# C++

极夜酱

# 目录

**1 Hello World! 1**

1.1 Hello World! . . . . . 1

1.2 数据类型 . . . . . 5

1.3 输入输出 . . . . . 8

1.4 表达式 . . . . . 12

**2 分支 16**

2.1 逻辑运算符 . . . . . 16

2.2 if . . . . . 18

2.3 switch . . . . . 22

**3 循环 24**

3.1 while . . . . . 24

3.2 for . . . . . 30

3.3 break or continue? . . . . . 36

**4 数组 39**

4.1 数组 . . . . . 39

4.2 字符串 . . . . . 45

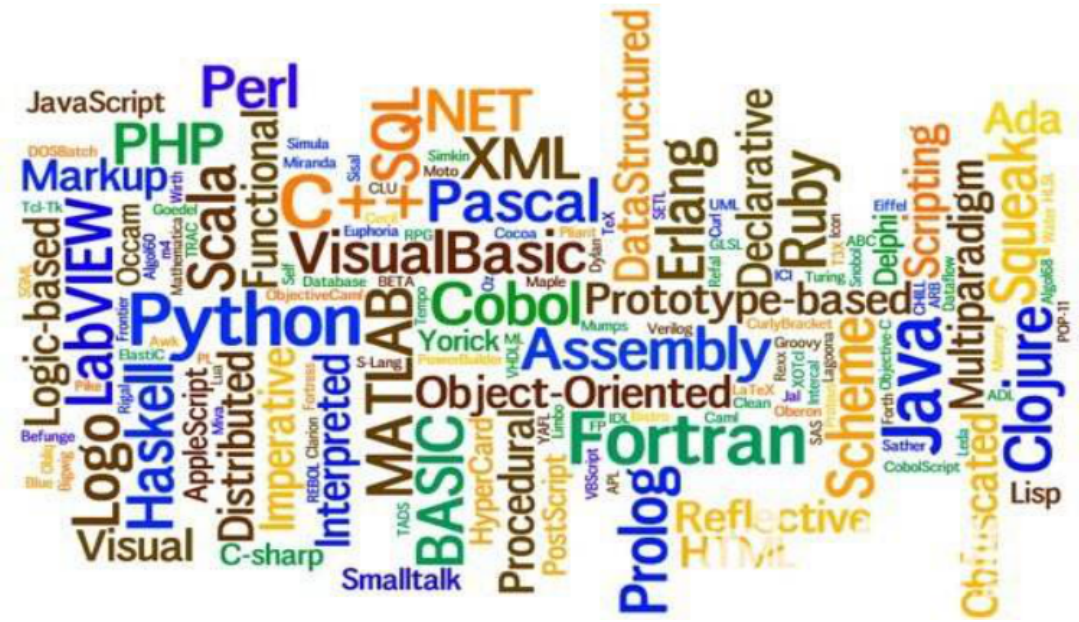
# Chapter 1 Hello World!

## 1.1 Hello World!

### 1.1.1 编程语言（Programming Language）

程序是为了让计算机去解决某些问题，它由一系列指令构成。但是计算机并不能理解人类的语言，即使是最简单的，例如“计算一下 1+2 是多少”。

计算机采用的是二进制（binary），也就是只能够理解 0 和 1，因此编程语言用于作为人类与计算机之间沟通的桥梁。




通过使用编程语言来描述解决问题的步骤，从而让计算机一步一步去执行。流程图（flow chat）成为了一种程序的图形化表示方式。

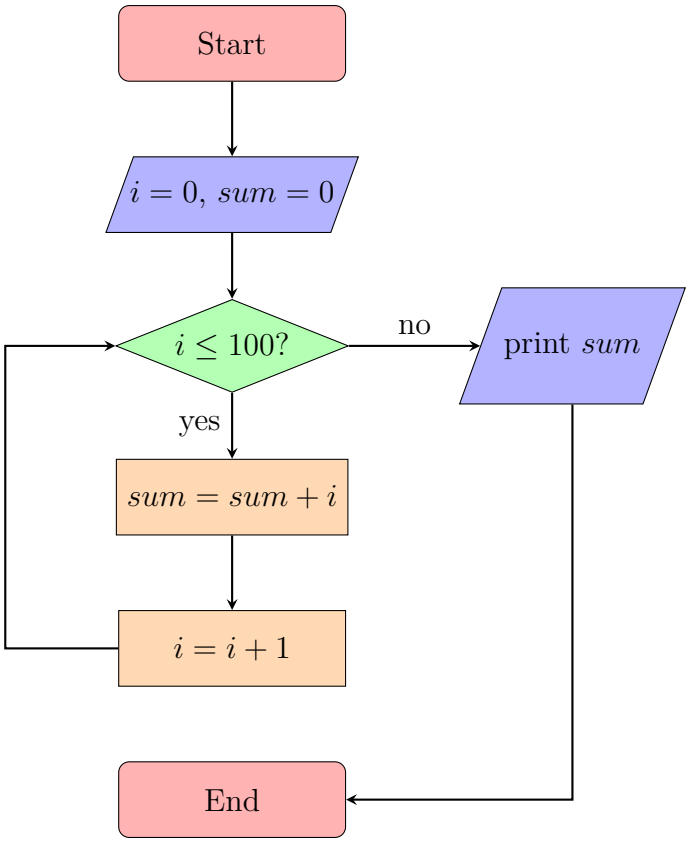



图 1.1: 计算 $\sum_{i=1}^{100} i$ 的流程图

### 1.1.2 Hello World!

Hello World 是学习编程的第一个程序，它的作用是向屏幕输出"Hello World!"。

**Hello World!**

```cpp
#include <iostream>

using namespace std;

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

**运行结果**

```
Hello World!
```

#include <stdio.h> 用于包含输入输出库的头文件（header file），这样才能够在程序中进行输入输出相关的操作。

using namespace std 表示使用 std 命名空间。

main() 是程序的入口，程序运行后会首先执行 main() 中的代码。cout 的功能是在屏幕上输出数据，endl 表示输出一个换行符。最后的分号用于表示一条语句的结束，注意不要使用中文的分号。

return 0 表示 main() 运行结束，返回值为 0，一般返回 0 用于表示程序正常结束。

不同编程语言的 Hello World 写法大同小异，可以看出编程语言的基本结构是相似的。

**C**

```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

**Java**

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
```

```
    }
}
```

Python

```
print("Hello World!")
```

### 1.1.3 注释（Comment）

注释就是对代码的解释和说明，它并不会程序所执行。注释能提高程序的可读性，让人更加容易了解代码的功能。

注释一般分为单行注释和多行注释：

1. 单行注释：以//开头，该行之后的内容视为注释。
2. 多行注释：以/* 开头，*/结束，中间的内容视为注释。

注释

```
/*
* Author: Terry
* Date: 2022/11/16
*/

#include <iostream>      // header file

using namespace std;

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

## 1.2 数据类型

### 1.2.1 数据类型（Data Types）

在计算机中，每个数据一般都有一个对应的类型，基础数据类型包括：

1. 整型
    - 短整型 short
    - 整型 int
    - 长整型 long
    - 长长整型 long long
2. 浮点型
    - 单精度浮点数 float
    - 双精度浮点数 double
3. 字符型 char

不同的数据类型所占的内存空间大小不同，因此所能表示的数值范围也不同。

| 数据类型 | 大小 | 取值范围 |
|---|---|---|
| short | 2 字节 | $-2^{15} \sim 2^{15}-1$ |
| int | 4 字节 | $-2^{31} \sim 2^{31}-1$ |
| long | 4 字节 | $-2^{31} \sim 2^{31}-1$ |
| long long | 8 字节 | $-2^{63} \sim 2^{63}-1$ |
| float | 4 字节 | 7 位有效数字 |
| double | 8 字节 | 15 位有效数字 |
| char | 1 字节 | $-128 \sim 127$ |

### 1.2.2 变量（Variable）

变量是用来存储数据的内存空间，每个变量都有一个类型，变量中只能存储对应类型的数据。

```
1 int num = 10;
2 double salary = 8232.56;
```

变量的命名需要符合规范：

1. 由字母、数字和下划线组成，不能以数字开头
2. 不可以使用编程语言中预留的关键字
3. 使用英语单词，顾名思义

关键字是编程语言内置的一些名称，具有特殊的用处和意义，因此不应该作为变量名，防止产生歧义。

| asm | auto | break | case | catch |
|---|---|---|---|---|
| char | class | const | continue | default |
| delete | do | double | else | enum |
| extern | float | for | friend | goto |
| if | inline | int | long | new |
| operator | private | protected | public | register |
| return | short | signed | sizeof | static |
| struct | switch | template | this | throw |
| try | typedef | union | unsigned | virtual |
| void | volatile | while | | |

表 1.1: 关键字

### 1.2.3 常量（Constant）

变量的值在程序运行过程中可以修改，但有一些数据的值是固定的，为了防止这些数据被随意改动，可以将这些数据定义为常量。

在数据类型前加上 const 关键字，即可定义常量，常量一般使用大写表示。如果在程序中尝试修改常量，将会报错。

**常量**

```
#include <iostream>

using namespace std;

int main()
{
    const double PI = 3.1415;
    PI = 4;
    return 0;
}
```

**运行结果**

error: assignment of read-only variable "PI"

## 1.3 输入输出

### 1.3.1 cout

cout 是输出流对象，用来向屏幕输出数据。但是有些需要输出的字符在编程语言中具有特殊含义，因此这些特殊的字符，需要经过转义后输出。

| 转义字符 | 描述 |
| --- | --- |
| \\ | 反斜杠 \ |
| \' | 单引号 ' |
| \" | 双引号 " |
| \n | 换行 |
| \t | 制表符 |

表 1.2: 转义字符

**转义字符**

```
#include <iostream>

using namespace std;

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

**运行结果**

```
"Hello
World"
```

在对变量的值进行输出时，可以使用格式控制符改变输出的格式。

**长方形面积**

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    int length = 10;
    int width = 5;
    double area = length * width;

    cout << "Area = " << length << " * " << width << " = "
            << fixed << setprecision(2) << area << endl;
    return 0;
}
```

**运行结果**

```
Area = 10 * 5 = 50.00
```

### 1.3.2 printf()

printf() 是 C 语言中的输出函数，包含在头文件 stdio.h 中，用于向屏幕输出指定格式的文本，使用对应类型的占位符可以更加方便地输出变量的值。

| 数据类型 | 占位符 |
|---|---|
| int | %d |
| float | %f |
| double | %f |
| char | %c |

表 1.3: 占位符

```
printf("Area = %d * %d = %.2f\n", length, width, area);
```

### 1.3.3 cin

有时候一些数据需要从键盘输入，cin 可以读取对应类型的数据，并赋值给相应的变量。

在使用 cin 前，通常会使用 cout 先输出一句提示信息，告诉用户需要输入什么数据。

**圆面积**

```
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

int main()
{
    const double PI = 3.14159;
    double r;
    double area;

    cout << "Radius: ";
    cin >> r;

    area = PI * pow(r, 2);
    cout << "Area = " << fixed << setprecision(2) << area << endl;

    return 0;
}
```

**运行结果**

```
Radius: 5
Area = 78.54
```

头文件 cmath 中定义了一些常用的数学函数，例如 pow(x, y) 可用于计算 $x$ 的 $y$ 次方。

## 1.4 表达式

### 1.4.1 算术运算符

大部分编程语言中的除法与数学中的除法意义不同。

当相除的两个数都为整数时，那么就会进行整除运算，因此结果仍为整数，例如 $21 \ / \ 4 = 5$。

如果相除的两个数中至少有一个为浮点数时，那么就会进行普通的除法运算，结果为浮点数，例如 $21.0 \ / \ 4 = 5.25$。

取模（modulo）运算符% 用于计算两个整数相除之后的余数，例如 $22 \ \% \ 3 = 1$、$4 \ \% \ 7 = 4$。

**逆序三位数**

```cpp
#include <iostream>

using namespace std;

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

    cout << "Enter a 3-digit integer: ";
    cin >> num;

    a = num / 100;
    b = num / 10 % 10;
    c = num % 10;

    cout << "Reversed: " << c*100 + b*10 + a << endl;
    return 0;
```

```
19 }
```

**运行结果**

```
Enter a 3-digit integer: 520
Reversed: 25
```

### 1.4.2 复合运算符

使用复合运算符可以使表达式更加简洁。例如 `a = a + b`可以写成 `a += b`，-=、*=、/=、%= 等复合运算符的使用方式同理。

当需要给一个变量的值加/减 1 时，除了可以使用 `a += 1`或 `a -= 1`之外，还可以使用 ++ 或--运算符，但是 `++`和 `--`可以出现在变量之前或之后：

| 表达式 | 含义 |
|---|---|
| a++ | 执行完所在语句后自增 1 |
| ++a | 在执行所在语句前自增 1 |
| a-- | 执行完所在语句后自减 1 |
| --a | 在执行所在语句前自减 1 |

表 1.4: 自增/自减运算符

**自增/自减运算符**

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int n = 10;
8
```

```
9     cout << n++ << endl;
10    cout << ++n << endl;
11    cout << n-- << endl;
12    cout << --n << endl;
13
14    return 0;
15 }
```

**运行结果**

```
10
12
12
10
```

### 1.4.3 隐式类型转换

在计算机计算的过程中，只有类型相同的数据才可以进行运算。例如整数 + 整数、浮点数/浮点数等。

但是很多时候，我们仍然可以对不同类型的数据进行运算，而并不会产生错误，例如整数 + 浮点数。这是由于编译器会自动进行类型转换。在整数 + 浮点数的例子中，编译器会将整数转换为浮点数，这样就可以进行运算了。

编译器选择将整数转换为浮点数，而不是将浮点数转换为整数的原因在于，浮点数相比整数能够表示的范围更大。例如整数 8 可以使用 8.0 表示，而浮点数 9.28 变为整数 9 后就会丢失精度。

隐式类型转换最常见的情形就是除法运算，这也是导致整数/整数 = 整数、整数/浮点数 = 浮点数的原因。

### 1.4.4 显式类型转换

有些时候编译器无法自动进行类型转换，这时就需要我们手动地强制类型转换。

**显式类型转换**

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    int total = 821;
    int num = 10;
    double average = (double)total / num;
    cout << "Average = " << fixed << setprecision(2) << average << endl;
    return 0;
}
```

**运行结果**

```
Average = 82.10
```

# Chapter 2 分支

## 2.1 逻辑运算符

### 2.1.1 关系运算符

编程中经常需要使用关系运算符来比较两个数据的大小，比较的结果是一个布尔值（boolean），即 True（非 0）或 False（0）。

在编程中需要注意，一个等号 = 表示赋值运算，而两个等号 == 表示比较运算。

| 数学符号 | 关系运算符 |
|---|---|
| $<$ | < |
| $>$ | > |
| $\leq$ | <= |
| $\geq$ | >= |
| $=$ | == |
| $\neq$ | != |

### 2.1.2 逻辑运算符

逻辑运算符用于连接多个关系表达式，其结果也是一个布尔值。

1. 逻辑与 &&：当多个条件全部为 True，结果为 True。

| **条件 1** | **条件 2** | **条件 1 && 条件 2** |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

2. 逻辑或 ||：多个条件至少有一个为 True 时，结果为 True。

| **条件 1** | **条件 2** | **条件 1 \|\| 条件 2** |
|---|---|---|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

3. 逻辑非!：条件为 True 时，结果为 False；条件为 False 时，结果为 True。

| **条件** | **! 条件** |
|---|---|
| T | F |
| F | T |

## 2.2 if

### 2.2.1 if

if 语句用于判断一个条件是否成立，如果成立则进入语句块，否则不执行。

**年龄**

```
#include <iostream>

using namespace std;

int main()
{
    int age;
    cout << "Enter your age: ";
    cin >> age;
    if(age > 0 && age < 18)
    {
        cout << "Minor" << endl;
    }
    return 0;
}
```

**运行结果**

```
Enter your age: 17
Minor
```

### 2.2.2 if-else

if-else 的结构与 if 类似，只是在 if 语句块中的条件不成立时，执行 else 语句块中的语句。

## 闰年

```cpp
#include <iostream>

using namespace std;

int main()
{
    int year;
    cout << "Enter a year: ";
    cin >> year;

    /*
        * A year is a leap year if it is
        * 1. exactly divisible by 4, and not divisible by 100;
        * 2. or is exactly divisible by 400
        */
    if((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
    {
        cout << "Leap year" << endl;
    }
    else
    {
        cout << "Common year" << endl;
    }

    return 0;
}
```

### 运行结果

```
Enter a year: 2020
Leap year
```

### 2.2.3 if-else if-else

当需要对更多的条件进行判断时，可以使用 if-else if-else 语句。

字符

```cpp
#include <iostream>

using namespace std;

int main()
{
    char c;
    cout << "Enter a character: ";
    cin >> c;

    if(c >= 'a' && c <= 'z')
    {
        printf("Lowercase\n");
    }
    else if(c >= 'A' && c <= 'Z')
    {
        printf("Uppercase\n");
    }
    else if(c >= '0' && c <= '9')
    {
        printf("Digit\n");
    }
    else
    {
        printf("Special character\n");
    }

    return 0;
}
```

**运行结果**

```
Enter a character: T
Uppercase
```

## 2.3 switch

### 2.3.1 switch

switch 结构用于根据一个整数值，选择对应的 case 执行。需要注意的是，当对应的 case 中的代码被执行完后，并不会像 if 语句一样跳出 switch 结构，而是会继续向后执行，直到遇到 break。

**计算器**

```
#include <iostream>

using namespace std;

int main() {
    int num1, num2;
    char op;

    cout << "Enter an expression: ";
    cin >> num1 >> op >> num2;

    switch (op)
    {
    case '+':
        cout << num1 << " + " << num2 << " = " << num1 + num2 << endl;
        break;
    case '-':
        cout << num1 << " - " << num2 << " = " << num1 - num2 << endl;
        break;
    case '*':
        cout << num1 << " * " << num2 << " = " << num1 * num2 << endl;
        break;
    case '/':
        cout << num1 << " / " << num2 << " = " << num1 / num2 << endl;
        break;
    default:
```

```
27          cout << "Error! Operator is not supported" << endl;
28          break;
29      }
30
31      return 0;
32  }
```

**运行结果**

```
Enter an expression: 5 * 8
5 * 8 = 40
```

# Chapter 3 循环

## 3.1 while

### 3.1.1 while

while 循环会对条件进行判断，如果条件成立，就会执行循环体，然后再次判断条件，直到条件不成立。

while 循环的次数由循环变量的变化决定，因此 while 循环一般都包括对循环变量的初值、判断和更新。

```
1 int i = 1;          // initial value
2 while(i <= 5)       // condition
3 {
4     cout << "In loop: i = " << i << endl;
5     i++;            // update
6 }
7 cout << "After loop: i = " << i << endl;
```

while 循环的特点是先判断、再执行，因此循环体有可能会执行一次或多次，也有可能一次也不会执行。

**平均身高**

```
1 #include <iostream>
2 #include <iomanip>
3
4 #define NUM_PEOPLE 5
5
6 using namespace std;
7
8 int main()
```

```
{
    double height;
    double total = 0;

    int i = 1;
    while (i <= NUM_PEOPLE)
    {
        cout << "Enter person " << i << "'s height: ";
        cin >> height;
        total += height;
        i++;
    }

    double average = total / NUM_PEOPLE;
    cout << "Average height: "
         << fixed << setprecision(2) << average << endl;

    return 0;
}
```

**运行结果**

```
Enter person 1's height: 160.8
Enter person 2's height: 175.2
Enter person 3's height: 171.2
Enter person 4's height: 181.3
Enter person 5's height: 164
Average height: 170.50
```

**统计元音、辅音数量**

```
#include <iostream>

using namespace std;

```

```cpp
int main()
{
    char c;
    int vowel = 0;
    int consonant = 0;

    cout << "Enter an English sentence: ";

    while((c = cin.get()) != '\n')
    {
        if (c == 'a' || c == 'A' ||
            c == 'e' || c == 'E' ||
            c == 'i' || c == 'I' ||
            c == 'o' || c == 'O' ||
            c == 'u' || c == 'U')
        {
            vowel++;
        }
        else if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        {
            consonant++;
        }
    }

    cout << "Vowel = " << vowel << endl;
    cout << "Consonant = " << consonant << endl;
    return 0;
}
```

**运行结果**

```
Enter an English sentence: Hello World!
Vowel = 3
Consonant = 7
```

### 3.1.2 do-while

do-while 循环是先执行一轮循环体内的代码后，再检查循环的条件是否成立。如果成立，则继续下一轮循环；否则循环结束。

do-while 循环是先执行、再判断，因此它至少会执行一轮循环。do-while 一般应用在一些可能会需要重复，但必定会发生一次的情景下。例如登录账户，用户输入账户和密码后，检查是否正确，如果正确，那么就成功登录；否则继续循环让用户重新输入。

需要注意，do-while 循环的最后有一个分号。

```
do {
    // code
} while(condition);
```

**整数位数**

```
#include <iostream>

using namespace std;

int main()
{
    int num;
    int n = 0;

    cout << "Enter an integer: ";
    cin >> num;

    do
    {
        num /= 10;
        n++;
    } while(num != 0);
```

```
18
19      cout << "Digits: " << n << endl;
20      return 0;
21  }
```

**运行结果**

```
Enter an integer: 123
Digits: 3
```

**猜数字**

```
1   #include <iostream>
2   #include <cstdlib>
3   #include <ctime>
4
5   using namespace std;
6
7   int main()
8   {
9       srand(time(NULL));          // set random seed
10
11      // generate random number between 1 and 100
12      int answer = rand() % 100 + 1;
13      int num = 0;
14      int cnt = 0;
15
16      do
17      {
18          cout << "Guess a number: ";
19          cin >> num;
20          cnt++;
21
22          if(num > answer)
23          {
```

```
24              cout << "Too high" << endl;
25          }
26          else if(num < answer)
27          {
28              cout << "Too low" << endl;
29          }
30      } while(num != answer);
31
32      cout << "Correct! You guessed " << cnt << " times." << endl;
33      return 0;
34  }
```

**运行结果**

```
Guess a number: 50
Too high
Guess a number: 25
Too low
Guess a number: 37
Too low
Guess a number: 43
Too high
Guess a number: 40
Too high
Guess a number: 38
Too low
Guess a number: 39
Correct! You guessed 7 times.
```

## 3.2 for

### 3.2.1 for

while 循环将循环变量的初值、条件和更新写在了三个地方，但是这样不容易明显地看出循环变量的变化。

for 循环将循环变量的初值、条件和更新写在了一行内，中间用分号隔开。对于指定次数的循环一般更多地会采用 for 循环，而对于不确定次数的一般会采用 while 循环。

```
1 for(int i = 0; i < 5; i++)
2 {
3     cout << "i = " << i << endl;
4 }
```

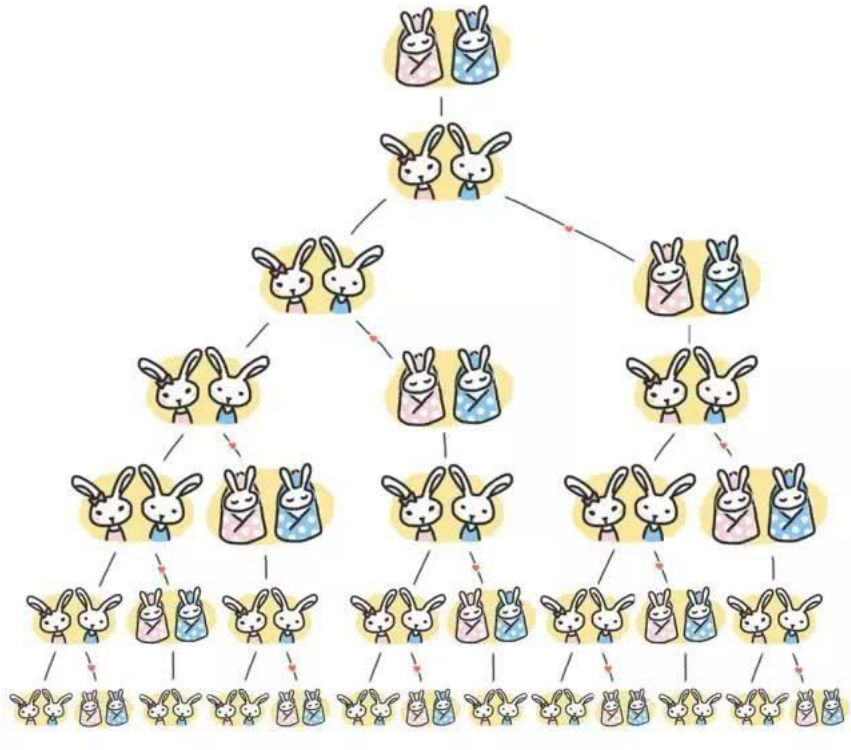
累加

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int sum = 0;
8      for(int i = 1; i <= 100; i++)
9      {
10         sum += i;
11     }
12     cout << "Sum = " << sum << endl;
13     return 0;
14 }
```

**运行结果**

```
Sum = 5050
```

## 斐波那契数列



```
#include <iostream>

using namespace std;

int main()
{
    int n;
    cout << "Enter the number of terms: ";
    cin >> n;

    if(n == 1)
    {
        cout << "1" << endl;
```

```
    }
    else if(n == 2)
    {
        cout << "1, 1" << endl;
    }
    else
    {
        int num1, num2, val;
        num1 = 1;
        num2 = 1;
        cout << "1, 1";

        for(int i = 3; i <= n; i++)
        {
            val = num1 + num2;
            cout << ", " << val;
            num1 = num2;
            num2 = val;
        }
        cout << endl;
    }

    return 0;
}
```

**运行结果**

```
Enter the number of terms: 10
1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

### 3.2.2 嵌套循环

循环也可以嵌套使用，外层循环每执行一次，内层循环就会执行多次。

```
for(int i = 0; i < 2; i++)
```

```
{
    for(int j = 0; j < 3; j++)
    {
        cout << "i = " << i << ", j = " << j << endl;
    }
}
```

**运行结果**

```
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 1, j = 0
i = 1, j = 1
i = 1, j = 2
```

**九九乘法表**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1*1=1 | 1*2=2 | 1*3=3 | 1*4=4 | 1*5=5 | 1*6=6 | 1*7=7 | 1*8=8 | 1*9=9 |
| 2*1=2 | 2*2=4 | 2*3=6 | 2*4=8 | 2*5=10 | 2*6=12 | 2*7=14 | 2*8=16 | 2*9=18 |
| 3*1=3 | 3*2=6 | 3*3=9 | 3*4=12 | 3*5=15 | 3*6=18 | 3*7=21 | 3*8=24 | 3*9=27 |
| 4*1=4 | 4*2=8 | 4*3=12 | 4*4=16 | 4*5=20 | 4*6=24 | 4*7=28 | 4*8=32 | 4*9=36 |
| 5*1=5 | 5*2=10 | 5*3=15 | 5*4=20 | 5*5=25 | 5*6=30 | 5*7=35 | 5*8=40 | 5*9=45 |
| 6*1=6 | 6*2=12 | 6*3=18 | 6*4=24 | 6*5=30 | 6*6=36 | 6*7=42 | 6*8=48 | 6*9=54 |
| 7*1=7 | 7*2=14 | 7*3=21 | 7*4=28 | 7*5=35 | 7*6=42 | 7*7=49 | 7*8=56 | 7*9=63 |
| 8*1=8 | 8*2=16 | 8*3=24 | 8*4=32 | 8*5=40 | 8*6=48 | 8*7=56 | 8*8=64 | 8*9=72 |
| 9*1=9 | 9*2=18 | 9*3=27 | 9*4=36 | 9*5=45 | 9*6=54 | 9*7=63 | 9*8=72 | 9*9=81 |

```
#include <iostream>

using namespace std;
```

```
4
5  int main()
6  {
7      for(int i = 1; i <= 9; i++)
8      {
9          for(int j = 1; j <= 9; j++)
10         {
11             cout << i << "*" << j << "=" << i*j << "\t";
12         }
13         cout << endl;
14     }
15     return 0;
16 }
```

**打印图案**

```
1  *
2  **
3  ***
4  ****
5  *****
```

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      for(int i = 1; i <= 5; i++)
8      {
9          for(int j = 1; j <= i; j++)
10         {
11             cout << "*";
12         }
13         cout << endl;
```

```
14      }
15      return 0;
16  }
```

## 3.3 break or continue?

### 3.3.1 break

break 可用于跳出当前的 switch 或循环结构。在一些情况下，在循环的中途已经完成了某个目标，没有必要再进行剩余的循环，这时就可以使用 break 跳出循环。

例如在判断一个数 $n$ 是否为素数时，利用循环逐个判断 $2 \sim n-1$ 之间的数是否能整除 $n$。只要发现其中有一个数能整除 $n$，就证明 $n$ 不是素数，可以跳出循环，不必再进行剩余的检查。

**素数**

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    int n;
    cout << "Enter an integer: ";
    cin >> n;

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

    if(is_prime)
```

```
23      {
24          cout << n << " is a prime number" << endl;
25      }
26      else
27      {
28          cout << n << " is not a prime number" << endl;
29      }
30
31      return 0;
32  }
```

**运行结果**

```
Enter an integer: 17
17 is a prime number
```

### 3.3.2 continue

continue 与 break 使用方法类似，但是它并不是跳出循环，而是跳过本轮循环，直接开始下一轮循环。

**正数平方和**

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       int n = 10;
8       cout << "Enter " << n << " integers: ";
9
10      int sum_square = 0;
11      for(int i = 0; i < n; i++)
```

```
12      {
13          int num;
14          cin >> num;
15          if(num < 0)
16          {
17              continue;
18          }
19
20          sum_square += num * num;
21      }
22
23      cout << "Sum of squares of positive integers: "
24              << sum_square << endl;
25
26      return 0;
27  }
```

**运行结果**

```
Enter 10 integers: 5 7 -2 0 4 -4 -9 3 9 5
Sum of squares of positive integers: 205
```

# Chapter 4 数组

## 4.1 数组

### 4.1.1 数组（Array）

数组能够存储一组类型相同的元素，数组在声明时必须指定它的大小（容量），数组的大小是固定的，无法在运行时动态改变。数组通过下标（index）来访问某一位置上的元素，下标从 0 开始。

```
1 int arr[5] = {3, 6, 8, 2, 4};
```

| arr[0] | arr[1] | arr[2] | arr[3] | arr[4] |
|---|---|---|---|---|

如果在声明数组时没有指定数组的大小，那么将根据初始化的元素个数来确定。

```
1 int arr[] = {3, 6, 8, 2, 4, 0, 1, 7};
```

通过下标可以访问数组中的元素，下标的有效范围是 0 ～ 数组的长度 - 1，如果使用不合法的下标就会导致数组越界。

```
1 printf("%d\n", arr[0]);     // 3
2 printf("%d\n", arr[3]);     // 2
3 printf("%d\n", arr[7]);     // 7
```

当数组的容量比较大时，可以使用循环来初始化数组。

```
1 int arr[10];
2
3 for(int i = 0; i < 10; i++) {
4     arr[i] = i + 1;
5 }
```

查找数据

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

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int key;
    printf("Enter the key: ");
    scanf("%d", &key);

    bool found = false;
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            found = true;
            break;
        }
    }

    if (found) {
        printf("%d exists.\n", key);
    } else {
        printf("%d not found!\n", key);
    }

    return 0;
}
```

**运行结果**

```
Enter the number of elements: 5
Enter the elements: 4 8 9 2 3
Enter the key: 2
2 exists.
```

**最大值/最小值**

```c
#include <stdio.h>

int main() {
    int num[] = {7, 6, 2, 9, 3, 1, 4, 0, 5, 8};
    int n = sizeof(num) / sizeof(num[0]);
    int max = num[0];
    int min = num[0];

    for(int i = 1; i < n; i++) {
        if(num[i] > max) {
            max = num[i];
        }
        if(num[i] < min) {
            min = num[i];
        }
    }

    printf("Max = %d\n", max);
    printf("Min = %d\n", min);
    return 0;
}
```

**运行结果**

```
Max = 9
Min = 0
```

### 4.1.2 二维数组（2-Dimensional Array）

二维数组由行和列两个维度组成，行和列的下标同样也都是从 0 开始。在声明二维数组时，需要指定行和列的大小。二维数组可以看成是由多个一维数组组成的，因此二维数组中的每个元素都是一个一维数组。

```
1 int arr[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
```

| | | | |
|---|---|---|---|
| arr[0][0] | arr[0][1] | arr[0][2] | arr[0][3] |
| arr[1][0] | arr[1][1] | arr[1][2] | arr[1][3] |
| arr[2][0] | arr[2][1] | arr[2][2] | arr[2][3] |

在初始化二维数组时，为了能够更直观地看出二维数组的结构，可以将每一行单独写在一行中。

```
1 int arr[3][4] = {
2     {1, 2, 3, 4},
3     {5, 6, 7, 8},
4     {9, 10, 11, 12},
5 };
```

对于容量较大的二维数组，可以通过两层循环进行初始化。

```
1 int arr[3][4];
2
3 for(int i = 0; i < 3; i++) {
4     for(int j = 0; j < 4; j++) {
5         arr[i][j] = 0;
6     }
7 }
```

## 矩阵运算

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1-0 & 3-0 \\ 1-7 & 0-5 \\ 1-2 & 2-1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ -6 & -5 \\ -1 & 1 \end{bmatrix}$$

```c
#include <stdio.h>

int main() {
    int A[3][2] = {
        {1, 3},
        {1, 0},
        {1, 2}
    };
    int B[3][2] = {
        {0, 0},
        {7, 5},
        {2, 1}
    };
    int C[3][2];

    printf("Matrix Addition\n");
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 2; j++) {
            C[i][j] = A[i][j] + B[i][j];
            printf("%3d", C[i][j]);
        }
        printf("\n");
    }

```

```
25      printf("Matrix Subtraction\n");
26      for(int i = 0; i < 3; i++) {
27          for(int j = 0; j < 2; j++) {
28              C[i][j] = A[i][j] - B[i][j];
29              printf("%3d", C[i][j]);
30          }
31          printf("\n");
32      }
33
34      return 0;
35  }
```

**运行结果**

```
Matrix Addition
  1  3
  8  5
  3  3
Matrix Subtraction
  1  3
 -6 -5
 -1  1
```

## 4.2 字符串

### 4.2.1 ASCII

美国信息交换标准代码 ASCII（American Standard Code for Information Interchange）一共定义了 128 个字符。

| ASCII | 字符 | ASCII | 字符 | ASCII | 字符 | ASCII | 字符 |
|---|---|---|---|---|---|---|---|
| 0 | NUT | 32 | (space) | 64 | @ | 96 | ` |
| 1 | SOH | 33 | ! | 65 | A | 97 | a |
| 2 | STX | 34 | " | 66 | B | 98 | b |
| 3 | ETX | 35 | # | 67 | C | 99 | c |
| 4 | EOT | 36 | $ | 68 | D | 100 | d |
| 5 | ENQ | 37 | % | 69 | E | 101 | e |
| 6 | ACK | 38 | & | 70 | F | 102 | f |
| 7 | BEL | 39 | ’ | 71 | G | 103 | g |
| 8 | BS | 40 | ( | 72 | H | 104 | h |
| 9 | HT | 41 | ) | 73 | I | 105 | i |
| 10 | LF | 42 | * | 74 | J | 106 | j |
| 11 | VT | 43 | + | 75 | K | 107 | k |
| 12 | FF | 44 | , | 76 | L | 108 | l |
| 13 | CR | 45 | - | 77 | M | 109 | m |
| 14 | SO | 46 | . | 78 | N | 110 | n |
| 15 | SI | 47 | / | 79 | O | 111 | o |
| 16 | DLE | 48 | 0 | 80 | P | 112 | p |
| 17 | DC1 | 49 | 1 | 81 | Q | 113 | q |
| 18 | DC2 | 50 | 2 | 82 | R | 114 | r |
| 19 | DC3 | 51 | 3 | 83 | S | 115 | s |
| 20 | DC4 | 52 | 4 | 84 | T | 116 | t |
| 21 | NAK | 53 | 5 | 85 | U | 117 | u |
| 22 | SYN | 54 | 6 | 86 | V | 118 | v |
| 23 | TB | 55 | 7 | 87 | W | 119 | w |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 24 | CAN | 56 | 8 | 88 | X | 120 | x |
| 25 | EM | 57 | 9 | 89 | Y | 121 | y |
| 26 | SUB | 58 | : | 90 | Z | 122 | z |
| 27 | ESC | 59 | ; | 91 | [ | 123 | { |
| 28 | FS | 60 | < | 92 | \ | 124 | \| |
| 29 | GS | 61 | = | 93 | ] | 125 | } |
| 30 | RS | 62 | > | 94 | ^ | 126 | ~ |
| 31 | US | 63 | ? | 95 | _ | 127 | DEL |

ASCII

```c
#include <stdio.h>

int main() {
    for(int i = 0; i < 128; i++) {
        printf("%d - %c\n", i, i);
    }
    return 0;
}
```

### 4.2.2 字符串（String）

字符数组通常被称为字符串，字符串有两种初始化的方式。一种与普通数组的初始化类似，逐个写出每一个字符，最后需要手动添加 \0 字符，表示字符串的结束符；另一种是直接使用双引号，这种写法无需手动添加 \0。

```c
char str[8] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};
char str[8] = "program";
```

\0 占一个字符的大小，因此在设置字符串的大小时需要考虑 \0。

占位符%s 可以对字符串进行输入输出操作，使用 scanf() 和 gets() 都可以用于读取字符串，但是 scanf() 只会读取到空格为止，而 gets() 会读取到回车为止。

**字符串输入输出**

```c
#include <stdio.h>

int main() {
    char str1[32];
    printf("Enter string 1: ");
    gets(str1);
    puts(str1);

    char str2[32];
    printf("Enter string 2: ");
    scanf("%s", str2);
    printf("%s\n", str2);

    return 0;
}
```

**运行结果**

```
Enter string 1: hello world
hello world
Enter string 2: hello world
hello
```

**字符统计**

```c
#include <stdio.h>

int main() {
    char str[32];
```

```c
    printf("Enter a string: ");
    gets(str);
    printf("Character to search: ");
    char c = getchar();

    int cnt = 0;
    int i = 0;
    while (str[i] != '\0') {
        if (str[i] == c) {
            cnt++;
        }
        i++;
    }

    printf("\'%c\' appears %d times in \"%s\".\n", c, cnt, str);
    return 0;
}
```

**运行结果**

```
Enter a string: this is a test
Character to search: t
't' appears 3 times in "this is a test".
```

### 4.2.3 字符串函数

头文件 <string.h> 中定义了一些常用的字符串处理函数。

#### strlen()

计算字符串的长度。

**strlen()**

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

int main() {
    char s[] = "hello world";
    printf("Length: %d\n", strlen(s));
    return 0;
}
```

**运行结果**

```
Length: 11
```

### strcpy()

字符串复制，调用者需要确保字符串的大小足够。

**strcpy()**

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

int main() {
    char s1[32] = "hello world";
    char s2[32] = "program";

    strcpy(s1, s2);
    printf("s1 = %s\n", s1);
    printf("s2 = %s\n", s2);
    return 0;
}
```

**运行结果**

```
s1 = program
s2 = program
```

### strcat()

字符串拼接，调用者需要确保字符串的大小足够。

**strcat()**

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

int main() {
    char s1[32] = "hello";
    char s2[32] = "world";

    strcat(s1, s2);
    printf("s1 = %s\n", s1);
    printf("s2 = %s\n", s2);
    return 0;
}
```

**运行结果**

```
s1 = helloworld
s2 = world
```

### strcmp()

字符串比较，依次比较字符串中每个字符的 ASCII 码值。通过判断 strcmp() 的返回值，可以得知两个字符串比较后的结果。

- 负数：字符串 1 < 字符串 2
- 正数：字符串 1 > 字符串 2
- 0：字符串 1 == 字符串 2

**strcmp()**

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

int main() {
    char s1[32] = "communication";
    char s2[32] = "compare";
    printf("%d\n", strcmp(s1, s2));
    return 0;
}
```

**运行结果**

```
-1
```

### 4.2.4 字符串数组

字符串数组是一个二维的字符数组，或者可以理解为是由多个字符串组成的数组。

```c
char str[4][12] = {"C++", "Java", "Python", "JavaScript"};
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | C | + | + | \0 | | | | | | | | |
| **1** | J | a | v | a | \0 | | | | | | | |
| **2** | P | y | t | h | o | n | \0 | | | | | |
| **3** | J | a | v | a | S | c | r | i | p | t | \0 | |

```
printf("str[0] = %s\n", str[0]);        // C++
printf("str[1] = %s\n", str[1]);        // Java
printf("str[0][0] = %c\n", str[0][0]);  // C
printf("str[0][1] = %c\n", str[0][1]);  // +
```