第一章 C++入门及简单的顺序结构

编程是一种控制计算机的方式，和我们平时双击打开文件、关机、重启没有任何区别。

一、软件环境

1. 编译软件的安装与使用

二、编写一个简单的 C++程序——手速练习

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World" << endl;
    return 0;
}
```

三、语法基础

1. 变量的定义

变量必须先定义，才可以使用。不能重名。

变量定义的方式：

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 5;
    int b, c = a, d = 10 / 2;

    return 0;
}
```

常用变量类型及范围：

类型	关键字
布尔型	bool
字符型	char
整型	int
浮点型	float
双浮点型	double

2. 输入输出

整数的输入输出：

```
#include <iostream>

using namespace std;

int main()
{
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
    return 0;
}
```

字符串的输入输出：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string str;
    cin >> str;
    cout << str;
    return 0;
}
```

输入输出多个不同类型的变量：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int a, b;
    string str;

    cin >> a;
    cin >> b >> str;

    cout << str << " !!! " << a + b << endl;

    return 0;
}
```

3. 表达式

   整数的加减乘除四则运算：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int a = 6 + 3 * 4 / 2 - 2;

    cout << a << endl;

    int b = a * 10 + 5 / 2;

    cout << b << endl;

    cout << 23 * 56 - 78 / 3 << endl;

    return 0;
}
```

运算符	描述	实例
+	把两个操作数相加	A + B 将得到 30
-	从第一个操作数中减去第二个操作数	A - B 将得到 -10
*	把两个操作数相乘	A * B 将得到 200
/	分子除以分母	B / A 将得到 2
%	取模运算符，整除后的余数	B % A 将得到 0

浮点数（小数）的运算：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    float x = 1.5, y = 3.2;

    cout << x * y << ' ' << x + y << endl;

    cout << x - y << ' ' << x / y << endl;

    return 0;
}
```

整型变量的自增、自减：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int a = 1;
    int b = a ++ ;

    cout << a << ' ' << b << endl;

    int c = ++ a;

    cout << a << ' ' << c << endl;

    return 0;
}
```

变量的类型转换：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    float x = 123.12;

    int y = (int)x;

    cout << x << ' ' << y << endl;

    return 0;
}
```

4. 顺序语句

(1) 输出第二个整数：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int a, b, c;
    cin >> a >> b >> c;
    cout << b << endl;
    return 0;
}
```

(2) 计算 (a + b) * c 的值

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int a, b, c;

    cin >> a >> b >> c;

    cout << (a + b) * c << endl;

    return 0;
}
```

(3) 带余除法

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int a, b;

    cin >> a >> b;

    int c = a / b, d = a % b;

    cout << c << ' ' << d << endl;

    return 0;
}
```

(4) 求反三位数：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int n;
    cin >> n;

    int a = n % 10;
    n = n / 10;
    int b = n % 10;
    n = n / 10;
    int c = n;

    cout << a << b << c << endl;

    return 0;
}
```

(5) 交换两个整数

swap(a,b)

(6) 输出菱形

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    char c;

    cin >> c;

    cout << "  " << c << endl;
    cout << " " << c << c << c << endl;
    cout << c << c << c << c << c << endl;
    cout << " " << c << c << c << endl;
    cout << "  " << c << endl;

    return 0;
}
```

第二章 printf 语句与 C++中的判断结构

学习语言最好的方式就是实践，每当掌握一个新功能时，就要立即将这个功能应用到实践中。

一、printf 输出格式

注意：使用 printf 时最好添加头文件 #include <cstdio>。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    printf("Hello World!");

    return 0;
}
```

1. Int、float、double、char 等类型的输出格式：

(1) Int：%d

(2) Float: %f，默认保留 6 位小数
(3) Double: %lf，　默认保留 6 位小数
(4) Char: %c，回车也是一个字符，用’ \n’ 表示

```
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a = 3;
    float b = 3.12345678;
    double c = 3.12345678;
    char d = 'y';

    printf("%d\n", a);
    printf("%f\n", b);
    printf("%lf\n", c);
    printf("%c\n", d);

    return 0;
}
```

2. 所有输出的变量均可包含在一个字符串中：

```
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a = 3;
    float b = 3.12345678;
    double c = 3.12345678;
    char d = 'y';

    printf("int a = %d, float b = %f\ndouble c = %lf, char d = %c\n", a, b, c, d);

    return 0;
}
```

练习：输入一个字符，用这个字符输出一个菱形：

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    char c;

    cin >> c;

    printf("  %c\n", c);
    printf(" %c%c%c\n", c, c, c);
    printf("%c%c%c%c%c\n", c, c, c, c, c);
    printf(" %c%c%c\n", c, c, c);
    printf("  %c\n", c);

    return 0;
}
```

练习：输入一个整数，表示时间，单位是秒。输出一个字符串，用”时:分:秒”的形式表示这个时间。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int t;

    cin >> t;

    int hours = t / 3600;
    int minutes = t % 3600 / 60;
    int seconds = t % 60;

    printf("%d:%d:%d\n", hours, minutes, seconds);

    return 0;
}
```

3. 扩展功能

(1) Float, double 等输出保留若干位小数时用：%.4f, %3lf

```
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    float b = 3.12345678;
    double c = 3.12345678;

    printf("%.4f\n", b);
    printf("%.3lf\n", c);

    return 0;
}
```

(2) 最小数字宽度

a. %8.3f，表示这个浮点数的最小宽度为 8，保留 3 位小数，当宽度不足时在前面补空格。

```
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a = 3;
    float b = 3.12345678;
    double c = 3.12345678;

    printf("%5d\n", a);
    printf("%8.4f\n", b);
    printf("%7.3lf\n", c);

    return 0;
}
```

b. %-8.3f，表示最小宽度为 8，保留 3 位小数，当宽度不足时在后面补上空格

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a = 3;
    float b = 3.12345678;
    double c = 3.12345678;

    printf("%-5d!\n", a);
    printf("%-8.4f!\n", b);
    printf("%-7.3lf!\n", c);

    return 0;
}
```

c. %08.3f，表示最小宽度为 8，保留 3 位小数，当宽度不足时在前面补上 0

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a = 3;
    float b = 3.12345678;
    double c = 3.12345678;

    printf("%05d\n", a);
    printf("%08.4f\n", b);
    printf("%07.3lf\n", c);

    return 0;
}
```

二、if 语句

1. 基本 if-else 语句

当条件成立时，执行某些语句；否则执行另一些语句。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a;
    cin >> a;

    if (a > 5)
    {
        printf("%d is big!\n", a);
        printf("%d + 1 = %d\n", a, a + 1);
    }
    else
    {
        printf("%d is small!\n", a);
        printf("%d - 1 = %d\n", a, a - 1);
    }

    return 0;
}
```

Else 语句可以省略：

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a;
    cin >> a;

    if (a > 5)
    {
        printf("%d is big!\n", a);
        printf("%d + 1 = %d\n", a, a + 1);
    }

    return 0;
}
```

当只有一条语句时，大括号可以省略：

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a;
    cin >> a;

    if (a > 5)
        printf("%d is big!\n", a);
    else
        printf("%d is small!", a);

    return 0;
}
```

练习：输入一个整数，输出这个数的绝对值。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int x;

    cin >> x;

    if (x > 0) cout << x << endl;
    else cout << -x << endl;

    return 0;
}
```

练习：输入两个整数，输出两个数中较大的那个。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a, b;
    cin >> a >> b;

    if (a > b)
        cout << a << endl;
    else
        cout << b << endl;

    return 0;
}
```

If-else 语句内部也可以是 if-else 语句。

练习：输入三个整数，输出三个数中最大的那个。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a, b, c;
    cin >> a >> b >> c;

    if (a > b)
    {
        if (a > c) cout << a << endl;
        else cout << c << endl;
    }
    else
    {
        if (b > c) cout << b << endl;
        else cout << c << endl;
    }

    return 0;
}
```

2. 常用比较运算符

(1) 大于 >

(2) 小于 <

(3) 大于等于 >=

(4) 小于等于 <=
(5) 等于 ==
(6) 不等于 !=

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a, b;
    cin >> a >> b;

    if (a > b) printf("%d > %d\n", a, b);
    if (a >= b) printf("%d >= %d\n", a, b);
    if (a < b) printf("%d < %d\n", a, b);
    if (a <= b) printf("%d <= %d\n", a, b);
    if (a == b) printf("%d == %d\n", a, b);
    if (a != b) printf("%d != %d\n", a, b);

    return 0;
}
```

3. If-else 连写：
输入一个 0 到 100 之间的分数，
如果大于等于 85，输出 A；
如果大于等于 70 并且小于 85，输出 B；
如果大于等于 60 并且小于 70，输出 C；
如果小于 60，输出 D；

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int s;
    cin >> s;

    if (s >= 85)
    {
        printf("A");
    }
    else if (s >= 70)
    {
        printf("B");
    }
    else if (s >= 60)
    {
        printf("C");
    }
    else
    {
        printf("D");
    }

    return 0;
}
```

练习：

1. 简单计算器输入两个数，以及一个运算符+, -, *, /，输出这两个数运算后的结果。当运算符是/，且除数是 0 时，输出” Divided by zero!”；当输入的字符不是+, -, *, /时，输出” Invalid operator!” 。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a, b;
    char c;
    cin >> a >> b >> c;
    if (c == '+') cout << a + b << endl;
    else if (c == '-') cout << a - b << endl;
    else if (c == '*') cout << a * b << endl;
    else if (c == '/')
    {
        if (b > 0)
        {
            cout << a / b << endl;
        }
        else
        {
            cout << "Divided by zero!" << endl;
        }
    }
    else
    {
        cout << "Invalid operator!" << endl;
    }

    return 0;
}
```

2. 判断闰年。闰年有两种情况：
(1) 能被 100 整除时，必须能被 400 整除；
(2) 不能被 100 整除时，被 4 整除即可。
输入一个年份，如果是闰年输出 yes，否则输出 no。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int year;
    cin >> year;

    if (year % 100 == 0)
    {
        if (year % 400 == 0) cout << "yes" << endl;
        else cout << "no" << endl;
    }
    else
    {
        if (year % 4 == 0) cout << "yes" << endl;
        else cout << "no" << endl;
    }

    return 0;
}
```

三、条件表达式

(1) 与 &&

(2) 或 ||

(3) 非

例题：输入三个数，输出三个数中的最大值。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int a, b, c;

    cin >> a >> b >> c;

    if (a >= b && a >= c) cout << a << endl;
    else if (b >= a && b >= c) cout << b << endl;
    else cout << c << endl;

    return 0;
}
```

练习：用一条 if 语句，判断闰年。

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main()
{
    int year;

    cin >> year;

    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
        cout << "yes" << endl;
    else
        cout << "no" << endl;

    return 0;
}
```

第三章 C++中的循环结构

学习编程语言语法是次要的，思维是主要的。如何把头脑中的想法变成简洁的代码，至关重要。

学习循环语句只需要抓住一点——代码执行顺序！

一、while 循环

可以简单理解为循环版的 if 语句。If 语句是判断一次，如果条件成立，则执行后面的语句；while 是每次判断，如果成立，则执行循环体中的语句，否则停止。

```cpp
#include <iostream>

using namespace std;

int main()
{
    int i = 0;
    while (i < 10)
    {
        cout << i << endl;
        i ++ ;
    }

    return 0;
}
```

练习：求 1~100 中所有数的立方和。

```cpp
#include <iostream>

using namespace std;

int main()
{
    int i = 1, sum = 0;
    while (i <= 100)
    {
        sum += i * i * i;
        i ++ ;
    }
    cout << sum << endl;
    return 0;
}
```

练习：求斐波那契数列的第 n 项。f(1)=1, f(2)=1, f(3)=2, f(n)=f(n-1) + f(n-2)。

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;

    int a = 1, b = 1, i = 1;
    while (i < n)
    {
        int c = a + b;
        a = b;
        b = c;
        i ++ ;
    }

    cout << a << endl;

    return 0;
}
```

死循环：循环永久执行，无法结束。我们要避免写出死循环。

```cpp
#include <iostream>

using namespace std;

int main()
{
    int x = 1;

    while (x == 1) puts("!");

    return 0;
}
```

二、do while 循环

do while 循环不常用。

do while 语句与 while 语句非常相似。唯一的区别是，do while 语句限制性循环体后检查条件。不管条件的值如何，我们都要至少执行一次循环。

```cpp
#include <iostream>

using namespace std;

int main()
{
    int x = 1;
    while (x < 1)
    {
        cout << "x!" << endl;
        x ++ ;
    }

    int y = 1;
    do
    {
        cout << "y!" << endl;
    } while (y < 1);

    return 0;
}
```

三、for 循环

基本思想：把控制循环次数的变量从循环体中剥离。

for (init-statement : condition: expression)

{

    statement

}

init-statement 可以是声明语句、表达式、空语句，一般用来初始化循环变量；

condition 是条件表达式，和 while 中的条件表达式作用一样；可以为空，空语句表示 true

expression 一般负责修改循环变量，可以为空

```cpp
#include <iostream>

using namespace std;

int main()
{
    for (int i = 0; i < 10; i ++ )
    {
        cout << i << endl;
    }

    return 0;
}
```

练习：求 1~100 中所有数的立方和。

练习：求斐波那契数列的第 n 项。f(1)=1, f(2)=1, f(3)=2, f(n)=f(n-1) + f(n-2)。

init-statement 可以定义多个变量，expression 也可以修改多个变量。

例如求 1 * 10 + 2 * 8 + 3 * 7 + 4 * 6：

```cpp
#include <iostream>

using namespace std;

int main()
{
    int sum = 0;
    for (int i = 1, j = 10; i < j; i ++, j -- )
    {
        sum += i * j;
    }

    cout << sum << endl;

    return 0;
}
```

四、跳转语句

1. break

   可以提前从循环中退出，一般与 if 语句搭配。

   例题：判断一个大于 1 的数是否是质数：

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;

    bool is_prime = true;
    for (int i = 2; i < n; i ++ )
        if (n % i == 0)
        {
            is_prime = false;
            break;
        }

    if (is_prime) cout << "yes" << endl;
    else cout << "no" << endl;

    return 0;
}
```

2. continue

   可以直接跳到当前循环体的结尾。作用与 if 语句类似。

   例题：求 1~100 中所有偶数的和。

```cpp
#include <iostream>

using namespace std;

int main()
{
    int sum = 0;

    for (int i = 1; i <= 100; i ++ )
    {
        if (i % 2 == 1) continue;
        sum += i;
    }

    cout << sum << endl;

    return 0;
}
```

五、多层循环

```cpp
#include <iostream>

using namespace std;

int main()
{
    for (int i = 0, k = 1; i < 10; i ++ )
    {
        for (int j = 0; j < 10; j ++, k ++ )
        {
            cout << k << ' ';
        }
        cout << endl;
    }

    return 0;
}
```

练习：打印 1~100 中的所有质数

```cpp
#include <iostream>

using namespace std;

int main()
{
    for (int i = 2; i <= 100; i ++ )
    {
        bool is_prime = true;
        for (int j = 2; j < i; j ++ )
        {
            if (i % j == 0)
            {
                is_prime = false;
                break;
            }
        }
        if (is_prime) cout << i << endl;
    }

    return 0;
}
```

练习：输入一个 n，打印 n 阶菱形。n 是奇数。

n=9 时的结果：

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;

    int cx = n / 2, cy = n / 2;

    for (int i = 0; i < n; i ++ )
    {
        for (int j = 0; j < n; j ++ )
            if (abs(i - cx) + abs(j - cy) <= n / 2)
                cout << "*";
            else cout << ' ';
        cout << endl;
    }

    return 0;
}
```

第四章 C++中的数组

程序 = 逻辑 + 数据，数组是存储数据的强而有力的手段。

1. 一维数组

1.1 数组的定义

数组的定义方式和变量类似。

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int a[10], b[20];
    float f[33];
    double d[123];
    char c[21];

    return 0;
}
```

1.2 数组的初始化

在 main 函数内部，未初始化的数组中的元素是随机的。

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int a[3] = {0, 1, 2};           // 含有3个元素的数组，元素分别是0,1,2
    int b[] = {0, 1, 1};            // 维度是3的数组
    int c[5] = {0, 1, 2};           // 等价于c[] = {0, 1, 2, 0, 0}
    char d[3] = {'a', 'b', 'c'};    // 字符数组的初始化

    return 0;
}
```

1.3 访问数组元素

通过下标访问数组。

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int a[3] = {0, 1, 2}; // 数组下标从0开始

    cout << a[0] << ' ' << a[1] << ' ' << a[2] << endl;

    a[0] = 5;

    cout << a[5] << endl;

    return 0;
}
```

练习题 1： 使用数组实现求斐波那契数列的第 N 项。

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int n;
    int f[100];

    cin >> n;

    f[0] = 0, f[1] = 1;
    for (int i = 2; i <= n; i ++ ) f[i] = f[i - 1] + f[i - 2];

    cout << f[n] << endl;

    return 0;
}
```

练习题 2：输入一个 n，再输入 n 个整数。将这 n 个整数逆序输出。

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int n;
    int a[100];

    cin >> n;
    for (int i = 0; i < n; i ++ ) cin >> a[i];
    for (int i = n - 1; i >= 0; i -- ) cout << a[i] << ' ';
    cout << endl;

    return 0;
}
```

练习题 3：输入一个 n，再输入 n 个整数。将这个数组顺时针旋转 k(k <= n)次，最后将结果输出。

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int n, k;
    int a[100];

    cin >> n >> k;
    for (int i = 0; i < n; i ++ ) cin >> a[i];

    reverse(a, a + k);
    reverse(a + k, a + n);
    reverse(a, a + n);

    for (int i = 0; i < n; i ++ ) cout << a[i] << ' ';
    cout << endl;

    return 0;
}
```

练习题 4：输入 n 个数，将这 n 个数按从小到大的顺序输出。

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int n, k;
    int a[100];

    cin >> n >> k;
    for (int i = 0; i < n; i ++ ) cin >> a[i];

    for (int i = 0; i < n; i ++ )
        for (int j = i + 1; j < n; j ++ )
            if (a[i] > a[j])
                swap(a[i], a[j]);

    for (int i = 0; i < n; i ++ ) cout << a[i] << ' ';
    cout << endl;

    return 0;
}
```

练习题 5：计算 2 的 N 次方。N <= 10000

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int a[10000], size = 1, n;
    a[0] = 1;

    cin >> n;
    while (n -- )
    {
        int t = 0;
        for (int i = 0; i < size; i ++ )
        {
            t += a[i] * 2;
            a[i] = t % 10;
            t /= 10;
        }
        if (t) a[size ++ ] = t;
    }

    for (int i = size - 1; i >= 0; i -- ) cout << a[i];
    cout << endl;

    return 0;
}
```

2. 多维数组

多维数组就是数组的数组。

Int a[3][4]; // 大小为 3 的数组，每个元素是含有 4 个整数的数组。

Int arr[10][20][30] = {0}; // 将所有元素初始化为 0
// 大小为 10 的数组，它的每个元素是含有 4 个整数的数组
// 这些数组的元素是含有 30 个整数的数组

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int b[3][4] = {         // 三个元素，每个元素都是大小为4的数组
        {0, 1, 2, 3},       // 第1行的初始值
        {4, 5, 6, 7},       // 第2行的初始值
        {8, 9, 10, 11}      // 第3行的初始值
    }

    return 0;
}
```

练习题：输入一个 n 行 m 列的矩阵，从左上角开始将其按回字形的顺序顺时针打印出来。

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int n, m;
    int arr[50][50];

    cin >> n >> m;
    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < m; j ++ )
            cin >> arr[i][j];

    bool st[50][50] = {false};
    int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
    int d = 1, x = 0, y = 0;
    for (int i = 0; i < n * m; i ++ )
    {
        int a = x + dx[d], b = y + dy[d];
        if (a < 0 || a >= n || b < 0 || b >= m || st[a][b])
        {
            d = (d + 1) % 4;
            a = x + dx[d], b = y + dy[d];
        }
        cout << arr[x][y] << ' ';
        st[x][y] = true;
        x = a, y = b;
    }
    cout << endl;

    return 0;
}
```

第五章 C++中的字符串

字符串是计算机与人类沟通的重要手段。

1. 字符与整数的联系——ASCII 码

每个常用字符都对应一个-128~127 的数字，二者之间可以相互转化：

```cpp
#include <iostream>

using namespace std;

int main()
{
    char c = 'a';
    cout << (int)c << endl;

    int a = 66;
    cout << (char)a << endl;
    return 0;
}
```

常用 ASCII 值：’ A’ - ‘Z’ 是 65~90，’ a’ - ‘z’ 是 97-122，’ 0’ - ‘9’ 是 48-57。

字符可以参与运算，运算时会将其当做整数：

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 'B' - 'A';
    int b = 'A' * 'B';
    char c = 'A' + 2;

    cout << a << endl;
    cout << b << endl;
    cout << c << endl;

    return 0;
}
```

练习：输入一行字符，统计出其中数字字符的个数，以及字母字符的个数。

2. 字符数组

字符串就是字符数组加上结束符’ \0’ 。

可以使用字符串来初始化字符数组，但此时要注意，每个字符串结尾会暗含一个’ \0’ 字符，因此字符数组的长度至少要比字符串的长度多 1！

```cpp
#include <iostream>

using namespace std;

int main()
{
    char a1[] = {'C', '+', '+'};            // 列表初始化，没有空字符
    char a2[] = {'C', '+', '+', '\0'};      // 列表初始化，含有显示的空字符
    char a3[] = "C++";                      // 自动添加表示字符串结尾的空字符
    char a4[6] = "Daniel";                  // 错误：没有空间可存放空字符！

    return 0;
}
```

2.1 字符数组的输入输出：

```cpp
#include <iostream>

using namespace std;

int main()
{
    char str[100];

    cin >> str;                 // 输入字符串时，遇到空格或者回车就会停止！
    cout << str << endl;        // 输出字符串时，遇到空格或者回车不会停止
    printf("%s\n", str);

    return 0;
}
```

读入一行字符串，包括空格：

```cpp
#include <iostream>

using namespace std;

int main()
{
    char str[100];

    gets(str);

    cout << str << endl;

    return 0;
}
```

2.2 字符数组的常用操作

下面几个函数需要引入头文件:
#include <string.h>

(1) strlen(str)，求字符串的长度
(2) strcmp(a, b)，比较两个字符串的大小，a < b 返回-1，a == b 返回 0，a > b 返回 1。这里的比较方式是字典序！
(3) strcpy(a, b)，将字符串 b 复制给从 a 开始的字符数组。

```
#include <iostream>
#include <string.h>

using namespace std;

int main()
{
    char a[100] = "hello world!", b[100];

    cout << strlen(a) << endl;

    strcpy(b, a);

    cout << strcmp(a, b) << endl;

    return 0;
}
```

2.3 遍历字符数组中的字符：

```
#include <iostream>
#include <string.h>

using namespace std;

int main()
{
    char a[100] = "hello world!";

    for (int i = 0; i < strlen(a); i ++ ) cout << a[i] << endl;

    return 0;
}
```

练习：给定一个只包含小写字母的字符串，请你找到第一个仅出现一次的字符。如果没有，输出 “no”。

练习：把一个字符串中特定的字符全部用给定的字符替换，得到一个新的字符串。

3. 标准库类型 string

可变长的字符序列，比字符数组更加好用。需要引入头文件：
#include <string>

3.1 定义和初始化

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s1;              // 默认初始化，s1是一个空字符串
    string s2 = s1;         // s2是s1的副本
    string s3 = "hiya";     // s3是该字符串字面值的副本
    string s4(10, 'c');     // s4的内容是 cccccccccc
    return 0;
}
```

3.2 string 上的操作

(1) string 的读写：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s1, s2;

    cin >> s1 >> s2;
    cout << s1 << s2 << endl;

    return 0;
}
```

注意：不能用 printf 直接输出 string，需要写成：printf(“%s” , s.c_str());

(2) 使用 getline 读取一整行

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s;

    getline(cin, s);

    cout << s << endl;

    return 0;
}
```

(3) string 的 empty 和 size 操作（注意 size 是无符号整数，因此 s.size() <= -1 一定成立）：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s1, s2 = "abc";

    cout << s1.empty() << endl;
    cout << s2.empty() << endl;

    cout << s2.size() << endl;

    return 0;
}
```

(4) string 的比较：

支持 > < >= <= == !=等所有比较操作，按字典序进行比较。

(5) 为 string 对象赋值：

```
string s1(10, 'c'), s2;          // s1 的内容是 cccccccccc；s2 是一个空字符串
s1 = s2;                         // 赋值：用 s2 的副本替换 s1 的副本
                                 // 此时 s1 和 s2 都是空字符串
```

(6) 两个 string 对象相加：

```
string s1 = "hello, ", s2 = "world\n";
string s3 = s1 + s2;                    // s3 的内容是 hello, world\n
s1 += s2;                               // s1 = s1 + s2
```

(7) 字面值和 string 对象相加：

做加法运算时，字面值和字符都会被转化成 string 对象，因此直接相加就是将这些字面值串联起来：

string s1 = “hello”, s2 = “world”; // 在 s1 和 s2 中都没有标点符号
string s3 = s1 + “, “ + s2 + ‘\n’;

当把 string 对象和字符字面值及字符串字面值混在一条语句中使用时，必须确保每个加法运算符的两侧的运算对象至少有一个是 string：

string s4 = s1 + “, “; // 正确：把一个 string 对象和有一个字面值相加
string s5 = “hello” +”, “; // 错误：两个运算对象都不是 string

string s6 = s1 + “, “ + “world”; // 正确，每个加法运算都有一个运算符是 string
string s7 = “hello” + “, “ + s2; // 错误：不能把字面值直接相加，运算是从左到右进行的

3.3 处理 string 对象中的字符

可以将 string 对象当成字符数组来处理：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s = "hello world";

    for (int i = 0; i < s.size(); i ++ ) cout << s[i] << endl;

    return 0;
}
```

或者使用基于范围的 for 语句：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s = "hello world";

    for (char c : s) cout << c << endl;

    for (char &c : s) c = 'a';

    cout << s << endl;

    return 0;
}
```

练习：密码翻译，输入一个只包含小写字母的字符串，将其中的每个字母替换成它的后继字母，如果原字母是’z’，则替换成’a’。

练习：输入两个字符串，验证其中一个串是否为另一个串的子串。

第六章 C++中的函数

函数让代码变得更加简洁。

1. 函数基础

一个典型的函数定义包括以下部分：返回类型、函数名字、由 0 个或多个形参组成的列表以及函数体。

1.1 编写函数

我们来编写一个求阶乘的程序。程序如下所示：

```
int fact(int val)
{
    int ret = 1;
    while (val > 1)
        ret *= val -- ;
    return ret;
}
```

函数名字是 fact，它作用于一个整型参数，返回一个整型值。return 语句负责

结束 fact 并返回 ret 的值。

1.2 调用函数

```
int main()
{
    int j = fact(5);
    cout << "5! is " << j << endl;

    return 0;
}
```

函数的调用完成两项工作：一是用实参初始化函数对应的形参，二是将控制权转移给被调用函数。此时，主调函数的执行被暂时中断，被调函数开始执行。

1.3 形参和实参

实参是形参的初始值。第一个实参初始化第一个形参，第二个实参初始化第二个形参，依次类推。形参和实参的类型和个数必须匹配。

```
fact( “hello” );          // 错误：实参类型不正确
fact();                   // 错误：实参数量不足
fact(42, 10, 0);          // 错误：实参数量过多
fact(3.14);               // 正确：该实参能转换成 int 类型，等价于 fact(3);
```

形参也可以设置默认值，但所有默认值必须是最后几个。当传入的实参个数少于形参个数时，最后没有被传入值的形参会使用默认值。

1.4 函数的形参列表

函数的形参列表可以为空，但是不能省略。

```
void f1() {/* …. */}              // 隐式地定义空形参列表
void f2(void) {/* … */}           // 显式地定义空形参列表
```

形参列表中的形参通常用逗号隔开，其中每个形参都是含有一个声明符的声明。即使两个形参的类型一样，也必须把两个类型都写出来：

```
int f3(int v1, v2) {/* … */}          // 错误
int f4(int v1, int v2) {/* … */}      // 正确
```

1.5 函数返回类型

大多数类型都能用作函数的返回类型。一种特殊的返回类型是 void，它表示函数不返回任何值。函数的返回类型不能是数组类型或函数类型，但可以是指向数组或者函数的指针。

1.6 局部变量、全局变量与静态变量

局部变量只可以在函数内部使用，全局变量可以在所有函数内使用。当局部变量与全局变量重名时，会优先使用局部变量。

2. 参数传递

2.1 传值参数

当初始化一个非引用类型的变量时，初始值被拷贝给变量。此时，对变量的改动不会影响初始值。

```
#include <iostream>

using namespace std;

int f(int x)
{
    x = 5;
}

int main()
{
    int x = 10;

    f(x);
    cout << x << endl;

    return 0;
}
```

2.2 传引用参数

当函数的形参为引用类型时，对形参的修改会影响实参的值。使用引用的作用：避免拷贝、让函数返回额外信息。

```
#include <iostream>

using namespace std;

int f(int &x)
{
    x = 5;
}

int main()
{
    int x = 10;

    f(x);
    cout << x << endl;

    return 0;
}
```

2.3 数组形参

在函数中对数组中的值的修改，会影响函数外面的数组。

一维数组形参的写法：

```
// 尽管形式不同，但这三个 print 函数是等价的
void print(int *a) {/* … */}
void print(int a[]) {/* … */}
void print(int a[10]) {/* … */}
```

```cpp
#include <iostream>

using namespace std;

void print(int a[])
{
    for (int i = 0; i < 10; i ++ ) cout << a[i] << endl;
}

int main()
{
    int a[10];

    for (int i = 0; i < 10; i ++ ) a[i] = i;

    print(a);

    return 0;
}
```

多维数组形参的写法：

```
// 多维数组中，除了第一维之外，其余维度的大小必须指定
void print(int (*a)[10]) {/* … */}
void print(int a[][10]) {/* … */}
```

```cpp
#include <iostream>

using namespace std;

void print(int a[][10])
{
    for (int i = 0; i < 10; i ++ )
    {
        for (int j = 0; j < 10; j ++ )
            cout << a[i][j] << ' ';
        cout << endl;
    }
}

int main()
{
    int a[10][10];

    for (int i = 0; i < 10; i ++ )
        for (int j = 0; j < 10; j ++ )
            a[i][j] = j;

    print(a);

    return 0;
}
```

3. 返回类型和 return 语句

return 语句终止当前正在执行的函数并将控制权返回到调用该函数的地方。return 语句有两种形式：

return;

return *expression*;

3.1 无返回值函数

没有返回值的 return 语句只能用在返回类型是 void 的函数中。返回 void 的函数不要求非得有 return 语句，因为在这类函数的最后一句后面会隐式地执行 return。

通常情况下，void 函数如果想在它的中间位置提前退出，可以使用 return 语句。return 的这种用法有点类似于我们用 break 语句退出循环。

```
void swap(int &v1, int &v2)
{
    // 如果两个值相等，则不需要交换，直接退出
    if (v1 == v2)
        return;
    // 如果程序执行到了这里，说明还需要继续完成某些功能

    int tmp = v2;
    v2 = v1;
    v1 = tmp;
    // 此处无须显示的 return 语句
}
```

3.2 有返回值的函数

只要函数的返回类型不是 void，则该函数内的每条 return 语句必须返回一个值。return 语句返回值的类型必须与函数的返回类型相同，或者能隐式地转换函数的返回类型。

```
#include <iostream>

using namespace std;

int max(int x, int y)
{
    if (x > y) return x;

    return y;
}

int main()
{
    int x, y;
    cin >> x >> y;

    cout << max(x, y) << endl;

    return 0;
}
```

4. 函数递归

在一个函数内部，也可以调用函数本身。

```cpp
#include <iostream>

using namespace std;

int fact(int n)
{
    if (n <= 1) return 1;
    return n * fact(n - 1);
}

int main()
{
    int n;
    cin >> n;

    cout << fact(n) << endl;

    return 0;
}
```

第七章 类、结构体、指针、引用

类可以将变量、数组和函数完美地打包在一起。

1. 类与结构体

   类的定义：

```cpp
class Person
{
    private:
        int age, height;
        double money;
        string books[100];

    public:
        string name;

        void say()
        {
            cout << "I'm " << name << endl;
        }

        int get_age()
        {
            return age;
        }

        void add_money(double x)
        {
            money += x;
        }
};
```

类中的变量和函数被统一称为类的成员变量。

private 后面的内容是私有成员变量，在类的外部不能访问；public 后面的内容是公有成员变量，在类的外部可以访问。

类的使用：

```
#include <iostream>

using namespace std;

const int N = 1000010;

class Person
{
    private:
        int age, height;
        double money;
        string books[100];

    public:
        string name;

        void say()
        {
            cout << "I'm " << name << endl;
        }

        int set_age(int a)
        {
            age = a;
        }

        int get_age()
        {
            return age;
        }

        void add_money(double x)
        {
            money += x;
        }
} person_a, person_b, persons[100];

int main()
{
    Person c;
```

```
    c.name = "yxc";          // 正确！访问公有变量
    c.age = 18;              // 错误！访问私有变量
    c.set_age(18);           // 正确！set_age()是共有成员变量
    c.add_money(100);

    c.say();
    cout << c.get_age() << endl;

    return 0;
}
```

结构体和类的作用是一样的。不同点在于类默认是 private，结构体默认是 public。

```
struct Person
{
    private:
        int age, height;
        double money;
        string books[100];

    public:
        string name;

        void say()
        {
            cout << "I'm " << name << endl;
        }

        int set_age(int a)
        {
            age = a;
        }

        int get_age()
        {
            return age;
        }

        void add_money(double x)
        {
            money += x;
        }
} person_a, person_b, persons[100];
```

2. 指针和引用

   指针指向存放变量的值的地址。因此我们可以通过指针来修改变量的值。

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10;
    int *p = &a;
    *p += 5;
    cout << *p << endl;

    return 0;
}
```

数组名是一种特殊的指针。指针可以做运算：

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a[5] = {1, 2, 3, 4, 5};

    for (int i = 0; i < 5; i ++ ) cout << *(a + i) << endl;

    return 0;
}
```

引用和指针类似，相当于给变量起了个别名。

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10;
    int &p = a;
    p += 5;
    cout << p << endl;

    return 0;
}
```

3. 链表

```cpp
#include <iostream>
#include <string.h>

using namespace std;

const int N = 10000;

struct Node
{
    int val;
    Node *next;
} *head;

int main()
{
    for (int i = 1; i <= 5; i ++ )
    {
        Node *p = new Node();
        p->val = i;
        p->next = head;
        head = p;
    }

    for (Node *p = head; p; p = p->next) cout << p->val << ' ';
    cout << endl;

    return 0;
}
```

第八章 C++ STL

STL 是提高 C++编写效率的一个利器。

1. #include <vector>

vector 是变长数组，支持随机访问，不支持在任意位置 O(1)插入。为了保证效率，元素的增删一般应该在末尾进行。

声明

#include <vector>	头文件
vector<int> a;	相当于一个长度动态变化的 int 数组
vector<int> b[233];	相当于第一维长 233，第二位长度动态变化的 int 数组
struct rec{…};	
vector<rec> c;	自定义的结构体类型也可以保存在 vector 中

size/empty

size 函数返回 vector 的实际长度（包含的元素个数），empty 函数返回一个 bool 类型，表明 vector 是否为空。二者的时间复杂度都是 O(1)。

所有的 STL 容器都支持这两个方法，含义也相同，之后我们就不再重复给出。

clear

clear 函数把 vector 清空。

迭代器

迭代器就像 STL 容器的“指针”，可以用星号“*”操作符解除引用。

一个保存 int 的 vector 的迭代器声明方法为：

vector<int>::iterator it;

vector 的迭代器是“随机访问迭代器”，可以把 vector 的迭代器与一个整数相加减，其行为和指针的移动类似。可以把 vector 的两个迭代器相减，其结果也和指针相减类似，得到两个迭代器对应下标之间的距离。

begin/end

begin 函数返回指向 vector 中第一个元素的迭代器。例如 a 是一个非空的 vector，则*a.begin()与 a[0]的作用相同。

所有的容器都可以视作一个“前闭后开”的结构，end 函数返回 vector 的尾部，即第 n 个元素再往后的“边界”。*a.end()与 a[n]都是越界访问，其中 n=a.size()。

下面两份代码都遍历了 vector<int>a，并输出它的所有元素。

```
for (int l = 0; l < a.size(); l ++) cout << a[i] << endl;
for (vector<int>::iterator it = a.begin(); it != a.end(); it ++) cout << *it << endl;
```

front/back

front 函数返回 vector 的第一个元素，等价于*a.begin() 和 a[0]。

back 函数返回 vector 的最后一个元素，等价于*==a.end() 和 a[a.size() – 1]。

push_back() 和 pop_back()

a.push_back(x) 把元素 x 插入到 vector a 的尾部。

b.pop_back() 删除 vector a 的最后一个元素。

2. #include <queue>

头文件 queue 主要包括循环队列 queue 和优先队列 priority_queue 两个容器。

声明

```
queue<int> q;
struct rec{…}; queue<rec> q;      //结构体 rec 中必须定义小于号
priority_queue<int> q;        //  大根堆
priority_queue<int, vector<int>, greater<int> q;     //  小根堆
priority_queue<pair<int, int>>q;
```

循环队列 queue

push 从队尾插入

pop 从队头弹出

front 返回队头元素

back 返回队尾元素

优先队列 priority_queue

push 把元素插入堆

pop 删除堆顶元素

top 查询堆顶元素（最大值）

3. #include <stack>

头文件 stack 包含栈。声明和前面的容器类似。

push 向栈顶插入

pop 弹出栈顶元素

4. #include <deque>

双端队列 deque 是一个支持在两端高效插入或删除元素的连续线性存储空间。它就像是 vector 和 queue 的结合。与 vector 相比，deque 在头部增删元素仅需要 O(1)的时间；与 queue 相比，deque 像数组一样支持随机访问。

[] 随机访问

begin/end，返回 deque 的头/尾迭代器

front/back 队头/队尾元素

push_back 从队尾入队

push_front 从队头入队

pop_back 从队尾出队

pop_front 从队头出队

clear 清空队列

5. #include <set>

头文件 set 主要包括 set 和 multiset 两个容器，分别是“有序集合”和“有序多重集合”，即前者的元素不能重复，而后者可以包含若干个相等的元素。set 和 multiset 的内部实现是一棵红黑树，它们支持的函数基本相同。

声明

```
set<int> s;
struct rec{…}; set<rec> s;    // 结构体 rec 中必须定义小于号
multiset<double> s;
```

size/empty/clear

与 vector 类似

迭代器

set 和 multiset 的迭代器称为“双向访问迭代器”，不支持“随机访问”，支持星号(*)解除引用，仅支持”++”和--“两个与算术相关的操作。

设 it 是一个迭代器，例如 set<int>::iterator it;

若把 it++，则 it 会指向“下一个”元素。这里的“下一个”元素是指在元素从小到大排序的结果中，排在 it 下一名的元素。同理，若把 it--，则 it 将会指向排在“上一个”的元素。

begin/end

返回集合的首、尾迭代器，时间复杂度均为 O(1)。

s.begin() 是指向集合中最小元素的迭代器。

s.end() 是指向集合中最大元素的下一个位置的迭代器。换言之，就像 vector 一样，是一个“前闭后开”的形式。因此--s.end()是指向集合中最大元素的迭代器。

insert

s.insert(x)把一个元素 x 插入到集合 s 中，时间复杂度为 O(logn)。

在 set 中，若元素已存在，则不会重复插入该元素，对集合的状态无影响。

find

s.find(x) 在集合 s 中查找等于 x 的元素，并返回指向该元素的迭代器。若不存在，则返回 s.end()。时间复杂度为 O(logn)。

lower_bound/upper_bound

这两个函数的用法与 find 类似，但查找的条件略有不同，时间复杂度为 O(logn)。

s.lower_bound(x) 查找大于等于 x 的元素中最小的一个，并返回指向该元素的迭代器。

s.upper_bound(x) 查找大于 x 的元素中最小的一个，并返回指向该元素的迭代器。

erase

设 it 是一个迭代器，s.erase(it) 从 s 中删除迭代器 it 指向的元素，时间复杂度为 O(logn)

设 x 是一个元素，s.erase(x) 从 s 中删除所有等于 x 的元素，时间复杂度为 O(k+logn)，其中 k 是被删除的元素个数。

count

s.count(x) 返回集合 s 中等于 x 的元素个数，时间复杂度为 O(k +logn)，其中 k 为元素 x 的个数。

6. #include <map>

map 容器是一个键值对 key-value 的映射，其内部实现是一棵以 key 为关键码的红黑树。Map 的 key 和 value 可以是任意类型，其中 key 必须定义小于号运算符。

声明

map<key_type, value_type> name;

例如：

map<long, long, bool> vis;

map<string, int> hash;

map<pair<int, int>, vector<int>> test;

size/empty/clear/begin/end 均与 set 类似。

Insert/erase

与 set 类似，但其参数均是 pair<key_type, value_type>。

find

h.find(x) 在变量名为 h 的 map 中查找 key 为 x 的二元组。

[]操作符

h[key] 返回 key 映射的 value 的引用，时间复杂度为 O(logn)。

[]操作符是 map 最吸引人的地方。我们可以很方便地通过 h[key]来得到 key 对应的 value，还可以对 h[key]进行赋值操作，改变 key 对应的 value。

第九章 位运算与常用库函数

C++帮我们实现好了很多有用的函数，我们要避免重复造轮子。

1. 位运算

& 与

| 或

~ 非

^ 异或

>> 右移

<< 左移

常用操作：

(1) 求 x 的第 k 位数字 x >> k & 1

(2) lowbit(x) = x & -x，返回 x 的最后一位 1

2. 常用库函数、

(1) reverse 翻转

翻转一个 vector：

```
reverse(a.begin(), a.end());
```

翻转一个数组，元素存放在下标 1~n：

```
reverse(a + 1, a + 1 + n);
```

(2) unique 去重

返回去重之后的尾迭代器（或指针），仍然为前闭后开，即这个迭代器是去重之后末尾元素的下一个位置。该函数常用于离散化，利用迭代器（或指针）的减法，可计算出去重后的元素个数。

把一个 vector 去重：

```
int m = unique(a.begin(), a.end()) – a.begin();
```

把一个数组去重，元素存放在下标 1~n：

```
int m = unique(a + 1, a + 1 + n) – (a + 1);
```

(3) random_shuffle 随机打乱

用法与 reverse 相同

(4) sort

对两个迭代器（或指针）指定的部分进行快速排序。可以在第三个参数传入定义大小比较的函数，或者重载“小于号”运算符。

把一个 int 数组（元素存放在下标 1~n）从大到小排序，传入比较函数：

```
int a[MAX_SIZE];
bool cmp(int a, int b) {return a > b; }
sort(a + 1, a + 1 + n, cmp);
```

把自定义的结构体 vector 排序，重载“小于号”运算符：

```
struct rec{ int id, x, y; }
vector<rec> a;
bool operator <(const rec &a, const rec &b) {
    return a.x < b.x || a.x == b.x && a.y < b.y;
}
sort(a.begin(), a.end());
```

(5) lower_bound/upper_bound 二分

lower_bound 的第三个参数传入一个元素 x，在两个迭代器（指针）指定的部分上执行二分查找，返回指向第一个大于等于 x 的元素的位置的迭代器（指针）。

upper_bound 的用法和 lower_bound 大致相同，唯一的区别是查找第一个大于 x 的元素。当然，两个迭代器（指针）指定的部分应该是提前排好序的。

在有序 int 数组（元素存放在下标 1~n）中查找大于等于 x 的最小整数的下标：

```
int l = lower_bound(a + 1, a + 1 + n,. x) – a;
```

在有序 vector<int> 中查找小于等于 x 的最大整数（假设一定存在）：

```
int y = *--upper_bound(a.begin(), a.end(), x);
```