1.7.2：以下程序的运行结果为？

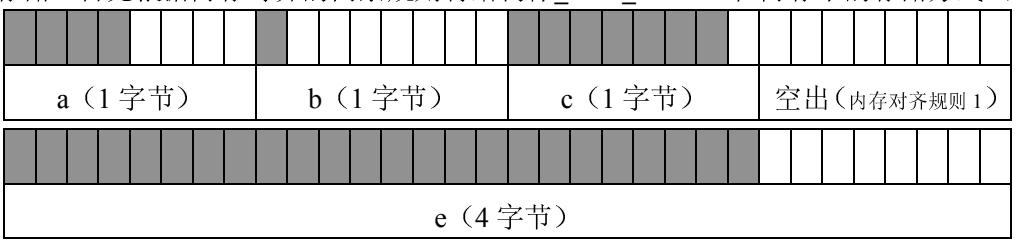
```
#include <stdio.h>
#include <stdlib.h>
typedef struct
{
    unsigned int a:4;
    char b:1;
    char c:7;
    unsigned int e:28;
    unsigned int f:4;
}_TEST_STRUCT;
int main()
{
    TEST_STRUCT stBitFld;
    stBitFld.b = 1;
    stBitFld.c = 0xFFFFFFFF;
    printf("%d, %d\n", (int)sizeof(stBitFld), (int)(stBitFld.b));
    return 0;
}
```

【答案】12，-1

【来源公司】联发科（2017）

【考点】内存对齐，位段

【解题思路】本题综合考察考生对位段和内存对齐的理解，比如结构体中的成员变量 a，默认类型为 int，本应占 4 个字节，但是由于采用位段，强制将其变成 4 个位，但现代计算机存储变量均以字节为单位，因此成员变量 a 实际占用一个字节的内存，并且只使用低四位进行存储。首先根据内存对齐的两条规则将结构体_TEST_STRUCT 在内存中的存储方式画出：



例题 1.7.3：有结构体定义如下：

```
struct Test
{
    int a;
    char s[6];
};
```

赋值合法的有：

A. char a[10]; a[10] = “hello”

B. char a[10] = {‘h’, ‘e’, ‘l’, ‘l’,’o’};

C. struct Test t1 = {1};

D. struct Test t1 = {1, “hello”}; struct Test t2 = t1;

E. int a[10]; int b[10]; a = b

F. char a[6][6]; char *p = a[0]

【答案】BCDF

【来源公司】联发科（2017）

【考点】表达式

【解题思路】数组只有在定义的时候可以用赋值语句直接赋值，因此 A 非法；数组名在大多数情况下可以理解为指针常量，指向数组的第一个元素，而常量之间不可以直接赋值，因此 E 错误。

例题 1.7.4：下面程序的输出是？（）

```c
int main(void)
{
    enum team { my , your = 9 , his , her = his + 3};
    printf("%d %d %d %d\n",my , your , his , her);
    return 0;
}
```

A. 0 9 10 13

B. 8 9 11 13

C. 8 9 10 13

D. 0 9 12 9

【答案】A

【来源公司】趋势科技（2017）

【考点】枚举

【解题思路】第一个枚举成员的默认值为整型的 0，后续枚举成员的值在前一个成员上加 1。枚举型可理解为预处理指令#define 的替代。

例题 1.7.5：有结构体定义如下：

```
struct T
{
    char a;
    int *d;
    int b;
    int c:16;
    double e;
};
T *p;
```

在 64 位系统以及 64 位编译器下，以下描述正确的是 （C）

A. sizeof(p) == 24

B. sizeof(*p) == 24

C. sizeof(p->a) == 1

D. sizeof(p->e) == 4

【答案】C

【考点】结构体，位段

【解题思路】

选项 A 中，p 是一个指针，因此 p 本身的长度为 8 个字节；

选择 B 中，*p 应是整个结构体的长度。根据结构体对齐原则，应为 8 + 8 + 4 + 4 + 8=32;

选项 C 中，结构体 p 中 a 变量是 char 型的，占 1 个字节；

选择 D 中，结构体 p 中 e 变量是 double 类型的，在 64 位系统中占 8 个字节。只有 C 选项正确。

1.8 位操作

例题 1.8.1：已知二进制数 a 是 00101101，如果想通过整型变量 b 与 a 做异或运算使变量 a 的高 4 位取反，地 4 位不变，则二进制数 b 的值应该是（）

A. 11111111

B. 0

C. 1111

D. 11110000

【答案】D

【来源公司】联发科（2017）

【考点】进制运算

【解题思路】本题考察异或算子的概念，考生应牢记任何数（0 和 1）与 1 异或都将取反，与 0 异或是始终不变的这个特性。

例题 1.8.2：一下程序的输出结果是：

```
#include <stdio.h>
#include <string.h>
int main()
{
    int x = 2, y, z;
    x *= (y=z=5);
    z = 3;
    x += (y & z);
    x += (y && z);
    printf("%d\n",x);
    return 0;
}
```

【答案】12

【来源公司】趋势科技（2017）

【考点】位操作，逻辑操作符

【解题思路】本题关键是区分&是位运算，而&&是逻辑与，当 y=5，z=3 时， y&z 的结果就是 101&011 得到 001，y&&z，y 和 z 都不为 0，所以值为 1，最后得到 10+1+1=12

第二章 数据结构基础

2.1 线性表

例题 2.1.1：已知单链表的头指针为 head，请写出函数 int IsLoop(NODE *head)，判断该单链表是否存在环。

【参考答案】

```
1   // 判断链表是否有环
2   #define false 0
3   #define true  1
4   typedef int Status
5   Status IsLoop(NODE *head) // 假设为带头结点的单链表
6   {
7       if (head == NULL)
8           return false;
9
10      NODE *slow = head->next; // 初始时，慢指针从头结点开始走 1 步
11      if (slow == NULL)
12          return false;
13
14      NODE *fast = slow->next; // 初始时，快指针从头结点开始走 2 步
15      while (fast != NULL && slow != NULL)
16      {
17          if (fast == slow)
18              return true;
19
20          slow = slow->next; // 慢指针每次走一步
21
22          fast = fast->next;
23          if (fast != NULL)
24              fast = fast->next;
25       }
26
27      return false;
```

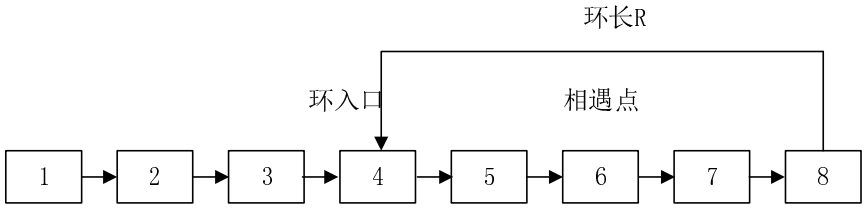
```
28    }
```

【来源公司】中兴（2018）

【考点】快慢指针，单链表

【解题思路】本题考察快慢指针的应用。判断环是否存在的最常用的方法是使用快慢指针，慢指针 slow 从链表头开始，依次往后遍历；而快指针 fast 从链表头开始，每次移动两个结点，即快指针遍历的速度为慢指针的两倍。如果快指针或慢指针遇到 NULL 指针，则说明该单链表不存在环；而如果快慢指针相遇，则说明存在环。

为了方便描述问题，假设有如下单链表：



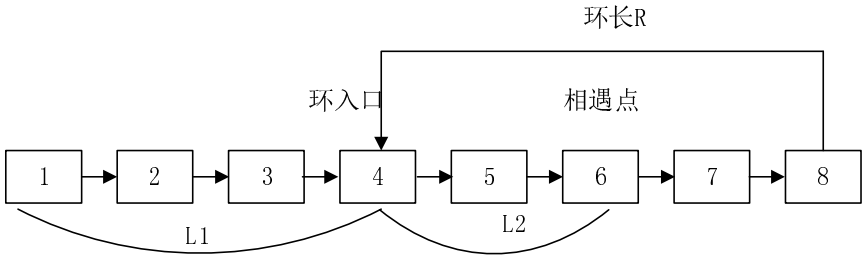
可以看出，该单链表存在长度 R=5 的环，其环入口为 4 号结点。那么，快慢指针的路径为：

slow	fast
1	1
2	3
3	5
4	7
5	4
6	6

两个指针经过 6 步，最终在 6 号结点相遇，则可以判断该单链表存在环。

【考题拓展】

1． 如何通过程序找到环的入口（在上述例子中，也就是找到编号为 4 的结点的地址）？同样以上述例子加以说明，假设头结点和环入口之间的距离为 L_1，环入口和相遇点之间的距离为 L_2，相遇时 slow 指针走过的距离为 S_{slow}，fast 指针走过的距离为 S_{fast}，如下图所示



那么可以得到：

$$S_{slow} = L_1 + L_2 \quad (2.2.1)$$

$$S_{fast} = 2\,S_{slow} \quad (2.2.2)$$

而此时，fast 指针在环中已经走了 $n(n \geq 1)$ 个整圈的距离加上 L_2，因此 S_{fast} 还可以写成：

$$S_{fast} = L_1 + L_2 + nR \quad (2.2.3)$$

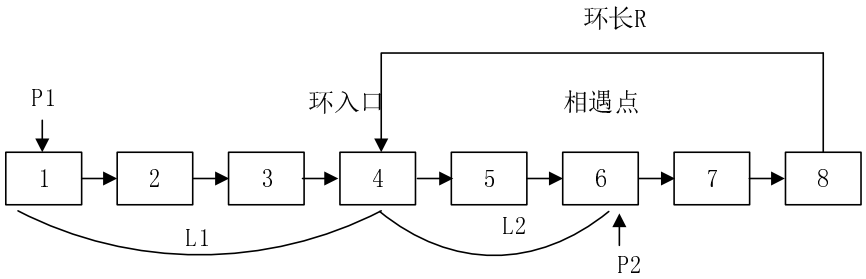
将式（2.2.3）减去式（2.2.1），可以得到：

$$S_{slow} = nR \quad (2.2.4)$$

将式（2.2.4）代入（2.2.1），可得：

$$L_2 + L_1 = nR \quad (2.2.5)$$

式（2.2.5）说明，在 L_2 的基础上，再走 L_1 的长度，便可以达到环路的整圈的倍数，即环的入口处。根据式（2.2.5）提供的思路，我们可以使用两个变量 p1 和 p2 来找到环的入口：



其中 p1 指向链表的第一个结点，p2 指向快慢指针的相遇结点，两个指针以相同的速度每次向后移动一步，当 p1 和 p2 相遇时，相遇的结点即为环的入口。

以上图为例，p1 和 p2 所走的路径为：

p1	p2
1	6
2	7
3	8
4	4

可见，当 p1、p2 指针在 4 号结点处相遇，而 4 号结点恰巧为环的入口。请同学们根据以上的算法，完成程序的编写。

2． 如何求出环的长度 R？

在快慢指针第一次相遇的地方，设置计数器 counter=0，让快慢指针继续运行，slow 指针每走一步，counter 计数器加一，等到快慢指针第二次相遇时，counter 计数器的值就是环的长度了。大家思考一下是为什么呢?也请大家用程序实现该功能。

例题 2.1.2：已知单链表的结点定义：

typedef struct Node{

int data;

Node *next;

} Node, *List;

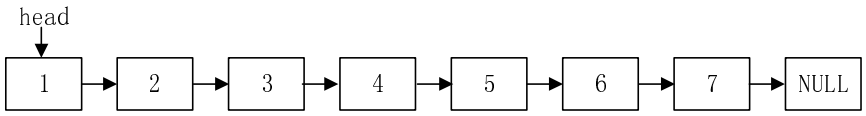
并且单链表的头指针为 head，请写出函数 Node * reverseList(List head)，完成链表的反向。

【来源公司】大疆、中兴（2018）

【考点】单链表，递归

【解题思路】

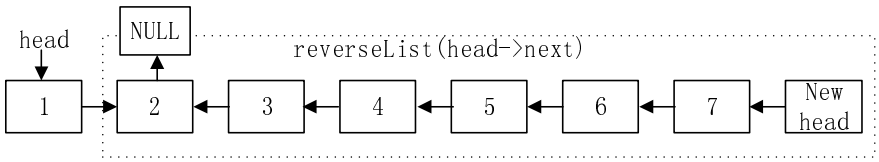
为了便于分析，假设有如下单链表：



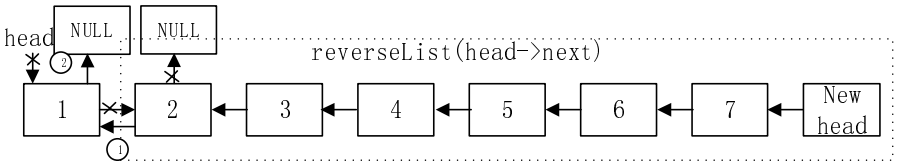
我们的目标是设计一个函数 Node * reverseList(List head)，使得整个链表实现反转，同时返回新链表的头指针。递归的思想是将整个问题的规模减小，那么，我们退一步来思考：假如我们已经先调用反转函数，将除了第一个结点以外的所有结点已经实现了反转，即：

Node *newhead =reverseList(head->next)

则接下来应该如何做呢？



由上图可知，我们应该先将 2 号结点的指针域指向 1 号结点，再让 1 号结点的指针域指向 NULL：



以上就是递归的核心思想，整个代码如下所示：

```
1   Node * reverseList(List head)
2   {
3       //如果链表为空或者链表中只有一个元素
4       if(head == NULL || head->next == NULL)
5       {
```

```
6           return head;
7       }
8       else
9       {
10          //先反转后面的链表，走到链表的末端结点
11          Node *newhead = reverseList(head->next);
12          //再将当前结点设置为后面结点的后续结点
13          head->next->next = head;
14          head->next = NULL;
15          return newhead;
16      }
17  }
```

递归算法显得十分简洁，但是缺点是当单链表较大时，对堆栈的要求比较高。因此，大家思考一下，应该如何使用普通递推的方法完成单链表的反向呢？

例题 2.1.3：下面哪项是数组优于链表的特点？

A. 方便删除

B. 方便插入

C. 长度可变

D. 存储空间小

【答案】D

【来源公司】趋势科技（2017）

【考点】数据和链表的优缺点

【解题思路】ABC 是链表的优势，由于链表中一般需要在每个结点中增设指针域，因此相比数组，链表比较浪费空间。

例题 2.1.4：现有一个带有以下内容的空的哈希表：

Size:7

Startiing index:0

Hash function:f(x)=(3x+4)mod7

当使用封闭散列将序列 1,3,8,10 插入表中时，表的内容是什么？

A. 8,_,_,_,_,_,10

B. 1,8,10,_,_,_,3

C. 1,_,_,_,_,_,3

D. 1,10,8,_,_,_,3

【答案】B

【来源公司】趋势科技（2018）

【考点】哈希表

【解题思路】

f(1)=0，因此 1 放入哈希表下标为 0 的位置；f(3)=6，因此 3 放入哈希表下标为 6 的位置；f(8)=0，发生冲突，则按照封闭散列的规则（f(8)+1）%7=1，放入位置为 1 的地方；f(10)=6，发生冲突，按照封闭散列的规则（f(10)+1）%7=0，依旧冲突，（f(10)+2）%7=1，依旧冲突，（f(10)+2）%7=2，不冲突，因此答案为 B。

例题 2.1.5：以下哪项关于数据结构的陈述是不正确的？

A.数组是紧凑表，是静态的数据结构

B.一个链表中的数据元素不需要存储在内存中相邻的空间中

C.指针域存储链表的下一个数据元素

D.链表是包含数据和指向下一个结点的指针的结点集合。

【答案】C

【来源公司】趋势科技（2018）

【考点】线性表

【解题思路】

C 中，错在指针应该存储的是链表的下一个数据元素的地址。

例题 2.1.6：

输入一个单向链表，输出该链表中**倒数**第 k 个结点，链表的倒数第 1 个结点为链表的尾指针。

输入描述：

第一行：链表结点个数

第二行：链表的所有值，空格隔开

第三行： k 的值

输出描述：

输出一个整数

示例：

输入	输出
8 1 2 3 4 5 6 7 8 4	5

【参考答案】

```
1   #include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include <math.h>
typedef struct node{
    int data;
    struct node *next;
}Node,*List;
//初始化时将头结点中的数据清零
void init(List * L)
{
    *L=(List)malloc(sizeof(node));
    (*L)->data=0;
    (*L)->next=NULL;
}
void insert(List L,int n)
{
    List p=(List)malloc(sizeof(node));
    p->data=n;
    p->next=NULL;
    List l=L;
    while((l->next)!=NULL)
        l=l->next;
    l->next=p;
    L->data++;
}
int main()
{
    int num=0;
    while(scanf("%d",&num)!=EOF)
    {
        int k;
        int i;
        List L1;
        init(&L1);
```

```
36          List p=L1;
37          //依次插入链表数据
38          for(i=0;i<num;i++)
39          {
40              int a;
41              scanf("%d",&a);
42              insert(L1,a);
43          }
44          //读入 K 值
45          scanf("%d",&k);
46          if(k==0)
47          {
48              printf("0\n");
49          }
50          //找到正数第 len-k+1 个结点
51          else
52          {
53              int len=L1->data;
54              for(i=0;i<(len-k+1);i++)
55              {
56                  p=p->next;
57              }
58              printf("%d\n",p->data);
59          }
60          //退出前释放内存
62          for(i=0;i<=num;i++)
63          {
64              List tmp=L1;
65              L1=L1->next;
66              free(tmp);
67          }
68      }
69      return 0;
70  }
```

【考点】带头结点的单链表

【解题思路】

对于单链表，要准确输出倒数第几个结点上的信息，借助头结点中存储的链表长度就能轻松完成。因此，以上的程序使用带头结点的单链表，并且让头结点的数据域存储单链表的长度。

例题 2.1.7：设散列表的长度为 8，散列函数 H(k)=k mod 7，初始记录关键字序列为(32,24,15,27,20,13)，计算用链地址法作为解决冲突方法的平均查找长度是（）

A. 1.5

B. 1.6

C. 1.4

D. 2

【参考答案】A

【考点】哈希表

【解题思路】

序列对应的散列值为 4,3,1,6,6,6

因此 32 查找长度 1,24 查找长度 1，15 查找长度 1,27 查找长度 1,20 查找长度 2，13 查找长度 3。平均查找长度（1+1+1+1+2+3）/6=1.5

例题 2.1.8：下面有关数据结构的说法是正确的？

A. 数组和链表都可以随机访问

B. 数组的插入和删除可以 O(1)

C. 哈希表没有办法做范围检查

D. 以上说法都不正确

【参考答案】B

【考点】线性表

【解题思路】

A 中，数组可以直接通过下标得到存储的值因此支持随机，访问链表是链式存储结构时无法支持随机访问，要访问一个指定位置的元素必须从头开始做指针移动。

C 中哈希表支持直接通过关键码得到值，其实数组就是一种哈希表，下标就是关键码，可通过下标直接得到值，因此哈希表肯定需要做范围检查，也有办法做范围检查。

B 中，如果数组插入和删除都是最后一个元素，时间度复杂度就是 O(1)。

例题 2.1.9：（判断题）线性表中每个元素都有一个直接前驱和一个直接后继()

A. 对

B. 错

【参考答案】B

【考点】线性表

【解题思路】

线性结构的基本特征为:

1．集合中必存在唯一的一个“第一元素”；

2．集合中必存在唯一的一个“最后元素”；

3．除最后一个元素之外，均有唯一的后继；

4．除第一个元素之外，均有唯一的前驱。

例题 2.1.10：以下结构类型可用来构造链表的是（）

A．struct aa{ int a； int * b； }；

B．struct bb{ int a； bb * b； }；

C．struct cc{ int * a； cc b； }；

D．struct dd{ int * a； aa b； }；

【参考答案】B

【考点】线性表的定义

【解题思路】

链表由两部分组成，一是数据域，而是指针域，想构造链表需要有一个指向此结构体的指针，因此选 B

例题 2.1.11：线性表的顺序存储结构是一种()

A. 随机存取的存储结构

B. 顺序存取的存储结构

C. 索引存取的存储结构

D. Hash 存取的存储结构

【参考答案】A

【考点】线性表

【解题思路】

线性表有两种存储结构：

1.顺序存储结构---顺序表。顺序表以数组形式出现，可以取任意下标访问，所以是一种随机存取的存储结构。

2.链式存储结构---链表。链表以链表的形式出现，必须从头开始访问，所以是一种顺序存取的存储结构。

例题 2.1.12：以下关于单向链表说法正确的是

A． 如果两个单向链表相交，那他们的尾结点一定相同

B．快慢指针是判断一个单向链表有没有环的一种方法

C．有环的单向链表跟无环的单向链表不可能相交

D． 如果两个单向链表相交，那这两个链表都一定不存在环

【参考答案】ABC

【考点】线性表

【解题思路】

A.单链表的每个结点都具有唯一的前驱结点和唯一的后继结点，所以当两个单链表存在相交的结点时，这两个链表则同时拥有这个结点，以及这个结点的所有后继结点，当这个公共结点是尾结点时，他们则只含有公共一个结点--尾结点。

B.快慢指针是判断单链表是否有环的一种方法：两个指针，每次移动的步长为 2 叫做快指针，每次移动步长为 1 的指针叫做慢指针。快慢指针同时从头结点出发，当快指针率先到达 NULL 的时候，则说明此单链表中不存在环，当快指针追上慢指针的时候，说明此单链表中存在环。

C.有环的单向链表和无环的单向链表不能相交，因为当相交的时候，无环的单向链表也会被迫存在一个环，只不过这个环的”起点“可能不是原来单向链表的头结点。

D．两个单向链表之间相交可以存在环。

例题 2.1.12：线性表采用链式存储时，结点的存储地址（）

A. 必须是连续的

B. 连续与否均可

C. 必须是不连续的

D. 和结点的存储地址相连续

【参考答案】B

【考点】线性表

【解题思路】

存储地址可以随意分配，链表中有指针域可以找到下一个链表结点的存储地址，连续与不连续都可以使用指针域连起来。

例题 2.1.13：一个长度为 100 的循环链表，指针 A 和指针 B 都指向了链表中的同一个结点，A 以步长为 1 向前移动，B 以步长为 3 向前移动，一共需要同时移动多少步 A 和 B 才能再次指向同一个结点____。

A．99

B．100

C．101

D．49

E．50

F．51

【参考答案】E

【考点】快慢指针

【解题思路】每次移动一步，B 都比 A 多走 2 个结点，而在此相遇时，要求 B 比 A 多走 100

个结点，因此需要 100/2=50 步。

例题 2.1.14：以下与数据的存储结构无关的术语是()

A. 循环队列

B. 链表

C. 哈希表

D. 栈

【参考答案】D

【考点】存储结构

【解题思路】

栈可以是顺序存储，也可以是链式存储，与存储结构无关。循环队列是队列的顺序存储结构，链表是线性表的链式存储结构，用散列法存储的线性表叫散列表，都与存储结构有关。

存储结构是数据的逻辑结构用计算机语言的实现，常见的存储结构有：顺序存储 ， 链式存储 ， 索引存储 ，以及 散列存储 。其中散列所形成的存储结构叫 散列表（又叫哈希表） ，因此哈希表也是一种存储结构。栈只是一种抽象数据类型，是一种逻辑结构，栈逻辑结构对应的顺序存储结构为顺序栈，对应的链式存储结构为链栈，循环队列是顺序存储结构，链表是线性表的链式存储结构。

例题 2.1.15：完成在双向循环链表结点 p 之后插入 s 的操作是（）

A．p->next=s; s->prior=p; p->next->prior=s; s-next=p->next

B．p->next->prior=s; p->next=s; s->prior=p; s->next=p->next

C．s->prior=p; s->next=p->next; p->next=s; p->next->prior=s

D．s->prior=p; s->next=p->next; p->next->prior=s; p->next=s

【参考答案】D

【考点】存储结构

【解题思路】

此类题目主要关注是否断链。先把待插入的结点的两个链安排好，再去调整原来的结点。如果先安排原来的结点的链，则几乎都会产生断链的情况。所以 A,B 不可选，不用多看。C 项中，调整原来结点时，先调整的是 p->next = s，最后还用 p->next->prior,此时的 p->next 已经是 s 了。

例题 2.1.16：下列叙述中正确的是

A. 线性表链式存储结构的存储空间一般要少于顺序存储结构

B. 线性表链式存储结构与顺序存储结构的存储空间都是连续的

C. 线性表链式存储结构可以使连续的,也可以是不连续的

D. 以上说法均错误

【参考答案】C

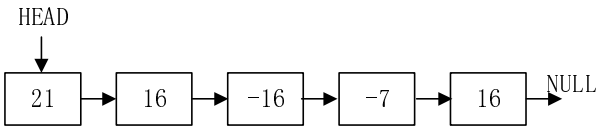
【考点】存储结构

【解题思路】

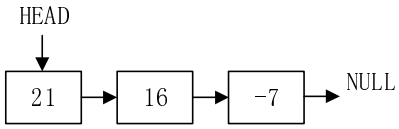
线性表的顺序存储结构要求逻辑关系上相邻的元素在物理位置上也相邻，这样方便了随机存取，但是在插入和删除元素时，需要移动大量元素，而线性表的链式存储则不要求逻辑上相邻的元素在物理位置上也相邻，因此它没有顺序存储结构的可随机存取的优点，不过在插入和删除元素时比较方便。

另外顺序存储每个元素只要存元素的内容，链式存储还需要多一块区域来存储相邻结点的地址，所以线性表链式存储结构的存储空间一般要多于顺序存储结构。

例题 2.1.17：用单链表保存 m 个整数，结点的结构为：[data][link]，且|data|≤n（n 为正整数）。现 要求设计一个时间复杂度尽可能高效的算法，对于链表中 data 的绝对值相等的结点，仅保留第一次出 现的结点而删除其余绝对值相等的结点。例如，若给定的单链表 head 如下：



则删除结点后的 head 为：



要求：

1）给出算法的基本设计思想。

2）使用 C 或 C++语言，给出单链表结点的数据类型定义。

3）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

4）说明你所设计算法的时间复杂度和空间复杂度。

【参考答案】

1）算法的基本设计思想

算法的核心思想是用空间换时间。使用辅助数组记录链表中已出现的数值，从而只需对链表进行一趟扫描。

因为|data|≤n，故辅助数组 q 的大小为 n+1，各元素的初值均为 0。依次扫描链表中的各结点，同时检查 q[|data|]的值，如果为 0，则保留该结点，并令 q[|data|]=1；否则，将该结点从链表中删除。

2）使用 C 语言描述的单链表结点的数据类型定义

```
1  typedef struct node {
```

```
    int data;
    struct node *link;
}NODE;
Typedef NODE *PNODE;
```

3）算法实现

```
void func (PNODE h,int n)
{
    PNODE p=h,r;
    int *q,m;
    // 申请 n+1 个位置的辅助空间
    q=(int *)malloc(sizeof(int)*(n+1));
    // 数组元素初值置 0
    for(int i=0;i<n+1;i++)
    *(q+i)=0;
    while(p->link!=NULL)
    {
        m=p->link->data>0? p->link->data:-p->link->data;
        // 判断该结点的 data 是否已出现过
        if(*(q+m)==0)
        {
            *(q+m)=1;   // 首次出现
            p=p->link;  // 保留
        }
        else           // 重复出现
        {
            r=p->link;  // 删除
            p->link=r->link
            free(r);
        }
    }
    free(q);
}
```

4）参考答案所给算法的时间复杂度为 O(m)，空间复杂度为 O(n)。

【考点】简单算法，线性表元素删除

例题 2.1.20：若线性表最常用的操作是存取第 n 个元素及其前驱和后继元素的值,为节省时间应采用的存储方式()

A. 单链表

B. 双向链表

C. 单循环链表

D. 顺序表

【答案】D

【考点】存储方式

【解题思路】

单链表只有一个指针域，是指向直接后继的。没有指向直接前驱。循环链表也是只指向直接后继。只有双向链表有两个指针域，分别指向直接前驱和后继，但是无法直接定位第 N 个结点。要存取值得修改两个指针顺序表是在计算机内存中以数组的形式保存的线性表，它是数组，不用考虑修改指针，只用修改下标。相比较而言，数组在本题的情况下更加节省时间。

例题 2.1.21：设一个有序的单链表中有 n 个结点，现要求插入一个新结点后使得单链表仍然保持有序，则该操作的时间复杂度（）

A. O(log2n)

B. O(1)

C. O(n2)

D. O(n)

【答案】D

【考点】时间复杂度

【解题思路】

首先要找到插入位置，由于单链表不能像顺序表那样二分查找，因此只能顺序查找，查找的时间复杂度为 O（n）。其次是插入，链表的插入操作时间复杂度是 O（1），因此总的操作时间复杂度为 O(n)。

例题 2.1.22：以下关于链式存储结构说法错误的是

A. 查找结点时链式存储比顺序存储快

B. 每个结点是由数据域和指针域组成

C. 比顺序存储结构的存储密度小

D. 逻辑上不相邻的结点物理上可能相邻

【答案】D

【考点】时间复杂度

【解题思路】

链接存储结构是在计算机中用一组任意的存储单元存储线性表的数据元素。

链式存储结构特点：

1、比顺序存储结构的存储密度小 (每个结点都由数据域和指针域组成，所以相同空间内假设全存满的话顺序比链式存储更多)。

2、逻辑上相邻的结点物理上不必相邻。

3、插入、删除灵活 (不必移动结点，只要改变结点中的指针)。

4、查找结点时链式存储要比顺序存储慢。

5、每个结点是由数据域和指针域组成。

所以 A 选项是错的。

例题 2.1.23：在一个单链表中，q 的前一个节点为 p，删除 q 所指向节点，则执行

A．free q

B．q->next=p->next;free p;

C．p->next=q->next;free p;

D．p->next=q->next;free q;

E．free p;

F．q->next=p->next;free q

【答案】D

【考点】链表的删除

例题 2.1.24：带头结点的单链表 head 为空的判定条件（ ）

A. head==NULL

B. head->next==NULL

C. head->next==head

D. head!=NULL

【答案】B

【考点】带头结点的单链表

例题 2.1.25：给定如下 C 程序：

```
1   typedef struct node_s{
2       int item;
3       struct node_s* next;
4   }node_t;
5   node_t* reverse_list(node_t* head)
6   {
7       node_t* n=head;
8       head=NULL;
9       while(n){
10      _________
```

```
11      }
12      return head;
13  }
```

空白处填入以下哪项能实现该函数的功能？

A．node_t* m=head; head=n; head->next=m; n=n->next;

B．node_t* m=n; n=n->next; m->next=head; head=m;

C．node_t* m=n->next; n->next=head; n=m; head=n;

D．head=n->next; head->next=n; n=n->next;

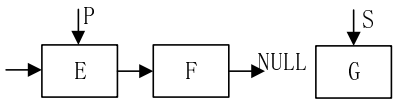
【答案】B

【考点】带头结点的单链表

【解题思路】

本题使用递推完成链表的反向，注意在递推过程中不要出现断链的现象。将各个选项逐个代入验证即可。

例题 2.1.26：若建立以下链表结构,指针 P,S 分别指向如图所示结点：



则不能将 S 所指结点插入到链表末尾的语句组是?

A．p=p-> next; s->next=p;p-> next=s;

B．s->next= '\0'; p=p-> next; p-> next=s;

C．p=p-> next; s ->next=p-> next;p-> next=s;

D．p= (*p).next; (* s).next=(* p).next; (*p).next=s;

【答案】A

【考点】单链表的插入操作

【解题思路】

由于 S 所指向的结点必须为尾结点，因此其指针域应该指向 NULL，因此 A 错误。另外 BCD 中的方法大家也要能够分析出来。

例题 2.1.27：能用二分法进行查找的是

A. 顺序存储的有序线性表

B. 线性链表

C. 二叉链表

D. 有序线性链表

【答案】A

【考点】线性表

【解题思路】

在计算机科学中，折半搜索，也称二分查找算法、二分搜索，是一种在有序数组中查找某一特定元素的搜索算法。搜素过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜素过程结束；如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组为空，则代表找不到。这种搜索算法每一次比较都使搜索范围缩小一半。

二分查找需要满足两点要求：

1.序列有序；

2.可以随机访问；

因此答案选择 A。

例题 2.1.28：两个无环点链表 L1，L2，其长度分别为 m 和 n(m>n),判定 L1,L2 是否相交的时间复杂度最少是____，空间复杂度最少是（不包括原始链表 L1,L2）_____。

【答案】O（m+n）,O(1)

【考点】线性表

【解题思路】

同时遍历两个链表，判断两个遍历指针的值是否相同，相同时或者某一链表遍历结束时退出循环，因此时间复杂度应该为较短的链表的长度。

例题 2.1.29：哈希表的地址区间为 0-8，哈希函数为 H(K)=K mod 9。采用线性探测法处理冲突，并将关键字序列（12,21,43,5,39）依次存储到哈希表中，则元素 39 存放在哈希表中的地址是____。

A． 6

B． 8

C． 1

D． 3

【答案】O（m+n）,O(1)

【考点】哈希表

【解题思路】

12 对应的哈希函数值为 3；21 对应的哈希函数值为 3+1=4；43 对应的哈希函数值为 7；5 对应的哈希函数值为 5；39 对应的哈希函数值为 3+1+1+1=6。

例题 2.1.30：在构造 HASH 表中，通常有哪几种处理冲突的方法？

【答案】开放定址法、再散列函数法、链地址法、公共溢出区法。

【考点】哈希表

例题 2.1.31：设某散列表的长度为 1000，散列函数为除留余数法，H(K)=K%P,则 P 通常情况

下最好选择（）。

A. 997

B. 998

C. 999

D. 1000

【答案】A

【考点】哈希表

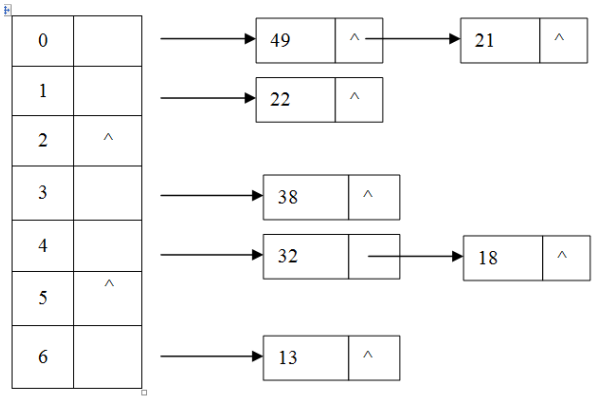
【解题思路】

使用除留余数法的一个经验是，若散列表表长为 m，通常 p 为小于或等于表长（最好接近 m)的最大质数或不包含小于 20 质因子的合数。

例题 2.1.31：设哈希函数 H(k)=K mod 7,哈希表的地址空间为 0～6，对关键字序列{32,13,49,18,22,38,21}，按链地址法处理冲突的办法构造哈希表，并指出查找各关键字需要进行几次比较。

【答案】

用链地址法处理冲突，构造散列表如下：



查找关键字{32,13,49,22,38}的比较次数为 1，查找{18,21}的比较次数为 2。

【考点】哈希表

例题 2.1.32 哈希查找中 k 个关键字具有同一哈希值，若用线性探测法将这 k 个关键字对应的记录存入哈希表中，至少要进行（）次探测

A. k

B. k+1

C. k(k+1)/2

D. 1+k(k+1)/2

【答案】A

【考点】哈希表

【解题思路】

题中问的的是“至少”，那么设表原来为空表。

第一个：直接找到坑，并入坑，1 次；
第二个：和第一个同 hash，找到坑被第一个给占了，找下一个，入坑，2 次；
第三个：第一个被占了，第二个也被占了，找第三个，入坑，3 次；
…
第 k 个：k 次；
一共：1+2+3+…+n=（1+n）*n/2 次

例题 2.1.32 在哈希法存储中，冲突指的是（）

A. 不同关键字值对应到相同的存储地址

B. 两个数据元素具有相同序号

C. 两个数据元素的关键字值不同，而非关键字值相同

D. 数据元素过多

【答案】A

【考点】哈希表

【解题思路】

1.哈希函数：

哈希法又称散列法、杂凑法以及关键字地址计算法等，相应的表成为哈希表。基本思想：首先在元素的关键字 K 和元素的位置 P 之间建立一个对应关系 f，使得 P=f(K),其中 f 成为哈希函数。

创建哈希表时，把关键字 K 的元素直接存入地址为 f(K)的单元；查找关键字 K 的元素时利用哈希函数计算出该元素的存储位置 P=f(K).

创建哈希表时，把关键字 K 的元素直接存入地址为 f(K)的单元；查找关键字 K 的元素时利用哈希函数计算出该元素的存储位置 P=f(K).

2.哈希冲突：

当关键字集合很大时，关键字值不同的元素可能会映像到哈希表的同一地址上，即 K1!=K2，但 f(K1)=f(K2),这种现象称为 hash 冲突，实际中冲突是不可避免的，只能通过改进哈希函数的性能来减少冲突。

例题 2.1.33 一般情况下，建立散列表时难以避免出现散列冲突，常用处理散列冲突的方法之一是开放定址法，该方法的基本思想是什么？

【答案】A

【考点】哈希表

开放定址法就是一旦发生冲突，就去寻找下一个空的散列地址，只要散列表足够大，空的散列地址总能找到，并将记录存入。

Hi=(H(key)+di)%m,i=1,2,…,k(k<=m-1)，其中 H(key)为散列函数，m 为散列表长，di 为增量序列。di 可有下列三种取法：

（1）di=1,2,3,…,m-1，称为线性探测再散列；

（2）di=$\mathbf{1^2,-(1^2),2^2,-(2^2),3^2,\ldots,(-k^2),(k <= m/2)}$, 称二为次探测再散列；

（3）di=伪随机数序列，称为伪随机探测再散列。

所谓伪随机数，用同样的随机种子，将得到相同的数列。

2.3 栈和队列

例题 2.3.1：已知栈的入栈顺序为 abcde，则不可能的出栈序列为：

A． edcba

B．dceab

C．decba

D． abcde

【答案】B

【来源公司】烽火科技，中兴通讯，阿里巴巴（2017）

【考点】栈的性质：先进后出

【解题思路】：注意：本题在入栈过程中可能穿插一些出栈，因此可能情况很多。

A：比较好理解 a,b,c,d,e 依次入栈，e 为栈顶，依次出栈，顺序为 e,d,c,b,a

B：B 选项中先出栈的为 d，因此可以推断 d 出栈时，e 元素还未入栈，此时栈的变化为：

d 出栈前	d 出栈后
d	
c	c
b	b
a	a

此时 c 变成新栈顶；c 出栈后，b 变成新栈顶；

c 出栈前	c 出栈后
c	
b	b
a	a

接着 B 选项中出栈的为 e，因此必须先执行 e 的入栈再执行 e 的出栈：

e 入栈后	e 出栈后
e	
b	b
a	a

此时栈中只剩下 ab，显然必须 b 先出栈后 a 才能执行出栈，因此 B 选项错误。

C：C 选项为 B 选项的正确版本。

D：D 选项的出入栈的顺序为：a 进 a 出，b 进 b 出，c 进 c 出，d 进 d 出，因此 D 正确。

例题 2.3.2：若进栈序列为 1,2,3.4 假定进栈和出栈可以穿插进行，则可能的出栈序列是()

A． 2,4,1,3

B． 3,1,4,2

C． 3,4,1,2

D． 1,2,3,4

【答案】D

【来源公司】CVTE（2017）

【考点】栈的性质：先进后出

【解题思路】参考**例题 2.4.1**。

例题 2.3.3：一个栈的入栈顺序为：ABCDEFG，则可能的出栈顺序有（）

A、CGFEDBA

B、FEGDCBA

C、CGFDEBA

D、ABCDEFG

E、ABCGFED

【参考答案】ABDE

【来源公司】CVTE（2018）

【考点】堆栈

【解题思路】参考**例题 2.4.1**。

例题 2.3.4：假设有一个栈的输入序列为 1,2,3.....，100，若其输出序列的第一个数是 100，则第 40 个是（）

A、59

B、60

C、不确定

D、61

【参考答案】D

【来源公司】CVTE（2017）

【考点】栈的性质

【解题思路】第一个出栈的是 100，说明不存在穿插的情况，因此有序输出就可以。

例题 2.3.5 使用栈实现一个队列的功能，要求给出算法和思路。

【参考答案】

设 2 个栈为 A,B,一开始均为空.

入队操作:将新元素 push 入栈 A;

出队操作:

(1)先判断栈 B 是否为空；

(2)如果为空，则将栈 A 中所有元素依次 pop 出并 push 到栈 B；

(3)将栈 B 的栈顶元素 pop 出；

【考点】栈的性质

例题 2.3.6 下列数据结构具有记忆功能的是？

A. 队列

B. 循环队列

C. 栈

D. 顺序表

【参考答案】C

【考点】栈的性质

【解题思路】

栈的特点是 FILO，后进栈的先出栈，所以你对一个栈进行出栈操作，出来的元素肯定是你最后存入栈中的元素，所以栈有记忆功能。而队列是先进先出，你取队列的第一个元素，得到的是你最先存入队列的元素，而不是上一个存入队列的元素，所以没有记忆功能。

浏览器的后退功能就是用栈实现的，我们在浏览第一个网页 A，点网页里的一个标题，进入网页 B，再在网页 B 里点击一个标题，进入网页 C，这时连续按后退退回网页 A，这说明浏览网页有记忆功能，所以说栈有记忆功能。

例题 2.3.7 对于循环队列()

A. 无法判断队列是否为空

B. 无法判断队列是否为满

C. 队列不可能满

D. 以上说法都不是

【参考答案】D

【考点】循环队列

【解题思路】

队列头尾相接的顺序存储结构称为循环队列。循环队列为空时：rear==front；循环队列为满时：（rear+1）%maxsize==front。

例题 2.3.8 和顺序栈相比,链栈有一个比较明显的优势是()

A. 通常不会出现栈满的情况

B. 通常不会出现栈空的情况

C. 插入操作更容易实现

D. 删除操作更容易实现

【参考答案】A

【考点】栈

【解题思路】

AB 中，因为顺序栈用数组实现，必须事先确定栈的大小，对内存的使用效率并不高，无法避免因数组空间用光而引起的溢出问题：而链栈因为动态申请内存，一般不会出现栈满情况，空栈还是会出现的。

CD：因为都是栈，栈先进后出，只能在栈顶进行插入和删除操作，所以链栈在这点对于顺序栈并无优势。

例题 2.3.9 下面数据结构能够支持随机的插入和删除操作、并具有较好的性能的是____。通常不会出现栈满的情况

A. 数组和链表

B. 链表和哈希表

C. 哈希表和队列

D. 队列和堆栈

E. 堆栈和双向队列

F. 双向队列和数组

【参考答案】B

【考点】线性表

【解题思路】

1，数组是在定义的时候申请一块连续的内存空间，访问某个元素只需要通过下标就可以，但是随机插入和删除都要移动后面所有的元素，所以，数组肯定不行；

2，链表，是非连续的空间，通过指针访问，所以随机插入和删除通过指针之间的操作很方便，但是如果要查询一个数的时候还是得依次便利，但是题目问的是随机插入和删除，所以，链表可以；

3，栈，所有的操作都是在栈顶，如果要随机插入或者删除某个数也必须依次对其他数就行操作，所以，栈也排除；

4，队列，通过队头和队尾指针进行读入数据和删除数据，如果直接在队尾添加数据很方便，但是，题目中是随机，所以，队列排序；

5，哈希表通过键值对操作，只要知道相关的 key 很容易就行读取和删除，插入某个元素也通过 key 很方便，所以，哈希表肯定可以；

例题 2.3.10 用不带头结点的单链表存储队列,其队头指针指向队头结点,队尾指针指向队尾结点,则在进行出队操作时()

A. 仅修改队头指针

B. 仅修改队尾指针

C. 队头、队尾指针都可能要修改

D. 队头、队尾指针都要修改

【参考答案】C

【考点】队列

【解题思路】

多于一个元素时，只需要修改对头指针就行了；但当只有一个元素时，head==rear，此时出队的话，队列就会变成空，需要同时修改对头和队尾指针，不然会超出边界。

例题 2.3.11 在链队列中,即使不设置尾指针也能进行入队操作()

A. 对

B. 错

【参考答案】A

【考点】队列

【解题思路】

若使用不设置尾指针的链表作为链队列的存储结构，在进行入队操作的时候需要遍历整个链队列至队尾，然后在进行插入。这当然是可行的，只是效率有所下降。 如果只使用一个指针又要保持效率的话，可以使用只带尾指针的循环单链表作为存储结构，这样出队和入队的开销都是 O(1)。

例题 2.3.11 对于队列操作数据的原则是（）

A. 先进先出

B. 后进先出

C. 先进后出

D. 不分顺序

【参考答案】A

【考点】队列

例题 2.3.12 在一个链队列中，若 f、r 分别为队首、队尾指针，则插入 s 所指结点的操作为 ( )

A．f->next=s;f=s;

B．r->next=s;r=s;

C．s->next=r;r=s;

D．s->next=f;f=s;

【参考答案】B

【考点】队列

【解题思路】

由于队列只能在队尾进行插入操作，因此应该修改队尾指针，排除 AD 两项；首先将原队尾元素的指针域指向新的队尾，即 r->next=s，再修改队尾指针，使得队尾指针指向新的队尾元素 s，即 r=s。

例题 2.3.13 在具有 n 个单元的顺序存储的循环队列中，假定 front 和 rear 分别为队头指针和队尾指针，则判断队空的条件为_____。

A．rear％n= = front

B．front+l= rear

C．rear= = front

D．(rear+l)％n= front

【参考答案】C

【考点】循环队列

例题 2.3.14 栈和队列的共同特点是()。

A．只允许在端点处插入和删除元素

B．都是先进后出

C．都是先进先出

D．没有共同点

【参考答案】A

【考点】栈和队列

【解题思路】

栈是先进后出，队列是先进先出，不过，两个都只能在端点进行操作，而不能跳过端点直接对中间的值进行操作。

例题 2.3.15 一个队列的入队序列是 1，2，3，4，则队列的输出序列是。

A. 4，3，2，1

B. 1，2，3，4

C. 1，4，3，2

D. 3，2，4，1

【参考答案】B

【考点】队列的性质

【解题思路】

队是先进先出，所以出队顺序跟进来顺序一样。

例题 2.3.16 一个队列的进队顺序是 1,2，....n，若进队和出队可以交替进行，则出队顺序可能是（）

A. 1,2,.....,n

B. 1,2,4,3,5,6,.....,n

C. n,n-1,....,1

D. 以上均有可能

【参考答案】A

【考点】队列的性质

【解题思路】

由于队列在两端进行操作，因此队列进出的顺序不会因为进出的交替进行而有所改变。

例题 2.3.17 设数组 Data[0..m]作为循环队列 SQ 的存储空间，front 为队头指针，rear 为队尾指针，则执行出队操作的语句为（）。

A．front=front+1

B．front=(front+1)% m

C．rear=(rear+1)%m

D．front=(front+1)%(m+1)

【参考答案】D

【考点】循环队列

【解题思路】

1.队空条件：rear==front

2.队满条件：(rear+1) %QueueSIze==front，其中 QueueSize 为循环队列的最大长度

3.计算队列长度：（rear-front+QueueSize）%QueueSize

4.入队：（rear+1）%QueueSize

5.出队：（front+1）%QueueSize

这里出队（front+1）%QueueSize，QueueSize 是数组大小，也就是 m+1。

例题 2.3.18

若用一个大小为 6 的数组来实现循环队列，且当前 rear 和 front 的值分别为 0 和 3 ，当从队列中删除一个元素，再加入两个元素后， rear 和 front 的值分别为 。

A．1 和 5

B．2 和 4

C．4 和 2

D．5 和 1

【参考答案】B

【考点】循环队列

【解题思路】

大小为 6 的数组：下标从 0-5;从前面出队,从后面入队。front(前)=3。rear(后)=0。当出队列中删除一个元素,也就是出队,即 front+1:=4。再插入两个元素,即 rear+2=2。

例题 2.3.19 设有一个顺序循环队列中有 M 个存储单元，则该循环队列中最多能够存储_____

个队列元素；当前实际存储_____个队列元素（设头指针 F 指向当前队头元素的前一个位置，尾指针 R 指向当前队尾元素的位置）。

【参考答案】M-1；（R-F+M）%M

【考点】循环队列

例题 2.3.20 解决计算机与打印机之间速度不匹配问题，须要设置一个数据缓冲区，应是一个_____结构。

【参考答案】队列

【考点】队列

例题 2.3.21 一个输入受限的双端队列（即仅允许一端输入，但两端都可以输出），当输入的序列）是（1,2,3,4），不可能得到的输出序列是（）

A.（1,3,2,4）

B.（1,4,2,3）

C.（4,2,3,1）

D.（4,3,2,1）

【参考答案】C

【考点】栈和队列

例题 2.3.22 将一棵二叉树的根结点放入队列，然后循环地执行如下操作：将队头元素出队列，并将出队结点的所有子结点加入队中，直到队列为空为止。以上操作可以实现哪种遍历？

A. 前序遍历

B. 中序遍历

C. 后序遍历

D. 层序遍历

【参考答案】D

【考点】栈和队列

【解题思路】

根节点出队，子结点入队，则队列中恰好为第一层的所有节点，将第一层节点依次出队，子结点入队，队列中为第二层所有节点，以此类推为层序遍历

附：层序遍历参考代码

```
1    void Levelvisit(BiTree proot)
2    {
3        Node *queue[MAX];
4        int front,rear;
5        front=rear=0;
```

```
6         if(proot!=NULL)
7         {
8             //让根结点的地址入队列
9             queue[rear]=proot;
10            rear=(rear+1)%MAX;
11        }
12        while(front!=rear)
13        {
14            printf("%c",queue[front]->data);
15            //队头出队列之前，需要保留其左右孩子的地址
16            if(queue[front]->lcld!=NULL)
17            {
18                queue[rear]=queue[front]->lcld;
19                rear=(rear+1)%MAX;
20            }
21            //如果左孩子存在，将左孩子入队列
22            if(queue[front]->rcld!=NULL)
23            {
24                queue[rear]=queue[front]->rcld;
25                rear=(rear+1)%MAX;
26            }
27            //如果右孩子存在，将右孩子入队列
28            front=(front+1)%MAX;
29        }
30    }
```

例题 2.3.21 下面（）数据结构常用于函数调用。

A. 队列

B. 栈

C. 链表

D. 数组

【参考答案】D

【考点】栈和队列

【解题思路】

因为先调用的函数是最后返回的，栈是先进后出，不正好符合函数调用吗。比如：

```
intf2()
{
    ....
    return;
}
intf1(){
    .....
    f2();
    return;
}
main(){
    .....
    f1();
    return 0;
}
```

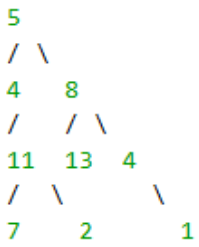
这里是 main 先执行，然后 f1，然后是 f2。但是 f2 先返回，然后是 f1,最后是 main。

2.5 树

例题 2.5.1：给出一个二叉树,用一个函数确定是否有一条从根结点到叶子结点的路径，这个路径上所有结点的值加在一起等于给定的 sum 的值。函数声明 hasPathSum 已经给出，写出程序设计思路并且实现该函数。

```
/**
* 二叉树结点定义如下:
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
* };
*/
int hasPathSum(TreeNode *root, int sum) {
}
```

示例：给定的二叉树和 sum = 22,



hasPathSum 函数的返回值是 1, 即存在一条 root-to-leaf 的路径 5->4->11->2 结点值相加等于 22

【来源公司】趋势科技（2017）

【考点】二叉树，递归

【解题思路】:

本题使用递归的思想来解题会比较轻松。基本思路为:首先判断根结点是否为叶子结点，若是，则判断根结点的值是否为 sum；否则，对左子树进行判断，此时调用递归函数 hasPathSum，但注意其输入值 sum 修改为 sum-root->val；若左子树存在该路径，则不对右子树进行判断，否则仍需要对右子树进行判断。其代码实现如下所示：

```
int hasPathSum(TreeNode *root, int sum)
{
    if(root==NULL)
        return 0;
```

```
5      if(root->left==NULL&&root->right==NULL&&sum==root->val)
6          return 1;
7      return hasPathSum(root->left,sum-root->val) ||
8      hasPathSum(root->right,sum-root->val);
9  }
```

例题 2.5.2：从结点数为 n 的二叉搜索树中查找一个元素时，其时间复杂度大致为（　）

A． O(n)

B． O(1)

C． O(log2n)

D． O(n2)

【答案】C

【来源公司】趋势科技（2017）

【考点】二叉树的性质

【解题思路】二叉查找树（Binary Search Tree），也称有序二叉树（ordered binary tree）,排序二叉树（sorted binary tree），是指一棵空树或者具有下列性质的二叉树：

1. 若任意结点的左子树不空，则左子树上所有结点的值均小于它的根结点的值；
2. 若任意结点的右子树不空，则右子树上所有结点的值均大于它的根结点的值；
3. 任意结点的左、右子树也分别为二叉查找树。
4. 没有键值相等的结点（no duplicate nodes）。

因此二叉查找树为有序树，所查找的值要么在根结点，要么在左子树或者右子树，并且严格地从上往下查找，因此最多遍历整棵树的深度，由于结点数为 n，因此树的深度约为 $\log_2 n$，查找的时间复杂度最多为 $\log_2 n$。

例题 2.5.3：具有 1000 个结点的二叉树的最小深度为（）(第一层深度为 1）

A． 11

B． 12

C． 9

D． 10

【答案】D

【来源公司】CVTE（2017）

【考点】树的深度

【解题思路】

相同结点时，深度最小时二叉树肯定是满二叉树，深度为 n 的满二叉树结点为 2 的 n 次方减 1，由 2 的 9 次方=512,、2 的 10 次方=1024，则具有 1000 个结点的二叉树深度最小为 10。

例题 2.5.4：设非空二叉树中度数为 0 的结点数为 n0，度数为 1 的结点数为 n1,度数为 2 的结点数为 n2,则下列等式成立的是（）

A． n0=n1+n2

B． n0=2n1+1

C． n0=n2+1

D． n0=n1+1

【答案】C

【来源公司】CVTE（2017）

【考点】树的性质，度的概念

【解题思路】

性质：任何一棵二叉树 T，如果其终端结点数为 n0，度为 2 的结点数为 n2，则 n0=n2+1。

推导：一方面，树的总结点数为 n=n0+n1+n2；另一方面，树的总分支线数为 n-1（除根结点外，每个结点头上都有一个分支线）=2*n2+1*n1。

综合以上二式：n0=n2+n1

例题 2.5.5：一棵深度为 4 的三叉树，最多有多少个结点？（）

A． 24

B． 40

C． 36

D． 54

【答案】B

【来源公司】CVTE（2017）

【考点】树的结点数

【解题思路】

最多时为满三叉树，层数从 1 到 4 分别为：1,3,9,27,所以总和 40。

例题 2.5.6：关于二叉树，下面说法正确的是()

A． 二叉树中至少有一个结点的度为 2

B． 一个具有 1025 个结点的二叉树，其高度范围在 11 到 1025 之间

C． 对于 n 个结点的二叉树，其高度为 nlogn

D． 二叉树的先序遍历是 EFHIGJK,中序遍历为 HFIEJKG,该二叉树的右子树的根为 G

【答案】BD

【来源公司】CVTE（2017）

【考点】二叉树的性质

【解题思路】

A 选项中，由于二叉树可以为空树。因此 A 错误；

B 选项中，每一层都只有一个结点时树的高度最大，为 1025；当为满二叉树时树的高度最小，根据性质可知，深度为 10 的满二叉树的结点数为 $2^{10}-1=1023$，因此 1025 个结点的二叉树最小深度为 11，B 正确；

C 选项中，为说明二叉树的种类；对于有 n 个结点的满二叉树而言，其深度为 $\log_2(n+1)$。

D 选项中，由先序遍历可知，根结点是 E,所以由中序遍历可知 JKG 是 E 的右子树，再观察右子树的先序遍历顺序，为 GJK，所以显然 G 是右子树的根结点，再观察中序遍历，可以知 JK 是 G 的左子树，D 正确。

例题 2.5.7：若完全二叉树的第七层有 11 个叶结点，则该完全二叉树结点总数最多是（）

A、74

B、233

C、75

D、234

【答案】A

【来源公司】CVTE（2018）

【考点】数据结构

【解题思路】由于题目限定为完全二叉树，因此第七层为最后一层，并且前六层为满二叉树，总数为 $2^6-1+11=74$

例题 2.5.8：如果一个二叉树的前序遍历结果是 abefcgd，下面哪一个不可能是它的中序遍历结果（）

A、ebfagcd　　B、aebfgcd　C、bfaegcd　　D、ebafgcd

【参考答案】

【来源公司】CVTE（2018）

【考点】数据结构

【解题思路】

由题干可知，a 为该二叉树的根；C 选项中，中序遍历说明二叉树的左子树为 bf，右子树为 egcd，这与前序遍历冲突，因此 C 错误。

例题 2.5.9：二进制树中序遍历生成序列 ABCDEFG，后序遍历为 ACBGFED。这个二叉树的左子树中的总结点数为？

A.1

B.2

C.3

D.4

【参考答案】C

【来源公司】趋势科技（2018）

【考点】数据结构

【解题思路】

由后序遍历可知，树的根结点为 D，再看中序遍历，可知左子树由 ABC 组成，个数为 3，选择 C。

例题 2.5.10 请简述完全二叉树和满二叉树的区别。

【参考答案】

完全二叉树，除最后一层可能不满以外，其他各层都达到该层结点的最大数，最后一层如果不满，该层所有结点都全部靠左排。

满二叉树，所有层的结点数都达到最大。

【考点】完全二叉树和满二叉树的区别

例题 2.5.11 在有 n 个结点的二叉链表中，值为非空的链域的个数为（ ）。

A. n-1

B. 2n-1

C. n+1

D. 2n+1

【参考答案】A

【考点】二叉树

【解题思路】

在有 N 个结点的二叉链表中必定有 2N 个链域。除根结点外，其余 N-1 个结点都有一个父结点。所以，一共有 N-1 个非空链域，其余 2N-（N-1）=N+1 个为空链域。

例题 2.5.12 设一棵完全二叉树中有 65 个结点，则该完全二叉树的深度为(___)。

A. 8

B. 7

C. 6

D. 5

【参考答案】B

【考点】二叉树

【解题思路】

由于完全二叉树除去最后一层，剩下的为满二叉树，而对于深度为 6 的满二叉树，其结点数为 $2^6-1=63$，而对于深度为 7 的满二叉树，其结点个数为 $2^7-1=127$，因此可以推断该完全二叉树的深度为 7。

例题 2.5.13 一棵高度为 h 的完全二叉树至少有（）结点。

A. 2 的 h 次方-1

B. 2 的(h-1)次方-1

C. 2 的(h-1)次方

D. 2 的 h 次方

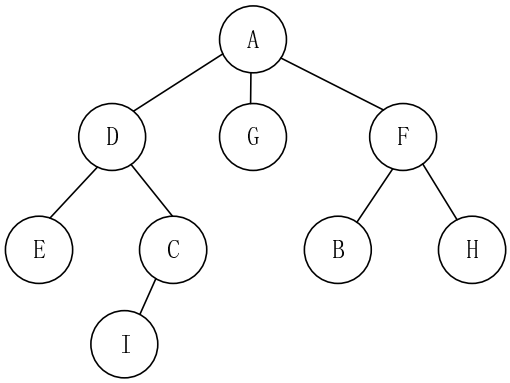
【参考答案】C

【考点】二叉树

【解题思路】

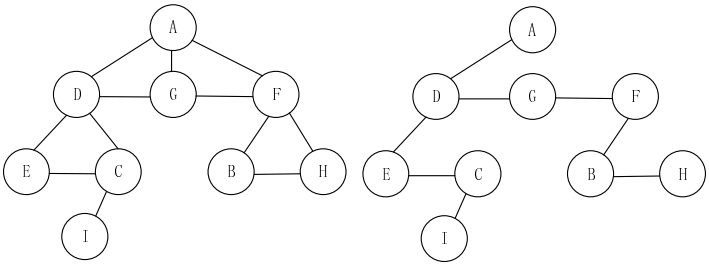
深度为 h-1 的满二叉树的结点数为 2 的 h-1 次方-1，因此深度为 h 的完全二叉树的结点数为至少为 2 的 h-1 次方。

例题 2.5.14 有一棵树，如下图表示，1.请求出该树的后根遍历序列；2.请画出该树对应的二叉树，并求出其先根遍历序列。



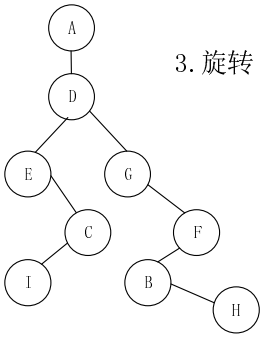
【参考答案】

1. 后后根遍历序列为 EICDGBHFA；
2. 对应的二叉树为



1. 加线

2. 抹线



3. 旋转

其先根遍历为 ADECIGFBH。

【考点】二叉树

【解题思路】

将树转换成二叉树的步骤是：

（1）加线。就是在所有亲兄弟结点之间加一条连线；

（2）抹线。就是对树中的每个结点，只保留他与第一个孩子结点之间的连线，删除它与其它孩子结点之间的连线；

（3）旋转。就是以树的根结点为轴心，将整棵树顺时针旋转一定角度，使之结构层次分明。

例题 2.5.13 设森林 F 由 n 棵树组成,它的第一棵树,第二棵树,…,第 n 棵树分别有 t1,t2,...,tn 个

节点,则与森林 F 对应的二叉树中,根节点的左子树有____个节点。

【参考答案】t1-1

【考点】二叉树

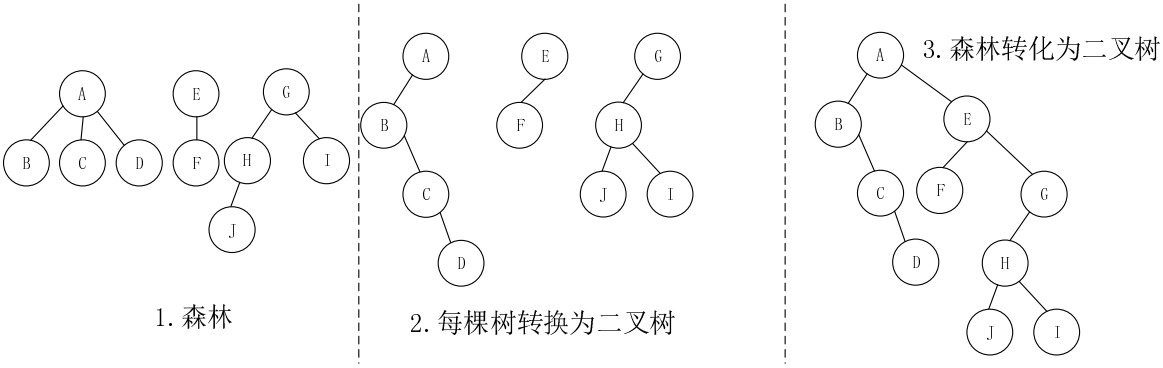
【解题思路】

将森林转换为二叉树的步骤是：

（1）先把每棵树转换为二叉树；

（2）第一棵二叉树不动，从第二棵二叉树开始，依次把后一棵二叉树的根结点作为前一棵二叉树的根结点的右孩子结点，用线连接起来。当所有的二叉树连接起来后得到的二叉树就是由森林转换得到的二叉树。

举例：



由上图可知，根节点的左子树有 t1-1 个节点。

例题 2.5.14 在完全二叉树中，若一个结点是叶结点，则它没（）

A. 左子结点

B. 右子结点

C. 左子结点和右子结点

D. 左子结点、右子结点和兄弟结点

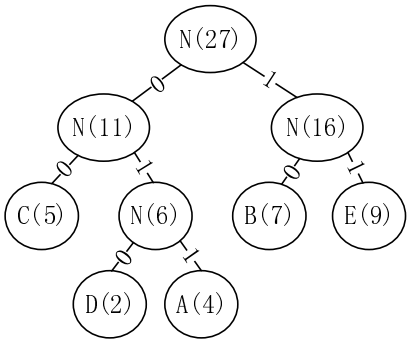
【答案】C

【考点】完全二叉树

例题 2.5.15 设有字符 a、b、c、d、e,它们出现的频率依次为 4、7、5、2、9，试画出对应的 Huffman 树，并求出每个字符的 Huffiman 编码。(编码时用左 0 右 1 规则)

【答案】

按照哈弗曼树的构造规则，对应的哈弗曼树如下所示:



那么，对应的结果为：a:011;b:10;c:00;d:010;e:11;

【考点】哈弗曼树

【解题思路】

一般可以按下面步骤构建：

1，将有权值的叶子结点按从小到大排列

2，取最小的两个权值的叶子结点，作为新结点（N1）的左右结点（左<右）

3，（N1）加入有序序列，使得序列仍然有序。

4，重复步骤以上步骤

例题 2.5.16 判断下列说法是否正确：Huffman 树中，非终端结点的权值是其左右孩子结点的权值之和。（）

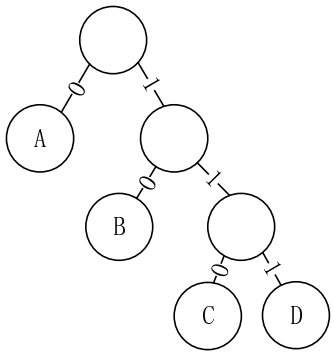
A. 正确

B. 错误

【答案】A

【考点】哈弗曼树

例题 2.5.17 已知如图的 Huffman 树，则电文 CDAA 的编码为____(遵循左 0 右 1 的原则)，若 A,B,C,D 的权值分别为 7,5,2,4，则该树的带权路径长度为____



【答案】11011100，35

【考点】哈弗曼树

【解题思路】

A 的编码为 0；C 的编码为 110；D 的编码为 111，因此 CDAA 为 11011100。

带权路径长度为：7*1+5*2+2*3+4*3=35。

例题 2.5.17 判断下列说法是否正确：若一棵非空二叉树的先序遍历和后序遍历具有相同的结点访问顺序，则它一定是一棵只有根结点的二叉树。（）

A. 正确

B. 错误

【答案】A

【考点】二叉树的遍历

【解题思路】

由于在先序遍历和后序遍历中，根节点分别出现在最前面后最后面，因此若二者相同，那么必然只有一个根节点。

例题 2.5.18 判断下列说法是否正确：已知完全二叉树的第 8 层有 8 个结点，则其叶子结点数是 64 个。

A. 正确

B. 错误

【答案】B

【考点】完全二叉树

【解题思路】

由题意可知，第 8 层为最后一层，因此叶子结点数应该为第 7 层的叶子结点数加第 8 层，为 $2^{7-1}-4+8=68$。

例题 2.5.19 已知二叉树中有 50 个叶子结点，则该二叉树的总结点数至少_____

A．99

B．100

C．102

D．103

【答案】A

【考点】二叉树的性质

【解题思路】

二叉树一个重要的性质为 n0=n2+1，其中 n0 为度为 0（叶子结点）的结点的个数，n2 为度为 2 的结点的个数，由于叶子结点为 50（偶数），完全有可能存在 n1=0 的情况，因此总结点数最少为 n=n0+n1+n2=50+49=99。

例题 2.5.20 假定一棵二叉树的结点个数为 50，则它的最小深度为_____，最大深度为_____。

【答案】6,50

【考点】二叉树的性质

【解题思路】

最小深度一定是一棵完全二叉树，由于 2^5<50<2^6 ，所以最小深度是 5+1；最大深度为一条单链表，即 50。

例题 2.5.21 一棵深度为 6 的满二叉树有_____个分支节点和_____个叶子。

【答案】6,50

【考点】二叉树的性质

【解题思路】

深度为 6 的慢二叉树共有 $2^6-1=63$ 个结点，其中最后一层均为叶子结点，个数为 $2^{6-1}=32$，因此剩余均为分支结点，个数为 63-32=31。

例题 2.5.22 设 n，m 为一棵二叉树上的两个结点，在中序遍历时，n 在 m 前的条件是（）

A. n 在 m 右方

B. n 是 m 祖先

C. n 在 m 左方

D. n 是 m 子孙

【答案】C

【考点】二叉树的遍历

【解题思路】

中序遍历的特点是先遍历左子树，在遍历根节点，再遍历右子树，因此当一个结点在另外一个结点左边时，满足题意。

例题 2.5.23 设与森林 F 对应的二叉树为 B, B 有 m 个结点，B 的根为 p, p 的右子树结点个数为 n,森林 F 中第一棵树的结点个数是().

A. m-n

B. m-n+1

C. n

D. n+1

【答案】B

【考点】树和二叉树的转换

【解题思路】

森林变成二叉树时，将第一棵树的根结点作为整合后大二叉树的根结点，第一棵树的其他结点都转为大二叉树左子树，后面来的 2 到 n 个树都作为大二叉树的右子树。

例题 2.5.24 若一棵完全二叉树有 768 个结点，则该二叉树中叶结点的个数是（）。

A. 257

B. 258

C. 384

D. 385

【答案】C

【考点】树和二叉树的转换

【解题思路】

解法一：由二叉树的性质 $n=n_0+n_1+n_2$ 和 $n_0=n_2+1$ 可知，$n=2n_0-1+n_1$，及 $2n_0-1+n_1=768$，显然 $n_1=1$，$2n_0=768$，则 $n_0=384$。

解法二：完全二叉树的叶子结点只可能出现在最下两层，由题可计算完全二叉树的高度为 10。第 10 层的叶子结点数为 $768-(2^9-1)=257$；第 10 层的叶子结点在第 9 层共有 $[257/2]=129$ 个父结点，第 9 层的叶子结点数为 $2^{9-1}-129=127$，则叶子结点的总数为 257+127=384。

例题 2.5.25 判断下列说法是否正确：用二叉树的前序遍历和后序遍历可以导出二叉树的中序遍历。（）

A. 正确

B. 错误

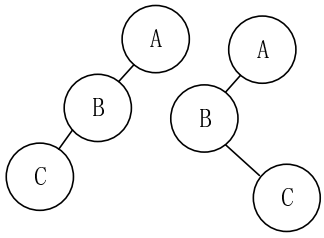
【答案】B

【考点】二叉树的遍历

【解题思路】

性质:已知前序遍历和中序遍历，可确定一棵二叉树，已知后序遍历和中序遍历，可确定一颗二叉树。

而已知前序后后序，无法确定一颗二叉树。已知前序为 ABC，后序为 CBA，那么下图中的二叉树均满足：



例题 2.5.26 假设一棵完全二叉树含有 456 个结点，则度为 0、1、2 的结点个数分别为（　）

A. 227,1,228

B. 228,1,227

C. 228,0,228

D. 不确定

【答案】B

【考点】二叉树的性质

【解题思路】

完全二叉树的性质：

n 个节点的完全二叉树，当 n 为奇数，每一个分支节点都有左右儿子，也就没有度为 1 的点。当 n 为偶数，编号最大的那个分支节点只有左儿子，其他分支节点左右儿子都有，也就是会有一个度为 1 的点。另外，非空二叉树的叶子节点比度为 2 的节点数量多 1，即 n0 = n2 +1 。所以选 B

例题 2.5.26 一棵非空二叉树的前序序列和中序序列正好相同，则该二叉树一定满足_____。

A. 其中任意一结点均无左孩子

B. 其中任意一结点均无右孩子

C. 是一棵完全二叉树

D. 是任意一棵二叉树

【答案】B

【考点】二叉树的性质

【解题思路】

第三章 常用算法

3.1 简单问题编程：

例题 3.1.1：编写一个函数，作用是把一个 char 组成的字符串数组循环右移 n 个。比如原来是 “1234567”，如果 n=2，移位后应该是“6712345”。

【来源公司】CVTE（2018）

【考点】数组的基本操作函数

【解题思路】：重点解决移位一次的实现逻辑，封装成函数 LoopOne，然后可以通过 for 循环实现 n 次的移位，封装成函数 LoopMove。

【参考答案】

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void LoopOne(char *pStr,int len)
{
    if(len<=1)
        return;
    char last=pStr[len-1];
    int i;
    for(i=len-2;i>=0;i--)
    {
        pStr[i+1]=pStr[i];
    }
    pStr[0]=last;
    return;
}
void LoopMove(char *pStr, int steps,int len)
{
    int i;
    for(i=0;i<steps;i++)
        LoopOne(pStr,len);
    return;
```

```
}
int main()
{
    char ppstr[10] = "123456789";
    LoopMove(ppstr,8,strlen(ppstr));
    printf("%s", ppstr);
    return 0;
}
```

例题 3.1.2：不使用库函数，实现内存复制功能 void *mymemcpy(void*dest, const void *src, size_t n)，其功能与 memcpy 函数相同;

【来源公司】CVTE（2018）

【考点】memcpy 函数的理解

【解题思路】:

函数原型：void *mymemcpy(void*dest, const void *src, size_t n);

功能：从源 src 所指的内存地址的起始位置开始，拷贝 n 个字节的数据到目标 dest 所指的内存地址的起始位置中。

说明：

1）src 和 dest 所指内存区域不能重叠，函数返回指向 dest 的指针。如果 src 和 dest 以任何形式出现了重叠，它的结果是未定义的。

2）与 strcpy 相比，memcpy 遇到'\0'不结束，而且一定会复制完 n 个字节。只要保证 src 开始有 n 字节的有效数据，dest 开始有 n 字节内存空间就行。

3）如果目标数组本身已有数据，执行 memcpy 之后，将覆盖原有数据（最多覆盖 n 个）。如果要追加数据，则每次执行 memcpy()后，要将目标地址增加到要追加数据的地址。

4）source 和 destin 都不一定是数组，任意的可读写的空间均可。

【参考答案】

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void *mymemory(void *dst,const void *src,size_t s)
{
    const char* psrc=(char *)src;
    char* pdst=(char *)dst;

    if(psrc==NULL||pdst==NULL)
```

```
10          return NULL;
11      size_t i;
12      if(pdst>psrc&&pdst<(psrc+s))
13      {
14          for(i=s-1;i!=-1;i--)
15              pdst[i]=psrc[i];
16      }
17      else
18      {
19          for(i=0;i<s;++i)
20              pdst[i]=psrc[i];
21      }
22      return dst;
23  }
24  int main()
25  {
26     char buf[100]="abcdefghijk";
27     mymemory(buf+2,buf,5);
28     printf("%s\n",buf+2);
29     return 0;
30  }
```

例题 3.1.3：利用数组实现约瑟夫环的问题。

【来源公司】CVTE（2018）

【考点】数组和递归的结合

【解题思路】：约瑟夫环问题的具体描述是：设有编号为 1,2，……，n 的 n 个（n>0）个人围成一个圈，从第 1 个人开始报数，报到 m 时停止报数，报 m 的人出圈，才从他的下一个人起重新报数，报到 m 时停止报数，报 m 的出圈，……，如此下去，知道剩余 1 个人为止。当任意给定 n 和 m 后，设计算法求 n 个人出圈的次序。

【参考答案】

```
1   #include <stdio.h>
2   #include <string.h>
3   #include <stdlib.h>
4   void yuesefu(int a[],int n,int start,int m)
5   {
```

```
6       if(1==n)
7       {
8           printf("%d\n",a[0]);
9           return ;
10      }
11      int i,count;
12      /*********依次报数，到 m 为止********/
13      for(i=start,count=1;count<m;count++)
14      {
15          i=(1+i)%n;
16      }
17      printf("%d ",a[i]);
18      int next_start=i;
19      /*********剔除空位，调整数组********/
20      int j;
21      for(j=i+1;j<n;j++)
22      {
23          a[j-1]=a[j];
24      }
25      /*********问题规模减小，进行递归调用********/
26      yuesefu(a,n-1,next_start,m);
27  }
28  int main()
29  {
30     int n,m,i;
31     scanf("%d %d",&n,&m);
32     int a[1000];
33     for(i=0;i<n;i++)
34     {
35         a[i]=i+1;
36     }
37     yuesefu(a,n,0,m);
38     return 0;
39  }
```

例题 3.1.4：

“某商店规定：三个空汽水瓶可以换一瓶汽水。小张手上有十个空汽水瓶，她最多可以换多少瓶汽水喝？”答案是 5 瓶，方法如下：先用 9 个空瓶子换 3 瓶汽水，喝掉 3 瓶满的，喝完以后 4 个空瓶子，用 3 个再换一瓶，喝掉这瓶满的，这时候剩 2 个空瓶子。然后你让老板先借给你一瓶汽水，喝掉这瓶满的，喝完以后用 3 个空瓶子换一瓶满的还给老板。如果小张手上有 n 个空汽水瓶，最多可以换多少瓶汽水喝？

输入描述：

输入文件最多包含 10 组测试数据，每个数据占一行，仅包含一个正整数 n（1<=n<=100），表示小张手上的空汽水瓶数。n=0 表示输入结束，你的程序不应当处理这一行。

输出描述：

对于每组测试数据，输出一行，表示最多可以喝的汽水瓶数。如果一瓶也喝不到，输出 0。

示例：

输入	输出
3	1
10	5
81	40
0	

【考点】递归

【参考答案】

```c
#include <stdio.h>
int change bottle(int n)
{
    if(n<=1)
        return 0;
    else if(n==2)
        return 1;
    else
       return n/3+change bottle(n%3+n/3);
}
int main()
{
    int a[10]={0};
    int i=0;
    while(1)
    {
        int n=-1;
        scanf("%d",&a[i]);
        if(a[i]==0)
            break;
        i++;

    }
```

```
    for(i=0;i<=9&&a[i]!=0;i++)
    {
        printf("%d\n",change bottle(a[i]));
    }
    return 0;
}
```

【解题思路】：本题采用递归的方式会比较好解决问题。如上述代码所示，change_bottle 为递归函数。对于 n 个瓶子，首先判断 n 是否小于等于 1，若小于等于 1 则应该返回 0；再判断 n 是否等于 2，等于 2 时可以返回 1；其他情况下，由题意可知，n 个空瓶子只能先换来 n/3 瓶汽水，喝完这 n/3 瓶汽水后，又会产生 n/3 个空瓶，加上之前余下的 n%3 个空瓶，又能够调用 change_bottle 函数进行计算，完成递归。

例题 3.1.5：计算字符串最后一个单词的长度，单词以空格隔开。

输入描述：一行字符串，非空，长度小于 5000。

输出描述：整数 N，最后一个单词的长度。

示例：

输入	输出
Hello world	5

【考点】字符串处理

【参考答案】

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main()
{
    char tmp[5000]={0};
    while(fgets(tmp,sizeof(tmp),stdin)!=NULL)
    {
        int len=strlen(tmp);
        int i;
        int lw_len=0;
        //从后往前遍历，碰到非字母时停止
        for(i=len-1;i>=0;i--)
        {
            if(isalpha(tmp[i]))
                break;
        }
```

```
18          //计算最后一个单词的长度
19          while(isalpha(tmp[i--]))
20          {
21              lw_len++;
22              if(i<0)
23                  break;
24          }
25          printf("%d\n",lw_len);
26      }
27  }
```

【解题思路】：由于题目中已经说明单词以空格隔开，那么只要从后往前遍历，碰到空格时停止，记录下最后一个单词的首字母的下标，然后便可以计算出长度。本题中使用了<ctype.h>文件下的库函数 isalpha，可以用于判断一个字符是否为字母。

例题 3.1.5：

原理：ip 地址的每段可以看成是一个 0-255 的整数，把每段拆分成一个二进制形式组合起来，然后把这个二进制数转变成一个长整数。

举例：一个 ip 地址为 10.0.3.193 每段数字相对应的二进制数

10　00001010

0　00000000

3　00000011

193　11000001

组合起来即为：00001010 00000000 00000011 11000001,转换为 10 进制数就是：167773121，即该 IP 地址转换后的数字就是它了。

输入描述：每次输入两行

第一行：　字符串型 IP 地址

第二行：10 进制型的 IP 地址

输出描述：

第一行：输出转换成 10 进制的 IP 地址

第一行：输出转换后的字符串型 IP 地址

示例：

输入	输出
10.0.3.193 167969729	167773121 10.3.3.193

【考点】字符串处理，位操作

【参考答案】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
//ip_str2int 函数用于将字符串 IP 转换为整型 IP
unsigned int ip_str2int(char *str)
{
    unsigned int a,b,c,d,ip;
    sscanf(str,"%d.%d.%d.%d",&a,&b,&c,&d);
    ip=(a<<24)+(b<<16)+(c<<8)+d;
    return ip;
}
//ip_int2str 函数用于将整型 IP 转换为字符串 IP
char * ip_int2str(unsigned int ip_int)
{
    static char str[100];
    unsigned int a,b,c,d,ip;
    a=(ip_int&0xff000000)>>24;
    b=(ip_int&0x00ff0000)>>16;
    c=(ip_int&0x0000ff00)>>8;
    d=ip_int&0x000000ff;
    sprintf(str,"%d.%d.%d.%d",a,b,c,d);
    return str;
}
int main()
{
    int ip_int;
    char str[100]={0};
    //每次读两行，第一行为字符串，第二行为整数
    while(scanf("%s %u",str,&ip_int)!=EOF)
    {
```

```
        printf("%u\n",ip_str2int(str));
        printf("%s\n",ip_int2str(ip_int));
    }
    return 0;
}
```

【解题思路】

本题关键是设计两个函数，一个用于将字符串型的 IP 地址转换为整型 IP，还有一个将整型 IP 转换为字符串型的 IP 地址。在函数 ip_str2int 中，使用函数 sscanf 可以方便地提取字符串中的四个整型数；在函数 ip_int2str 中，使用位操作可以方便地将整型 IP 中包含的四个 IP 位提取出来。

例题 3.1.6：

Lily 上课时使用字母数字图片教小朋友们学习英语单词，每次都需要把这些图片按照大小（ASCII 码值从小到大）排列收好。请大家给 Lily 帮忙，通过 C 语言解决。

输入描述：

Lily 使用的图片包括"A"到"Z"、"a"到"z"、"0"到"9"。输入字母或数字个数不超过 1024。输入可能包含多组测试数据。

输出描述：

Lily 的所有图片按照从小到大的顺序输出。

示例：

输入	输出
Ihave1nose2hands10fingers	0112Iaadeeefghhinnnorsssv

【考点】基本排序操作

【参考答案】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
void bubble_sort(char a[],int n)
{
    int i,j;
    for(i=1;i<=n-1;i++)
    {
        for(j=0;j<=n-i-1;j++)
```

```
        {
            if(a[j]>a[j+1])
            {
                char tmp;
                tmp=a[j];
                a[j]=a[j+1];
                a[j+1]=tmp;
            }
        }
    }
}
int main()
{
    char str[1024];
    while(scanf("%s",str)!=EOF)
    {
        int len=strlen(str);
        bubble_sort(str,len);
        puts(str);
    }
    return 0;
}
```

【解题思路】

本题实则考察考生对基本排序算法编程的能力。使用基本的冒泡排序便可以完成题目要求。有的企业会直接要求写出冒泡排序算法的代码，以考察考生的基本编程能力。

例题 3.1.7：

有一只兔子，从出生后第 3 个月起每个月都生一只兔子，小兔子长到第三个月后每个月又生一只兔子，假如兔子都不死，问每个月的兔子总数为多少？

输入描述：

输入 int 型表示 month

输出描述：

输出兔子总数 int 型

示例：

输入	输出
9	34

【参考答案】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
int mature_num(int month)
{
    if(month>=0&&month<=2)
        return 0;
    else if(month==3)
        return 1;
    else
        return mature_num(month-1)+mature_num(month-2);
}
int total(int month)
{
    if(month>=0&&month<=2)
        return 1;
    else return total(month-1)+mature_num(month);
}
int main()
{
    int month;
    int num;
    while(scanf("%d",&month)!=EOF)
    {
    num=total(month);
    printf("%d\n",num);
    }
    return 0;
}
```

【考点】递归算法

【解题思路】

本题使用递归算法会比较好解决。递归的关键是找到题目中的递推关系。本题中的递推关系包括两个，一是每个月成熟的兔子的个数，而是每个月兔子的总数。

首先，每个月成熟的兔子应该为多少只？函数 mature_num 递归函数用于计算每个月成熟的兔子的只数。每个成熟的兔子个数=上个月成熟的兔子的个数+新成熟的兔子的个数。而新成熟的兔子的个数的应该为两个月前刚出生的兔子的个数，也就是两个月前所有成熟的兔子的个数（每个成熟的兔子每个月都生一只小兔子），经过两个月后，这些小兔子刚刚成熟。在函数中的体现就是 11、12 行的代码。

接着，再解决每个月兔子的总数的问题就轻松了。每个兔子的总个数=上个月兔子的个数+新出生的兔子的个数。而新出生的兔子的个数就是上个月所有成熟的兔子的个数。因此可以写成 total 函数中的 18 行代码。

例题 3.1.8

扑克牌游戏大家应该都比较熟悉了，一副牌由 54 张组成，含 3~A、2 各 4 张，小王 1 张，大王 1 张。牌面从小到大用如下字符和字符串表示（其中，小写 joker 表示小王，大写 JOKER 表示大王）：

3 4 5 6 7 8 9 10 J Q K A 2 joker JOKER

输入两手牌，两手牌之间用"-"连接，每手牌的每张牌以空格分隔，"-"两边没有空格，如：4444-jokerJOKER。

请比较两手牌大小，输出较大的牌，如果不存在比较关系则输出 ERROR。

基本规则：

（1）输入每手牌可能是个子、对子、顺子（连续 5 张）、三个、炸弹（四个）和对王中的一种，不存在其他情况，由输入保证两手牌都是合法的，顺子已经从小到大排列；

（2）除了炸弹和对王可以和所有牌比较之外，其他类型的牌只能跟相同类型的存在比较关系（如，对子跟对子比较，三个跟三个比较），不考虑拆牌情况（如：将对子拆分成个子）；

（3）大小规则跟大家平时了解的常见规则相同，个子、对子、三个比较牌面大小；顺子比较最小牌大小；炸弹大于前面所有的牌，炸弹之间比较牌面大小；对王是最大的牌；

（4）输入的两手牌不会出现相等的情况。

输入描述：

输入两手牌，两手牌之间用"-"连接，每手牌的每张牌以空格分隔，"-"两边没有空格，如 4 4 4 4-joker JOKER。

输出描述：

输出两手牌中较大的那手，不含连接符，扑克牌顺序不变，仍以空格隔开；如果不存在比较关系则输出 ERROR。

示例：

输入	输出
4 4 4 4-joker JOKER	joker JOKER

【参考答案】

```c
#include <stdio.h>
#include <string.h>
//定义每次出的牌的类型和值
typedef struct
{
    char type;
    int value;
}poker;
//type id 函数根据字符串的长度判断牌的类型
void type id(char *str,char *type,int *value)
{
    switch(strlen(str))
    {
        //长度为 1，说明为单张
        case 1:
        {
            *type='1';
            if(*str=='J')
                *value=11;
            else if(*str=='Q')
                *value=12;
            else if(*str=='K')
                *value=13;
            else if(*str=='A')
                *value=14;
            else
                *value=*str-'0';
            break;
        }
        //长度为 2，说明为单张 10
        case 2:
        {
            *type='1';
            *value=10;
            break;
        }
        //长度为 3，说明为对子
        case 3:
        {
            *type='2';
            if(*str=='J')
                *value=11;
            else if(*str=='Q')
                *value=12;
            else if(*str=='K')
                *value=13;
            else if(*str=='A')
                *value=14;
            else
```

```
50                  *value=*str-'0';
51              break;
52          }
53          //长度为 5，说明为 10 一对或者三张或者单张大小王
54          case 5:
55              {
56                  if(*(str+1)=='0')
57                  {
58                      *type='2';
59                      *value=10;
60                      break;
61                  }
62                  else if(*str=='j')
63                      {
64                          *type='1';
65                          *value=100;
66                          break;
67                      }
68                  else if(*str=='J')
69                      {
70                          *type='1';
71                          *value=1000;
72                          break;
73                      }
74                  else
75                      {
76                          *type='3';
77                          if(*str=='J')
78                              *value=11;
79                          else if(*str=='Q')
80                              *value=12;
81                          else if(*str=='K')
82                              *value=13;
83                          else if(*str=='A')
84                              *value=14;
85                          else
86                              *value=*str-'0';
87                          break;
88                      }
89              }
90              //长度为 7，说明为炸
91          case 7:
92                      {
93                          *type='4';
94                          if(*str=='J')
95                              *value=11;
96                          else if(*str=='Q')
97                              *value=12;
98                          else if(*str=='K')
99                              *value=13;
100                         else if(*str=='A')
101                             *value=14;
102                         else
103                             *value=*str-'0';
104                         break;
105                     }
106             //长度为 8，说明为三张 10
107         case 8:
```

```
                    {
                        *type='3';
                        *value=10;
                        break;
                    }
            //长度为 9，说明为顺子
        case 9:
                    {
                        *type='5';
                        if(*str=='A')
                            *value=1;
                        else
                            *value=*str-'0';
                        break;
                    }
            //长度为 10，说明为顺子中有 10
        case 10:
                    {
                        *type='5';
                        if(*(str+1)!='0')
                        *value=*str-'0';
                        else
                            *value=10;
                        break;
                    }
            //长度为 11，说明为王炸或者四张 10
        case 11:
                    {
                        if(*str=='j'||*str=='J')
                        {
                            *type='4';
                            *value=1000;
                            break;
                        }
                        else
                        {
                            *type='4';
                            *value=10;
                            break;
                        }
                    }

    }
}
int main()
{
    poker poker1;
    poker poker2;
    char string[100]={0};
    gets(string);
    char tmp1[100]={0};
    int i=0;
    //将前面的牌对应的字符串记录在 tmp1 中
    while(string[i]!='-')
    {
        tmp1[i]=string[i];
        i++;
    }
```

```
    //判断第一副牌的类型
    type_id(tmp1,&poker1.type,&poker1.value);
    //将后面的牌对应的字符串记录在 tmp2 中
    char tmp2[100]={0};
    i++;
    int j=0;
    while(string[i]!='\0')
    {
        tmp2[j++]=string[i++];
    }
    //判断第二副牌的类型
    type_id(tmp2,&poker2.type,&poker2.value);
    //类型相同则进行比较
    if(poker1.type==poker2.type)
    {
        if(poker1.value>poker2.value)
            puts(tmp1);
        else puts(tmp2);
    }
    //其中一幅牌为炸时，输出炸
    else if(poker1.type=='4'||poker2.type=='4')
    {
        if(poker1.type=='4')
            puts(tmp1);
        else puts(tmp2);
    }
    else puts("ERROR");
    return 0;
}
```

【解题思路】

本题的关键是用结构体 poker 来表示每手牌的类型和大小。类型 1、2、3、4 分别代表单张、对子、三张和炸弹。type_id 函数根据字符串的长度判断牌的类型，其中应该重点注意 10 和大小王的特殊性（10 占据两个字符，poker 占据 5 个字符）。具体细节可参看代码中的注释。

3.2 常用排序算法

例题 3.2.1：对长度为 n 的线性表进行排序，考虑最糟糕的情况，比较次数不是 n(n-1)/2 的是：

A．冒泡排序

B．选择排序

C．快速排序

D．堆排序

【答案】D

【来源公司】烽火科技（2017）

【考点】各类排序的时间复杂度比较

【解题思路】：在最糟糕的情况下，除了堆排序算法的比较次数是 $O(n\log_2 n)$，其他的都是 n(n-1)/2。

附：各类排序的复杂度及稳定性比较

排序方法	最差时间复杂度	平均时间复杂度	稳定度	空间复杂度
冒泡排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
快速排序	$O(n^2)$	$O(n\log_2 n)$	不稳定	$O(\log_2 n)$~O(n)
选择排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
插入排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
希尔排序	$O(n^2)$	$O(n*\log_2 n)$	不稳定	O(1)
堆排序	$O(n*\log_2 n)$	$O(n*\log_2 n)$	不稳定	O(1)

例题 3.2.2：简单描述快速排序的算法的基本原理，并写出核心代码

【答案】快速排序采用一种分而治之的策略。简单来说，就是先从数列中取一基准数，将比这个数大的数全放在它的右边，小于或者等于它的数放到它的左边，再对左右区间分别进行快速排序，直到各区间只有一个数为止。整个排序过程可以递归进行，以此达到整个数据变成有序序列。其中，分区操作为快速排序的核心操作。

【来源公司】大华（2017）

【考点】快速排序

【参考代码】：

```
1   /*分区操作*/
2   int partition(int p[],int len)
3   {
4       if(len<=1)
5           return 1;
6       int mid=p[0];
```

```
7       int left=0;
8       int right=len-1;
9       while(left<right)
10      {
11          while(p[right]>=mid && left<right)
12              right--;
13          p[left]=p[right];
14          while(p[left]<=mid  && left<right)
15              left++;
16          p[right]=p[left];
17      }
18      p[left]=mid;
19      return left;
20  }
21  /*快速排序*/
22  void Quick_sort(int p[],int len)
23  {
24      int pivot;
25      if(len<=1)
26          return;
27      pivot=partition(p,len);
28      Quick_sort(p,pivot);
29      Quick_sort(p+pivot+1,len-pivot-1);
30  }
```

例题 3.2.3：设一组初始关键字记录关键字为（19,15,12,18,21,36,45,10),则以 19 位基准记录的一趟快速排序结束后的结果为()

A．10,15,12,18,19,36,45,21

B．10,15,12,18,19,45,36,21

C．15,10,12,18,19,36,45,21

D．10,15,12,19,18,45,36,21

【答案】A

【来源公司】CVTE（2017）

【考点】快速排序分区操作

【解题思路】

先从后往前扫描，比 19 小的与 19 交换，再从前往后扫描，比是 19 大的与 19 交换，以此类推，具体可以参照**例题 3.2.2** 中的程序代码。

依次为：

A．19，15，12，18，21，36，45，10//从后往前扫描 10 比 19 小，交换

B．10，15，12,，18，21，36，45，19，从前往后扫描，21 比 19 大，交换

C．10，15，12，18，19，36，45，21//19 前边都比 19 小，后边都比 19 大，一趟比较结束

例题 3.2.4：以下选项中采用分治方法的算法有（）

A． 堆排序算法

B． 插入排序算法

C． 归并排序算法

D． 二分查找算法

E． 快速排序算法

【答案】CDE

【来源公司】CVTE（2017）

【考点】排序思想

例题 3.2.5：一个系统使用快速排序为 1000 个名字排序至少需要 100 秒，请问给 100 个名字排序至少需要多少时间？

A.50.2 秒

B.6.7 秒

C.72.7 秒

D.11.2 秒

【答案】CDE

【来源公司】趋势科技（2017）

【考点】快速排序

【解题思路】

由于快速排序的时间复杂度为 $n\log_2 n$，那么 1000 个数据的时间为：

$k*1000\log_2 1000=100$,由此可以解算出

$$k=\frac{1}{30\log_2 10},$$

那么对于 100 个数据，时间为

$$\frac{1}{30\log_2 10}*100\log_2 100=\frac{20}{3},$$

因此结果选 B。

第四章　Linux 系统应用

4.1 网络基础：

例题 4.1.1：负责将 IP 地址转换成 MAC 地址是什么协议？

【答案】ARP 协议——ARP 协议是“Address Resolution Protocol”（地址解析协议）的缩写。在局域网中，网络中实际传输的是“帧”，帧里面是有目标主机的 MAC 地址的。在以太网中，一个主机要和另一个主机进行直接通信，必须要知道目标主机的 MAC 地址。但这个目标 MAC 地址是如何获得的呢？它就是通过地址解析协议获得的。所谓“地址解析”就是主机在发送帧前将目标 IP 地址转换成目标 MAC 地址的过程。ARP 协议的基本功能就是通过目标设备的 IP 地址，查询目标设备的 MAC 地址，以保证通信的顺利进行。

【来源公司】烽火科技（2017）

【考点】TCP/IP 各层协议名称及作用

例题 4.1.2：什么是 VPN？

【答案】VPN（Virtual Private Network）即虚拟专用网络，属于远程访问技术。VPN 的主要功能是：在公用网络上建立专用网络，进行加密通讯。在企业网络中有广泛应用。VPN 网关通过对数据包的加密和数据包目标地址的转换实现远程访问。VPN 有多种分类方式，主要是按协议进行分类。VPN 可通过服务器、硬件、软件等多种方式实现。

【来源公司】烽火科技（2017）

【考点】VPN

例题 4.1.3：网络中 80 和 8080 端口，一般用于什么协议？

【答案】80 端口和 8080 端口是两种不同的端口。

80 端口是为 HTTP（HyperText Transport Protocol)即超文本传输协议开放的，此为上网冲浪使用次数最多的协议，主要用于 WWW（World Wide Web）即万维网传输信息的协议。可以通过 HTTP 地址（即常说的“网址”）加“:80”来访问网站，因为浏览网页服务默认的端口号都是 80，因此只需输入网址即可，不用输入“:80”了。

8080 端口同 80 端口，是被用于 WWW 代理服务的，可以实现网页浏览，经常在访问某个网站或使用代理服务器的时候，会加上“:8080”端口号。另外 Apache Tomcat web server 安装后，默认的服务端口就是 8080。

【来源公司】国泰新点（2017）

【考点】常用端口号

【考点拓展】

21	FTP（File Transfer Protocol，文件传输协议）服务
23	Telnet（远程登录）服务，是 Internet 上普遍采用的登录和仿真程序。

25	SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）服务器所开放，主要用于发送邮件，如今绝大多数邮件服务器都使用该协议。
53	DNS（Domain Name Server，域名服务器）服务器所开放，主要用于域名解析，DNS 服务在 NT 系统中使用的最为广泛。
69	TFTP 是 Cisco 公司开发的一个简单文件传输协议，类似于 FTP。
80	HTTP（HyperText Transport Protocol，超文本传输协议）开放的，这是上网冲浪使用最多的协议，主要用于在 WWW（World WideWeb，万维网）服务上传输信息的协议。
109、110	109 端口是为 POP2（Post Office Protocol Version 2，邮局协议 2）服务开放的，110 端口是为 POP3（邮件协议 3）服务开放的，POP2、POP3 都是主要用于接收邮件的。
8080	同 80 端口，是被用于 WWW 代理服务的，可以实现网页。

例题 4.1.4：OSI 七层网络模型为？

【答案】OSI 七层网络模型及其主要作用

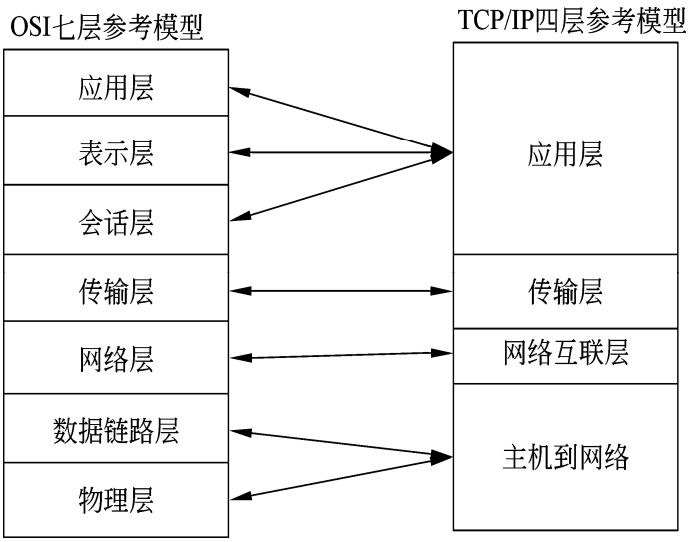
应用层	是应用程序访问网络服务的接口（电子邮件应用程序，浏览器），应用层协议包括 Telnet、FTP、HTTP 等
表示层	1. 数据的编码（.jpg .txt .mp3 等格式） 2. 实现特定功能，如加密
会话层	对应用会话的管理，同步
传输层	1. 决定本次传输传输的方式（TCP/UDP） 2. 错误检测、流量控制 3. 确定本次传输所用的端口号（每个服务都对应一个端口号）
网络层	1. 提供逻辑地址（收件人和发件人的 ip 地址） 2. 路由（由算法完成）
数据链路层	成帧（写入收件人和发件人的 MAC 地址），用 MAC 地址访问媒介，进行错误检测和修正
物理层	设备之间比特流的传输；包括实际物理接口。

【来源公司】国泰新点（2017）

【考点】标准网络模型

【考点拓展】

OSI 七层模型与 TCP 四层参考模型之间的对应关系如下图所示：



例题 4.1.5：交换机和路由器分别工作在 OSI 七层模型的哪两层？。

【答案】一般来说，交换机是小型局域网中端到端通信的枢纽，而对于交换机而言，它是通过 MAC 地址来实现端对端通信的，因此交换机工作于数据链路层；路由器是局域网访问外网的主要媒介，一般来说，路由器需要给同一个局域网中的各个设备分配不冲突的 ip 地址，因此路由器工作于网络层。

【来源公司】国泰新点（2017）

【考点】OSI 七层模型

例题 4.1.6：DNS 是什么？它的作用是什么？

【答案】DNS 是域名系统(Domain Name System)的缩写，它是由解析器和域名服务器组成的。域名服务器是指保存有该网络中所有主机的域名和对应 IP 地址，并具有将域名转换为 IP 地址功能的服务器。其中域名必须对应一个 IP 地址，而 IP 地址不一定有域名。

域名系统采用类似目录树的等级结构。域名服务器为客户机/服务器模式中的服务器方，它主要有两种形式：主服务器和转发服务器。将域名映射为 IP 地址的过程就称为“域名解析”。在 Internet 上域名与 IP 地址之间是一对一（或者多对一）的，域名虽然便于人们记忆，但机器之间只能互相认识 IP 地址，它们之间的转换工作称为域名解析，域名解析需要由专门的域名解析服务器来完成，DNS 就是进行域名解析的服务器。DNS 命名用于 Internet 等 TCP/IP 网络中，通过用户友好的名称查找计算机和服务。当用户在应用程序中输入 DNS 名称时，DNS 服务可以将此名称解析为与之相关的其他信息，如 IP 地址。因为，你在上网时输入的网址，是通过域名解析系统解析找到了相对应的 IP 地址，这样才能上网。其实，域名的最终指向是 IP。

【来源公司】国泰新点（2017）

【考点】DNS 服务器

例题 4.1.7：192.168.1.1/255.255.0.0 子网的可能 IP 主机地址范围是：

【答案】192.168.1.2~192.168.255.254

【来源公司】国泰新点（2017）

【考点】ip 地址

【解题思路】本题应注意的是子网掩码为 255.255.0.0，因此前 16 位 ip 应该固定不变，代表同一个网段，可变的部分为后 16 位，代表同一网段中的不同主机。但是，192.168.1.1 为该网段的路由地址，192.168.255.255 为该网段的广播地址，一般不作为主机地址使用。因此结果为 192.168.1.2~192.168.255.254。

例题 4.1.8：Linux C 进行 TCP 网络编程时，调用 send 发送数据并返回成功，能否表示对端已经接收到数据？如果不能如何才能保证？

【答案】TCP 发送数据的接口有 send，write，sendmsg。在内核中这些函数有一个统一的入口，即 sock_sendmsg()。由于 TCP 是可靠传输，所以对 TCP 的发送接口很容易产生误解，比如 ret = send(...); 错误的认为 ret 的值是表示有 ret 个字节的数据已经发送到了接收端。其实真相并非如此。我们知道，TCP 的发送和接收在在内核中是有对应的缓冲的：

```
struct sock{
...
struct sk_buff_head     receive_queue;     //接收的数据报队列
struct sk_buff_head     write_queue;     //即将发送的数据报队列
...
}
```

对于发送端而言，用户空间调用 send(data)等发送接口将数据发送，内核会将 data 拷贝到内核空间的 socket 对应的缓冲中，即 sock.write_queue。而 send()函数的返回值仅仅是表示本次 send()调用中成功拷贝的字节数（用户空间拷贝至内核空间对应的 sock 缓冲队列）。具体发送和接收端的接收就由 TCP 协议完成，虽然 TCP 是可靠传输，但是这个前提是发送端和接收端的网络是连接的状态。这样，对于调用 send()发送的用户而言，如果想要确定接收方是否成功接受数据，就得需要靠其他的办法查询，比如接收端设置应答机制。

【来源公司】大华（2017）

【考点】TCP 网络编程

【解题思路】本题应注意的是子网掩码为 255.255.0.0，因此前 16 位 ip 应该固定不变，代表同一个网段，可变的部分为后 16 位，代表同一网段中的不同主机。但是，192.168.1.1 为该网段的路由地址，192.168.255.255 为该网段的广播地址，一般不作为主机地址使用。因此结果为 192.168.1.2~192.168.255.254。

例题 4.1.9: 若两台主机在同一个子网中，则两台主机的 IP 地址分别与它们的子网掩码相“与”的结果是?

A． 全 0

B． 全 1

C． 相同

D． 不同

【答案】C

【来源公司】趋势科技（2017）

【考点】子网掩码

【解题思路】子网掩码和主机号相与的结果为网段号，由于两个主机处于同一网段，因此结果相同。比如，子网掩码为 255.255.255.0，主机号分别为 192.168.1.2 和 192.168.1.3，二者与子网掩码相与的结果均为 192.168.1.0。

例题 4.1.10：在浏览器里打开网址 http://www.trendmicro.com，以下哪个协议一定不会被用到?

A. SMTP

B. TCP

C. UDP

D. DNS

E. ARP

【答案】A

【来源公司】趋势科技（2017）

【考点】网络模型，基本网络协议

【解题思路】SMTP，全名 Simple Mail Transport Protocal，译为简单邮件传输协议，也是基于 TCP 连接的一种协议，本题中未提及 html 页面嵌入邮件发送代码，故不考虑。另外，TCP/UDP 是肯定会用到其中之一的，都有可能被用到，DNS 为域名解析，将域名转化为 ip 地址，因此是必然要用到的，ARP 为地址解析协议，将 ip 地址转化为相应的物理地址，因此也是必须要用到的。

例题 4.1.11：当一台 PC 从一个网络移到另一个网络时，以下说法正确的是?

A． 它的 IP 地址和 MAC 地址都会改变

B． 它的 IP 地址可能会改变，MAC 地址不会改变

C． 它的 MAC 地址会改变，IP 地址不会改变

D． 它的 MAC 地址、IP 地址都不会改变

【答案】B

【来源公司】趋势科技（2017）

【考点】IP 地址和 MAC 地址的概念

【解题思路】当一主机移动到另一个网络时，因为各个网络的网络地址不同，因此 IP 地址

会发生改变；而 MAC 地址固化在网卡中，全球惟一，不会发生变化。

例题 4.1.12：将域名转换为 IP 地址是由()服务器完成？

A. WINS

B. DHCP

C. DNS

D. IIS

【答案】C

【来源公司】趋势科技（2017）

【考点】域名解析服务器的作用

【解题思路】WINS 实现的是 IP 地址和计算机名称的映射，它集中管理计算机名称和 IP 地址，作用范围是内网； DNS 实现的是 IP 地址与域名的映射，范围是整个互联网； IIS 是互联网信息服务，是微软公司提供的基于 Windows 的互联网基础服务；DHCP 为动态主机配置协议，局域网的网络协议，使用 UDP 协议工作，主要用于集中地管理，分配 IP 地址，使得网络中的主机动态的获取 IP 地址，网关地址，DNS 服务器地址等信息

例题 4.1.12：下面协议中用于 WWW 传输控制的是？

A． URL

B． SMTP

C． HTTP

D． HTML

【答案】C

【来源公司】趋势科技（2017）

【考点】常用协议

【解题思路】

URL（Uniform/Universal Resource Locator 的缩写，统一资源定位符）是对可以从互联网上得到的资源的位置和访问方法的一种简洁的表示，是互联网上标准资源的地址。 SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议,它是一组用于由源地址到目的地址传送邮件的规则，由它来控制信件的中转方式，SMTP 协议属于 TCP/IP 协议簇，它帮助每台计算机在发送或中转信件时找到下一个目的地。超文本传输协议(HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的 WWW 文件都必须遵守这个标准。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。 HTML 超级文本标记语言是标准通用标记语言下的一个应用，也是一种规范，一种标准，它通过标记符号来标记要显示的网页中的各个部分。网页文件本身是一种文本文件，通过在文本文件中添加标记符，可以告诉浏览器如何显示其中的内容。

例题 4.1.13：（多选题）关于 TCP 协议以下说法正确的是：（）

A． 通讯双方被动关闭的一方进入 TIME_WAIT 状态

B． TIME_WAIT 状态会持续 2 个 MSL

C． TIME_WAIT 状态会持续 1 个 MSL

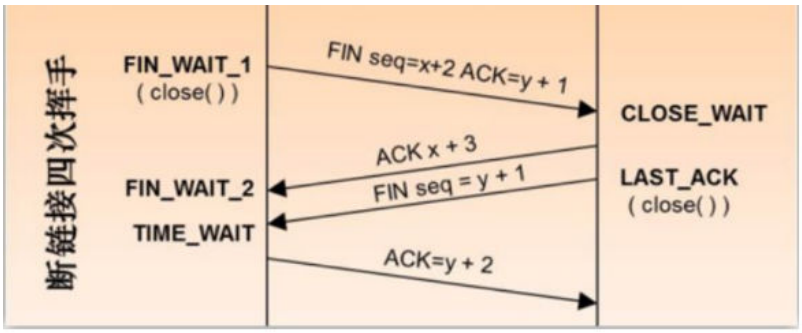
D． 通讯双方主动关闭的一方进入 TIME_WAIT 状态

【答案】BD

【来源公司】CVTE（2017）

【考点】TCP 断开过程

【解题思路】



TCP 断开过程可以描述如下：

假设 Client 端发起中断连接请求，也就是发送 FIN 报文。Server 端接到 FIN 报文后，意思是说"我 Client 端没有数据要发给你了"，但是如果你还有数据没有发送完成，则不必急着关闭 Socket，可以继续发送数据。所以你先发送 ACK，告诉 Client 端，你的请求我收到了，但是我还没准备好，请继续你等我的消息"。这个时候 Client 端就进入 FIN_WAIT 状态，继续等待 Server 端的 FIN 报文。当 Server 端确定数据已发送完成，则向 Client 端发送 FIN 报文，告诉 Client 端，好了，我这边数据发完了，准备好关闭连接了。Client 端收到 FIN 报文后，就知道可以关闭连接了，但是他还是不相信网络，怕 Server 端不知道要关闭，所以发送 ACK 后进入 TIME_WAIT 状态，如果 Server 端没有收到 ACK 则可以重传。Server 端收到 ACK 后，就知道可以断开连接了。Client 端等待了 2MSL 后依然没有收到回复，则证明 Server 端已正常关闭，那好，我 Client 端也可以关闭连接了。Ok，TCP 连接就这样关闭了！

例题 4.1.14：（多选题）TCP 首部报文信息中跟建立链接有关的是（）

A． PSH

B． SYN

C． FIN

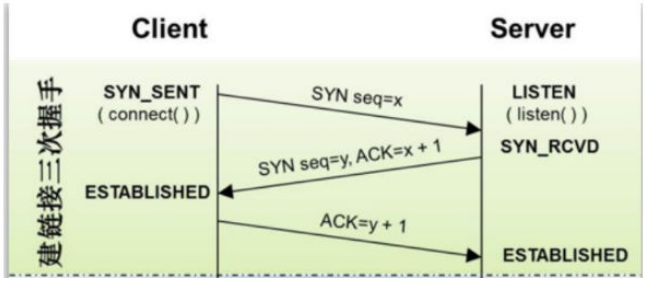
D． ACK

【答案】BD

【来源公司】CVTE（2017）

【考点】TCP 连接过程

【解题思路】TCP 连接过程如下图所示：



第一次：客户端发含 SYN 位，SEQ_NUM = x 的包到服务器。（客 -> SYN_SEND）

第二次：服务器发含 ACK，SYN 位且 ACK_NUM = x + 1，SEQ_NUM = y 的包到客户机。（服 -> SYN_RECV）

第三次：客户机发送含 ACK 位，ACK_NUM = y + 1 的包到服务器。（客 -> ESTABLISH，服 -> ESTABLISH）

例题 4.1.15：IP 地址 205.140.36.68 的哪一部分表示网络号()

A. 205

B. 205.140

C. 68

D. 205.140.36

【答案】D

【来源公司】CVTE（2017）

【考点】网络基础知识

【解题思路】

这是一个 C 类地址，为 192~223 打头，前三个字节表示网段，最后一个字节表示主机号。

例题 4.1.16：在网络安全中，以下哪项用于查找与特定 IP 地址相关联的 MAC 地址？

A.Tracert

B.ARP

C. DNSSPOOF

D.telnet ＜ip＞

【答案】B

【来源公司】趋势科技（2018）

【考点】网络基础知识

【解题思路】

A 选项中，tracert 是一个简单的网络诊断工具，可以列出分组经过的路由结点，以及它在 IP 网络中每一跳的延迟。

B 为 ARP 协议，其作用就是进行 IP 地址和物理地址之间的转换。

C 选项中，dnsspoof 启用 DNS 欺骗，如果是请求解析某个域名，dnsspoof 会让该域名

重新指向另一个 IP 地址(黑客所控制的主机)，如果 dnsspoof 嗅探到局域网内有 DNS 请求数据包，它会分析其内容，并用伪造的 DNS 响应包来回复请求者。如果是反向 IP 指针解析，dnsspoof 也会返回。

D 选项中，telnet ＜ip＞是通过 telnet 协议连接某个主机，与本题无关。

例题 4.1.17：在计算机网络中，下列哪一项是由传输层协议控制？

A.应用到应用之间的通信

B.进程到进程之间的通信

C.点到点之间的通信

D.端到端之间的通信

【答案】D

【来源公司】趋势科技（2018）

【考点】网络基础知识

【解题思路】

传输层一个重要的作用就是确定传输的端口号，因此传输层控制着端到端的通信。另外，网络层可以确定收发的 IP 地址，因此网络层控制着点到点之间的通信。

例题 4.1.18：请解析 IP 地址和对应的掩码，进行分类识别。要求按照 A/B/C/D/E 类地址归类，不合法的地址和掩码单独归类。

所有的 IP 地址划分为 A,B,C,D,E 五类：

A 类地址 1.0.0.0~126.255.255.255;

B 类地址 128.0.0.0~191.255.255.255;

C 类地址 192.0.0.0~223.255.255.255;

D 类地址 224.0.0.0~239.255.255.255；

E 类地址 240.0.0.0~255.255.255.255

私网 IP 范围是：

10.0.0.0～10.255.255.255

172.16.0.0～172.31.255.255

192.168.0.0～192.168.255.255

输入描述：

多行字符串。每行一个 IP 地址和掩码，用~隔开。

输出描述：

统计 A、B、C、D、E、错误 IP 地址或错误掩码、私有 IP 的个数，之间以空格隔开。

示例：

输入	输出
10.70.44.68~255.254.255.0 1.0.0.1~255.0.0.0 192.168.0.2~255.255.255.0 19..0.~255.255.255.0	1 0 1 0 0 2 1

【考点】网络编程基础

【参考答案】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>
//A 类 IP 的范围
#define A_min 1u<<24
#define A_max (127u<<24)-1
//B 类 IP 的范围
#define B_min 128u<<24
#define B_max (192u<<24)-1
//C 类 IP 的范围
#define C_min 192u<<24
#define C_max (224u<<24)-1
//D 类 IP 的范围
#define D_min 224u<<24
#define D_max (240u<<24)-1
//E 类 IP 的范围
#define E_min 240u<<24
#define E_max -1u
//第一种私有 IP 的范围
#define P1_min 10u<<24
#define P1_max (11u<<24)-1
```

```
27  //第二种私有 IP 的范围
28  #define P2_min ((172u<<24)+(16u<<16))
29  #define P2_max ((172u<<24)+(32u<<16)-1)
30  //第三种私有 IP 的范围
31  #define P3_min ((192u<<24)+(168u<<16))
32  #define P3_max ((192u<<24)+(169u<<16)-1)
33  //合法子网掩码判断
34  int isvalidmask(char *p)
35  {
36      struct in_addr ip;
37      int err;
38      unsigned int tmp;
39      //用 inet_pton 初步判断子网掩码是否合法
40      err=inet_pton(AF_INET,p,&ip);
41      if(err>0)
42      {
43          unsigned int a[4]={0};
44          sscanf(p,"%d.%d.%d.%d",&a[0],&a[1],&a[2],&a[3]);
45          if(a[0]==255&&a[1]==255&&a[2]==255&&a[3]==255)
46              return 0;
47          tmp=(a[0]<<24)+(a[1]<<16)+(a[2]<<8)+a[3];
48          tmp=~tmp+1;
49          if((tmp&(tmp-1))==0)
50              return 1;
51          else return 0;
52      }
53      else return 0;
54
55  }
56  int main()
57  {
58  //ABCDE 以及私有 IP 地址 p 类和错误 IP 地址 f 类
59      int a=0,b=0,c=0,d=0,e=0,p=0,f=0;
60      char str[100]={0};
```

```
61      while(scanf("%s",str)!=EOF)
62      {
63
64      char *ip_str,*mask;
65      ip_str=strtok(str,"~");
66      mask=strtok(NULL,"~");
67      if(isvalidmask(mask)==1)
68      {
69          int err;
70          unsigned int ip_int;
71          struct in_addr ip;
72          //首先判断 ip 是否合法
73          err=inet_pton(AF_INET,ip_str,&ip);
74          //分类
75          if(err>0)
76          {
77              unsigned int arr[4]={0};
78  sscanf(ip_str,"%d.%d.%d.%d",&arr[0],&arr[1],&arr[2],&arr[3]);
79              ip_int=(arr[0]<<24)+(arr[1]<<16)+(arr[2]<<8)+arr[3];
80              if(ip_int>=A_min&&ip_int<=A_max)
81                  a++;
82              if(ip_int>=B_min&&ip_int<=B_max)
83                  b++;
84              if(ip_int>=C_min&&ip_int<=C_max)
85                  c++;
86              if(ip_int>=D_min&&ip_int<=D_max)
87                  d++;
88              if(ip_int>=E_min&&ip_int<=E_max)
89                  e++;
90              if(ip_int>=P1_min&&ip_int<=P1_max)
91                  p++;
92              if(ip_int>=P2_min&&ip_int<=P2_max)
93                  p++;
94              if(ip_int>=P2_min&&ip_int<=P2_max)
```

```
95                  p++;
96              }
97              else f++;
98          }
99          else f++;
100
101         }
102         printf("%d %d %d %d %d %d %d\n",a,b,c,d,e,f,p);
103         return 0;
104     }
```

【解题思路】

本题的难点在于如何判断一个子网掩码是否合法。Isvalidmask 函数中，使用了一种简单易行的判断方法。对于合法的子网掩码，比如 255.255.255.0，其取反加 1 后的值为 0.0.1.0；同时 255.255.255.0-1=255.255.254.255；而 0.0.1.0&255.255.254.255=0，因此我们可以总结，对于合法的子网掩码 mask，满足(~mask+1)&(mask-1)=0;另外我们可以利用 inet_pton 库函数的返回值来判断 IP 地址是否合法。另外，strtok 函数在处理有分隔符的字符串分解时，十分方便。

例题 4.1.19：请简要分析 TCP 和 UDP 的区别

【参考答案】

1. TCP 基于连接而 UDP 无连接

2.TCP 要求系统资源较多，UDP 较少；

3.UDP 程序结构较简单

4.流模式（TCP）与数据报模式(UDP);

5.TCP 保证数据正确性，UDP 可能丢包

6.TCP 保证数据顺序，UDP 不保证

4.2 文件 IO：

例题 4.2.1：在使用标准 C 库时, 下面哪个选项使用只读模式打开文件？

A． fopen("foo.txt", "r")

B． fopen("foo.txt", "r+")

C． fopen("foo.txt", "w")

D． fopen("foo.txt", "w+")

E． fopen("foo.txt", "a")

【答案】A

【来源公司】趋势科技（2017）

【考点】fopen 打开模式

【解题思路】

r 为只读方式，该文件必须存在；

r+为读/写方式，该文件也必须存在；

w 打开只写文件，若文件存在则长度清为 0，即该文件内容消失，若不存在则创建该文件；

w+打开可读/写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件；

a 以附加的方式打开只写文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留；

a+ 以附加方式打开可读写的文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾后，即文件原先的内容会被保留。

例题 4.2.2：下列关于 linux IO 描述正确的是（）

A、Linux IO 模型分为阻塞 IO 模型、非阻塞 IO 模型、IO 复用模型、信号驱动和异步 IO。

B、IO 复用模型中 select 和 poll 的原理是顺序扫描遍历所有 fd 集合。

C、使用 IO 复用模型的好处是可以在单线程中监听多个 IO 操作。

D、epoll 优于 select 和 poll 模型的原因在于使用共享内存机制，避免内核空间和用户空间的来回拷贝。

【参考答案】ABCD

【来源公司】CVTE（2018）

【考点】Linux IO 模型

【解题思路】

Linux 五种 IO 模型为：

1)阻塞 I/O：用户进程调用一个 I/O 函数，则进程会一直阻塞，直到数据拷贝完成，I/O

结束。

2)非阻塞 I/O：用户进程调用一个 I/O 函数时，I/O 操作未结束时，进程立马返回一个错误值，反复进行系统调用，直至 I/O 完成。

3) I/O 复用：主要使用 select，poll，epoll 函数，能够实现同时对多个读操作，多个写操作进行 I/O 端口的监听，直到有数据可读或可写时，才调用真正 I/O 操作函数。

4)信号驱动 I/O ：当套接口允许信号驱动 I/O，并安装了信号处理函数，进程不会阻塞，当数据准备好后，进程会收到一个 SIGIO 信号，可以在信号处理函数中调用 I/O 操作函数处理数据。

5)异步 I/O：用户进程调用 I/O 操作后，用户进程可以立即去做别的事情，当 I/O 操作完成后，内核会给用户进程发送通知，告诉用户进程已经完成了。

IO 复用的 3 种方式为：

select：select 函数监视的文件描述符分 3 类，分别是 writefds、readfds、和 exceptfds。调用后 select 函数会阻塞，直到有描述符就绪（有数据 可读、可写、或者有 except），或者超时（timeout 指定等待时间，如果立即返回设为 null 即可），函数返回。当 select 函数返回后，可以通过遍历 fdset，来找到就绪的描述符。

poll 和 select 本质上没有区别，主要区别是 poll 没有限制监听的最大数量。

epoll 主要使用 epoll_create，epoll_ctl, epoll_wait 三个函数。通过 epoll_create 函数创建 epoll 文件描述符，通过 epoll_ctl 注册一个文件描述符，一旦某个文件描述符准备就绪时，内核会采用类似 callback 的回调机制，迅速激活这个文件描述符，当进程调用 epoll_wait 时便得到通知。

三者相比，epoll 具有的显著特点是

1.epoll 不会随着 FD 的增加而降低效率。select 和 poll 都是需要不断轮询所有的 FD 集合，而 epoll 的监听设备就绪时，会调用回调函数，将就绪 FD 放入就绪链表中,epoll_wait 只需判断链表是否为空就可以了，大大节省了 CPU 的时间。

2.select 和 poll 每次调用时都要将需要监听的所有 FD 给 select/poll 系统调用，意味着将用户太态的 socket 列表拷贝到内核态，如果句柄数量过多，将会非常低效。而 epoll 在调用 epoll_wait 时不需要再传递 socket 句柄到内核，因为内核已经在 epoll_ctl 时拿到了需要监听的句柄，实际上在调用 epoll_create 时内核就已经在内核态存储需要监听的句柄，每次调用 epoll_ctl 时只是往数据结构中增添新的句柄，减少了用户态到内核态拷贝的开销。

因此，ABCD 选项都是正确的

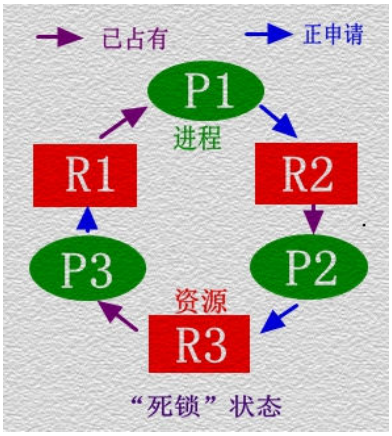
4.3 多进程和多线程

例题 4.3.1：多线（进）程编程中经常出现死锁，造成死锁的必要条件有哪些？如何预防和解除死锁？

【答案】

1.“死锁”的含义

所谓死锁： 是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程，如下图所示。



产生死锁的四个必要条件：

（1） 互斥条件：一个资源每次只能被一个进程使用。即在上图中，R1~R3 只能同时被一个资源所占用。

（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。如上图中，进程 P2 因申请 R3 而阻塞的同时，对持有的 R2 保持不放。

（3） 不剥夺条件:进程已获得的资源，在未使用完之前，不能强行剥夺。

（4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

预防死锁：

预防死锁的方法是使四个必要条件中的第 2、3、4 个条件之一不能成立，来避免发生死锁。至于必要条件 1，因为它是由设备的固有特性所决定的，不仅不能改变，还应加以保证。

1. 摒弃“请求和保持”条件。

在采用这种方法时，系统规定所有进程在开始运行之前，都必须一次性的申请其在整个运行过程所需的全部资源。此时，若系统有足够的资源分配给某进程，便可把其需要的所有资源分配给进程，这样，该进程在整个运行期间便不会再提出资源要求，从而摒弃了请求条件。但在分配资源时，只要有一种资源不能满足某进程的要求，即使其它所需各资源都空闲，

也不分配给该进程，而让该进程等待。由于在该进程的等待期间，它并未占用任何资源，因而也摒弃了保持条件，从而避免发生死锁。

这种方法的缺点是：首先表现为资源被严重浪费，因为一个进程是一次性地获得其整个运行过程中所需的全部资源的，且独占资源，其中可能有些资源很少用，甚至在整个运行期间都未使用，这就严重的恶化了系统资源的利用率；其次是使进程延迟运行，仅当进程在获得了其所需的全部资源后，才能开始运行，但可能因有些资源已长期被其它进程占用而使等待该资源的进程迟迟不能运行。

2. 摒弃“不剥夺”条件

在采用这种方法时系统规定，进程是逐个地提出对资源的要求的。当一个已经保持了某些资源的进程，再提出新的资源请求而不能立即满足时，必须释放它已经保持了的所有资源，待以后需要时再重新申请。这意味着某一进程已经占有的资源，在运行过程中会被占时释放掉，也可认为是被剥夺了，从而摒弃了“不剥夺”条件。

3. 摒弃“环路等待”条件

这种方法中规定，系统将所有资源按类型进行线性排队，并赋予不同的序号。例如，令输入机的序号为 1，打印机的序号为 2，磁带机为 3，磁盘为 4.所有进程对资源的请求必须严格按照资源序号递增的次序提出，这样，在所形成的资源分配图中，不可能再出现环路，因而摒弃了“环路等待”条件。

死锁解除的主要方法有：

1. 资源剥夺法。挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程。但应防止被挂起的进程长时间得不到资源，而处于资源匮乏的状态。

2. 撤销进程法。强制撤销部分、甚至全部死锁进程并剥夺这些进程的资源。撤销的原则可以按进程优先级和撤销进程代价的高低进行。

3. 进程回退法。让一（多）个进程回退到足以回避死锁的地步，进程回退时自愿释放资源而不是被剥夺。要求系统保持进程的历史信息，设置还原点。

【来源公司】烽火科技（2017）

【考点】进程死锁

例题 4.3.2：进程间通信有哪几种方式

【答案】管道、读写锁、socket、信号量、消息队列、共享内存

【来源公司】大华（2017）

【考点】进程间通信

例题 4.3.3：以下四句中正确的叙述为?

A. 操作系统的一个重要概念是进程，不同的进程所执行的代码一定也不同

B. 为了避免发生进程死锁，各进程应逐个申请资源

C. 操作系统用 PCB(进程控制块)管理进程，用户进程可能从 PCB 中读出与本身运行

状态相关的信息

D. 进程同步是指某些进程之间在逻辑上相互制约的关系

【答案】D

【来源公司】趋势科技（2017）

【考点】进程控制、进程同步的概念

【解题思路】

A “一定也不同” 错误。一个简单的例子就是父进程在执行到 fork（）函数时，将会产生子进程，而父子进程所执行的代码段完全相同，但是其拥有的资源、产生的数据、PID 号可能完全不同，因此 A 错误。

B “各进程应逐个申请资源”这句错误。可以通过合理的资源分配算法来确保永远不会形成环形等待的封闭进程链，从而避免死锁。该方法支持多个进程的并行执行，为了避免死锁，系统动态的确定是否分配一个资源给请求的进程。

C “用户进程” 错误，用户进程不能读取状态信息，系统进程（内核）才可以。

D 正确。异步环境下的一组并发进程因直接制约而互相发送消息、进行互相合作、互相等待，使得各进程按一定的速度执行的过程称为进程间的同步。

例题 4.3.4 下面程序一共会在屏幕上输出多少个“-” ？

```
#include<iostream>
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
using namespace std;
int main( )
{
    int i;
    for(i = 0; i < 2; i++)
    {
        cout<<"-\n";
        fork( );
        cout <<"-\n";
    }
    cout << endl;
    return 1;
}
```

【答案】9

【来源公司】趋势科技（2017）

【考点】fork()函数的特性

【解题思路】

将首先执行的执行的进程称作 A，不考虑 fork()，其本身会执行 4 次 cout；A 进程第一次执行到 12 行的 fork()时，产生子进程 B，B 从第 13 行开始执行，一共执行 3 次 cout；A 在第二次执行到 12 行的 fork()时，产生子进程 C，C 只执行一次第 13 行的 cout 便退出 for 循环；B 由于循环未结束，执行到 12 行的 fork()时，产生子进程 D，D 只执行一次第 13 行的 cout 便退出 for 循环；因此结果为 4+3+1+1=9；

例题 4.3.5：（多选题）以下说法正确的有()

A． 多个进程操作同一个文件时，应该要考虑到文件的一致性问题

B． 可通过文件在不同进程间进行数据传递和共享

C． 可以通过全局变量在不同进程间传递数据

D． 一个进程可以访问到所有物理内存空间

【答案】AB

【来源公司】CVTE（2017）

【考点】进程

【解题思路】

C 选项中，只有同一组线程才可以通过全局变量进行数据传递，而不同的进程之间需要通过 IPC 进行通信，也可以通过 B 选项中的文件进行通信；D 选项中，一个进程只能访问到属于它自己的内存范围，而属于其他进程的内存地址以及系统的内存地址却无权访问。

例题 4.3.6：引入多道程序技术以后，处理器的利用率()

A． 降低了

B． 没有变化，只是程序的执行方便了

C． 大大提高

D． 没有影响

【答案】C

【来源公司】CVTE（2017）

【考点】多任务处理

【解题思路】

多道程序技术即计算机内存中同时存放几道相互独立的程序，多道程序在宏观上是并行的：同时进入系统的几道程序都处于运行过程中，即它们先后开始了各自的运行，但都未运行完毕；

多道程序在微观上是串行的：从微观上看，内存中的多道程序轮流地或分时地占有 CPU。

多道程序技术的优点:

1.提高 CPU 的利用率。在多道程序环境下，多个程序共享计算机资源当某个程序等待 I/O 操作时，CPU 可以执行其他程序，大大提高 CPU 的利用率。

2.提高设备的利用率。在多道程序环境下，多个程序共享系统的设备，大大提高系统设备的利用率。

3.提高系统的吞吐量。在多道程序环境下，减少了程序的等待时间，提高了系统的吞吐量。

例题 4.3.7：以下说法正确的有()

A. 在时间片轮询调度算法中，时间片越短则 CPU 利用率越高

B. 优先级越高的进程占用 CPU 的运行时间就一定越多

C. 在遍历大型二维数组 int a[x][y]时，先遍历 x 或先遍历 y 的处理时间都是一样的

D. 使用 cache 可以提高 CPU 的利用率

【答案】D

【来源公司】CVTE（2017）

【考点】编译和体系结构

【解题思路】

选项 A 中，时间片轮询调度是一种古老而又简单的算法，广泛运用于无操作系统的微处理器中。在系统中，每个进程被分配一个时间段，称作时间片，即该进程允许运行的时间。如果在时间片结束时进程还在运行，则 CPU 将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束 ，则 CPU 当即进行切换。调度程序所要做的就是维护一张就绪进程列表，当进程用完它的时间片后，它被移到队列的末尾。

时间片轮询调度中有趣的一点是如何确定时间片的长度。从一个进程切换到另一个进程是需要一定时间的，因为要保存和装入寄存器值及内存映像等保护现场的工作，更新各种表格和队列等。假如进程切换，有时称为上下文切换，需要的时间为 5 毫秒，再假设时间片长度设定为 20 毫秒，则在做完 20 毫秒有用的工作之后，CPU 将花费 5 毫秒来进行进程切换。CPU 时间的 20%被浪费在了管理开销上。进程切换时间一定的情况下，如果时间片长度设定的越小时，这种浪费更明显。所以，时间片长度与 CPU 利用率是一对不可调和的矛盾，必须处理好它们之间的关系。

为了提高 CPU 效率，我们可以将时间片长度设得大一些，这时浪费的时间只有就会相对减小。但在一个分时系统中，各个任务对时间片长度的要求是不一致的。例如在一个系统中，可能要求每秒钟更新一下显示内容，每几十毫秒要扫描一下按键，每几毫秒要检测一下串口缓冲区等。可见，各个任务对时间的依赖程度是不一样的。如果时间片设得太长，某些对实时性要求高的任务可能得不到执行，使得系统的实时性变差。

总之，时间片的设定应满足对实时性要求最高的那个任务，这样才能确保每个任务都可以及时得到执行而不被错过。

B 选项中，进程的优先级越高，说明它获得 CPU 的可能性越大，与时间无关。

C 选项中，由于二维数组是按行顺序进行存储的，所以先遍历 x，再遍历 y 相对要快一点。

例题 4.3.8：有关多线程，多进程的描述错误的是？（）

A． 子进程获得父进程的数据空间，堆和栈的复制品

B． 线程可以与同进程的其他线程共享数据，但是它拥有自己的栈空间且拥有独立的执行序列

C． 线程执行开销小，但是不利于资源管理和保护

D． 进程是 CPU 调度和分派的基本单位。

【答案】D

【来源公司】趋势科技（2017）

【考点】进程和线程

【解题思路】

选 D，进程是操作系统分配资源的基本单位，线程是 cpu 调度的基本单位。

例题 4.3.9：编程中设计并发服务器，使用多进程与多线程 ，请问有什么区别？

【答案】

进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。因此，进程是系统资源分配的最小单位。

线程：在一个进程中，可以有多个线程，同组线程共享全局数据。相对与进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。因此，线程是 CPU 进行调度的最小单位。

两者都可以提高程序的并发度，提高程序运行效率和响应时间。线程和进程在使用上各有优缺点。线程执行开销小，但不利于资源管理和保护；而进程正相反。同时，线程适用于在 SMP(Symmetric Multi-Processing 对称多处理结构)。

【考点】进程和线程的概念

4.5 数据库：

例题 4.5.1：表名为 student，姓名字段为 name，搜索出所有姓“张”的人名的 sql 语句为

【答案】Select name from student where name like “张%”；

【来源公司】国泰新点（2017）

【考点】SQL 语句

【解题思路】本题考查关键字 like 以及通配符%。通配符%表示匹配对个字符，而另外一个通配符_则表示匹配单个字符。

例题 4.5.2：表名为 student，姓名字段为 name，以 name 字段为关键词进行倒序输出的 sql 语句为？

【答案】Select * from student order by name desc；

【来源公司】国泰新点（2017）

【考点】SQL 语句

【解题思路】本题考查关键字 order by 以及 desc。另外，升序为 asc。

例题 4.5.3：下列哪一项不是 SQL 约束类型的一种？

A. Primary key

B. Foreign key

C. Alternate key

D.Unique

【答案】C

【来源公司】趋势科技（2018）

【考点】SQL 约束

【解题思路】

SQL 包括 6 种约束，分别是 NOT NULL 非空约束，UNIQUE 唯一性约束，primary key 主键约束，foreign key 外键约束，check 检查约束，default 默认约束。

1. PRIMARY KEY 约束：

主键必须包含唯一的值。主键列不能包含 NULL 值。每个表都应该有一个主键，并且每个表只能有一个主键。

2. UNIQUE 约束：

UNIQUE 和 PRIMARY KEY 约束均为列或列集合提供了唯一性的保证。PRIMARY KEY 约束拥有自动定义的 UNIQUE 约束。请注意，每个表可以有多个 UNIQUE 约束，但是每个表只能有一个 PRIMARY KEY 约束。

3. FOREIGN KEY 外键约束：

一个表中的 FOREIGN KEY 指向另一个表中的 PRIMARY KEY。请看下面两个表：

PERSON 表

ID_P	Name	City
1	Adams	London
2	Bush	NewYork
3	Carter	BeiJing

ORDERS 表

ID_O	OrderNo	ID_P
1	23123	3
2	12324	2
3	34545	2

请注意，"Orders" 中的 "Id_P" 列指向 "Persons" 表中的 "Id_P" 列。

"Persons" 表中的 "Id_P" 列是 "Persons" 表中的 PRIMARY KEY。

"Orders" 表中的 "Id_P" 列是 "Orders" 表中的 FOREIGN KEY。

FOREIGN KEY 约束用于预防破坏表之间连接的动作,也能防止非法数据插入外键列，因为它必须是它指向的那个表中的值之一。

4. check 检查约束：

CHECK 约束用于限制列中的值的范围。如果对单个列定义 CHECK 约束，那么该列只允许特定的值。如下所示：

```
CREATE TABLE Persons
(
    P_Id int NOT NULL CHECK (P_Id>0),
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
)
```

表 Persons 中，P_Id 项被限定为必须为整数，否则无法插入。

5. default 默认约束：

DEFAULT 约束用于向列中插入默认值。如果没有规定其他的值，那么会将默认值添加到所有的新记录。如下所示：

```
CREATE TABLE Persons
(
    P_Id int NOT NULL,
    LastName varchar(255) NOT NULL,
```

```
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255) DEFAULT 'Sandnes'
)
```

表 Persons 中，City 的默认值被设置为 Sandnes。

6. NOT NULL 约束：强制列不接受 NULL 值。NOT NULL 约束强制字段始终包含值。这意味着，如果不向字段添加值，就无法插入新记录或者更新记录。下面的 SQL 强制 "P_Id" 列和"LastName"列不接受 NULL 值：

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

4.6 Linux 系统操作基础

例题 4.6.1 对所有用户的环境变量设置，应放在哪个文件下面（）

A. /etc/bashrc

B. /etc/profile

C. ~/.bash_profile

D. ~/.bashrc

【来源公司】CVTE（2017）

【考点】配置文件

【参考答案】AB

【解题思路】AB 均为全局的配置文件，而 CD 选项为家目录下的文件，不同用户登录时所对相应的家目录都不同，因此想对所有人生效，必须修改/etc/bashrc 或者/etc/profile。

例题 4.6.2 下列关于如何查看当前 linux 系统的状态（如 CPU 使用，内存使用、负载情况等）正确的是（）

A、可以使用 top 命令分析 CPU 使用，内存使用，负载等情况。

B、其他选项的描述都不正确

C、可以使用 free 查看内存整体的使用情况

D、可以使用 cat/proc/meminfo 查看内存更详细的情况。

【参考答案】ACD

【来源公司】CVTE（2017）

【考点】Linux 命令

【解题思路】

A 选项中，top 命令：是 Linux 下常用的性能分析工具，能够实时显示系统中各个进程的资源占用情况；

C 选项中，free 命令：用来查看内存使用情况的主要命令；

D 选项中，cat/proc/meminfo 命令：得出更详细的内存占用信息；

例题 4.6.3 关于 GDB 的使用，下面说法错误的是（）

A、可以用 watch 为表达式（变量）expr 设置一个观察点，当表达式值有变化时，马上停住程序。

B、使用 step 调试可以进入被调用函数体内，可简写为 s。

C、如果需要在源程序第 10 行设置断点，则可以使用 b10 进行设置。

D、如果想 GDB 调试 C/C++程序，在编译时，我们必须要把调试信息加到可执行文件中，使用 gcc -c 可以做到这一点。

【参考答案】D

【来源公司】CVTE（2018）

【考点】GDB 调试

【解题思路】

加入调试信息应该为 gcc –g，-c 选项代表只编译不链接。

例题 4.6.4 系统中有用户 user1 和 user2，同属于 user 组，在 user1 用户目录下有一文件 file1，他拥有 644 的权限，如果 user2 用户想修改 user1 用户目录下的 file1 文件，应拥有什么权限（）

A、746

B、744

C、646

D、664

【答案】D

【来源公司】CVTE（2018）

【考点】linux 文件系统

【解题思路】

已知二者属于同一组，文件 file1 属于 user1，因此 user2 应该在同组中对文件 file1 拥有写权限，写权限的值为 2，因此在 644 的同组权限基础上加 2 即可。

例题 4.6.5 在 shell 编程中，关于$2 描述正确的是（）

A、表示对 shell 程序的第 2 个位置参数$2 进行赋值

B、表示 shell 程序的第 2 个位置参数

C、表示 shell 程序携带了 2 个位置参数

D、可用$10 引用第十个位置参数

【参考答案】BD

【来源公司】CVTE（2017）

【考点】shell 编程

【解题思路】

和 C 语言的 main 函数类似，在运行一个 bash 脚本时 我们也可以给脚本程序传递参数，如有脚本程序 test.sh：

```
1  echo "shell 脚本本身的名字: $0"
2  echo "传给 shell 的第一个参数: $1"
3  echo "传给 shell 的第二个参数: $2"
4  echo "传给 shell 的第二个参数: $3"
5  echo "传给 shell 的第二个参数: $4"
6  echo "传给 shell 的第二个参数: $5"
```

```
7   echo "传给 shell 的第二个参数: $6"
8   echo "传给 shell 的第二个参数: $7"
9   echo "传给 shell 的第二个参数: $8"
10  echo "传给 shell 的第二个参数: $9"
11  echo "传给 shell 的第二个参数: $10"
```

在终端中运行该脚本：

```
1  bash test.sh 1 2 3 4 5 6 7 8 9 10
```

程序输出为：

```
1   shell 脚本本身的名字: test.sh
2   传给 shell 的第一个参数: 1
3   传给 shell 的第二个参数: 2
4   传给 shell 的第二个参数: 3
5   传给 shell 的第二个参数: 4
6   传给 shell 的第二个参数: 5
7   传给 shell 的第二个参数: 6
8   传给 shell 的第二个参数: 7
9   传给 shell 的第二个参数: 8
10  传给 shell 的第二个参数: 9
11  传给 shell 的第二个参数: 10
```

例题 4.6.6 在使用 mkdir 命令创建新的目录时，在其父目录不存在时先创建父目录的选项是（）

A. –m

B. –d

C. -f

D. -p

【参考答案】D

【考点】shell 编程

【解题思路】

解析：此题考的是 Linux 命令 mkdir 的命令参数。

A 选项为设定目录权限；

D 选项为所要创建的目录的父目录不存在时，先创建父目录；

B,C 两个选项均不是 mkdir 的命令参数。