

C++ 面向对象程序设计

人工智能 学院

田原

课程纲要

①

第二章 C++简单程序设计

C++语言概述

基本数据类型和表达式

数据的输入与输出

算法的基本控制结构

深度探索

本章主要内容

□ C++语言概述



词法信息

□ 基本数据类型和表达式

□ 数据的输入与输出



句法信息

□ 算法的基本控制结构



文法信息

□ 深度探索



扩充信息

课程纲要

①

第二章

C++语言概述

基本数据类型和表达式

数据的输入与输出

算法的基本控制结构

类型别名与类型判断

深度探索

C++语言的产生

□ C++是从C语言发展演变而来的，最初的被称为“带类的C”，1983年正式取名为C++，第一个国际标准C++ 98，目前是C++20

□ 全面兼容C

它保持了C的简洁、高效和接近汇编语言等特点

对C的类型系统进行了改革和扩充

C++也支持面向过程的程序设计，不是一个纯正的面向对象的语言

□ 支持面向对象的方法

C++语言程序实例

```
//2_1.cpp
#include <iostream>
using namespace std;
void main(void)
{
    cout<<"Hello!\n";
    cout<<"Welcome to c++!\n";
}
```

运行结果:

Hello!

Welcome to c++!

字符集

□大小写的英文字母：A~Z, a~z

□数字字符：0~9

□特殊字符：

| | | | | | | | |
|--------|-----|---|---|---|---|---|----|
| • 空格 | ! | # | % | ^ | & | * | |
| _(下划线) | | + | = | - | ~ | < | > |
| / | \ | ' | " | ; | . | , | () |
| [] | { } | | | | | | |

词法记号

① 关键字 C++预定义的单词

如: bool、char、double、else、float、.....

② 标识符 程序员声明的单词，它命名程序正文中的一些实体

③ 文字 在程序中直接使用符号表示的数据。

④ 操作符 用于实现各种运算的符号。

C++定义了操作符替代名: and、or、not、nor、....

⑤ 分隔符 () {} , : ; 用于分隔各个词法记号或程序正文

⑥ 空白符 空格、制表符（TAB键产生的字符）、换行符
（Enter键所产生的字符）和注释的总称

课程纲要

①

第二章

C++语言概述

基本数据类型和表达式

数据的输入与输出

算法的基本控制结构

深度探索

基本数据类型



常量与变量

```
#include <iostream>
using namespace std;
void main(void)
{
```

变量先声明后使用

符号常量

```
    const int PRICE = 30;
    int num, total;
```

整形变量

整形常量

```
    num = 10;
    total = num * PRICE;
    cout << total << endl;
```

浮点型变量

实数型常量

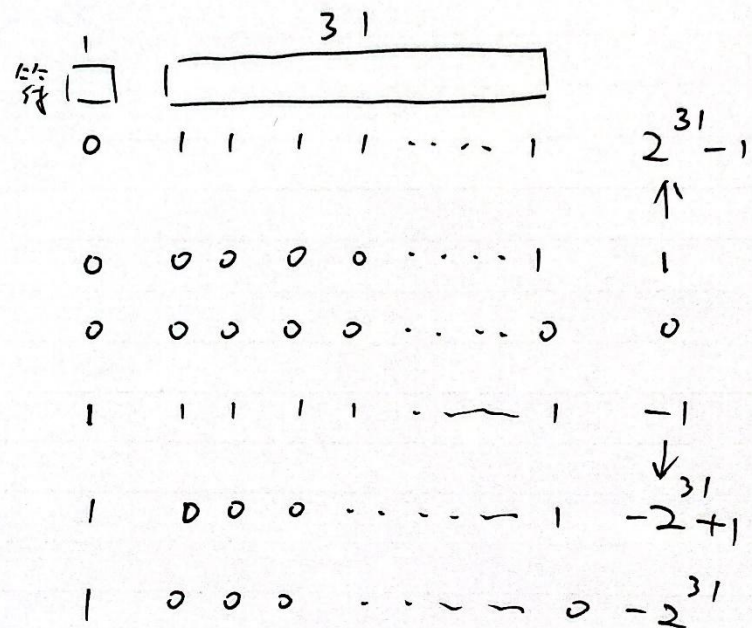
```
    float v, r, h;
    r = 2.5;
    h = 3.2;
    v = 3.14159 * r * r * h;
    cout << v << endl;
```

```
}
```

整形数据及其取值范围

| 类型 | 说明符 | 字节 | 数值范围 |
|----------------|-------|----|---------------------------|
| 短整 | short | 2 | $-32768 \sim 32767$ |
| 基本 | int | 4 | $-2^{31} \sim (2^{31}-1)$ |
| 长整 | long | 4 | $-2^{31} \sim (2^{31}-1)$ |
| 无符号 | | | |
| unsigned short | | 2 | $0 \sim 65535$ |
| unsigned [int] | | 4 | $0 \sim (2^{32}-1)$ |
| unsigned long | | 4 | $0 \sim (2^{32}-1)$ |

int 32 位 补码 .

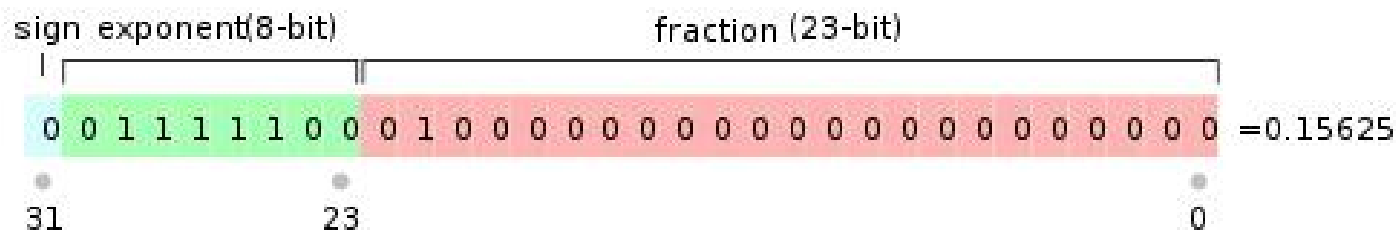


浮点数据及其取值范围

| 类型 | 说明符 | 字节数 | 数值范围 |
|----------|-------------|-----|---|
| 单精度浮点数 | float | 4 | $-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$ |
| 双精度浮点数 | double | 8 | $-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$ |
| 长型双精度浮点数 | long double | 8 | $-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$ |

浮点数据及其取值范围

■ float



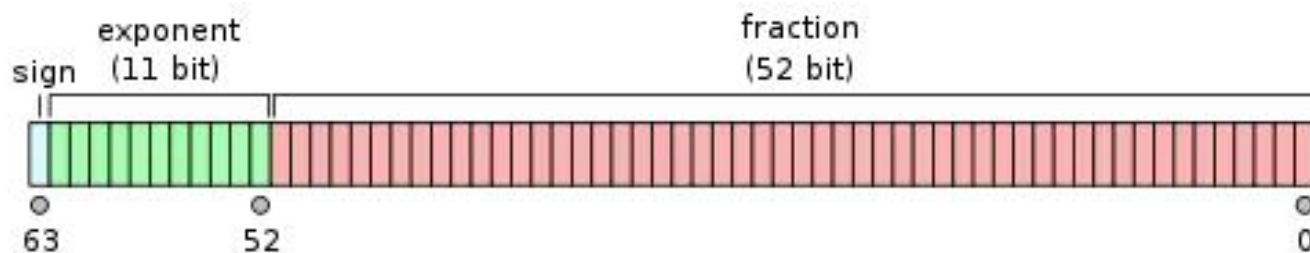
$$y = \pm 1.F * 2^{E-127}$$

$$1 \leq 1.F < 2$$

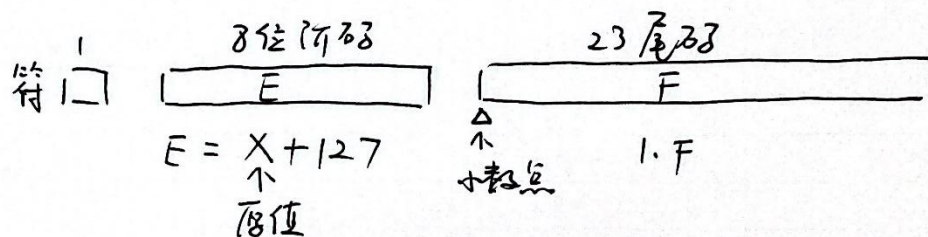
$$0 \leq E < 2^8 - 2$$

$$-2^{128} \leq y \leq +2^{128}$$

■ double



float 32 位



$$0 \leq E \leq 128 + 127$$

$$-127 \leq X \leq 128$$

-127 和 128 有特殊用处

~~127 和 128~~

$$-126 \leq X \leq 127$$

$$1 \leq 1.F < 2$$

$$-2 \cdot 2^{127} \leq \text{float} < 2 \cdot 2^{127}$$

$$-2^{128} < \text{float} < 2^{128}$$

$$-3.4 \times 10^{38} < \text{float} < 3.4 \times 10^{38}$$

$$(-2^{+128}, -2^{-126}] \cup [2^{-126}, 2^{128})$$

布尔型数据

□ 布尔型变量的说明：

例：**bool flag;**

□ 布尔型数据的取值：

只有 **false** 和 **true** 两个值

字符(串)型数据

□ 字符常量

单引号括起来的一个字符，如：'a', 'D', '?', '\$'

□ 字符变量

用来存放字符常量

例：char c1,c2;

c1='a';

c2='A';

□ 字符数据在内存中的存储形式

以ASCII码存储，占1字节，用7个二进制位

字符(串)型数据

□ 字符转义符

- 用途：与反斜杠搭配的一些特殊字符称为转义字符，转义字符可以表示特殊的意义，或者表示不容易表示的字符。
- 例如：想要定义一个字符变量，并赋值 “。
- `char c = ' ' ;` 错
`char c = \" ' ;` 对

| 转义字符 | 意义 | ASCII码值（十进制） |
|-------------------|----------------------|--------------|
| <code>\a</code> | 响铃(BEL) | 007 |
| <code>\b</code> | 退格(BS)，将当前位置移到前一位 | 008 |
| <code>\f</code> | 换页(FP)，将当前位置移到下页开头 | 012 |
| <code>\n</code> | 换行(LF)，将当前位置移到下一行开头 | 010 |
| <code>\r</code> | 回车(CR)，将当前位置移到本行开头 | 013 |
| <code>\t</code> | 水平制表(HT)（跳到下一个TAB位置） | 009 |
| <code>\v</code> | 垂直制表(VT) | 011 |
| <code>\\</code> | 代表一个反斜线字符“\” | 092 |
| <code>\'</code> | 代表一个单引号（撇号）字符 | 039 |
| <code>\"</code> | 代表一个双引号字符 | 034 |
| <code>\?</code> | 代表一个问号 | 063 |
| <code>\0</code> | 空字符(NULL) | 000 |
| <code>\ddd</code> | 1到3位八进制数所代表的任意字符 | 三位八进制 |
| <code>\xhh</code> | 1到2位十六进制数所代表的任意字符 | 二位十六进制 |

字符(串)型数据

- 字符数据的使用方法
 - 字符数据和整型数据之间可以运算。
 - 字符数据与整型数据可以互相赋值。

```
1 char c='a';           // 'a'的ASCII码是97, 'A'的ASCII是65
2 c = c-32;             // 相当于 c = 97-32
3 cout << c << "的ASCII码是" << int(c) << endl;
```

- 字符串常量

例:"CHINA"

"a"

'a'

所以: char c;

c="a";

| | | | | | |
|---|---|---|---|---|----|
| C | H | I | N | A | \0 |
|---|---|---|---|---|----|

| | |
|---|----|
| a | \0 |
|---|----|

| |
|---|
| a |
|---|



基本数据类型

基本的内置类型

C++ 为程序员提供了种类丰富的内置数据类型和用户自定义的数据类型。下表列出了七种基本的 C++ 数据类型：

| 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 |
| 无类型 | void |
| 宽字符型 | wchar_t |

一些基本类型可以使用一个或多个类型修饰符进行修饰：

- signed
- unsigned
- short
- long

基本数据类型

| 类型 | 位 | 范围 |
|--------------------|-----------|--|
| char | 1 个字节 | -128 到 127 或者 0 到 255 |
| unsigned char | 1 个字节 | 0 到 255 |
| signed char | 1 个字节 | -128 到 127 |
| int | 4 个字节 | -2147483648 到 2147483647 |
| unsigned int | 4 个字节 | 0 到 4294967295 |
| signed int | 4 个字节 | -2147483648 到 2147483647 |
| short int | 2 个字节 | -32768 到 32767 |
| unsigned short int | 2 个字节 | 0 到 65,535 |
| signed short int | 2 个字节 | -32768 到 32767 |
| long int | 8 个字节 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 |
| signed long int | 8 个字节 | -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 |
| unsigned long int | 8 个字节 | 0 到 18,446,744,073,709,551,615 |
| float | 4 个字节 | 精度型占4个字节 (32位) 内存空间, +/- 3.4e +/- 38 (~7 个数字) |
| double | 8 个字节 | 双精度型占8 个字节 (64位) 内存空间, +/- 1.7e +/- 308 (~15 个数字) |
| long double | 16 个字节 | 长双精度型 16 个字节 (128位) 内存空间, 可提供18-19位有效数字。 |
| wchar_t | 2 或 4 个字节 | 1 个宽字符 |

基本数据类型

注意，各种类型的存储大小与系统位数有关，但目前通用的以64位系统为主。

以下列出了32位系统与64位系统的存储大小的差别（windows 相同）：

| Windows vc12 | | Linux gcc-5.3.1 | | Compiler |
|--------------|-----|-----------------|--------|----------------|
| win32 | x64 | i686 | x86_64 | Target |
| 1 | | 1 | 1 | char |
| 1 | | 1 | 1 | unsigned char |
| 2 | | 2 | 2 | short |
| 2 | | 2 | 2 | unsigned short |
| 4 | | 4 | 4 | int |
| 4 | | 4 | 4 | unsigned int |
| 4 | | 4 | 8 | long |
| 4 | | 4 | 8 | unsigned long |
| 4 | | 4 | 4 | float |
| 8 | | 8 | 8 | double |
| 4 | | 4 | 8 | long int |
| 8 | | 8 | 8 | long long |
| 8 | | 12 | 16 | long double |

变量初始化

- 定义一个变量的同时，设置初始值

```
int a = 3;  
double f = 3.56;  
char c = 'a' ;
```

```
int a(3);  
int a = {3}  
int a{3}
```

列表初始化使用
条件比较严格：
初始化时不允许
信息丢失。

列表初始化

```
double pi = 3.1415926;  
int a(pi), a = pi;  
int a{pi}, c = {pi}
```



□ 隐含转换

- 原则：将低类型数据转换为高类型数据

• char short int unsigned int long unsigned long float double

低 —————→ 高

混合运算时数据类型的转换

- 当参与运算的操作数必须是bool型时，如果操作数是其它类型，编译系统会自动将非0数据转换为true，0转换为false。
- 位运算的操作数必须是整数，当二元位运算的操作数是不同类型的整数时，也会自动进行类型转换。
- 赋值运算要求左值与右值的类型相同，若类型不同，编译系统会自动将右值转换为左值的类型。

□优点：数据安全。

□缺点：不可控。

混合运算时数据类型的转换

□解决方法：

(1) 为了避免不同的数据类型在运算中出现类型问题，应尽量使用同种类型数据。

(2) 显示转换，采用强制类型转换：

例如：

float c;

int a, b;

c=float(a)/float(b); 或 c= (float)a/(float)b;

类型说明符（表达式）；
（类型说明符）表达式；

类型别名

□ 给类型起一个特殊意义的名字，使用关键字typedef 或者 using:

例如:

```
typedef double Area, Volume;
```

```
typedef int Natural;
```

```
Natural i1,i2;
```

```
Area a;
```

```
Volume v;
```

```
using Area = double;
```

```
using Volume = double;
```

类型判断

□ 根据要赋值的表达式或者已定义变量的类型为新变量赋值，使用关键字 `auto` 和函数 `decltype`（）。

例如：

```
auto i= 1, j =2;
```

```
auto i= 1+2;
```

```
auto i = 1, j = 3.1415926;
```



```
int i= 1;
```

```
decltype(i) j = 2;
```

基本运算符及表达式

1) 算术运算: + - * / %

2) 赋值运算: = +=, -=, *=, /=, %=,

<<=, >>=, &=, ^=, |=

3) 关系运算: < <= > >= == !=

4) 逻辑运算: !(非) &&(与) ||(或)

5) 条件运算: $x = a > b ? a : b$

6) 逗号运算: $a = 3 * 5, a * 4$



$a = 3 * 5 ;$
 $a * 4 ;$

7) sizeof运算: sizeof(int) sizeof(a)

位运算符

(1) 按位与 (&)

□运算规则：将两个运算量的每一个位进行逻辑与操作

□举例：计算 $3 \& 5$

3: 0 0 0 0 0 0 1 1

5: (&) 0 0 0 0 0 1 0 1

3 & 5: 0 0 0 0 0 0 0 1

□用途：(与1或为0，与1或保持原值)

- 将某一位置0，其它位不变。例如：
将 char 型变量 a 的最低位置 0: $a = a \& 0xfe$;
- 取指定位。例如：有 char c; int a;
取出 a 的低字节，置于 c 中: $c = a \& 0xff$;

位运算符

(2) 按位或 (|)

□将某些位置1，其它位不变。（与1或为1，与0或保持原值）

(3) 按位异或 (^)

□使特定位置翻转（与0异或保持原值，与1异或取反）

(4) 按位取反 (~)

(5) 位移

□左移运算 (<<)

左移后，低位补0，高位舍弃。

□右移运算 (>>)

右移后，低位：舍弃

高位：无符号数：补0

有符号数：补“符号位”

2.2 基本数据类型和表达式

第二章 简单程序

运算符优先级

C++运算符优先级表

| 优先级 | 运算符 | 说明 | 结合性 |
|-----|------------------|--------------------------|------|
| 1 | :: | 范围解析 | 自左向右 |
| 2 | ++ -- | 后缀自增/后缀自减 | |
| | () | 括号 | |
| | [] | 数组下标 | |
| | . | 成员选择 (对象) | |
| 3 | -> | 成员选择 (指针) | 自右向左 |
| | ++ -- | 前缀自增/前缀自减 | |
| | + - | 加/减 | |
| | ! ~ | 逻辑非/按位取反 | |
| | (type) | 强制类型转换 | |
| | * | 取指针指向的值 | |
| | & | 某某的地址 | |
| | sizeof | 某某的大小 | |
| | new, new[] | 动态内存分配/动态数组内存分配 | |
| | delete, delete[] | 动态内存释放/动态数组内存释放 | |
| 4 | .* ->* | 成员对象选择/成员指针选择 | 自左向右 |
| 5 | * / % | 乘法/除法/取余 | |
| 6 | + - | 加号/减号 | |
| 7 | << >> | 位左移/位右移 | |
| 8 | < <= | 小于/小于等于 | |
| | > >= | 大于/大于等于 | |
| 9 | == != | 等于/不等于 | |
| 10 | & | 按位与 | |
| 11 | ^ | 按位异或 | |
| 12 | | 按位或 | |
| 13 | && | 与运算 | 自右向左 |
| 14 | | 或运算 | |
| 15 | ?: | 三目运算符 | |
| 16 | = | 赋值 | |
| | += -= | 相加后赋值/相减后赋值 | |
| | *= /= %= | 相乘后赋值/相除后赋值/取余后赋值 | |
| | <<= >>= | 位左移赋值/位右移赋值 | |
| | &= ^= = | 位与运算后赋值/位异或运算后赋值/位或运算后赋值 | |
| 17 | throw | 抛出异常 | 自左向右 |
| 18 | , | 逗号 | |

高

低

表达式语句

□格式：

表达式;

□表达式语句与表达式的区别：

- 表达式可以包含在其它表达式中，而语句不可。
- 例：if ((a=b)>0) t=a;
- 不可写为：if ((a=b;)>0) t=a;

语句

- 空语句
- 声明语句
- 表达式语句
- 选择语句
- 循环语句
- 跳转语句
- 复合语句
- 标号语句

课程纲要

①

第二章

C++语言概述

基本数据类型和表达式

数据的输入与输出

算法的基本控制结构

深度探索

I/O流

□ 数据输入输出是通过I/O流来实现的，使用预定于的流对象cin和cout，并配合预定义的插入符（<<）和提取符（>>）

- 向标准输出设备（显示器）输出

例：int x;

```
cout<< "x= " <<x<<endl;
```

- 从标准输入设备（键盘）输入

例：int x, y;

```
cin>>x>>y;
```

简单的I/O控制

| 操纵符名 | 含义 |
|-------------------|-------------------|
| dec | 数值数据采用十进制表示 |
| hex | 数值数据采用十六进制表示 |
| oct | 数值数据采用八进制表示 |
| ws | 提取空白符 |
| endl | 插入换行符，并刷新流 |
| ends | 插入空字符 |
| setprecision(int) | 设置浮点数的小数位数(包括小数点) |
| setw(int) | 设置域宽 |

简单的I/O控制

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

using namespace std;

int main()
{
    cout<<10<<hex<<10<<oct<<10<<dec<<10<<endl;

    cout << setw(5) << 123 << endl;
    cout << setw(5) << 1234 << endl;
    cout << setw(5) << 12345 <<endl;
    cout << setw(5) << 123456 <<endl;

    cout<<setprecision(3)<<3.1415<<endl;
    return 0;
}
```

课程纲要

①

第二章

C++语言概述

基本数据类型和表达式

数据的输入与输出

算法的基本控制结构

深度探索

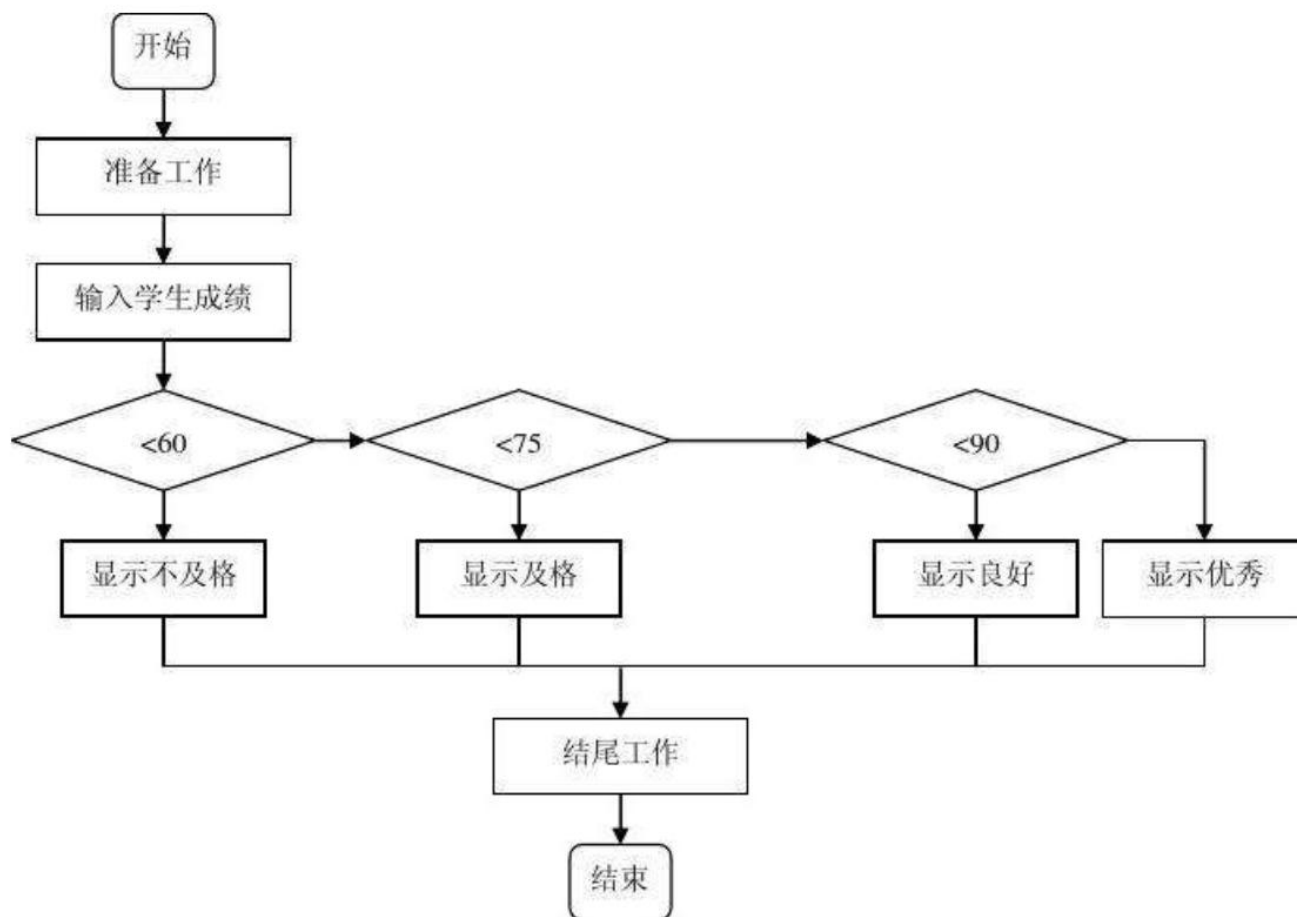
基本的控制结构

- 顺序结构
- 选择结构 (if、switch)
- 循环结构 (while、do-while、for)

流程图

| 符号 | 符号名称 | 功能说明 |
|---|----------|---|
|  | 起止框 | 表示算法的开始和结束 (注：一个算法只能有一个开始处，但可以有多多个结束处) |
|  | 处理框 | 表示执行一个步骤(框中指出执行的内容) |
|  | 判断框 | 表示要根据条件选择执行路线，离开的箭头会多于一个 |
|  | 输入输出框 | 表示需要用户输入或由计算机自动输出的信息 |
|  | 流程线(指向线) | 指示流程的方向 |

流程图



判断结构

(1) if 语句

□ if (表达式) 语句1
 else 语句2

例: `if (x>y) cout<<x;`
 `else cout<<y;`

□ if (表达式) 语句

例: `if (x>y) cout<<x;`

判断结构

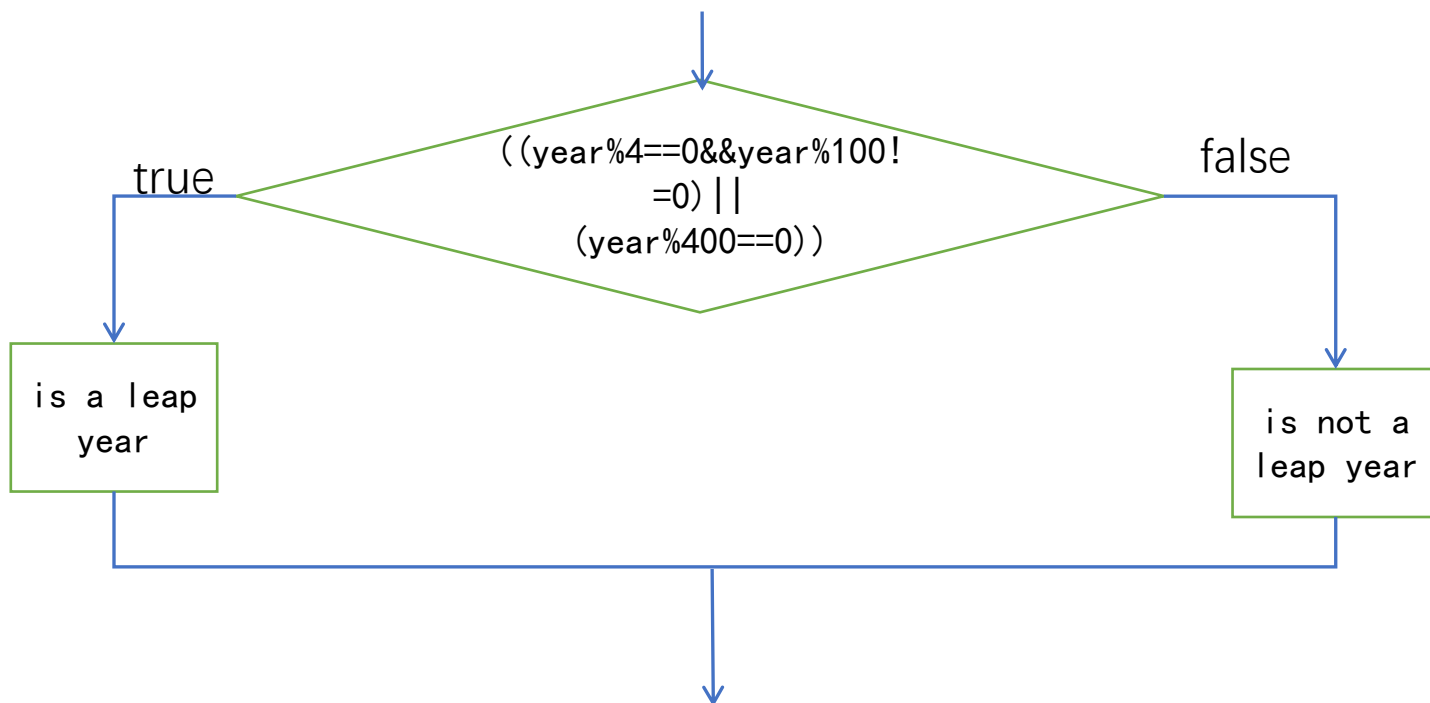
(1) if 语句

普通闰年：公历年份是4的倍数，且不是100的倍数的，为闰年（如2004年、2020年等就是闰年）。

世纪闰年：公历年份是整百数的，必须是400的倍数才是闰年（如1900年不是闰年，2000年是闰年）。

□例2-1

输入一个年份，判断是否闰年。



```
#include <iostream>
using namespace std;
void main(void)
```

```
{ int year;
  bool IsLeapYear;
```

```
  cout << "Enter the year: ";
  cin >> year;
```

```
  IsLeapYear = ((year % 4 == 0 &&year % 100 != 0)|| (year % 400 == 0));
  if (IsLeapYear)
    cout << year << " is a leap year" << endl;
  else
    cout << year << " is not a leap year" << endl;
}
```

运行结果:

Enter the year: 2000

2000 is a leap year

判断结构

(2) 嵌套的if 语句

□一般形式

if (表达式)

if (表达式) 语句 1

else 语句 2

else

if (表达式) 语句 3

else 语句 4

□注意

语句 1、2、3、4 可以是复合语句，每层的 if 与 else 配对，或用 { } 来确定层次关系。

判断结构

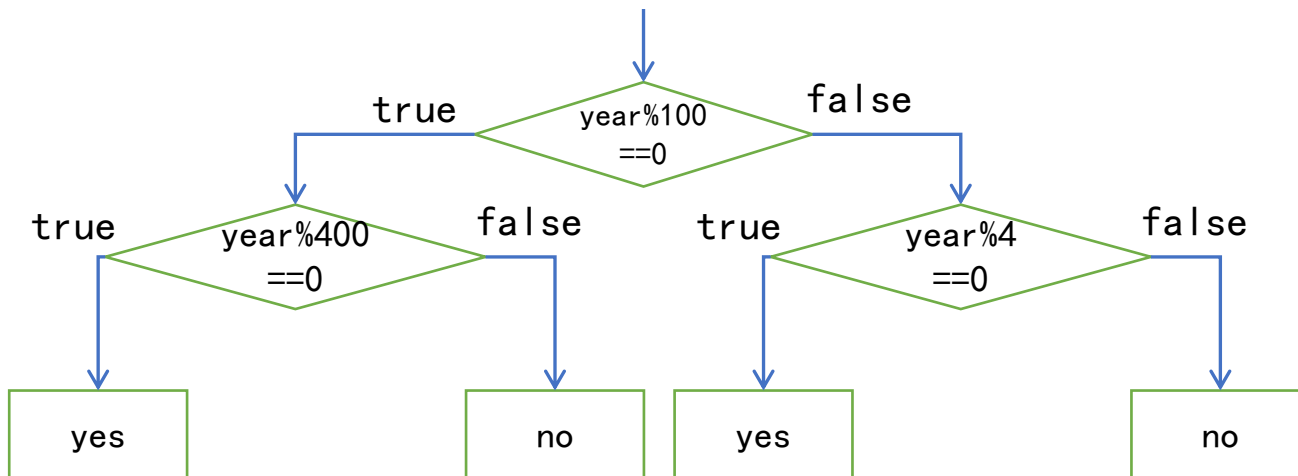
(1) if 语句

普通闰年：公历年份是4的倍数，且不是100的倍数的，为闰年（如2004年、2020年等就是闰年）。

世纪闰年：公历年份是整百数的，必须是400的倍数才是闰年（如1900年不是闰年，2000年是闰年）。

□例2-1

输入一个年份，判断是否闰年。




```
#include <iostream>
using namespace std;
void main(void)
{   int year;
    cout << "Enter the year: ";
    cin >> year;
    if(year % 100 == 0){
        if(year % 400 ==0){
            cout << year << " is a leap year" << endl;
        }else{
            cout << year << " is not a leap year" << endl;
        }
    }else{
        if(year % 4 ==0){
            cout << year << " is a leap year" << endl;
        }else{
            cout << year << " is not a leap year" << endl;
        }
    }
}
```

运行结果:

Enter the year: 2000

2000 is a leap year

判断结构

(3)if...else if语句

□一般形式

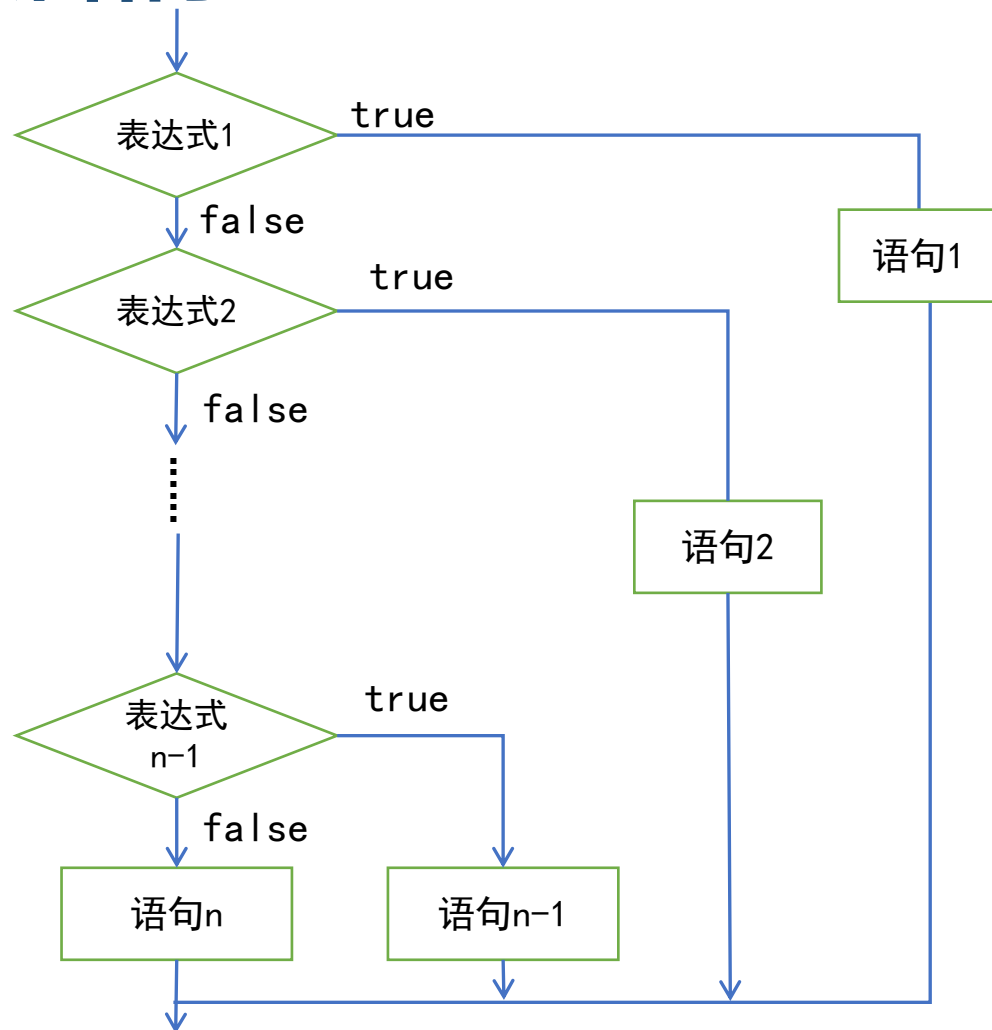
if (表达式1) 语句1

else if (表达式2) 语句2

else if (表达式3) 语句3

...

else 语句 n



判断结构

(4) switch 语句

□一般形式

switch (表达式)

```
{ case 常量表达式 1: 语句1
  case 常量表达式 2: 语句2
    |
  case 常量表达式 n: 语句n
  default:           语句n+1
}
```

每个常量表达式的值
不能相同, (有break)
次序不影响执行结果。

□执行顺序

以case中的常量表达式值为入口标号, 由此开始顺序执行。出口是break。

判断结构

(4) switch 语句

□注意事项:

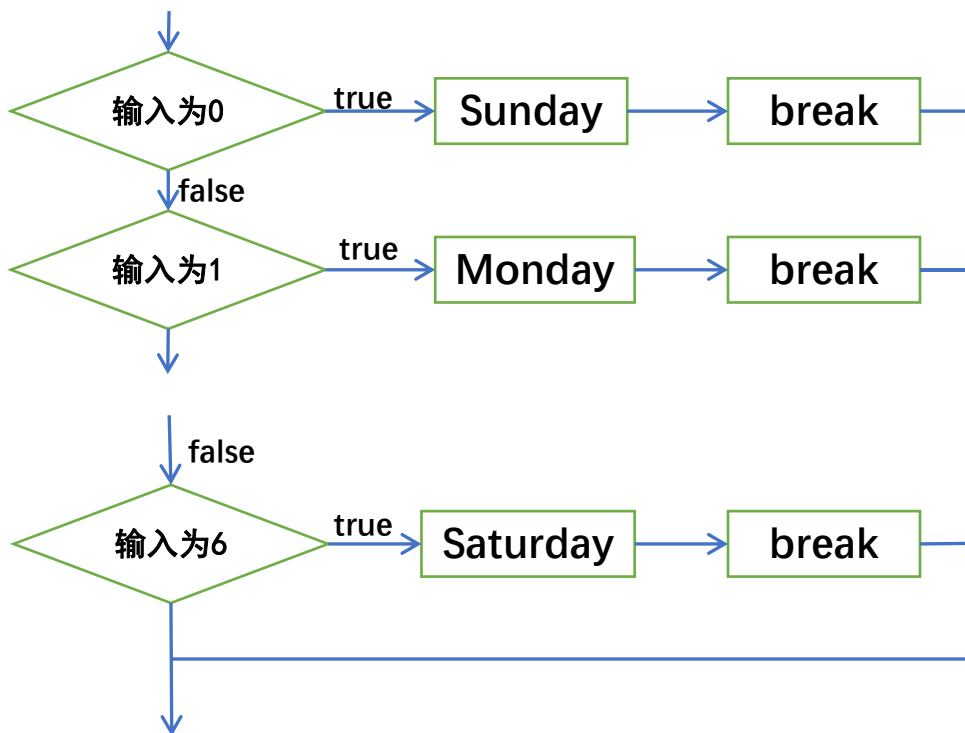
- case分支可包含多个语句，且不用{ }。
- switch后面表达式可以是int型、char型、enum型。
- 若干分支执行内容相同可共用一组语句。

判断结构

(4) switch 语句

□例2-4

输入一个0~6的整数，转换成星期输出。



```
#include <iostream>
using namespace std;
void main(void)
{ int day;
  cin >> day;
  switch (day)
  {
```

```
    case 0:cout << "Sunday" << endl;  break;
    case 1:cout << "Monday" << endl;  break;
    case 2:cout << "Tuesday" << endl;  break;
    case 3: cout << "Wednesday" << endl;  break;
    case 4:cout << "Thursday" << endl;  break;
    case 5:cout << "Friday" << endl;  break;
    case 6:cout << "Saturday" << endl;  break;
    default:
```

```
        cout << "Day out of range Sunday .. Saturday" <<
endl;
```

```
                break;
```

```
    }
```

```
}
```

运行结果:

2

Tuesday

循环结构


(1) while 语句

□形式

- while (表达式) 语句

□执行顺序

先判断表达式的值，为 true 时，再执行语句。



可以是复合语句，其中必须含有改变条件表达式值的语句。

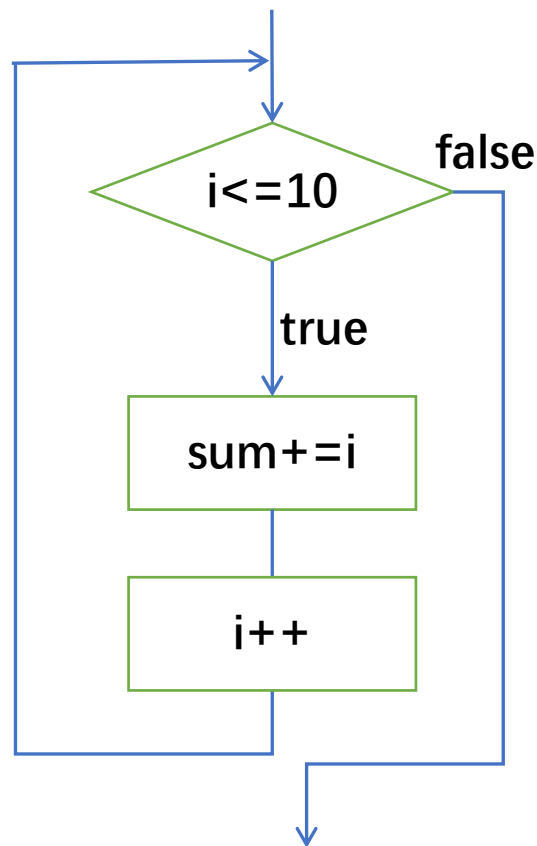
循环结构

(1) while 语句

□例2-5

求自然数1~10之和

分析：本题需要用累加算法，累加过程是一个循环过程，可以用while语句实现。




```
#include<iostream>
using namespace std;
void main()
{
    int i(1), sum(0);
    while(i<=10)
    {
        sum+=i; //相当于sum=sum+i;
        i++;
    }
    cout<<"sum="<<sum
    <<endl;
}
```

运行结果:

55

循环结构

(2) do-while 语句

□一般形式

do 语句

while (表达式)

可以是复合语句，其中必须含有改变条件表达式值的语句。

□执行顺序

先执行循环体语句，后判断条件。表达式为 true 时，继续执行循环体。

□与while 语句的比较：

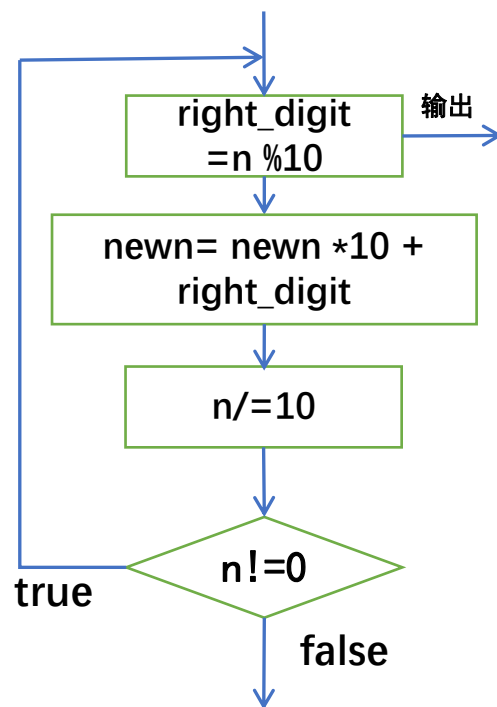
while 语句执行顺序：先判断表达式的值，为true 时，再执行语句

循环结构

(2) do-while 语句

□例2-6

输入一个整数，将各位数字反转后输出。



```
#include <iostream>
using namespace std;
void main(void)
{
    int n, right_digit, newnum = 0;
    cout << "Enter the number: ";
    cin >> n;

    cout << "The number in reverse order is ";
    do
    {
        right_digit = n % 10;
        newnum = newnum * 10 + right_digit;
        n /= 10; //相当于n=n/10
    }
    while (n != 0);
    cout<< newnum <<endl;
}
```

运行结果:

Enter the number: 365

The number in reverse order is 563

循环结构

程序1:

```
#include<iostream>
using namespace std;
void main()
{
    int i, sum(0);
    cin>>i;
    while(i<=10)
    {
        sum+=i;
        i++;
    }
    cout<<"sum="<<sum<<endl;
}
```

**while
v.s.
do-while**

程序2:

```
#include<iostream>
using namespace std;
void main()
{
    int i, sum(0);
    cin>>i;
    do {
        sum+=i;
        i++;
    } while(i<=10);
    cout<<"sum="<<sum<<endl;
}
```

循环结构

(3) for 语句

□语法形式

for (初始语句; 表达式1; 表达式2) 语句

循环前先求解



每次执行完循环体后求解

执行的条件:
为true时执行循环体

循环结构

(3) for 语句

□一般使用方法

```
for (int i; i<=100; i++){  
    cout<<i<<endl;  
}
```

```
cout<<i<<endl;
```



□注意：初始语句、表达式1、表达式2都可以省略，但是分号不能省略

```
for(;;) //死循环
```

循环结构

(3) for 语句

□for语句可以用于模拟while语句

//程序1

```
for( ; i<100;i++ ){  
    sum +=i;  
}
```

//程序2

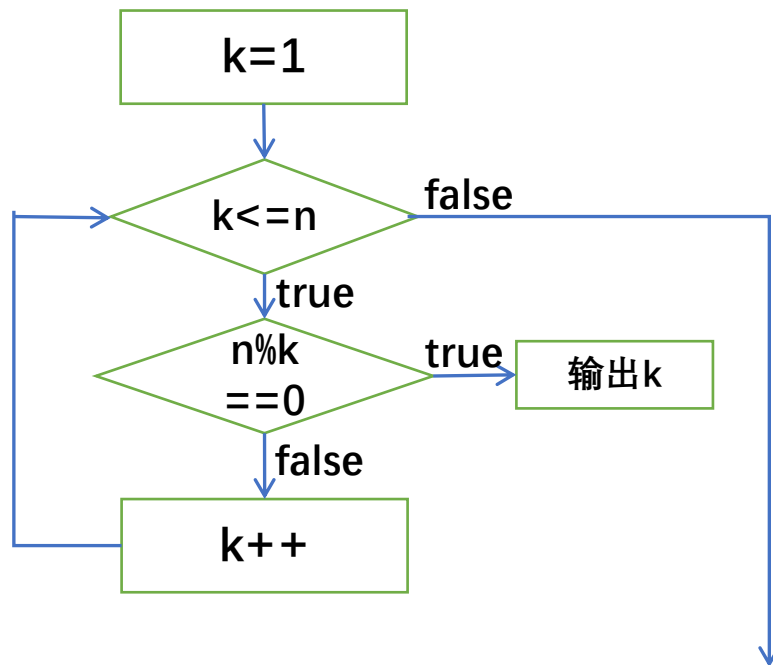
```
while(i<100){  
    sum +=i;  
    i++;  
}
```


循环结构

(3) for 语句

□例2-7

输入一个整数，求出
它的所有因子。



```
#include <iostream>
using namespace std;
void main(void)
{
    int n, k;
```

```
    cout << "Enter a positive integer: ";
    cin >> n;
    cout << "Number " << n << " Factors ";
```

```
    for (k=1; k <= n; k++)
        if (n % k == 0)
            cout << k << " ";
    cout << endl;
}
```

运行结果:

Enter a positive integer: 7

Number 7 Factors 1 7

循环结构

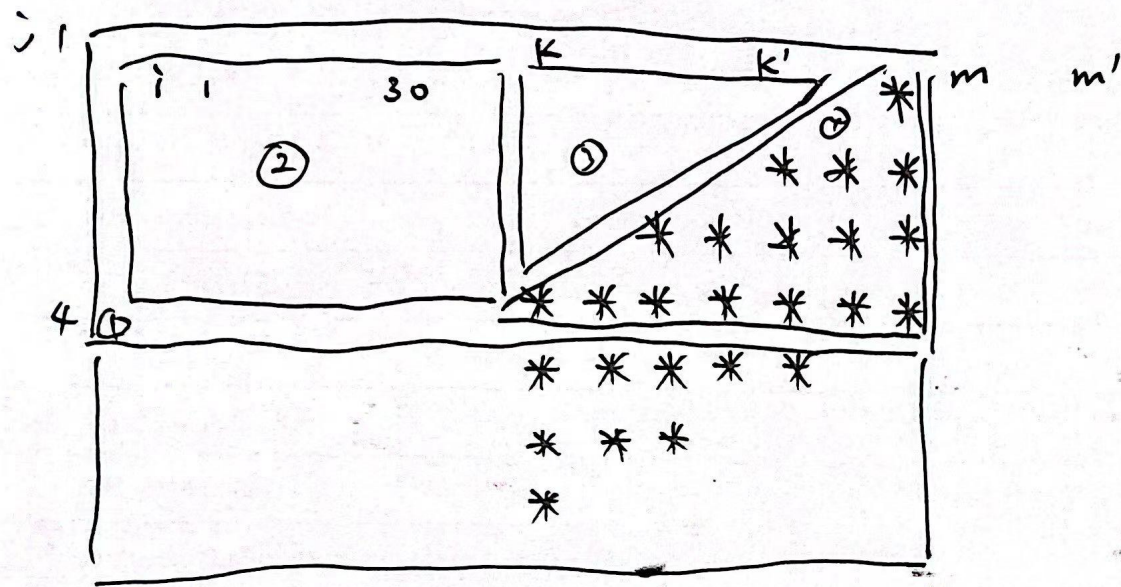
(3) for 语句

□例2-9

打印：

```

                                *
                                ***
                                *****
← 30个字符 →  *****************
                                *****
                                ***
                                *
```



| j | k' | $k' = wj + b$ |
|-----|------|---------------|
| 1 | 6 | $6 = w + b$ |
| 2 | 4 | $4 = 2w + b$ |
| 3 | 2 | $w = -2$ |
| 4 | 0 | $b = 8$ |

$$k' = -2 \cdot j + 8$$

| j | m' | $m' = wj + b$ |
|-----|------|---------------|
| 1 | 1 | $1 = wj + b$ |
| 2 | 3 | $3 = 2w + b$ |
| 3 | 5 | $w = 2$ |
| 4 | 7 | $b = -1$ |

$$m' = 2 \cdot j - 1$$

```
#include<iostream>
using namespace std;
void main()
{ int i, j, n=4;
  for(i=1;i<=n;i++) //输出前4行图案
  { for(j=1;j<=30;j++)
    cout<<' '; //在图案左侧空30列
    for(j=1;j<=8-2*i;j++)
      cout<<' ';
    for(j=1;j<=2*i-1;j++)
      cout<<'*';
    cout<<endl;
  }
```

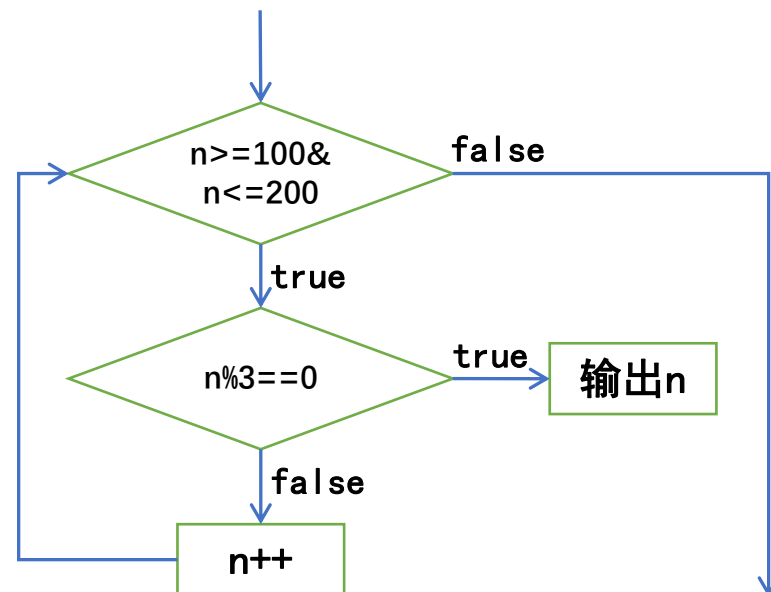
```
for(i=1;i<=n-1;i++) //输出后3行图案
{ for(j=1;j<=30;j++)
    cout<<' '; //在图案左侧空30列
  for(j=1;j<=7-2*i;j++)
    cout<<'*';
  cout<<endl;
}
}
```

循环结构

(3) for 语句

□例2-9

100~200之间能被3整除的数。



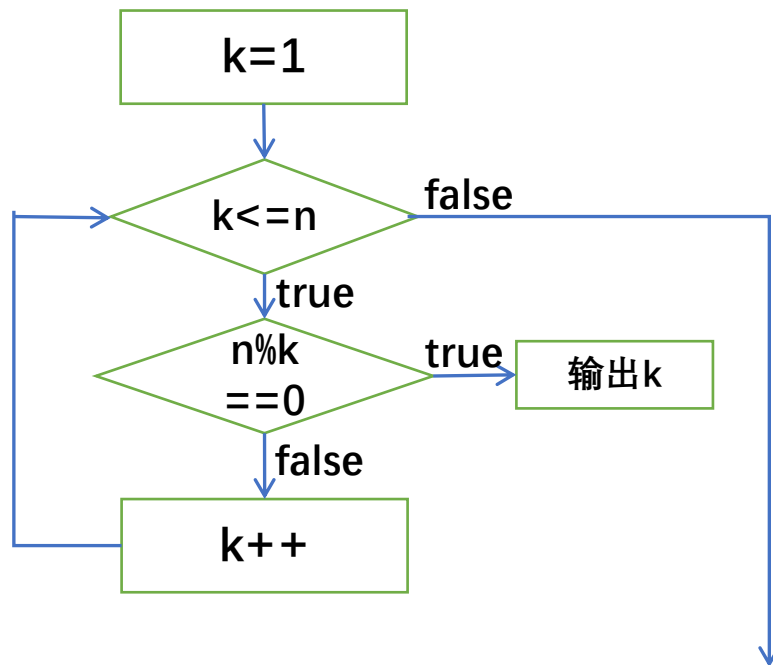
```
#include <iostream>
using namespace std;
void main()
{
    int n;
    for(n=100; n<=200; n++)
    {    if (n%3!=0)
        cout<<n;
    }
}
```


循环结构

(3) for 语句

□例2-7

输入一个整数，求出它的所有因子。



```
#include <iostream>
using namespace std;
void main(void)
{
    int n, k;
```

```
    cout << "Enter a positive integer: ";
    cin >> n;
    cout << "Number " << n << " Factors ";
```

```
    for (k=1; k <= n; k++)
        if (n % k == 0)
            cout << k << " ";
    cout << endl;
}
```

运行结果:

Enter a positive integer: 7

Number 7 Factors 1 7

其他控制语句

□break语句

- 使程序从循环体和switch语句内跳出，继续执行逻辑上的下一条语句。不宜用在别处。

□continue 语句

- 结束本次循环，接着判断是否执行下一次循环。

□goto语句

- 跳转到制定的语句标号。

其他控制语句

```
#include<iostream>
using namespace std;
void main()
{
    for(int i; i<=10;i++)
    {
        if(i==4) break;
        cout<<i<<endl;
    }
}
```

```
#include<iostream>
using namespace std;
void main()
{
    for(int i; i<=10;i++)
    {
        if(i==4) continue;
        cout<<i<<endl;
    }
}
```

其他控制语句

□goto语句

- 不具有结构性，不建议使用。
- 跳出多重循环时，可以使用。

```
for(...){  
    for(...){  
        for(...){  
            ...  
            for( ){  
                goto A;  
            }  
            ...  
        }  
    }  
}  
A: ....
```

课程纲要

①

第二章

C++语言概述

基本数据类型和表达式

数据的输入与输出

算法的基本控制结构

深度探索

深度探索

□ 变量的定义与声明

- 定义，就是编译器创建了一个变量，规定类型，并为这个变量分配一块内存。
- 声明，就是告诉编译器变量的类型，编译器并不为其分配内存。
- 声明 \subset 定义。

□ 为什么要规定变量的类型？

- 每种类型的全部特性都蕴含在了它所执行的操作当中，CUP所执行的指令并不对操作数的类型加以区分，对所有操作数都执行相同的操作，编译器需要根据变量的数据类型选择适当的指令。

深度探索

例如：

```
short a = -1; //a的二进制表示为0xff
```

```
unsigned short b = 65535; // b的二进制表示为0xffff
```

```
int c ,d;
```

```
c = a; //对a执行“符号扩展”，得到0xffff，赋给c
```

```
d = b; //对b执行“零号扩展”，得到0x00ff，赋给d
```


课后作业

- 完成奇数课后题
- 完成时间：下节课前



HOMEWORK