1

1. **使用函数可以增加程序的可读性。**
   以一个具有说明性的函数调用来代替一组语句，不仅可以使程序显得更加简洁，而且可以使程序执行的脉络显得更加清晰。
2. **使用函数便于对计算过程自顶向下的层次化描述。**
   在程序中以函数名代表计算过程，就可以通过对函数的定义实现对计算过程的逐步分解和细化。这样一种自顶向下逐步分解和细化的对计算过程的描述，符合我们正常的思维过程，有利用我们从宏观到细节，完整准确地把握和描述程序。
3. **使用函数还可以增加代码的可重用性。**
   在函数定义完成之后，在程序中需要执行相同或类似计算功能的地方只需要对该函数进行调用即可，而不必再一一重复相应的计算语句。这样就可以大大减少程序设计中的编码工作量。
4. **使用函数可以显著增加代码的可维护性。**
   通过函数定义，可以把与指定功能相关的语句集中在一个明确的范围内，这样就大大方便了在程序调试时的故障定位。当某一功能的实现细节，如数据格式、计算方法等发生变化时，只会影响其所涉及的相关函数而不会影响程序的顶层结构和其他函数。同时，由于使用了函数，在程序中同一种功能的多处使用都可以通过对一个函数的调用来完成。当程序需要进行调试或修改时，因为一种功能只定义了一次，所以修改只在一处进行即可。这不仅减少了程序调试和修改时的工作量，更可以避免对多处代码进行修改时由于疏忽而引起的程序行为的不一致。

2

1. 函数名、参数的个数及类型、函数返回值的类型，总称为函数原型
2. 知道了一个函数的原型，就知道了一个函数需要通过什么样的方式被调用，以及如何使用函数的返回值。通过函数原型，我们可以知道如何正确地使用相应的函数，而不必了解函数具体定义的代码。

3

通过参数接收函数外部的数据，通过返回值向函数外部传递计算结果。

4

1. 可以被赋给具有相同类型的变量，也可以直接用在其他表达式中。
2. 不一定。

5

- ***形式参数***：当定义一个函数时，函数的参数只是一个标识符，从形式上说明函数需要某种类型的参数。
- ***实际参数***：只有当函数被调用时，参数才被具体化为实际的数据。

6

1. ***说明序列***：由一个或多个说明语句组成，说明在函数中使用但在函数外部定义的实体，如变量和函数等。
2. ***变量定义序列***：定义函数内部用于保存中间结果的各个变量。
3. ***执行语句序列***：描述函数所要执行的各个操作。

7

可以；无影响。

8

可以；当在函数中访问一个变量且该函数中有同名的局部变量时，程序选择该局部变量。

9

被调用函数的原型在对该函数调用的语句之前声明过，并且在编译时有定义。

10

所谓值传递，就是在进行函数调用时，程序首先对各个参数表达式进行求值操作，并将求值的结果作为实际参数传递给函数。因此函数的参数可以是任何形式的表达式，只要其类型与形式参数的类型一致即可。

11

可以；无影响，除非是全局变量或者是使用指针类型的参数。

12

否。

13

- <stdio.h>数据输入、输出
- <math.h>数值计算
- <ctye.h>字符类型判断
- <string.h>字符串处理
- <stdlib.h>通用数据处理

14

可以；把头文件中函数定义的代码复制到源程序中。

15

C

16

B

17

```
#include <stdio.h>
void func(int s, int st, int j);
```

```c
void func(int s, int st, int j)
{
    int k;

    printf("%d = %d", s, st++);
    for(k = st; k <= j; k++)
        printf(" + %d", k);
        putchar('\n');
}
int main()
{
    int s, i, j, k, st, sum, got = 0;

    scanf("%d", &s);
    for(i = 1; i <= s / 2; i++){
        sum = i;
        for(j = i + 1; sum < s; j++){
            sum += j;
            if(sum == s){
                st = i;
                func(s, st, j);
                got++;
            }
        }
    }
    if(got == 0)
        printf("NONE\n");

    return 0;
}
```

18

```c
#include <stdio.h>
#include <math.h>
#define PI 3.14159265
double area(double a, double b, double ang_c);
double area(double a, double b, double ang_c)
{
    return a * b * sin(ang_c * PI / 180.0) / 2.0;
}
int main()
{
    double a, b, ang_c, s;

    scanf("%lf%lf%lf", &a, &b, &ang_c);
    s = area(a, b, ang_c);
    printf("The area is %f\n", s);

    return 0;
}
```

19

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

```
double f_to_c(int f);
double c_to_f(int c);
double f_to_c(int f)
{
    return (f - 32.0) * 5 / 9;
}
double c_to_f(int c)
{
    return 9.0 * c / 5 + 32;
}
int main()
{
    int i, c, f;

    printf("C    F           F     C\n");
    for(i = 1; i <= 10; i++){
        c = 30 + i;
        f = 130 - i * 10;
        printf("%d   %-5.1f      %-3d   %-4.1f\n", c, c_to_f(c), f, f_to_c(f));
    }

    return 0;
}
```

20

```
#include <stdio.h>
bool istu(int n1, int n2, int n3);
bool istu(int n1, int n2, int n3)
{
    return n1 < n2 && n2 > n3;
}
int main()
{
    int n1, n2, n3, temp;

    scanf("%d%d", &n1, &n2);
    while(scanf("%d", &n3) != EOF){
        if(istu(n1, n2, n3))
            printf("%d %d %d\n", n1, n2, n3);
        n1 = n2;
        n2 = n3;
    }

    return 0;
}
```

21

```
#include <stdio.h>
#include <math.h>
double area_tri(double x1, double y1, double x2, double y2, double x3, double
y3);
double area_tri(double x1, double y1, double x2, double y2, double x3, double
y3)
```

```
{
    return fabs((x1 - x3) * (y2 - y3) - (x2 - x3) * (y1 - y3)) / 2;
}
int main()
{
    double x1, y1, x2, y2, x3, y3, area = 0;

    scanf("%lf%lf", &x1, &y1);
    scanf("%lf%lf", &x2, &y2);
    while(scanf("%lf%lf", &x3, &y3) != EOF){
        area += area_tri(x1, y1, x2, y2, x3, y3);
        x2 = x3;
        y2 = y3;
    }
    printf("%.3f", area);
}
```

22

第1种

```
#include <stdio.h>
int reversal(int n);
int reversal(int n)
{
    int temp;

    if(n == 0)
        return 0;
    printf("%d", n % 10);
    reversal(n /= 10);
}
int main()
{
    int n;

    scanf("%d", &n);
    while(n % 10 == 0)
        n /= 10;
    reversal(n);

    return 0;
}
```

第2种

```
#include <stdio.h>
int reversal(int n);
int reversal(int n)
{
    int temp, i = 1;

    if(n / 10 == 0)
        return n;
    temp = n;
```

```
    while(temp / 10 != 0){
        temp /= 10;
        i *= 10;
    }
    return temp + 10 * reversal(n % i);
}
int main()
{
    int n;

    scanf("%d", &n);
    printf("%d", reversal(n));

    return 0;
}
```

23

```
#include <stdio.h>
int reversal(int n);
int reversal(int n)
{
    int temp = 0;

    while(n / 10 != 0){
        temp += n % 10;
        temp *= 10;
        n /= 10;
    }
    temp += n;
    return temp;
}
int main()
{
    int m, n;

    scanf("%d%d", &m, &n);
    printf("%d", m * reversal(n));

    return 0;
}
```

24

```
#include <stdio.h>
int stair(int n);
int stair(int n)
{
    switch(n){
        case 1:
            return 1;
        case 2:
            return 2;
        case 3:
            return 4;
```

```
        default:
            return stair(n - 1) + stair(n - 2) + stair(n - 3);
    }
}
int main()
{
    int n;

    scanf("%d", &n);
    printf("%d", stair(n));

    return 0;
}
```