

一、 单项选择题

1. 若有以下说明语句：

```
struct date
{ int year;
  int month;
  int day;
}brithday;
```

则下面的叙述不正确的是__C__.

- A) struct 是声明结构体类型时用的关键字
- B) struct date 是用户定义的结构体类型名
- C) brithday 是用户定义的结构体类型名
- D) year, day 都是结构体成员名

2. 以下对结构变量 stu1 中成员 age 的非法引用是 B

```
struct student
{ int age;
  int num;
}stu1,*p;
p=&stu1;
```

- A) stu1.age
- B) student.age
- C) p->age
- D) (*p).age

3. 设有如下定义：

```
struck sk
{   int a;
    float b;
}data;
int *p;
```

若要使 P 指向 data 中的 a 域，正确的赋值语句是 C

- A) p=&a;
- B) p=data.a;
- C) p=&data.a;
- D) *p=data.a;

4. 设有以下说明语句：

```
typedef struct stu
{ int a;
  float b;
} stutype;
```

则下面叙述中错误的是 (D)。

- A、 struct 是结构类型的关键字
- B、 struct stu 是用户定义的结构类型
- C、 a 和 b 都是结构成员名
- D、 stutype 是用户定义的结构体变量名

5. 语句 `int *p;`说明了 C。

- A) p 是指向一维数组的指针
- B) p 是指向函数的指针, 该函数返回一 `int` 型数据
- C) p 是指向 `int` 型数据的指针 // 指针的定义教材 P223
- D) p 是函数名, 该函数返回一指向 `int` 型数据的指针

6. 下列不正确的定义是 (A)。

- A. `int *p=&i, i;`
- B. `int *p, i;`
- C. `int i, *p=&i;`
- D. `int i, *p;`

选项 A 先定义一个整型指针变量 p, 然后将变量 i 的地址赋给 p。然而此时还未定义变量 i 因此编译器无法获得变量 i 的地址。(A 与 C 对比, 选项 C 先定义变量 i, 则在内存中为 i 分配空间, 因此 i 在内存空间的地址就可以确定了; 然后再定义 p, 此时可以为 p 赋 i 的地址, C 正确)

7. 若有说明: `int n=2, *p=&n, *q=p;` 则以下非法的赋值语句是: (D)

- A) `p=q`
- B) `*p=*q`
- C) `n=*q`
- D) `p=n`

p, q 同为整型指针变量, 二者里面仅能存放整型变量的地址。

选项 A, q 中为地址, 因此可将此地址赋给 p

选项 B, *p 表示 p 所指向对象 n 的内容, 即一个整数; *q 表示 q 所指向对象的内容, 由于在定义 q 时为其初始化, 将 p 中 n 的地址给 q, 因此 p 中存放 n 的地址, *q 表示 q 所指向对象 n 的内容. 因此 `*p=*q` 相当于 `n=n;`

选项 C, `n=*q` 等价于 `n=n;`

选项 D, p 中只能存放地址, 不能将 n 中的整数值赋给 p

8. 有语句: `int a[10];` 则 B 是对指针变量 p 的正确定义和初始化。

- A) `int p=*a;`
- B) `int *p=a;`
- C) `int p=&a;`
- D) `int *p=&a;`

选项 A, a 是数组名, 不是指针变量名, 因此不可用 *标注数组名 a

选项 C, a 是数组名, 数组名就是地址, 无需再用地址符号。而且在定义指针变量 p 时, 应在变量名前加 *, 标明 p 是指针变量

选项 D, a 是数组名, 数组名就是地址, 无需再用地址符号。

9. 若有说明语句 “`int a[5], *p=a;`”, 则对数组元素的正确引用是 (C)。

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

首先定义一个整型数组 a, a 的长度为 5, 然后定义一个指针变量 p, 并同时 对 p 进行初始化, 将数组 a 的地址赋给 p。因此此时 p 中存放的数组 a 的首地址, 即数组中第一个元素 `a[0]` 的地址。

对于数组元素下标的引用(详见 p144), 一般形式 数组名[下标] 其中下标 为逻辑地址下标, 从 0 开始计数, 方括号中的下标可以是变量, 可以是表达式, 但结果一定要是整数。

选项 A, p 中存放的是地址, 不是整数, 不能做数组元素的下标

选项 B, a 是数组名, 数组名就是地址, 不是整数, 不能做数组元素的下标

选项 C, (重点!!! 详见 p231~234) $p+2$ 表示指向同一数组中的下两个元素的地址, 当前 p 指向 $a[0]$, 则 $p+2$ 表示 $a[2]$ 的地址, 因此 $*(p+2)$ 表示 $a[2]$ 的内容

10. 有如下程序

```
int a[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, *P=a;
```

则数值为 9 的表达式是 B

- A) $*P+9$ B) $*(P+8)$ C) $*P+=9$ D) $P+8$

(重点!!! 详见 p231~234)

首先定义一个整型数组 a , a 的长度为 5, 然后定义一个指针变量 P , 并同时
对 P 进行初始化, 将数组 a 的地址赋给 P 。因此此时 P 中存放的数组 a 的首地址,
即数组中第一个元素 $a[0]$ 的地址。

数组中 9 对应的是 $a[8]$, 选项 B, $P+8$ 表示数组中后 8 个元素的地址, 即 $a[8]$
的地址。 $*(P+8)$ 则表示该地址内所存放的内容, 即 $a[8]$ 的值。

选项 A, $*P$ 表示 P 所指向对象的内容, 此时 P 指向 $a[0]$, $*P$ 即 $a[0]$ 的值 1。
 $*P+9=1+9=10$

选项 C, $*P$ 表示 P 所指向对象的内容, 此时 P 指向 $a[0]$, $*P$ 即 $a[0]$ 的值。
因此 $*P+=9$ 即 $*P = *P+9$, 等价于 $a[0]=a[0]+9$ 。

选项 D, $P+8$ 表示数组中后 8 个元素的地址, 即 $a[8]$ 的地址, 而非 $a[8]$ 中的
值。

11. 在 C 语言中, 以 D 作为字符串结束标志

- A) $'\backslash n'$ B) $' '$ C) $' 0'$ D) $'\backslash 0'$

12. 下列数据中属于“字符串常量”的是 (A)。

- A. “a” B. {ABC} C. 'abc\0' D. 'a'

若干个字符构成字符串

在 C 语言中, 用单引号标识字符; 用双引号标识字符串

选项 B, C, 分别用 {} 和 ' ' 标识字符串

选项 D, 标识字符。

13. 已知 $\text{char } x[] = \text{"hello"}, y[] = \{'h', 'e', 'a', 'b', 'e'\};$, 则关于两个数组长
度的正确描述是 B。

- A) 相同 B) x 大于 y C) x 小于 y D) 以上答案都不对

C 语言中, 字符串后面需要一个结束标志位 $'\backslash 0'$, 通常系统会自动添加。

对一维数组初始化时可采用字符串的形式 (例如本题数组 x), 也可采用字符集
的形式 (例如本题数组 y)。在以字符串形式初始化时, 数组 x 不仅要存储字
符串中的字符, 还要存储字符串后的结束标志位, 因此数组 x 的长度为 6; 在以
字符集合形式初始化时, 数组 y , 仅存储集合中的元素, 因此数组 y 长度为 5

二、 读程序

基本输入输出及流程控制

1.

```
#include <stdio.h>
main()
{int a[2][3]={ {1, 2, 3}, {4, 5, 6} };
  int b[3][2], i, j;
  for(i=0;i<=1;i++)
    {for(j=0;j<=2;j++)
      b[j][i]=a[i][j];
    }
  for(i=0;i<=2;i++)
    {for(j=0;j<=1;j++)
      printf("%5d", b[i][j]);
    }
}
```

运行结果为:

1 4 2 5 3 6

2.

```
#include <stdio.h>
f(int b[],int n)
{int i,r;
  r=1;
  for (i=0;i<=n;i++)
    r=r*b[i];
  return (r);
}
main()
{int x, a[]={1, 2, 3, 4, 5, 6, 7, 8, 9};
  x=f(a, 3);
  printf( "%d\n" , x);
}
```

运行结果为:

24

3.

```

#include"stdio.h"
main()
{int j,k;
 static int x[4][4],y[4][4];
 for(j=0;j<4;j++)
  for(k=j;k<4;k++)
   x[j][k]=j+k;
 for(j=0;j<4;j++)
  for(k=j;k<4;k++)
   y[k][j]=x[j][k];

 for(j=0;j<4;j++)
  for(k=0;k<4;k++)
   printf("%d, ", y[j][k]);
}

```

运行结果为:

0, 0, 0, 0, 1, 2, 0, 0, 2, 3, 4, 0, 3, 4, 5, 6

函数

1.

```

#include <stdio.h>
int Sub(int a, int b)
{return (a- b);}
main()
{int x, y, result = 0;
 scanf("%d,%d", &x,&y );
 result = Sub(x,y ) ;
 printf("result = %d\n",result);
}

```

当从键盘输入:6, 3 运行结果为:

result =3

2.

```

#include <stdio.h>
int min( int x, int y )
{ int m;
 if ( x> y ) m = x;

```

```

else      m = y;
return(m);
}
main()    {
    int  a=3,b=5,abmin ;
    abmin = min(a,b);
    printf( "min  is  %d" ,  abmin);
}

```

运行结果为:

min is 5

3.

```

#include<stdio.h>
func(int x) {
    x=10;
    printf( "%d,  ",x);
}
main( )
{  int x=20;
    func(x);
    printf( "%d", x);
}

```

运行结果为:

10, 20

在 main 函数中调用函数 func，main 函数将 20 作为实参穿给 func，并转向开始执行 func.

main() 对应的内存

func () 对应的内存



func () 执行 x=10;; 其内存中 x 变为 10.



func() 执行 printf(“%d, ”, x); 即输出 func 函数对应内存中 x 的值，输出的是 10。至此，func 函数执行结束，返回 main 函数。

main 函数执行 printf(“%d”, x); 此时输出 main 函数对应内存中的 x，即 20

4.

```
#include <stdio.h>
int m=4;
int func(int x,int y)
{   int m=1;
    return(x*y-m);
}
main()
{int a=2,b=3;
    printf("%d\n",m);
    printf("%d\n",func(a,b)/m);
}
```

运行结果为：

4
1

整型变量 m 在函数外定义，因此 m 为全局变量，其作用于范围为其定义位置开始，一直到整个程序结束。因此 func 与 main 函数都可以访问 m

程序首先执行 main 函数

执行 printf(“%d\n”, m); 即输出 m 中的值 4，并换行。

执行 printf(“%d\n”, func(a, b)/m); 即输出表达式 func(a, b)/m 的值，为了计算该表达式，

需要调用函数 func。此时 main 将 a, b 中的 2 和 3 值作为实参传递给 func 的 x 和 y

程序开始转向执行 func 函数，此时 func 中的 x 为 2，y 为 3

执行 int m=1; 此句定义了一个局部变量 m 并赋值为 1。m 的作用域为其所在的复合

语句，即 func 的函数体，因此在 func 的函数体重，有限访问局部变量 m。

执行 `return(x*y-m);` 即 `return (2*3-1)` ;返回的是整数 5.

func 函数返回至 main 函数中的被调用处

main 函数中 `func(a,b)` 的值为 5, `func(a,b)/m=5/4=1`, 注意, 在 main 函数中访问的 `m` 为全局变量 `m`, 此时 main 函数无法访问 func 中的 `m`, 因为不在 func 中 `m` 的作用域。

5.

```
#include <stdio.h>
int fun(int a, int b)
{ if(a>b)    return(a);
  else      return(b);
}
main()
{ int x=15, y=8, r;
  r= fun(x,y);
  printf("r=%d\n", r);
}
```

运行结果为: **r=15**

程序首先执行 main 函数

执行 `r= fun(x,y);` 即将 `func(x,y)` 的值赋给 `r`, 为了计算该表达式, 需要调用函数 `func`。此时 main 将 `x,y` 中的 15 和 8 值作为实参传递给 `func` 的 `a` 和 `b`

程序开始转向执行 `func` 函数, 此时 `func` 中的 `a` 为 15, `b` 为 8

执行 `if` 语句;判断 `if` 后面的表达式, `a>b` 成立, 因此执行相应的操作 `return(a);` 即返回 `a` 的值。

func 函数返回至 main 函数中的被调用处

main 函数中 `func(x,y)` 的值为 15, 即将 15 赋给 `r`。

执行 `printf("r=%d\n", r);` 即输出 `r=15`

6.

```
#include <stdio.h>
int fac(int n)
{ int f=1,i;
  for(i=1;i<=n;i++)
    f=f * i;
```

```

    return(f);
}
main()
{ int j,s;
  scanf( "%d" ,&j);
  s=fac(j);
  printf("%d!=%d\n", j, s);
}

```

如果从键盘输入 3， 运行结果为： **3!=6**

程序首先执行 main 函数

执行 `r= fun(x,y);` 即将 `func(x,y)` 的值赋给 `r`，为了计算该表达式，需要调用函数 `func`。此时 `main` 将 `x,y` 中的 15 和 8 值作为实参传递给 `func` 的 `a` 和 `b`

程序开始转向执行 `func` 函数，此时 `func` 中的 `a` 为 15，`b` 为 8

执行 `if` 语句;判断 `if` 后面的表达式，`a>b` 成立，因此执行相应的操作 `return(a);` 即返回 `a` 的值。

`func` 函数返回至 `main` 函数中的被调用处

`main` 函数中 `func(x,y)` 的值为 15，即将 15 赋给 `r`。

执行 `printf("r=%d\n", r);` 即输出 `r=15`

7.

```

#include <stdio.h>
unsigned fun6(unsigned num)
{ unsigned k=1;
  do
  { k*=num%10;
    num/=10;
  }while(num);
  return k;
}
main()
{ unsigned n=26;
  printf( "%d\n" ,fun6(n));
}

```

运行结果为： **12**

程序首先执行 main 函数

执行 `printf(“%d\n”, fun6(n));` 即输出表达式 `func(6)` 的值, 为了计算该表达式, 需要调用

函数 `func`。此时 `main` 将 `n` 中的 26 作为实参传递给 `func` 的 `num`
程序开始转向执行 `func` 函数, 此时 `func` 中的 `num` 为 26

执行 `do-while` 语句

第 1 次循环

执行 `k*=num%10`, 即 `k=k*(num%10)=1*(26%10)=6`

执行 `num/=10`; 即 `num=num/10=26/10=2`

`while` 后面循环条件为 `num`, 此时 `num` 为 2, 是非 0 值, 即表示循环条件成立,

继续执行循环体。此时 `k` 为 6

第 2 次循环

执行 `k*=num%10`, 即 `k=k*(num%10)=6*(2%10)=12`

执行 `num/=10`; 即 `num=num/10=2/10=0`

`while` 后面循环条件为 `num`, 此时 `num` 为 0, 表示循环条件不成立,

结束循环

执行 `return k`; 即返回至 `main` 函数中的被调用处

执行 `main` 函数

继续执行 `printf(“%d\n”, fun6(n));` 即输出 12

8.

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

```
int max(int x, int y);
```

```
main()
```

```
{ int a,b,c;
```

```
  a=7;b=8;
```

```
  c=max(a,b);
```

```
  printf("Max is %d",c);
```

```
}
```

```
max(int x, int y)
```

```
{ int z;
```

```
z=x>y? x : y;
```

```
return(z) ;
```

```
}
```

运行结果为:

Max is 8

指针

1.

```
#include <stdio.h>
main ( )
{ int x[ ] = {10, 20, 30, 40, 50 };
  int *p ;
  p=x;
  printf ( "%d" , *(p+2) );
}
```

运行结果为:

30

首先定义一个整型数组 x，x 的长度为 5;然后定义一个指针变量 p;对 p 进行初始化，将数组 x 的地址赋给 p。因此此时 p 中存放的数组 x 的首地址，即数组中第一个元素 x[0]的地址。

然后执行 printf 语句，输出表达式*(p+2) 的值。p+2 表示以 p 当前指向的位置起始，之后第 2 个元素的地址，即 a[2]的地址。*(p+2) 则表示该地址内所存放的内容，即 a[2]的值 30, 因此输出 30

2.

```
#include <stdio.h>
main( )
{ char s[]=" abcdefg" ;
  char *p;
  p=s;
  printf( "ch=%c\n" , *(p+5));
}
```

运行结果为:

ch=f

首先定义一个字符型数组 s，并用字符串 abcdefg 对 s 进行初始化；然后定义一个字符型指针变量 p；对 p 进行初始化，将数组 s 的地址赋给 p。因此此时 p 中存放的数组 s 的首地址，即数组中第一个元素 s[0]的地址。

然后执行 printf 语句，输出表达式*(p+5) 的值。p+5 表示以 p 当前指向的位置起始，之后第 5 个元素的地址，即 a[5]的地址。*(p+5) 则表示该地址内所存放的内容，即 a[5]的值 f，因此输出 ch=f

3.

```
#include<stdio.h>
main ( )
{ int a[]={1, 2, 3, 4, 5} ;
  int x, y, *p;
  p=a;
  x=*(p+2);
  printf("%d: %d \n", *p, x);
}
```

运行结果为:

1:3

首先定义一个整型数组 a，并对 a 进行初始化；然后定义整型变量 x, y，整型指针变量 p；再将数组 a 的地址赋给 p。因此此时 p 中存放的数组 a 的首地址，即数组中第一个元素 a[0]的地址。执行 x=*(p+2)；p+2 表示以 p 当前所指向的位置起始，之后第 2 个元素的地址，即 a[2]的地址。*(p+2)则表示该地址内所存放的内容，即 a[2]的值 3，然后再把 3 赋给 x

然后执行 printf 语句，先输出表达式*p 的值。此时*p 表示的是 p 所指向变量的内容，即 a[0]的值 1。再输出一个冒号。然后再输出 x 中的值 3。

4.

```
#include<stdio.h>
main()
{ int arr[ ]={30, 25, 20, 15, 10, 5}, *p=arr;
  p++;
  printf( "%d\n" ,*(p+3));
}
```

运行结果为: 10

首先定义一个整型数组 arr，并对 arr 进行初始化；然后定义整型指针变量 p；再将数组 arr 的地址赋给 p。因此此时 p 中存放的数组 arr 的首地址，即数组中第一个元素 a[0]的地址。

执行 p++，即 p=p+1。p+1 表示以 p 当前所指向的位置起始，之后第 1 个元素的地址，即 arr[1]的地址，然后再将 arr[1]的地址赋给 p，执行完此语句后，p 不再指向 arr[0]，而是指向 arr[1]。

然后执行 printf 语句，输出表达式*(p+3) 的值。p+3 表示以 p 当前指向的位

置起始(此时 p 指向 arr[1]), 之后第 3 个元素的地址, 即 arr[4] 的地址。*(p+3) 则表示该地址内所存放的内容, 即 arr[4] 的值 10, 因此输出 10

5.

```
#include <stdio.h>
main( )
{
    int  a[ ]={1, 2, 3, 4, 5, 6};
    int  x, y, *p;
    p = &a[0];
    x = *(p+2);
    y = *(p+4);
    printf( “*p=%d, x=%d, y=%d\n” , *p, x, y);
}
```

运行结果为:

***p=1, x=3, y=5**

首先定义一个整型数组 a, 并对 a 进行初始化; 然后定义整型变量 x, y, 整型指针变量 p; 再将数组元素 a[0] 的地址赋给 p。

执行 x=*(p+2); p+2 表示以 p 当前所指向的位置起始, 之后第 2 个元素的地址, 即 a[2] 的地址。*(p+2) 则表示该地址内所存放的内容, 即 a[2] 的值 3, 然后再把 3 赋给 x

执行 y = *(p+4); p+4 表示以 p 当前所指向的位置起始, 之后第 4 个元素的地址, 即 a[4] 的地址。*(p+4) 则表示该地址内所存放的内容, 即 a[4] 的值 5, 然后再把 5 赋给 y

执行 printf 语句, 先输出表达式 *p 的值。此时 *p 表示的是 p 所指向变量的内容, 即 a[0] 的值 1。再输 x 的值 3。再输出 y 的值 5。

6.

```
#include<stdio.h>
main( )
{
    static char a[ ]=" Program" , *ptr;
    for(ptr=a, ptr<a+7; ptr+=2)
        putchar(*ptr);
}
```

运行结果为:

Porm

首先定义一个字符型数组 a, 并对 a 进行初始化; 然后定义字符型指针变量 p;

执行 for 语句 `ptr=a` 为表达式 1, 将数字 a 的地址赋给 ptr; 表达式 2 (循环条件) `ptr<a+7`; 表达式 3 为 `ptr+=2`, 即 `ptr= ptr+2`;

第 1 次执行循环体

执行 `putchar(*ptr)`; 即输出*ptr 所对应的字符。此时 ptr 指向数组中的第 1 个元素, 即

`a[0]`, 因此*ptr 表示 `a[0]` 中的值, 即 'P' .

执行完循环体, 转向执行表达式 3, 即 `ptr= ptr+2`。ptr+2 表示以 ptr 当前所指向的位

置起始, 之后第 2 个元素的地址, 即 `a[2]` 的地址, 然后将 `a[2]` 的地址赋给 ptr。a[2] 的地

址等价于 `a+2`, 因此循环条件 `ptr<a+7` 成立, 继续执行循环体

第 2 次执行循环体

执行 `putchar(*ptr)`; 即输出*ptr 所对应的字符。此时 ptr 指向数组中的第 3 个元素, 即

`a[2]`, 因此*ptr 表示 `a[2]` 中的值, 即 'o' .

执行完循环体, 转向执行表达式 3, 即 `ptr= ptr+2`。ptr+2 表示以 ptr 当前所指向的位

置起始, 之后第 2 个元素的地址, 即 `a[4]` 的地址, 然后将 `a[4]` 的地址赋给 ptr。a[4] 的地

址等价于 `a+4`, 因此循环条件 `ptr<a+7` 即 `a+4<a+7` 成立, 继续执行循环体

第 3 次执行循环体

执行 `putchar(*ptr)`; 即输出*ptr 所对应的字符。此时 ptr 指向数组中的第 5 个元素, 即

`a[4]`, 因此*ptr 表示 `a[4]` 中的值, 即 'r' .

执行完循环体, 转向执行表达式 3, 即 `ptr= ptr+2`。ptr+2 表示以 ptr 当前所指向的位

置起始, 之后第 2 个元素的地址, 即 `a[6]` 的地址, 然后将 `a[6]` 的地址赋给 ptr。

a[6] 的地址等价于 `a+6`, 因此循环条件 `ptr<a+7` 即 `a+6<a+7` 成立, 继续执行循环体

第 4 次执行循环体

执行 `putchar(*ptr)`; 即输出*ptr 所对应的字符。此时 ptr 指向数组中的第 7 个元素, 即

`a[6]`, 因此*ptr 表示 `a[6]` 中的值, 即 'm' .

执行完循环体, 转向执行表达式 3, 即 `ptr= ptr+2`。ptr+2 表示以 ptr 当前所指向的位

置起始, 之后第 2 个元素的地址, 即 `a[8]` 的地址, 然后将 `a[8]` 的地址赋给 ptr。

a[6] 的地址等价于 `a+8`, 因此循环条件 `ptr<a+7` 即 `a+8<a+7` 不成立, 结束循

环。

7.

```
#include <stdio.h>
char s[]=" ABCD" ;
main()
{ char *p;
  for(p=s;p<s+4;p++)
    printf( "%c %s\n" , *p, p);
}
```

运行结果为：

A ABCD

B BCD

C CD

D D

首先定义一个字符型数组 s，并对 s 进行初始化；数组 s 是全局变量，其有效范围从其定义开始至整个程序结束。

执行 main 函数

定义一个字符型指针 p。

执行 for 语句 p=s 为表达式 1，将数组 s 的首地址赋给 p；表达式 2（循环条件）p<s+4；表达式 3 为 p++，即 p= p+1；

第 1 次执行循环体

执行 printf("%c %s\n" , *p, p);即以字符 %c 形式输出*p 所对应的字符。此时 p 指

向数组中的第 1 个元素，即 s[0]，因此*p 表示 a[0]中的值，即' A' .

然后再以字

符串%s 的形式输出以 p 中地址为首地址的整个字符串，即输出 ABCD

执行完循环体，转向执行表达式 3，即 p= p+1。p+1 表示以 p 当前所指向的

位置起始，之后 1 个元素的地址，即 s[1]的地址，然后将 a[1]的地址赋给 p。

s[1]的地址等价于 s+1，因此循环条件 p<s+4 成立，继续执行循环体

第 2 次执行循环体

执行 printf("%c %s\n" , *p, p);即以字符%c 形式输出*p 所对应的

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/488044027062006050>