1.无参函数之北京欢迎您

> 编写形如void fun()的函数，该函数在屏幕上输出一个短句“Welcome to Beijing!”（不输出引号）。
> 然后在主函数main()中调用该函数。
>
> ---
>
> 输入
>
> > 无输入。
>
> ---
>
> 输出
>
> > Welcome to Beijing!

```c
#include <stdio.h>
void fun(){
    printf("Welcome to Beijing!");
}
int main(){
    fun();
}
```

2.无参函数之十以内的自然数

> 编写形如void fun()的函数，该函数在屏幕上从小到大输出10以内的自然数（含0和10），这些数之间用空格隔开。
> 然后在主函数main()中调用该函数。
>
> ---
>
> 输入
>
> > 无输入
>
> ---
>
> 输出

> > 0 1 2 3 4 5 6 7 8 9 10

```
#include <stdio.h>
void fun(){
    int i;
    for(i=0;i<=10;i++) printf("%d ",i);
}
int main(){
    fun();
}
```

3.无返回值函数之两数的积

> 编写形如void fun(int a, int b)的函数，该函数输出形参a和b的积。
> 在主函数main()中输入两个整数m和n，然后调用函数fun(int, int)，并将m和n作为该函数的实参。
>
> ---
>
> 输入
>
> > 输入两个整数m和n，以空格分开。
>
> ---
>
> 输出
>
> > 输出m和n的积。

```
#include <stdio.h>
void fun(int a,int b){
    printf("%d",a*b);
}
int main(){
    int m,n;
    scanf("%d%d",&m,&n);
    fun(m,n);
}
```

4.函数之左邻

编写形如int fun(int a)的函数，该函数的返回值仍然是整数，是形参a的自然数左邻。
在主函数main()中输入一个正整数m；然后调用函数int fun(int)，并将m作为该函数的实参；最后输出该函数的返回值到屏幕。

输入

> 输入一个正整数m。

输出

> 输出m的左邻。

```
#include <stdio.h>
void fun(int a){
    printf("%d",--a);
}
int main(){
    int m;
    scanf("%d",&m);
    fun(m);
}
```

5.函数之点与点之间的距离

编写形如float fun(int a1, int b1, int a2, int b2)的函数，该函数的返回值是一个非负浮点数，是点(a1,b1)与点(a2,b2)之间的距离。
在主函数main()中输入四个整数，分别代表两个点的坐标。然后调用函数fun(int, int, int, int)，并将这四个整数作为该函数的实参。最后输出该函数的返回值到屏幕，小数点后必须保留2位有效数字（四舍五入），不足补零。

输入

> 输入四个整数m、n、p、q，以空格分开，分别代表两个点的坐标。

输出

> 输出点(m,n)到点(p,q)之间的距离，小数点后必须保留2位有效数字（四舍五入），不足补零。

```
#include <stdio.h>
#include <math.h>
void fun(int a1,int b1,int a2,int b2){
    double d;
    d = pow(pow(a1-a2,2)+pow(b1-b2,2),0.5); //当然，算方根也可以用squt();squt和pow返回的都是double型
    printf("%.2f",d);
}
int main() {
    int m, n, p, q;
    scanf("%d%d%d%d", &m, &n, &p, &q);
    fun(m, n, p, q);
}
```

6.地址传递之斗转星移

> 编写形如void fun(int *a1, int *b1, int *a2, int *b2)的函数，该函数帮助姑苏慕容将四个形参指向的整数互换（a1指向的整数与a2指向的整数互换，b1指向的整数与b2指向的整数互换）。
> 在主函数main()中输入四个整数；然后调用函数fun(&a1,&b1,&a2,&b2)，并将四个整数的地址作为该函数的实参传入；函数调用完毕后输出四个整数的值。
>
> ---
>
> 输入
>
> > 输入四个整数m、n、p、q，以空格分开，分别代表两个点的坐标。
>
> ---
>
> 输出
>
> > 输出函数fun(&a1,&b1,&a2,&b2)调用完毕后，m、n、p、q的值。

方法一：不用指针（虽然题目是要求用指针的，但是不用也能过）

```
#include <stdio.h>
#define SWAP(a,b) {int temp;temp = a;a=b;b=temp;}
void fun(int a1,int b1,int a2,int b2){
    SWAP(a1,a2);
    SWAP(b1,b2);
    printf("%d %d %d %d",a1,b1,a2,b2);
```

```
}
int main() {
    int m, n, p, q;
    scanf("%d%d%d%d", &m, &n, &p, &q);
    fun(m, n, p, q);
}
```

方法二：用指针

```
#include <stdio.h>
#define SWAP(a,b) {int temp;temp = a;a=b;b=temp;}
void fun(int *a1,int *b1,int *a2,int *b2){
    SWAP(*a1,*a2);
    SWAP(*b1,*b2);
    printf("%d %d %d %d",*a1,*b1,*a2,*b2);
}
int main() {
    int m, n, p, q;
    scanf("%d%d%d%d", &m, &n, &p, &q);
    fun(&m, &n, &p, &q);
}
```

7.地址传递之跳蚤跳跳跳

> 编写形如void fun(int *a, int *b, int t)的函数，该函数帮助电子跳蚤以y轴为对称轴从第一象限上的点跳到第二象限，再以x轴为对称轴跳到第三象限，再以y轴为对称轴跳到第四象限，再以x轴为对称轴跳到第一象限……如此跳来跳去，参数t为跳的次数（t > 0）；最终该函数将形参a、b指向的整数赋值为电子跳蚤最后的落点。
> 在主函数main()中输入三个正整数m、n、t，m、n分别代表第一象限的点坐标，t代表跳的次数；然后调用函数fun(&m, &n, t)，并将整数m、n的地址和整数t作为该函数的实参传入；函数调用完毕后输出整数m、n的值。
>
> ---
>
> 输入
>
> > 输入三个正整数m、n、t，以空格分开，m、n代表第一象限点的坐标，t代表跳的次数。
>
> ---
>
> 输出
>
> > 输出函数fun(&m, &n, t)调用完毕后，m、n的值。

方法一：不用指针（同7）

```
#include <stdio.h>
void fun(int m,int n,int t){
    int f;
    f = t%4;
    switch(f){
        case 0: break;
        case 1: m = -m;break;
        case 2: m = -m;n=-n;break;
        case 3: n = -n;break;
    }
    printf("%d %d",m,n);
}
int main() {
    int m, n, t;
    scanf("%d%d%d", &m, &n, &t);
    fun(m, n, t);
}
```

方法二：用指针

```
#include <stdio.h>
void fun(int *m,int *n,int t){
    int f;
    f = t%4;
    switch(f){
        case 0: break;
        case 1: *m = -*m;break;
        case 2: *m = -*m;*n=-*n;break;
        case 3: *n = -*n;break;
    }
    printf("%d %d",*m,*n);
}
int main() {
    int m, n, t;
    scanf("%d%d%d", &m, &n, &t);
    fun(&m, &n, t);
}
```

8.函数之数组求和问题

> 编写形如int fun(int arr[], int t)的函数，该函数计算并返回整型数组arr[]各个元素之和，形参t为数组元素个数。
> 在主函数main()中输入数组元素个数N和N个整数，并将N个整数存入数组中；然后调用函

> 数fun(int[], int)，并将数组名和整数N作为该函数的实参传入；函数调用完毕后输出该函数的返回值。
>
> ---
>
> 输入
>
> > 输入包含两行：
> > 第一行是N（0 < N < 1000）。
> > 第二行是N个整数，代表数组各个元素的值，邻近两数之间用一个空格隔开。
>
> ---
>
> 输出
>
> > 输出函数fun(int[], int)调用完毕后的返回值。

方法一：老老实实用数组

```c
#include <stdio.h>
int fun(int arr[], int t) {
    int sum = 0,i;
    for (i = 0; i < t; i++) {
        sum += arr[i];
    }
    return sum;
}

int main() {
    int N,i,arr[1000],result;
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        scanf("%d", &arr[i]);
    }
    result = fun(arr, N);
    printf("%d\n", result);
}
```

方法二：不用数组，甚至不用函数（结果对但是不保证代码检查能过）

- 相比方法一，更节约内存，当然速度也稍微快一点

```c
#include <stdio.h>
int main() {
    int N,result=0,i;
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
```

```
        int a;
        scanf("%d", &a);
        result+=a;
    }
    printf("%d\n", result);
}
```

9.函数之数组降序排序

> 编写形如void fun(int arr[], int t)的函数，该函数将整型数组arr[]各个元素从大到小排序，排序之后的元素仍然存在arr[]中，形参t为数组元素个数。
> 在主函数main()中输入数组元素个数N和N个整数，并将N个整数存入数组中；然后调用函数fun(int[], int)，并将数组名和整数N作为该函数的实参传入；函数调用完毕后输出数组中各个元素的值。
>
> ---
>
> 输入
>
> > 输入包含两行：
> > 第一行是N（0 < N < 1000）。
> > 第二行是N个整数，代表数组各个元素的值，邻近两数之间用一个空格隔开。
>
> ---
>
> 输出
>
> > 函数fun(int[], int)调用完毕后，输出arr[]数组中各个元素的值，邻近两数之间用一个空格隔开。

方法一：穷举法（原始冒泡排序）
时间复杂度$O(n^2)$

```
#include <stdio.h>
void fun(int arr[], int t) {
    int i,j;
    for (i = 0; i < t-1; i++) {
        for (j = 0; j < t-i-1; j++) {
            if (arr[j] < arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
```

```
}
int main() {
    int N, arr[1000], i;
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        scanf("%d", &arr[i]);
    }
    fun(arr, N);
    for (i = 0; i < N; i++) {
        printf("%d", arr[i]);
        if (i != N-1) printf(" ");
    }
    printf("\n");
}
```

方法二：双指针法（鸡尾酒排序）
时间复杂度：最好$O(n)$，最坏$O(n^2)$，接近$O(n^2)$

```
#include <stdio.h>
void fun(int arr[], int t) {
    int left = 0,right = t - 1,j;
    while (left < right) {
        for (j = left; j < right; j++) {
            if (arr[j] < arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
        right--;
        for (j = right; j > left; j--) {
            if (arr[j - 1] < arr[j]) {
                int temp = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = temp;
            }
        }
        left++;
    }
}
int main() {
    int N, arr[1000], i;
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        scanf("%d", &arr[i]);
    }
    fun(arr, N);
    for (i = 0; i < N; i++) {
        printf("%d", arr[i]);
```

```
        if (i != N-1) printf(" ");
    }
    printf("\n");
}
```

方法三：快速排序
时间复杂度：$O(n\log n)$，最坏$O(n^2)$，但是更接近于$O(n\log n)$

```
#include <stdio.h>
void quick_sort(int arr[], int left, int right) {
    int pivot = arr[(left + right) / 2],i = left, j = right;
    if (left >= right) return;
    while (i <= j) {
        while (arr[i] > pivot) i++;
        while (arr[j] < pivot) j--;
        if (i <= j) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j--;
        }
    }
    quick_sort(arr, left, j);
    quick_sort(arr, i, right);
}
void fun(int arr[], int t) {
    quick_sort(arr, 0, t - 1);
}
int main() {
    int N, arr[1000], i;
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        scanf("%d", &arr[i]);
    }
    fun(arr, N);
    for (i = 0; i < N; i++) {
        printf("%d", arr[i]);
        if (i != N-1) printf(" ");
    }
    printf("\n");
}
```

方法四：堆排序
时间复杂度：$O(n\log n)$

```
#include <stdio.h>
void heapify(int arr[], int n, int i) {
```

```c
    int smallest = i,left = 2 * i + 1,right = 2 * i + 2;
    if (left < n && arr[left] < arr[smallest]) {
        smallest = left;
    }
    if (right < n && arr[right] < arr[smallest]) {
        smallest = right;
    }
    if (smallest != i) {
        int temp = arr[i];
        arr[i] = arr[smallest];
        arr[smallest] = temp;
        heapify(arr, n, smallest);
    }
}

void heap_sort(int arr[], int n) {
    int i;
    for (i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    for (i = n - 1; i > 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}

void fun(int arr[], int t) {
    heap_sort(arr, t);
}

int main() {
    int N, arr[1000], i;
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        scanf("%d", &arr[i]);
    }
    fun(arr, N);
    for (i = 0; i < N; i++) {
        printf("%d", arr[i]);
        if (i != N-1) printf(" ");
    }
    printf("\n");
}
```

10.函数之数组最大值问题

> 编写形如int fun(int arr[], int t)的函数，形参t为数组元素个数。该函数寻找并返回数组最大值的位置下标；如果最大值有多个，则输出第一个最大值的位置下标。
> 在主函数main()中定义一维整型数组x，输入数组元素个数N，并且依据如下公式给数组各个元素赋值：
> x[i] = N *sin(i* 0.16)
> 然后调用函数fun(int[], int)，并将数组名x和整数N作为该函数的实参传入；函数调用完毕后输出该函数的返回值。
>
> ---
>
> 输入
>
> > 输入一个整数N（0 < N < 1000）。
>
> ---
>
> 输出
>
> > 按照题目要求输出。

方法一：老老实实用数组

```c
#include <stdio.h>
#include <math.h>   // 使用数学函数 sin
int fun(int arr[], int t) {
    int max_value = arr[0],max_index = 0,i;
    for (i = 1; i < t; i++) {
        if (arr[i] > max_value) {
            max_value = arr[i];
            max_index = i;
        }
    }
    return max_index;
}
int main() {
    int N,result,i,x[1000];
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        x[i] = N * sin(i * 0.16);
    }
    result = fun(x, N);
    printf("%d\n", result);
}
```

方法二：不用数组甚至不用定义函数（当然，结果检测是对的，代码检测不一定过）

- 相比方法一，更节约内存，当然速度也稍微快一点

```c
#include <stdio.h>
#include <math.h>   // 使用数学函数 sin
#define MAX(a,b) ((a>b)?a:b)
#define FORMULA(i) N * sin(i * 0.16)
int main() {
    int N,result=0,i,a,b,arr[1000];
    scanf("%d", &N);
    a=FORMULA(0);
    for (i = 1; i < N; i++) {
        b = FORMULA(i);
        if (a<b) result=i;
        a = MAX(a,b);
    }
    printf("%d\n", result);
}
```